CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Five-dimensional local positioning using neural networks

Master of Science thesis in Computer Science – Algorithms, languages and logic

FREDRIK FURUFORS

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2017

Master's thesis 2017

Five-dimensional local positioning using neural networks

FREDRIK FURUFORS

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2017

Five-dimensional local positioning using neural networks
FREDRIK FURUFORS

Supervisor: Mary Sheeran, Computer Science and Engineering
Company representative: Oscar Sjöberg, Micropos Medical AB (publ)
Examiner: Richard Johansson, Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The Micropos Medical RayPilot system is an accessory to existing external beam radiotherapy installations which provides real-time positional feedback. The sensor plate, placed between the patient and the treatment couch, localizes the transmitter implant.

Typeset in LaTeX
Revised: June 20, 2017
Gothenburg, Sweden 2017

Five-dimensional local positioning using neural networks

FREDRIK FURUFORS
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

In this thesis, a method for real-time transmitter localization is evaluated. An existing system has acted as testbed for the evaluation. This system uses an electromagnetic transmitter and a receiver board with 16 antennas. The antenna values are used to recover the transmitters position and two angles, the five dimensions. The proposed solution is an inverse modelling feed-forward neural network, a multilayer perceptron, which is trained and evaluated with the use of the TensorFlow library.

The project resulted in a purely software based estimator which requires no change to the testbed and can act as a drop in replacement for the previous algorithm. The new estimator has accomplished improvements in estimation speed (more than 100× faster), expansion of the volume in which the position can be recovered (27× larger), enlarged range of angles (10 % per axis) and has improved the precision of the position estimates (error at the 95th percentile reduced to $\sim \frac{1}{3}$ of the previous implementation). The new algorithm is a substantial improvement on the previous implementation, enabling new use cases for the system.

Keywords: Function approximation, Inverse modelling, Neural networks, Multilayer perceptron, Machine learning, Localization

Acknowledgements

I am most grateful for the support I've recieved in my efforts to make a relevant thesis. Mary Sheeran, thanks for the guidance throughout the project and your effort to keep me on track with regards to writing and scientific approach. Oscar Sjöberg, thanks for some of the most advanced, and absurd, discussions I've ever had and more new ideas for improvements than I was able test. Johan Linder, for your quick repairs of all the lab equipment my experiments broke. The Micropos team: Tomas, Andréas, Bo, Roman, Kauko, Markus and Hanna, for keeping the spirit up, providing an excellent platform to build upon and enduring my never-ending babble about neural networks, optimization, L2-spaces, radial errors, graphics cards, et cetera. Thanks to Olof Mogren for the initial input on how to approach the problem and my examiner Richard Johansson who provided valuable feedback at the halftime presentation. Thanks to all great open-source maintainers out there who provided me with state of the art tools and documentation. Thanks to the countless number of teachers who have supported me through all of my education.

Last but not least, thanks to my family for constant support.

Fredrik Furufors, Gothenburg, May 2017

Contents

List of Figures

List of Tables

Glossary

ADC analog-to-digital converter. 29, 53, 55, 66
ANN artificial neural network. 1–3, 10, 11, 19, 23, 27–30, 47, 48, 52–54, 56, 58, 63, 65, 66, 91
ASIC application specific integrated circuit. 53

CPU central processing unit. 33, 53
CT computed tomography. 9
CTV clinical target volume. 5, 6

EBRT external beam radiotherapy. 1, 5, 7, 11–13, 52
EMF electromagnetic field. 2, 8, 9, 13
epoch A measurement of the amount of training in an artifical neural network. In one epoch every datum is presented to the training algorithm once. 23

FPGA field-programmable gate array. 1, 4, 33, 66

gating An on/off-signal which may be used to modulate the behaviour of external components. For example a signal indicating whether a measure is valid. 27, 52
GGAM gradient guided approximate minimum. 16, 19, 27
GPU graphics processing unit. 31, 33, 46, 53, 63, 66

HNN hardware neural network. 1, 3, 66

JSON javascript object notation. 35

LabVIEW Laboratory Virtual Instrumentation Engineering Workbench. A graphical programming language developed by National Instrument. The main application area is orchestration of measurement instruments. 14, 16
LINAC linear accelerator. 7, 8

MLP multi-layer perceptron. 19, 21–23, 27, 28, 38, 65

PTV planning target volume. 6

RSS received signal strength. 11

TPU tensor processing unit. 53

1

Introduction

In inverse modelling, a set of observations is used to estimate the causing factors. This thesis explores artificial neural networks (ANNs) as a solution to recover five degrees of freedom in an electromagnetic transmitter from 16 measurements of the electromagnetic field emanating from the transmitter. Inverse modelling using ANNs is a powerful tool, with the ability to model multidimensional, nonlinear relationships. It requires little area-specific knowledge, even when other modelling approaches would require such expertise. Inverse models can also be used when no known mathematical model is available [1]. Inverse modelling using an ANN allows the designer to *train* a system, using a number of input-output pairs. A well designed ANN can produce good output estimations on novel input data with modest hardware requirements.

This thesis describes the process of implementing an ANN for direct inverse modelling of an implant's position in a patient undergoing external beam radiotherapy (EBRT). Two alternative solutions are proposed, the first a purely software-based estimator, the other is an embedded hardware neural network (HNN) which is trained on a workstation computer, *off-chip* training, and implemented on an field-programmable gate array (FPGA) which can utilize the inherent parallelism in ANNs. As the project progressed the focus shifted towards the software based estimator.

1.1 Context

Micropos Medical AB (publ) is a Gothenburg-based company which has developed a tumour localization system called RayPilot. The system consists of a receiver plate and a transmitter which is implanted in close proximity to the tumor, shown on the cover page. The receiving plate contains 16 antennas which measure the electromagnetic signal sent out by the transmitter implant, see Figure 1.1.

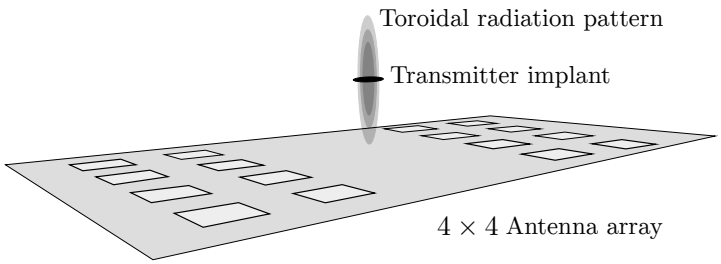


Figure 1.1: An electromagnetic transmitter implant is placed in close proximity to the tumor, the 4×4 antenna array collects data used for localization.

A software suite uses data from the antennas to produce a positional estimate of the implant, see Figure 1.2. The algorithm which recovers the implant's position from the electromagnetic field image is the subject of this thesis.

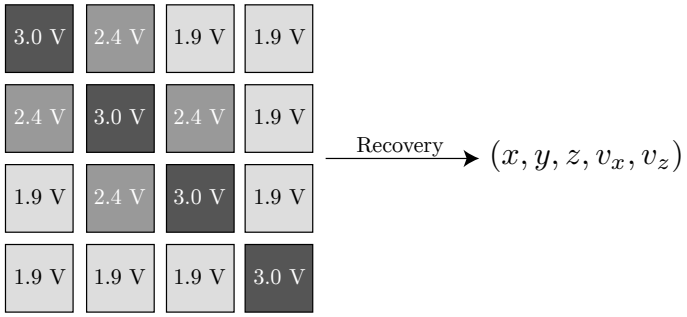


Figure 1.2: A program recovers the positional data from the measurements of the 4×4 antenna array.

The estimate is used to improve the precision in the radiotherapy sessions by providing feedback on the tumour's position. In addition to the tracking aspect, some versions of the implant are also capable of measuring the amount of radiation which reaches them. This feature is called *dosimetry* and provides additional safety to the patient by giving feedback on the radiation dose.

1.2 Motivation

While many articles has been published on the topic of inverse modelling using ANNs, this application is novel. The directional dependency of the electromagnetic field (EMF) produced by the transmitter in combination with second-order factors such a antenna coupling and external noise, makes a direct mathematical model infeasible with regards to research and development costs.

Multi-layer feed-forward neural networks with at least one hidden layer have been proven capable of universal function approximaton [2]. Hence this type of ANNs is a candidate for the inverse modelling task.

1.3 Goals and challenges

The project has multiple goals pertaining to different aspects of the RayPilot system's performance:

- Maintaining or improving the accuracy of the positional predictions, see Section 4.4 for further details
- Extend the recoverable volume as defined in Section 3.1.2
- Decrease the latency of the predictions; the current algorithm has a minimum latency of about 10–20 ms. Previous studies has already shown that an ANN solution can reduce the latency [3]
- Demonstrate that development of an embedded HNN solution is feasible thanks to the use of modern tools

Associated with these goals are some challenges. The biggest challenge may be the choice of neural network configuration. There are few general results in the field and each implementation needs to be tailored. The approach is that of experimentation and comparative analysis of proposed models. There is no guarantee that a neural network solution will outperform the current algorithms used by the company. A negative result may guide future work in the field to different strategies.

A clear drawback of a neural network solution to this problem is the need to argue for its correctness. How does one for example prove that the network will produce reasonable estimates in all of the measurement volume without performing an exhaustive search? This is however the reality for most physical measurement systems; this question will have to be addressed by the company; in this project the system is evaluated against a reasonably large set of test examples with known position and antenna value pairs.

A challenge in working with neural network's is to optimize the networks ability to generalize, i.e. get a good interpolation between the known examples. A common problem is overfitting where the trained network only makes good predictions on data from the training set, but bad estimates on new data points. Measures such as *early stopping* by using *holdout validation* [4] and weight-*regularization* [5] can prevent overfitting.

1.4 Delimitations

The end product is a proof-of-concept which focuses on the scientific/engineering aspect of the problem. The project is delimited to a stand-alone solution capable of performing position estimations from the antenna values but will not be tightly integrated with the RayPilot system. The result may at a later point be integrated by the company.

This thesis treats the RayPilot hardware in a somewhat idealized manner. There are revisions of the hardware contained in the RayPilot system; differences emanating from them are not considered. During the project, a fixed set of hardware has been used to investigate the proposed recovery algorithm. Under these conditions,

each instance of the algorithm will only be valid for a specific system.

1.5 Thesis organization

Chapter 2 provides background on radio-therapy, the RayPilot system and previous work on the topic.

Chapter 3 puts the thesis in context by providing theoretical background on neural networks, FPGAs and the equipment used throughout the project.

Chapter 4 introduces the methodology and the performance measures upon which the evaluation of the projects results are based.

Chapter 5 contains the results and metrics with commentary on the performance.

Chapter 6 draws conclusions based on the results and discusses the contributions of this project.

2
Background

2.1 Prostate cancer and treatment

Prostate cancer is the most common diagnosed cancer among Western European men; in 2008 an estimated 913 000 cases were reported worldwide [6]. Curative treatment methods include active surveillance, radical surgery (prostatectomy), radio- and brachytherapy [7]. This thesis focuses on advances in EBRT for treatment of prostate cancer. EBRT is a form of radiotherapy where the patient is situated on a treatment couch and the X-ray beam of a linear accelerator is aimed at the clinical target volume (CTV).

2.1.1 External beam radio-therapy

An EBRT treatment is composed of a number of *fractions*, 5–40 depending on the method, in which the patient undergoes a radiation exposure according to a treatment plan. In order to localize the tumor and identify the treatment volume, medical imaging is done at the start of the treatment.

2.1.1.1 Hypofractionation

Conventional EBRT treatment plans deliver around 80 Gy[1] divided into circa 40 fractions. In two recent studies, HyPro [8] and CHHiP [6], conventional and hypofractionational planning have been evaluated in large scale randomized trials. Hypofractionation uses, as the name implies, fewer fractions, 5–20, but delivers a comparable total dose of radiation. The CHHiP study reported that *"Hypofractionated high-dose radiotherapy seems equally well tolerated as conventionally fractionated treatment at 2 years"* [6]. There are many benefits of fewer treatment sessions including economical, logistical and resource aspects for both the patient and the clinics.

In conventional fractionation the law of large numbers applies to a higher degree *vis-à-vis* hypofractionation; on average the prostate is in its expected position. In hypofractionated plans, movements of the prostate can potentially have a larger impact since each intermittent positional deviation will amount to larger deviations from the dosage planning, given the higher radiation intensity. Given the method's

[1] Gy: Gray, the SI unit which measures ionizing radiation dose. 1 Gy = 1 J/kg.

increased sensitivity to organ movement, real-time movement tracking systems are emerging.

2.1.2 Prostate movement during radio-therapy

There are two distinct forms of prostate movement: those that occur during the fractions, *intrafractional* movements, and those that occur between fractions, *inter-fractional* movements. The intrafractional movements can be monitored by using the RayPilot system. Interfractional movements are detected by medical imaging prior to the treatment session and the data from these images also serve as a reference for calibration of the transmitters position in relation to the tumour when using the RayPilot system.

In Figure 2.1 is an example from a radiotherapy session where the target volume displays considerable migration, here up to 9 mm in the longitudinal direction, i.e. on a line between the patient's hips; other sessions from the study showed movements of up to 16 mm [9].

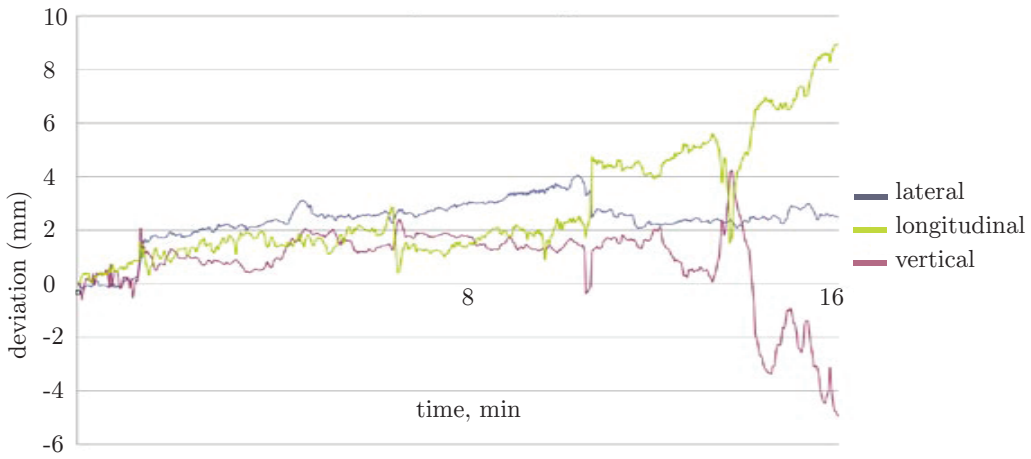


Figure 2.1: Deviations from an *in-vivo* study of target movement, visualized per dimension during radiotherapy. Used with permission from the authors.

During the radiotherapy, a volume slightly larger than the prostate is radiated, the planning target volume (PTV), in order to ensure that the required dose reaches the CTV. Minimization of the amount of radiation which reaches healthy tissue is important for the patients' quality of life after treatment. One of the ways to accomplish this is real-time position tracking of the prostate during the treatment. Such monitoring allows for smaller margins around the CTV. Studies have shown significant *organ movement* both between and during fractions; in [10] a deviation of >3 mm was reported 7.7 % of the monitored time in 742 studied fractions. In [10] the Calypso local positioning system by Varian Medical Systems was used to measure the deviations.

2.1.2.1 Tracking prostate movement

The RayPilot system has a couple of competitors; the two primary are developed by manufacturers of linear accelerators (LINACs): Elekta's Clarity which uses ultrasound and Varian's Calypso which uses electromagnetic resonance. Another approach for positional feedback of tumours in soft tissue is magnetic resonance radiation therapy (MR/RT); this method combines magnetic resonance imaging with radiotherapy; computer vision algorithms are applied to recover the position.

2.1.2.2 Reducing prostate movement

An additional measure to tracking, fixtures can be used in order to reduce organ migration. One approach is a rectal retractor rod, a fixture which also separates sensitive, healthy tissue from the prostate thereby aiming to reduce adverse side effects. A recent study used a combination of the RayPilot system and a rectal retractor rod observed deviations of >3 mm in 16.0 % of the monitored time in 260 fractions [11].

While not a direct fixture, bladder filling routines has been evaluated as a procedure to reduce interfractional movements, by normalizing the prostate's surrounding environment. In a study of this method, different levels of bladder filling moved the prostate >1 cm in 30 % of the patients [12].

