Beamformed Channel Matrix Positioning Using 5G Testbench CSI data With a Deep-Learning Pipeline

Andre Ráth

LUND
UNIVERSITY

Department of Electrical and Information Technology

MSc Thesis TFRT-insert number
ISSN 0280–5316

Department of Electrical and Information Technology
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck.
Lund 2022

Abstract

Within the telecommunications industry, a positioning system for estimating user equipment (UE) location using information already available at the basestation (BS) has an enormous number of potential uses. The link between physical position and the network channel state enables potential positioning systems to function by understanding the network channel state dependency on location, using a model-based, data-based, or a combined approach.

A key exploitable phenomenon linked to position is that of multi-path propagation, wherein transmissions can arrive from multiple directions to the BS, with a unique propagation pattern corresponding to a unique environment. In fifth generation wireless technology (5G), multi-path components are already exploited for beamforming with massive multiple-input multiple-output (MIMO) technology. Basestations therefore have a preexisting pipeline for obtaining beamformed channel matrices from channel state information (CSI) transmitted by the UE. A data-driven approach using multi-path propagation phenomena for positioning is possible through utilizing the already available beamformed channel matrix in the basestation.

In this thesis the practical data-driven deep-learning approach for UE positioning in 5G using beamformed channel matrices is examined. Real-world data is utilized to judge the applicability of the approach, with measurements done on a commercial-grade Ericsson 5G testbench in both non-line-of-sight (NLoS) and line-of-sight (LoS) scenarios. Using a similar approach as other papers in the field, a supervised deep-learning approach is used for instantaneous position estimation. For improving positioning accuracy through trajectory estimation, a novel approach of using particle filtering with network ensemble outputs for kernel density estimation of an observation probability density function is proposed. The results show that using the outlined methods position is possible to estimate in real-world pedestrian tests with a mean accuracy of 2-5 meters, even with NLoS conditions and poor underlying GNSS training data quality.

Acknowledgements

The writing of this thesis would not have been possible without the technical, practical, or emotional support and education of the large network of loved-ones, friends, former colleagues, professors, and advisors surrounding me, along with the coffee-machines at Ericsson that kept me awake during the long hours.
Foremost, I would like to extend thanks to my co-supervisors Prof. Bo Bernhardsson and Dino Pjanic, along with my primary supervisor Prof. Fredrik Tufvesson. Bo and Fredrik ensured the solid grounding of the thesis in theory, and made possible the existence of it in the first place. I would also like to extend Dino Pjanic a special thank-you for going above and beyond to spend many long hours with me making the measurements necessary to have a thesis based on real data, and to help me navigate the internal world of Ericsson in search for further advice and insight on the often opaque functions of the experimental equipment.

Contents

1

Introduction

1.1 Background

Positioning in 5G

Wireless communications have in the past decades become a ubiquitous part of everyday life. Most recently, fifth generation wireless technology - 5G - has seen large-scale adoption worldwide. Further developments in this field can be seen as crucial as applications requiring higher data-rates and/or better latency stability such as autonomous driving and augmented reality reach maturity. One avenue to achieve these goals is to enable more efficient use of existing resources through the utilization of AI for 'smart' devices. More specifically, machine-learning has become a widespread focus area for the telecommunications industry, enabling a multitude of potential use-cases.

The research areas for utilizing machine learning on 5G include - but are not limited to - user equipment (UE) localization [Burghal et al., 2020a], line-of-sight or non-line-of-sight identification, communication scenario identification, channel modelling and prediction with machine learning, beamforming, anomaly prediction, and more [Huang et al., 2021a] [Huang et al., 2021b] [Santos et al., 2020].

Of the listed use-cases, positioning is one with a great deal of industry interest. A viable positioning system in wireless telecommunications where the 5G gNb has an estimate for the position of every user in a Multi-user MIMO scenario would enable a large number of commercial improvements in 5G networks. These can include e.g. location-aware communications, predictive network resource allocation, better handover management, demand prediction, and more [Björnson et al., 2019].

Furthermore, speculating on the future of IoT, using UEs as proxies for position and device density detection and sending privacy-preserving general device-density data to cloud data-centers has a huge potential for unlocking commercial and public uses [Traboulsi, 2022] [Mogyorósi et al., 2022], e.g. 'smart' real-time automated public transport allocation and traffic redirection using real-time population density estimates.

In prior work, among several other references, research in 5G positioning include using ML for high-precision indoor positioning in well-controlled environments [Bast et al., 2020], positioning in simulated environments [Guo et al., 2020], multi-anchor and Dense 5G Network positioning [Koivisto et al., 2017] [Guerra et al., 2018], fused approaches [Yang et al., 2020], and smartphone-based sensor fusion [Davidson and Piché, 2016].

Many of the approaches for positioning are theoretical and method-driven, with restricted access to commercial-grade 5G access-points set up such that experiments can be run in real-world conditions. This gives a window of opportunity for this thesis to contribute to the field with a data- and application-driven approach. Ericsson Lund has a commercial-grade 5G basestation set up with proper experimental equipment to conduct research in a real-world setting, which will become the goal of this thesis.

In summary, a full basestation-side positioning pipeline driven by analysis of measured real-world data will be proposed in this thesis. The work will thereby fit into an under-explored niche in industry and academia.

Initial approach

No matter the objective of the ML algorithm on wireless data, the feature(s) used are of importance. There are multiple methods widely used in literature, some model-driven some data-driven, and some a combination of the two. The utilized network features commonly include e.g. various signal energy and delay features such as received signal strength, power delay profile [Gante et al., 2018], time of arrival and angle of arrival and others [Burghal et al., 2020b] [Wen et al., 2019]. The commonality between these is that the quantities and/or the models are themselves derived from physics assumptions underlying the network channel. Information on more 'raw' channel quantities utilized within the e.g. basestation hardware for beamforming are seldom used due to difficulty of access, as to obtain them the information stored directly in Channel State Information (CSI) feedback messages must be extracted.

CSI is defined as information describing the radio propagation channel between user equipment (UE) and the network base-station. For MIMO radio systems, such as in 5G, CSI data includes channel properties extracted from the Sounding Reference Signal (SRS), including complex-valued multidimensional data structures describing the impulse response matrix between the UE and the beam-domain basestation, where dimensions correspond to the frequency and spatial beam domains. The expectation is that the SRS data contains more information about the state of the network than many other more derived quantities, though at the cost of being high-dimensional and unreadable w.r.t. human intuition [Li et al., 2017].

The goal then with channel impulse response matrices obtained from SRS CSI data is that as the directional beam-information within them correlates extremely well with user position. As a user moves, the channel state changes according to

which beams provide a good transmission link between the UE and the basestation.

1.2 Prior Work

Considering the enormous range of research for positioning using various wireless signal phenomena, only literature that this thesis builds on directly in application and data domain are covered in this section - in other words, non-Line of Sight (nLoS) non-simulated CSI for a single basestation with beam-data in a realistic environment preferably not taken indoors.

By filtering for the above mentioned aspects, from a wide literature survey only three papers were selected as direct citations. For a brief but more broad overview of the field outside these three papers, see surveys [Wen et al., 2019], [Mogyorósi et al., 2022], [Davidson and Piché, 2016] [Burghal et al., 2020b] along with the discussion covered in the introduction Section 1.1.

Direct comparison on an 5G Ericsson testbench

The paper [Malmström et al., 2019] is the one with the most directly comparable experiment setup to the one available for this thesis. In it, they investigate the use of neural networks and random forests to estimate UE position for a non-line-of-sight urban outdoors scenario. For features, they had access to beam-data with 48 beams while and extracted the best received reference signal beam power. They also collected data outdoors on a 5G testbed provided by Ericsson, at a carrier frequency of 15 GHz and using a 8x8 antenna array with 56 directional beams.

Their best results in NLoS using data selected subsequently or consecutively as test data - therefore with some level of domain seperation - gave them a mean error of around 7 meters with the random forest and 12 meters with their neural network architectures. They also used a car driven at walking-speed with an antenna as their positioning vehicle, along with a high-accuracy GPS receiver sampling at 10 Hz.

CSI data from a 5G Huawei testbench

The paper [Decurninge et al., 2018], an analogue in terms of output data domain, used a Huawei 5G testbed on the University of the Chinese Academy of Science campus in Huairou. CSI data was measured for a single-antenna UE with a MIMO 32 dual-polarized antenna array at the BS, with 56 resulting antennas due to the loss of 8 from hardware issues. Instead of directly obtaining beamforming channel matrices from the testbench, they obtain raw CSI data and calculate channel covariance. They used a standard GPS receiver sampling at 1 Hz as their ground-truth data, and measured over a large outdoors area with mostly LoS with some minor NLoS parts.

They then applied a shallow neural network and k-nearest neighbor, giving them their best result of around 8 meters mean accuracy.

Practical indoors positioning

The paper [Widmaier et al., 2019] does not use 5G, and instead uses a 64-antenna universal transmitter with 1024 subcarriers to accomplish indoor positioning. Their paper is one of the few that examines the generalizability of their method to data taken over several days. Though their data was collected indoors, they used a realistic office environment with obstructions and people walking around.

Their achieved mean error of around 1 meter in the NLoS scenario by using a relatively deep neural network demonstrates the viability of this approach. The neural-network method used in this thesis will therefore be derived from the idea used in the third paper [Widmaier et al., 2019] of using deep-learning. Despite a similar machine-learning approach, due to their different wireless network setup and transmission environment, their results will not be directly comparable to that of this thesis.

1.3 Objectives and thesis structure

Objectives

The primary objective of this thesis is to create a positioning pipeline on beamformed channel matrix data obtained from a 5G Ericsson testbench, and hopefully build upon the papers mentioned in the prior section to hopefully obtain superior results. In more detail, the objectives of this thesis can be viewed as follows:

1. Set up a CSI beamformed channel matrix data measurement and data extraction process using a real commercial-grade 5G testbench supplied by Ericsson.

2. Execute the measurement process to collect usable data and then analyze the obtained data for viability with machine learning.

3. Formulate a pipeline to obtain position estimates from the beamformed channel matrix data in both LoS and NLoS scenarios.

Thesis Structure

The main part of the thesis, Chapters 2-9, can be approximately divided into three sections. The first part, Chapter 2-5, covers all the necessary theory background and understanding of the methodology. Chapter 6-7 is then the data acquisition and analysis part of the thesis, with Chapters 7-9 then introducing the specific used methods and their positioning results.

1.4 Platforms and tools

The platforms and tools needed for this thesis include:

- A proprietary commercial-grade 5G basestation set up at Ericsson, Lund, referred to as just the *basestation* with 64 directional beams.
- A proprietary 5G test-UE using a Qualcomm 8350 chip, referred to as just the *'UE'*
- 4K video stream for ensuring network activity on the 5G-connected UE
- A OnePlus Nord 5G AC2003 GNSS-capable android device
- A computer with an AMD Ryzen 4800H, 16GB of RAM and a CUDA-capable RTX 2060 with 6GB of VRAM for all processing done to obtain results
- C++ with boost::interprocess and pybind11 to create the log-extraction software
- A Python environment with algorithms primarily running using Numpy, PyTorch, Scikit-learn, Pandas, scipy, matplotlib, and pymap3d - used for all machine learning, visualizaiton, and higher-level data processing

Note that for this thesis almost all results will be in root-Mean Squared Error (rMSE) or Mearn Squared-error (MSE). If not written otherwise, assume the quantity for rMSE is in meters and meters-squared for MSE.

1.5 Limitations

The primary limitations in this thesis are the GPS data and the computation power available to process machine learning results. The obtained GPS ground truth accuracy is around 3 meters, which is insufficient and will prove to be the limiting factor in the LoS results later in this thesis. Note that more advanced GPS algorithms exist to reduce the error - but the time needed to implement such systems falls outside of the scope of this thesis.

Another limitation is the lack of man-hours to do measurements and the high failure rate thereof. With the current setup at Ericsson Lund, two people are required to obtain any measurements, and the basestation might reset or otherwise be interrupted during the measurement process, making results useless. In addition, only a limited form of the channel beam data could be achieved; instead of whenever it was updated, it was logged whenever it was used. This proved to be a detriment, leading to data 'clumping'.

One more issue was with the 137 potential frequency channels - only three could be successfully extracted for the channel matrix: the highest, lowest, and middle band. This greatly limited the analysis doable on the frequency-diverse aspect of the data, and also made the usage of e.g. convolutional neural networks unviable.

2

Basics of Wireless Communication and 5G

2.1 Brief history of telecommunications - 1G to 5G

As of 2022, there have been five industry-acknowledged generations of mobile communications adopted worldwide. The first generation, termed 1G, first arose around the 70s. In following decade, the governments of the Nordic countries jointly deployed the NMT (Nordic Mobile Telephony), joining other standards such as TACS (Total Access Communication System) and AMPS (Advanced Mobile Phone System). Overall, 1G technologies were based on analog transmission, and were limited to audio - voice - services [Jia et al., 2018], [Dahlman et al., 2018].

In the 90s, the necessity of developing digital data transmission led to the introduction of what was termed 2G, the second generation of wireless telecommunications. This era was characterized by the spread of SMS and digital voice. Of the systems developed, Global System for Mobile communication (GSM) spread the widest, becoming the first semi-global standard [Jia et al., 2018].

Originally created in Europe, the success of multinational cooperation for GSM led to a push for global standardization. In 1998 a global partnership termed the *Third-Generation Partnership Project* (3GPP) for the development of the third generation of mobile telecommunications (3G) was created. Due to its success, 3GPP would outlast 3G itself and now continues to determine telecommunications standards [Jia et al., 2018] [Dahlman et al., 2018].

3G was developed in part to enable e.g. video calling, email and basic mobile internet, dominating in the 2000s. By the end of the generation, datarates could theoretically peak in the dozens of Mbits/s. These speeds were enabled by the 3G evolution known as HSPA (High-Speed Packet Access) and HSPA+ [Jia et al., 2018].

The 2010s saw the introduction and domination of the fourth generation (4G) of wireless communication, globally standardized by 3GPP as LTE (Long Term Evolution) [Jia et al., 2018]. An evolution of HSPA, LTE is built on OFDMA technology - see Section 2.3 for more details. Another technology with mass-adoption

in 4G was MIMO, turning multi-path component echoes from a source of noise to a way to increase datarates and reliability - see Sections (2.3, 2.4, 2.8).

Alongside 4G came the global mass adoption of smartphones, necessitating extended mobile broadband, with 4G reaching mobile speeds of hundreds of Mbits/s. Usecases enabled and developed alongside 4G include smartphone apps, media streaming, mobile gaming, and even initial forays into emerging technologies e.g. augmented reality (AR), and Internet of Things (IoT).

Discussions on fifth generation (5G) of wireless telecom, termed 5G NR (Next Radio), began after the release of 4G LTE. Generally, three loose usecase categories can be identified for 5G. First is enhanced mobile broadband, with speeds reaching several Gbits/s. Secondly, massive machine-type communication, with ultra-low power and power-consumption. Finally, ultra-reliable low latency communication, for critical applications necessitating extremely high reliability. Significant technologies introduced for 5G include e.g. massive-MIMO with intelligent beamforming and broad-spectrum mmWave. [Dahlman et al., 2018].

Overall, in the past decades through five generations of wireless telecommunications, the technical advances and the mass-adoption of global standards have brought with it ubiquitous accessibility of the internet through mobile data. Datarates have gone from Kbits/s to Gbits/s and mobile coverage in many countries has become near-ubiquitous. Developments on future telecommunication standards are underway, with the industry on the path to '6G'. These future developments will continue unlocking more advanced usecases, with current trends showing no signs of slowing down.

2.2 Fundamentals of Wireless Transmission

As EM spectrum available for the purposes of wireless transmission is limited, efficient usage is required. Large frequency bands are regulated by international agreements and national laws, with bands allocated either for a specific use-case, as a free spectrum, or to specific operators. The latter is the case for bands used in 4G LTE and 5G, with the limited and expensive spectrum available to the operators necessitating advanced scheduling and multiplexing. This is covered in more detail in Section 2.7.

In addition to frequency, adequate physical coverage is also fundamental to modern wireless communications systems. By assuming that signal quality and strength are only maintained to an adequate level within a set area near an antenna 3G, 4G network operators accomplish this by splitting geographic areas into hexagonal 'cells'. Within each cell, network access is ensured by (one or more) transmitting basestations (BS), with directional antennas ensuring minimal interference and optimal power efficiency. This hexagonal pattern allows for cellular towers to be placed on sites at the intersection of three cells, with every 120 degree interval and specific frequency band servicing a cell with a basestation. A graphical representa-

tion of this can be seen in Figure 2.1.

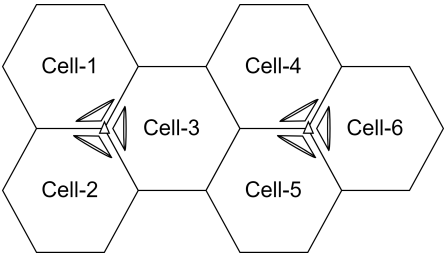


Figure 2.1 A representation of an ideal hexagonal-grid cell network, with cellular towers containing basestations transmitting signal directionally into the cells.

In reality, environmental considerations lead to the cells being only a rough approximation. Electromagnetic waves, depending on their frequencies, experience reflection, scattering, and diffraction. This causes large-scale fluctuations in signal quality, known as large-scale fading. [Tse and Viswanath, 2005] An example of this can be seen in shadowing, where geography can lead to regions physically 'close' to a transmitter being hidden from the transmitter. For a visualization of how this can impact the 'idealized' network cell layout, see Figure 2.2.

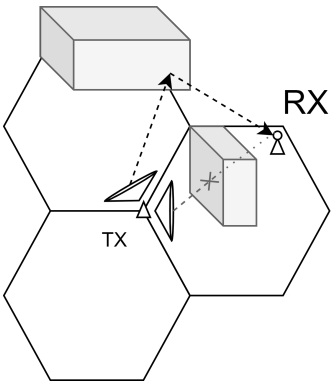


Figure 2.2 In a representation of an ideal hexagonal-grid cell network, an obstacle causes shadowing, leading to a receiver being connected to a transmitter of a network cell that the UE geographically is not physically in.

2.3 The Wireless Channel and Multipath propagation

In wireless communications, the transmission link between the transmitter (TX) and receiver (RX) is through a wireless propagation channel. This can be defined as the medium linking the TX and RX, where the properties of the medium largely determine theoretical information capacity and transmission behavior [Molisch, 2010].

A time-domain signal transmitted through the wireless propagation channel experiences reflection, scattering and diffraction and signal travel time. This leads to propagation delay, noise, and distortion for the signal. Overall, many of these can be viewed as multipath propagation phenomena in an environment with user mobility, EM noise, shadowing, and more.

Focusing in on multipath propagation, it is used to refer to signals propagating from a transmitter being able to reach a receiver through multiple pathways. From the receiver, the different travel lengths lead to the same signal traveling through different multipath components (MPC) varying in phase. Therefore, when adding up at the receiver, the signal can either constructively or destructively interfere with itself. See Figure 2.3 for a representation of this. The self-interference causes small-scale fluctuations in signal quality, leading to what is known as small-scale fading [Molisch, 2010].

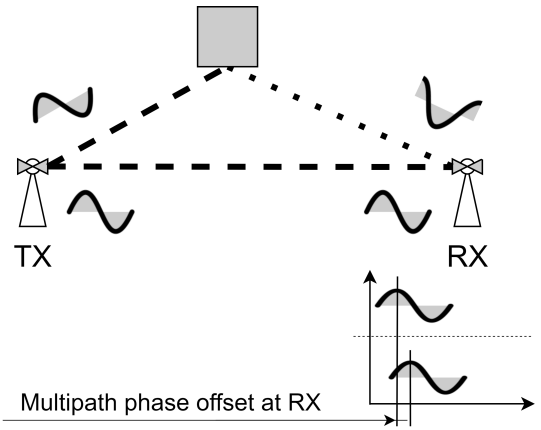


Figure 2.3 A multipath wireless propagation scenario from a transmitter (TX) and receiver (RX) involving two pathways. The different pathways lead to the same signal arriving with different phase at the RX, leading to constructive/destructive interference and small-scale fading.

The differing geographic lengths of the propagation paths and the relatively constant speed of electromagnetic waves mean the signal from the transmitter to the receiver arrives 'spread out' in time. The impact of this on the impulse response model is covered in Chapter 2.5.

2.4 Line-of-sight

Multipath propagation means the transmitter can be connected not only with line-of-sight (LoS) on the receiver, unobstructed by physical geometry or other phenomena, but also maintain with no direct line-of-sight (nLoS) - when obstructions exist. These two scenarios have different channel behaviors, with the LoS having a dominant secular component from the direct LoS MPC. Visual examples for NLoS and LoS scenarios in multipath propagation can be seen in Figure 2.4.

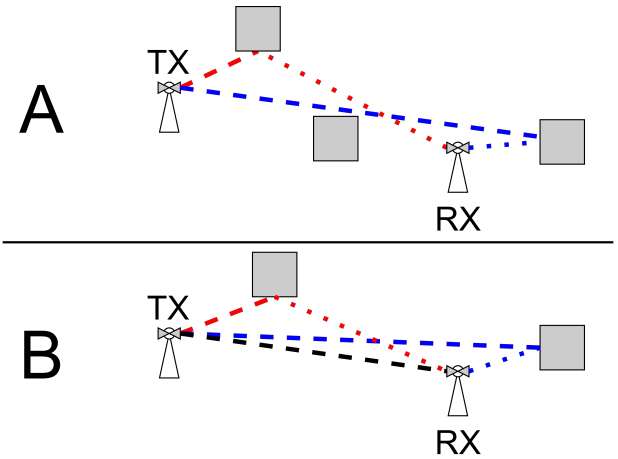


Figure 2.4 Potential NLoS (in subfigure A) and LoS (in subfigure B) propagation pathways from a TX and RX. LoS scenarios have a dominant specular component.

In a naive positioning scenario using simple triangulation for a NLoS case, measuring the received power and signal direction in the RX to derive TX angle and distance would lead to detecting a misleading TX position. E.g. Figure 2.5.

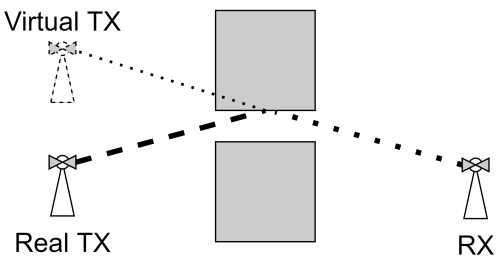


Figure 2.5 An NLoS scenario with a dominant reflection, where a naive positioning using network parameters would give misleading results.

2.5 Input/Output Impulse response model

Examining the transmission link as a system with an input signal from the transmitter and the output signal from the receiver, a statistical input-output model can be formulated from physical models (See Chapter 2 in [Tse and Viswanath, 2005] and Chapter 6 from [Molisch, 2010]).

The impact of multipath propagation is of particular importance for the input-output model, and can be understood intuitively by considering the time-domain of multipath propagation. The differing geographic lengths of the MPCs and the near-constant speed of electromagnetic waves mean the signal from the transmitter to the receiver arrives 'spread out' in time. Therefore, an input-to-ouput impulse response would no longer be a delta function [Molisch, 2010], and is modellable as a convolution of the input time-signal with the channel-specific time-varying impulse-response. Mathematically, this simplified input-output model can be expressed as in (2.1), giving a standard form of a linear time-varying system.

$$y(t) = \int_{-\infty}^{\infty} h(\tau,t)x(t-\tau)d\tau, \tag{2.1}$$

where $y(t)$ is the output signal, $x(t)$ the input signal, and $h(\tau,t)$ the time-varying impulse response of the channel (See Section 2.2.1 in [Tse and Viswanath, 2005]). To define the time-varying impulse response of a channel two useful assumptions for a statistical characterization of a channel can be introduced. The first is that the channel is Wide-Sense Stationary (WSS) and the second is that the channel's MPCs come from Uncorrelated Scatterers (US).

The WSS assumption is that the statistical properties - though not the fading realizations - of the examined wireless channel between the RX and TX are time invariant. Over a small enough area - around ten times the signal wavelength - the change can be considered small enough for WSS to hold [Molisch, 2010]. From this, with RX movement leading to changing position in time, one can define time intervals over which the WSS property can be assumed to hold as statistical properties experience only minor real change [Molisch, 2010].

With the WSS assumption, the time dependency of the impulse-response is due to the environment, receiver, and/or transmitter being non-stationary. In the special case where they are stationary, the impulse response becomes time-invariant.

The US assumption is that the phase of one MPC is uncorrelated with the phase of another MPC arriving with a different delay. Practically, the scatterers that lead to the MPCs for the RX have a random spatial distribution. The US and WSS assumptions together form the WSSUS assumption, with which the impulse response can be considered the result of a sum of the effect of N groups of MPCs.

The WSSUS assumption can be mathematically described as in (2.2), where $h_i(t,\tau)$ is the 'i' tap belonging to the MPC 'i', and $c_i(t)$ is the time-varying (if the system is nonstationary) complex coefficient corresponding to the MPC 'i'.

$$h(\tau,t) = \sum_{i=0}^{N} h_i(\tau,t) = \sum_{i=0}^{N} c_i(t)\delta(\tau - \tau_i) \tag{2.2}$$

Applying a Fourier transform to the impulse response $h(\tau,t)$, as shown in (2.3), shows that the effect of multipath propagation is that the transfer function of the channel changes with the channel frequency - meaning the subcarriers have different transfer-functions. See Section 2.7 for more on frequency-division.

$$H(f,t) = \int_{-\infty}^{\infty} h(\tau,t)e^{2\pi j\tau f}d\tau = \sum_{i=0}^{N}\int_{-\infty}^{\infty} h_i(\tau,t)e^{2\pi j\tau f}d\tau \tag{2.3}$$

In the special case where the system can be considered stationary, the $h(\tau,t)$ impulse response simplifies down to a time-invariant impulse response $h(\tau)$, as described in (2.4). This is particularly relevant for channel estimation, where the system impulse response can be considered time-invariant until a new sample is taken. This is covered in more detail in Section 2.9.

$$h(\tau) = \sum_{i=0}^{N} h_i(\tau) = \sum_{i=0}^{N} c_i\delta(\tau - \tau_i) \tag{2.4}$$

As a next step, environmental noise can be modelled as additive to this input-output model. In the simplest case, as a circular symmetric complex Gaussian. This gives us the continuous-time time-varying impulse response model from one transmitter to one receiver in (2.5), where $w(t)$ is the noise component.

$$y(t) = \int_{-\infty}^{\infty} h(\tau,t)x(t-\tau)d\tau + w(t) \tag{2.5}$$

This can be generalized to discrete time with a tap-delay spread T_d, for example with a number of l channel filter taps h_l. This gives us (2.6) in familiar form, with discrete time points m [Tse and Viswanath, 2005].

$$y[m] = \sum_{l} h_l[m]x[m-l] + w[m] \tag{2.6}$$

Giving a simple example, assuming a limited number L of taps l in a channel, transmitting some instantaneous signal $x[0]$ at time 0 without followup transmissions will result in the reciever y observing the output as seen in (2.7), with a discrete-time impulse response at time l [Tse and Viswanath, 2005].

$$y[l] = h_l[l]x[0] + w[l], \qquad l = 0,1,2,\ldots \tag{2.7}$$

For discrete-time narrowband channels, meaning when for a given signal x with a bandwith of W and a tap-delay spread of T_d where $T_d << W^{-1}$, (2.6) can then be approximated, as in (2.8). The approximation follows from the assumption that the

transfer function from (2.3) changes slowly, meaning that at any carrier frequency f_c, within a small enough frequency-band region around it the differences are negligible and the frequency transfer function can be considered to have only one 'tap'. This only holds in explicitly narrowband channels, or cases where the wideband can be viewed as a collection of distinct narrowband channels - e.g. the signals sent through the channel are orthogonal.

$$y[m] \approx h[m]x[m] + w[m] \tag{2.8}$$

In Section 2.8 on MIMO in 5G networks, this single receiver to single transmitter model will be generalized to multiple transmitters and multiple receivers, allowing for direction to be discriminated. For this, one can generalize the time-varying impulse response to take MPC directionality as additional input parameters. The two directions definable for a directional RX and TX are the Direction of Departure (DOD) Ω and the Direction of Arrival (DOA) Ψ. Directional components vary slowly compared to the phase, which due to fading changes rapidly.

Combining the DOA and DOD components with a generalized WSSUS condition in which MPC components from different directions fade independently, a system impulse response using MPC components similar to that described in (2.2) can be defined. This is again generalizable as the sum of MPC taps, and is described in (2.9) [Molisch, 2010].

$$h(\tau,t,\Omega,\Psi) = \sum_{i=0}^{N} h_i(\tau,t,\Omega,\Psi) = \sum_{i=0}^{N} c_i(t)\delta(\tau-\tau_i)\delta(\Psi-\Psi_i)\delta(\Omega-\Omega_i) \tag{2.9}$$

In certain cases if only the DOA is available and if assuming a stationary environment, TX and RX, then the double-directional time-variant impulse response in (2.9 can be simplified to a time-invariant single-direction impulse response as seen in (2.10). Additionally, the Ψ DOA, when viewing the RX sensor as a 2D surface, can be further broken down into longitude λ and latitude ϕ.

$$h(\tau,\Psi) = \sum_{i=0}^{N} h_i(\tau,\Psi) = \sum_{i=0}^{N} c_i\delta(\tau-\tau_i)\delta(\Psi-\Psi_i) \tag{2.10}$$

With a narrowband channel discrete model from (2.8) and using discretization of the DOA in latitude and longitude, the discrete-time DOA-dependant narrowband channel transfer function can be approximated as a single complex number, with the mathematical model described in (2.11).

$$y[m,\Psi_\lambda,\Psi\phi] \approx h[m,\Psi_\lambda,\Psi\phi]x[m] + w[m,\Psi_\lambda,\Psi\phi] \tag{2.11}$$

Note that in this thesis the details of baseband conversion are not covered, for more details on that see the books [Molisch, 2010] and [Tse and Viswanath, 2005]. All definitions of the impulse response in this chapter have been made without baseband conversion or notation.

2.6 Received signal power falloff with distance

As the goal of this thesis is a positioning system, one aspect worth investigating is how the received power falls with distance. This is because assuming a positioning system benefits from fewer nonlinearities and utilizes some sort of estimation based on per-beam received 'beam-energy', viewing the distance a beam 'travels' is a better proxy for position than just beam energy on its own.

On short distances, the free-space travel law holds and it makes intuitive sense that beam energy is inversely correlated with the second power of distance. However, the radio environment is not a 'clean' energy propagation environment due to obstructions and other physical phenomena, and on longer distances the free-space travel law does not accurately describe energy propagation - e.g., in NLoS scenarios, the effective distance travelled for a radio wave to reach the UE from the basestation might be much longer than a straight line through all the obstructions. Past a certain threshold distance the received signal power is thus proportional to anywhere from the 1.5-th to the 5.5-th power of distance [Molisch, 2010].

From empirical measurements, a decent statistical approximation of the received power falloff with distance for most environments a two-stage model, with an initial decay exponent of n=2 and after a break-point starting to fall off with a decay exponent of n=4 [Molisch, 2010]. This radio propagation model arises from the two-ray model, with one approximate LoS wave and one ground-reflected wave. The resulting approximate model can be seen in Figure 2.6. Note that more advanced statistical approximations exist, but this simple model will work for this thesis.

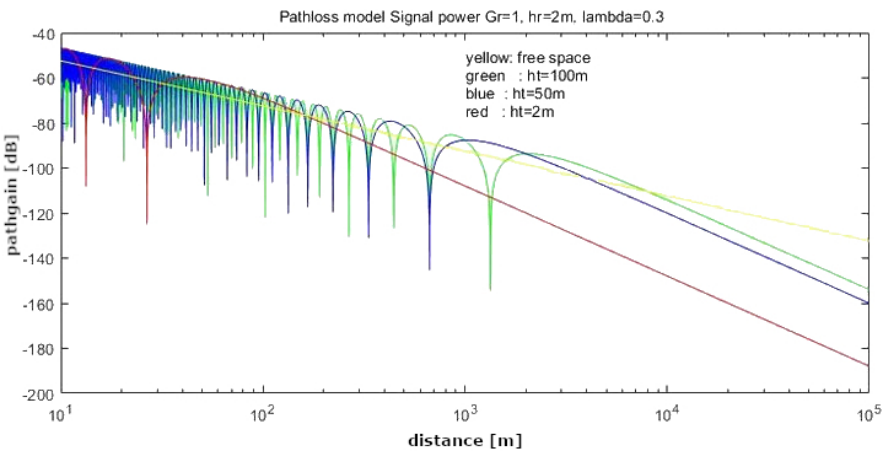


Figure 2.6 The statistical description of received signal power with the two-ray model with a RX height *hr* and a TX height *ht*. The different lines shows power falloff with distance in different TX height scenarios and in free-space. Taken from Fig 1. in [Omariba and Masese, 2019].

2.7 Multiple access transmission structure

For wireless systems, multiple devices - users - often communicate simultaneously in the same cell, requiring multiple access. More specifically within a network cell, time, frequency and space are shared between users. This is in addition to outside-cell transmissions potentially causing interference. With the limited time and frequency resources available for transmission, optimal usage of available space is critical. Furthermore, the uplink and the downlink - the transmissions from and to the UE and basestation - must also be distributed efficiently. In essence, multiplexing must be used to allocate resources precisely.

In past cellular systems, devices would communicate on for example different frequencies, with each user assigned a frequency (sub)band. This is known as Frequency-division multiple access (FDMA), and is an old and simple multiaccess method. FDMA is often used with Frequency Division Duplexing (FDD), where two (sub)bands are assigned to for each device, one for downlink transmissions and one for uplink transmissions. Another method was to assign timeslots to different users, known as time-division multiple access (TDMA) [Molisch, 2010]. Simplified representations of these are shown in Figure 2.7.

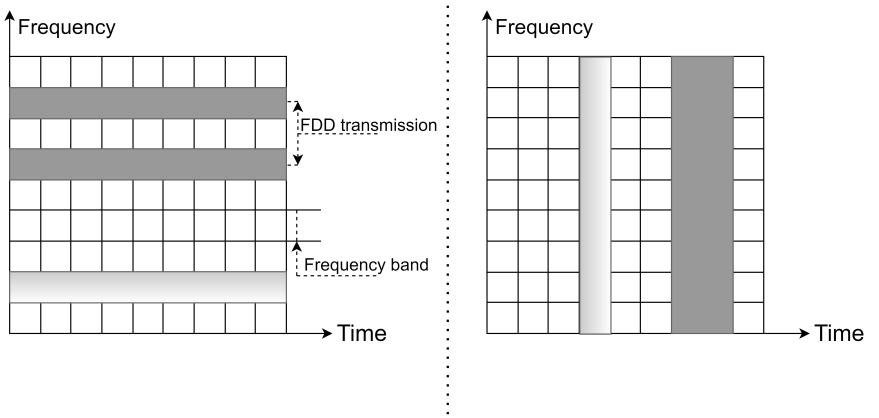


Figure 2.7 A visual representation of FDMA (left) and TDMA (right), with the filled-in areas of a specific shade representing a communication channel reserved for one user

For wireless networks in the 4G and 5G era, Orthogonal Frequency-Division Multiple Access (OFDMA) with its suptype of linearly precoded OFDMA (LA-OFDMA) is used to allocate resources. These methods use Orthogonal frequency-division multiplexing (OFDM), a specialized frequency-division multiplexing method using orthogonal (sub)carriers to minimize inter-carrier and inter-symbol interference [Molisch, 2010].

OFDMA combines the principles of FDMA with TDMA such that users are scheduled to use specific combinations of frequency bands to be orthogonal over a symbol duration, optimally utilizing the time-frequency grid. The subcarriers for a transmission can then be mathematically approximated as narrowband channels.

Scheduling in OFDMA is accomplished by subdividing the frequency-time grid into so-called resource elements, which are defined as one subcarrier for the duration of one OFDM symbol. [Molisch, 2010] These resource elements are then grouped into resource blocks, which are twelve subcarriers over the duration of one OFDM symbol. The symbol length itself is determined by subcarrier spacing and a required Guard Interval (GI), also called the Cyclic Prefix (CP). The GI/CP is a lengthening of the symbol length to ensure that the multipath components arriving delayed from the neighboring symbol do not interfere with the current symbol.

In summary, OFDMA transmissions to any single user are done with OFDM modulation on a scheduled number of resource blocks on orthogonal subcarriers. This can be seen on the time-frequency grid in Figure 2.8.

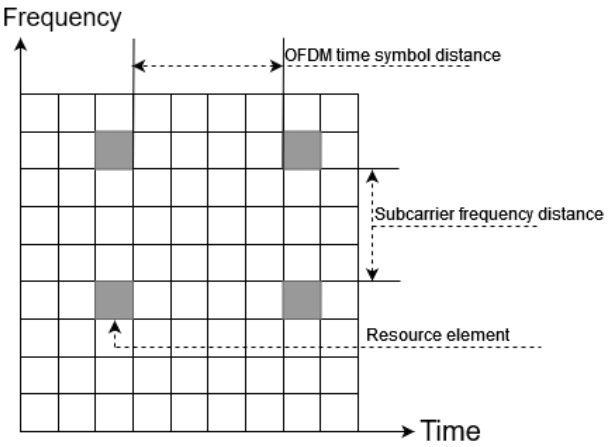


Figure 2.8 A visual representation OFDMA on the time-frequency grid. In this specific example, the filled-in resource elements are occupied by pilots.

Within a resource element, data is modulated using traditional modulation techniques such as Quadrature-Phase Shift Keying (QPSK) and Quadrature Amplitude Modulation (QAM). Bitrates then depend on the SNR of the user-specific channel.

With the advent of MIMO and directional beamforming - see Section 2.5 and the following Section 2.8 for more details - spatial division on top of OFDMA is also a possibility. The same resource block can be assigned to multiple spatially seperate users in a single cell such that their transmissions would minimally interfere. See Figure 2.9 for a visualization of MIMO-enabled spatial multiple-access.

2.8 MIMO and beam-centric design

In prior sections, Single-Input Single-Output (SISO) antennas have generally been assumed. Multiple-Input and Multiple-Output - abbreviated as MIMO - is the concept of having multiple antennas both on the RX side and the TX side for wireless transmission. Other combinations include single-input multiple-output (SIMO) and multiple-input single-output (MISO). These, when coupled with proper communication and statistical techniques, allow for more advanced transmission structures. The goal of this is generally to improve overall performance in wireless communication, though at the cost of increasing complexity.

To introduce MIMO methods in more detail, the concept of beamforming is critical. Beamforming refers to using various methods to direct wireless transmission energy; one example is by using several TX antennas to constructively interfere in specific ways. The 'beams' being formed can be thought of as location-aware constructive interference using MPC [Hampton, 2013].

Exploiting MIMO and beam-targeting, so-called spatial diversity and spatial multiplexing are possible. Spatial diversity refers to utilizing MIMO techniques to compensate for problems caused by different multipath components, essentially to improve wireless reliability and range. Spatial multiplexing refers to using multiple beams to transmit different information over multiple so-called 'beams', minimizing interference [Hampton, 2013]. A non-mathematical visualization of spatial multiplexing for multiple access can be seen in Figure 2.9.

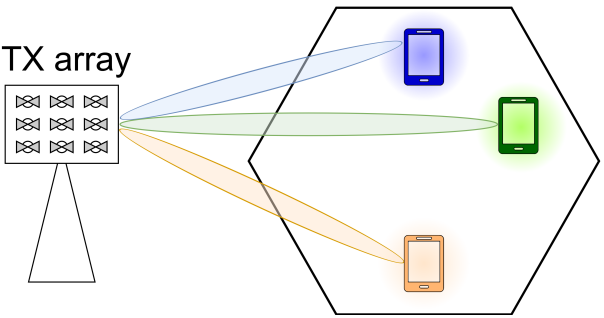


Figure 2.9 Abstract representation of spatial multiple access enabled by beamforming with a TX array, allowing parallel datastreams to multiple distinct UE.

To mathematically describe MIMO systems, an extension of the discrete-time narrowband impulse response/transfer function model from Section 2.5 in 2.8 can be introduced. With a generic MIMO setup consisting of N_{TX} number of TX antennas transmitting a single symbol over a narrowband (or equivalent) channel and N_{RX} number of RX antennas, the channel impulse responses $h[m]$ for the symbol transmission at time m from any TX antenna to any RX antenna have to be defined.

More specifically, for any given $i \in \{RX\}$ antenna at symbol time m, the symbol transmissions from all $j \in \{TX\}$ antennas are summed, each after being multiplied by their j TX to i RX antenna-to-antenna $h_{ij}[m]$ channel impulse responses. The result is (2.12) - where $y_i[m]$ is the output signal for any given i RX at symbol time m given all the $x_j[m]$ transmitted time m symbols. In addition, every RX antenna i has a noise $w_i[m]$ at symbol time m - this is often modelled as complex circular Gaussian.

$$y_i = \sum_{j=1}^{N_{TX}} (h_{ij}[m]x_j[m]) + w_i[m] \tag{2.12}$$

To describe the whole wireless channel from every TX antenna to every RX antenna at a given symbol time m in a single equation, introducing matrix notation is useful. The channel can then be described at symbol time m as in (2.17), with $\mathbf{H_m}$ channel matrix defined in (2.13), $\mathbf{R_m}$ recieved symbols defined in (2.14), $\mathbf{S_m}$ transmitted symbols defined in (2.15), and transmit antenna noise $\mathbf{W_m}$ in (2.16).

$$\mathbf{H_m} := \begin{bmatrix} h_{11}[m] & h_{12}[m] & \cdots & h_{1N_{TX}}[m] \\ h_{21}[m] & h_{22}[m] & \cdots & h_{2N_{TX}}[m] \\ \vdots & \vdots & \ddots & \vdots \\ h_{N_{RX}1}[m] & h_{N_{RX}2}[m] & \cdots & h_{N_{RX}N_{TX}}[m] \end{bmatrix} \tag{2.13}$$

$$\mathbf{R_m} := \begin{bmatrix} y_1[m] \\ y_2[m] \\ \vdots \\ y_{N_{RX}}[m] \end{bmatrix} \tag{2.14}$$

$$\mathbf{S_m} := \begin{bmatrix} x_1[m] \\ x_2[m] \\ \vdots \\ x_{N_{TX}}[m] \end{bmatrix} \tag{2.15}$$

$$\mathbf{W_m} := \begin{bmatrix} w_1[m] \\ w_2[m] \\ \vdots \\ w_{N_{RX}}[m] \end{bmatrix} \tag{2.16}$$

Finally, with the full MIMO wireless channel equation at symbol time m being defined as:

$$\mathbf{R_m} = \mathbf{H_m}\mathbf{S_m} + \mathbf{W_m} \tag{2.17}$$

Importantly, the channel matrix $\mathbf{H_m}$ describes the MIMO wireless channel, determining channel properties. Often the symbol time m is removed for brevity, so

matrices from (2.17) are denoted $\mathbf{H}, \mathbf{S}, \mathbf{R}, \mathbf{W}$. Additionally, the channel matrix $\mathbf{H}$ is often replaced by the normalized channel matrix $\mathbf{H^{norm}} := \sqrt{\rho}\mathbf{H}$, where ρ is the signal-to-noise power ratio at the RX. For brevity, in the rest of this section $\rho = 1$ is assumed, giving us $\mathbf{H^{norm}} = \mathbf{H}$.

To understand beamforming through the channel matrix, a spatial multiplexing method called eigenbeamforming is an intuitive example. An important assumption for closed-loop eigenbeamforming is that either the RX or TX has enough information about the rank-r $\mathbf{H}$ channel matrix to obtain the precoding matrix $\mathbf{V}_{N_{TX} \times N_{TX}}$ on the TX side and a decoding matrix $\mathbf{U}_{N_{RX} \times N_{RX}}$ on the RX side. $\mathbf{V}$ and $\mathbf{U}$ relate to the Channel Matrix $\mathbf{H}$ through (2.18). Eigenbeamforming then gives r independent SISO channels to transmit through - in short, spatial multiplexing.

Mathematically underpinning eigenbeamforming is the singular value decomposition (SVD) of complex matrices. First, as $\mathbf{H}$ is a $N_{RX} \times N_{TX}$ complex matrix of rank r, it is possible to apply SVD, where the singular values are denoted as $\{\sigma_i\}$, with $1 \leq i \leq r$, and $\sigma_1 \geq \sigma_2 \geq ... \sigma_r$. Using the Hermitian operator $\underline{H}$, we then get the SVD relation in (2.18), using the unitary matrices $\mathbf{U}_{N_{RX} \times N_{RX}}$, $\mathbf{V}_{N_{TX} \times N_{TX}}$ and the diagonal matrix $D_{r \times r} = diag\,(\sigma_1, \sigma_2, ..., \sigma_r)$.

$$\mathbf{H} = \mathbf{U}_{N_{RX} \times N_{RX}} \begin{bmatrix} D_{r \times r} & 0 \\ 0 & 0 \end{bmatrix} \mathbf{V}^{\underline{H}}_{N_{TX} \times N_{TX}} \tag{2.18}$$

All matrices in (2.18) can be computed using $\mathbf{H}$. The transmitter can precode the symbol transmit vector $\mathbf{S}$ from (2.15) using the precoding matrix $\mathbf{V}$, giving the precoded transmit vector $\mathbf{S_p}$ as seen in (2.19).

$$\mathbf{S_p} = V\mathbf{S} \tag{2.19}$$

At the RX, the received signal can be multiplied using the $\mathbf{U}$ matrix, giving the decoded recieved signal vector $\mathbf{R_p}$, as seen in (2.20) through substituting (2.17).

$$\begin{aligned} \mathbf{R_p} &= U^{\underline{H}}\mathbf{R} \\ &= U^{\underline{H}}\mathbf{H}\mathbf{S} + U^{\underline{H}}\mathbf{W} \end{aligned} \tag{2.20}$$

Bringing in transmit vector precoding from (2.19) and substituting in the SVD of $\mathbf{H}$ for (2.20), we get (2.21).

$$\begin{aligned} \mathbf{R_p} &= U^{\underline{H}}UDV^{\underline{H}}V\mathbf{S} + U^{\underline{H}}\mathbf{W} \\ &= D\mathbf{S} + U^{\underline{H}}\mathbf{W} \end{aligned} \tag{2.21}$$

Importantly, (2.21) gives r independent SISO channels from each of the transmitted symbols $\mathbf{S}$ to each of the recieved $\mathbf{R_p}$, accomplishing effective spatial multiplexing through MIMO.

2.9 Channel estimation

With the transmission structure defined in Section 2.7 and the concept of the impulse response model covered in Section 2.5, the final general concept to cover is how the TX/RX of any specific uplink and downlink transmission can estimate the channel properties in real-time.

Knowledge of channel properties - or the channel matrix **H** - is important for beamforming in MIMO (see Section 2.8). The concept of channel estimation is therefore to use various statistical methods and channel models to estimate the time-(in)variant channel impulse response of a wireless channel between the TX and RX [Apelfrojd, 2018].

One method for estimating the channel is through reserving some resource blocks in the channel for the user equipment and/or the basestation to transmit so-called pilot signals, which are known to both sides of the wireless communication channel. See Figure 2.8 for an example of how pilot signal resource usage could appear for an OFDMA communication system. Using pilot data for channel estimation is known as training-based channel estimation. The disadvantage of using training data is that resources have to be used for channel estimation. An alternative would be blind estimation, where no pilot sequences are used and instead channel state is statistically estimated using received data. In practice a blind approach has worse performance, and is seldom used [Hampton, 2013].

In the simplest of cases with no noise and a time-invariant channel, knowing the input and the output of a signal is sufficient to estimate the complex-valued time-invariant channel impulse response. In a more realistic scenario where the channel is time-variant and noise is non-negligible, the channel can never be perfectly estimated. Instead, at specific times, a statistical estimate can be made with or without priors on the channel and the noise appearance. The outdated channel can be compensated for by various channel estimate prediction methods [Apelfrojd, 2018].

A way to mathematically approximate the pilot-measurement equation for SISO channel estimation with data only at time τ, using the familiar discrete-time impulse-response model from Section 2.5, can be seen in (2.22).

$$y_K[\tau] = h_K[\tau]\Phi[\tau] + w[\tau] \tag{2.22}$$

, where $y_K[\tau]$ is a row vector with K elements and is the output signal at sampled time τ with K time-frequency measurement-location points. $h_K[\tau]$ is the channel impulse response, a K-element vector for the K time-frequency measurement locations. $\Phi_{K\times K}[\tau]$ is a matrix consisting of the known pilot signals transmitted at the K time-frequency (resource) measurement locations. As this is channel behavior, all values are complex. The noise $w[\tau]$ has identical dimensionality to $y_K[\tau]$, and is often modelled as a circular symmetric complex Gaussian. Assuming orthogonal time-frequency measurement locations with no inter-channel interference and a single antenna RX/TX, $\Phi[\tau]$ is a diagonal matrix [Apelfrojd, 2018].

Introducing MIMO from Section 2.8 into (2.22) using N_{TX} transmitters and N_{RX} receivers, the result is (2.23). $y_K[\tau]$ becomes the $\mathbf{R}_{N_{RX}\times K}[\tau]$ complex matrix, where each RX antenna has a dedicated time-frequency-resource row. $\Phi_{K\times K}[\tau]$ gains an increased dimensionality, becoming the matrix $\Phi_{(K\cdot N_{TX})\times K}[\tau]$, with the pilot signal transmitted from each antenna over each resource. $h_K[\tau]$ becomes a complex channel matrix $\mathbf{H}_{N_{RX}\times(K\cdot N_{TX})}[\tau]$ - the MIMO channel matrix on each of the K resources [Apelfrojd, 2018].

$$\mathbf{R}_{N_{RX}\times K}[\tau] = \mathbf{H}_{N_{RX}\times(K\cdot N_{TX})}[\tau]\Phi_{(K\cdot N_{TX})\times K}[\tau] + \mathbf{W}[\tau] \tag{2.23}$$

In practice, when measuring the downlink channel from the basestation TX to the UE RX for MIMO arrays, the 'true' N_{RX} is transformed from multiple RX antennas N_{RX} to discretized direction of arrivals Ψ_i. There are then N_Ψ number of discrete detectable direction-of-arrivals.

With directionality, the estimated $\mathbf{H}[\tau]$ channel matrix can be described as two-dimensional or three-dimensional. The two-dimensional case is similar to that specified previously, but with $\mathbf{H}_{N_\Psi\times(K\cdot N_{TX})}$. With three-dimensions, the MIMO antenna array is viewed as a 2D surface with discrete vertical and horizontal antennas, giving $N_{\Psi,h}$ number of discrete horizontal directions-of-arrival and $N_{\Psi,v}$ discrete vertical directions-of-arrival. The $\mathbf{H}[\tau]$ channel matrix in this case becomes $\mathbf{H}_{N_{\Psi,h}\times N_{\Psi,v}\times(K\cdot N_{TX})}$. If the channel is MISO, then $\mathbf{H}_{N_{\Psi,h}\times N_{\Psi,v}\times K}$ can be considered analogous to $h(\tau,\Psi)$ seen in Section 2.5.

In the above Equations (2.22, 2.23), we can see both that the number of unknowns can outnumber the number of equations, and that the noise at any time interval τ can distort the channel estimate. Therefore, measurements from multiple time intervals before τ and prior knowledge of the statistical properties of the channel must be used to obtain usable results. The purpose of channel estimation is therefore using the known pilot signal(s) Θ and the resulting received signal y at all resources and antennas for all time points up to the current τ, and estimating a current channel impulse response matrix $h_K[\tau]$, and/or a given integer-time-shifted channel impulse response matrix $h_K[\tau+m]$ [Apelfrojd, 2018].

The most common form of channel estimation is the family of linear estimators. Linear estimators allow for the estimation of any arbitrary $h_K[\tau+m]$ using a simple vector multiplication with a weight vector W, where every element is a matrix of the required dimensionality, and the vectorized notation for the y at all times up to τ. Different sub-types of linear estimators use different methodology to calculate the weight vector. Mathematically, linear estimators are described in (2.24), using elements $W[i]$ of the general weight vector W [Apelfrojd, 2018].

$$\hat{h}_K[\tau+m] = \sum_{i=0}^{\tau} W[i]\, y_K[i] \tag{2.24}$$

The specifics of linear estimators and pilot design are not covered in this thesis for brevity, to read more see [Apelfrojd, 2018], [Hampton, 2013], [Molisch, 2010].

2.10 Channel State Information

The broadcasting of pilot reference signals from the basestation when combined with channel estimation techniques from e.g. Section 2.9 enables the usage of closed-loop beamforming, modulation, code rate, etc...

Specifically, in 5G NR the basestation sends a reference signal termed the Channel State Information Reference Signal (CSI-RS). Once the downlink channel - termed Physical downlink Data Shared Channel (NR-PDSCH) - from the basestation to the UE is established, other reference signals such as the Demodulation Reference Signal (DM-RS) and the Phase Tracking Reference Signal (PTRS) are sent downlink. As a seperate downlink transmission, the Physical Broadcast Channel and with it the primary and secondary synchronization signals enable a UE to initially detect and then synchronize with a new cell.

In summary, once connection with the basestation is established, all the reference signals, but especially the CSI-RS are combined with channel sounding and estimation techniques to then acquire channel properties and Channel State Information (CSI). 5G NR specific CSI is termed CSI-NR. The UE then reports CSI-NR and other channel properties back to the base-station through reference signals on the uplink channel (PUSCH). Uplink reference signals include uplink DM-RS and PTRS along with the Sounding Reference Signal (SRS). For this thesis, the SRS is the most significant, and contains most general CSI, including the effect of multipath components, scattering, fading, signal power loss, etc. Through the uplink-transmitted CSI-NR, the basestation can close the transmission feedback loop, and ensure high-quality transmission. A visual summary of CSI feedback can be seen in Figure 2.10

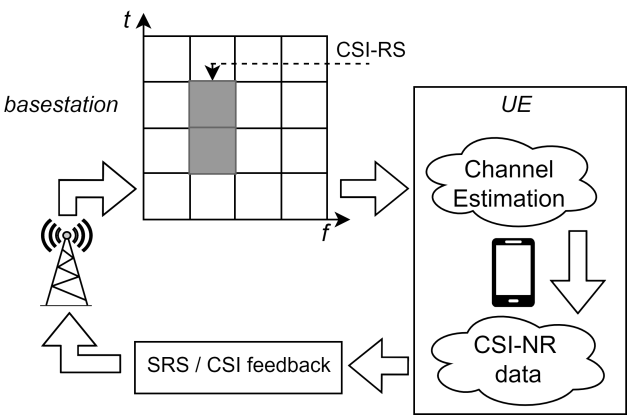


Figure 2.10 Representation of CSI-RS downlink transmission from a basestation; a UE obtains CSI-NR data through channel estimation, then reports the CSI back uplink through SRS and other reference signals.

The information contained in the CSI-NR includes, among others, the Channel Quality Indicator (CQI), the Precoding Matrix Indicator (PMI), and the Rank Indicator (RI). The specifics of the reported indicators and their contents depend on the codebook used and the details thereof are not covered in this thesis for brevity.

As an example of a CSI-NR use-case, the reported indicators can give a suggestion to the basestation of what according to the UE's internal state estimate would be a good suggested precoding matrix. As shown by the example in Section 2.8, a transmitted precoding matrix to the base-station enables beamforming. The basestation will then use the suggested precoding matrix and the and allocate resources according to the needs of the other users in the system to calculate a suitable precoding matrix. Due to the other users, the used precoding matrix might not be identical to what the UE suggested through its CSI-NR reporting.

As beamforming requires a good knowledge of the channel matrix $\mathbf{H}$, as shown in Section 2.8, the SRS report provides the basestation with a per-UE and per-resource-block channel estimate. In 5G there are 273 physical resource blocks (PRB) along the spectrum (see resource blocks in Section 2.7), in practice and on commercial-grade basestation hardware from Ericsson AB, only a subset of PRB channel estimates are updated per SRS report.

The significance of CSI and channel estimates for positioning, as per the topic of this thesis, comes from the dependence of the channel matrix estimate on position. This correlation, as explored briefly in Section 2.3, is due in part to the multipath phenomena, radio propagation pathways through the environment, and on travel distance. Such dependence on location invites speculation on using the CSI-NR feedback system to determine the position of each UE from their reported channel estimate. Furthermore, as on varying channel frequencies radio waves propagate in different ways, channel estimates on high MIMO beam/antenna counts spanning the entire 5G spectrum provide enormous amounts of data to use in a positioning algorithm. An outline of a positioning system can be seen in Figure 2.11

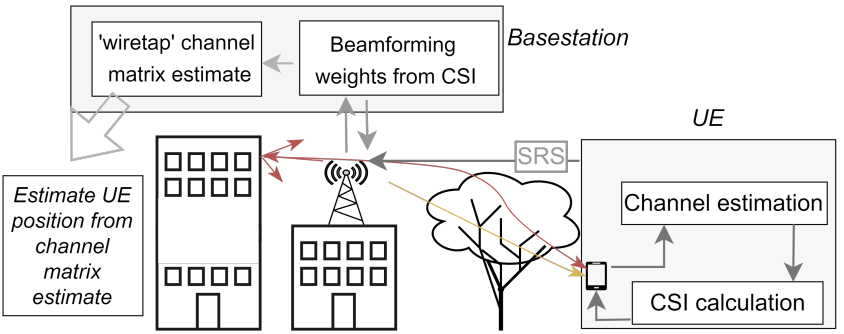


Figure 2.11 Theoretical positioning feedback-loop for determining UE position using SRS feedback of CSI-NR

3

Coordinate Systems, GPS Positioning

3.1 UE Positioning and GNSS

Positioning and localization refers to determining the position of a body in relation to a known fixed reference point within a defined coordinate system. Optionally, other qualities such as direction of motion, orientation, velocity, acceleration can also fall under the category of positioning, such as when positioning is used for navigation. Exploiting different physical phenomena, a large variety of positioning systems have been developed. A trivial example of using wireless communications as a positioning system is by taking the location of the connected basestation as an approximate for the UE in question.

Overall, positioning systems can be categorized three ways as mentioned in [Groves, 2013]: Real-time or post-processed, static or mobile positioning, and self-positioning and remote positioning. Ideally, 5G wireless systems would be usable for remote basestation-based real-time positioning of mobile UE devices. In essence, to track the location of individual non-static UEs through their transmitted channel matrices.

A necessary component to evaluate and training machine learning-based positioning systems is to use a prior known ground truth position. An example of this could be to use predetermined fixed positions in a well-controlled measured environment, or to use another existent positioning system. Constraining data recording to fixed locations might limit real-world applicability of a positioning system, as characteristics originating from natural movement can be lost.

In the case for using an existent positioning system for data collection of outdoors positioning, satellite navigation is the most practical choice. Satellite navigation systems are widely available, and provide decent accuracy in realistic conditions. A subtype of these systems, termed Global navigation satellite systems (GNSS) are freely available for civilian use and widespread in networked devices, serving as the basis for most civilian navigation software available. Usage of GNSS

necessitates careful selection of measurement environment, as system precision varies heavily from environmental factors. [Zhu et al., 2018]

The principle of satellite navigation and GNSS specifically is to have at least four satellites detectable by a device at any point on earth. The satellites are on inclined orbits known as 'constellations', where calibration is done through onboard atomic clocks and on-ground stations. For a visualization of how GNSS constellations appear from space-based and ground-based observer, see Figure 3.1.

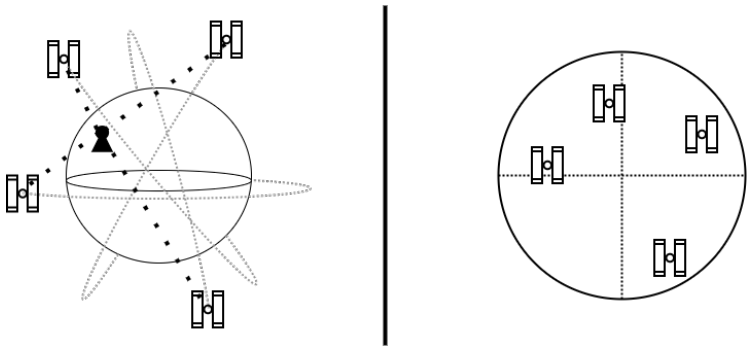


Figure 3.1 A representation of a GNSS constellation of 4 satellites transmitting to a ground-based reciever from space-view (left) and ground-view (right)

GNSS satellites transmit radio signals which can be picked up by equipment on earth that use passive ranging and do not transmit any information back to the satellites [Groves, 2013]. The information transmitted by the satellites include highly-accurate timing parameters and orbital position information, enabling high-precision positioning from the UE. For more reading on the specifics of satellite navigation and of navigation more generally, see [Groves, 2013].

As previously mentioned, local environmental factors can serve to increase uncertainty in exclusively GNSS-based positioning systems. Obstructions can block or distort signals, reflections can give false path lengths. For this reason, modern outdoors positioning and navigation systems use a combination of positioning systems to provide a more reliable position estimate. Additionally, signals from different GNSS systems, such as GPS and Gallileo can be fused to further increase reliability.

With their wide array of sensors and GNSS compatibility, smartphones provide an ideal tool for obtaining position data. Conveniently, tracking the position of such GNSS-capable UE devices through their channel state information is the very goal of this thesis. Therefore, the use of GNSS-based outdoors positioning as ground truth for testing and training wireless network and ML-based outdoors positioning is pertinent, with both the position estimate and uncertainty obtainable along with corresponding data time-stamps.

3.2 The coordinate frame

To position an object, one must construct a pair of coordinate frames. First, an arbitrary point on the object must be selected as the origin. A set of non-planar axes corresponding to the positional degrees of freedom must also be defined on the origin, e.g. with direction of motion of the object being the first axis. Finally, a reference gives relative position for the origin and its fixed axes. The reference is also a point with a set of non-planar axes. A coordinate frame can also be understood as a coordinate system with an associated set of measurements that can be used to described objects within that coordinate system. The selection of which coordinate frame in a positioning pair is the origin and which is the reference is arbitrary and trivially interchangeable. The reference and the tracked object's origin with their corresponding axes comprise the coordinate frame pair defining the position of an object. If the axes of both the origin and the reference are orthogonal, then they are termed orthogonal coordinate frames [Groves, 2013].

For positioning of an object in 3D space to a reference point within a given orthogonal coordinate frame, there are six degrees of freedom; three for object position and three for orientation. E.g. in Cartesian coordinates x, y, z determine position while the Tait–Bryan angles of ψ, θ, ϕ determine orientation. A simplified 2D representation of UE positioning in a coordinate frame can be seen in 3.2, with four degrees of freedom.

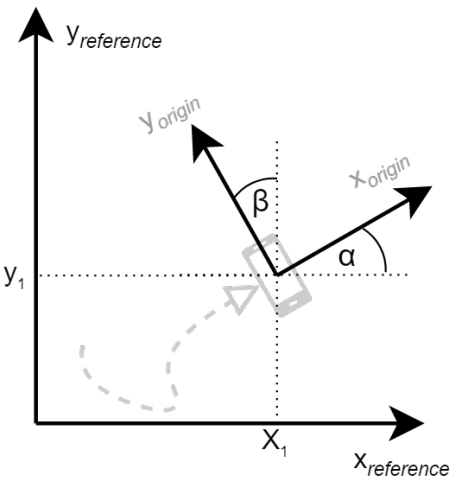


Figure 3.2 Positioning of an object in a 2D space using a 2D coordinate frame. The object origin is the center of mass of a UE, and the first axis of the origin is the direction of motion of the UE.

3.3 Local Tangent Plane Coordinate System

Commercially available GNSS systems output curvilinear coordinates as seen in the Appendix Figure 11.1, using different global ellipsoid models for the coordinate frame, such as the World Geodetic System 1984 (WGS84) or the International Terrestrial Reference Frame (ITRF) [Groves, 2013].

Curvilinear coordinates are well suited for global positioning but bring with them redundant mathematical complexities for local positioning. On small-enough scales the interest is more in the position from some local reference point in terms of Cartesian coordinates, rather than a global state in Geodetic coordinates. If measurement noise is larger than the modelling errors from treating the surface of earth as non-curved, a transformation onto a local Cartesian frame brings only minimal distortion. By convention, local Cartesian coordinate frames have orthogonal axes on the tangent plane pointing East-North-Up (ENU) [Drake, 2002]. The ENU realization of the local tangent plane coordinate frame can be seen in Figure 3.3.

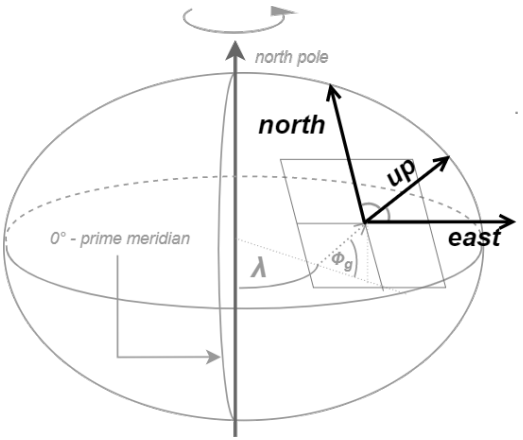


Figure 3.3 East-North-Up (ENU) local tangent plane coordinate system

To convert small changes from curvilinear coordinates to a local tangent, an intermediary conversion into small changes in earth-centered earth-fixed (ECEF) coordinates is needed, done by applying Taylor expansion about latitude, longitude and height on (11.2). This can be seen in (11.1). Having brought the tangent-projected small changes on the ellipsoid into ECEF coordinates, a rotation matrix as seen in (3.1) brings the coordinates from ECEF (dx, dy, dz) to ENU (de, dn, du) coordinates [Drake, 2002].

$$\begin{pmatrix} de \\ dn \\ du \end{pmatrix} = \begin{pmatrix} -\sin\lambda & \cos\lambda & 0 \\ -\sin\Phi_g\cos\lambda & -\sin\Phi_g\sin\lambda & \cos\Phi_g \\ \cos\Phi_g\cos\lambda & \cos\Phi_g\sin\lambda & \sin\Phi_g \end{pmatrix} \begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} \tag{3.1}$$

4

Particle Filtering

4.1 Bayesian filtering of underlying states

While developing a positioning system using 5G channel matrix feedback, viewing position as an isolated observation purely based off any single obtained channel matrix is a naïve approach. In a more realistic case, each UE will be moving through the world according to some underlying kinematic system dynamic, with the position snapshots obtained by processing the channel matrix $\mathbf{H}$.

Assuming meaningful position information is contained within the channel matrix and partially extracted, the processed position estimates can almost be viewed as noisy observations of the underlying position state. It is therefore pertinent to utilize state behavior and prior observations in addition to the current observation to filter for a more accurate estimate of the true current state. *Filtering* in this context refers to using observations up-to and including the current observation to find the underlying state at the current time.

The family of approaches for recursively obtaining an estimate of an unknown state probability density function using past observations is known as *Recursive Bayesian estimation*. For system state estimation in a Markovian process with no inputs, state and measurement noise, and uneven sample-times, a system can represented using the state equations as seen in (4.1), where x_k is the process state at time t, Δt_k is the elapsed time between sample $k+1$ and sample k, v_k the state noise, $f(x_k, v_k, \Delta t_k)$ the system dynamics, e_k the measurement noise, and y_k the measurement. A Markovian system is one in which the probability of a state in step k depends only on the state in step $k-1$, with the general Markov model and observation model seen in (4.2)

$$\begin{aligned} x_{k+1} &= f(x_k, v_k, \Delta t_k) \\ y_k &= h(x_k) + e_k \end{aligned} \tag{4.1}$$

$$\begin{aligned} x_{k+1} &\sim p(x_{k+1}|x_k) \\ y_k &\sim p(y_k|x_k) \end{aligned} \tag{4.2}$$

If the assumption is made that the system behavior and state x_k is Markovian (or is approximately Markovian), then according to Bayesian estimation principles the prior of x_k can be obtained through the Chapman-Kolmogorov equation, as seen in (4.3) [Papoulis, 1984]. Note that an irregular sampling rate in an otherwise Markovian system does not break the Markovian property.

$$\begin{aligned} p(x_k|y_{1:k-1}) &= \int p(x_k|x_{k-1}, y_{1:k-1})\, p(x_{k-1}|y_{1:k-1})\, dx_{k-1} \\ &= \int p(x_k|x_{k-1})\, p(x_{k-1}|y_{1:k-1})\, dx_{k-1} \end{aligned} \tag{4.3}$$

Using (4.3) for the state prior update step when combined with (4.4) as the posterior updates and with the posterior update (4.5) forms the basis of recursive Bayesian estimation algorithms.

$$p(x_k|y_{1:k}) = \frac{p(x_k|y_{1:k-1})\, p(y_k|x_k)}{p(y_k|y_{1:k-1})} \tag{4.4}$$

$$p(y_k|y_{1:k-1}) = \int p(y_k|x_k)\, p(x_k|y_{1:k-1})\, dx_k \tag{4.5}$$

For linear or linearizable system dynamics with known near-zero-mean-Gaussian noise v_t, e_k, Kalman filters or related Extended Kalman Filters and Unscented Kalman filters are optimal and commonly used [Julier and Uhlmann, 2004] [Gustafsson, 2010]. This is because the minimum mean square estimate and its covariance can be calculated through an analytic calculation using the linear state dynamics and the known noise characteristics, giving a finite-dimensional representation of the posterior distributions.

For positioning using 5G channel matrices with machine learning as per the process outlined in Section 2.9, Kalman Filtering is unfeasible. Though the kinematics of movement have well-known dynamics that are suitable for linearization and therefore Extended Kalman filters, the observations from machine learning systems generally end up having extremely non-zero-mean-Gaussian and time-varying noise characteristics.

Modelling the state changes in a system with highly-non-Gaussian noise can instead be done through numerical approximations, e.g. point mass filters where a discretized state-space is used to numerically estimate the prior, enabling arbitrary non-Gaussian and non-linear systems to be approximated. However, point mass filters suffer from quadratic complexity in grid size, and do not enable higher-resolution grids in the parts of the state-space where more information is contained. Another more common numerical approximation method that enables a sort of 'dynamic grid' are Sequential Monte Carlo methods - also known as Particle Filters (PF) [Gustafsson, 2010].

4.2 Particle Filters

Particle filters create an evolving posterior estimate through sampling a large set N_p of simulated particles $\left\{P^i\right\}_{i=1}^{N_p}$ that each have a time-evolving internal state representation x_k^i and an associated weight $w_{k|k}^i$, where the weight represents the posterior: $w_{k|k}^i = p\left(x_{1:k}^i|y_{1:k}\right)$ and $\sum_{i=1}^{N_p} w_{k|k}^i = 1$. The posterior probability $p(x_{1:k}|y_{1:k})$ can then be approximated with N_p particles as shown in (4.6) [Gustafsson, 2010].

$$p(x_{1:k}|y_{1:k}) \approx \sum_{i=1}^{N_p} w_{k|k}^i \delta\left(x_{1:k} - x_{1:k}^i\right) \tag{4.6}$$

When viewed over time, the set of $\left\{P^i\right\}_{i=1}^{N_p}$ particles form a set of N_p state-trajectories. The particles, the observations, and the system dynamics then approximate the probability distribution of the state x_k at time k. The advantage of particle filters is that the state probability distribution can have any form, all that is required is knowledge of system dynamics, v_k and e_k - to calculate $p\left(y_k|x_k^i\right)$ and $p\left(x_{k+1}^i|x_k^i\right)$. The basic mathematical description of the update process for a particle in a particle filter can be seen in (4.7), with the measurement update step in (4.8) where c_k is the normalization weight [Gustafsson, 2010].

$$\begin{aligned} p\left(x_{1:k+1}^i|y_{1:k}\right) &= p\left(x_{k+1}^i|x_{1:k}^i, y_{1:k}\right) p\left(x_{1:k}^i|y_{1:k}\right) \\ &= p\left(x_{k+1}^i|x_{1:k}^i\right) w_{k|k}^i \end{aligned} \tag{4.7}$$

$$\begin{aligned} w_{k|k}^i &= \frac{1}{c_k} w_{k|k-1}^i p\left(y_k|x_k^i\right) \\ c_k &= \sum_{i=1}^{N_p} w_{k|k-1}^i p\left(y_k|x_k^i\right) \end{aligned} \tag{4.8}$$

Then, sampling with importance using a proposal distribution q, chosen to reflect the 'value' of a sample as shown in (4.9) gives the ability to adjust the posterior according to the importance, as shown in (4.10) [Gustafsson, 2010]

$$x_{k+1}^i \sim q\left(x_{k+1}|x_k^i, y_{k+1}\right) \tag{4.9}$$

$$\begin{aligned} p\left(x_{k+1}|y_{1:k}\right) &= \sum_{i=1}^{N} \frac{p\left(x_{k+1}^i|x_k^i\right)}{q\left(x_{k+1}^i|x_k^i, y_{k+1}\right)} w_{k|k}^i \delta\left(x_{1:k+1} - x_{1:k+1}^i\right) \\ &= \sum_{i=1}^{N} w_{k+1|k}^i \delta\left(x_{1:k+1} - x_{1:k+1}^i\right) \end{aligned} \tag{4.10}$$

4.3 Particle filter in practice

Resampling, Stratified resampling

Particle filters also suffer from the curse of dimensionality; constant coverage of more states requires exponentially more particles. This requires a minimal number of states to be kept track of per particle. Intelligent resampling techniques enable the particles with more importance to be duplicated, and particles with less importance to be destroyed - thereby decreasing the computational complexity of the PF, though the number of states must still be kept relatively low. One common resampling technique is *stratified resampling* [Kitagawa, 1996].

Stratified resampling aims select samples in a fairly uniform way - between 0 and 2/N apart, where N is the number of samples. This is done by dividing the cumulative sum of the weights into N parts, then randomly sampling one particle from every part, according to which weight it would belong to. Stratified sampling is fairly common in literature, and it will be used in this thesis.

Algorithm

With a brief overview of the mathematics in Section 4.1, a practical implementation of a particle filter with a changing sample-time can be seen in Algorithm 1.

Algorithm 1: A basic implementation of a particle filter

Result: time-vector of state estimates $\hat{x}_{1:M}$
Input: Particle count N_p
Input: Observed state extractor from particle: $Px()$
Input: Observation model and noise (vector) $meas_{1:M}()$
Input: State noise and dynamics(vector) simulator $sim_{1:M}()$
Input: Resample indexing algorithm $Resample()$
Data: Observation vector $y_{1:M}$, sample time vector $\Delta t_{1:M}$
Initialize: particles: $P_k := \left\{P^i\right\}_{i=0}^{N_p} \leftarrow initialize(y_0)$;
Initialize: weights: $w_{1:k} \leftarrow \frac{1}{N_p}$;
Initialize: output vector: $\hat{x}_{0:M} \leftarrow 0$;
for k = *1 to* M **do**
  Simulate particle motion with noise: $P_k \leftarrow sim_k(P_{k-1}, \Delta t_k)$;
  Update weights from measurement model: $w_k \leftarrow meas_k(y_k, P_k)$;
  Normalize weights: $w_k^i \leftarrow \frac{w_k^i}{\sum_i w_k^i}$;
  Obtain state estimate: $\hat{x}_k \leftarrow \frac{1}{N_p} \sum_{i=0}^{N_p} w_k^i Px(P^i)$;
  Resample particles: $P_k \leftarrow P_k\left[Resample\left(w_k\right)\right]$. ;
end

5

Machine Learning with CSI

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. [Mitchell, 1997]

5.1 Introduction to Machine Learning

Machine learning (ML) can be viewed as the study of algorithms that can 'learn' approximations of the underlying systems using interaction, or generated/recorded data, even if the underlying systems are highly complex and/or nonlinear. The idea is that for simpler systems mathematical or physical modelling and simulations work well, more complex real-world problems often do not have an elegant solution, or have a solution too time-intensive to exactly calculate. In such cases, assuming sufficient quantities of good-enough quality data can be obtained, approximate solutions generated by machine-learning algorithms can fill the gap.