2.2 Micropos Medical and RayPilot

Micropos Medical's product RayPilot is sold as a third-party accessory to existing EBRT linear accelerators. The RayPilot system consists of the sensor plate (placed onto the linear accelerator's treatment couch), the transmitter implant and a workstation running the RayPilot software, see figure 2.2. The RayPilot system can also be used for proton therapy, a specialized form of EBRT using protons instead of photons. The sensor plate which is placed over the treatment couch, see Figure 2.2, is an approximately 30 mm thick board containing 16 antennas. This form factor was chosen to integrate well with typical installations.

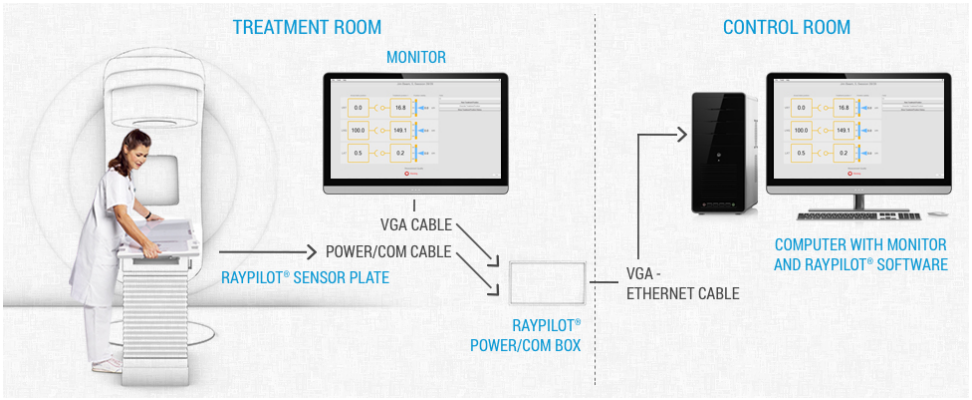


Figure 2.2: The RayPilot system consists of the sensor plate (placed onto the linear accelerator's treatment couch), the transmitter implant and a workstation running the RayPilot software. Figure used with permission from Micropos Medical.

2.2.1 The RayPilot sensor plate

The sensor plate gathers information about the EMF emitted from the RayPilot implant. It contains 16 antennas in two groups of 8, as shown in Figure 1.1. In the area between the antennas is the "low attenuation zone" which is designed not to interfere and withstand the radiation from the LINAC.

Given the flat layout, the antenna measurements are placed in a plane bisecting the transmitter's electromagnetic field, a compromise between the quality of the collected data and the physical form factor.

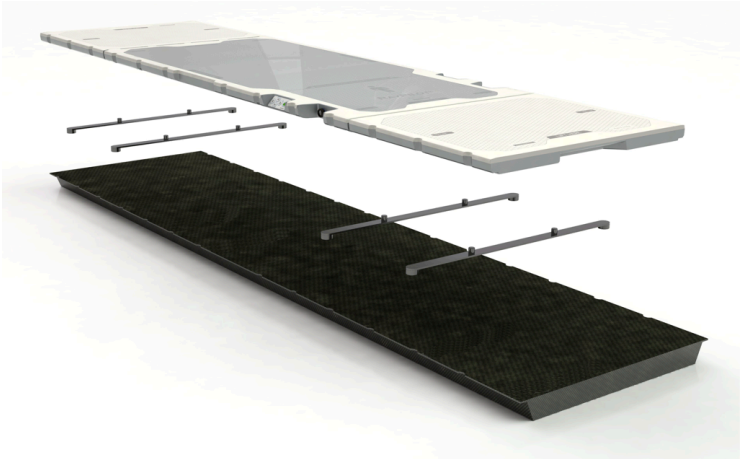


Figure 2.3: The RayPilot sensor plate is placed onto the carbon fiber treatment couch. Exact, replicable placement is enabled by the use of fixtures.

The voltage is the time-averaged, absolute-valued intensity of the EMF travelling perpendicularly through the antenna's cross-section. The underlying physical phenomenon is described in Faraday's law of induction [13].

2.2.2 The RayPilot transmitter

The transmitter is cylindrical and measures $3 \times 17\,\text{mm}$. It is implanted under local anesthesia, using a special purpose needle and remains implanted throughout the whole treatment period. Inside the implant is an antenna, which is driven by a reference signal from the sensor plate. The reference signal is a sinusoidal wave with well defined amplitude and frequency. When this signal is sent through the transmitter it generates EMF which has a toroidal radiation pattern as illustrated in Figure 2.4. The radiation pattern visualizes the signal intensity's directional dependence. Here the transmitter is aligned with the y-axis. The EMF is rotationally symmetric around the implant's axis. The radiation diagram highlights the angular dependency of the dipole antenna—the signal is strong in the directions perpendicular to the transmitter. The sensor plate and the transmitter are connected via a shielded cable.

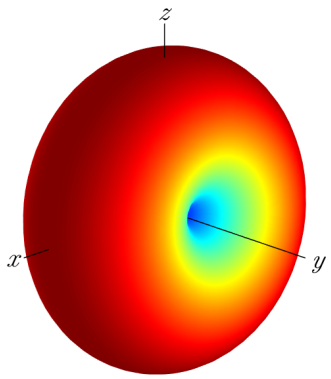


Figure 2.4: The toroidal radiation pattern of the RayPilot transmitter. Here the implant is aligned with the y-axis. The simulation was generated with the Matlab Antenna toolbox.

The implant is placed in close proximity to the tumour, inside the prostate. Prior to each fraction, a computed tomography (CT) scan is taken to reaffirm the implant's position in relation to the tumour. Small changes in position can be corrected by adding an offset in the RayPilot software. The illustration in Figure 2.5 shows an implanted transmitter.

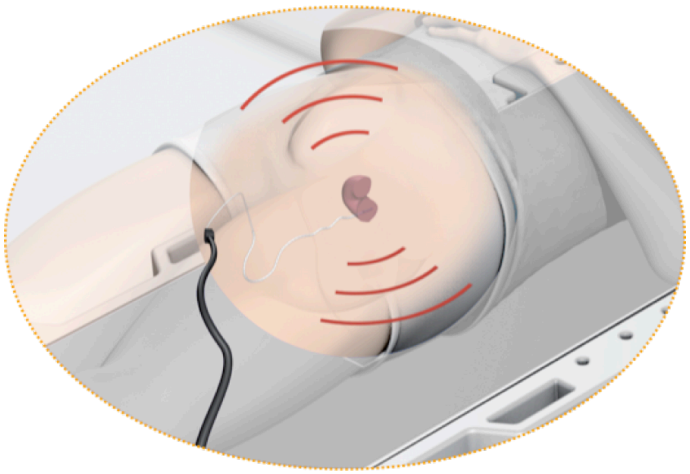


Figure 2.5: Implanted transmitter. The white part of the cable is integrated into the transmitter, while the black part is an extension cable which connects the system to the transmitter.

2.3 Previous work

The company has performed two studies where neural networks have been trained in order to recover the position. In a study from 1999, a feed-forward neural network with one hidden layer of 34 nodes was used to recover the position of a magnet with the use of 16 magnetic sensors [3]. This experimental setup was a prototype of the RayPilot product. The neural network was able to produce an estimate in 10 ms on late 1990's hardware. The network models were trained for approximately 45 minutes and the estimations on the test set had a mean square error of 0.050–0.052 mm radially; the sensors were placed in a 20×20 mm grid and the training volume was a cube with 10 mm sides [3]. The software used was NeuroShell 2, a commercial tool designed for stock market predictions. The result was deemed successful as a proof of concept.

In a 2006 thesis project, ANNs were once again implemented, now using the current RayPilot technology, but with a 4-by-3 antenna array instead of the current 4-by-4 [14]. Two approaches were evaluated: the NeuroShell 2 (NS2) tool and a feed forward neural network implemented in Matlab (ML). The outcome of this experiment is presented in Table 2.1. The numbers in the network type are the neuron count in the layers: *input-hidden-output*; the nomenclature is introduced in Section 3.3.2. The study was limited to networks with a single hidden layer with a low number of neurons.

Table 2.1: Results from the company's previous attempts at position estimations done with a neural network on the RayPilot data.

Network type	Training set size	Test set size	Mean error	Std. dev.	Max error
FFNN: 12-8-5 (NS2)	6221	1555	3.40 mm	1.36 mm	12.83 mm
FFNN: 12-8-5 (NS2)	5782	1994	5.40 mm	2.30 mm	15 mm
FFNN: 12-9-5 (ML)	6000	3000	2.16 mm	0.92 mm	8.10 mm

The thesis report has remarkably little information or motivation concerning the network types used—however neural networks were not the main focus of the thesis. The report gave some hints on future work: *"Since we have shown that it is probable that you can create an Artificial Neural Network that performs just as good as using an Optimization Algorithm does[,] it would be interesting to pursue this investigation further. Trying to find an optimal network for the problem and/or setting up guidelines as to how the networks shall be created and trained would probably be a good thing."* [14].

2.4 Related work

While the specific application is novel ANNs has been used in similar applications where one of the more common is localization using wireless sensor networks.

In [15] neural networks are used to determine the position of WIFI clients connected to a set of wireless access points. The input to the network was the received signal strength (RSS) between the client and the different access points. When using multiple RSS inputs the method is called *multilateration*, as opposed to the more common trilateration. The underlying phenomenon is analogous to the work at hand; however, there are some major differences: two dimensional position instead of five dimensional, omni-directional transmitter antennas instead of the dipole pattern used here, randomly ordered antennas (access points) instead of ordered grid, different target ranges and completely different requirements on the accuracy. For a 40×40m room the system described in [15] achieves an accuracy of approximately 1 m; translated to the work at hand that roughly represents 2.5 mm in a volume with sides of 100 mm.

The work presented in [16] uses an experimental setup very similar to [15]. This work dives deeper into filter and pre-processing optimizations which are not directly applicable to the work at hand. In contrast to [15] this work uses a grid based layout for the access points, which is more similar to the antenna array of the RayPilot sensor board.

2.5 Conclusion

The state of EBRT-treatments is in flux; the transition towards hypofractionation and increased awareness of organ movement is driving a demand for organ tracking

during the treatments. Micropos has an interesting product and a mature hardware system which constitutes a good testbed for the proposed research.

If the project's goals can be fulfilled, the results will likely have a positive impact on a large number of patients undergoing EBRT. The timing and conditions for the project at hand are right.

3

Context

This chapter will provide the necessary theoretical background to follow and understand the project, including the experimental setup, the current algorithm used in RayPilot and background on artificial neural networks (ANNs).

3.1 Experimental setup

This section provides background on how the data used for modelling is collected, what the experimental limitations are, and provides definitions concerning the collected data.

3.1.1 Coordinate system

The data is oriented according to standard mathematical orientations, shown in Figure 3.1. The y-axis points towards the patient's head and the x-axis towards the patient's left. Consequently, the z-axis points perpendicularly upwards from the treatment couch. The angles v_x and v_z are defined around the x and z axes. The EMF emitted from the transmitter is symmetrical around the y-axis, hence the v_y angle is non-recoverable.

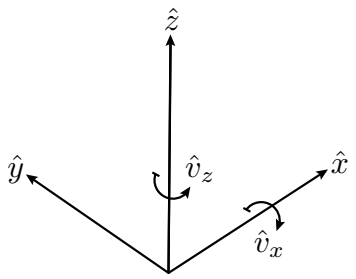


Figure 3.1: The coordinate system is a standard mathematical right-handed system.

For simplicity this project deals only with this notation of positions; however, the RayPilot software conforms to the IEC 61217 standard [17], an industry standard which defines a set of coordinate systems used in EBRT.

3.1.2 Data collection and test fixtures

To recover the transmitter's position the company currently uses a polynomial regression model. This model is constructed with data generated using robotized fixtures called *Autosetups*. The fourth incarnation of the Autosetup is shown in Figure 3.2. It has the capability of moving the transmitter ±150 mm in the axial dimensions and ±45° around the two rotational axes. The servos for the x-,y- and z-dimensions have repeatability of 0.1 mm. The fixture is controlled and automated using an existing LabVIEW interface and stores the positional setpoint and the associated antenna values. The device was used to generate training and verification data. The data generated is usable since the human body is almost transparent to the electromagnetic waves at the transmitter's frequency; this has been verified by the company.

During the project, version three and four of the Autosetups were used. Version four is superior in all aspects; however, it was not fully functional until week 14 of the project. Hence, some databases were generated with the predecessor which has a smaller range of motion: ±90 mm and ±40°.

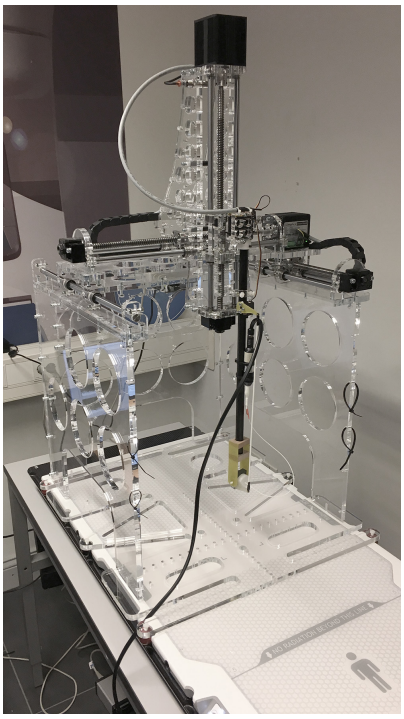


Figure 3.2: The fourth generation Autosetup, a robotized fixture for precise transmitter positioning over a sensor plate. Picture used with permission from Micropos Medical AB.

The core question of this project is whether a neural network can be adapted to recover the positional data from the antenna values and whether it can outperform previous methods used by the company with regards to precision and recoverable measurement volume. The recoverable volume, $\mathbb{R}_v$, is here defined to be the volume in which the estimate, $\hat{\mathbf{p}}$, deviates less than 1 mm radially from the physical position

$\mathbf{p}$:

$$\mathbb{R}_v = \{\mathbf{p} \mid \|\hat{\mathbf{p}} - \mathbf{p}\| \leq 1\,\text{mm}, \mathbf{p} \in \mathbb{P}\}$$

where $\mathbb{P}$ is the maximum trainable volume, limited here by the transmitter fixture. With the fixture in Figure 3.2 this volume is:

$$\mathbb{P} = \{\mathbf{p} = (x, y, z) \mid \|x - x_0\| \leq 15\,\text{cm}, \|y - y_0\| \leq 15\,\text{cm}, \|z - z_0\| \leq 15\,\text{cm}\}$$

where $\mathbf{p}_0 = (x_0, y_0, z_0)$ is located in the center of the fixture.

With the current algorithm $\mathbb{R}_v$ is approximately $\pm 5\,\text{cm}$ in each direction from $\mathbf{p}_0$ which can be varied slightly to fit the intended application. Potentially, several overlapping polynomial models with different $\mathbf{p}_0$ could be trained to increase $\mathbb{R}_v$; this approach will not be explored in this project.

3.1.2.1 Training databases

The Autosetups generate tab separated, flat file databases containing the setpoint position, angles and the associated antenna voltages. The databases are identified by `DB#` where `#` is a serial number. During this project a number of such databases have been used. The properties of these databases are listed in Table 3.1. The time needed to generate a standard database, such as `DB271` is approximately 4 hours. A database for the maximum possible volume, see `DB1071`, takes just over six days to generate[1].

Table 3.1: Properties for the training databases used throughout the project.

Identifier	**X** (mm)	**Y** (mm)	**Z** (mm)	$\mathbf{V}_x$ (°)	$\mathbf{V}_z$ (°)	**Datapoints**	**Autosetup**
`DB271`	37–137	44–144	80–180	-40–40	-40–40	7776	v3
`DB476`	0–180	0–180	0–180	-40–40	-40–40	161 051	v3
`DB1071`	20–280	0–300	0–300	-45–45	-45–45	175 616	v4

The databases used for training contains equidistantly spaced points along each axis, see Figure 3.3a. For example, `DB271` uses $6^5 = 7776$ data points, i.e. each dimension is split into 5 steps. In the case of the x-axis this implies a step size of $(137 - 37)/(6 - 1) = 20\,\text{mm}$. When researching extended volumes, `DB476` was the one of two primary data source and it used 11 data points per dimension. One important aspect of these training sets are that the data span the whole training volume. Of these databases, `DB476` has the highest spatial resolution while `DB1071` uses a spacing closet to the original data set `DB271`. This is the explanation for the number of data points in the databases in Table 3.1.