The relation of the Channel matrix with regards to relative position of the UE and the basestation is highly complex and nonlinear. Considering this, for positioning from CSI-NR data, as shown in Figure 2.11, ML algorithms seem pertinent.

ML algorithms most often take the form of an over-parameterized function, where the parameters are optimized through some iterative algorithm over a function that measures how well the ML model is performing. A very general description of this function is shown in Figure 5.1, where: E is the measured performance, P_m^T is an instantiation of the performance measuring function P for task T with hyperparameters m, f is the ML model, D is the measured dataset or environment.

$$E = P_m^T(f, D) \tag{5.1}$$

The desired final result is a system that can generalize from the training environment into real-world applications. In summary, ML can be considered as a way for systems to 'intelligently' learn an environment, find underlying mathematical structures, and to be able to meaningfully extend the rules learned from the training process to the real-world [Géron, 2019].

5.2 Supervised learning

Considering the breadth of the field, the first concept to explore in ML with regards to CSI positioning is the specific category within which positioning algorithms can belong to. There are several ways to categorize ML systems. The definitions and categories vary in literature, with different criteria leading to different categorizations that can be combined at-will to specify ML behavior [Géron, 2019]. For positioning in CSI data, the task is to use known pairs of CSI data and corresponding location at time of recording, and learn how to predict a continuous position for future CSI data. The format of the training data means that CSI-based positioning would be considered a form of supervised regression.

In supervised learning, input datapoints are given along with known labels for each datapoint. The ML system is then set to 'learn' associations between the input data and their known labels - or some other rule requiring information about both. Regression means that some continuous information associated with each datapoint is the desired output from the ML algorithm. In literature, the data is usually denoted as $\mathbf{X}$, with the first dimension corresponding to the number of datapoints. The associated labels/values are then denoted as $\mathbf{Y}$, with an identical first dimension. The performance metric (5.1) most commonly used for supervised regression is the loss-function, with a smaller 'loss' output corresponding to a better fit to the data. This takes the form of (5.2), where l is the loss, J_m the loss-function instantiated with meta-parameters m, the ML supervised regression model f with parameters θ, the unvalued data X, and the associated data values Y.

$$l = J_m\left(f(X,\theta),Y\right) \tag{5.2}$$

Therefore, a supervised regression algorithm will seek to minimize the loss function J_m over the dataset $D = \mathbf{X} \cup \mathbf{Y}$ through iterating over the parameters θ during training. Mathematically, the training procedure for regression to obtain optimal ML parameters can be described as in (5.3).

$$\theta = \arg\min_{\theta}\left[J_m\left(f(\mathbf{X},\theta),\mathbf{Y}\right)\right] \tag{5.3}$$

Mean squared error (MSE) and various derived errors functions is one of the most common of the error functions used for supervised regression in literature. The simplest form of mean-squared error can be seen in (5.4), which shows the optimization procedure using MSE for a supervised regression model f with parameters θ, where X_i and Y_i denote the i-th element of the unvalued part of the dataset $\mathbf{X}$ and associated values data $\mathbf{Y}$.

$$\theta = \arg\min_{\theta}\left[MSE\left(f(\mathbf{X},\theta),\mathbf{Y}\right)\right] = \arg\min_{\theta}\left[\frac{1}{n}\sum_{i=0}^{n-1}\left(f(X_i,\theta)-Y_i\right)^2\right] \tag{5.4}$$

5.3 Optimization for Machine Learning

Looking at (5.3), it is clear that training machine learning for regression becomes a sort of optimization problem, where a ML model with behavior set by parameters θ is optimized to minimize a loss function with training data X, Y. For simple optimization problems such as linear regression (Section 5.6), a closed-form solution can be found that exactly determines the optimal parameters. In most cases however, the optimum can not be exactly calculated. Instead, iterating parameters using local gradients on the cost function result in a step-by-step descent in loss, which after some number of steps gets arbitrarily close to a local or global minimum. This is known as gradient descent, e.g. in Figure 5.1 using local gradients of some arbitrary non-convex function. For convex optimization, the minima found is always the global minimum. However, as most ML models are non-convex, generally some local minimum is found instead.

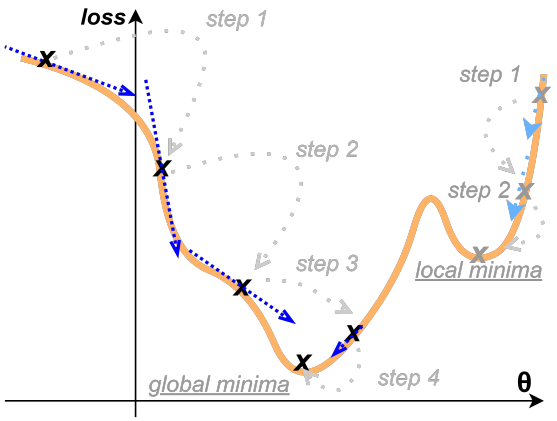


Figure 5.1 An example of a gradient descent algorithm on an arbitrary non-convex loss-function with two separate initializations. One ends up settling at the global minimum, while the other at a local minima.

As long as the data used for training is representative of the future use-case, and assuming highly unrepresentative local minima are rejected, ML models finding local minima is not necessarily a problem. The goal of ML is to create a model that generalizes outside the data used for building the model. Therefore, obtaining the optimized regression model on CSI data as described in Section 5.2 is in truth using the loss-function as a surrogate optimization problem to hopefully approximate the wider unknown problem. Especially for ML models with high complexity, most local minima are good approximates of the overall global minimum [Goodfellow et al., 2016].

Stochastic Gradient descent

Of the gradient descent algorithms used in practice for ML, one of the simplest is known as Stochastic Gradient Descent (SGD) [Robbins and Monro, 1951]. The idea behind SGD is twofold. First, it is computationally faster to calculate an approximate gradient using a randomly selected small subset of independent samples from the data than using the full dataset. Next, it introduces a stochasticity to the system, enhancing generalizability through functioning as a sort of regularization - see Section 5.5. The algorithm for SGD can be seen in Algrithm (2), and serves as an example of a stochastic optimization algorithm [Goodfellow et al., 2016].

Algorithm 2: Stochastic Gradient Descent

Result: Optimal parameter vector θ
Input: Learning rate ε, with some epoch-based update rule $U(\varepsilon, epoch)$
Input: number of epochs N_{epochs}, number of batches per Epoch N_{batch}
Data: Training dataset $X_{training}, Y_{training}$, minibatch size M_{batch}
Data: Initialized parameter vector θ
for $e = 0$ *to* $\left(N_{epochs} N_{batch}\right)$ **do**
  Sample elements $X_t^{(i)}, Y_t^{(i)}$ from $X_{training}, Y_{training}$;
  Calculate estimated gradient: $\hat{g} \leftarrow \frac{1}{M_{batch}} \nabla_\theta J_m \left(f(X_t^{(i)}, \theta), Y_t^{(i)} \right)$;
  Update parameters: $\theta \leftarrow \theta - \varepsilon \hat{g}$;
  Update learning rate: $\varepsilon \leftarrow U(\varepsilon, e)$;
end

ADAM

Though SGD is still widely used in practice, many other stochastic optimization algorithms build upon SGD - of which one of the more popular ones is ADAM [Kingma and Ba, 2014]. ADAM belongs to a family of adaptive learning rate optimization algorithms, combining advances from prior algorithms such as RMSProp and Momentum-SGD [Goodfellow et al., 2016]. Momentum methods can be described as taking the moving averages of past gradients of iterations to obtain a 'velocity' term that moves the iterations forward in some direction at some speed - thereby speeding convergence. Adaptive learning rate algorithms dynamically adjust the learning rate ε through calculations on the gradient. Adaptive learning therefore reduces the number of hyperparameters to tune.

The performance of ADAM is not necessarily superior to that of SGD [Choi et al., 2019]. Improvements, such as in the method 'AdamW' [Loshchilov and Hutter, 2017] combined with AMSGrad [Reddi et al., 2018] close the performance gap of ADAM compared to SGD, while remaining advantageous w.r.t convergence and hyperparameter tuning. Therefore AdamW-AMSGrad is sufficient for this thesis, and will be the utilized algorithm. See the Appendix, Algorithm (5) for more details.

5.4 Dataset management

In addition to the optimization procedure, the full training procedure is also important to briefly consider. In most supervised ML models, there are two sets of parameters to tune: the parameters within the ML algorithm proper, and the so-called meta-parameters, which determine the architecture of the ML algorithm, along with its training behaviour. To obtain accurate information on the performance of an ML algorithm, a dataset must be used that has not been used to optimize the parameter or metaparameter set. Therefore, the training process of any given supervised-learning algorithm requires at the very least three different sets of data [Géron, 2019]:

- The **training dataset**, on which the ML optimizes its training parameters through the appropriate loss-function.
- The **validation dataset**, on which the performance of a ML algorithm is tested, for any arbitrary instantiation of hyperparameters. Therefore, the validation dataset is then used to find an optimal combination of hyperparameters.
- The **test dataset**, which is used to judge the performance of the final ML model. The test dataset is left untouched until the final performance measurement.

A training process for supervised regression ML can be seen on Figure 5.2. The models are trained on the training dataset and evaluated on the validation dataset for hyperparameter adjustment. The training-validation cycle iterates until satisfactory validation performance is achieved. Thereafter, the algorithm is optionally retrained on the combined validation and training datasets, and evaluated on the test dataset.

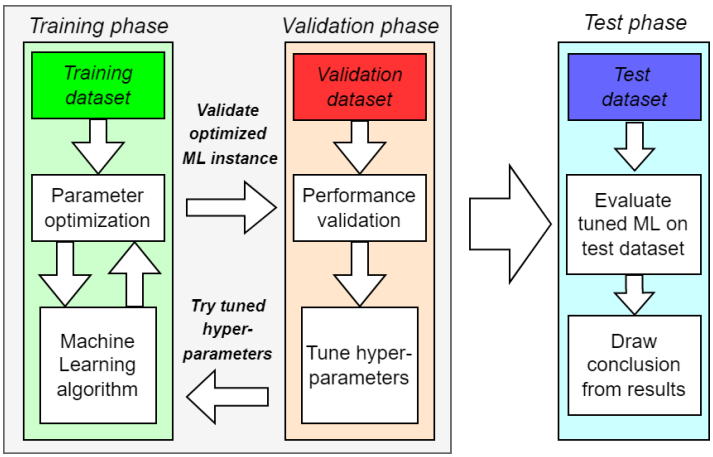


Figure 5.2 The usage of training, validation, and test data in ML

Critically, though the test dataset is a less-optimistic estimate for real-world performance than the validation or training datasets are, certain data acquisition conditions - such as identical measurement setups - can still lead to the results from a test dataset being non-representative of real-world performance. Though there are at a minimum three required datasets, if desired additional datasets can be defined to get a further refined measure of true performance. For example, there can be multiple validation datasets, each optimizing a certain set of hyperparameters.

With regards to CSI positioning, using channel matrices, as seen in Chapter 2 to regress for position obtained through UE-based GPS signals converted to local relative ENU coordinates, as seen in Chapter 3, requires at minimum a training, a test, and a validation dataset. The form and dimensionality of these can be seen in Equations (5.5) With position denoted as Y. The first dimension in both sides of the data pair is the number of measurements, denoted M.

Training dataset:

$$\left[\mathbf{H}^{Training}_{M_{Training}\times(K\cdot N_{TX})\times N_{\Psi,h}\times N_{\Psi,v}}\right] \cup \left[(Y_{ENU})^{Training}_{M_{Training}}\right]$$

Validation dataset:

$$\left[\mathbf{H}^{Validation}_{M_{Validation}\times(K\cdot N_{TX})\times N_{\Psi,h}\times N_{\Psi,v}}\right] \cup \left[(Y_{ENU})^{Validation}_{M_{Validation}}\right] \tag{5.5}$$

Test dataset:

$$\left[\mathbf{H}^{Test}_{M_{Test}\times(K\cdot N_{TX})\times N_{\Psi,h}\times N_{\Psi,v}}\right] \cup \left[(Y_{ENU})^{Test}_{M_{Test}}\right]$$

To simplify the underlying system the ML algorithm must approximate, and to reduce training times, for the purposes of this thesis ENU local coordinates from a nearby relative frame can simply be viewed as EN coordinates (east-north). This holds if within the datasets minimal elevation differences are present - for the reference point, the mean elevation of the ENU position training data could serve as a good zero-point.

Another consideration to make is the complex-valued nature of the input data for Channel matrix estimates. Existing implementations of ML algorithms are seldom made with complex values in mind; therefore, to keep development time low, initial experiments will use only the amplitude of the complex data. Additionally, when transitioning from antenna-space to beam-space, part of the phase is already used to reconstruct directionality. Therefore, even if only using amplitude in beam-space, complex phase is still used in the pre-processing step.

In summary, using a separated training-validation-test datasets with the extracted channel matrices and the corresponding GPS EN measurement of a local coordinate frame to feed into ML algorithm is the framework for ML-enabled 5G CSI positioning. If needed, additional datasets for further iteration on e.g. trajectory smoothing could be created.

Cross-validation

One extremely common and simple method for getting the same data to 'go further' with regards to finding the ideal ML model hyperparameters is to not just split the data into training validation and test datasets, but to do several training and validation splits. Essentially, the training and validation datasets are combined, usually shuffled, and then split to create N folds - or data sub-sets. Each fold then contains $\frac{1}{N}$-th of the original data.

After splitting the data, each individual fold is designated as a validation set in its own iteration, with the rest of the non-validation folds in a given iteration used to train the model for that iteration. The performance of the model is then evaluated on the fold designated for the given iteration as the validation set. The final model cross-validation score can then be put to be the mean of each iterations' validation losses. A visual example of the $N = 5$-fold cross validation process-flow can be seen in Figure 5.3.

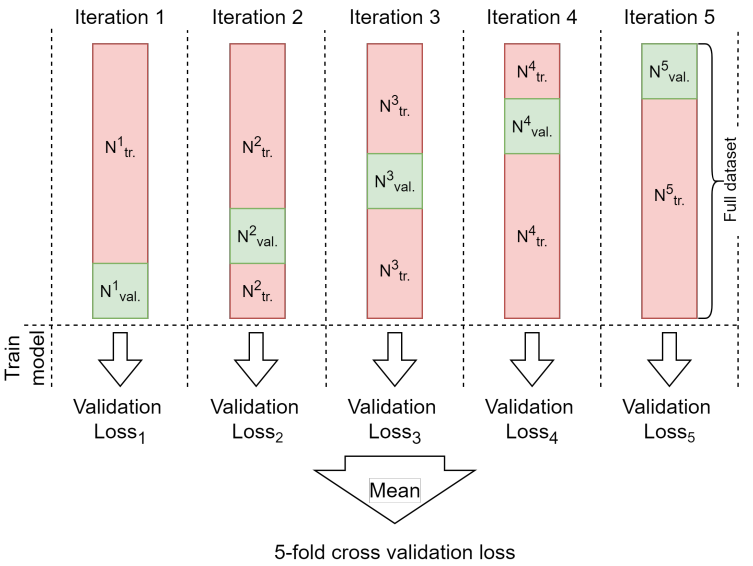


Figure 5.3 A visual demonstration of obtaining a validation performance of an ML model using 5-fold cross-validation

A disadvantage of CV is that while it does allow a larger dataset to be effectively used as a validation dataset, it also results in a model requiring far more processing time to train than with a simple training-validation split. For this reason, in this thesis CV will only be used on the 'classical' machine learning models described in Section 5.6. Neural nets are too compute-intensive to cross-validate using the resources available to write this thesis.

5.5 Overfitting and regularization

Considering the oftentimes over-parameterized nature of ML algorithms, some of the lack of generalizablility in ML systems come from finding patterns in the training data that are not representative of the wider dataset. This leads to a dilemma of ML; higher-complexity systems can potentially model more elaborate systems, but also can find 'false' patterns in the data. This phenomenon is termed overfitting, in which a mathematical/physical model of a system optimized for found data utilizes variance/patterns inherent to the data, but not to the underlying model [Géron, 2019]. A simple example of overfitting can be seen in Figure 5.4, where polynomial regression of a higher order fits better with the training data, but with the resulting model being a poor fit to the true underlying model.

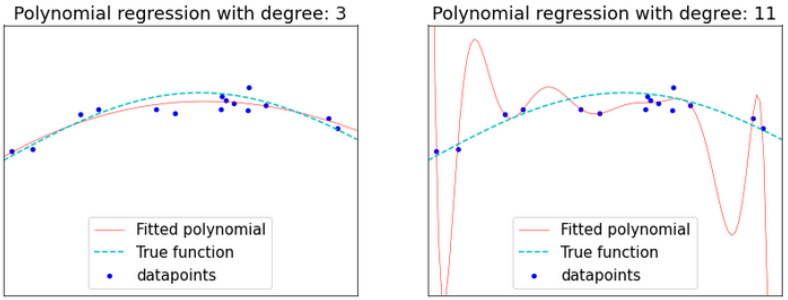


Figure 5.4 A polynomial regression of degree 11 overfits to data, while a lower-degree polynomial scales better outside the training data.

Overfitting leads to decreased predictive power on data outside the domain of the training data, especially if the new data is obtained through slightly different measurement/simulation setups. See also Occam's razor, which is a principle that for two models of differing complexities but identical known predictive performance, the model with less complexity is generally preferable.

To detect overfitting, and to tune the model to improve real-world performance while decreasing training performance, the separated datasets in Section 5.4 can be used. They provide a more realistic indication of ML performance. Reserving data for validation and testing means that less data available for training, which decrease the final network performance - assuming no further adjustments are done. However, it enables the tuning of model hyperparameters, including those that control regularization, a term for strategies that tackle overfitting.

There are many regularization methods, some specific to certain algorithms, others applicable across different ML fields. In this section, some of the more generalizable ones will be covered, while model-specific regularization methods will be covered in the relevant (sub)sections. A selection of methods for reducing overfitting and improving performance in ML will be discussed in the below subsections.

Early stopping

One of the simplest methods to reduce overfitting for iteratively optimized ML models, it consists of stopping the parameter optimization loop over the training data before a (local) minima is found for the loss-function - the number of training iterations can be considered as a hyperparameter inherent to the training process. The assumption behind this method is that a local minimum in the loss-function over the training dataset does not exactly correspond to the local minimum over the 'real-world' task.

During the training process, the mismatch in the training and validation sets appears as after the error initially falls dramatically along with the training data error, the validation error appears to reach a 'plateau', after which it slowly begins to rise again - even as the training data error falls further.

A trivial example of an early-stop algorithm is to set the training to stop some empirical number of iterations away. In practice, the most common form of early stopping is to simply record the parameters associated with the lowest validation error, and return those instead of the final model parameters.

Data-based regularization

- **More data:** Increasing the amount of training data from the same domain can greatly reduce the possibility of overfitting. Even outside improving overfitting, simply obtaining more data is unreasonably effective at improving results [Géron, 2019].

- **Task Sharing:** Widening the domain of the training dataset by increasing the number of related but different 'tasks' a model must optimize for can intuitively lead to the ML model to find the underlying data structures, rather than learn just noise. However, bringing in data that falls too far out the predicted use-case can instead lead to the opposite problem of underfitting, where model complexity is too low to accurately capture the behavior of a larger underlying system [Géron, 2019].

- **Data preprocessing, feature selection:** Manipulating the data before feeding it through the training proceedure can also be a tool to reduce overfitting. Filtering out outliers, applying proper normalization, reducing noise characteristics, selecting/creating/merging known useful features, etc.

- **Data augmentation:** While more data is the best way to get better results, obtaining more data is also one of the hardest parts of ML. However, a way around this is to artificially 'create' data from existing points. For certain applications of ML, such as image recognition, this is trivial - a flipped image of a bird still shows a bird. For others, there might be no obvious solution.

Noise injection

A simple alternative and arguably a sub-type of data augmentation, noise injection consists of artificially injecting noise at various parts of the model [Goodfellow et al., 2016]. There are, depending on the algorithm, different parts of the data pipeline that noise injection can be done to, and are the next discussed points.

- **Output noise injection:** At times, the output/label value Y in the dataset is not the 'real' ground truth. All measurement systems have error, and therefore even if the X input from the dataset is a good representation, maximizing with respect to Y might not be optimal for the 'real' underlying system. One way to model this is to inject noise modelling the uncertainty in the Y values. E.g. for training an outdoors positioning system using GPS data as ground truth, output noise injection would seem a conducive setup, as GPS is a known imperfect measurement of position.
- **Input and hidden variable noise injection:** Similar to output noise injection, modelling the probability distribution of input variables can be done by injecting noise. In addition, certain ML methods (such as deep learning) utilize hidden variables that feed from one part of the model to another; these variables can also have noise injected.

Explicit regularization

A term for a family of regularization methods, explicit regularization adds a so-called 'regularization term' to the loss-function. This regularization term can 'punish' certain scenarios indicative of overfitting. An example regularization term can be seen for a generalized loss-function in supervised ML in (5.6), where $\tilde{J}_\alpha$ denotes the regularized loss-function with the $\alpha \in [0, \text{inf})$ hyperparameter weighing the importance of the norm penalty function Ω. The size of the weights in a weight-based ML model is then typicall penalized by Ω [Goodfellow et al., 2016].

$$\min_\theta \left[\tilde{J}_\alpha\left(f(\mathbf{X},\theta),\mathbf{Y}\right)\right] = \min_\theta \left[J\left(f(\mathbf{X},\theta),\mathbf{Y}\right) + \alpha\Omega(\theta)\right] \tag{5.6}$$

The two most common forms of explicit regularization are L_1 and L_2 regularization. As only weight-based models are covered in this thesis, for brevity L_1 and L_2 regularization will only be shown for weight-based models, and ignoring bias.

- L_2 **regularization**: In literature L_2 is also commonly known as Tikhonov regularization or weight decay. For weight-based ML models, the regularization consists of penalizing the loss according to the square of the L_2-norm of a vector containing the weights, meaning: $\Omega(\theta) = \frac{1}{2}\|w\|_2^2$. The resulting regularized loss-function can be seen in (5.7), where the weight-based supervised regression model is denoted f, with the vector w denoting weights. The overall result of L_2 is a shift of the optimal minima during optimization, decreasing the effect of weights that contribute little to the reduction of loss.

$$\tilde{J}_{\alpha}^{(L_2)}(f(\mathbf{X},w),\mathbf{Y}) = J(f(\mathbf{X},w),\mathbf{Y}) + \alpha\frac{1}{2}\|w\|_2^2 \tag{5.7}$$

- L_1 **regularization**: Less common than L_2 regularization, L_1 for weight-based ML models consists of penalizing the loss according to the L_1-norm of a vector containing the weights, meaning: $\Omega(\theta) = |w|_i$. The resulting regularized loss-function can be seen in (5.8). The overall result of L_1 is different than that of L_2, and is that weights that contribute little are shifted towards 0 - leading to a sparse weight vector.

$$\tilde{J}_{\alpha}^{(L_1)}(f(\mathbf{X},w),\mathbf{Y}) = J(f(\mathbf{X},w),\mathbf{Y}) + \alpha\sum_i |w|_i \tag{5.8}$$

Ensemble methods

Ensemble methods are another way to tackle overfitting. The concept behind ensemble methods is simple; a group of well-performing but dissimilar models, if they have their results aggregated in some way to create an ensemble model, may often outperform any single of its constituent models in the validation and test results. The background behind this is that model errors, if they are somewhat uncorrelated, can 'cancel out' if a sufficiently large number of models are aggregated [Goodfellow et al., 2016].

Therefore, for an effective ensemble method, the models used must be trained separately and with enough differences in their training outcome for errors to be uncorrelated. This can be achieved by e.g. using models with different architectures or hyperparameters, a different loss function, or simply where the optimization procedure does not always converge to one (or a few) solution points, leading to differing final models even if the hyperparameters are identical [Goodfellow et al., 2016].

Notably, neural networks (see Section 5.7) do not tend to converge to identical solution points [Goodfellow et al., 2016]; stochastic elements to the learning process along with high model complexity result in a large variety in model outcome. Stochastic elements include e.g. randomized initial states, minibatch randomization, SGD, stochastic data augmentation, etc... From this, neural network-based algorithms can utilize ensemble methods on identical architectures, so long as they are trained separately and in a non-deterministic manner.

Due to the effectiveness of ensemble methods, in published academic papers on neural networks ML models are generally compared according to their performance when not in an ensemble [Goodfellow et al., 2016].

5.6 Non-Neural Network Machine-Learning algorithms for regression

Literature on positioning using CSI data often use various forms of neural networks (See Section 1.2), but for tests on data viability and to sanity-check results from machine-learning, a few common regression methods will also be examined.

- **Linear Regression:** The $\hat{Y}$ predicted values are obtained by multiplying X with a weight matrix θ and adding a bias - shown for a multivariate $\hat{Y}$ in (5.9). θ is found by optimizing over the training dataset $Y_{training}, X_{training}$ for the MSE (5.4). Linear regression has a closed-form solution for the weights θ, shown in (5.10) - where $X_{tr}, Y_{tr} := X_{training}, Y_{training}$.

$$\hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{N_y} \end{bmatrix} = \theta^T X = \begin{bmatrix} \theta_{0,1} & \theta_{1,1} & \cdots & \theta_{N_x,1} \\ \theta_{0,2} & \theta_{1,2} & \cdots & \theta_{N_x,2} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{0,N_y} & \theta_{1,N_y} \cdots & \theta_{N_x,N_y} & \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_{N_x} \end{bmatrix} \tag{5.9}$$

$$\theta = \left(X_{tr}^T X_{tr}\right)^{-1} X_{tr}^T Y_{tr} \tag{5.10}$$

- **Nonlinear regression:** Expanding on linear regression, the same algorithm can be used to create a nonlinear model. A simple way to accomplish this is by using nonlinear operations on the input features X, then applying linear regression on the resulting expanded feature vector. E.g. polynomial regression of order n consists of taking up to the n-th polynomial of every input feature as a new feature.

- **LASSO regression:** Short for *Least Absolute Shrinkage and Selection Operator Regression*, LASSO regression consists of introducing the L_1 regularization to (linear) regression [Géron, 2019]. The resulting optimization equation can be seen in (5.11), replacing the MSE (5.4).

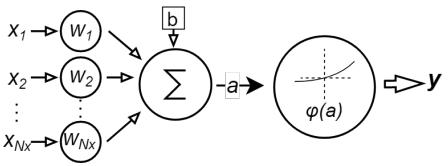
$$\forall e \in \left\{1, 2, \cdots, N_y\right\} : \theta_e = \underset{\theta_e}{\arg\min} \left[MSE\left(\theta_e^T X, Y_e\right) + \alpha \sum_i |\theta_e|_i \right] \tag{5.11}$$

- **Elastic-Net Regression:** Similar to LASSO regression, Elastic-Net regression introduces regularization to the MSE loss of regression. In the case of Elastic-net regression, both L_1 and L_2 regularization is applied [Géron, 2019]. The resulting optimization equation can be seen in (5.12), which holds for $\forall e \in \left\{1, \cdots, N_y\right\}$ elements of Y_e, and where r is the mix ratio controlling the relation between L_1 and L_2.

$$\theta_e = \underset{\theta_e}{\arg\min} \left[MSE\left(\theta_e^T X, Y_e\right) + \alpha \frac{1-r}{2} \|\theta_e\|_2^2 + \alpha r \sum_i |\theta_e|_i \right] \tag{5.12}$$

5.7 Introduction to Deep Learning

Artificial Neural Networks (ANN), underlay much of machine learning. ANNs are directed graphs where nodes are artificial neurons (AN), with values flowing across graph edges. ANs generally consist of an activation function with a weighted sum of input values and a bias term. An example AN can be seen in Figure 5.5, where x_k is the k-th input, w_k the corresponding input weight, b the bias, a the activation function input, ϕ the activation function, and Y the output. During optimization the weights, bias, and occasionally the activation function is fitted.



$$a = b + \sum_{k=1}^{N_x} x_k w_k$$
$$y = \phi(a)$$

Figure 5.5 An artificial neuron and the corresponding equation

A common graph architecture for ANNs is to arrange the ANs in 'layers'. A common example of this arrangement is feedforward ANNs with each layer neuron inputs originating from data of previous layer outputs. The result is information propagating forward through the ANN. For optimization the gradient of the loss-function of the entire ANN propagates backwards from the output, in a process known as *backpropagation* - see [Mitchell, 1997] for more on backpropagation.

When there are more than two layers in layer-based architecture, the term *Deep Learning* is commonly used, e.g. for the deep feedforward ANN in Figure 5.6.

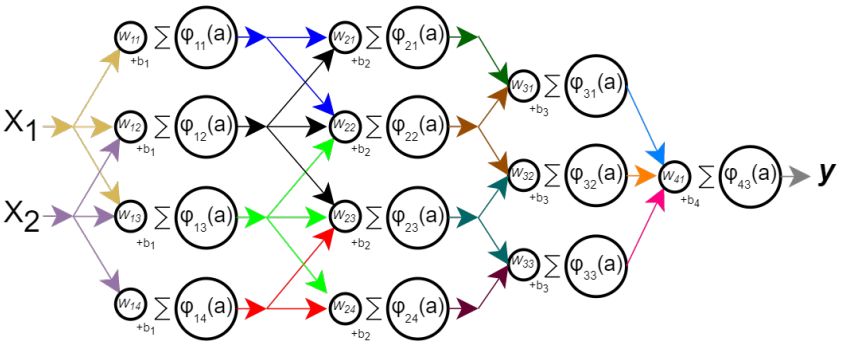


Figure 5.6 A deep feedforward ANN, where the notation is generalized from Figure 5.5 to layers, e.g. W_{ln} is the weight vector for the n-th neuron in the l-th layer.

5.8 Deep learning

Due to the universal approximation theorem (e.g. for rectified linear unit feedforward networks, see [Sonoda and Murata, 2017]), ANs with nonlinear activation functions in a large-enough network of the right architecture can approximate any arbitrary function. This partially explains the applicability of deep-learning for ML, making them a favorable choice to approximate complex models.

Complex underlying physical phenomena, high feature numbers and large quantities of data lead to CSI data being inherently favorable for deep-learning. The question then becomes the optimal architecture to use for CSI data. Due to the complexity of finding an optimal architecture, a certain amount of intuitive reasoning is necessary. The specific chosen architectures are discussed in Chapter 8. For this, certain pre-defined layer, cell, and activation function types that are applied in this thesis for positioning in CSI are discussed in this section.

Fully connected layer: The simplest layer, the fully connected layer means that all neuron outputs from the previous layer feed into every neuron on the current layer. The matrix output of a fully connected layer can be seen in (5.13), where l is the current layer number, h_l is the output vector of layer l, ω_l is the matrix containing all input weights of all neurons in layer l, ϕ_l is the activation function of all neurons in layer l, and b_l is the bias term. The layer indication can also be placed in the vector/matrix exponent with parentheses, e.g. h_l can also be labeled as $h^{(l)}$

$$h_l = \phi_l\left(\omega_l^T h_{l-1} + b_l\right) \tag{5.13}$$

$$\omega_l^T = \begin{bmatrix} (\omega_l)_{1,1} & (\omega_l)_{1,2} & \cdots & (\omega_l)_{1,N_{l-1}} \\ (\omega_l)_{2,1} & (\omega_l)_{2,2} & \cdots & (\omega_l)_{2,N_{l-1}} \\ \vdots & \vdots & \ddots & \vdots \\ (\omega_l)_{N_l,1} & (\omega_l)_{N_l,2} & \cdots & (\omega_l)_{N_l,N_{l-1}} \end{bmatrix}, \qquad h_l = \begin{bmatrix} (h_l)_1 \\ (h_l)_2 \\ \vdots \\ (h_l)_{N_l} \end{bmatrix}$$

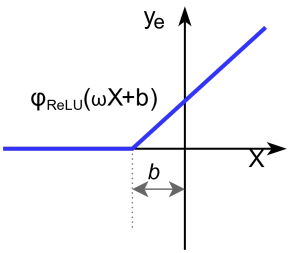
The fully connected layers, when chained together, form a fully connected block. The equation for a fully connected block is simply (5.13) nested within itself k times, where k is the number of layers in the given block. E.g, (5.14) has the equation of a 3-layer fully-connected feedforward block.

$$h_{out} = \phi_3\left(\omega_3^T \phi_2\left(\omega_2^T \phi_1\left(\omega_1^T h_{in} + b_1\right) + b_2\right) + b_3\right) \tag{5.14}$$

Within a single fully-connected block, the primary considerations when defining the architecture then become the neuron counts of each layer, the depth (layer count) of the block, and the type of activation function used.

Partially connected layer: Similar to the fully connected layer, only some weights are set to zero - meaning the variables are blocked from propagating down a certain number of graph edges.

Activation Function - ReLU An exceedingly simple but effective nonlinear activation function, Rectified Linear Units (ReLU) output the maximum of either a linear function of the inputs multiplied by weights plus the bias or 0, whichever is bigger. The node equation and graph is shown in Figure 5.7.



$$a = b + \sum_{k=1}^{N_x} x_k w_k$$
$$y_e = \phi(a) = max(0, a)$$

Figure 5.7 A ReLU node and the corresponding equation

Dropout regularization Dropout is a regularization method through network node manipulation. During training, a layer labeled to have a 'dropout' with a $p \in [0, \text{inf})$ will, for each neurons/nodes in the layer, have a p probability of dropping out. In SGD and similar optimization procedures where data is fed in small batches during the training process, the dropped neurons from dropout are re-randomized every batch. In summary, randomly selected neurons in the layer will output 0 for that mini-batch, thereby creating random permutations of partially connected layers from fully connected layers during the training process. An example of dropout can be seen in Figure 5.8.

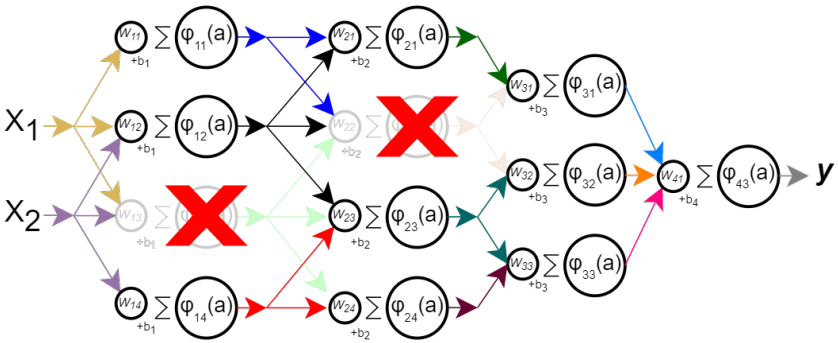


Figure 5.8 Dropout of $p = 0.25$ on the deep feedforward ANN from Figure 5.6

5.9 Fingerprinting and data collection

The general thought behind positioning with CSI channel matrix estimates is that a location roughly corresponds to a specific CSI channel matrix. In other words, locations in space have an approximate 'fingerprint'. The data collection process for fingerprinting can be done through two different methods: either designating locations in space on a grid and collecting data on these gridpoints, or a more natural 'continuous' position can be recorded through emulating the natural positioning scenario - in summary, assigning continuous position to datapoints recorded through natural movement. See Figure 5.9 for a visual explanation of the two categories.

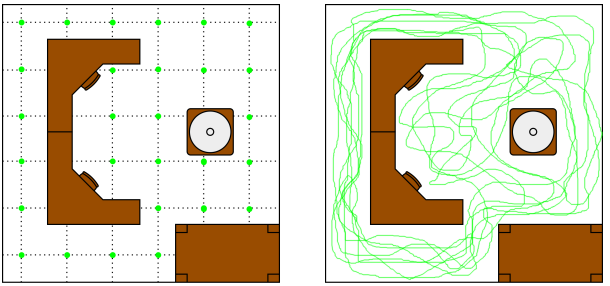


Figure 5.9 Location data collection for fingerprinting in an office environment: left for grid-based location collection on the green points, right for continuous position data collection along walking pathways.

The advantage of the continuous data collection approach over grid-based training data is that a dataset containing continuous trajectory data is more true to how data will be collected in a practical use-case, and how real-world UE positioning will look like. In addition, collecting continuous position data based on realistic navigation scenarios enables the usage of time-series position/trajectory estimation methods integrated into the training loop.

There are a number of advantages of the grid-based approach, leading to why it is often used in literature. Firstly, it allows for easier automation of data collection during training. Furthermore, unlike for the continuous position data collection approach, grid-based collection can compensate for the issue of position data bias: discrete grid points are easier to measure in a way that no region is disproportionately weighted in terms of number of datapoints. Another disadvantage of continuous position estimation is that of ground truth accuracy - With gridded points, it is easy to know the 'true' ground truth of positions, even with a flawed positioning system. This does not hold for continuous position measurement.

Despite the advantages of grid-based data collection, for this thesis continuous datapoint collection was chosen specifically to try to mirror realistic navigation conditions - and to integrate the time-element into the training process.

6

Measurement Setup

6.1 Measurement location and route planning

The location of the 5G proprietary commercial-grade basestation operated by Ericsson for research purposes is on the roof of the Ericsson office at Mobilvägen 12, Lund. Its sector antenna faces approximately north, though the exact details of where it faces is unknown. The area is shown in Figure 6.1, with the approximate location of the basestation and its sector direction shown.

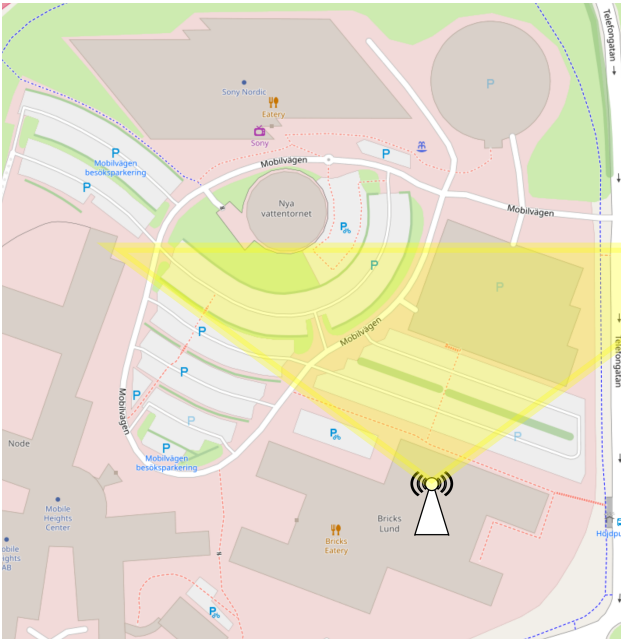


Figure 6.1 Geographic location of the Ericsson office in Lund on openstreetmaps, with basestation location and sector direction shown.

To decide a region within the geographic area to do measurements in, a few aspects have to be considered. First and foremost, as the training process is based on GPS data, GPS signal has to be relatively strong throughout the route to generate training data. To investigate the validity of the proposed ML approach, both LoS and NLoS scenarios should be investigated, with optimally minimum location difference of the two.

Additionally, two kinds of data is desired for the two types of potential positioning scenarios: one 'path' data for large-area positioning investigation on known pathways, and one 'dense-data', with the location for the dense-data having very high measurement density for a non-path-like region. This area data is to investigate the ability of the algorithm to handle arbitrary random-walks - the training data in this scenario will consist of a systematic dense walk covering the area, while the validation and training data will consist of a non-comprehensive 'directionless' walk in the covered region.

Of the places within the region, an obvious contender is the parking garage in the east, as shown by the pink circle in Figure 6.2. The roof of the parking garage has visual LoS on the basestation, even if it is not in the 'best-case' scenario, as it is outside the center of the beam. Furthermore, a direct comparison between LoS and NLoS scenarios can be made by moving around the edge of the roof for a LoS pathway and on the ground around the garage for an NLoS scenario. The distance between the two cases is fairly small, giving an ideal comparison dataset.

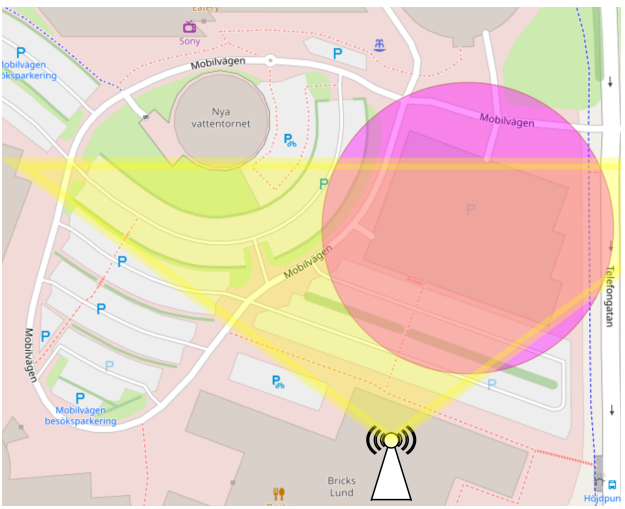


Figure 6.2 Selected measurement area - an open-topped parking garage - within the geographic area of the Ericsson office in Lund, as seen on openstreetmaps. The parking garage area allows for LoS and NLoS measurements in close proximity

In summary, a minimum of three datasets are desired: an area-covered dense-walk training set with a 'natural' random-walk validation and test dataset, a LoS path training set for positioning in a large area along a predictable path, and a NLoS path data nearby to compare LoS and NLoS effects. For the route direction two possibilities emerge: either looping around with only a single direction recorded, or turn around at a certain point to cover both to-and-from scenarios. The latter was chosen, in part to have a more general scenario for the ML model.

The plan for recording the chosen route and path-data can be seen in Figure 6.3, with both the rooftop LoS scenario in purple and the ground-level NLoS scenario in blue, each respectively henceforth referred to as LoS-A and NLoS-A.The approximate area to densely record data is then labeled with the vivid green geometry on the roof, and the datasets will be termed 'LoS-D'. Due to the time of data recording (July), the rooftop of the garage was mostly unpopulated by cars, lowering the number of obstructions. The dense area was specifically chosen as it had no obstructions to the freedom of motion, thereby enabling relatively even dense coverage.

Figure 6.3 The collection plans for the three datasets: the blue path is the approximate NLoS predictable path, the purple path on the top of the garage is the approximate LoS pathway, and the vivid green geometry shows the approximate region for the dense-walk dataset.

6.2 Obtaining GPS data

For a theoretical real-world system based on similar principles, the used training GPS position would be based on either commercial UE-based sensor-fused EMU and internal GPS antenna position outputs or using some more accurate positioning system. The latter scenario falls outside the domain of the thesis, however.

As a commercial UE, a commercial Android-based smartphone device with specifications given in Table 6.1 was used to record GPS data with the open-source android app 'gpstest' [Barbeau, 2022], which uses the Android GnssCapabilities API to obtain postion at 1 [Hz] sample-rate from the dual-frequency GNSS systems: GPS, GLONASS, QZSS, BeiDou, Galileo, NavIC, along with various satellite-based augmentation systems (SBAS).

Phone model:	OnePlus Nord 5G AC2003
Processor:	Qualcomm SM7250 Snapdragon 765G 5G (7 nm)
GNSS support:	Yes, Dual-band A-GPS, GLONASS, GALILEO, BDS, SBAS, NavIC
Operating System:	OxygenOS 11 (Android 11 variant)

Table 6.1 Smartphone and GNSS specifications used for GNSS recording

The 'gpstest' app outputs a data file containing raw satellite data, NMEA data, and position fix from the UE's internal GNSS module - utilizing UE IMU sensor fusion to increase accuracy. As position is the desired ground truth, not pure GPS, the position fix output is extracted from the text file by filtering for specific keywords.

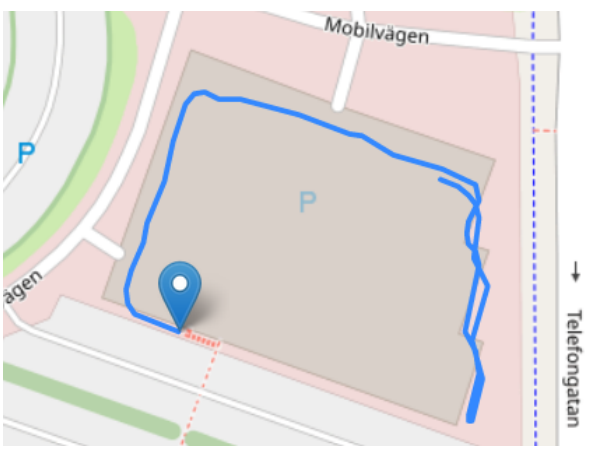


Figure 6.4 An example of extracted GPS data, showing a small section of the garage-top LoS path training dataset.

6.3 Logging SRS from a 5G Ericsson basestation

The outdoors commercial-grade 5G basestation set to the 100 MHz bandwidth configuration with the antenna panel on the roof of Ericsson was used with proprietary 5G-capable Android-based Ericsson testbed UEs. Using the CSI feedback loop structure from Section 2.10, the aim is to then extract the UE-specific SRS data (in the form of a channel matrix estimate) from the internal processes.

In the proprietary Ericsson baseband hardware, the internal beam-space representation of the channel matrix can be extracted either when it is updated or when it is used to transmit data. Though it would suit positioning better, obtaining the channel matrix whenever it is updated was unsuccessful due to software issues. As a fallback, the alternative logging tap is used for measurements in this thesis, wherein channel matrices are extracted when data is sent to the UE. Measurement stability and sample-size was encouraged by ensuring the UE had high data-rate requirements throughout the channel matrix measurements - specifically by continuously streaming 4k YouTube video streams.

The channel matrices for the specific proprietary UE and 5G testbed scenario support a 1/2/4-antenna UE and 32/64 basestation directional antennas (beams/directions). Furthermore, the 100 MHz configuration in 5G supports 273 PRBs, which in this specific case are themselves allocated in 2, 4, and 8 PRB 'blocks'. This means that there can be 35, 69, or 137 frequency channels. Together, this means that theoretically each full channel estimate $\mathbf{H}$ SRS data extraction consists of 35072 complex values, as shown in (6.1).

$$Max\left[\text{Capacity}\left(\mathbf{H}\right)\right] = Max\left[N_{Channels}N_{tx}N_{Direction}\right] = 137 \cdot 4 \cdot 64 \tag{6.1}$$

With 35072 complex values extracted potentially every few milliseconds, hardware bandwidth becomes a major concern. Logging proceeded by streaming the data over a network connection to a .log text file, with each SRS written with 8 directions per line, with large redundancies in e.g. UE identification and other metadata. This large redundancy vastly increases the network datarates required for log streaming. Furthermore, unused features add additional bandwidth overhead. Overall, due to the bandwidth limitation of logging, only three of the 137 potential frequency channels could be reliably retrieved, leaving 768 input features, or in other words 12 $H_{8\times 8}$ sub-channel matrices. The resulting single-measurement datapoint, with the dimensionality definition of $\mathbf{H}_{(K\cdot N_{TX})\times N_{\psi,h}\times N_{\psi,v}}$, is: $\mathbf{H}_{12,8,8}$.

To maximize the differentiability in the data, the three frequency bands chosen were the lowest, highest, and middle in the frequency band. The reasoning behind this is that nearby frequency bands behave similarly, with the inverse also holding.

The final logging aspect is that of time. In this case, two timestamps are logged: the frame number corresponding to the actual network time, and the UTC timestamp corresponding to the time a given line was written to the log. The delay between the two is small, with inaccuracies from GNSS leading to this delay having a negligible effect on the results from this thesis.

6.4 Channel matrix extraction from the SRS log

With the log obtained, the next step is that of data extraction, to retrieve the $\mathbf{H}_{12,8,8}$ matrix from the log file. The format of the log is such that at down to sub-millisecond intervals, a number of (from 1 to all 12) Channel/UE antenna pair sub-channel matrices are written into the file, with every 8 directions corresponding to one line. The directions themselves are stored as 4 hex-digits for both the real (RE) and the imaginary (IM) component, along with the per-line exponent to decode the Imaginary/real numbers.

Alongside the SRS data, a large number of other metadata and miscellaneous information is written per-line, which for the purposes of data extraction can be ignored. An additional important datapoint is the UE identifier enabling filtering the logging for the desired UE as during recording other measurement activities could occasionally also take place. A representation of the log file format can be seen in Figure 6.5.

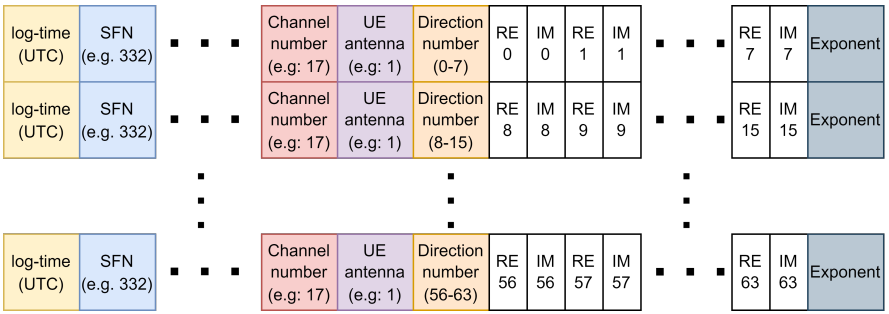


Figure 6.5 The format of a single logging instance, containing the SRS channel estimate for a single frequency channel and UE antenna, with timekeeping done with both UTC logging time and network System Frame Numbers (SFN). Not shown datapoints include the UE identifier, the sub-frame number, and more.

To understand the timing (frame numbers) aspect, a brief introduction is necessary. In 5G, the UE and the BS must maintain time-synchronization. For this, they have a ticking internal clock, with the two highest levels being the System Frame Number, which is stored as a 10-bit integer. Every tick on the SFN corresponds to 10 milliseconds of passed time. The sub-frame number goes from 0 to 9, and ticks every millisecond. From this, the maximum amount of time that can pass before the system clock resets is around 10.24 seconds. For data collection, this presents an issue: e.g. if the connection drops for over 20.48 seconds. In such a case, falling back on the UTC timestamp to see how many 10.24 second instances passed during the connection loss in the log gives an approximate solution to the number of SFN that passed, working well enough in practice.

To extract the logs into a usable data-format a C++ regex-based parser was developed as a python module for this thesis. The module takes every SFN with valid logging data, and combines all logging instances in that SFN into a sparse representation of the $\mathbf{H}_{12\times8\times8}$ channel matrix. If the same sub-channel matrix $H_{8\times8}$ is sampled multiple times in a single SFN, the newest one is chosen. The parser also ignores incomplete sub-channel matrices, e.g. if the logging only outputted 32 of the 64 directions. The process flow for extracting sparse channel matrices from the logging can be seen in Figure (6.6).

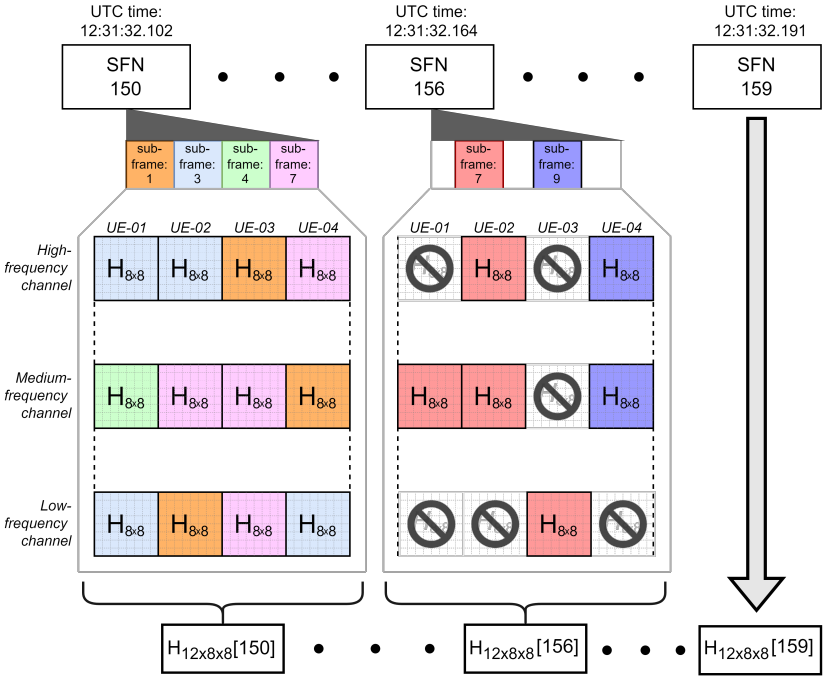


Figure 6.6 The process flow of extracting the (potentially sparse) channel matrices $\mathbf{H}_{12\times8\times8}[SFN] = \mathbf{H}_{12\times64}[SFN]$ for each valid SFN from the log file. Not shown is the UTC time and SFN time synchronization procedure.

The parser also unpacks the SFN numbers from a repeating sequence of 0 to 1023 into a timer counting upwards throughout the measurement. The series of channel matrices $\mathbf{H}_{12\times8\times8}[SFN] = \mathbf{H}_{12\times64}[SFN]$ is then stored in a 3D matrix $\mathbf{H}^{full}_{N^{data}\times12\times64}$, where N_{data} is the number of unique (unpacked) SFN numbers containing a new measurement of at least one sub-channel matrix.

For a list of measurements taken, see in the Appendix under Tables (11.1). Some associated properties examining data quality are also shown.

7

Data Preparation and analysis

7.1 Data preparation for regression

Using the time-series sparse channel matrix $H_{N_{data} \times 12}$ obtained through the measurement and log extraction process detailed in Chapter 6 as input for position regression requires further processing in the form of de-sparsifying and normalizing **H**, then interpolating GPS position onto the data using UTC timestamps.

De-sparsifying **H** is the first step in data pre-processing. The simplest temporally valid method for de-sparsifying data without known priors is forward-filling latest known values. Forward-filling for the channel matrix **H** is visualized in Figure 7.1.

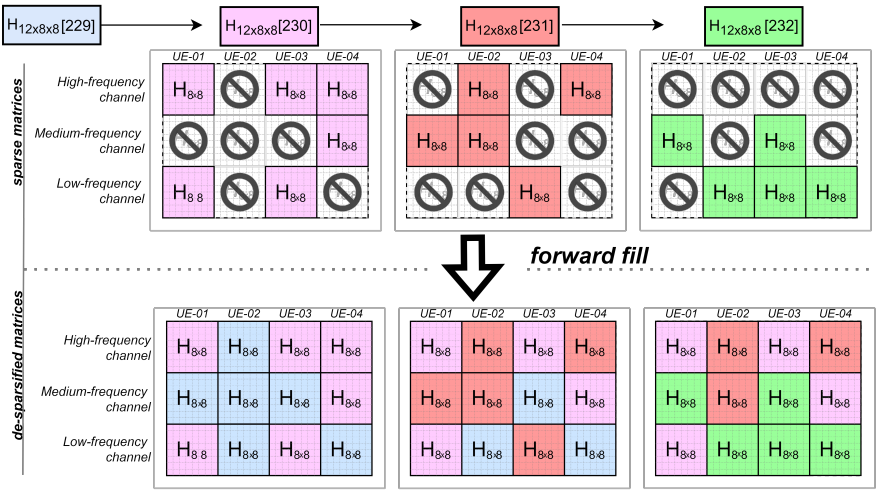


Figure 7.1 Using forward-filling on sparse channel matrices $\mathbf{H}_{12\times8\times8}[SFN]$.

The number of updated channel matrices are also recorded, and can be used for update weighting during particle filtering. Resampling to ensure only datapoints with a significant difference between channel matrices from one time-point to the next are kept is an optional step. However, considering the consensus on data quantity being a powerful regularization tool, resampling is best kept to a minimum.

The normalization step can be tuned with more sophisticated statistics, but in this thesis due to time constraints only linear scaling was utilized. Accordingly, the minimum value was scaled to $0+0i$, and the absolute of the maximum value of all datasets would be scaled to somewhere around 1. For this normalization, only training data was used to obtain the scaling factor, thereby preventing the contamination of the validation and test datasets with future information.

To assign positioning 'ground truths' to the channel matrix data $\mathbf{H}_{12}$, the UTC timestamp of both the Channel matrix data and the GPS is used. First, the two datasets are synchronized, then linear time-interpolation from the GNSS data is used to create interpolated trajectories, through which the 'ground truth' Y_{EN} coordinate pairs for each channel matrix are generated. Finally, all channel matrices that fall outside the bounds of the GPS measurement are discarded. The position interpolation process is shown in Figure 7.2.

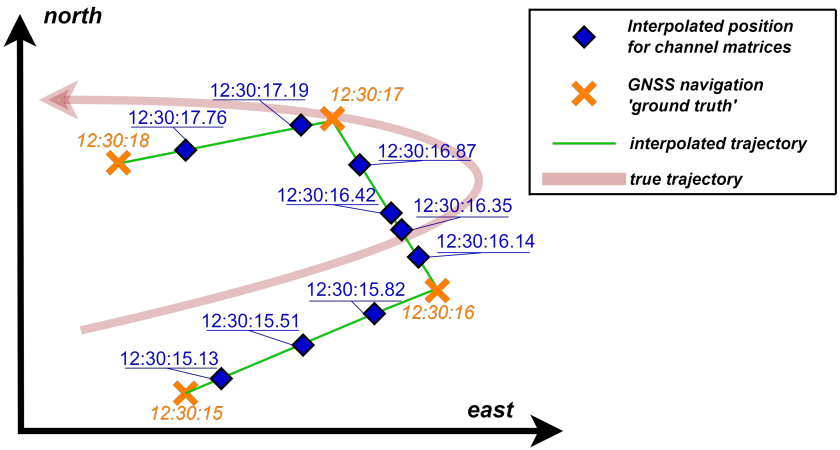


Figure 7.2 Assigning position to channel matrices $\mathbf{H}_{12\times8\times8}[SFN]$ using shared UTC timestamps with the GNSS dataset and simple linear interpolation. Also shown is the 'real' path the data was taken on, demonstrating that though GNSS might serve as an approximate for position, it is imperfect.

GNSS inaccuracy can be modelled during the training process by injecting Gaussian noise of similar magnitude as the GNSS measurement onto the training regressor Y_{train} every epoch during the training process. This also functions as output regularization as mentioned in Section 5.5.

7.2 Data analysis and discussion - data coverage

To examine the validity of data obtained through the measurement and processing pipeline, a brief analysis of the channel matrix data is pertinent. An often repeated phrase in the field of data-science is *"Garbage in, garbage out"*; Machine Learning without good data can not obtain good results.

Two aspects to the input data for ML are of particular significance: the coverage, and the quality. The data must have good-enough coverage of the area to allow an ML model to generalize, and the data must also contain enough information for an ML to find meaningfully patterns. Furthermore, redundant information should be reduced if possible to reduce the possibility of overfitting.

Large-scale data behavior

A proxy for data coverage w.r.t. positioning using SRS channel estimates is to examine the length of the time-delay between matrix updates and the number of layers updated on average. Using the dataset names as defined in Table 11.1, per-datasets results for the sample-number data analysis are shown in Table 7.1 below. Note that unfortunately as the Ericsson basestation underwent a system update that made collecting data problematic after 2022 August 18, separate test datasets for the LoS scenarios were not capturable - instead the LoS-Dv and the LoS-A2 were separated into validation and test parts.

Name	UE_{num}	N_{data}	updMSM	avgSD	stdSD	MaxSD [s]
NLoS-A1	4032	54996	6.539	32.303	251.90	14.180
NLoS-A2	6624	55305	6.003	32.306	281.00	11.21
NLoS-T_(a)	320	12169	5.386	26.663	152.14	3.40
NLoS-T_(b)	448	6887	5.662	27.494	175.64	3.46
NLoS-T_(c)	576	4449	5.702	27.290	177.91	3.30
LoS-A1	1216	48908	6.175	64.001	388.20	12.26
LoS-A2	5152	19634	5.937	84.211	426.31	4.76
LoS-Dtr	1280	29778	6.066	111.614	540.41	13.68
LoS-Dv	1344	19884	6.066	98.189	498.85	9.819

Table 7.1 Some derived quantities to check the quality of the data with regards to the sample rate, the number of samples, and the density of layers. 'updMSM' refers to the Mean sub-channel matrix updates per point, 'avgSD' refers to the Mean Sample Delay (in [ms]), 'stdSD' refers to the standard deviation of the sample delays (in [ms]), and finally 'MaxSD' refers to the Max Sample Delay (in [s]). The NLoS-T dataset lost connection two times, leaving three sub-datasets - _a, _b, _c - with long interrupted connection gaps.

Every 30-120 milliseconds, around 6 of the 12 discussed H_1 sub- channel matrices are updated. The update-rate seems to vary, with LoS scenarios having a notice-

ably longer delay between samples, even if the average of approximately 6 updates per sample holds. This disparity can be seen visually by comparing the sub-channel matrix update rates in Figure 7.3 for the LoS dense dataset and Figure 7.4 for the NLoSpath dataset. Visible in the figure are also the occasional gaps and potential connection losses, usually in the range of a dozen seconds or so. Another important empirical result of layer-update analysis is that despite the nonLoSA2 database consisting of repeated walks over the same path, there is no immediately visible repeating pattern in connection losses.

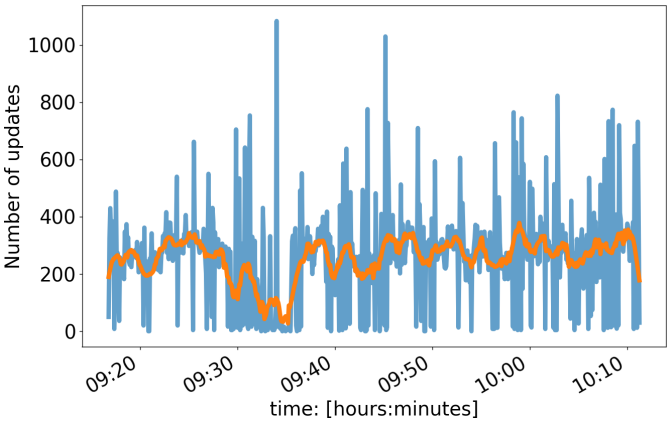


Figure 7.3 The sum total of independent sub-channel matrix $\mathbf{H}_{8\times8}$ updates over 5 second intervals for the LoS-Dtr dataset

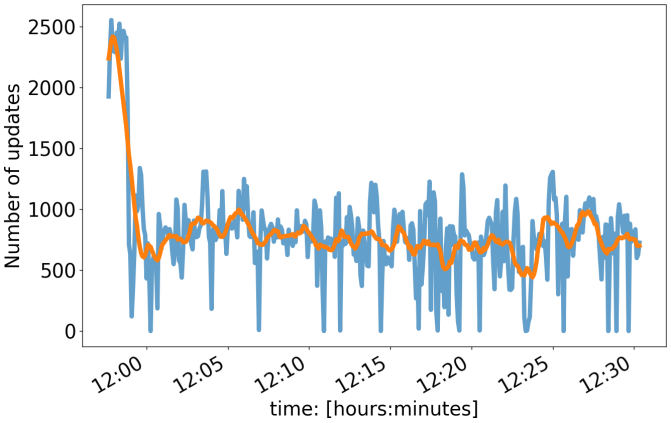


Figure 7.4 The sum total of independent sub-channel matrix $\mathbf{H}_{8\times8}$ updates over 5 second intervals for the NLoS-A2 dataset

With consistency generally holding in the overall update rate, the question remains if all individual sub-channel matrices H_1 are updated at a useful rate, and hopefully at an even relative rate. Extending Figures 7.3, 7.4 into three dimensions, the sampling behaviour for every UE RX antenna at every frequency band can be examined: as seen in Figure 7.5 for the LoS-Dtr dataset and Figure 7.6 for the NLoS-A2 dataset.

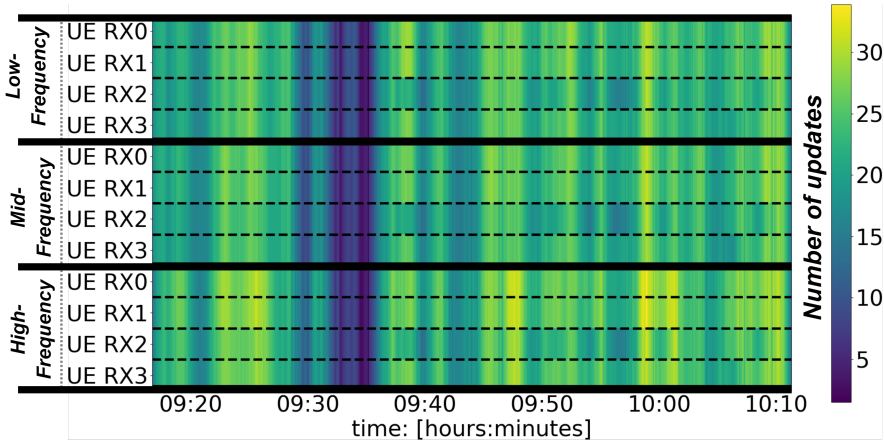


Figure 7.5 The sub- channel matrix $\mathbf{H}_{8\times8}$ updates over 5 second intervals for each UE RX antenna in three frequency bands in the LoS-Dtr dataset

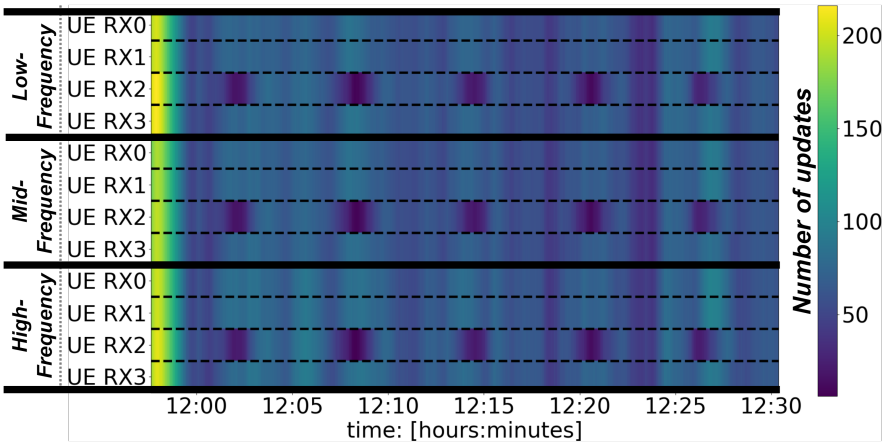


Figure 7.6 The sub- channel matrix $\mathbf{H}_{8\times8}$ updates over 5 second intervals for each UE RX antenna in three Frequency bands in the NLoS-A2 dataset

From the above, it is immediately visible that the individual sub- channel matrices seem to be updated in groups, but roughly at similar update rates throughout, though with the occasional dip visible in e.g. UE antenna RX2 in the NLoS-A2 dataset from Figure 7.6. Overall, the examined data demonstrate that the domain coverage of the examined datasets is both even across time and across the sub-channel matrices - properties needed to confirm data validity for the utilization of ML.

Furthermore, the examined datasets seem to confirm the expectation that worse transmission conditions generally result in a higher frequency of channel estimate updates. The underlying assumption being that maintaining a poorer quality channel at identical datarates requires more channel overhead. Channel quality in the NLoS scenario also seems to be relatively consistent, even when datasets were taken days apart.

Extrapolating from the previous statement on the relation between channel quality and the mean sample delay, it appears that the LoS Dense-grid data was collected under better channel conditions than the LoS path-data. This holds with the observation during measurement that the video stream during the recording of the LoS-A1 dataset ran into datarate limitations. No problems with the video stream originating from datarate limitations was apparent while recording the LoS-Dtr or LoS-Dv datasets. Furthermore, the LoS-A2 dataset seems to have mildly different channel characteristic, albeit with a still relatively high mean sample delay.

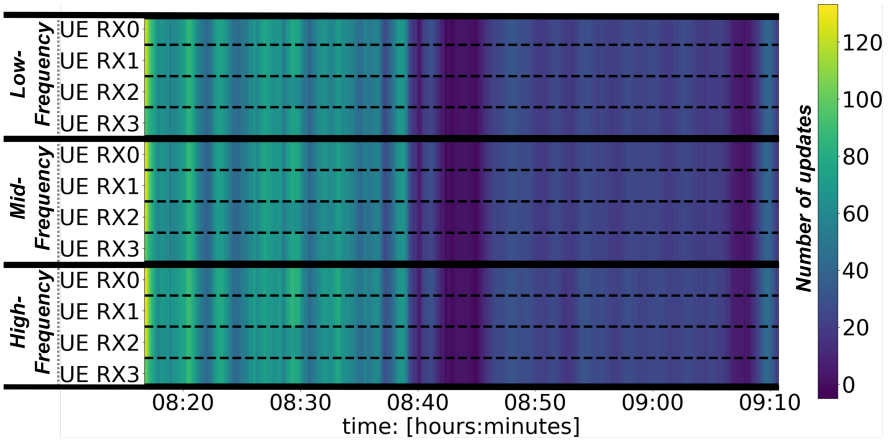


Figure 7.7 The sub- channel matrix $\mathbf{H}_{8\times8}$ updates over 5 second intervals for every Frequency band and UE RX antenna pair in the LoS-A1 dataset

Investigating the discrepancy in the LoS-A1 dataset further, Figure 7.7 shows the per-layer number of updates every 5 seconds having two stages: one pseudo-NLoS stage lasting until 8:40, and one LoS phase, more similar in form to the dense

dataset seen in the LoS-Dtr dataset. The reason behind the channel quality discrepancies for the different sub-sections of the LoS scenarios is unclear. One cause that can be probably be ruled-out is that of differing environmental conditions. When recording the 'LoS-A1' path dataset, nothing visibly shifted in the environment from the first half to the second half. The switch from the low channel quality (and long video buffer times) to good channel quality and LoS behavior happened at an arbitrary point.

Further testing is required to pinpoint behavioral specifics behind the lack of consistency in the around LoS scenario. Potential causes can include e.g. interference by another unnoticed ongoing experiment, connection to a secondary (supposedly disabled) antenna array, more cars around the edge of the garage roof leading to more NLoS sub-scenarios, or just some other unknown phenomena. Due to data being recorded on Ericsson testbenches, certain issues would require a deep dive into protected behavior or using in-house diagnostics.

The last anomaly in the obtained data w.r.t. consistency is that of the NLoS-T dataset, more specifically the connection loss. The reasons behind it are unknown, but the sample refresh rate of the resulting dataset can be seen in Figure 7.8. However, since this dataset is a test dataset, there is no worry with the resulting gap in coverage for the training and validation process. Instead, the lost connections will prove as an excellent test for the performance of the tracking algorithm with regards to more realistic conditions.

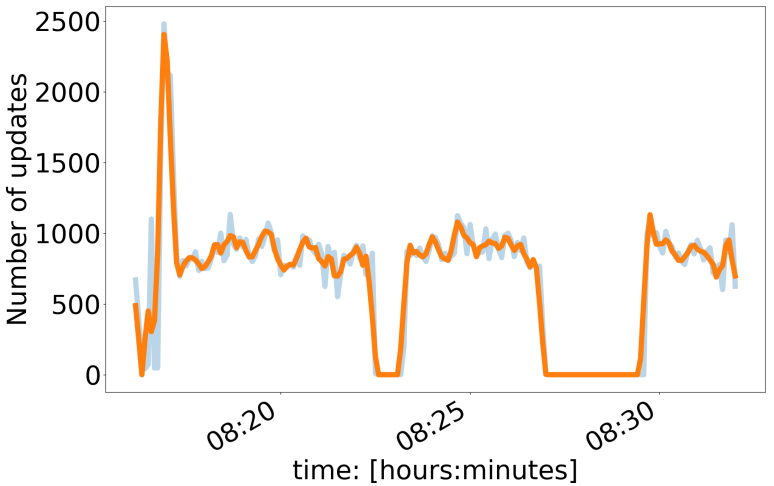


Figure 7.8 The sum total of independent sub-channel matrix $\mathbf{H}_{8\times8}$ updates over 5 second intervals for the NLoS-T dataset, showing the two connection losses. Despite the long connection gaps, the connected sections appear to have similar connection patterns.

Small-scale data behavior

With data regularity demonstrated on the scale of minutes to hours, it is also worth looking at the small-scale regularity of data. While large-scale coverage of the measurement domain is a necessary condition for ML to generalize well, fluctuations or irregularities in sample-rates on the smaller scales may be an aspect for particle filtering.

As a first pass, using the per- sub-channel matrix visualizations from before but zoomed-in on a sub-second scale shows a sort of uneven 'clustering' in both the NLoSand the LoS scenarios, as seen in Figure 7.9 for the NLoS-A2 dataset and Figure 7.10 for the LoS-Dtr dataset. The data 'clusters' arrive at semi-regular intervals, though with visible variance, and with the occasional sample or two arriving between the clusters.

The clusters occur every few seconds - and at around 1 m/s pedestrian walking speed, this would lead to up to a few meter 'gaps' in the regressed position data. With a measured GNSS positioning accuracy at around 3 meters, the data 'clustering' phenomenon could be a minor but not dominant source of inaccuracy in the final ML process.

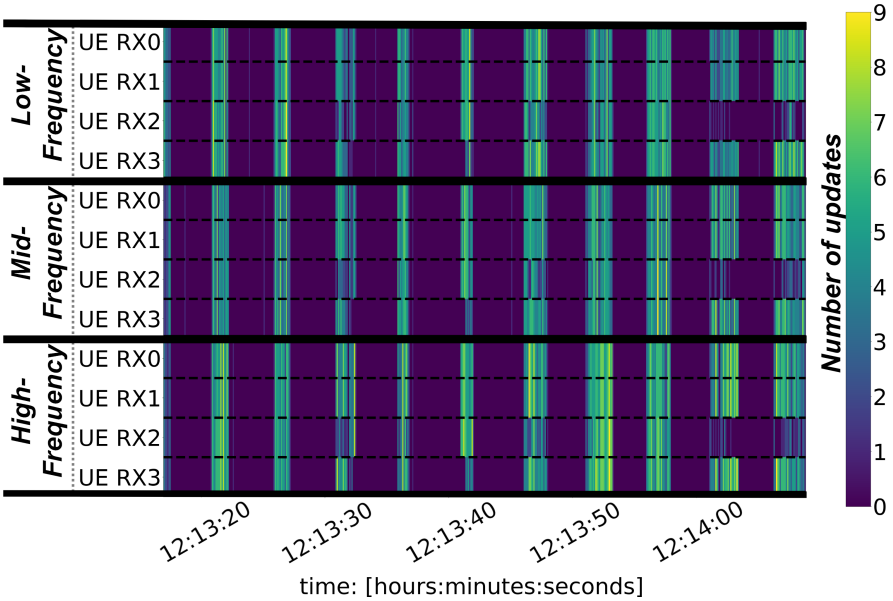


Figure 7.9 The sub- channel matrix $\mathbf{H}_{8\times8}$ updates over 100 millisecond intervals for every Frequency band and UE tx antenna pair in the NLoS-A2 dataset, showing 'clustering' behavior. The NLoSscenario appears to lead to larger data 'clusters' as compared to the LoS scenario.

The 'size' of the clusters appears to be the source of the difference between the LoS and NLoSscenarios with regards to mean sample delay. The LoS scenario as seen in Figure 7.10 shows far smaller number of samples per 'cluster' than the NLoSscenario as shown in Figure 7.9.