3.1.2.2 Evaluation databases

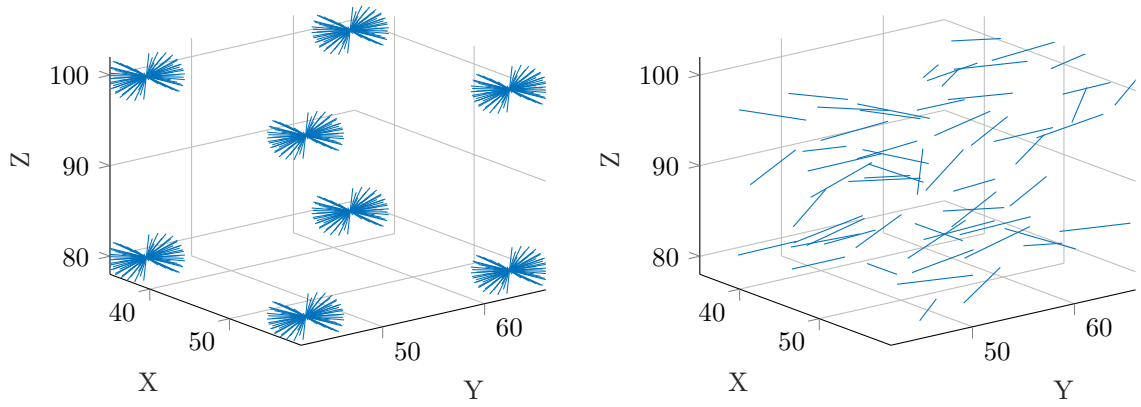
While the training databases used equidistantly spaced data points in a grid, the evaluation databases used random points spanning the same data space. The evaluation-mode of the Autosetup generates the desired number of data points.

[1] At the end of the project a change was proposed to the order of positions and angles. This, and some spin-off optimizations, resulted in a reduction of the time taken by 80%

The LabVIEW program uses a random number generator to generate and measure uniformly distributed data points, see Figure 3.3b.

These "evaluation" databases are used for three purposes:

- **Validation sets**—Used during the hyper-parameter search (introduced in Section 4.2.3) and training, as cross-validation (introduced in Section 4.2.2.3).
- **Test sets**—Given a finalized predictor, this data is used for performance evaluation.
- **Additional training data**—Since the training set generated by Autosetup has data points on a grid, additional uniformly distributed data can be added giving a more diverse training set; this is explored in Section 5.3.3.



(a) `DB271` training database.

(b) `DB271_eval_1` evaluation database.

Figure 3.3: Distribution of training and evaluation data points from a subset of `DB271`. Each line represents the placement and orientation of the transmitter.

3.2 Current implementation

The algorithm used to recover the transmitter's position in the RayPilot system has been replaced multiple times. The current implementation was developed as a master thesis project in 2006 and is called gradient guided approximate minimum (GGAM) [14]. The GGAM algorithm improved on the previous algorithm primarily in estimation speed and by introducing an upper bound on the run-time.

GGAM is an iterative search algorithm based on the classic gradient descent algorithm, which is the subject of Section 3.2.2. The GGAM modifications are introduced and motivated in Section 3.2.3 as well as some key performance figures from the thesis project [14].

3.2.1 Line search

One prerequisite for the gradient descent algorithm is a line search algorithm. Given a function and an interval, a line search aims to find a local optimum of the function

on the interval. One common method is the bisection algorithm described below [18, pp.17–19].

Algorithm 1: Bisection line search

Result: On convergence, x_n is a local minimum of $\phi(x)$ between a_0 and b_0
$n \leftarrow 0$
select an interval $[a_0, b_0]$
$x_0 \leftarrow \frac{a_0+b_0}{2}$
repeat
  compute the derivative $\phi'(x_n)$
  if $\phi'(x_n) < 0$ **then**
    $a_{n+1} \leftarrow x_n$
    $b_{n+1} \leftarrow b_n$
  else
    $a_{n+1} \leftarrow a_n$
    $b_{n+1} \leftarrow x_n$
  end
  $n \leftarrow n + 1$
  $x_n \leftarrow \frac{a_n+b_n}{2}$
until $|x_n - x_{n-1}| < \epsilon$;

3.2.2 Gradient descent

The gradient descent algorithm is an iterative, multidimensional optimization technique which uses the function's gradient as the direction for a line search. The gradient is the multidimensional equivalent to a function derivative:

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

where $\mathbf{x} = [x_1, \ldots, x_n]$. The gradient is a vector pointing in the direction of maximal growth. In the algorithm below, each iteration performs a line search in the direction of maximal decline, $-\nabla f(\mathbf{x})$, thereby finding a local minimum if the algorithm converges. There are many variations and applications of gradient descent algorithms, see textbooks such as [18] and [19]. One application is the back-propagation algorithm used in training of neural networks, see Section 3.3.3.

Algorithm 2: Gradient descent, line search variation

Result: In case of convergence, $\mathbf{x}_n$ is an approximative, local minimum for $f(\mathbf{x}_n)$
$n \leftarrow 0$
select starting point $\mathbf{x}_0$
repeat
  compute the gradient $\mathbf{d} = \nabla f(\mathbf{x}_n)$
  formulate the 1-dimensional optimization problem $\phi(\eta) = f(\mathbf{x}_n - \eta\mathbf{d})$
  find $\hat{\eta}$ which minimizes $\phi(\eta)$ using the algorithm in Listing 1
  $\mathbf{x}_{n+1} \leftarrow \mathbf{x}_n - \hat{\eta}\mathbf{d}$
  $n \leftarrow n + 1$
until $|\mathbf{x}_n - \mathbf{x}_{n-1}| < \epsilon$;

3.2.3 Gradient guided approximate minimum

This variation on gradient descent algorithm is described in Listing 3. It was designed as part of the 2006 thesis project [14]. The main differences from algorithm in Listing 2 are:

- Fixed number of iterations instead of the ϵ-convergence criteria.
- No line search; instead the step size is decreased if the scalar product of the gradients of two consecutive iterations is negative. A negative scalar product indicates that the search direction has changed dramatically, the multidimensional equivalent of a u-turn. This requires fewer costly evaluations of $f(\mathbf{p}_i)$.

Algorithm 3: Gradient guided approximate minimum (GGAM) [14]

Result: After n iterations $\mathbf{p}_n$ is an approximate minimum
set n as number of iterations
set $\mathbf{p}_0$ as the center of the search space
$\mathbf{d}_{\text{prev}} \leftarrow \nabla f(\mathbf{p}_0)$
set γ as initial step length
for $i \leftarrow 1$ **to** n **do**
  update search direction $\mathbf{d} \leftarrow \nabla f(\mathbf{p}_{i-1})$
  if $\mathbf{d} \cdot \mathbf{d}_{prev} < 0$ **then**
    $\gamma \leftarrow \frac{\gamma}{2}$
  end
  $\mathbf{p}_i \leftarrow \mathbf{p}_{i-1} + \gamma d$
  if $\mathbf{p}_i$ *is outside the search space* **then**
    $\mathbf{p}_i \leftarrow$ closest point inside search space
  end
  $\mathbf{d}_{\text{prev}} \leftarrow \mathbf{d}$
end

3.2.4 Polynomial model

The polynomial model is constructed from a training database using least square polynomial regression [20], i.e. solving an overdetermined equation system to find the coefficients. The current implementation uses 16 sets of five-dimensional, fourth degree multivariate polynomials, $\mathcal{P}_4$, i.e. one polynomial per antenna. The resulting function transforms a test position to a vector of estimated antenna values:

$$\mathcal{P}_4(x, y, z, v_x, v_z) = \mathcal{P}_4(\mathbf{p}) = \hat{\mathbf{a}} = [\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_{16}]$$

Where each of the $n = 1 \ldots 16$ polynomials has terms of the form:

$$\mathcal{P}_4(x, y, z, v_x, v_z) \cdot \hat{\mathbf{u}}_n = \sum_{a=0}^{4} \sum_{b=0}^{4} \sum_{c=0}^{4} \sum_{d=0}^{4} \sum_{e=0}^{4} c_{(a,b,c,d,e)} x^a y^b z^c v_x^d v_z^e$$

Hence each of the 16 polynomials in $\mathcal{P}_4$ has $5^5 = 3125$ coefficients which are determined using the 7776 equations given by the standard sized training database. Each of the 16 polynomials are overdetermined. In total this model has 50 000 coefficients.

Prior to the regression, the search space is linearly scaled to a fixed range for each of the five dimensions and rescaled after the optimization.

3.2.5 Optimization algorithm

The function under optimization is the L2-distance between the estimated, $\hat{\mathbf{a}}$, and the measured antenna vectors, $\mathbf{a}$:

$$\mathcal{E}_{\mathcal{P}_4}(\mathbf{p}) = ||\mathcal{P}_4(\mathbf{p}) - \mathbf{a}||_2 = ||\hat{\mathbf{a}} - \mathbf{a}||_2$$

Minimization of $\mathcal{E}_{\mathcal{P}_4}(\mathbf{p})$ results in a position estimate, $\hat{\mathbf{p}}$. The current implementation uses the GGAM algorithm, described in Listing 3. The gradient is defined as:

$$\nabla\mathcal{E}_{\mathcal{P}_4}(\mathbf{p}) = \left(\frac{\partial\mathcal{E}_{\mathcal{P}_4}(\mathbf{p})}{\partial x}, \frac{\partial\mathcal{E}_{\mathcal{P}_4}(\mathbf{p})}{\partial y}, \frac{\partial\mathcal{E}_{\mathcal{P}_4}(\mathbf{p})}{\partial z}, \frac{\partial\mathcal{E}_{\mathcal{P}_4}(\mathbf{p})}{\partial v_x}, \frac{\partial\mathcal{E}_{\mathcal{P}_4}(\mathbf{p})}{\partial v_z}\right)$$

where each partial derivative if calculated numerically as:

$$\frac{\partial\mathcal{E}_{\mathcal{P}_4}(\mathbf{p})}{\partial x} = \frac{\mathcal{E}_{\mathcal{P}_4}(\mathbf{p} + \Delta x\mathbf{u}_x) - \mathcal{E}_{\mathcal{P}_4}(\mathbf{p})}{\Delta x}$$

where $\Delta x\mathbf{u}_x$ is a small step along the x-axis. Since the GGAM-algorithm moves in the direction of $-\nabla\mathcal{E}_{\mathcal{P}_4}(\mathbf{p})$ it iteratively produces antenna estimates which better and better match the measured antenna vector. However, the resulting estimate is just a local minimum, i.e. the algorithm is not guaranteed to find the globally optimal position.

3.3 Artificial neural networks

In the following text, a subtype of artificial neural networks (ANNs) is explained from the most basic component, the neuron, to the full multi-layer perceptron (MLP) that will be evaluated for potential future usage in the RayPilot's recovery process.

3.3.1 Artificial neuron

The artificial neuron stems from attempts to mathematically model the behaviour of biological neurons. Mathematically, the artificial neuron model has a simple expression:

$$y(\mathbf{x}) = \varphi\left(\mathbf{w}\mathbf{x} + b\right) \tag{3.1}$$

where $\mathbf{x}$ are the inputs; $\mathbf{w}$ represents a weight for each input and represents that input's importance to the neuron; b is called the bias and models the neuron's tendency to fire; finally φ is a non-linear activation function which forms the response characteristic. Generally, the input, $\mathbf{x}$, is multidimensional and the output from the activation function, φ, is scalar. In the neuron, the outgoing signal is determined by a set of incoming signals. Thereby it performs a decision—a simple computation.

Equation 3.1 can be drawn schematically to illustrate the data flow, see Figure 3.4.

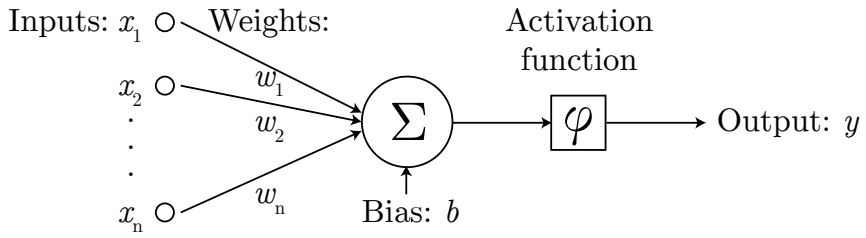


Figure 3.4: A schematic representation of Equation 3.1. The data flow is indicated by the arrows.

The choice of activation function can affect the learning abilities of the network [21]. Some common choices of activation functions are:

- Standard logistic function (sigmoidal)

$$\varphi(a) = g(a) = \frac{1}{1 + e^{-a}}$$

- Hyperbolic tangent (sigmoidal)

$$\varphi(a) = \tanh a = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

- Rectifier

$$\varphi(a) = \max(0, a)$$

Of special interest for the application at hand are the sigmoidal functions. These functions have the following characteristics; the domain $[-\infty, \infty]$ maps onto a finite range $[a, b]$, they are differentiable and the output is monotonically increasing. Why these properties are desirable is discussed in Section 3.3.4. The function plots in Figure 3.5 show these features qualitatively.

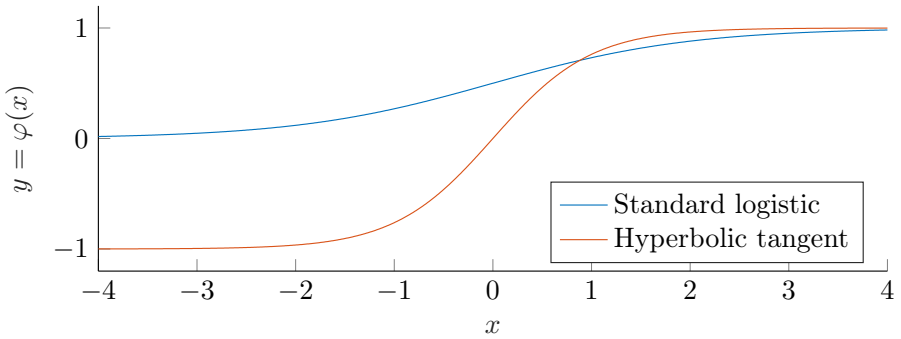


Figure 3.5: Sigmoidal functions *squash* the domain $[-\infty, \infty]$ onto a finite range $[a, b]$, are differentiable and monotonically increasing.

Remark:
The hyperbolic tangent and the standard logistic function are actually

equivalent except for scaling and offset, i.e. they share the same shape:

$$\begin{aligned}\tanh x &= \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^x(1 - e^{-2x})}{e^x(1 + e^{-2x})} \\ &= g(2x) - \frac{e^{-2x} + 1 - 1}{1 + e^{-2x}} = 2g(2x) - 1.\end{aligned}$$

3.3.2 Multilayer perceptron

The multi-layer perceptron (MLP) arranges multiple perceptrons (neurons), as introduced in Section 3.3.1, into a layered structure. The MLP is a *feed-forward* neural network, meaning that the neurons are nodes in a directed acyclic graph; there are other node topologies such as recurrent networks, these are not discussed here. Another property of MLPs is that the nodes in each layer are fully connected, meaning that every neuron in each layer is connected to every neuron in the adjacent layers, see Figure 3.6.

The MLP has applications in *regression*, i.e. continuous function fitting, and *classification*, i.e. labeling of data by means of discrete function fitting. The application at hand is an example of non-linear function approximation; hence, the MLP is here used as a regressor.

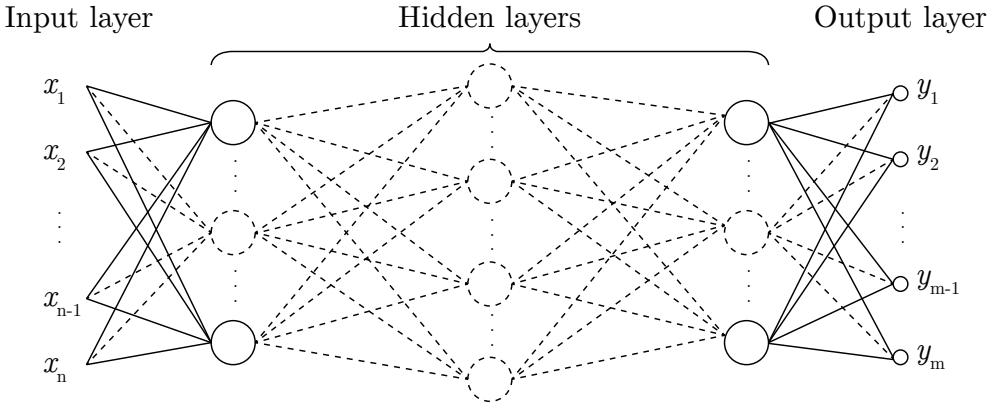


Figure 3.6: Neural network nomenclature. Each circle, *node*, represents a neuron and the lines show the interconnections between the neurons.