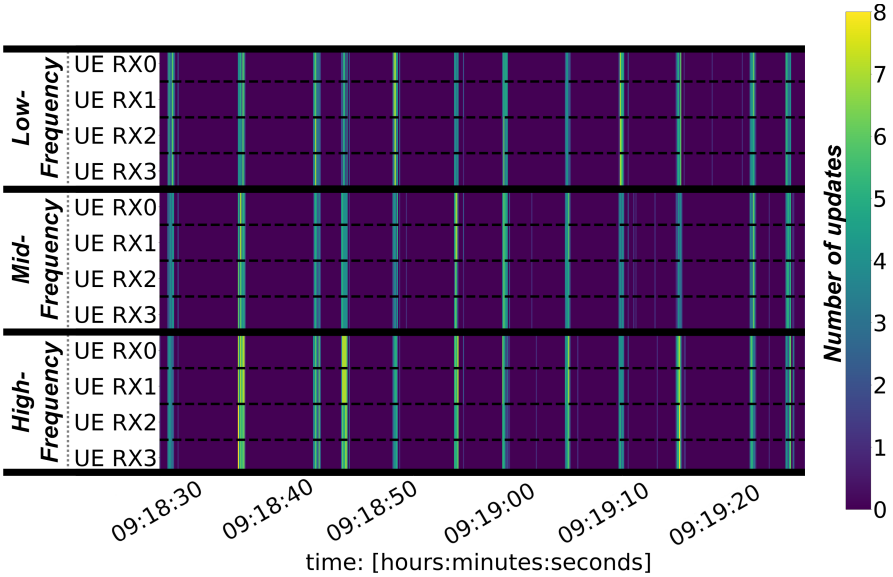


Figure 7.10 The sub- channel matrix $\mathbf{H}_{8\times8}$ updates over 100 millisecond intervals for every Frequency band and UE tx antenna pair in the LoS-Dtr dataset, showing 'clustering' behavior. The LoS scenario appears to lead to smaller data 'clusters' as compared to the NLoSscenario.

To systematically investigate the sample delay behavior of an entire dataset, binning all the datasets into a histogram visualization is possible. The result for the LoS-Dtr dataset can be seen in Figure 7.11, while the result for the NLoS-A2 dataset can be seen in Figure 7.12.

The results correspond to what is expected - after an initial spike of around 100× incidence rate, the sample-delay density tapers off such that with a log-increase in bin-size, the number of delays falling into the bin stays constant. The initial spike then corresponds approximately to the 'clusters' of data, while the slow logarithmic fall of sample delay density comes from both the variability in 'cluster' frequency and from the stochastic lone sample or two between the clusters.

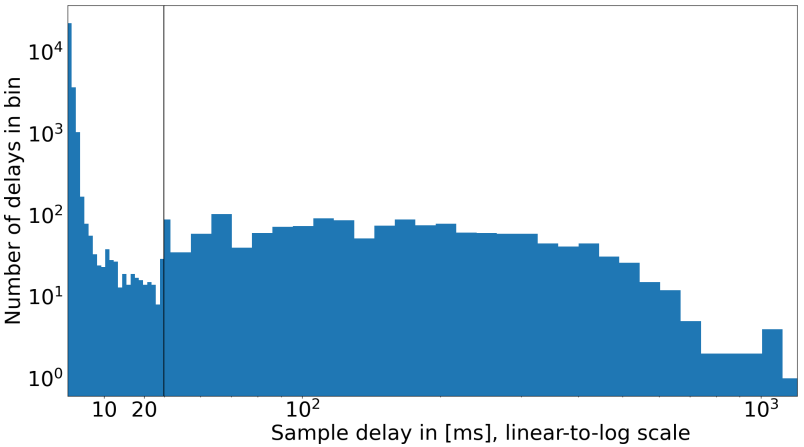


Figure 7.11 Linear-start log-scale histogram of the sample delays in the LoS-Dtr dataset. The number of delays of 1, 2, ..., 25 [ms] are shown in the linear binning section, with log-incidence rate on the Y-axis. Above 25 [ms], the bins are log-scale.

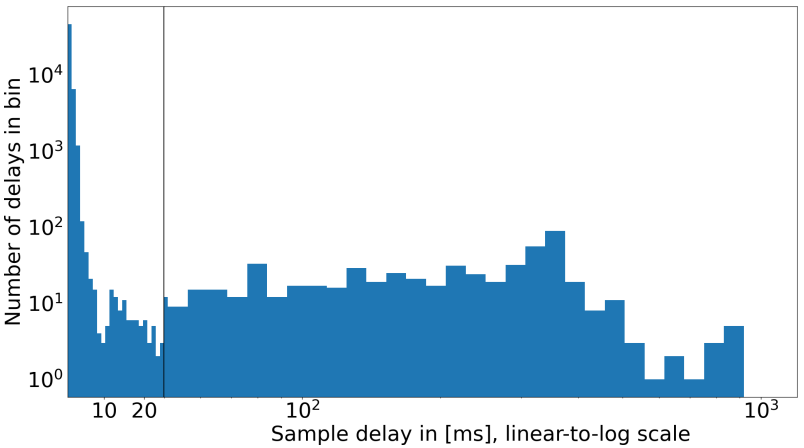


Figure 7.12 Linear-start log-scale histogram of the sample delays in the NLoS-A2 dataset. The number of delays of 1, 2, ..., 25 [ms] are shown in the linear binning section, with log-incidence rate on the Y-axis. Above 25 [ms], the bins are log-scale.

7.3 Data analysis and discussion - information content

The obtained data consists of complex-valued $\mathbf{H}_{12\times8\times8}$ matrices, with Section 7.2 demonstrating relatively good data coverage where at constant data-rates the sample-delay is proportional to the channel quality. In this section, the information content of the data is confirmed, and basic feature selection is investigated.

With regards to feature selection, there are three distinct 'feature' classes easily identifiable: Frequency channels, UE RX antennas, and the phase/amplitude of the complex numbers. Of these three features classes, the easiest to discard for this thesis is the phase of the complex numbers.

Phase data

The suspicion from Section 5.4 using the material covered in Chapter 2 is that from the complex matrices per-element phase is not very useful. To give a visual example of phase containing little valuable information, Figure 7.13 shows the relative uselessness of phase w.r.t. the positioning task using GNSS positional ground-truth.

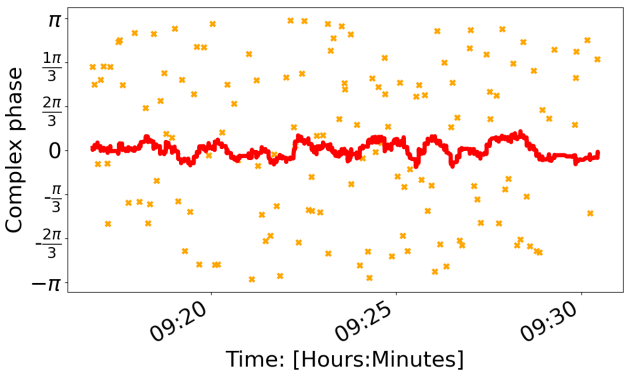


Figure 7.13 A snapshot of the phase in radians of a single direction $h_{[8,6]}$ in a sub-channel matrix H_1 of the LoS-Dtr database. The phase rapidly and stochastically oscillates between -π and π, with GNSS inaccuracy around 3.5 meters leaving phase useless for positioning. The red curve shows the moving average of complex phase over 30 samples.

The explanation for the lack of useful information in phase requires an exploration of the underlying physics. Phase generally contains information on three aspects: the angle of arrival to the RX, the angle of departure from the TX, and movement under a wavelength, especially for LoS scenarios. Of the three above aspects, the angle of departure from the basestation is already extracted from the phase information by the built-in beam-domain transformation within the basestation, before it reaches the ML model. The angle of arrival to the UE contains information about

the orientation of the UE, but in practice for most positioning setups the orientation of the UE is uncorrelated with the direction of movement and position of the UE.

Determining the orientation of the phone through phase might become useful for future applications such as for when the UE is fixed to a vehicle - but for pedestrian tracking, which is the dataset created for this thesis, it is useless. Furthermore, small-scale movement and sub-wavelength position is too noisy to reliably utilize, especially since the positioning accuracy of the ground-truth GNSS is an order of magnitude greater than the utilized 5G radio wavelengths are.

Amplitude data

The amplitude of outputs changes far more slowly than the phase, with longer trends containing information on large-scale fading rather than on small-scale fading. The assumption is that even in NLoScases, the amplitude of a beam depends on environmental geometry enabling MPCs to reach the UE, thereby acting as a slowly-varying correlate to position. To give an example of this, see Figure 7.14 for the complex amplitude with the phase data shown in Figure 7.13

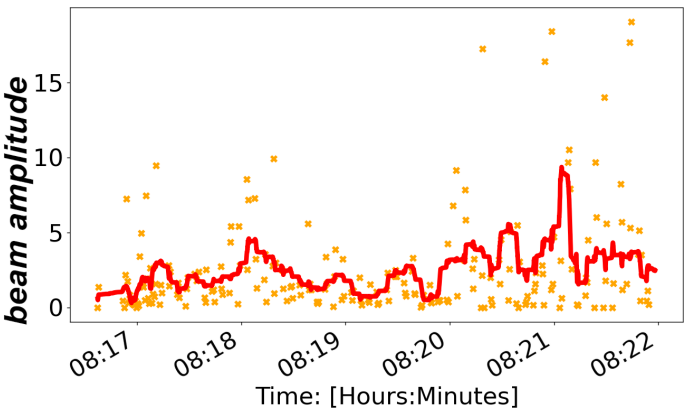


Figure 7.14 A snapshot of the unitless complex amplitude output of a single direction $h_{[8,4]}$ in a sub-channel matrix H_1 of the NLoS-T database. The amplitude varies relatively slowly, potentially correlating with position. The red curve shows the moving average of the amplitude over 30 samples.

To confirm that the position correlates with beam amplitude, analyzing the path datasets is pertinent. As there is a periodicity of position as the path is walked back-and-forth on, it is expected the output complex amplitude will show a similar behavior. On the NLoS-A1 dataset for example, there are 5 back-and-forth cycles on the NLoS path seen in Figure 6.3. The expectation is then that certain outputs will have very visible periodicity to their energies, e.g. as confirmed in Figure 7.15.

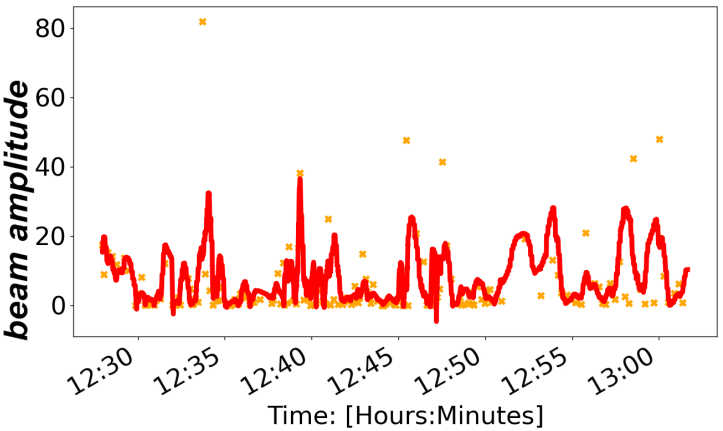


Figure 7.15 A snapshot of the unitless complex amplitude output of a single direction $h_{[8,5]}$ in a sub-channel matrix H_1 of the NLoS-A1 database. The red curve shows a moving average value over 100 samples. The amplitude varies over 5 periods with the periodicity expected from the path dataset.

The periodicity becomes more clear when the mean of the amplitude for all the outputs in a single sub-channel matrix are taken, as shown in Figure 7.16.

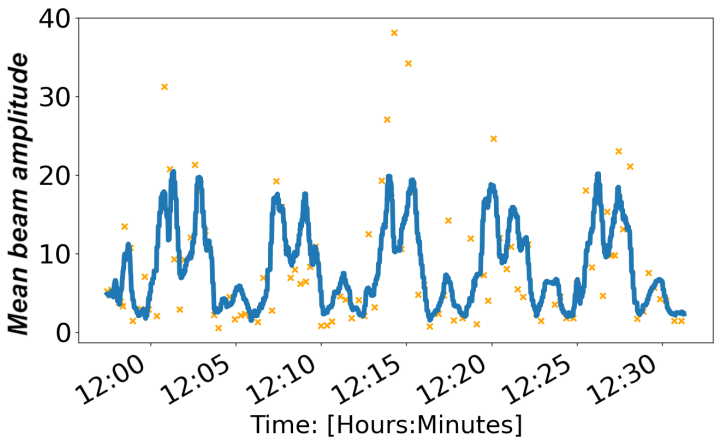


Figure 7.16 The mean unitless complex amplitude of all the outputs in the low-frequency UE RX 0 sub-channel matrix H_1 of the NLoS-A2 database. The amplitude of the outputs vary over 5 periods with the periodicity expected from the path dataset. The blue curve shows the moving average over 100 samples.

In summary, despite prior work on simulated data showing improvements in positioning accuracy from utilizing complex values (See Section 1.2), the benefits of phase diminish on meter-scales after angles of departure are taken into account. Therefore, for the purposes of this thesis complex phase is disregarded and only amplitude is used.

UE RX antenna and frequency-band information content

The next two collections of features are the different sub-channel matrices on the different frequency bands and the different UE RX antennas. For an initial examination of the utility of both UE layers and different frequency bands, Figures (7.17,7.18) show the mean amplitude across all directions in each sub- channel matrix $\mathbf{H}_1$ in the LoS-Dtr dataset and NLoS-A2 datasets respectively.

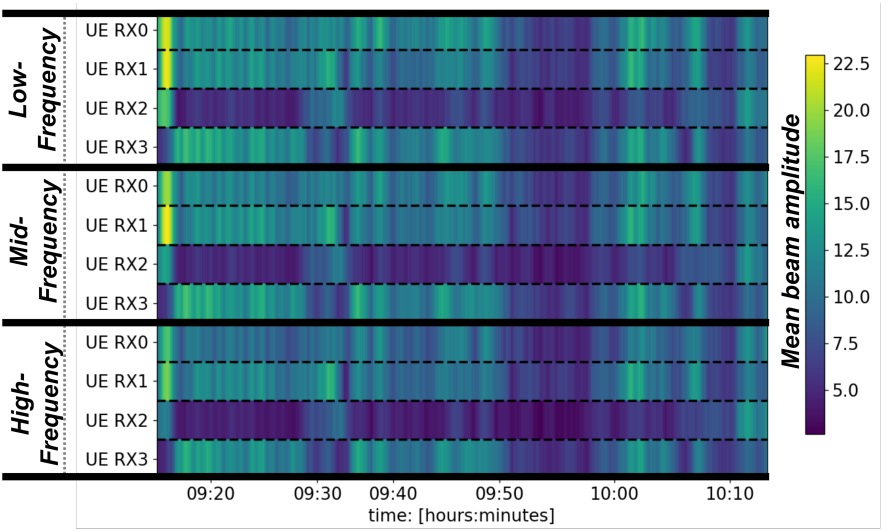


Figure 7.17 The mean complex amplitudes for each sub- channel matrix $\mathbf{H}_{8\times8}$ in the LoS-Dtr dataset

W.r.t UE RX antennas, as direction of arrival is ignored it is not clear if there is a significant benefit in terms of MPC information through taking dependency on UE RX antennas into account. However, as the exact functions of the systems at the Ericsson basestation are not fully known, there is the possibility of some form of positional correlate from e.g. antenna prioritization or orthogonal multiplexing. For this reason, an empirical analysis of the information content of the UE RX antennas is pertinent. If no significant information is gained by handling the different UE antenna layers separately, then instead the multiple antenna layers might instead be utilizable for noise suppression.

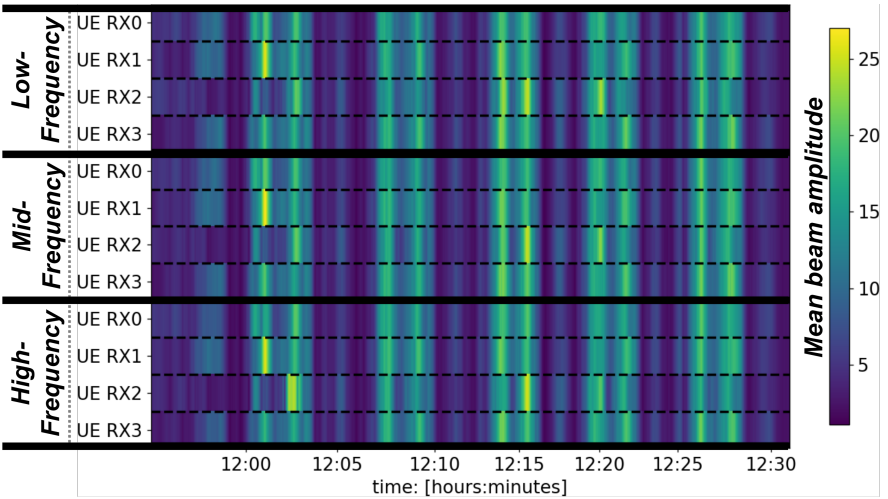


Figure 7.18 The mean complex amplitudes for each sub- channel matrix $\mathbf{H}_{8\times 8}$ in the NLoS-A2 dataset

For frequency bands on the other hand, the intuition as per the physics described in Chapter 2 is that the different frequency bands show slightly different MPC behaviors. The expectation is then that for positioning scenarios substantially benefiting from MPCs, increased frequency resolution could lead to better results, assuming the frequency bands are far enough apart to see significant difference in MPC behavior. Should the frequency bands be too close, then MPC behavior can become negligible and a higher resolution in the frequency-band can only be used for noise suppression in data pre-processing.

In practice, using Figures (7.17,7.18) it can be seen that both UE antenna diversity and different frequency bands appear to give differing behaviors w.r.t. mean complex amplitudes. Despite the visible differences however, what is more striking is the high level of correlation visible across all sub-channel matrix mean amplitudes. Therefore, while the utility of using the different sub- channel matrices $\mathbf{H}_1$ can not be ruled out, neither can the consideration that the minor increase in informational quantity from keeping the UE RX antenna and/or the frequency band features is not proportional to the potential for overfitting that introducing 3-12 times the data features may bring. The consequence of this is that different combinations of ML input features will be investigated, to empirically examine the consequences of vastly increasing data dimensionality to gain non-substantial amounts of data.

The investigated input types will then be: the full channel matrix $\mathbf{H}_{full}$, UE-averaged channel matrix $\mathbf{H}_3^{mUE}$, frequency-averaged channel matrix $\mathbf{H}_4^{mFr}$ and finally the fully-averaged channel matrix $\mathbf{H}_1^{FA}$.

7.4 Data pre-processing

Before applying ML models onto the data, it is worth considering what - if any - preprocessing steps could improve results. Looking at the form of the data extracted, in theory being the beam amplitude transfer function but for this basestation probe appearing to be beam energy, gives a hint towards a simple processing change. As mentioned in Section 2.6, wireless transmission energy initially falls at distance squared, and at a certain distance, due to scattering and MPC behavior, energy falls by a power of four. Therefore, there are two relevant transforms: taking one or more nth-roots of the input, from square-root to 6th-root of the input channel matrix data **H**. For this thesis, the square- and fourth- root are tested to see if they improve UE positioning accuracy by hopefully making distance more linearly correlated with the input data.

To further improve upon linearity, the data should mostly remain linear when scaling it before input. The easiest method for this is to just divide it with a scalar number, and keep the minimum at 0. Optionally, outlier detection could be used to reduce the effect of high individual input complex amplitudes on the linear scaling, thereby making it semi-linear but without outliers. Such outlier detection requires further characterization of the input data however.

Another discussed aspect in Section 5.6 are polynomial kernel methods, under the umbrella of polynomial regression. In it, the input feature vector is taken and turned into an n-factor polynomial using polynomial coefficients. E.g, a feature vector $\hat{x}_{1\times 3}^{lin}$ will, using a 2-factor polynomial transformation, be turned into the feature vector $\hat{x}_{1\times 9}^{pol-2}$ as per (7.1).

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \\ \downarrow \\ \begin{bmatrix} 1 & x_1 & x_2 & x_3 & x_1^2 & x_2^2 & x_3^2 & x_1x_2 & x_1x_3 & x_2x_3 \end{bmatrix} \tag{7.1}$$

Due to memory and processing-time constraints, only the fully-averaged channel matrix will have polynomial features. The polynomial transform used is of factor 2, as feature counts explode past any reasonable point of computation for anything above that. To give an example, a feature size of 64 results in 2049 features with polynomial regression of factor 2, and increasing to factor 3 will result in 131073 features; completely unfeasible to compute. Even if only the per-sub- channel- matrix interaction features are generated, taking all 12 layers would lead to over 24000 features. Finally, intuition suggests minimal potential gains and a greater degree of overfitting considering the sheer number of features when applying a polynomial transform greater than factor two.

For splitting the LoS-A2 and the LoS-Dtr validation datasets into validation and test sets respectively, the first approximate 12000 datapoints of the datasets were designated the validation data, with the rest designated as the test dataset.

7.5 Regression using non-deep-learning ML

In this chapter so far both sufficient coverage and informational content of the input data has been confirmed to a reasonable extent. It is then worth examining the intuition that a highly-complex ML model such as a Deep ANN is required for useful positioning, instead of something along the line of Elastic-Net regression - see Section 5.6. As a comparison, two 'naive' positioning approaches are also included: random-guess, where position is guessed randomly somewhere in the measurement area, and mean-guess, where position is always guessed at the mean position of all the samples. The 'results' of these are shown in Table 7.2.

Dataset	Input type	Model type	tr. MSE	val. MSE	test MSE
LoS-D	N/A	Mean-guess	67.2	50.7	53.8
LoS-A	N/A	Mean-guess	518.1	570.3	560.4
NLoS-A	N/A	Mean-guess	724.8	770.5	713.7
LoS-D	N/A	Random-guess	185.8	168.2	172.3
LoS-A	N/A	Random-guess	971.8	1005.5	1021.4
NLoS-A	N/A	Random-guess	1403.4	1453.5	1407.4

Table 7.2 Baseline scores; 'mean guess': points are at the mean training position. 'random-guess': points are scattered in the area spanned by the training data.

As the non-deep-learning ML models, Linear Regression (Lin.Regr.), Linear Regression with Polynomial Features (Lin.Regr. P.), Elastic-Net regression (EN.Regr.), Elastic-Net regression with polynomial features (EN.Regr. P.) are applied to the LoS-Dense, the LoS-path, and the NLoS-path scenarios. The different Elastic-Net regression models have their hyperparameters tuned through 3-fold Cross-Validation on the training data, then the different model types are compared using the validation dataset. The different combinations of averaged sub-channel matrices as detailed in Section 7.3 are also examined using the various different non-deep-learning methods.

The expectation from the underlying physics (See section 2.4) is that linear regression should be able to make a reasonable prediction for the LoS scenario, as it should present a dominant specular component w.r.t. direction, where in shorter ranges beam energy depends on distance. However, the NLoSscenario should not perform nearly as well through linear regression.

Furthermore, it is expected that taking the square root of the input dataset for LoS scenarios and combine it with the 4-th root of the input dataset for NLoSscenarios should improve results for well-behaved datasets. For a demonstration of the importance of taking the square root of the input data, a selection of models and a LoS and NLoSdataset are regressed for using without taking the square root first, with results shown in Table 7.3. For comparison, similar results on the full square-rooted channel matrix are shown in Table 7.4.

Dataset	Input type	Model type	tr. MSE	val. MSE	test MSE
LoS-D	$\mathbf{H}_{full}$	Lin.Regr.	24.0	26.7	27.4
NLoS-A	$\mathbf{H}_{full}$	Lin.Regr.	337.4	461.0	366.9
LoS-D	$\mathbf{H}_{full}$	EN.Regr.	25.5	23.3	25.3
NLoS-A	$\mathbf{H}_{full}$	EN.Regr.	394.8	394.2	385.3

Table 7.3 Scores for basic ML algorithms using the Full channel matrix without taking the square root first.

Dataset	Input type	Model type	tr. MSE	val. MSE	test MSE
LoS-D	$\sqrt{\mathbf{H}}_{full}$	Lin.Regr.	17.9	19.8	20.3
LoS-A	$\sqrt{\mathbf{H}}_{full}$	Lin.Regr.	113.9	252.6	245.8
NLoS-A	$\sqrt{\mathbf{H}}_{full}$	Lin.Regr.	244.4	336.3	261.2
LoS-D	$\sqrt[4]{\mathbf{H}}_{full} \cup \sqrt{\mathbf{H}}_{full}$	Lin.Regr.	14.0	19.3	27.4
LoS-A	$\sqrt[4]{\mathbf{H}}_{full} \cup \sqrt{\mathbf{H}}_{full}$	Lin.Regr.	74.4	281.3	233.1
NLoS-A	$\sqrt[4]{\mathbf{H}}_{full} \cup \sqrt{\mathbf{H}}_{full}$	Lin.Regr.	181.9	293.3	223.2
LoS-D	$\sqrt{\mathbf{H}}_{full}$	EN.Regr.	18.9	17.4	18.9
LoS-A	$\sqrt{\mathbf{H}}_{full}$	EN.Regr.	124.9	246.5	237.0
NLoS-A	$\sqrt{\mathbf{H}}_{full}$	EN.Regr.	289.5	286.1	263.6
LoS-D	$\sqrt[4]{\mathbf{H}}_{full} \cup \sqrt{\mathbf{H}}_{full}$	EN.Regr.	15.7	17.9	25.4
LoS-A	$\sqrt[4]{\mathbf{H}}_{full} \cup \sqrt{\mathbf{H}}_{full}$	EN.Regr.	96.6	241.4	225.4
NLoS-A	$\sqrt[4]{\mathbf{H}}_{full} \cup \sqrt{\mathbf{H}}_{full}$	EN.Regr.	218.1	279.4	221.5

Table 7.4 Scores for basic ML algorithms using the Full channel matrix

From just the first results on using regression with the full channel matrix, it is immediately obvious that the LoS-D is well-behaved, with significantly improved results over random guessing. Unexpectedly, even NLoSand the misbehaving LoS-A datasets gave better-than-random results, though still vastly underperforming the proper LoS scenario.

Furthermore, taking the square root input greatly improves results on all scenarios: LoS-D, LoS-A and NLoS. Furthermore, also shown from these early results is that the 4-th root does not improve results for the well-behaved LoS scenario, but does consistently improve results for the NLoSscenario.

The misbehaving LoS-A datasets also appear to be more 'NLoS' in character than 'LoS', further reinforcing the intuition from before. The result is, however, that the original stated goal of using the LoS-A dataset to have a direct comparison to the NLoS-A dataset is not possible. Due to time and access restrictions, re-recording the NLoS-A data is not possible. Instead, the dataset will be viewed as a performance test in the case of misbehaving data.

In the next Tables (7.5 - 7.6), the combinations of different dataset preprocessing methods are examined.

Dataset	Input type	Model type	tr. MSE	val. MSE	test MSE
LoS-D	$\sqrt{\mathbf{H}_3^{mUE}}$	Lin.Regr.	21.3	19.3	19.3
LoS-A	$\sqrt{\mathbf{H}_3^{mUE}}$	Lin.Regr.	131.1	232.3	205.3
NLoS-A	$\sqrt{\mathbf{H}_3^{mUE}}$	Lin.Regr.	265.8	299.6	260.4
LoS-D	$\sqrt[4]{\mathbf{H}_3^{mUE}} \cup \sqrt{\mathbf{H}_3^{mUE}}$	Lin.Regr.	17.4	17.4	27.4
LoS-A	$\sqrt[4]{\mathbf{H}_3^{mUE}} \cup \sqrt{\mathbf{H}_3^{mUE}}$	Lin.Regr.	103.1	227.8	186.1
NLoS-A	$\sqrt[4]{\mathbf{H}_3^{mUE}} \cup \sqrt{\mathbf{H}_3^{mUE}}$	Lin.Regr.	254.2	254.2	220.1
LoS-D	$\sqrt{\mathbf{H}_3^{mUE}}$	EN.Regr.	21.5	18.3	19.0
LoS-A	$\sqrt{\mathbf{H}_3^{mUE}}$	EN.Regr.	133.9	228.9	198.7
NLoS-A	$\sqrt{\mathbf{H}_3^{mUE}}$	EN.Regr.	294.7	277.8	264.5

Table 7.5 Scores for basic ML algorithms using the UE-averaged channel matrix

Dataset	Input type	Model type	tr. MSE	val. MSE	test MSE
LoS-D	$\sqrt{\mathbf{H}_1^{FA}}$	Lin.Regr.	23.3	20.6	20.6
LoS-A	$\sqrt{\mathbf{H}_1^{FA}}$	Lin.Regr.	134.6	226.5	195.4
NLoS-A	$\sqrt{\mathbf{H}_1^{FA}}$	Lin.Regr.	271.7	287.8	262.7
LoS-D	$\sqrt[4]{\mathbf{H}_1^{FA}} \cup \sqrt{\mathbf{H}_1^{FA}}$	Lin.Regr.	19.5	18.4	27.4
LoS-A	$\sqrt[4]{\mathbf{H}_1^{FA}} \cup \sqrt{\mathbf{H}_1^{FA}}$	Lin.Regr.	108.0	217.2	174.28
NLoS-A	$\sqrt[4]{\mathbf{H}_1^{FA}} \cup \sqrt{\mathbf{H}_1^{FA}}$	Lin.Regr.	206.7	237.2	220.4
LoS-D	$\sqrt{\mathbf{H}_1^{FA}}$	Lin.Regr. P	8.6	19.5	16.8
LoS-A	$\sqrt{\mathbf{H}_1^{FA}}$	Lin.Regr. P	34.5	218.7	141.3
NLoS-A	$\sqrt{\mathbf{H}_1^{FA}}$	Lin.Regr. P	112.5	332.1	193.7
LoS-D	$\sqrt{\mathbf{H}_1^{FA}}$	EN.Regr.	26.4	18.8	21.7
LoS-A	$\sqrt{\mathbf{H}_1^{FA}}$	EN.Regr.	135.6	224.9	191.7
NLoS-A	$\sqrt{\mathbf{H}_1^{FA}}$	EN.Regr.	291.9	272.5	267.2
LoS-D	$\sqrt{\mathbf{H}_1^{FA}}$	EN.Regr. P	25.3	18.6	20.6
LoS-A	$\sqrt{\mathbf{H}_1^{FA}}$	EN.Regr. P	97.9	211.4	163.2
NLoS-A	$\sqrt{\mathbf{H}_1^{FA}}$	EN.Regr. P	321.3	308.6	276.1

Table 7.6 Scores for basic ML algorithms using the fully-averaged channel matrix

Dataset	Input type	Model type	tr. MSE	val. MSE	test MSE
LoS-D	$\sqrt{\mathbf{H}_4^{mFr}}$	Lin.Regr.	20.43	19.97	20.25
LoS-A	$\sqrt{\mathbf{H}_4^{mFr}}$	Lin.Regr.	126.85	237.53	236.1
NLoS-A	$\sqrt{\mathbf{H}_4^{mFr}}$	Lin.Regr.	308.94	256.99	264.44
LoS-D	$\sqrt[4]{\mathbf{H}_4^{mFr}} \cup \sqrt{\mathbf{H}_4^{mFr}}$	Lin.Regr.	16.50	18.90	27.40
LoS-A	$\sqrt[4]{\mathbf{H}_4^{mFr}} \cup \sqrt{\mathbf{H}_4^{mFr}}$	Lin.Regr.	96.229	263.49	220.02
NLoS-A	$\sqrt[4]{\mathbf{H}_4^{mFr}} \cup \sqrt{\mathbf{H}_4^{mFr}}$	Lin.Regr.	199.24	254.20	220.09
LoS-D	$\sqrt{\mathbf{H}_4^{mFr}}$	EN.Regr.	20.91	18.62	19.93
LoS-A	$\sqrt{\mathbf{H}_4^{mFr}}$	EN.Regr.	131.17	234.83	229.23
NLoS-A	$\sqrt{\mathbf{H}_4^{mFr}}$	EN.Regr.	279.42	292.39	267.54

Table 7.7 Scores for basic ML algorithms on the Frequency-averaged ch. matrix

From the above results, the following can be reasoned:

- Polynomial features do not significantly effect validation results, and any small benefits come at a cost of greatly increasing the possibility of overfitting without regularization. For NLoS especially, overfitting becomes a bigger issue, but the gains in test MSE from having more data (as both the training and validation datasets are used to obtain test results in regression) show that nonlinear ML has potential. Regularization also seems to unduly penalize the NLoS scenario.
- Reducing the amount of data available does not seem to greatly effect performance, except for reducing the possibility of overfitting. Validation and test results seem to be at most slightly reduced, even if both the UEs and the Frequencies are averaged out. In fact, for the NLoS scenario, by far the best validation result snf the best training:validation proportion came from simple linear regression on the combined square and 4-th root datasets.

Overall, this analysis with linear and EN.Regr. indicates the following: positioning is possible in both LoS and nLoS, taking the square root of the input data improves all results, taking the fourth-root is advisable for NLoS beam-paths, and polynomial features are of minimal benefit due to overfitting. Furthermore, using the different sub- channel- matrices as features does not meaningfully effect validation results as compared to averaging them out to create a lower-dimensional input.

The above results aid in finding a compromise between low feature-counts and data-utility for ML. Due to the 4x higher dimensionality at minimal performance benefit and with no underlying physical intuition, the full channel matrices and Frequency-averaged channel matrices will not be utilized. The two investigated input data-types will therefore be the fully-averaged $2 \times 8 \times 8$ dimensional $\mathbf{X}_H^{FA} := \sqrt{\mathbf{H}_1^{FA}} \cup \sqrt[4]{\mathbf{H}_1^{FA}}$ and a higher $6 \times 8 \times 8$ dimensional $\mathbf{X}_H^{mUE} := \sqrt{\mathbf{H}_3^{mUE}} \cup \sqrt[4]{\mathbf{H}_3^{mUE}}$.

8

Applied Deep Learning

8.1 Architecture

The optimal found structure for deep-learning consists of three discrete 'blocks' of fully-connected layers with ReLU nodes. To ensure scalability of the system to a more general system applicable to a larger area, network architecture was mostly unchanged between the ML models for the different datasets. Though in reality analyzing the exact function different layers in a deep ANN w.r.t. information processing is an entire research field, and for this thesis an exercise in futility. However, it is often convenient to 'assign' a designated task each block of layers is intended to fulfill. The three blocks are, along with their intended approximate purpose:

1. **Data Input block:** The idea behind the data input block is that it serves to take the input data and narrow it down to a small number of dimensions (20-40) for an intermediary latent-space. Through empirical experimentation, the narrowing seems doable through three layers, of which the first's output is 2-3 times the dimensionality of the input data, the second layer's output is half the dimensionality of the first layer, and finally the third layer outputs into the desired reduced dimensionality. When the input datatype is changed between $\mathbf{X}_H^{FA}$ and $\mathbf{X}_H^{mUE}$, only the input block is changed.

2. **Process block:** The expectation is that the process block takes the reduced dimensions generated through the *data input block* and with minimal or no dimensional reduction feeds it through 1-3 layers to process nonlinear relations for the data in the lower-dimensional latent space. Optionally, a small continuous decrease of 0-20% for layer output dimensionality can be introduced to further squeeze the data.

3. **Positioning block:** The expectation is that the positioning block takes the process block's output latent-space and through 2-4 layers rapidly narrows it down to two dimensions to regress for position. Note that the last layer has two outputs corresponding to position.

A visualization of the intended three blocks with only one part of the input data can be seen in Figure 8.1.

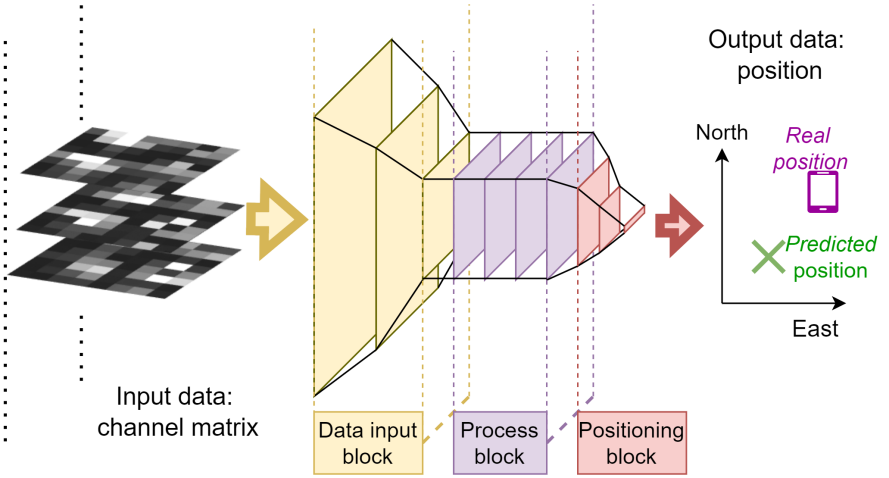


Figure 8.1 A visualization of the 3-block deep neural network architecture used in this thesis along with the intended input and output. In this example, the Data input consists of 3 fully-connected layers, the process block 4 fully-connected layers, and the positioning block 3 fully-connected layers. Note that though in this visualization layers are shown as a 2-dimensional plane, in reality all dimensions are flattened.

The final models can be seen in Table (8.1). A much wider hyperparameter-space than shown in Appendix Table (11.2) was explored in general trial-and-error to find a smaller subset of viable hyperparameters, with the space spanned by the narrowed set of hyperparameters then randomly sampled-from during the training-validation loop using a uniform distribution to find the final models.

Dataset	Input	N_{max}^{input}	$D^{proc.}$	$N_{max}^{proc.}$	A^{proc}	$D^{pos.}$	$N_{max}^{pos.}$	A^{pos}
LoS-D	$\mathbf{X}_H^{FA}$	329	1	33	1	4	19	0.95
LoS-A	$\mathbf{X}_H^{FA}$	365	1	35	1	2	21	1
NLoS-A	$\mathbf{X}_H^{FA}$	360	1	35	1	2	19	1
LoS-D	$\mathbf{X}_H^{mUE}$	432	1	28	1	2	28	0.8
LoS-A	$\mathbf{X}_H^{mUE}$	494	3	35	0.9	2	21	1
NLoS-A	$\mathbf{X}_H^{mUE}$	546	2	27	0.9	2	17	1

Table 8.1 Found optimal single-ML deep-learning models. N_{max}^{block} refers to the number of neurons in the largest layer in the block, D^{layers} refers to the depth of the block in terms of layer number, and A^{layers} refers to the $N_{min}^{block}/N_{max}^{block}$ ratio, with each block having the maximum layer be the first, and the minimum layer the last.

8.2 Applied Regularization and training hyperparameters

Dropout

Utilizing dropout regularization (see Section 5.8) for most layers improved results substantially. After fine-tuning it on validation data, a different per-layer and per-block dropout was used.

1. Dropout for the *data input block* was set to 5% for the first input layer, and 10% to the other layers in the block.
2. Dropout for the *Process block* was set to an even 12%
3. Dropout for the *Positioning block* was set to 0% for the last and before-last layer, and 10% for other layers should they exist.

Noise injection

Three types of noise injection were used. These were additive output noise, additive input noise, and multiplicative input noise. The function and magnitude of each is:

- **Additive output noise:** The GNSS datasource used consistently obtained a STD accuracy of around 3 meters. To both model this uncertainty in training position and to prevent overfitting, an additive Gaussian noise with zero-mean and an STD of 2.8 meters was added on to the training data, changing with each iteration. The underlying assumption that GNSS inaccuracy can be modelled with an additive Gaussian is of course a poor approximation, but without more advanced GNSS data processing and information about the GNSS measurement device IMU sensor fusion, no knowledge about noise characteristics is possible other than the STD accuracy.
- **Additive input noise:** Due to a lack of time to properly investigate priors on the input data, the per-direction noise is not well known. Instead, a simple model of Gaussian additive and multiplicative noise is used. The additive noise w is sampled from a zero-mean normal distribution: $w \sim \mathcal{N}(0, \sigma^2)$. The σ standard deviation of the additive noise is empirically 2 magnitudes (10e-2) smaller than the smallest nonzero input feature for all datasets. Therefore, additive noise is there to 'simulate' roughness on the zero-value directions.
- **Multiplicative input noise:** The multiplicative noise is there to create magnitude-dependent noise for the features, and consists of multiplying the input features by $(1-Z)$, where $Z \sim \mathcal{N}(0, \sigma^2)$ with std σ of the same magnitude as for the additive input noise.

One final processing step is to ensure that the input can not take negative values by taking the noise-injected input feature's absolute value as input. This is done as even with noise, amplitude and derived quantities should never be negative.

Early Stopping

The applied method of Early Stopping was slightly more complicated than the one described in Section 5.5. The validation loss was tracked both with a moving average of the past 6 epoch's validation errors and with the best previous loss. The model scoring better than the previous retained best model w.r.t. validation accuracy but was also part of a moving average better than the previous best moving average was kept. This more complicated setup was to filter out the noise found in the validation accuracy, and only keep models that were significantly better than earlier models - thereby hopefully improving generalization to the final test dataset.

In summary, even if the model was trained for an extended amount of time, only the epoch resulting in the model with the lowest *general* validation loss was kept. The step-by-step of the early stopping can be seen in Algorithm 3.

Algorithm 3: The developed early-stop and training algorithm

Result: Trained best model $\mathbf{M}_{\mathbf{best}}$, training T_k and validation V_k history
Input: Number of Epochs N_e, Minimum Validation Error V_{min}
Input: Initialized ML model $\mathbf{M}_0$
Input: Training iterator $\mathbf{M}_k, T_k \leftarrow Train\left(\mathbf{M}_{k-1}, \mathbf{X}_t r, \mathbf{Y}_t r\right)$
Input: Validation function $V_k \leftarrow Valid\left(\mathbf{M}_k, \mathbf{X}_v, \mathbf{Y}_v\right)$
Data: training dataset $\mathbf{Y}_{tr}, \mathbf{X}_{tr}$, validation dataset $\mathbf{Y}_v, \mathbf{X}_v$
Initialize: Empty Validation, training history vector: $\mathbf{V}_0$, $\mathbf{T}_0$;
Initialize: Model temporary storage vector: $\mathbf{m}_6 \leftarrow 6 \times [NaN]$;
for $k \leftarrow 0$ ***while*** $k < N_e$ **do**
  Train model for one epoch: $\mathbf{M}_k, T_k \leftarrow Train\left(\mathbf{M}_{k-1}, \mathbf{X}_{tr}, \mathbf{Y}_{tr}\right)$;
  Update validation error: $V_k \leftarrow Valid\left(\mathbf{M}_k, \mathbf{X}_v, \mathbf{Y}_v\right)$;
  Epoch check: If $\{V_k > V_{min}\}$ then N_e++ ;
  Remove oldest element in model temporary storage: $pop(\mathbf{m}_6)$;
  if $V_k < V_{best} - 0.1$ **then**
    Add model: $append\left[\mathbf{m}_6\right] with \left[\mathbf{M}_k\right]$;
  else
    Add empty element: $append\left[\mathbf{m}_6\right] with \left[NaN\right]$;
  end
  if $mean\left(V_{max(0,k-6):k}\right) < \bar{v}_{best} - 0.1$ ***and***
   $min\left(V_{max(0,k-6):k}\right) < V_{best} - 0.1$ **then**
    Update best validation: $V_{best} \leftarrow V_{\arg\min_k (V_{max(0,k-6):k})}$;
    Update best model: $\mathbf{M}_{best} \leftarrow m_{\arg\min_k (V_{max(0,k-6):k})}$;
    Update best average validation: $\bar{v}_{best} \leftarrow mean\left(V_{max(0,k-6):k}\right)$;
  end
  $k++$
end

Explicit regularization

Alongside the above regularization methods, weight decay - L^2 regularization - is implemented by the AdamW optimizer. For weight decay, a standard 0.001 performed well enough, with other regularization methods overshadowing the contribution of weight decay.

L^1 regularization in its classic sense is uncommon in many models, as instead other methods for introducing sparse weights (and/or pruning) for deep learning give better results [Hoefler et al., 2021]. However, due to time-constraints on training times during the writing of this thesis, no such method was experimented with.

Training and optimization

For epoch selection, the epoch counter only began after validation loss fell significantly below that of linear regression on the same dataset, with an epoch-start MSE of 200 for the NLoS-A and LoS-A datasets, and 35 MSE for the LoS-D dataset. This was to correct for a very occasional slow initial convergence. The models were trained on a mobile RTX 2060 GPU, using CUDA and PyTorch.

Due to contamination of test results through early-stopping, unlike before in (7.5) networks were not re-trained on combined validation and training data when evaluating the test dataset. A secondary benefit to this is reduced training time, which was already an issue due to the wide architecture-space to explore.

Each model was trained on each training dataset and tuned using the corresponding validation dataset, with training going for 150 useful epochs at a learning rate of 0.001 using the AdamW-amsgrad optimizer specified in Section 5.3. Due to early-stopping, the actual epoch-model used was usually found below 150 epochs. An example training graph of the LoS-D $\mathbf{X}_H^{FA}$ model can be seen in Figure 8.2.

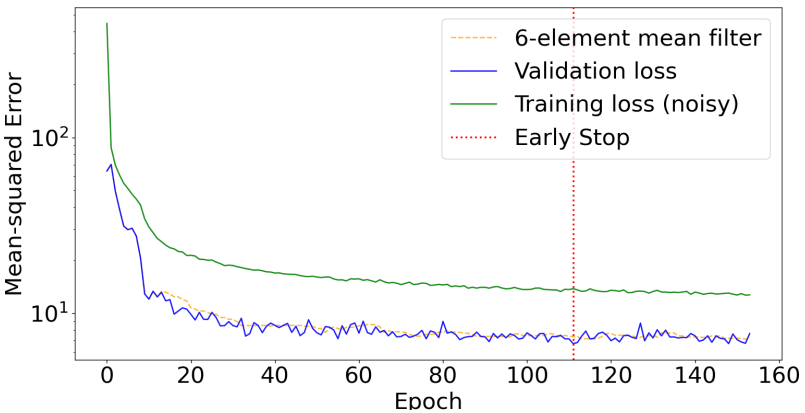


Figure 8.2 The training history of the 150-epoch LoS-D $\mathbf{X}_H^{FA}$ model. The training loss is much higher than shown in the result, as this includes all the injected noise.

8.3 Single-model Machine-Learning Results

The MSE results of the best discovered models for each dataset and input feature-pair using the architecture shown in Table (8.1) can be seen in Table (8.2). In addition to the standard 150 epoch model results shown in Table (8.2), also shown in Table (8.3) are the results when the best found models are trained on 1500 epochs with the learning rate halved after Epoch 150 with and without Early Stop. The expectation is then that there will be at most a minor improvement for validation MSE and test MSE. The non-early-stopped results should show significant overfitting, with good training MSE but no generalizability. E.g. The 1500 epochs training of NLoS-A $\mathbf{X}_H^{mUE}$ is shown in Appendix Figure 11.3.

Dataset	Input	Model	*tr.MSE*	*val.MSE*	*test.MSE*
LoS-D	$\mathbf{X}_H^{FA}$	single	3.9	6.7	9.0
LoS-A	$\mathbf{X}_H^{FA}$	single	5.4	81.1	85.4
NLoS-A	$\mathbf{X}_H^{FA}$	single	7.7	41.6	66.7
LoS-D	$\mathbf{X}_H^{mUE}$	single	2.7	6.2	8.3
LoS-A	$\mathbf{X}_H^{mUE}$	single	6.5	89.3	99.2
NLoS-A	$\mathbf{X}_H^{mUE}$	single	14.8	46.1	65.7

Table 8.2 Results on found optimal single-ML deep-learning models. The *model* column refers to applied regularization, where *single* is the baseline model.

Dataset	Input	Model	$\mathbb{E}_{stop}$ $\mathbb{E}_{end}$	*tr.MSE*	*val.MSE*	*testMSE*
LoS-D	$\mathbf{X}_H^{FA}$	1500$\mathbb{E}$	155	3.0	6.7	9.2
			1500	2.1	8.1	10.4
LoS-A	$\mathbf{X}_H^{FA}$	1500$\mathbb{E}$	109	5.4	81.1	85.4
			1500	4.1	108.4	114.8
NLoS-A	$\mathbf{X}_H^{FA}$	1500$\mathbb{E}$	97	7.7	41.6	66.7
			1500	2.0	47.8	67.1
LoS-D	$\mathbf{X}_H^{mUE}$	1500$\mathbb{E}$	241	1.9	7.3	9.2
			1500	1.0	8.0	10.1
LoS-A	$\mathbf{X}_H^{mUE}$	1500$\mathbb{E}$	89	6.5	89.3	99.2
			1500	2.1	128.1	129.1
NLoS-A	$\mathbf{X}_H^{mUE}$	1500$\mathbb{E}$	353	8.9	60.7	58.3
			1500	6.3	76.6	59.8

Table 8.3 Results on found optimal single-ML deep-learning models after Early-stop on 1500 epochs $\mathbb{E}$. Also shown are the resulting models after 1500 epochs. All regularization is used, and the Early Stop Epoch $\mathbb{E}_{stop}$ is also shown.

Discussion of single-ML results

The first noticeable aspect of the basic single-ML results is how much of an improvement the MSE rates are compared to the various regression models. Even in the worst case on the test data, the achieved MSE was more than twice as good for all scenarios than for even the best regression model. In meters (taking the square-root of the MSE), the error radius on the test data seems to be around 2.8-3 meters for the well-behaved LoS and around 8-10 meters for the poorly-behaved LoS and NLoS models.

The well-behaved LoS dataset has errors roughly similar to what is expected from the underlying ground-truth dataset. The suspicion is then that the ML model is limited in truth by practicalities of obtaining more accurate positional training data and not by the model or the underlying theory. A visual demonstration of the validity of these result is in Figure 8.3, where a point-cloud shows the predictions the single ML $\mathbf{X}_H^{mUE}$ model made on the LoS-Dv *test* dataset.

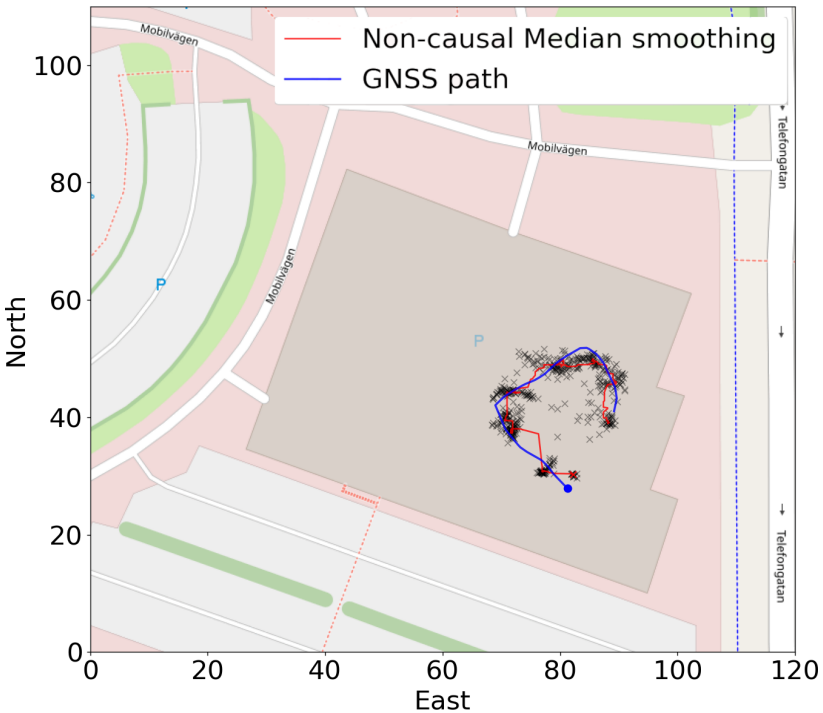


Figure 8.3 The predictions (black crosses) made by a single ML $\mathbf{X}_H^{mUE}$ model over a subset of the test data for the LoS-Dv dataset. The GNSS path is shown in blue, and shown in red is a non-causal median smoothing applied to the prediction data (forward and backwards 17 elements).

From just the MSE of the LoS-A2 results, the ML model appears to perform alright, with an MSE of around 80-100 [m^2] - a root-MSE of around 9-10 meters. Though it vastly outperforms linear regression on the same dataset, it is far worse than expected from a LoS dataset - which the training MSE values indicate it is.

Looking however at the results of the ML algorithm on the poorly behaved LoS-A2 and comparing them to GNSS data, something interesting is immediately visible. The GNSS data seems to 'think' that we are not on the roof of the building much of the time, giving an immediate almost 5-8 meter offset from the true location the data was recorded on. In fact, just from looking at the results Figure 8.8 and the route-plan from Figure 6.3, it is visible that in the case of the test data, the ML actually sometimes outperformed the GNSS 'ground truth' in predicting location.

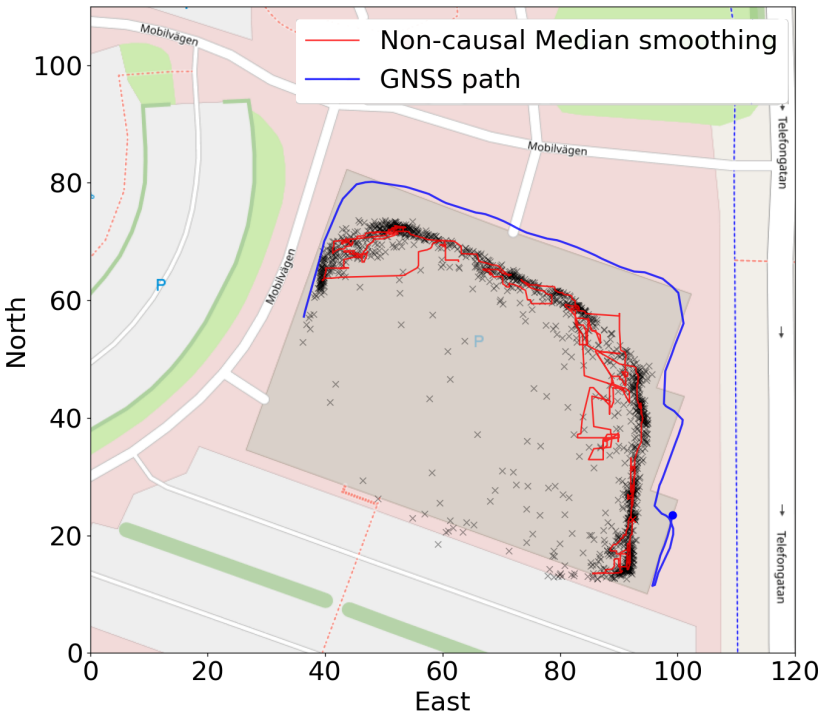


Figure 8.4 The predictions (black crosses) made by a single ML $\mathbf{X}_H^{mUE}$ model over a subset of the test data for the LoS-A dataset. The GNSS path is shown in blue, and shown in red is a non-causal median smoothing applied to the prediction data. Note that GNSS veers way offroute, while the predicted location is more accurate to how the 'true' walk looked like. Therefore, the low error is a combined artifact of GNSS data being poor and the LoS-A2 dataset having a different character as compared to the training LoS-A1 dataset.

For the NLoS-A test data, the test MSE results were 3-4 times better than for any of the regression models used in Section 7.5, achieving a MSE of around 60-70 $[m^2]$. This corresponds to a roughly 8 met root-MSE, which is already a fairly good start for raw output for a positioning system - especially one where there is already an intrinsic error while training. The resulting position prediction cloud for much of a single iteration can be seen in Figure 8.5.

Also note how much worse the system is at predicting the user in the north-east of the path. In that area the NLoS characteristics are exacerbated by foliage cover, while in the south-west though the basestation does not have direct LoS, there is more open-space and reflective surfaces for optimal MPCs. This shows that not all NLoS scenarios are 'made equal'.

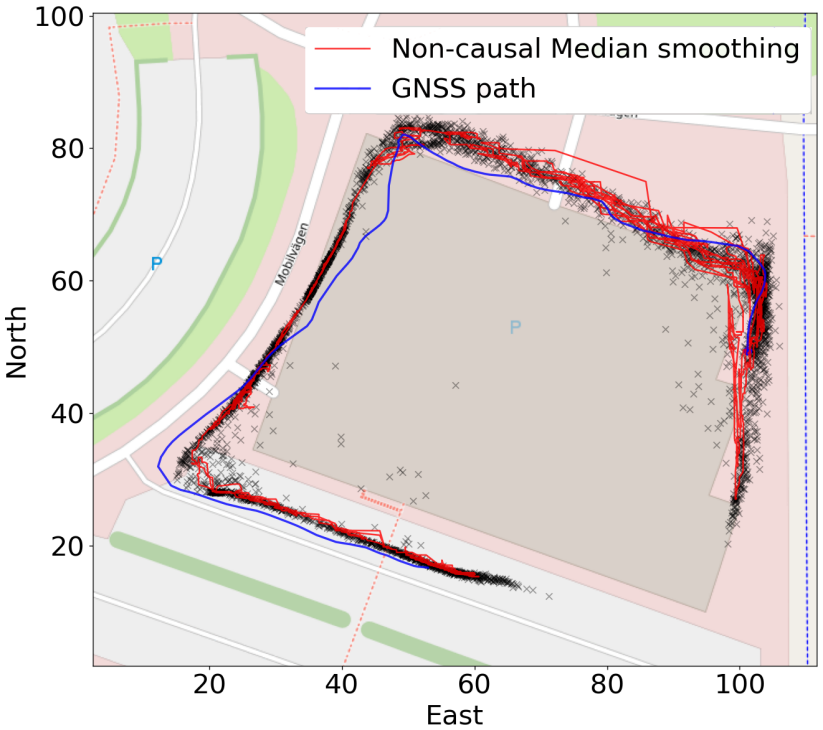


Figure 8.5 The predictions (black crosses) made by a single ML $\mathbf{X}_H^{mUE}$ model over a subset of the test data for the NLoS-A dataset. The GNSS path is shown in blue, and shown in red is a non-causal median smoothing applied to the prediction data. Note again that GNSS veers offroute. The 'true' user position could be somewhat reconstructed even by just the results of one ML algorithm.

8.4 Ensemble methods, results, and discussion

Applied Ensemble methods for CSI

Discussed in Section 5.5, Ensemble machine learning is generally a far more effective use of the (limited) compute resources available than overtraining, as seen in Table (8.3). As mentioned, in literature ensembles are not utilized when directly comparing ML methods as performance improves with increased numbers of component networks, thereby making it a function of compute power more-so than method. The goal of this thesis is work on developing a positioning pipeline, for which ensembles are a useful tool despite the latter statement.

Furthermore, Ensemble networks can be utilized for Particle Filtering to approximate a probability density function when combined with kernel density estimates, as discussed in Section 9.1. For this chapter, a simpler way of utilizing Ensemble methods for regression is to average out the outputs of the component models for every datapoints - which will be the method used in this section.

Deep Learning Ensembles do not necessarily require groups of networks with different architectures, but it could possibly improve results. Therefore, two Ensembles variants will be examined; uniform-architecture network ensembles and diverse-architecture network ensembles. A relatively small ensemble of 10 networks will be created for each type and then compared, each trained with early-stopping for a maximum of 150 epochs - thereby giving an overall similar training time for each ensemble as the more computationally inefficient 1500 epoch full-networks shown in Table (8.3).

The parameter-space in which the component networks have their architectures sampled from can be seen in Table (8.4). Any parameter not shown is kept identical to those described in prior chapters.

Input		N_{max}^{input}	$D_{proc.}$	$N_{max}^{proc.}$	A^{proc}	$D_{pos.}$	$N_{max}^{pos.}$	A^{pos}
$\mathbf{X}_H^{FA}$	[min]	[256]	[1]	[24]	[0.8]	[2]	[16]	[0.6]
	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓
	{max}	{384}	{3}	{38}	{1.0}	{4}	{22}	{1.0}
$\mathbf{X}_H^{mUE}$	[min]	[392]	[1]	[24]	[0.8]	[2]	[16]	[0.6]
	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓
	{max}	{768}	{3}	{38}	{1.0}	{4}	{22}	{1.0}

Table 8.4 Range of architectures sampled for the diverse-ML ensemble of deep-learning models - where the ideal model from Table (8.1) is guaranteed to be one of the samples. N_{max}^{block} refers to the number of neurons in the largest layer in the block, D^{layers} refers to the depth of the block in terms of layer number, and A^{layers} refers to the $N_{min}^{block}/N_{max}^{block}$ ratio, with each block having the maximum layer be the first, and the minimum layer the last.

Mean-Ensemble method results

Both a diverse ensemble sampled from the parameters in Table (8.4) with 10 and 50 models along with an optimal-model-ensemble with 10 models using the optimal parameters seen in Table (8.1) were run on the full dataset. The resulting MSE for each of the models and datasets can then be seen in Table (8.5).

The expectation is then that the Optimal and Diverse ensembles both improve results over the baseline model at a roughly equivalent level, but placing 50 models instead of 10 will not meaningfully improve results. The 50 models will have a larger diversity of 'guesses' however, which will be utilized for particle filtering as per Chapter 9.

Dataset	Input	Model	*tr.MSE*	*val.MSE*	*test.MSE*
LoS-D	$\mathbf{X}_H^{FA}$	10-Diverse	3.5	5.7	8.1
LoS-A	$\mathbf{X}_H^{FA}$	10-Diverse	6.9	76.0	81.8
NLoS-A	$\mathbf{X}_H^{FA}$	10-Diverse	10.8	38.9	63.4
LoS-D	$\mathbf{X}_H^{mUE}$	10-Diverse	3.0	5.2	7.3
LoS-A	$\mathbf{X}_H^{mUE}$	10-Diverse	8.4	79.4	91.6
NLoS-A	$\mathbf{X}_H^{mUE}$	10-Diverse	10.1	43.4	61.6
LoS-D	$\mathbf{X}_H^{FA}$	10-Optimal	3.9	5.7	8.2
LoS-A	$\mathbf{X}_H^{FA}$	10-Optimal	5.5	73.9	79.8
NLoS-A	$\mathbf{X}_H^{FA}$	10-Optimal	7.5	38.8	56.9
LoS-D	$\mathbf{X}_H^{mUE}$	10-Optimal	2.7	5.7	7.7
LoS-A	$\mathbf{X}_H^{mUE}$	10-Optimal	8.9	85.5	98.4
NLoS-A	$\mathbf{X}_H^{mUE}$	10-Optimal	6.7	43.3	58.2
LoS-D	$\mathbf{X}_H^{FA}$	50-Diverse	3.3	5.5	7.9
LoS-A	$\mathbf{X}_H^{FA}$	50-Diverse	6.3	74.8	81.2
NLoS-A	$\mathbf{X}_H^{FA}$	50-Diverse	11.4	41.8	59.8
LoS-D	$\mathbf{X}_H^{mUE}$	50-Diverse	2.9	5.2	7.2
LoS-A	$\mathbf{X}_H^{mUE}$	50-Diverse	7.9	80.7	92.1
NLoS-A	$\mathbf{X}_H^{mUE}$	50-Diverse	10.6	43.8	60.4

Table 8.5 MSE results on the different datasets using the two 10-element ensemble deep-learning models along with one 50-element ensemble model after Early-stop on 150 epochs. The *model* column refers to the type of ensemble model, where 10-Diverse refers to the 10 random chosen architectures within the hyperparameter bounds defined in Table (8.4) while also including a single instantiation of the optimal found model described in Table (8.1). The 10-Optimal then refers to training the Optimal found model 10 times in an ensemble. The 50-Diverse model is a 50-element variant of the diverse ensemble, and is also the same instantiation as the ensemble that will be used for Particle Filtering.

Discussion of Ensemble results

As expected, the MSE of the Training, Validation, and Test datasets were significantly reduced, with an around 10% improvement, through even a 10-model ensemble as compared to the ideal single-model result. This held for both the Diverse and the Optimal Ensembles. The 10-optimal also seems to modertately outperform the 10-diverse in the NLoS-A dataset, while the reverse is true for the LoS-D dataset.

Looking visually at the predictions, the Ensembles for both the LoS-A and the NLoS-A show visible improvement from using Ensembles as compared to the single model results from before. See Figure 8.6 for one of the ensemble LoS-D test results, and Figure 8.7 for one of the ensemble NLoS-A test results. Due to how similar the results were in these instances, only one of the possible models had their results shown.

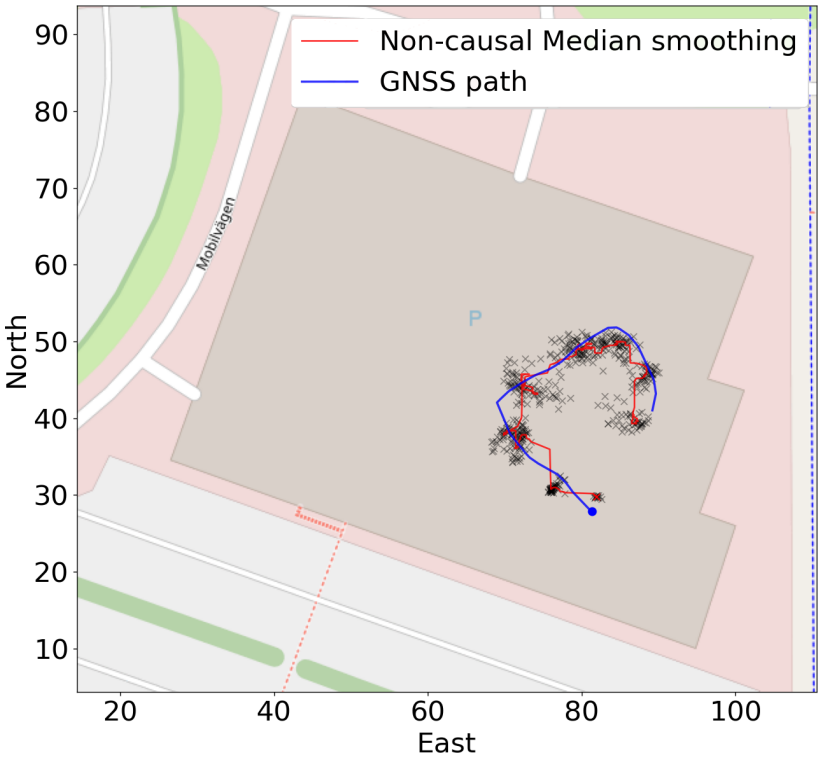


Figure 8.6 The predictions (black crosses) made by the 10-Diverse ML $\mathbf{X}_H^{FA}$ model over a subset of the test data for the NLoS-A dataset. The GNSS path is shown in blue, and shown in red is a non-causal median smoothing applied to the prediction data (forward and backwards 17 elements).

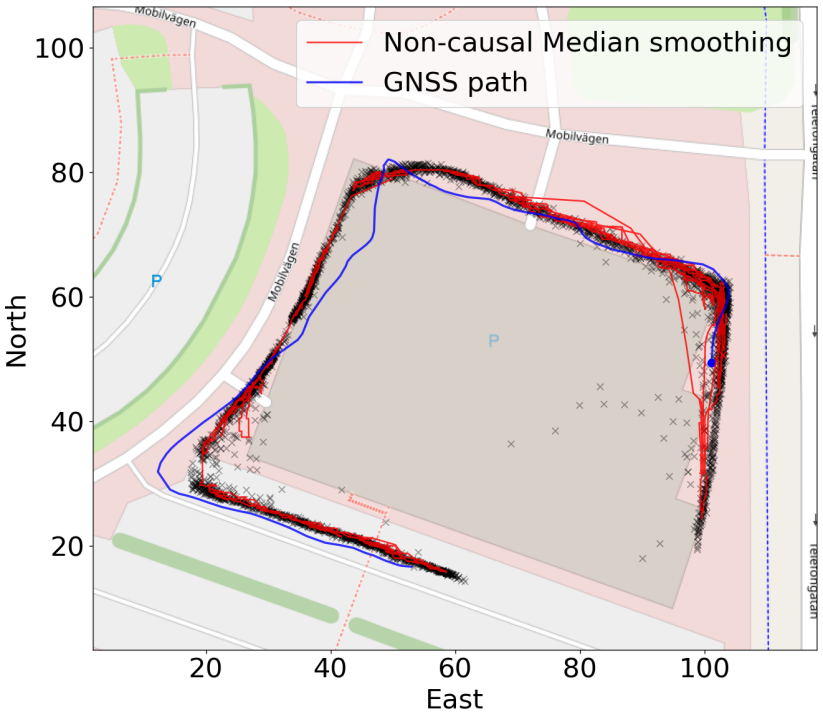


Figure 8.7 The predictions (black crosses) made by the 10-Diverse ML $\mathbf{X}_H^{FA}$ model over a subset of the test data for the NLoS-A dataset. The GNSS path is shown in blue, and shown in red is a non-causal median smoothing applied to the prediction data (forward and backwards 17 elements).

We speculate that the cause of the better performance in NLoS for the Optimal and in LoS for the Diverse model might be that a more diverse set of models is needed to gain an ensemble advantage of averaging-out uncorrelated errors when the task is simpler, while a more complicated task such as NLoS positioning might be more sensitive to models having a well-performing architecture. Overall, however, in both cases the end scores were fairly close and showed the expected significant performance improvement MSE over just using a single model - with the exception of the anomalous test result from the 1500 Epoch-trained and overfitted $\mathbf{X}_H^{mUE}$ single-model, which scored a good test result despite the expected bad validation result.

Interestingly, using the Fully-averaged $\mathbf{X}_H^{FA}$ dataset also appears to bring consistent improvements over the mean-UE $\mathbf{X}_H^{mUE}$ dataset - with the exception of the LoS-D dataset. Unexpectedly, this holds true even for the training error, despite the greater potential for overfitting a large number of input parameters brings with it. We speculate that the regularization techniques reduce the advantage of over-

parameterization, while the small information advantage that introducing separate frequencies brings with it is too complicated to properly parse with the relatively small models and limited available data.

Interestingly, the model in which $\mathbf{X}_H^{FA}$ outperformed the $\mathbf{X}_H^{mUE}$ dataset the most was in the LoS-A dataset - in training, but especially in the validation and test results. We speculate that the fewer number of input parameters enable a much more generalizable model which functions better to position points in a dataset with a somewhat different character. The visual result is however still not a dramatic improvement over using a single ML model, with the median-smoothed path still showing a lot of jumping, as visible in Figure 8.8.

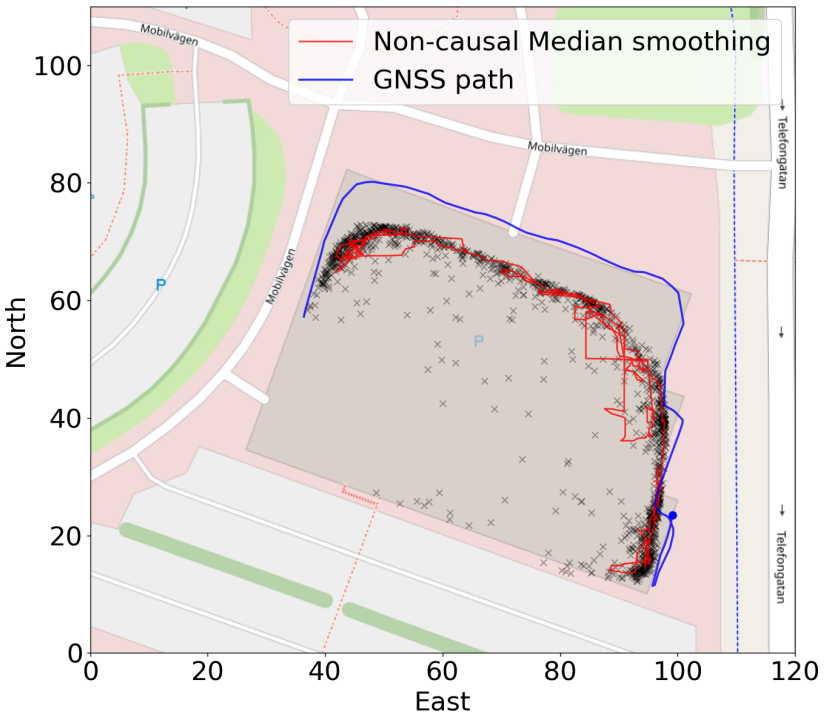


Figure 8.8 The predictions (black crosses) made by the 10-Diverse ML $\mathbf{X}_H^{FA}$ model over a subset of the test data for the LoS-A dataset. The GNSS path is shown in blue, and shown in red is a non-causal median smoothing applied to the prediction data (forward and backwards 17 elements).

Overall, the results from machine learning alone hold a lot of promise, and will be further utilized through particle filtering in the following Chapter 9 to estimate the underlying motion navigational pathways generating the independent samples.

9

Applied Particle Filtering

9.1 Kernel Density Estimation with Ensembles

Particle filtering for position using deep-learning

With a method for obtaining point guesses (or cluster-of-point guesses) developed using deep learning in Chapter 8, a way of turning individual non-time-dependant samples into position estimates is shown to be feasible. The next step is to then integrate time-domain knowledge into the estimation model.

In literature, time-based models such as Recurrent Neural Networks are sometimes used - see Section 1.2. Due to computational limitations, this approach is not feasible for this thesis. For navigation and positioning in most fields, the most common choice is to go with a form of Bayesian Filtering, such as Kalman Filters, Extended Kalman Filters, or Particle filters, or some hybrid of thereof - as discussed in Section 4.1. The choice depends largely on the linearizability of the system model and the observation noise characteristics.

In a hypothetical ideal positioning system for SRS channel matrices, the UE state vector x_k at time k is a markovian system with a known state evolution probability $Move(x_{k+1}|x_k)$, where the act of observing the state x_k through a channel matrix $\mathbf{H}$ can be viewed as akin to sampling from a probability distribution $\mathbf{H}^i \sim Hgen(\mathbf{H}|x_k)$. Then, assuming the markovian property, a chain of state observations $\hat{x}_{1:k}$ from channel matrices $\mathbf{H}_{1:k}$ can be created using Particle Filtering. The only additional needed assumption is then that a $PDF_k^{obs}\left(\tilde{x}^i|\mathbf{H}_k\right)$ instantaneous Observation probability density function is known, giving the observation probability for any arbitrary i state $\tilde{x}^i$ at time k.

The issue is then that deep-learning models $\mathbf{M}(\mathbf{H}_k)$ create $\hat{y}_k^{EN}$ point-observations. In essence, they can be viewed as a way of non-randomly sampling position observations y_k^{EN} from from an underlying unknown probability distribution $Obs_k(\hat{y}_k|\mathbf{H}_k)$, which can be viewed as representing the distribution of the possible outputs for an ideal position prediction system given the possible channel matrices $\mathbf{H}^i$ generated by all the states x that are in position y^{EN}.

The task is then to find a way to approximate the PDF corresponding to the ideal $Obs_k(\hat{y}_k|\mathbf{H}_k)$. For this, a way of obtaining multiple samples from ML models is required. There are two primary ways of introducing randomness to hopefully obtain multiple y_k^{EN} samples from the unknown underlying $Obs_k(\hat{y}_k|\mathbf{H}_k)$: by introducing a series of N noise-injected $\tilde{\mathbf{H}}_k^{1:N}$ to the input of the model $\mathbf{M}$, or by using a single channel matrix but sending it through a number of N different models $\mathbf{M}^{1:N}$, each creating a different prediction. There are many ways of generating different models, including taking trained models and applying dropout regularization on the validation/test results iteratively, re-training the same model using e.g. different training conditions, different epochs of the same model, re-training different architectures, etc...