The inner layers, i.e. not in- or output layers, are called *hidden* layers; the neurons in such layers are consequently called hidden neurons. MLPs with exactly one hidden layer with sigmiodal activation functions are capable of approximating any function $\mathbb{R}^n \to \mathbb{R}^m$ [22]; this is discussed in more detail in Section 3.3.4.

In the following paragraphs, a terse notation for MLPs is introduced. The following, common mathematical conventions hold: bold, capital letters ($\mathbf{W}$) represent tensors; regular weight, capital letters (W) matrices; bold, lowercase letters ($\mathbf{w}$) vectors; and regular weight, lowercase (w) scalars.

To extend the mathematical notation for the neuron, introduced in Equation 3.1, into layers, the weight vector $\mathbf{w}$ becomes a weight matrix $W_{i,j}$ where i denotes the neuron and j the inbound connections; the biases turn into a vector $\mathbf{b}_i$ where

i once again represents the associated neuron. Using this notation, the whole layer can be computed in a single expression:

$$\mathbf{y} = \varphi\left(W\mathbf{x} + \mathbf{b}\right)$$

where the in- and outputs of the activation function are now vectors; the activation function is generalized to element-wise application. To further extend the model to handle multiple layers, the weights can be treated as a tensor, $\mathbf{W}_{l,i,j}$, the biases as a matrix, $B_{l,i}$, and activation functions as a vector of functions, φ_l. Using this notation a full MLP with n layers ($n-1$ hidden and one output) can be described as:

$$\mathbf{y} = \varphi_n\Big(\mathbf{W}_n\varphi_{(n-1)}\Big(\mathbf{W}_{n-1}\varphi_{(n-2)}\Big(\dots\varphi_1\left(\mathbf{W}_1\mathbf{x} + B_1\right)\dots\Big) + B_{(n-1)}\Big) + B_n\Big) \tag{3.2}$$

where the layers are applied successively, in a chained manner; the innermost entity is the input signal as applied to the first hidden layer (Equation 3.3) and the outermost is the output layer (Equation 3.5).

$$\mathbf{y} = \varphi_n\Big(\mathbf{W}_n\varphi_{(n-1)}\Big(\mathbf{W}_{n-1}\varphi_{(n-2)}\Big(\dots\underbrace{\varphi_1\left(\mathbf{W}_1\mathbf{x} + B_1\right)}_{1^{\text{st}}\text{ hidden layer}}\dots\Big) + B_{(n-1)}\Big) + B_n\Big) \tag{3.3}$$

$$\mathbf{y} = \varphi_n\Big(\mathbf{W}_n\underbrace{\varphi_{(n-1)}\Big(\mathbf{W}_{n-1}\varphi_{(n-2)}\Big(\dots\varphi_1\left(\mathbf{W}_1\mathbf{x} + B_1\right)\dots\Big) + B_{(n-1)}\Big)}_{(n-1)^{\text{st}}\text{ hidden layer}} + B_n\Big) \tag{3.4}$$

$$\mathbf{y} = \underbrace{\varphi_n\Big(\mathbf{W}_n\varphi_{(n-1)}\Big(\mathbf{W}_{n-1}\varphi_{(n-2)}\Big(\dots\varphi_1\left(\mathbf{W}_1\mathbf{x} + B_1\right)\dots\Big) + B_{(n-1)}\Big) + B_n\Big)}_{\text{output layer}} \tag{3.5}$$

To reiterate, the defining features for a full MLP are:

- $\mathbf{W}_{l,i,j}$ —weights for each layer l, node i and inbound signal j.
- $B_{l,i}$ —biases for each layer l and node i.
- φ_l —activation functions for each layer l.

Each layer may have a different number of neurons; hence, the weight tensor is *jagged*, i.e. for each layer $l = \lambda$ the associated weight matrix, $\mathbf{W}_\lambda = W_{i,j}$, may have different dimensions. The same thing applies to the biases; for each layer $l = \lambda$ we get a bias vector $B_\lambda = \mathbf{b}_i$ with one element for each neuron in the layer.

3.3.3 Training the network

The process of training a given network is that of finding the weights and biases that produce the best input-output mapping. To give a quantitative measure of the goodness of this mapping, a *loss function* is used. The loss function is a part of the design specification. Generally, this function is a positive function where a low

value indicates a good fit. In regression, a common choice for the loss function is the L_2-norm:

$$L_2(\mathbf{p}, \hat{\mathbf{p}}) = ||\mathbf{p} - \hat{\mathbf{p}}||_2 = \left(\sum_{i=1}^{n} (p_i - \hat{p}_i)^2 \right)^{\frac{1}{2}}.$$

where $\mathbf{p}$ is a set of known, n-dimensional outputs and $\hat{\mathbf{p}}$ is the corresponding set of estimated outputs by the network. Consequently, a network with a perfect fit produces a L_2 loss of 0. The training process aims to minimize the cost over the training set, i.e. minimize the loss function.

When using ANNs for regression, *supervised training* is performed. This is commonly done via a method called error back-propagation. Optimizing ANNs is an area of active research. Most optimization techniques are modified versions of back-propagation such as RPROP [23], Adadelta [24] and Adam-optimization [25]. There are also stochastic optimization methods such as particle swarm optimization [18]. The method used here is *off-line* training, where the network is optimized prior to usage; the alternative is a system which is trained, or updated, during usage which is useful in systems where the characteristics change over time.

The training procedure is commonly organized into epochs. In each epoch, every datum is presented to the ANN once. Figure 3.7 illustrates the progression at different stages of training. In the figure the estimation errors are shown as lines between each set point and the estimated position. After only 100 epochs the ANN produces reasonable estimates, however after 100 000 epochs the average radial error is as low as 0.453 mm.

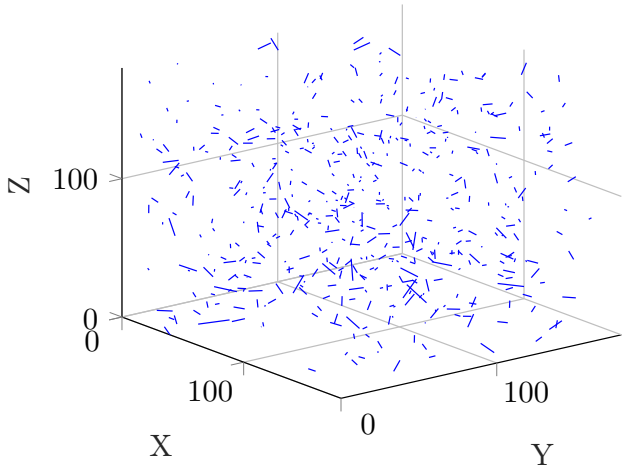
In general the trained ANN has the following defining features: topology, weights, biases and activation functions.

3.3.3.1 Error back-propagation

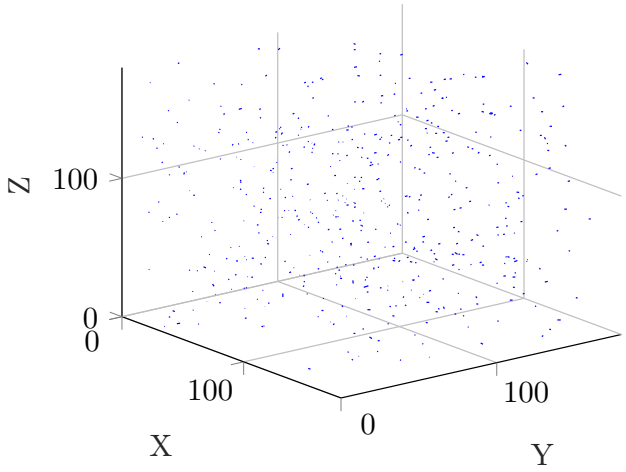
A common way to train MLP networks is the back-propagation family of algorithms. The back-propagation algorithm works in two phases: the forward pass which generates an estimated output $\hat{\mathbf{y}}$ for a given input, $\mathbf{x}$, and the backward pass which uses the error between the known and estimated output to adjust the weights and biases. The error is then propagated from the output layer through the hidden layers towards the first hidden layers. Corrections to the weights are made in order to reduce the error with respect to the known and estimated output pairs. The algorithm is quite involved and a thorough description is available in, among many others, the textbooks by Haykin [4, pp. 183–195] or Bishop [26, pp. 241–249].

3.3.3.2 Batch training

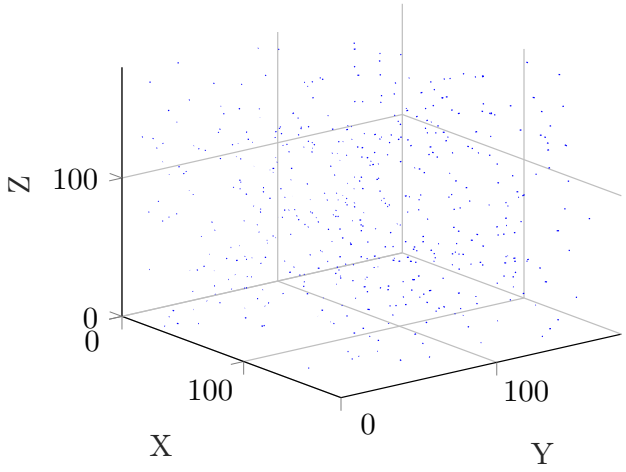
Both the standard back-propagation technique presented in [4, pp.183–195] and the ADAM-optimizer used in the TensorFlow implementation in this project can be used in one of two modes: sequential mode where the weights are updated after every example is shown and mini-batch mode where many, or all, examples are evaluated with the same weights and the weight updates are applied after the mini-batch. The mini-batch mode allows for more parallelism than the sequential mode, since multiple examples can be evaluated in parallel between the weight updates.



(a) 100 epochs, $\bar{\mathcal{E}}_r = 4.84\,\text{mm}$.



(b) 10 000 epochs, $\bar{\mathcal{E}}_r = 0.727\,\text{mm}$.



(c) 100 000 epochs, $\bar{\mathcal{E}}_r = 0.453\,\text{mm}$.

Figure 3.7: Estimation errors on novel test data for different amounts of training using the `DB476` database. Each line represents the deviation (x, y, z) between the set point and estimate

According to Haykin, it is good practice to randomize the order of the examples between epochs [4].

3.3.3.3 ADAM-optimization

In classic back-propagation a *learning rate* is chosen and is a scaling factor which applies to all weight updates. ADAM, which is short for adaptive moment estimation, keeps separate learning rates for each parameter in the model and additionally keeps two degrees of moments [25]; this supposedly improves the algorithm's performance in finding optimal weights, by reducing the risk of getting stuck in a local minimum. This algorithm has three parameters which can be customized the initial learning rate and two decay parameters for the first and second order moment estimates.

According to [25], this optimizer combines the advantages of two other popular back-propagation algorithms, AdaGrad [27] and RMSProp [28].

3.3.4 Function approximation with ANN

In a seminal paper by Hornik, it was rigorously proven that *"multilayer feedforward networks with as few as one hidden layer are indeed capable of universal approximation in a very precise and satisfactory sense"* [2]. Hence any $\mathbb{R}^m \rightarrow \mathbb{R}^n$-mapping can be approximated with one layer, including the mapping under study, which has the form $\mathbb{R}^{16} \rightarrow \mathbb{R}^5$. However, the proof in [2] is just that of an existence of such an approximation and gives no bounds of the required amount of neurons.

3.3.5 TensorFlow

The TensorFlow software library is an open-source toolkit targeting machine learning tasks. TensorFlow allows the programmer to give a high-level description of the problem and enables execution on heterogeneous, distributed platforms [29].

The library is developed by the Google Brain Team [30]. In this project, TensorFlow is used as a rapid prototyping tool for different network designs. The details are presented in Section 4.2.3. Most computations were performed on a high-end graphics card, Nvidia GTX1080TI. A comparison of the TensorFlow performance on different hardware used during the project is available in Section 5.2.3.2

4
Method

At the top level, the project is structured into three distinct phases:

1. Exploratory modeling to find a suitable network topology.
2. Implementation of the model in software.

In the first phase, the high-level TensorFlow library was applied on existing data sets provided by Micropos Medical and new sets generated during the project. The Tensorflow library allowed for rapid prototyping of neural networks of different configurations.

4.1 Choice of regression algorithm

It is probable based on the studies [3] and [14] that neural networks can be successfully implemented for the described purpose. In this section some further motivations for the use of this regression method are presented.

The polynomial regression method requires iterated function evaluation since it uses the *gradient descent* algorithm. The workstations used with the RayPilot system typically use 30% of its resources for the positioning while producing 30 estimates per second. The results from [3] highlight the fact that a trained ANN can make fast estimates. The neural network estimation is a set of matrix multiplications and applications of transfer functions. Hence the computation has a deterministic and fixed run time, as in the modified version of the gradient descent algorithm, GGAM—this is a desirable feature. Lower latency and higher estimation rate is of value for the product; it improves the system's ability to perform gating—the process where RayPilot signals to the linear accelerator's software that the prostate has deviated from the desired position.

4.2 Neural network implementation

The intention is to train a MLP to take the antenna inputs and produce an output that represents the position and angles of the transmitter. This is shown schematically in Figure 4.1.

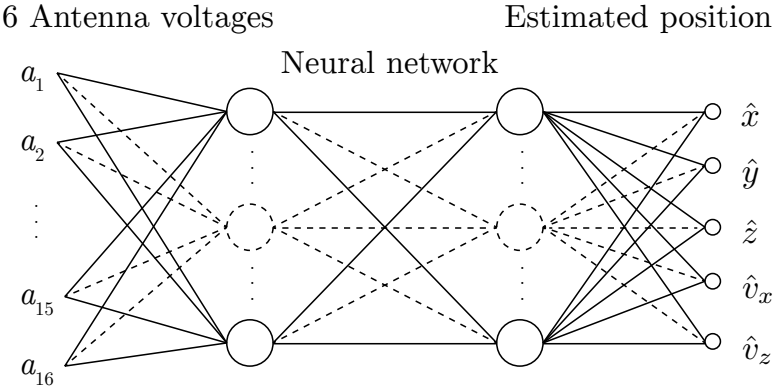


Figure 4.1: Schematic overview of the proposed positional recovery.

In order to create a MLP that performs well, as introduced in Section 3.3.2, some optimization on the construction of the network have to be made. The search for a suitable network topology is a nested problem, i.e. it is an optimization of an optimization; for each candidate topology the weights and biases needs to be optimized and the performance of the resulting estimator needs to be evaluated. Numerous training sessions will be performed to give a representation of each topology's performance. The parameters that govern the overall layout and training of the ANN are called *hyper-parameters* as opposed to the parameters that are associated with a specific layout, such as weights and biases. The procedure to determine the parameters is called *training* whereas the search for the topology is called a *hyper-parameter search*. In the following sections, the methods employed to solve these optimization problems are treated.

4.2.1 Data pre-processing

Before the details of the optimization methods can be introduced, some aspects of the data processing must be treated. The training data collected from the RayPilot test fixture has the properties listed in Table 4.1.

Table 4.1: The properties of the training data collected from the RayPilot test fixture.

		Data direction	
Property	**Data range**	**Training**	**Estimation**
Antenna voltages	1–3 V	Input	Input
x, y and z	0–300 mm	Input	Output
v_x and v_z	−45–45°	Input	Output

In order to gain efficient learning, the input data was normalized to the range of $[-1, 1]$. This procedure is also called *feature scaling*. The ranges of all input and output variables are known in advance from the training databases. The resulting model is only intended to give predictions within its trained volume; hence unseen

data will be confined to these intervals. The inputs are scaled as:

$$x_{\text{new}} = -1 + 2\frac{x - X_{\min}}{X_{\max} - X_{\min}}$$

where X denotes the set of all inputs, x, in the training data. The outputs are scaled correspondlingly:

$$y_{\text{new}} = -1 + 2\frac{y - Y_{\min}}{Y_{\max} - Y_{\min}}.$$

Another common procedure is to standardize the data, giving it zero mean and unit variance:

$$x_{\text{new}} = \frac{x - \bar{x}}{\sigma_X}.$$

Given that both the training databases and evaluation databases already have uniform distribution, normalization was deemed sufficient. The resulting dataflow is shown in Figure 4.2.

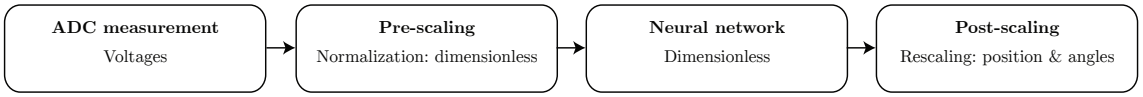


Figure 4.2: The stages of data processing and the interpretation of the data in each step.

Remark:

A late realization in the project was that the antenna measurements were done using a logarithmic analog-to-digital converter (ADC). Had this been known at an earlier stage, an investigation of different normalization methods on the inputs would have been warranted. However, the input normalization is mainly performed to aid the learning in the network; the self-learning abilities of the ANN should be able to adapt to the input, whether a good scaling is performed or not—as long as the information in the signals is preserved.

4.2.2 Network optimization

For any given network topology that is being evaluated, that ANN will need to be trained and performance measures have to be gathered, in order to compare the relative performance of different configurations. In Section 4.2.2.1, the implementation specific details of the training are presented. Later, the method of *hyper-parameter search* is defined, which is used to find an ANN configuration that performs well.

4.2.2.1 Training algorithm

The optimization goal is defined to be minimization of the squared error of each normalized output variable:

$$e_{\text{loss}} = \frac{1}{n}\sum_{i=1}^{n} |\hat{\mathbf{y}}_i - \mathbf{y}_i|^2$$

where the sum over n is each datum in the training set. The e_{loss} function appears to sum outputs of different units (mm and °), however in the normalization process, explained in Section 4.2.1, the outputs were made dimensionless. In the network representation, all quantities are dimensionless.

The process of weight and bias optimization is complex. TensorFlow offers a number of methods based on the classic back-propagation algorithm, see Section 3.3.3.1. The Adam optimization algorithm [25] was selected, since it is considered state of the art.

The activation function used in the hidden layers was optimized to enable good learning performance according to the general guidelines available in [4], [21]:

$$\varphi(a) = 1.7159 \tanh\left(\frac{2}{3}a\right).$$

This gives the activation function the range $[-1.7159, 1.7519]$ and the gain at zero is approximately unit-gain:

$$\varphi'(a) = \frac{d(c_1 \tanh c_2 x)}{dx} = c_1 c_2 \operatorname{sech}^2 c_2 a$$
$$\varphi'(0) = [\operatorname{sech} 0 = 1] = 1.7159\frac{2}{3} = 1.1424$$

This activation function is optimized for use with an input in the domain of $[-1, 1]$; better learning is obtained when the input data is contained inside the function domain with some padding [21]. Based on initial testing, linear activation functions were chosen for the output layer.

4.2.2.2 Weight and bias initialization

Prior to the first pass of the network optimization algorithm, each neuron has to be assigned an initial value. The approach chosen here is that proposed by Glorot and Bengio called Xavier initialization, where the neuron weights are initialized normally distributed with a variance dependent on the neuron's *fan-in* and *fan-out*, i.e. the number of incoming and outgoing connections, see Equation 4.1; their research has shown this initialization to be efficient in multilayer networks [31]. The biases are set to zero at initialization.

$$\text{Var}(\mathbf{w}) = \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}} \tag{4.1}$$

4.2.2.3 Holdout method

When training an ANN, input-output pairs are feed into the optimization algorithm used to iteratively adjust the weights and biases, to better reflect the relation between inputs and outputs. The aim is to gain a generalizing property where good predictions can be made from previously unseen inputs. To ensure this, the method of early stopping using a validation set is used [4]. When using the holdout method the input-output pairs are partitioned into three distinct sets:

- **Training set:** Used to adjust the network's parameters by, for example, error back-propagation methods (see Section 4.2.2.1).
- **Validation set:** Given a network in training, this data set is used to estimate how well the network generalizes by applying the model to new data. The average cost provides a measure on the correctness of the predictions.
- **Test set:** After the training when the best performing parameters (on the validation set) have been found the average cost on the test set is a measure on the trained network's performance.

4.2.2.4 Parallelism in the inference

The neurons in a feed-forward neural network are arranged into layers. The computations of each neuron in any given layer are independent and may therefore be executed in parallel. However the layers must be calculated sequentially since each layer is dependent on the input values from the previous layer. In Figure 4.3 the parallelism inside each node is visualized; and each node in every layer can be evaluated independent of the others, i.e. in parallel.

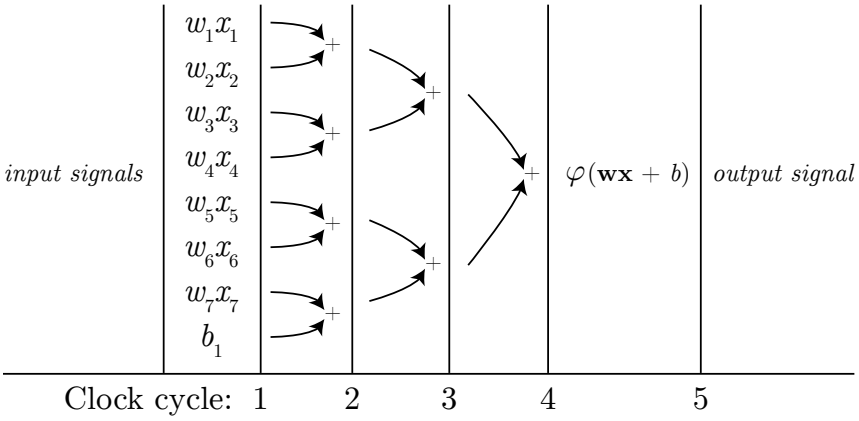


Figure 4.3: Internal parallelism in a neuron with 7 inputs which evaluates the function $\varphi(\mathbf{wx} + b)$

4.2.2.5 Parallelism in the training

The parallelism in the training includes the parallelism found in the inference, since each training example needs to be evaluated to generate the error which guides the back-propagation algorithm. At an even higher level of abstraction, each training example in a training batch, see Section 3.3.3.2, can be evaluated in parallel since the gradient based on the combined error of multiple training examples. In this project the training parallelism will be exploited by the use of a high-end graphics processing unit (GPU) and its many CUDA cores.

4.2.3 Hyper-parameter search

The hyper-parameters are the higher-level options that govern the layout and training of the network. The parameters of interest here are the number of nodes per layer, the number of layers, the training algorithm's learning rate and the batch size used during training.

The method employed here is called *grid-search*, in which all combinations of parameters, from a predefined set of values, are tested [32]. Visually, the individual parameter combinations will be vertices in a grid, hence the name. The algorithm in Listing 4 describes the procedure. If more than two parameters are to be searched, the algorithm generalizes straight forwardly; additional for-loops are added and the output becomes a tensor with as many dimensions as search parameters.

Algorithm 4: Hyper-parametser search, grid variation

```
Data: L_{1...l} is a list of desired number of layers & N_{1...n} ditto of nodes.
      Training, validation and test sets
Result: Performance measures for all the combinations of layers and nodes
let R be a p × n matrix intended to store some desired measurement
for i ← 1 to l do
    for j ← 1 to n do
        C_best will contain the best configuration
        foreach epoch do
            foreach mini-batch do
                perform training operation on training set
            end
            evaluate performance on validation set
            if validation result has improved then
                C_best ← network configuration
            end
        end
        use C_best to evaluate the test set
        store the performance metric in R_{i,j}
    end
end
```

The chosen search space is listed in Table 5.1; the number of hidden neurons per layer was chosen to be constant. In order to reduce the search space's dimensionality, the network configuration is treated separately from the batch size and learning rate. Moderating the number of nodes and layers aids the learning by reducing the weights and biases search space [4]. If too many nodes are used the network becomes prone to overfitting and if too few are used it might not be able to fully represent the underlying model, i.e. underfitting.

4.3 Implementation

The target platforms for the project are the software suite developed by Micropos Medical. In Figure 4.4 the overall organization of the languages used is presented.

The Python bindings for TensorFlow are used to design the neural network at a high level. The C♯ implementation is intended for integration into the software suite, while the C implementation was used for prototyping and standalone performance evaluations.

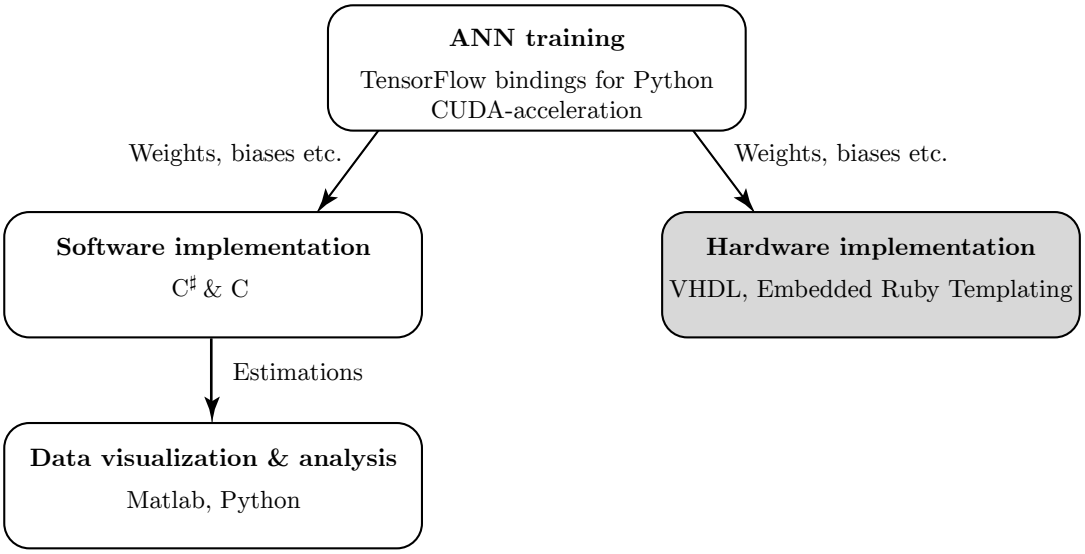


Figure 4.4: Languages used in the implementation and their domain. Training was performed on a GPU using while inference targeted central processing units (CPUs) and FPGAs.

4.3.1 ANN training in TensorFlow

In Listing 1, the single hidden layer MLP shown in Figure 4.5 is built from scratch in TensorFlow. Since TensorFlow is a dataflow language, and the gritty details are built-in functions, the example is quite compact.

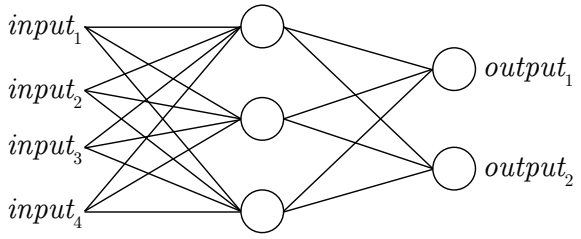


Figure 4.5: The toy MLP implemented in Listing 1.

Listing 1: A small MLP, shown in Figure 4.5.

```
NUM_FEATURES = 4
NUM_HIDDEN = 3
NUM_OUTPUTS = 2
X = tf.placeholder(tf.float32, [None, NUM_FEATURES])
Y = tf.placeholder(tf.float32, [None, NUM_OUTPUTS])

def model(x):
```

```
    # Activation function constants
    a = tf.constant(1.7159, dtype=tf.float32)
    b = tf.constant(2.0/3.0, dtype=tf.float32)
    hidden_weights = tf.Variable(tf.random_uniform([NUM_FEATURES, NUM_HIDDEN]))
    hidden_biases = tf.Variable(tf.zeros([NUM_HIDDEN))
    # 1.7159 * tanh( 2/3 * (W*x + b) ):
    hidden_layer = tf.scalar_mul(a, tf.nn.tanh(tf.scalar_mul(b,
↪    tf.add(tf.matmul(x, hidden_weights), hidden_biases))))

    output_weights = tf.Variable(tf.random_uniform([NUM_HIDDEN, NUM_OUTPUTS]))
    output_biases = tf.Variable(tf.zeros([NUM_OUTPUTS))
    return tf.add(tf.matmul(hidden_layer, output_weights), output_biases)

Y_hat = model(X)

loss = tf.nn.l2_loss(tf.subtract(Y, Y_hat))
train_op = tf.train.AdamOptimizer().minimize(loss)

sess = tf.Session()
sess.run(tf.global_variables_initializer())

# Some training data
trainx = [[1,2,3,4],...,[2,3,4,5]]
trainy = [[2,3],...,[1,2]]
# Some validation data
validx = [[1,3,3,4],...,[2,4,4,5]]
validy = [[2,4],...,[2,2]]

for i in range(0,10): # Train 10 epochs
    sess.run(train_op, feed_dict = {X: trainx, Y: trainy})

# Validation cost:
cost = sess.run(loss, feed_dict={X: validx, Y: validy})
print("%g" % (float(cost)))
```

A variation of the TensorFlow-script used throughout the project is listed in Listing 9 in Appendix D.2. This program performs training, validation and outputs performance statistics on the test set to a text file.

To get some qualitative feedback during the training, performance statistics were printed every 100th epoch, see Listing 2; the first figure is the number of epochs followed by statistics on the test set, using the currently best weights and biases, and finally the elapsed time since the previous output.

Listing 2: Example output of TensorFlow training session.

```
0   cost: 3620.38  E_rad: 64.4913   E_vx: 19.9328  E_vz: 15.0677  dur: 0s
100 cost: 89.7535  E_rad: 9.58531   E_vx: 2.35122  E_vz: 2.62714  dur: 27s
200 cost: 44.1303  E_rad: 6.83425   E_vx: 1.57363  E_vz: 1.92475  dur: 23s
300 cost: 28.3651  E_rad: 5.40191   E_vx: 1.39835  E_vz: 1.5074   dur: 22s
400 cost: 20.1648  E_rad: 4.80712   E_vx: 1.12527  E_vz: 1.26142  dur: 22s
...
```

The version used throughout this project was TensorFlow 1.0 running under Ubuntu 16.04LTS. TensorFlow was compiled from source and optimized for NVIDIA Compute Capability 6.1 to match the NVIDIA GTX1080TI's specifications.

4.3.2 Software implementation

At the end of a TensorFlow training session, the weights and biases are stored to files in javascript object notation (JSON) format. Small Python programs were written to generate C header files from these JSON records. Example outputs are shown in Listing 3 and 4. The code listed here was also ported with very minor adjustments to C♯ .

Listing 3: `biases.h`.

```
// Generated from DB475
const double bias_0[250] = {-0.018169883638620377, 0.023609327152371407,
 ↪  0.0014619482681155205, 0.019450053572654724...};
const double bias_1[250] = {-0.018169883638620377, 0.023609327152371407,
 ↪  0.0014619482681155205, 0.019450053572654724...};
const double bias_2[250] = {-0.018169883638620377, 0.023609327152371407,
 ↪  0.0014619482681155205, 0.019450053572654724...}
```

Listing 4: `weights.h`.

```
// Generated from DB475
#define INPUT_NODES 16
#define OUTPUT_NODES 5
#define HIDDEN_NODES 250
const double weight_0[16][250] = {{0.13025599718093872, -0.05873103439807892,
 ↪  -0.059857774525880814, -0.052826669067144394...}...};
const double weight_1[250][250] = {{0.13025599718093872, -0.05873103439807892,
 ↪  -0.059857774525880814, -0.052826669067144394...}...};
const double weight_2[250][5] = {{0.13025599718093872, -0.05873103439807892,
 ↪  -0.059857774525880814, -0.052826669067144394...}...};
```

Before the training each database was normalized using a program written in Ruby. The normalization procedure was described in Section 4.2.1. The program also generates a C header file containing the scaling factors. An example output is shown in Listing 5.

Listing 5: `scaling.h`.

```
// Generated from DB475
const double output_base[5] = {0.0, 0.0, 0.0, -40.0, -40.0};
const double output_scale[5] = {180.0, 180.0, 180.0, 80.0, 80.0};
const double input_base[16] = {1.096824, 1.085655, 1.297084, 1.209919, 0.748555,
 ↪  0.752851, 0.767847, 0.679667, 0.758084, 0.716923, 0.746837, 0.731763,
 ↪  1.064957, 1.056678, 1.05043, 0.95358};
const double input_scale[16] = {1.25397, 1.308722, 1.131815, 1.203515, 1.917314,
 ↪  1.875137, 1.927233, 1.89435, 1.869357, 1.946759, 1.891148, 1.916688,
 ↪  1.223041, 1.211247, 1.32364, 1.324343};
```

The estimation program is written as a standard UNIX text stream processor operating on `stdin` and `stdout` [33]. This design decision provided flexibility. The training and evaluation databases generated by the Autosetups differed in formatting, which was easily corrected by use of the UNIX `cut` command, as shown in Listing 6.