As the noise prior for the input channel matrices is not well known, and since ensembles of models were already trained to output point-clusters, the multiple-model ensemble of networks method of obtaining N random samples of y_k^{EN} was convenient to use. For this, the diverse models were then chosen to improve diversity of output points - the hope being that different models perform differently w.r.t. accuracy and consistency for different parts of the dataset, leading to an overall consistent prediction rate.

The final task is to then find a way to find an approximate observation PDF $\widehat{Obs}_k(\hat{y}_k|\mathbf{H}_k)$ using the set of $\hat{y}_k^{1:N}$ observations generated by the models $\mathbf{M}^{1:N}(\mathbf{H}_k)$. This field is known as *density estimation*, and two extremely common and simple methods for it are *Gaussian Mixture Models* and the non-parametric *Kernel Density Estimation*.

Kernel Density Estimation

In *Gaussian Mixture Models* a number of multivariate Gaussian distributions are fitted onto the discrete sample data such that entropy is minimized, while *Kernel Density Estimation* - also known as Parzen's window - consists of applying a variable kernel onto the discrete sample data to 'smooth' it out into a distribution [Chen, 2017]. Due to time-constraints while working on this thesis and with the assumption that the underlying PDF does not necessarily fall into neat Gaussian-like clusters, the chosen method for finding an approximate PDF was Kernel Density Estimation (KDE). Furthermore, KDE is non-parametric, requiring no underlying assumptions for the distribution other than the kernel type.

KDE for position estimation in the current context is defined as the density estimation shown in Equation (9.1), where $\hat{p}_k$ is the estimated density function of the unknown observation distribution P_k, the observed samples $Y_k^1, \cdots Y_k^i, \cdots, Y_k^N \in \mathbb{R}^d$ are machine-learning observations from the hypothetical distribution P_k, $\mathbf{K}$: $\mathbb{R}^d \rightarrow \mathbb{R}^1$ is the smoothing function (kernel) with a unit integral, and h is the bandwidth.

$$\hat{p}_k(z) = \frac{1}{Nh^d} \sum_{i=1}^{N} \mathbf{K}\left(\frac{z - Y_k^i}{h}\right) \tag{9.1}$$

A common smoothing function (kernel) for KDE is the Gaussian kernel, which is the one utilized for the implementation of the particle filter. The Gaussian Kernel can be seen in Equation (9.2).

$$\mathbf{K}(z) = \frac{e^{\frac{-||z||^2}{2}}}{z_{norm}}, \qquad z_{norm} = \int e^{\frac{-||z||^2}{2}} dz \tag{9.2}$$

Though there are methods to enable automatic bandwidth adjustment [Chen, 2017], for this thesis it seemed fairly obvious to use a similar bandwidth as the GNSS data uncertainty, at around a 2.8-3.2 meter bandwidth. Through empirical tuning of the bandwidth on the validation data, this number seemed optimal as well.

For an example of the generated approximate PDF using KDE on test data using a diverse-ensemble of 50 $\mathbf{X}_H^{FA}$ input-models, see Figures (9.1, 9.2). Shown are both a fairly ideal scenario in the LoS-D test dataset and a very scattered LoS-A2 test dataset point prediction.

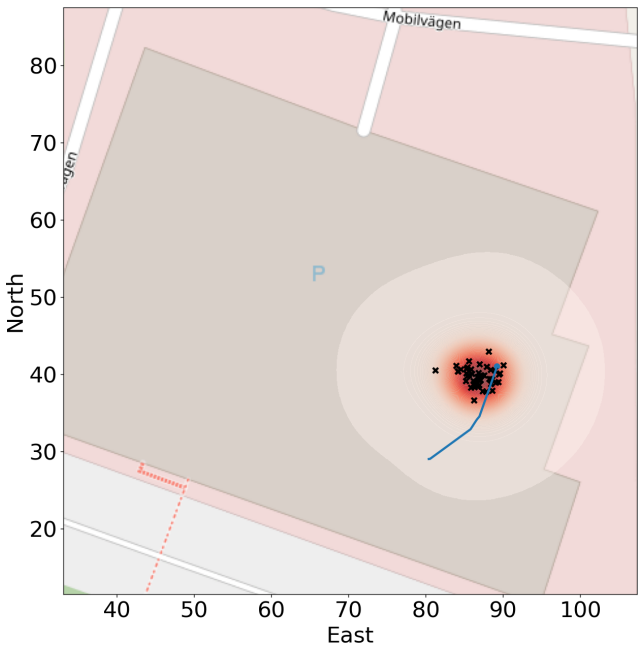


Figure 9.1 The resulting PDF contour-plot of KDE on 50 observations done by 50 different models on a single $\mathbf{X}_H^{FA}$ channel matrix datapoint in the well-behaved LoS-Dtr test sub-dataset. The 'ground-truth' GNSS-interpolated location is shown as a blue point, with the past few seconds of movement shown by the blue path. The PDF shown is extremely well-clustered, essentially indistinguishable from a gaussian.

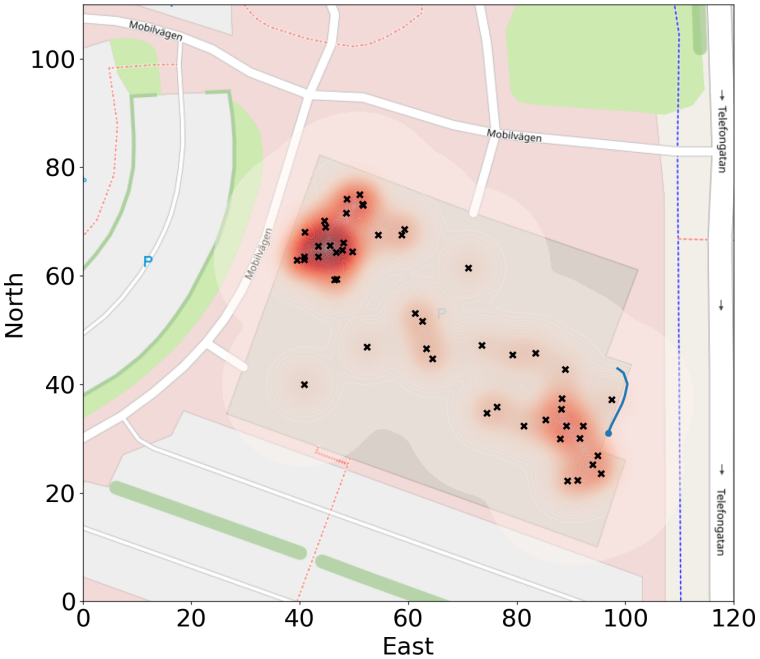


Figure 9.2 The resulting PDF contour-plot fitted using a KDE with a bandwidth of 3 on 50 observations done by 50 different models on a single $\mathbf{X}_H^{FA}$ channel matrix datapoint in the poorly-behaved LoS-A2 test sub-dataset. The 'ground-truth' GNSS-interpolated location is shown as a blue point, with the past few seconds of movement shown by the blue path. The PDF is very scattered, but with a distinct 'cluster' of detections near the actual position. The hope is that even in this very poorly behaved dataset, the few models that 'get it right' in each part coupled with the kinematics of the filtered particles lead to an actual path being 'smoothed' out of the data. Note the fact that the GNSS path is 'going off the building' - demonstrating the inaccuracies inherent to using GNSS data for the ground-truth

Overall, even in the poorly-behaved LoS scenario, the poor accuracy conceals the fact that different models perform better or worse w.r.t. accuracy. Therefore, if a sufficient number of diverse-enough models are utilized, then the hope is that a certain (changing) selection of models will be accurate, with the kinematics of the particle filter smoothing out so that only the 'proper' observations are retained for most samples.

For this reason, choosing a proper system model to model state evolution of the particles such that the 'poor' guesses are filtered out without requiring large compute overhead is important - as discussed in the following Section 9.2.

9.2 System Motion Model

Ensuring an accurate-enough state simulation without keeping track of too many states is critical for a well-functioning particle filter. To make the simplest possible representation of the motion model for a particle moving in a 2D plane, one can look at the coordinate frame from Section 3.2. It is quite visible that the UE has 3 degrees of positioning freedom: angle of motion α, and the X and Y coordinates of the reference frame - corresponding to the East and North coordinates in the EN reference frame. With this, movement on relatively small scales can be simply modelled as a change in angle of the direction of motion and a vector displacement of a certain size in the appropriate direction. This simple motion model is shown in Figure (9.3), with the simple mathematics for a particle i described in Equation (9.3).

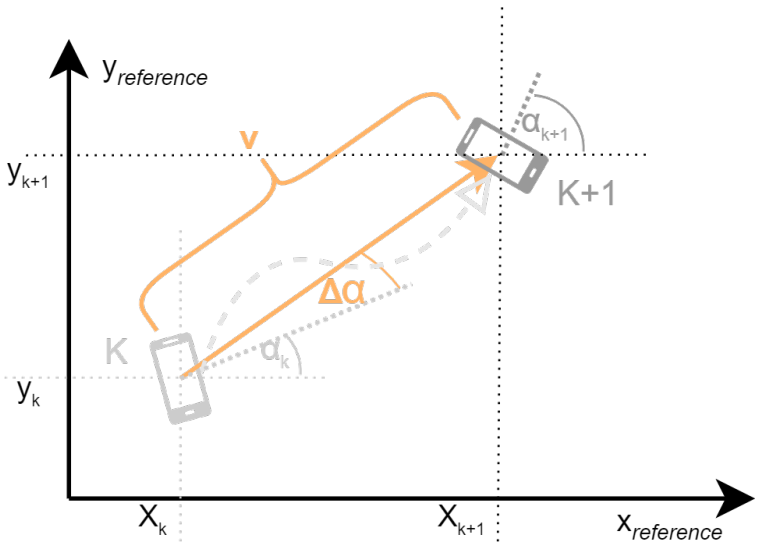


Figure 9.3 A simple model of 2D motion of a particle i, with a $\Delta\alpha$ shift in direction of movement angle α from time-point k to $k+1$, along with a movement vector of magnitude v in the new orentation's direction, thereby giving the new E^i_{k+1}, N^i_{k+1} position through the simple trigonometry described in Equation (9.3).

$$Pos^i_{k+1} = \begin{bmatrix} E^i_{k+1} \\ N^i_{k+1} \\ \alpha^i_{k+1} \end{bmatrix} = \begin{bmatrix} E^i_k + \cos\left(\alpha^i_k\right) v \\ N^i_k + \sin\left(\alpha^i_k\right) v \\ \alpha^i_k + \Delta\alpha^i \end{bmatrix} \tag{9.3}$$

As we want to keep states relatively low, and since the only other relatively consistent quantity is that of velocity, we define the state vector x_k for particle i at time

k using East, North, Orientation, and Speed. The change in orientation $\Delta\alpha$ and the change is velocity Δv was then modelled as additive zero-mean gaussian noise.

Adding velocity into the state-equation as a new state then gives the i-th particle the state vector x_k^i at time k, as shown in Equation (9.2) with the zero-mean Gaussians characterized with σ_v and σ_α.

$$x_{k+1}^i = \begin{bmatrix} E_{k+1}^i \\ N_{k+1}^i \\ \alpha_{k+1}^i \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} E_k^i + \cos\left(\alpha_k^i\right) v \\ N_k^i + \sin\left(\alpha_k^i\right) v \\ \alpha_k^i + \Delta\alpha_k^i \\ v_k^i + \Delta v_k^i \end{bmatrix} \tag{9.4}$$

$$\begin{aligned} \text{Where:} \quad & \forall i, \Delta v_k^i \sim \mathcal{N}(0, \sigma_v^2) \\ & \forall i, \Delta\alpha_k^i \sim \mathcal{N}(0, \sigma_\alpha^2) \end{aligned}$$

The above is a simple but effective movement model, but suffers from two deficiencies: there are no limits on what values velocity can take, and the variable sample-rate is not taken into account. Fixing both deficiencies is a relatively simple affair. For the variable sample rate, measuring the elapsed time in milliseconds between samples is trivial. After that, the effective velocity (the change in position) is multiplied by the second-normalized time Δt_k, while the change in angle $\delta\alpha_k^i$ and velocity $\delta\alpha_k^i$ is sampled from a normal distribution with a width multiplied by the second-normalized time. The time-adjustment then results in the system Equation (9.2), where Δt_k refers to the time elapsed between samples k and $k+1$.

$$x_{k+1}^i = \begin{bmatrix} E_{k+1}^i \\ N_{k+1}^i \\ \alpha_{k+1}^i \\ v_{k+1} \end{bmatrix} = \begin{bmatrix} E_k^i + \cos\left(\alpha_k^i\right) v\Delta t_k \\ N_k^i + \sin\left(\alpha_k^i\right) v\Delta t_k \\ \alpha_k^i + \Delta\alpha_k^i \\ v_k^i + \Delta v_k^i \end{bmatrix} \tag{9.5}$$

$$\begin{aligned} \text{Where:} \quad & \forall i, \Delta v_k^i \sim \mathcal{N}(0, \Delta t_k \sigma_v^2) \\ & \forall i, \Delta\alpha_k^i \sim \mathcal{N}(0, \Delta t_k \sigma_\alpha^2) \end{aligned}$$

Constraining the possible values the particle filter can take is also simple. Knowing that we are tracking a walking pedestrian and that 'going backward' can be modelled by changes in orientation instead of negative velocities, we can safely constrain the velocity to take up values between 0 and 1.8 $[m/s]$. In a future system, the type of user we are tracking can e.g. be modelled using hidden discrete states characterizing different turn-rates, velocities and speed-limits. For this, a hidden Markov Model would work. However, for this thesis, modelling a walking pedestrian was deemed sufficient as a proof-of-concept.

Furthermore, when observing the location of a particle, the East and North location is simply extracted, without any additive noise or modelling of doppler shift.

9.3 Particle Filter Adjustments

All components are in place for a standard particle filter implementation, but a few more adjustments are advisable to tackle certain issues with Particle Filters in wireless positioning contexts. The two big issues that plague Particle Filters are that of *sample degeneracy* and *sample impoverishment* [Li et al., 2013]. Tackling them in this thesis required some unique adjustments: partial resampling and weight-inertia.

Partial resampling and Weight Inertia

Sample degeneracy is when only a few samples of high-weight exist, with most having negligible or near-zero weights. This is generally countered by proper resampling, as detailed in Section 4.3. However, overdoing resampling can lead to an almost opposite problem: when particles with a high weight concentrate in a very small area and converge to be almost point-like; the space of explored states therefore diminishes rapidly. Balancing the two has led to a huge number of advanced techniques and research in the area, but due to the expanding scope of this thesis only simple fixes were investigated.

A frequently applied simple technique to balance between sample degeneracy and impoverishment is to only resample the particles if sample variance is high-enough. However, for this thesis this is a (mostly) unsuitable form of gating. The reason for this is due to the 'island cluster' form the observation PDF can take, with the extreme example seen in Figure (9.2). While the calculated variance of the samples might be quite high, in reality the particles could be in a few (e.g. up to a dozen) particle 'clusters', effectively leading to high-variance sample impoverishment.

While there are more complicated algorithms to combat high-variance sample impoverishment with more advanced algorithms [Li et al., 2013], a much simpler solution was also effective; in addition to a variance-threshold, when resampling only a randomly selected subset of particles were resampled. Despite initial experiments on the number of resampled particles varying depending on various quantities such as sample-time, only small performance gains were found despite large increases in algorithm complexity.

The other issue faced with particle weights is that of the observation PDF occasionally having high precision but low accuracy. This would throw the weight-based estimation step off along with increasing the number of particles in a less-useful area, leading to induced temporary sample degeneracy.

A simple empirically substantiated fix to weight 'jumpiness' is to use an exponential moving-average on the weights, shown in Equation (9.6) where $\alpha \in]0,1]$ controls the behavior of the EMA filter. Not much has been found in literature on applying an EMA filter to the weights, but it works for this thesis.

$$W_{k|k}^{1:N_p} = \alpha \left\{ \frac{1}{c_k} w_{k|k-1}^{i} p\left(y_k | x_k^i\right) \right\}^{1:N_p} + (1-\alpha) W_{k-1|k-1}^{1:N_p} \tag{9.6}$$

Resulting Algorithm

With all the steps taken and adapting the methods from Chapter 4, we can create the final particle filtering algorithm, as shown in Algorithm 4. The parameters utilized for the EMA was $\alpha = 0.5$, $N = 4000$, and $R_{num} = 1200$. To save CPU time the particle filter was not tested on the training data results.

Algorithm 4: The implementation of a particle filter applied to positioning

Result: time-vector of position estimates $\hat{y}_{0:K}$
Input: Particle count N_p
Input: ML predictors $M^{1:M}()$
Input: KDE bandwidth h
Input: number to resample R_{num}, variance threshold var_{min}
Input: State dynamics $sim()$
Input: Stratified Resampling $StratR()$
Data: Channel matrices array $\mathbf{H}_{1:K}$, sample time vector $\Delta t_{1:K}$
Initialize: First position estimate: $\hat{y}_0 \leftarrow mean(M^{1:M}(H_0))$;
Initialize: particles: $P^{1:N_p} := \left\{P^i\right\}_{i=0}^{N_p} \leftarrow initialize(\hat{y}_0)$;
Initialize: weights: $W^{1:N_p} \leftarrow \frac{1}{N_p}$;
for k = *1 to M* **do**
  Simulate particle motion with noise: $P_k \leftarrow sim\left(P_{k-1}, \Delta t_k\right)$;
  Create position ML output vector: $\tilde{y}^{1:M} \leftarrow M^{1:M}(H_k)$;
  Create measurement model using KDE: $meas_k \leftarrow KDE_h\left(\tilde{y}^{1:M}\right)$;
  Update temporary weights: $\tilde{w} \leftarrow meas_k\left(y_k, P_k\right)$;
  Normalize temporary weights: $\tilde{w}^i \leftarrow \frac{w^i}{\sum_i w^i}$;
  Update weights: $W^{1:N_p} \leftarrow \alpha\tilde{w}^{1:N_p} + (1-\alpha)W^{1:N_p}$;
  Obtain position estimate: $\hat{y}_k \leftarrow \frac{1}{N_p}\sum_{i=0}^{N_p} w_k^i (P^i)_{EN}$;
  if $variance(P_k) > var_{min}$ **then**
    Get resample sub-indeces: $I_{rs}^{1:R_{num}} \leftarrow argrand(W^{1:N_p})^{1:R_{num}}$;
    Get sub-weights: $\overline{W}^{1:R_{num}} \leftarrow W^{1:N_p}[I^{1:R_{num}}]$;
    Normalize sub-weights: $\overline{W}^i \leftarrow \frac{\overline{W}^i}{\sum_i \overline{W}^i}$;
    Get Resample indeces: $I_{Strat} \leftarrow StratR\left(\overline{W}^i\right)$;
    Resample particles: $P_k[I^{1:R_{num}}] \leftarrow P_k[I^{1:R_{num}}][I_{Strat}]$. ;
    Update weights: $W^{1:N_p}[I^{1:R_{num}}] \leftarrow W^{1:N_p}[I^{1:R_{num}}][I_{Strat}]$;
    Normalize updated weights:$W^i \leftarrow \frac{W^i}{\sum_i W^i}$;
  end
end

9.4 Final Results and Discussion

Applying and tuning the algorithm above on the observations created by the 50-Diverse ML model ensemble specified in Section 8.4, we get the MSE results compared to the 'ground truth' GNSS as shown in Table 9.1.

Dataset	Input	Model	*val.MSE*	*test.MSE*
LoS-D	$\mathbf{X}_H^{FA}$	50-Diverse Particle Filter	3.3	4.9
LoS-A	$\mathbf{X}_H^{FA}$	50-Diverse Particle Filter	22.6	25.1
NLoS-A	$\mathbf{X}_H^{FA}$	50-Diverse Particle Filter	27.3	20.4
LoS-D	$\mathbf{X}_H^{mUE}$	50-Diverse Particle Filter	3.0	4.5
LoS-A	$\mathbf{X}_H^{mUE}$	50-Diverse Particle Filter	24.0	28.4
NLoS-A	$\mathbf{X}_H^{mUE}$	50-Diverse Particle Filter	27.1	20.4

Table 9.1 Results on the 50-element ensemble deep-learning models with $N = 4000$ particles, weight EMA $\alpha = 0.5$, and $R_{num} = 1200$ resamples per iteration. Training MSE is left out as scores were good enough as-is. For the motion model σ_α^2 was set as 1.1 $[\frac{rad^2}{s}]$ while σ_v^2 was set to 0.5 $[\frac{m^2}{s^3}]$. These values were found through experimentation on the validation datasets.

To ensure that the particles are approximating physical behavior, the easiest check is to examine weighted mean particle velocities over the entire dataset. Pedestrian walking speed vary between 0.8-1.5 [m/s], which means that a well-calibrated particle filter should find out about this 'hidden' state. Indeed, the results reflect this, as shown in the Figure (9.4) for particles in the the LoS test dataset.

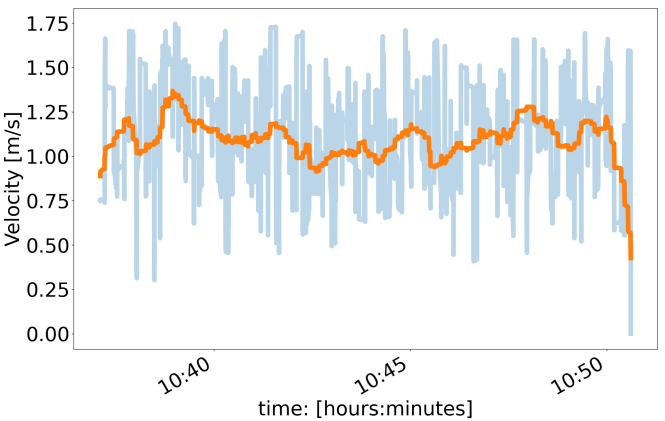


Figure 9.4 Mean particle velocities in the particle filter for the LoS test dataset. The semi-transparent blue line is the actual mean velocity for each sample, while the orange line shows the smoothed velocities.

nLoS particle filter results discussion

The results for the NLoS scenario show even with a numerical comparison to the flawed GNSS data shows a huge improvement over all prior results. The visual result is also striking, with e.g. even the difficult NLoS-A test data having clear, accurate, and time-coherent position tracking, as per Figure (9.5).

Even for the connection interruptions in the test data, the scenario is handled adequately, as after dropping connetion for 20 seconds the Particle Filter is reset, and convergence w.r.t. position is rapidly achieved. See Appendix Figure (11.5) for a visual demonstration of this.

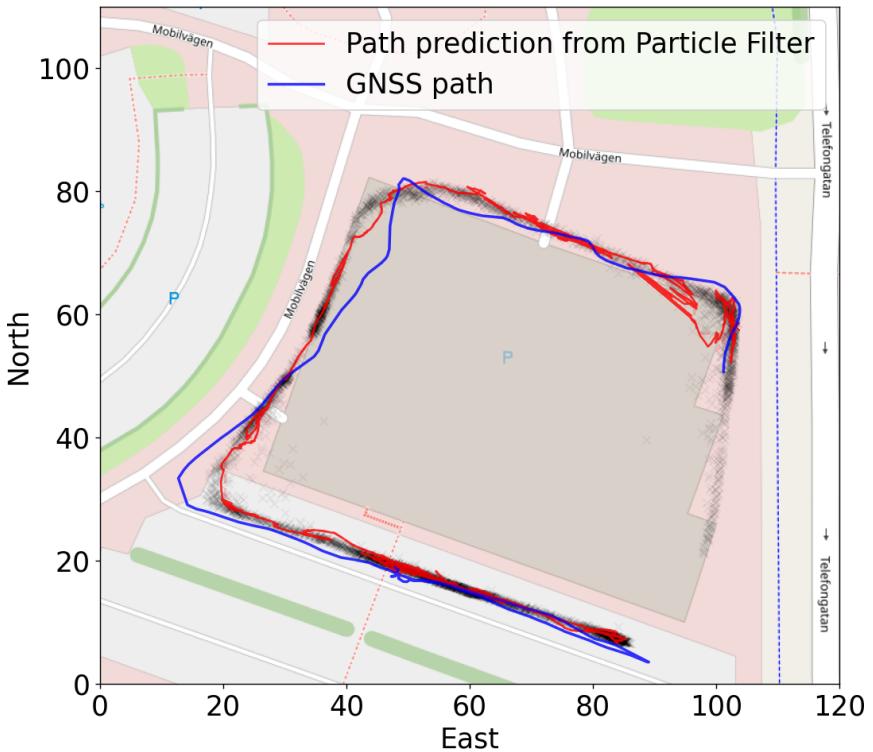


Figure 9.5 A part of the output of feeding the particle filter algorithm from Algorithm 4 the Diverse-50 ensemble outputs on the $\mathbf{X}_H^{FA}$ NLoS-A test. The small block smudges show the ML prediction density. The ability of the Particle Filter to generally discern the correct position is at times better than that of GNSS, even if the individual ML predictions are scattered.

Within the NLoS some locations still have major issues, e.g. the north-east corner of the garage under foliage. The expectation here is that the issue is apparent

when there is no good open-space for NLoS reflections off reflective surfaces to reach the UE.

With a visual investigation of some data in the northern region corner, we can use the same KDE-visual as in Figure (9.2) of some points by looking at Figure (9.6) of an example where the ML ensemble reports a point in the North to be at the North-East corner instead. However, note that even under foliage sometimes accurate guesses can be made, as in Figure (9.7). See also the Appendix Figures (), which each show different NLoS guess-scatters.

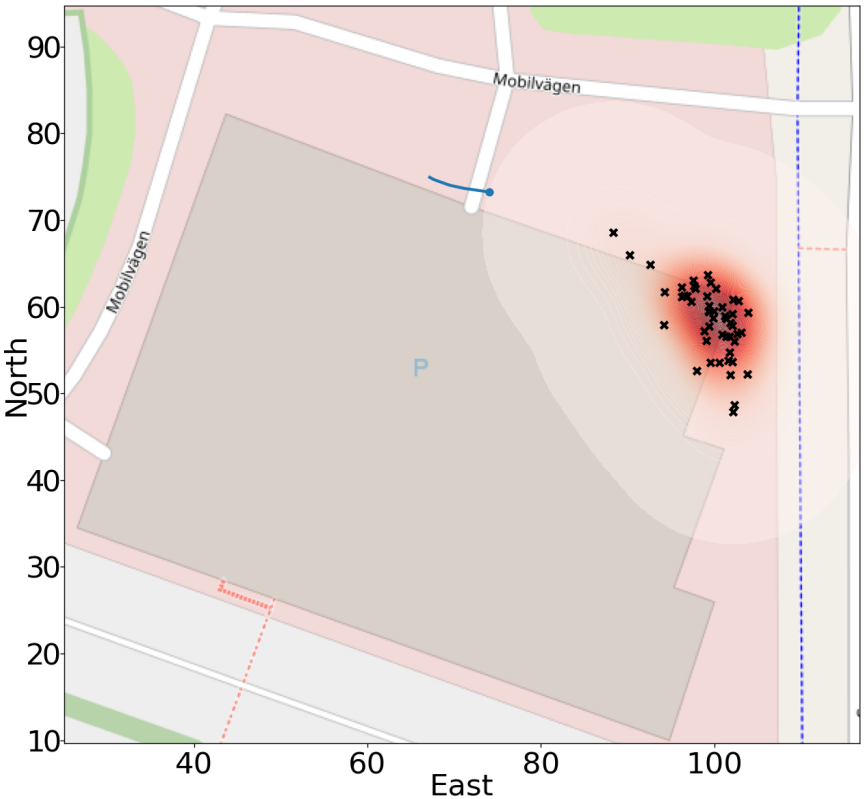


Figure 9.6 The resulting PDF contour-plot fitted using a KDE with a bandwidth of 3 on 50 observations done by 50 different models on a single $\mathbf{X}_H^{FA}$ channel matrix datapoint in the LoS-A2 test sub-dataset. The 'ground-truth' GNSS-interpolated location is shown as a blue point, with the past few seconds of movement shown by the blue path. The figure shows how the ML algorithm sometimes has issues positioning when in NLoS and under foliage, with few easy reflections. It seems to 'guess' with high precision but low accuracy in different spots.

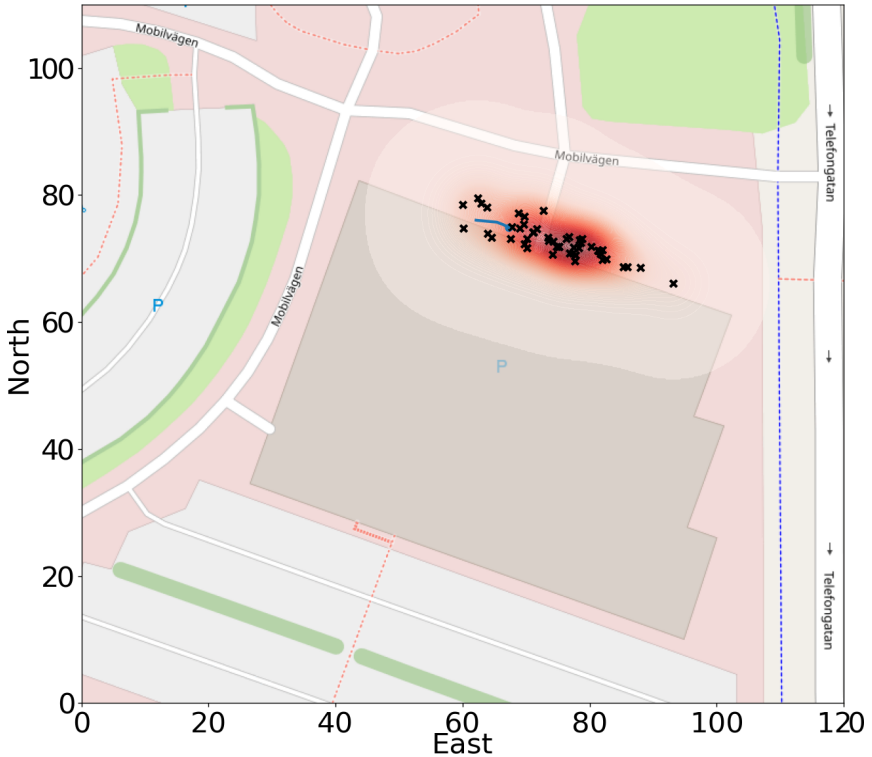


Figure 9.7 The resulting PDF contour-plot fitted using a KDE with a bandwidth of 3 on 50 observations done by 50 different models on a single $\mathbf{X}_H^{FA}$ channel matrix datapoint in the LoS-A2 test sub-dataset. The 'ground-truth' GNSS-interpolated location is shown as a blue point, with the past few seconds of movement shown by the blue path. The figure shows how in NLoS and under foliage with few easy reflections sometimes a good guess can be made, so long as there might be some amount of free-space in an area.

To further speculate on the accuracy characteristics on the NLoS results, and to give Figure (9.7) more context, there is a brief section roughly where the points are clustered in the center of the northern garage with a potential free-space propagation pathway that could allow for more NLoS direct reflections. Overall, more investigation of NLoS positioning behavior in real-world scenarios is required to support any conclusions made w.r.t. the results shown above - a topic for future research.

Basic functionality in terms of finding position and trajectory in NLoS scenarios in near-real-time on datasets recorded at different times using the combination of methods and data processing described in thesis has been demonstrated and hold a great deal of promise for future refinement and analysis.

LoS-A particle filter results discussion

Using the LoS-A test dataset, despite its poor MSE behavior for model-averaging and single-model results, the particle-filtering accuracy is at the point where the GNSS data inaccuracy as compared to 'true' position is the most significant contributing factor for the MSE - especially on the north side of the LoS-A data, where the GNSS drift is most egregious. The average MSE in the northern section is over 40, which does not accurately reflect the true error - see Figure (9.8).

Furthermore, while the scattered nature of the data still causes the particle filter to oscillate in certain regions on the LoS-A dataset, this is significantly diminished even compared to e.g. median smoothing - despite the latter using information from future data. Overall, despite the difference in data character from the training and the validation and test set, the positioning pipeline can obtain a reasonable position estimate. This indicates towards the positioning pipeline being viable beyond controlled experimental conditions.

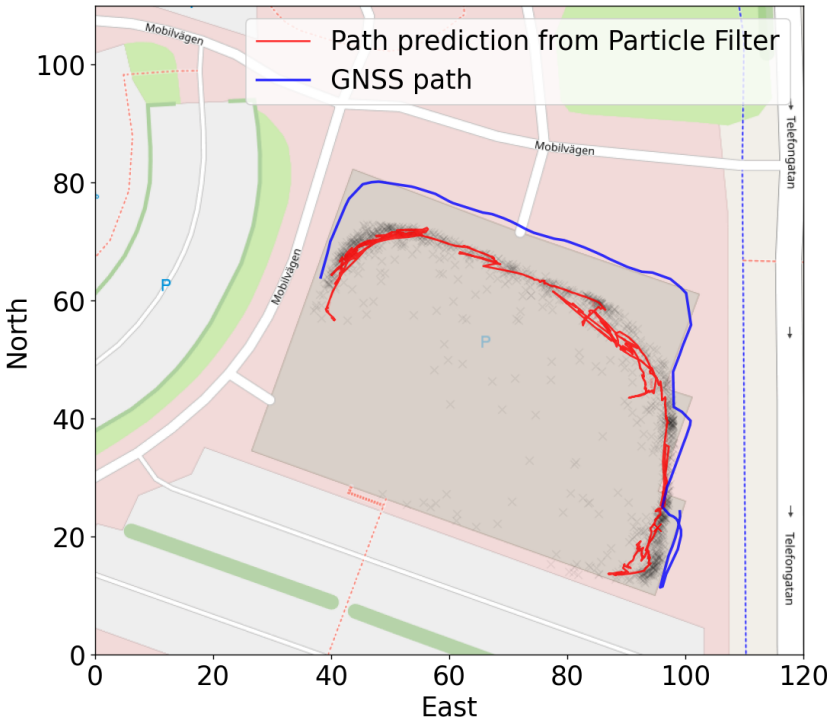


Figure 9.8 A part of the output of feeding the particle filter algorithm from Algorithm 4 the Diverse-50 ensemble outputs on the $\mathbf{X}_H^{FA}$ LoS-A test dataset. The predicted path for the particle filter is better than that of the GNSS in some parts, especially in the northern sections of the LoS-A test dataset.

LoS-D particle filter results discussion

For the LoS-D dataset, the MSE is so low at 4-5 m^2 that most of the error originates from GNSS being used. It represents the sort of 'perfect' conditions to test the positioning pipeline, in which both the strong LoS component and other MPC can be used to reconstruct position without much dataset domain shift interfering with results. Therefore, to get a 'true' ideal-scenario performance, a more accurate base-truth position must be used.

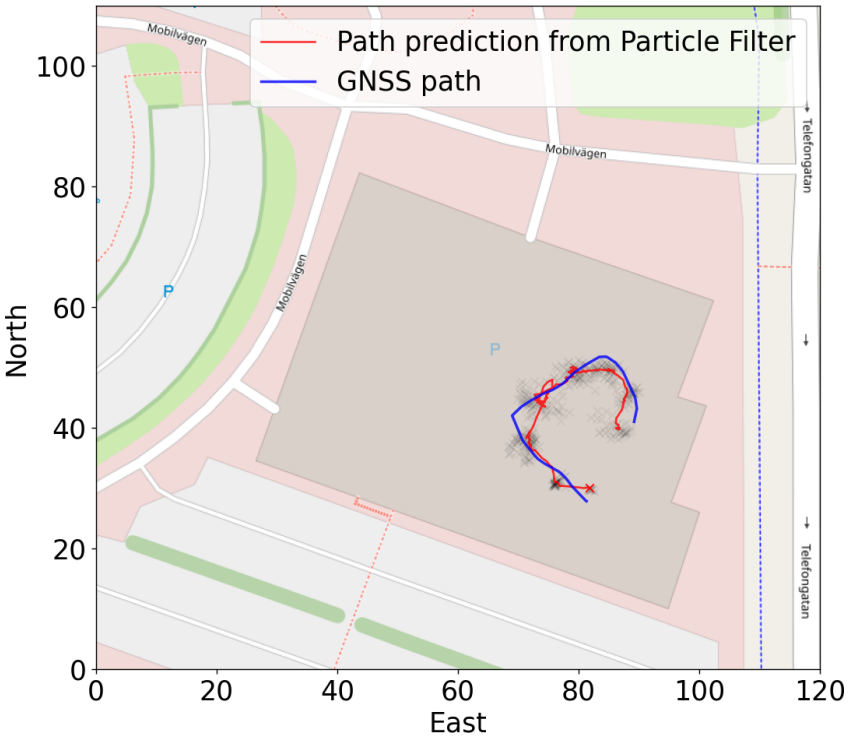


Figure 9.9 A part of the output of feeding the particle filter algorithm from Algorithm 4 the Diverse-50 ensemble outputs on the $\mathbf{X}_H^{FA}$ LoS-D test dataset. The predicted path for the particle filter is better than that of the GNSS in some parts, especially in the northern sections of the LoS-A test dataset.

In summary, the particle filter works well to take the scattered position outputs of an ensemble of ML network, and then generate proper and physically *mostly* feasible position trajectories - thereby more than halving the MSE of the overall pipeline compared to simply taking the mean of the ensemble. The true error is likely much lower, with much of the remaining error originating from the characteristics of the GNSS data.

10

Conclusions

10.1 Summary

The contents of this thesis document the theory, methods and the measurement results of a functioning positioning system using SRS matrices from CSI data using a real-world 5G basestation and collected data. All significant steps in the development process necessary to reconstruct the results were documented, in the order listed below:

1. Necessary theory and basic underlying methodology in Chapters 2-5
2. The Measurement process to obtain raw logs in Chapter 6,
3. Processing the raw logs into the python environment in Chapter 6,
4. Pre-processing the data in the python environment in Chapter 7,
5. Pre-ML Data analysis in Chapter 7,
6. Viability analysis of data with classical ML in Section 7.5,
7. Architecture, methods and considerations with fully-connected deep learning on both single and ensemble networks in Chapter 8
8. Applying particle filtering using basic KDE-based PDF estimation with Ensemble network outputs in Chapter 9.