Listing 6: UNIX style pipelines enables chaining of simple commands to accomplish more complex tasks .

```
# In the training database there are 5 fields preceding the antenna values
$ cut -f6-21 DB476.tsv | ./forward_network_DB476 > DB476_estimated.tsv

# In the evaluation files there are even more fields
$ cut -f14-29 DB476_eval.tsv | ./forward_network_DB476 > DB476_eval_estimated.tsv

# The program can be parallelized using GNU Parallel
# Here the input is split into chunks of 250 records
# The output order is not conserved
# The speedup is almost linear but is mainly useful for offline estimations
$ cut -f6-21 DB476.tsv | parallel --pipe -L250 ./forward_network_DB47 >
 ↪  DB476_estimated_parallel.tsv
```

Listing 7 is a full forward implementation of a fictional example network. Even though the training and design of a neural network is complex, the inference program is surprisingly straight forward.

Listing 7: `forward_network.c`

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "weights.h"
#include "biases.h"
#include "scaling.h"

double activation(double output) {
  return 1.7159 * tanh(2.0/3.0 * output);
}

void antenna2network(double input[INPUT_NODES], double output[INPUT_NODES]) {
  // From raw antenna values to [-1,1], constants declared in scaling.h
  for (int i = 0; i<INPUT_NODES; i++)
    output[i] = -1.0 + 2.0 * (input[i] - input_base[i]) / input_scale[i];
}

void network2position(double input[OUTPUT_NODES], double output[OUTPUT_NODES]) {
  // From network [-1,1] to position, constants declared in scaling.h
  for (int i = 0; i<OUTPUT_NODES; i++)
    output[i] = 0.5 * output_scale[i] * (input[i] + 1) + output_base[i];
}

void layer(int node_inputs, int nodes,
    const double inputs[node_inputs],
    const double weights[node_inputs][nodes],
    const double biases[nodes],
    double output[nodes],
    int output_layer /* true values indicates special case for output layer */
    ) {
  for (int node = 0; node < nodes; node++) {
    double input_sum = biases[node]; // Add bias
    // For each node in the layer: sum up the weighted inputs
```

```
    for (int node_input = 0; node_input < node_inputs; node_input++)
      input_sum += weights[node_input][node] * inputs[node_input];
    // Apply activation function if hidden layer
    output[node] = output_layer ? input_sum : activation(input_sum);
  }
}

int main() {
  // Declaration of intermediate signals between layers
  double input_antennas[INPUT_NODES], scaled_antennas[INPUT_NODES],
 ↪  out_1[HIDDEN_NODES], out_2[HIDDEN_NODES], prediction[OUTPUT_NODES],
 ↪  position[OUTPUT_NODES];
  char *line = NULL; //
  size_t len = 0;
  while(getline(&line, &len, stdin) != -1) {
    for (size_t i = 0; i < len; i++) // Replace all commas with dots
        line[i] = (line[i] == ',') ? '.' : line[i];

    // Parse antenna inputs
    sscanf(line, "%lf...\t%lf", &input_antennas[0], ... , &input_antennas[15]);

    antenna2network(input_antennas, scaled_antennas); // Pre-scaling
    // Sequentially apply the layers to the input signal
    layer(INPUT_NODES, HIDDEN_NODES, scaled_antennas, weight_0, bias_0, out_1, 0);
    layer(HIDDEN_NODES, HIDDEN_NODES, out_1, weight_1, bias_1, out_2, 0);
    layer(HIDDEN_NODES, OUTPUT_NODES, out_2, weight_2, bias_2, prediction, 1);
    network2position(prediction, position); // Post-scaling

    for (size_t i = 0; i < len; i++) // Replace newline with EOL
      line[i] = (line[i] == '\n' || line[i] == '\r') ? '\0' : line[i];
    // Append estimated position
    printf("%s\t%f\t%f\t%f\t%f\t%f\n", line, position[0], position[1],
 ↪  position[2], position[3], position[4]);
  }