To summarize the findings of this thesis work, the detailed process pipeline obtained an approximate root-MSE around 2-6 meters as compared to the GNSS data when evaluated on test data that had been set aside, with accuracy depending mostly on data conditions. These results comparable favorably to results found in literature on most outdoors (and sometimes indoors) positioning systems using similar quality and density real-world data - especially notable once the inaccuracies caused by poor-quality GNSS position are considered. The accuracy on the NLoS data in particular indicates the viability of this method for positioning in a real-world system.

10.2 Contributions and result comparison to prior work

Contributions to the field and Ericsson through this thesis are in four main areas:

The first contribution is mostly to work done at Ericsson, and it was the development of a high-perfomance and high-data-throughput log-to-python channel estimate extraction package, written in C++. It could read log files into usable numpy arrays at around 300-400 mb/s, and will prove useful for future work with the 5G testbed in Ericsson, Lund.

The second contribution to Ericsson is an in-depth analysis on the behavior of the beamformed channel data obtained from the 5G testbed. Furthermore, analysis on the viability of using amplitude for outdoors positioning and how much prediction results improve when taking energy-falloff with distance into consideration improves results when using a data-based approach.

The third contribution is through utilizing a deeper neural network on beamformed channel matrix data in a practical scenario on collected data, also including a large amount of regularization techniques that have not yet been seen in applications within this area.

The final and most novel approach was the combination of ensemble deep networks with KDE and a kinematic pedestrian model to apply tuned particle filtering on deep-learning position outputs. This technique has not been seen elsewhere in literature, and can be considered the most novel contribution to the field from this thesis. The results improved dramatically from utilizing this model, demonstrating the viability of it in future research.

Overall, this thesis has demonstrated the practical viability of positioning well under 10 meters of mean error in an outdoors NLoS setting, even when datasets are taken weeks apart and have significant behavioral differences. A conclusion to draw is that including physical information about kinematics when tracking an object is crucial, as is attempting to linearize the relation between distance and beam-amplitude when using beamforming channel matrices.

Comparing results

Taking only the two papers [Malmström et al., 2019] and [Decurninge et al., 2018] that have comparable problem domains into account, we can compare it with the final test result of this thesis - that being around 4.5 meters mean error for the NLoS and around 2.3-4.4 meters for the LoS scenario, with the 'real' accuracies being even better than that due to GNSS inaccuracies adding up to a few meters of error to the results.

Comparing it to either of the two papers with 7 and 8 meters accuracy in NLoS/semi-nLoS scenarios respectively, and the improvements are dramatic, especially for NLoS. This is even despite the paper [Malmström et al., 2019] written on the Ericsson testbench having been recorded on a fixed car with a better GNSS with better sample-rate and accuracy, giving it an inherent advantage in terms of data repeatability.

10.3 Future work

Data collection and feature preprocessing

Beyond just collecting more data, obtaining data of different character to get more generalizable results would be an important step. E.g. collecting an NLoS equivalent of the LoS-D dataset, collecting a *proper* LoS-A dataset, comparing channel matrices of different connected UEs, getting higher accuracy GNSS data, etc. The GNSS data accuracy in particular proved to be the greatest limiting factor, as extensively discussed in Chapter 9.

The data 'grouping' observed in Chapter 7 could most likely be solved by using an alternative to Youtube 4k streaming to load the channel, e.g. video calling. Alternatively, implementing an algorithm or app to load the channel with synthetic data at a constant rate would work as well. Another solution to getting more regular data would be to utilize logging that extracts the channel matrices when they are updated, not when they are used.

Feature preprocessing would also be an area in which a lot of possible 'easy' improvements lie. In this thesis, no serious attempt at noise characterization or outlier filtering was attempted on the channel matrices $\mathbf{H}$, which alone could improve ML results and give more information on the noise for particle filtering. Furthermore, the used fourth-power statistical approximation for received energy is an extremely simple approximation, and much more advanced versions exist. Using a sort of 'intelligent system' to find which statistical approximation is appropriate for transforming the channel matrices before ML positioning models are applied would be a significant advancement beyond just 'dumping' the square root and fourth-root into a positioning system.

Furthermore, just 'discarding' phase due to the poor accuracy of the GNSS data was a necessary step to prevent scope-creep in this thesis, but it is also a possible avenue to extract more features from. Finally, other CSI metrics could be utilized in addition to the SRS channel estimate, even if they are derived quantites from it. An example of this could be signal delays of signals arriving from each direction.

Another potential avenue for development would be to pre-train networks on simulated data before finalizing training on real-world data, thereby enabling them to learn physical models in a more 'friendly' environment before adjusting the resulting model to the lower-quality real-world environment.

One step that caused significant difficulties was actually assessing the final performance of the algorithms. Even if a 'real' system would have to utilize low-quality GNSS positioning data for training ML models on every basestation, for system development comparing model performance is a critical step in the research pipeline. However, comparing MSE offset of the prediction to the GNSS position lead to unrealistically pessimistic MSE performance ratings, which meant that the 'true' model performance is still unkown.

Further Developing ML models

On the ML side, this thesis was written without access to significant compute resources. For that reason, only a narrow range of relatively shallow deep-learning models and hyperparameters could be tested. Just tuning all the hyperparameters of the fully-connected model could contribute a great deal, let alone building more complex networks with more advanced architectures.

Furthermore, other models such as complex networks and Convolutional Neural Networks were thoroughly tested during the development of this thesis, though not documented due to similar or worse results and an already too-expanded thesis scope. However, other papers have utilized these methods - often on high-resolution simulated data - and obtained improved results. This presents an avenue of improvement for future systems, especially ones with more available data features or data of higher quality.

Training generative models (generating channel matrices from position) could also give a great deal of insight into the data that is currently not well known. In general, as mentioned in the prior subsection, more proper analysis of the data is needed before any further development can be done, as how exactly current ML systems determine position is opaque.

An additional area of research could be to use multi-stage ML and statistical models, where e.g. certain features are extracted such as whether there is currently LoS/nLoS, which direction motion is occuring, velocity and direction of motion for the user, etc. These could then give additional information about states, which then could be combined in various ways to produce more advanced models while reducing the possibility of overfitting.

Further Developing Particle Filtering

The utilized simple particle filter algorithm proved to improve results on its own, and opens the door to further improvements in this area down-the-line.

One area that could be improved is the pedestrian state behavior. This can e.g. include more accurate estimation of possible user movement states to constrain particle behavior, modelling of pedestrian walking behavior, using longer-term movement trends to smooth trajectory, etc.

Another potential improvement area is the algorithm itself. A few improvements of this form include using more advanced intelligent resampling techniques, applying quasi-random Monte Carlo principles to improve filtering.

Enhancing applicability

Though the investigated methods to produce position using channel matrices have been done with practical application as a goal, of course the developed system as a proof-of-concept does not take it all the way to the 'end-user'. Two distinct areas of improvement have been thought to be necessary before this could be attempted.

The first is to reduce the amount of labeled data needed. This can be done by e.g. using principles of semi-supervised learning, such as pre-training parts of the deep-learning models on CSI data that has no labeled position. For this, a different approach is needed than what is shown in this thesis, such as e.g. in the unsupervised training section turning the systems into auto-encoders.

The other area of improvement thought of but not implemented is to use a scale-invariant positioning system. Within a cell, have different 'nested sets' of ML algorithms be trained over larger areas with less accuracy and precision, but with more generalizability - then have ML algorithms on ever smaller areas with greater accuracy and precision. The end result is that the 'coarse' algorithms allocate positioning to the 'fine' algorithms, enabling fine-position coverage across an entire cell without needing a single 'universal' ML model.

10.4 Closing words on limitations and setbacks

In this thesis the intended proof of concept pipeline for positioning in a practical real-world 5G environment was successfully built, with results especially in the NLoS scenario that were significantly better than initially hoped for.

Though the thesis could be said to have been successful at its stated goal, and though the results of this thesis are highly competitive with other results found in literature, many practical considerations caused significant setbacks to obtaining further improved results.

As an example of a significant setback, getting data out of the basestation itself took most of the allocated time for the thesis, with the first 'good' measurements only occurring a month and a half before the submission deadline. A significant cause of this is the sheer volume of data and the poor logging format built for the basestation, necessitating the use of a custom-built regex parser in C++, taking 2 months alone to develop. Furthermore, the probe used to obtain the SRS channel matrices occasionally failed for no discernible reason, which could only be noticed after the completion of the measurement. Only around half of the time spent measuring actually produced usable data for this reason.

Even the data produced was an inferior reflection of the actual channel matrix data available within the basestation, as only three of the possible 137 frequency bands available for the specific basestation model were available - and logs were outputted only when the transmission was used, instead of when the SRS channel matrices were internally updated, thereby leading to the data clumping behavior observed in Chapter 7.

Another significant setback was the machine used for model training and data processing. Memory limitations on more advance processing techniques often hindered development, and considering the limited timescale to obtain results on obtained logs, a proper hyperparameter and architecture search became simply unfeasible using the designated machine with the time remaining.

Finally, GNSS inaccuracies proved to be a significant limitation when rating the true performance of the models. In particular, comparing model performance to models trained and tested on simulated data is simply not possible with any degree of confidence. While GNSS data is superior for testing model viability in the real-world, at the very least test-data for model development should be obtained with precise measurement tools that were unavailable for this thesis.

Bibliography

Apelfrojd, R. (2018). *Channel Estimation and Prediction for 5G Applications*. PhD thesis. Uppsala University.

Barbeau, S. (2022). *Gpstest v3.9.16*. https://github.com/barbeau/gpstest.

Bast, S. D., A. P. Guevara, and S. Pollin (2020). "CSI-based positioning in massive MIMO systems using convolutional neural networks". In: *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pp. 1–5. DOI: 10.1109/VTC2020-Spring48590.2020.9129126.

Björnson, E., L. Sanguinetti, H. Wymeersch, J. Hoydis, and T. L. Marzetta (2019). "Massive MIMO is a reality—what is next?: five promising research directions for antenna arrays". *Digital Signal Processing* **94**. Special Issue on Source Localization in Massive MIMO, pp. 3–20. ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2019.06.007. URL: https://www.sciencedirect.com/science/article/pii/S1051200419300776.

Burghal, D., A. T. Ravi, V. Rao, A. A. Alghafis, and A. F. Molisch (2020a). *A comprehensive survey of machine learning based localization with wireless signals*. arXiv: 2012.11171 [eess.SY].

Burghal, D., A. T. Ravi, V. Rao, A. A. Alghafis, and A. F. Molisch (2020b). *A comprehensive survey of machine learning based localization with wireless signals*. DOI: 10.48550/ARXIV.2012.11171. URL: https://arxiv.org/abs/2012.11171.

Chen, Y.-C. (2017). *A tutorial on kernel density estimation and recent advances*. DOI: 10.48550/ARXIV.1704.03924. URL: https://arxiv.org/abs/1704.03924.

Choi, D., C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl (2019). "On empirical comparisons of optimizers for deep learning". *ArXiv* **abs/1910.05446**.

Dahlman, E., S. Parkvall, and J. Skold (2018). *5G NR: The Next Generation Wireless Access Technology*. Elsevier Science. ISBN: 9780128143247. URL: https://books.google.se/books?id=C5poDwAAQBAJ.

Davidson, P. and R. Piché (2016). "A survey of selected indoor positioning methods for smartphones". *IEEE Communications Surveys & Tutorials* **19**:2, pp. 1347–1370.

Decurninge, A., L. G. Ordóñez, P. Ferrand, H. Gaoning, L. Bojie, Z. Wei, and M. Guillaud (2018). "CSI-based outdoor localization for massive MIMO: experiments with a learning approach". In: *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, pp. 1–6. DOI: 10.1109/ISWCS.2018.8491210.

Drake, S. (2002). *Converting GPS Coordinates (phil lambdal h) to Navigation Coordination (ENU)*. Tech. rep. DEFENCE SCIENCE and TECHNOLOGY ORGANISATION SALISBURY. URL: https://www.researchgate.net/profile/Samuel-Drake/publication/27253833_Converting_GPS_coordinates_phi_lambda_h_to_navigation_coordinates_ENU/links/00b7d516ca79c24fdb000000/Converting-GPS-coordinates-phi-lambda-h-to-navigation-coordinates-ENU.pdf.

Gante, J., G. Falcão, and L. Sousa (2018). *Beamformed fingerprint learning for accurate millimeter wave positioning*. arXiv: 1804.04112 [eess.SP].

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems - 2nd edition*. " O'Reilly Media, Inc."

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Groves, P. (2013). *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems, Second Edition*.

Guerra, A., F. Guidi, and D. Dardari (2018). "Single-anchor localization and orientation performance limits using massive arrays: MIMO vs. beamforming". *IEEE Transactions on Wireless Communications* **17**:8, pp. 5241–5255.

Guo, C., J. Yu, W.-F. Guo, Y. Deng, and J.-N. Liu (2020). "Intelligent and ubiquitous positioning framework in 5G edge computing scenarios". *IEEE Access* **8**, pp. 83276–83289. DOI: 10.1109/ACCESS.2020.2990639.

Gustafsson, F. (2010). "Particle filter theory and practice with positioning applications". *Aerospace and Electronic Systems Magazine, IEEE* **25**, pp. 53–82. DOI: 10.1109/MAES.2010.5546308.

Hampton, J. R. (2013). *Introduction to MIMO Communications*. Cambridge University Press. DOI: 10.1017/CBO9781107337527.

Hoefler, T., D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste (2021). "Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks." *J. Mach. Learn. Res.* **22**:241, pp. 1–124.

Huang, C., R. He, B. Ai, A. F. Molisch, B. K. Lau, K. Haneda, B. Liu, C.-X. Wang, M. Yang, C. Oestges, and Z. Zhong (2021a). "Artificial intelligence enabled radio propagation for communications-part i: channel characterization and antenna-channel optimization". In: arXiv: 2111.12227 [eess.SP].

Huang, C., R. He, B. Ai, A. F. Molisch, B. K. Lau, K. Haneda, B. Liu, C. Wang, M. Yang, C. Oestges, and Z. Zhong (2021b). "Artificial intelligence enabled radio propagation for communications-part ii: scenario identification and channel modeling". In: arXiv: 2111.12227 [eess.SP].

Jia, J., T. S. Durrani, and J. Chen (2018). "The innovation waves in mobile telecommunication industry". *IEEE Engineering Management Review* **46**:3, pp. 63–74. DOI: 10.1109/EMR.2018.2863253.

Julier, S. and J. Uhlmann (2004). "Unscented filtering and nonlinear estimation". *Proceedings of the IEEE* **92**:3, pp. 401–422. DOI: 10.1109/JPROC.2003.823141.

Kingma, D. P. and J. Ba (2014). *Adam: a method for stochastic optimization*. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

Kitagawa, G. (1996). "Monte carlo filter and smoother for non-gaussian nonlinear state space models". *Journal of computational and graphical statistics* **5**:1, pp. 1–25.

Koivisto, M., A. Hakkarainen, M. Costa, P. Kela, K. Leppanen, and M. Valkama (2017). "High-efficiency device positioning and location-aware communications in dense 5G networks". *IEEE Communications Magazine* **55**:8, pp. 188–195. DOI: 10.1109/MCOM.2017.1600655.

Li, T., S. Sun, T. P. Sattar, and J. M. Corchado (2013). "Fighting against sample degeneracy and impoverishment in particle filters: particularly on intelligent choices". *CoRR* **abs/1308.2443**. arXiv: 1308.2443. URL: http://arxiv.org/abs/1308.2443.

Li, X., X. Cai, Y. Hei, and R. Yuan (2017). "NLOS identification and mitigation based on channel state information for indoor wifi localisation". *IET Communications* **11**:4, pp. 531–537. DOI: https://doi.org/10.1049/iet-com.2016.0562. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-com.2016.0562. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-com.2016.0562.

Loshchilov, I. and F. Hutter (2017). *Decoupled weight decay regularization*. DOI: 10.48550/ARXIV.1711.05101. URL: https://arxiv.org/abs/1711.05101.

Malmström, M., I. Skog, S. M. Razavi, Y. Zhao, and F. Gunnarsson (2019). "5g positioning - a machine learning approach". In: *2019 16th Workshop on Positioning, Navigation and Communications (WPNC)*, pp. 1–6. DOI: 10.1109/WPNC47567.2019.8970186.

Mitchell, T. M. (1997). *Machine learning*. Vol. 1. 9. McGraw-hill New York.

Mogyorósi, F., P. Revisnyei, A. Pašić, Z. Papp, I. Törös, P. Varga, and A. Pašić (2022). "Positioning in 5G and 6G networks; a survey". *Sensors* **22**:13. ISSN: 1424-8220. DOI: 10.3390/s22134757. URL: https://www.mdpi.com/1424-8220/22/13/4757.

Molisch, A. (2010). *Wireless Communications*. Wiley - IEEE. Wiley. ISBN: 9780470741863. URL: https://books.google.se/books?id=nwa4wgEACAAJ.

Omariba, Z. and N. Masese (2019). "A study on path loss and shadowing for wireless communication channels". **8**, pp. 1–10.

Papoulis, A. (1984). *Probability, Random Variables, and Stochastic Processes*. Communications and information theory. McGraw-Hill. ISBN: 9780070484689. URL: https://books.google.se/books?id=3NRQAAAAMAAJ.

Reddi, S. J., S. Kale, and S. Kumar (2018). "On the convergence of adam and beyond". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=ryQu7f-RZ.

Robbins, H. and S. Monro (1951). "A stochastic approximation method". *The Annals of Mathematical Statistics* **22**:3, pp. 400–407. ISSN: 00034851. URL: http://www.jstor.org/stable/2236626 (visited on 2022-08-16).

Santos, G. L., P. T. Endo, D. Sadok, and J. Kelner (2020). "When 5G meets deep learning: a systematic review". *Algorithms* **13**:9. ISSN: 1999-4893. DOI: 10.3390/a13090208. URL: https://www.mdpi.com/1999-4893/13/9/208.

Sonoda, S. and N. Murata (2017). "Neural network with unbounded activation functions is universal approximator". *Applied and Computational Harmonic Analysis* **43**:2, pp. 233–268. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2015.12.005. URL: https://www.sciencedirect.com/science/article/pii/S1063520315001748.

Traboulsi, S. (2022). "Overview of 5G-oriented positioning technology in smart cities". *Procedia Computer Science* **201**. The 13th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 5th International Conference on Emerging Data and Industry 4.0 (EDI40), pp. 368–374. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2022.03.049. URL: https://www.sciencedirect.com/science/article/pii/S1877050922004628.

Tse, D. and P. Viswanath (2005). *Fundamentals of wireless communication*. Cambridge university press.

Wen, F., H. Wymeersch, B. Peng, W. P. Tay, H. C. So, and D. Yang (2019). "A survey on 5G massive MIMO localization". *Digital Signal Processing* **94**. Special Issue on Source Localization in Massive MIMO, pp. 21–28. ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2019.05.005. URL: https://www.sciencedirect.com/science/article/pii/S1051200419300569.

Widmaier, M., M. Arnold, S. Dorner, S. Cammerer, and S. ten Brink (2019). "Towards practical indoor positioning based on massive MIMO systems". In: *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pp. 1–6. DOI: 10.1109/VTCFall.2019.8891273.

Yang, H., W.-D. Zhong, C. Chen, and A. Alphones (2020). "Integration of visible light communication and positioning within 5G networks for internet of things". *IEEE Network* **34**:5, pp. 134–140.

Zhu, N., J. Marais, D. Bétaille, and M. Berbineau (2018). "GNSS position integrity in urban environments: a review of literature". *IEEE Transactions on Intelligent Transportation Systems* **19**:9, pp. 2762–2778. DOI: 10.1109/TITS.2017.2766768.

11

Appendix

Taylor expansion of Equation (11.2) about $\Phi_g \to \Phi_g + d\Phi_g$, $\lambda \to \lambda + d\lambda$, and $h \to h + dh$. Ignoring terms of $O(d\Phi_g^3)$, $O(d\lambda^3)$, $O(dhd\Phi_g^2)$, $O(dhd\lambda^2)$ and higher.

Citation: [Drake, 2002], Equation (2).

$$
\begin{aligned}
dx = &\left(\frac{R_o \cos\lambda \sin\Phi_g (1-e^2)}{\chi^3} - h\cos\lambda\sin\Phi_g\right) d\Phi_g \\
&- \left(\frac{R_o \sin\lambda\cos\Phi_g}{\chi} + h\sin\lambda\cos\Phi_g\right) d\lambda + \cos\Phi_g\cos\lambda dh \\
&+ \left(\frac{1}{4} R_o \cos\Phi_g \cos\lambda \left(-2 - 7e^2 + 9e^2\cos^2\Phi_g\right) - \frac{1}{2} h\cos\lambda\cos\Phi_g\right) d\Phi_g^2 \\
&+ \left(\frac{R_o \sin\lambda \sin\Phi_g (1-e^2)}{\chi^3} + h\sin\lambda\sin\Phi_g\right) d\lambda d\Phi_g - \cos\lambda\sin\Phi_g dhd\Phi_g \\
&- \left(\frac{R_o \cos\lambda\cos\Phi_g}{2\chi} + \frac{h}{2}\cos\lambda\cos\Phi_g\right) d\lambda^2 - \sin\lambda\cos\Phi_g dhd\lambda \\
dy = &\left(\frac{R_o \sin\lambda \sin\Phi_g (1-e^2)}{\chi^3} - h\sin\lambda\sin\Phi_g\right) d\Phi_g \\
&+ \left(\frac{R_o \cos\lambda\cos\Phi_g}{\chi} + h\cos\lambda\cos\Phi_g\right) d\lambda + \sin\Phi_g\cos\lambda dh \\
&+ \left(\frac{1}{4} R_o \cos\Phi_g \sin\lambda \left(-2 - 7e^2 + 9e^2\cos^2\Phi_g\right) - \frac{1}{2} h\sin\lambda\cos\Phi_g\right) d\Phi_g^2 \\
&- \left(\frac{R_o \cos\lambda \sin\Phi_g (1-e^2)}{\chi^3} + h\cos\lambda\sin\Phi_g\right) d\lambda d\Phi_g - \sin\lambda\sin\Phi_g dhd\Phi_g \\
&- \left(\frac{R_o \sin\lambda\cos\Phi_g}{2\chi} + \frac{h}{2}\sin\lambda\cos\Phi_g\right) d\lambda^2 + \cos\lambda\cos\Phi_g dhd\lambda \\
dz = &\left(\frac{R_o (1-e^2)\cos\Phi_g}{\chi^3} + h\cos\Phi_g\right) d\Phi_g + \sin\Phi_g dh \\
&+ \cos\Phi_g dhd\Phi_g + \left(\frac{R_o}{4}\sin\Phi_g \left(-2 - e^2 + 9e^2\cos^2\Phi_g\right) - \frac{h}{2}\sin\Phi_g\right) d\Phi_g^2
\end{aligned}
\tag{11.1}
$$

11.1 Global coordinates and curvilinear coordinates

For navigation on Earth, though inertial frames could be used in the intermediary calculations, the relation to a reference frame on a 'fixed' earth is the goal. For global positioning therefore an Earth-Centered Earth-Fixed (ECEF) non-inertial coordinate system is used. The X-axis is set to point from the origin to the intersection of the equator with the Conventional Zero Meridian, which defines the 0° longitude line on Earth's surface [Groves, 2013]. The Y-axis points 90° east from the X-axis, still on the equatorial plane. An Earth-fixed frame can be seen in Figure 11.1.

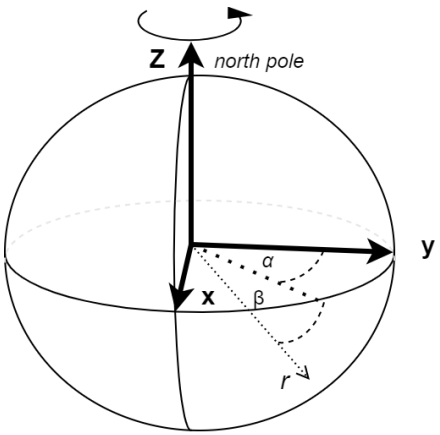


Figure 11.1 Illustration of a ECEF coordinate system, where a coordinate can also be given using two angles α and β along with distance from origin

The ECEF coordinate system enables positioning with respect to the origin at the center of earth - meaning either at the exact center of mass or at the centroid of the ellipsoid representation of Earth.

For practical applications, location relative to the surface of Earth is most appropriate. Using an approximate modelling of the oblate spheroid Earth through an oblate ellipsoid fit to the mean sea level overcomes the issues arising from an irregular surface. The ellipsoid model has an equatorial radius R_o, the polar radius R_p, and eccentricity e. The location of the center of this spheroid is not necessarily precisely at the center of mass, and instead has been measured through a common reference of satellite constellations.

The distance of an object from the surface of the ellipsoid model is the smallest distance of that object to a point on the surface - this is also known as the altitude of the point. It is then practical to define a point through its altitude and the point on the surface it is closest to. Points on the surface of an ellipsoid can be uniquely determined using two angles: longitude and latitude.

Longitude is the angle determining the east-west angular position of a point on

the surface of the ellipsoid representation of Earth, as measured from the prime meridian. Orthogonal to Longitude is the latitude, which determines the South-North angular position of a point on the ellipsoid surface. More specifically, in GNSS positioning Geodetic latitude is used [Groves, 2013]. Geodetic latitude is obtained by extending the normal of each point on the surface of the ellipsoid until it intersects the equatorial plane, then measuring the angle of intersection. Together, Geodetic latitude, longitude, and altitude are known as the curvilinear position and provide a set of orthogonal axes for positioning an object in a practical coordinate frame. The conversion from Geodetic coordinates (Φ_g, λ, h) to ECEF coordinates (X, Y, Z) can be seen in (11.2).

$$\begin{aligned} x &= \left(\frac{R_o}{\chi} + h\right)\cos\Phi_g \cos\lambda \\ y &= \left(\frac{R_o}{\chi} + h\right)\cos\Phi_g \sin\lambda \\ z &= \left(\frac{R_o(1-e^2)}{\chi} + h\right)\sin\Phi_g \end{aligned} \tag{11.2}$$

$$\text{Where } \chi = \sqrt{1 - e^2 \sin^2 \Phi_g}$$

A representation of curvilinear coordinates on an ellipsoid representation of Earth can be seen in Figure 11.2 [Groves, 2013].

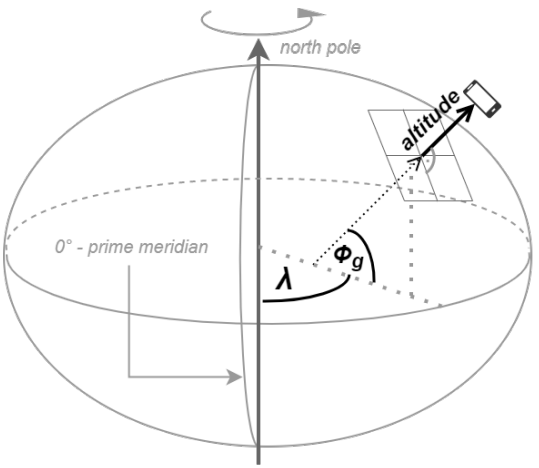


Figure 11.2 Curvilinear coordinates for positioning a UE in an ECEF coordinate system, with Geodetic latitude Φ_g and longitude λ

Algorithm 5: Adam [Kingma and Ba, 2014] with decoupled weight decay [Loshchilov and Hutter, 2017] combined with AMSGrad [Reddi et al., 2018] (AdamW-AMSGrad)

Result: Optimal parameter vector θ
Input: Learning rate γ
Input: Numerical stability coefficient: ε
Input: Running average coefficients: β_1, β_2
Input: Weight decay λ
Input: number of epochs N_{epochs}
Data: Training dataset $X_{training}, Y_{training}$, minibatch size M_{batch}
Data: Initialized parameter vector θ
Data: Initialized first moment $m \leftarrow 0$, Second moment: $v \leftarrow 0$, $v^{\hat{max}} \leftarrow 0$
for *t = 0 to* N_{epochs} **do**
  Sample elements $X_t^{(i)}, Y_t^{(i)}$ from $X_{training}, Y_{training}$;
  Calculate estimated gradient: $\hat{g} \leftarrow \frac{1}{M_{batch}} \nabla_\theta J_m \left(f(X_t^{(i)}, \theta), Y_t^{(i)} \right)$;
  Update parameters: $\theta \leftarrow \theta - \gamma \lambda \hat{g}$;
  Update first moment: $m \leftarrow \beta_1 m + (1 - \beta_1)\, \hat{g}$;
  Update second moment: $v \leftarrow \beta_2 v + (1 - \beta_2)\, \hat{g}^2$;
  First moment cont.: $\hat{m} \leftarrow \frac{m}{1-\beta_1^t}$;
  Second moment cont.:$\hat{v} \leftarrow \frac{v}{1-\beta_2^t}$;
  AMSGrad step: $v^{\hat{max}} \leftarrow max\left(v^{\hat{max}}, \hat{v}\right)$;
  Update learning rate: $\varepsilon \leftarrow U\left(\varepsilon, e\right)$;
  Update parameters using moments: $\theta \leftarrow \theta - \gamma \frac{\hat{m}}{\sqrt{v^{\hat{max}}} + \varepsilon}$;
end

Location, Data-name,	Start-date UTC: (SRS, GNSS)	Filename (SRS log) (GNSS log)	Purpose [Data_split]
NLoS-A NLoS-A1	2022-07-13 (12:27:57, 12:28:56)	SRS_log_noNLoS_A1.log gnss_log_noNLoS_A1.txt	Training
NLoS-A NLoS-A2	2022-07-18 (11:51:47, 11:58:04)	SRS_log_noNLoS_A2.log gnss_log_noNLoS_A2.txt	Val.
NLoS-A NLoS-T	2022-08-18 (08:16:12.5, 08:17:17)	SRS_log_noNLoS_test.log gnss_log_noNLoS_test.txt	Test
LoS-A LoS-A1	2022-07-20 (08:16:10, 08:17:19)	SRS_log_LOS_A1.log gnss_log_LOS_A1.txt	Training
LoS-A LoS-A2	2022-07-13 (13:05:46, 13:01:58)	SRS_log_LOS_A2.log gnss_log_LOS_A2.txt	Val. & Test [12000]
LoS-D (Tr) LoS-Dtr	2022-07-20 (09:15:16, 09:16:30)	SRS_log_LOS_D_T.log gnss_log_LOS_D_T.txt	Training
LoS-D (V) LoS-Dv	2022-07-20 (10:16:33, 10:17:03)	SRS_log_LOS_D_V.log gnss_log_LOS_D_V.txt	Val. & Test, [12000]

Table 11.1 Measurement datasets collected for the three detailed locations. NLoS-A refers to going around the garage, LoS-A refers to going around on the top of the garage, LoS-D (TR) refers to the directed dense training data for the rooftop. LoS-D (T) refers to the wandering dense validation data for the rooftop. Many further dataset recordings were made, such as separate LoS-A and LoS-D test datasets and re-measurements of all the LoS-A datasets. Unfortunately, data recording difficulties made these datasets unusable. Instead, for testing the Validation-intended datasets for LoS-A and LoS-D were split into testing and validation sets. [Data_split] refers to the index at which the data was split into validation and test datasets in these cases.

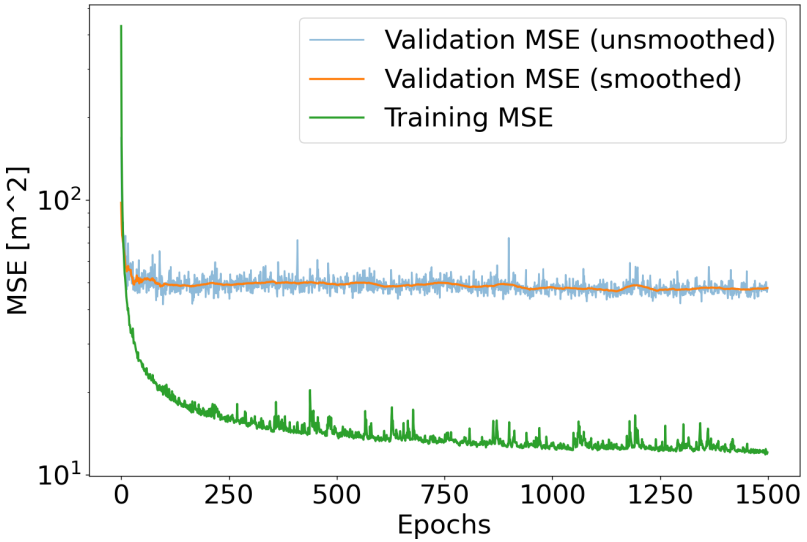


Figure 11.3 The training history of the 1500-epoch NLoS-A $\mathbf{X}_H^{FA}$ model. Overfitting is very clear, with the ideal validation plateau around 50-200 epochs.

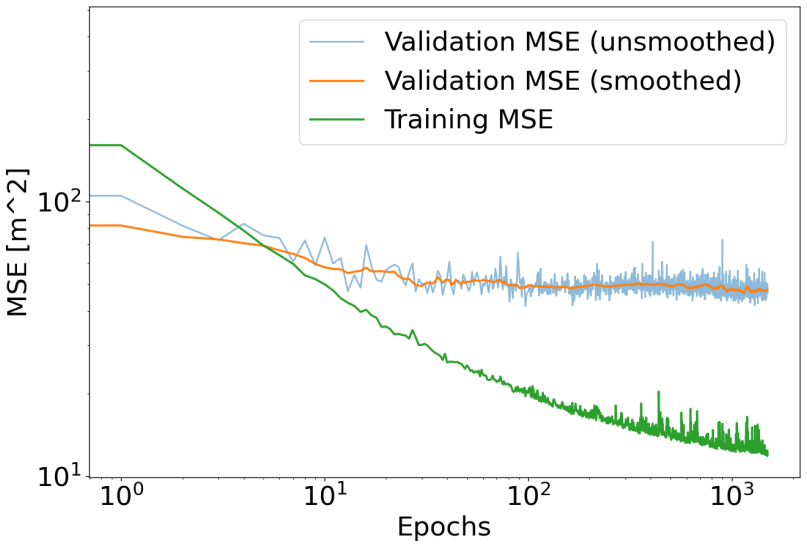


Figure 11.4 The training history of the 1500-epoch NLoS-A $\mathbf{X}_H^{FA}$ model. Overfitting is very clear, with the ideal validation plateau around 50-200 epochs. This time the Epoch-scale is logarithmic.

Input		N_{max}^{input}	$D_{proc.}$	$N_{max}^{proc.}$	A^{proc}	$D_{pos.}$	$N_{max}^{pos.}$	A^{pos}
$\mathbf{X}_H^{FA}$	[min]	[100]	[1]	[16]	[0.6]	[2]	[8]	[0.5]
	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓
	{max}	{800}	{7}	{48}	{1.0}	{6}	{30}	{1.0}
$\mathbf{X}_H^{mUE}$	[min]	[150]	[1]	[16]	[0.6]	[2]	[8]	[0.5]
	⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓
	{max}	{1000}	{7}	{48}	{1.0}	{6}	{30}	{1.0}

Table 11.2 The more thoroughly explored hyperparameter-space to find the final model and narrow down the parameter-space to use for generating architectures for the diverse-ensembles. Note that this is not the whole breadth of the explored architectures, only the one explored systematically. N_{max}^{block} refers to the number of neurons in the largest layer in the block, D^{layers} refers to the depth of the block in terms of layer number, and A^{layers} refers to the $N_{min}^{block}/N_{max}^{block}$ ratio, with each block having the maximum layer be the first, and the minimum layer the last.

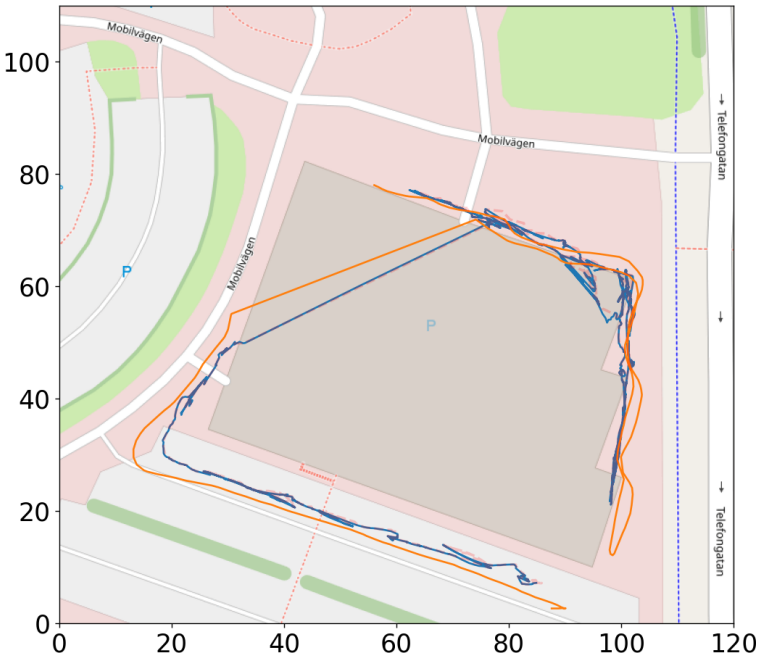


Figure 11.5 A part of the output of feeding the particle filter algorithm from Algorithm (4) the Diverse-50 ensemble outputs on the $\mathbf{X}_H^{FA}$ NLoS-A test. The orange line shows the GNSS path, the blue line the particle-filtered prediction. The large jump in location corresponds to an approximately 40 second connection loss, which the particle filter handles well by re-converging quickly to the new position