  free(line);
  exit(EXIT_SUCCESS);
}
```

4.4 Evaluation

Evaluation of the system is based on a series of benchmarks that have been defined as previous work by the company. They include positional and angular recovery in a range of different measurement volumes. The results of these benchmarks form a precision baseline for any new implementation. Proposed performance indicators are for each of the five variables:

- Mean error
- Standard deviation; or rather the sampled 95th percentile error
- Maximal observed error

The RayPilot targets sub-millimeter localization. With the current algorithm this limits the recoverable volume, see definition in Section 3.1.2, to around $\pm 5\,\text{cm}$ with the center chosen at the typical height of the patient's prostate. The ultimate goal would be to extend this volume to $\pm 15\,\text{cm}$, giving the system more flexibility with regards to patient positioning, and enabling it for use in breast cancer, for example.

For evaluation of trained estimators, new data is introduced, called the test set, which is generated by the robotized fixture and contains stored actual position, angles and antenna values. The trained MLP is then used to generate estimations creating pairs of known and estimated positions and angles. The exact definitions and evaluation metrics are introduced in Section 4.4.1.

4.4.1 Evaluation measures

In the following text, a number of definitions are made; great care has been taken to use these throughout the rest of this thesis, in order to make the text succinct and precise.

The *physical/known position*:

$$\mathbf{p} = [x, y, z, v_x, v_z] = [p_1, p_2, p_3, p_4, p_5].$$

The *estimated position*:

$$\hat{\mathbf{p}} = [\hat{x}, \hat{y}, \hat{z}, \hat{v}_x, \hat{v}_z] = [\hat{p}_1, \hat{p}_2, \hat{p}_3, \hat{p}_4, \hat{p}_5].$$

The *radial error*:

$$\mathcal{E}_r = \mathcal{E}_r(\mathbf{p}, \hat{\mathbf{p}}) = \sqrt{\mathcal{E}_x^2 + \mathcal{E}_y^2 + \mathcal{E}_z^2} = \sqrt{|x - \hat{x}|^2 + |y - \hat{y}|^2 + |z - \hat{z}|^2}.$$

Remark:

The radial error has a greater or equal magnitude than the largest error of the axial components:

$$\mathcal{E}_r \geq \max\left(\mathcal{E}_x, \mathcal{E}_y, \mathcal{E}_z\right)$$

since if we assume $\mathcal{E}_x$ to be larger than $\mathcal{E}_y$ and $\mathcal{E}_z$ then

$$\begin{aligned} \mathcal{E}_r^2 - \mathcal{E}_x^2 &= \mathcal{E}_x^2 + \mathcal{E}_y^2 + \mathcal{E}_z^2 - \mathcal{E}_x^2 \\ &= \mathcal{E}_y^2 + \mathcal{E}_z^2 \geq 0 \\ &\implies \mathcal{E}_r^2 \geq \mathcal{E}_x^2 \\ &\iff \mathcal{E}_r \geq \mathcal{E}_x \end{aligned}$$

hence, $\mathcal{E}_r$ is larger than the largest axial error, the same thing holds for $\mathcal{E}_y$ and $\mathcal{E}_z$ as largest axial error. In some graphs $\mathcal{E}_x$, $\mathcal{E}_y$ and $\mathcal{E}_z$ will be signed, other times it will indicate their magnitude; this should be clear from the context.

The *angular error* around the x-/z-axis:

$$\mathcal{E}_{v_x} = |\hat{v}_x - v_x|, \quad \mathcal{E}_{v_z} = |\hat{v}_z - v_z|.$$

The *mean radial error*, given n physical ($\mathbf{p}_i$) and estimated position ($\hat{\mathbf{p}}_i$) pairs:

$$\bar{\mathcal{E}}_r = \frac{1}{n}\sum_{i=1}^{n} \mathcal{E}_r(\mathbf{p}_i, \hat{\mathbf{p}}_i).$$

Mean errors for the angular errors ($\bar{\mathcal{E}}_{v_x}$, $\bar{\mathcal{E}}_{v_z}$) are defined analogously.

To study the worst case performance the max(·)-measure is used. Given a number of $\mathbf{p}$ and $\hat{\mathbf{p}}$ pairs, max($\mathcal{E}$) returns the largest error with regards to the error measurement $\mathcal{E}$.

The 95th percentile, p95(·), is used to give a measure on the typical performance, disregarding outliers. Given a number of $\mathbf{p}$ and $\hat{\mathbf{p}}$ pairs and an error measure $\mathcal{E}$, p95($\mathcal{E}$) is found by applying the error measure to the pairs, sort the resulting error values from low to high error into a list and selecting the value 95% into the sorted list.

The L2 cost function:

$$L_2(\hat{\mathbf{p}}) = ||\mathbf{p} - \hat{\mathbf{p}}||_2 = \left(\sum_{i=1}^{5} (p_i - \hat{p}_i)^2\right)^{\frac{1}{2}}.$$

This measure is the five-dimensional Euclidean distance between the estimate and the physical position. When this measure is used to compare the performance of different estimations, the square root is often dropped since

$$a, b \in \mathbb{R}^+ : \sqrt{a} < \sqrt{b} \iff a < b.$$

One example of this is the TensorFlow `tf.nn.l2_loss` function.

4.4.2 Verification

To verify the results of the position estimators, their performance is measured on novel data sets with random, uniformly distributed points spanning the measurement volume. This procedure is done off-line, i.e. data is collected in the same fashion as the generation of the training databases, and then the trained estimator (be it C or C♯) is applied to generate predictions of the output variables. The characteristics of the predictions will then be studied in order to determine the estimators performance or, in the case of a design or experimental failure, find any systematic errors. The verification performed in this report is intended to verify that the estimators produce sound estimations. This is separate and different to the company's verification procedures and regulatory routines.

5
Results

This chapter presents the outcomes from the modelling phase. Performance is evaluated according to the measures described in Section 4.4, and compared to the company's existing algorithm.

5.1 Network modelling

This section presents the results from the software based search for a well performing network configuration. This includes the hyper-parameter search and inspection of inference quality of the resulting estimators.

5.1.1 Hyper-parameter search

Separate hyper-parameter searches were performed for the data sets with different volumes. The rationale is that the larger volumes contain all the information of the standard volume and cover a $1.8^3 \approx 5.8$ (`DB476`) respective $3^3 \cdot 1.125^2 \approx 34.2$ larger (`DB1071`) data space. The expectation is that the larger data space will require more nodes, in comparison to the standard volume, to provide as good fit to the data.

5.1.1.1 Layers and nodes, standard volume

The search for a well performing configuration of nodes and layers was a grid search. The search space is defined in Table 5.1. While the variables spanning the search space were altered, all others where kept constant; a listing of the constant parameters is available in Table B.1 and B.2 in the Appendix. A word on notation, since the configurations are limited to constant number of nodes per hidden layer the notation layers $\times$ neurons is introduced; the number of neurons are hidden neurons per hidden layer.

Table 5.1: Hyper-parameters and bounds for the associated search space.

Parameter	Range
Nodes per layer	10–200, increments of 10
Layers	1–15
Batch size	50, 100, 250, 500, 750, 1000
Learning rate	0.0001, 0.0005, 0.001, 0.005, 0.01

In Figure 5.1, the average radial error is plotted as a function of layers and nodes per layer; 8 hidden layers of 130 nodes performed best with $\bar{\mathcal{E}}_r = 0.452\,\text{mm}$. The best performing combinations seem to form a valley along the combinations with around 1000 nodes: lowest $\bar{\mathcal{E}}_{v_x} = 0.581°$, $5 \times 180 = 900$; lowest $\text{p95}(\mathcal{E}_{v_x}) = 1.19\,\text{mm}$ $7 \times 160 = 1120$; lowest $\bar{\mathcal{E}}_r = 0.452\,\text{mm}$ and $\text{p95}(\mathcal{E}_r) = 0.801\,\text{mm}$, $8 \times 130 = 1040$. The v_z errors were smallest with a 14×130 configuration yielding $\bar{\mathcal{E}}_{v_z} = 0.4°$ and $\text{p95}(\mathcal{E}_{v_z}) = 0.964°$. With regards to the $\max(\cdot)$-measure the results were too noisy to draw any conclusions; alas, the lowest maximum radial error, $\max(\mathcal{E}_r)$, was 2.11 mm using 15 hidden layers of 90 nodes. Similarly, the $\text{p95}(\mathcal{E}_r)$ was minimized with 8 hidden layers of 130 nodes, resulting in 0.452 mm. All statistics are based on a novel test set of 4000 examples. Exact values for each set of parameters are available in Table A.1 in the Appendix.

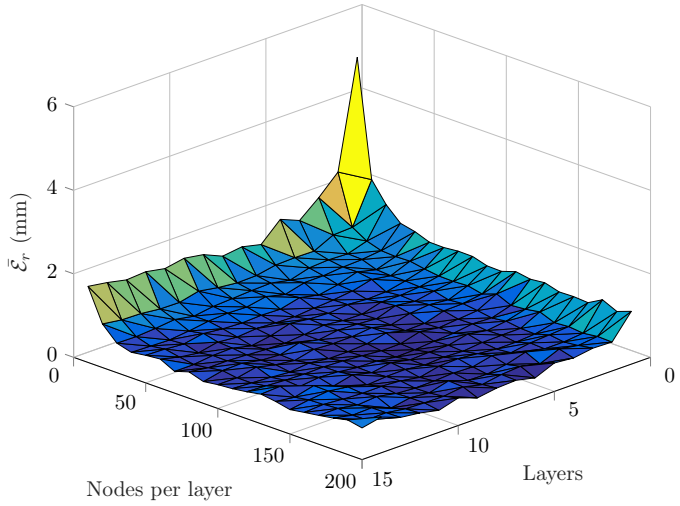
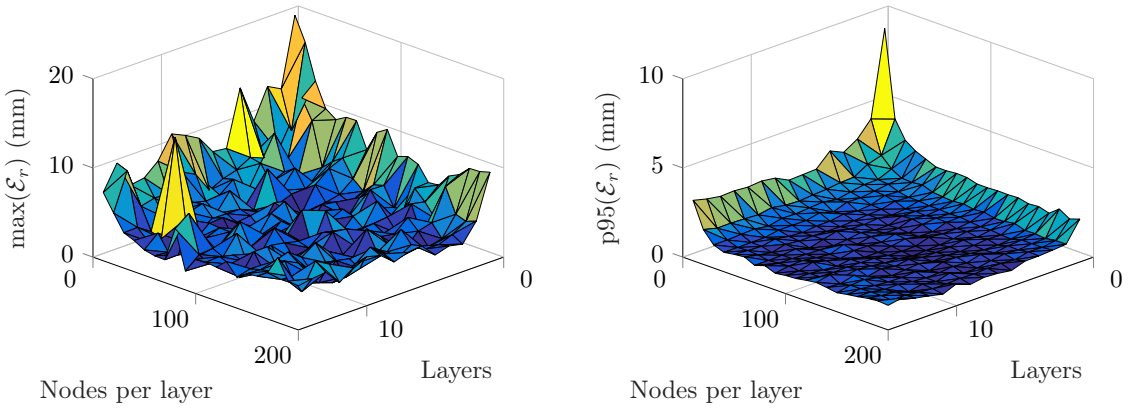


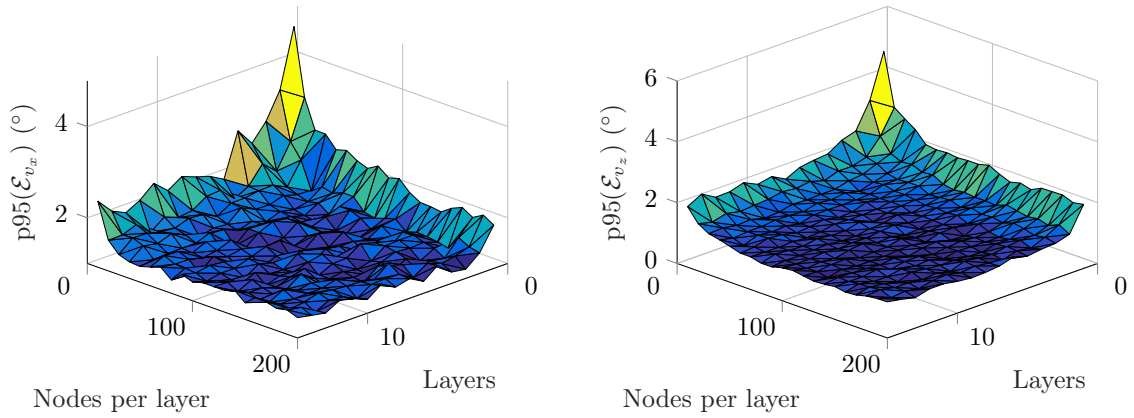
Figure 5.1: Average radial error as function of layers and nodes per layer.



(a) Maximal radial error as function of layers and nodes per layer.

(b) The radial error at the 95th percentile as function of layers and nodes per layer.

Figure 5.2: The maximum and 95th percentile radial errors as functions of hidden layers and nodes per layer.



(a) p95($\mathcal{E}_{v_x}$) was minimized with 7 hidden layers of 160 nodes, 1.19°.

(b) p95($\mathcal{E}_{v_z}$) was minimized with 14 hidden layers of 130 nodes, 0.964°.

Figure 5.3: p95(·)-errors for the angles as functions of hidden layers and nodes per layer.

Conclusion:

The optimal configuration for the 100 mm volume was found to be 8 hidden layers with 130 nodes each; this is based on the fact that the the p95(·)-measure for radial errors is the most important metric.

5.1.1.2 Layers and nodes, extended volume

While the chosen configuration for the standard volume performed well, it did not handle the expanded volume as well. This led to a new exploration to find more suitable layer/node configurations for the 180 mm and 300 mm data sets. Under the assumption that the extended volume would require a larger network than the standard volume, the search space in Table 5.2 was chosen. Additional details on the configuration are listed in Table B.3. In Table A.2 in the Appendix, the performance for each parameter set is listed.

Table 5.2: Hyper-parameters search space for the extended volume.

Parameter	Range
Nodes per layer	100–350, increments of 25
Layers	8, 10, 12, 14

Remark:

In order to better visualize the results of the test at hand, the graphs are shown from a different perspective from the graphs in section 5.1.1.1

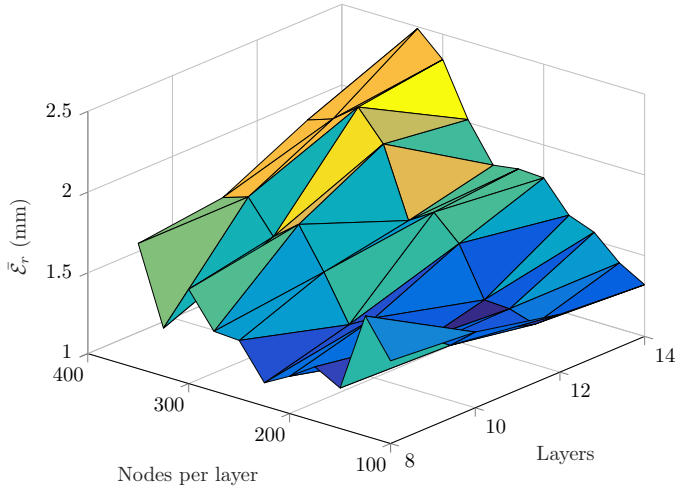
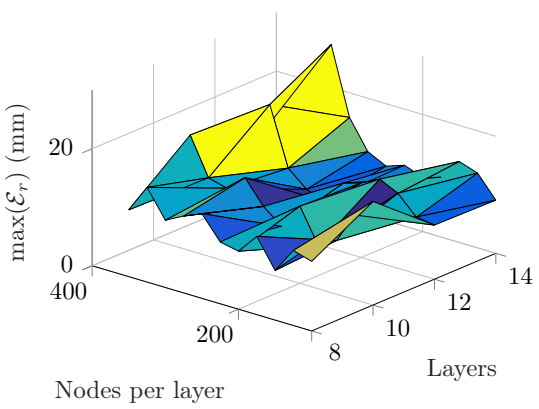
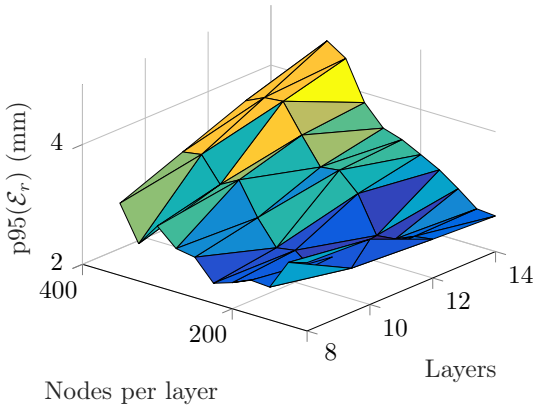


Figure 5.4: Mean radial error as function of layers and nodes per layer.

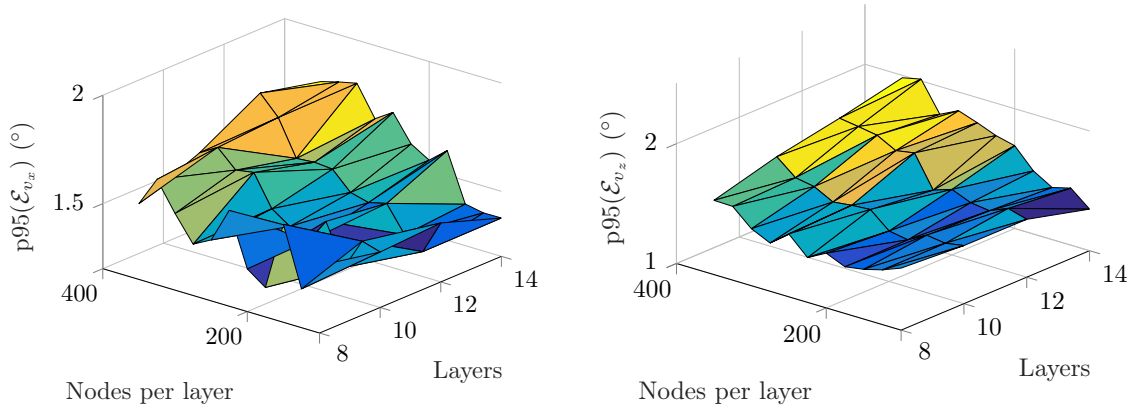


(a) The maximal radial error as function of layers and nodes per layer.



(b) The radial error at the 95th percentile as function of layers and nodes per layer.

Figure 5.5: The maximum and 95th percentile radial errors as functions of hidden layers and nodes per layer.



(a) The p95($\mathcal{E}_{v_x}$)-measure as function of layers and nodes per layer.

(b) The p95($\mathcal{E}_{v_z}$)-measure as function of layers and nodes per layer.

Figure 5.6: p95(·)-errors for the angles as functions of hidden layers and nodes per layer.

With regards to the average radial error, 8 hidden layers of 225 nodes performed best with $\bar{\mathcal{E}}_r = 1.14\,\text{mm}$. The lowest maximum radial error, $\max(\mathcal{E}_r)$ was 7.40 mm using 10 hidden layers of 175 nodes. The radial error at the 95th percentile was minimized with 10 hidden layers of 175 nodes, resulting in 2.29 mm.Best performance with regards to p95($\mathcal{E}_{v_x}$) was achieved with 12 layers of 125 hidden nodes, resulting in 1.31°. Best performance with regards to p95($\mathcal{E}_{v_z}$) was achieved with 14 layers of 100 hidden nodes, resulting in 1.35°.

The well performing networks had configurations of $8 \times 225 = 1800$, $10 \times 175 = 1750$, $12 \times 125 = 1500$ and $14 \times 100 = 1400$.

Conclusion:

For the 300 mm volume, 10 hidden layers with 175 nodes each performed best, based on the p95(·)-measure for radial errors. Generally, the larger volume required a larger node count as expected. This is roughly:

$$\frac{8}{8} \cdot \left(\frac{225}{130}\right)^2 \approx 3.0$$

times the number of weights, which represents the models' degrees of freedom.

5.1.1.3 Batch size

The batch size, introduced in Section 3.3.3.2, affects the performance of the training. In Figure 5.7, the same network has been trained five times per configuration, listed in Table 5.3. The cost function is plotted against the number of elapsed epochs in Figure 5.7.

Table 5.3: Hyper-parameters search space for the batch size.

Parameter	Range
Batch size	50, 100, 250, 500, 750, 1000

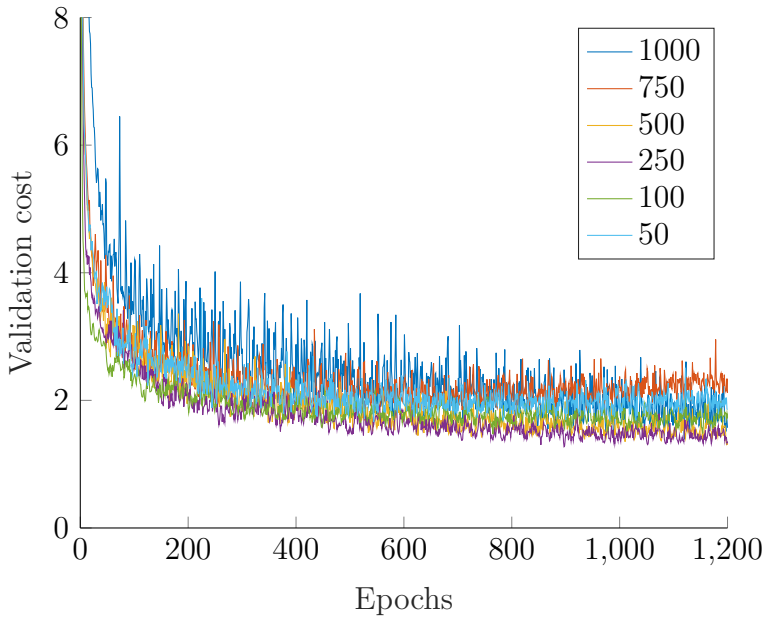


Figure 5.7: The cost function over 1200 epochs. The curves represent batch sizes 1000, 750, 500, 250, 100 and 50, averaged over five separate runs each. Best performance, both with regards to convergence rate and minimization of the cost function, was achieved with mini-batches of 250 samples.

Remark:

When the GPU acceleration was introduced, it mandated larger batch sizes to utilize the available parallelism. The result was slower convergence per epoch, but the improvement in speed by far outweighed this loss.

Conclusion:

A batch size of 250 samples performed best on the standard sized training set

5.1.1.4 Learning rate

ADAM optimization has an adaptive learning rate which is maintained per neuron; however, a start value is supplied in the TensorFlow implementation. The learning rate is a factor which scales how large the correction shall be in the training algorithm, i.e. it scales the step size used when iteratively updating the neurons. The parameter configurations which were evaluated are listed in Figure 5.4 and the results are plotted in Figure 5.8.

Table 5.4: Hyper-parameters search space for the learning.

Parameter	Range
Learning rate	0.0001, 0.0005, 0.001, 0.005, 0.01

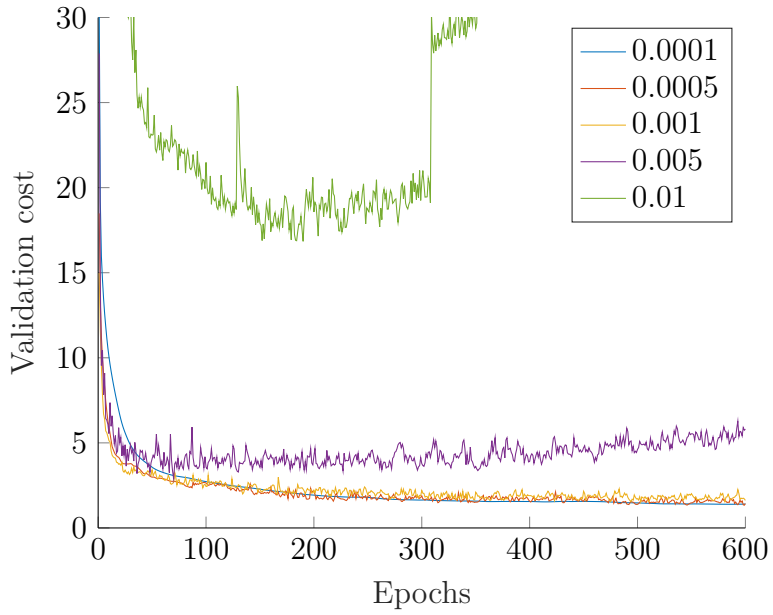


Figure 5.8: The cost function over 600 epochs for different learning rates. The curves represent batch sizes 0.0001, 0.0005, 0.001, 0.005 and 0.01, averaged over five separate runs each. Best performance, both with regards to convergence rate and minimization of the cost function, was achieved with a learning rate of 0.0005.

Conclusion:

A learning rate of 0.0005 was found to perform best on the standard sized data set. However, during the experiments 0.0001, 0.00025 and 0.0005 has been used with similar performance.

5.2 Performance

To reiterate, the performance goals defined in Section 1.3 were:

- Maintaining or improving the accuracy of the positional predictions, see Section 4.4 for further details
- Extending the recoverable volume as defined in Section 3.1.2
- Decreasing the latency of the predictions; the current algorithm has a minimum latency of about 10–20 ms. Previous studies has already shown that an ANN solution can reduce the latency [3]

Each of these performance oriented goals will be treated in the following sections. Where applicable, there will be comparisons to the polynomial model.

5.2.1 Accuracy

This sections deals with the goal to maintain or improve the accuracy of the positional and angular predictions. The first subsection compares the two methods in

the standard volume. In the two following subsections, properties of the errors are visualized and discussed.

5.2.1.1 Comparison with previous model

A central question is whether the ANN-based estimator can outperform the existing implementation, $\mathcal{P}_4$. Here the same database, `DB271`, is used to make a fair comparison between the algorithms. The standard procedure for the $\mathcal{P}_4$ model is to train it using a data set with the extent of `DB271` but evaluate the performance on a data set with restricted angles ($< 20°$); this gives the model some overhead in the evaluation process since the boundaries of the test set is in the interior of the training set. This process is replicated here and the same test set is used. The training of the ANN uses a separate set as validation data.

Table 5.5: Performance metrics for the standard volume. The performance statistics are for a test set of 2000 points with novel data with limited angles $< 20°$.

Model	$\bar{\mathcal{E}}_r$	p95($\mathcal{E}_r$)	max($\mathcal{E}_r$)	p95($\mathcal{E}_{v_x}$)	p95($\mathcal{E}_{v_z}$)
$\mathcal{P}_4$	0.670 mm	1.693 mm	8.530 mm	1.839°	1.996°
ANN 10×70	0.389 mm	0.690 mm	2.350 mm	1.525°	1.053°
ANN 10×70 / $\mathcal{P}_4$	58.1 %	40.8 %	27.5 %	82.9 %	53.6 %

Conclusion:

As demonstrated in Table 5.5 the ANN-based estimator clearly outperforms the $\mathcal{P}_4$ estimator with regards to all relevant performance metrics. The most important metric p95($\mathcal{E}_r$) is reduced by 59 %.

5.2.1.2 Spatial error distribution

An interesting aspect of the estimator is whether the errors have a spatial dependence. In order to visualize this, plots were made from the test sets, where a line connects the set point's (x, y, z)-position and the estimates $(\hat{x}, \hat{y}, \hat{z})$. The lengths and orientations of the lines would illustrate and highlight any systematic deviations. An example for the `DB476` database is shown in Figure 5.9, where a test set of 20 000 novel data points was used. The errors are evenly distributed and do not have any obvious spatial correlation. This is explored in more detail in the error correlation plots in Figure 5.11.

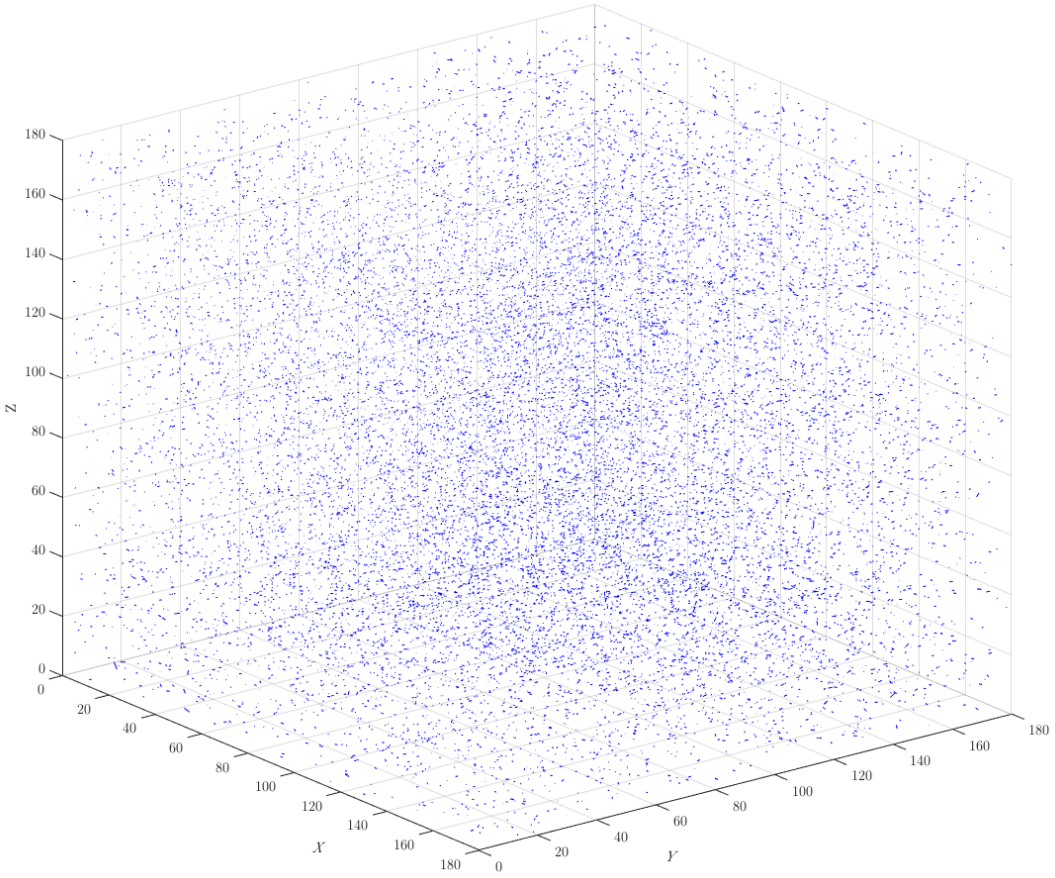


Figure 5.9: Spatial error distribution for a 10×250 ANN trained on database `DB476`. Each line connects a set point and the corresponding estimate.

Conclusion:

The lack of a spatial dependence validates the choice of method—it is capable of providing a good function approximation. The errors are only marginally larger close to the boundary than in the interior of the test set; hence, measurement volumes larger than 180 mm are plausible.

5.2.1.3 Error distributions, correlation

The error distributions give insight into the typical performance of the algorithm. Figure 5.10 shows the magnitude of the error in each dimension plotted as distributions. Additionally, the radial error is shown and has a different shape than the other curves. The explanation is simple; for the radial error to be small, each error in x, y and z has to be small simultaneously, which is unlikely. Hence, the radial errors have a different distribution than the axial errors.

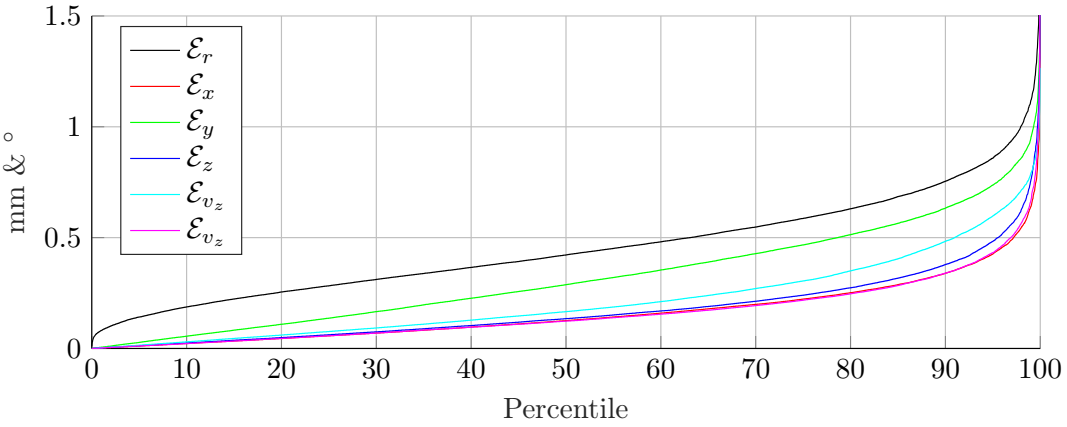


Figure 5.10: Error magnitude distributions per dimension and radially for `DB476`. About 98% of the test points have a radial error of less than 1 mm.

In figure 5.11, the errors in each dimension and the position in each dimension are shown as scatter plots. Each plot has the Pearson correlation coefficient [34] shown next to the x-axis; this is a measure of each position's linear relationship to the error in the associated dimension. The pairs having some linear dependence are $(x, \mathcal{E}_x)$, $(v_x, \mathcal{E}_y)$, $(v_x, \mathcal{E}_{v_x})$, $(z, \mathcal{E}_z)$ and $(y, \mathcal{E}_y)$; this indicates that there are still some improvements to make with regards to the goodness of fit. The peculiar shapes in the $(v_x, \mathcal{E}_{v_x})$ and $(v_z, \mathcal{E}_{v_z})$ plots are likely caused by the the fact that the training set only contains angles such that $v_x \in -40, -32, -24, -16, -8, 0, 8, 16, 24, 32, 40$; at these points the error is smaller and between the points larger.

Additionally, rows four and five in Figure 5.11 show that the positional recovery is stable with regards to different implantation angles; i.e. the performance is not degraded if the transmitter is implanted with a slight angle ($< 40°$) measured against the y-axis.

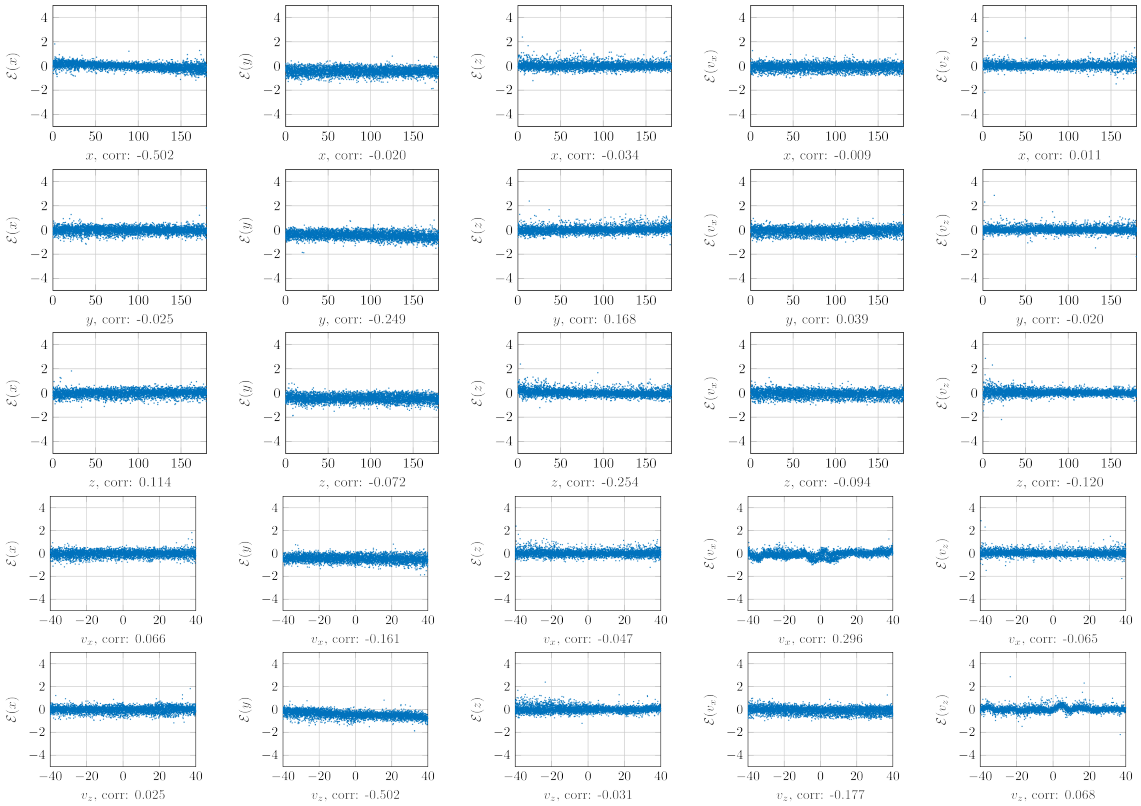


Figure 5.11: Correlation plot for DB476 using 20 000 novel datapoints as basis for the statistics. Larger versions are available in appendix C, figure C.13.

Conclusion:

The correlation plots in Figure 5.11 show that the algorithm produces good estimates for all implantation angles $< 40°$. The plots also indicate that any systematic errors are negligible.

5.2.2 Expanding the recoverable volume

One of the major goals of this project is to expand the recoverable volume, as defined in Section 3.1.2. Due to the issues with the fourth generation Autosetup, this had to be approached in two stages. After good hyper-parameters were found for each configuration, tests were performed to give details of the estimators' performance. Table 5.6 gives performance figures for the different configurations.

Table 5.6: Performance metrics for the expanded volumes. The performance statistics are for a test set with novel data.

$x \times y \times z \times v_x \times v_z$ (mm, °)	**Configuration**	$\bar{\mathcal{E}}_r$	p95($\mathcal{E}_r$)	p95($\mathcal{E}_{v_x}$)	p95($\mathcal{E}_{v_z}$)
$100 \times 100 \times 100 \times 80 \times 80$	8×130	0.301 mm	0.513 mm	1.103°	1.098°
$180 \times 180 \times 180 \times 80 \times 80$	10×250	0.453 mm	0.859 mm	0.598°	0.432°
$300 \times 300 \times 300 \times 90 \times 90$	10×175	0.956 mm	1.769 mm	1.238°	1.349°

Remark:

The databases used for the 180 mm model, `DB476` and the 300 mm model, `DB1071`, uses 161 051 and 175 616 data points respectively; however, the 300 mm model spans a 5.86× larger data space making its resolution lower. The effects of data spacing are addressed in Section 5.3.4. Given more time, this would have warranted further investigation; the results are nevertheless indicative.

Conclusion:

The results in Table 5.6 clearly indicate that the ANN-based estimator accomplishes the goal of extending the recoverable volume. The 300 mm volume is the largest data set the Autosetups can produce. Hence it may be possible to train the system to recover positions within an even larger volume.

5.2.3 Speed and latency measures

By lowering the latency of the data processing pipeline in the RayPilot system, the quality of the data presented to interfacing equipment improves in the temporal sense. Procedures such as gating, where the EBRT's beam is disabled when the target volume has deviated beyond a predefined threshold, benefit greatly from lowered latency.

Remark:

The algorithm is only one link in a long chain of processes which affect the latency. The data path, among other factors, includes electrical averaging, sampling delay, serial communication to the workstation, the algorithm and serial communication to interfacing units.

5.2.3.1 Estimation speed

The previous implementation, GGAM, was developed to improve the estimation time. The ANN estimator further improves the estimation time. Some rough performance measurements are shown in Table 5.7. The measurements for the ANNs were generated on a AMD Ryzen 7 1700 processor and generated by running an average over 100 000 estimations; the measurements for $\mathcal{P}_4$ are approximate and were provided by the company.

Table 5.7: Estimation speed for different estimators.

Model	Lang.	Latency	Estimates/s	Speedup
$\mathcal{P}_4$	C♯	$\sim 10\,\text{ms}$	~ 100	1×
ANN 10×80	C	$\sim 0.072\,\text{ms}$	$\sim 13\,930$	139.3×
ANN 10×250	C	$\sim 0.59\,\text{ms}$	~ 1540	15.4×

A rough estimate for the difference in execution time between the 10×80 and

10×250 networks is

$$\left(\frac{250}{80}\right)^2 \approx 9.77$$

which is the ratio of the number of multiplications performed in each layer for the two models; this is in line with the measured performance difference of $\sim 9.045\times$.

Conclusion:

The ANN's estimation speed, for the network configurations considered here, is so fast that the bottleneck has moved to the physical setup, more precisely the communication with the ADC. The ADC is the hardware which converts the physical voltages into digital representations. Furthermore, the run times are more deterministic than the search based implementation.

5.2.3.2 Training using TensorFlow

TensorFlow has the capability to use different target architectures such as various CPUs, GPUs and the custom designed application specific integrated circuit (ASIC) called tensor processing unit (TPU) [35]. This is achieved by using a high-level description with various back-ends, as described in Section 3.3.5.

The training performance of TensorFlow running on different hardware used throughout this project is presented in Table 5.8. In the table, the number of "threads" is an apples-to-oranges comparison. For the GPU, the thread count is the number of CUDA cores [36]; the intention is to showcase the amount of available parallelism. The CPU program is a sequential implementation; however, the compiler can make use of the SIMD (single instruction, multiple data) instructions such as Intel's Streaming SIMD Extensions 2 (SSE2) which utilizes the instruction level parallelism described in Section 4.2.2.4.

The switch from a laptop to a high-end workstation was mandated by the larger data sets associated with the larger volumes. The GPU from NVIDIA has a floating-point performance of 11.3 TFLOPS. In the context of Micropos' previous studies on ANN-based estimators, from 1999 and 2006, this was considered supercomputer processing power; In fact the worlds fastest supercomputer in 2001, the IBM Asci White, peaked at 7.226 TFLOPS [37]. Put in perspective, the Asci White consumed 6 MW [38] and cost $110 million [39] while the GTX 1080 Ti has a power consumption of 250 W and cost less than $900. Hence the GPU at hand is 37500× more power efficient than the IBM Asci White while delivering comparable performance, with less than two decades of technological progress separating them.

Table 5.8: Training performance of the same TensorFlow program executed on different hardware. The execution time was measured as the time required to complete 100 epochs of training using a 30 MB data set.

Hardware	Type	Clock freq.	Threads	Exec. time	Speedup
Intel® Core™ i5-3230M (MacBook Pro)	CPU	2.6 GHz	4	737 s	1x
AMD Ryzen™ 7 1700	CPU	3.65 GHz	16	426 s	1.73x
NVIDIA GeForce® GTX 1080 Ti	GPU	1.58 GHz	3582	12 s	62x

5.2.4 Model size

The file size used to store the model data is an interesting measure of the amount of information collected in the model. From Table 5.9, it is clear that the $\mathcal{P}_4$ model and the 10×70 ANN is of comparable size. Here both models have been saved to ASCII text files using the same numeric precision. This gives an interesting insight to the models suitability to the problem, the ANN-based estimator outperforms the polynomial model while requiring less space, i.e. fewer parameters.

Table 5.9: Comparison of storage space required to save the different models.

Model	Coefficients	Weights	Biases
$\mathcal{P}_4$	1095 kB	-	-
ANN 10×70	-	981 kB	15 kB
ANN 10×250	-	12.3 MB	57 kB

Conclusion:

The ANN-based estimator requires fewer coefficients to produce higher quality positional predictions in the standard volume. This measure, once again, validates the choice of method since a better approximation is generated using fewer degrees of freedom.

5.3 Additional experiments

In addition to the experiments defined at the start of the project, some *ad hoc* ideas were explored. Antenna gain normalization was tried as a method to add extra robustness towards variations in antenna gain. The alteration of the number of antennas was performed to gain insight, by extrapolation, into whether the RayPilot system had anything to gain from an increased number of antennas.

5.3.1 Antenna gain normalization

In this setup, each antenna voltage was modified using the average over the 16 antenna voltages. The results were mediocre. The angles had comparable results to the regular model, while the positional estimate, especially in the z-dimension, was significantly worse. If the experiment had been successful, the resulting system would have been more robust to production variations in antenna gain. Additional details of the configuration are found in Table B.4 in the Appendix.

Table 5.10: Performance comparison: Gain normalization. Three models were trained, two with and one without gain normalization. A novel data set of 20 000 points was used to generate the statistics.

Model	$\bar{\mathcal{E}}_r$	p95($\mathcal{E}_r$)	p95($\mathcal{E}_{v_x}$)	p95($\mathcal{E}_{v_z}$)
Standard	0.453 mm	0.859 mm	0.598°	0.432°
Gain-norm. div	0.631 mm	1.309 mm	0.705°	0.539°
Gain-norm. sub	0.720 mm	1.562 mm	0.635°	0.450°

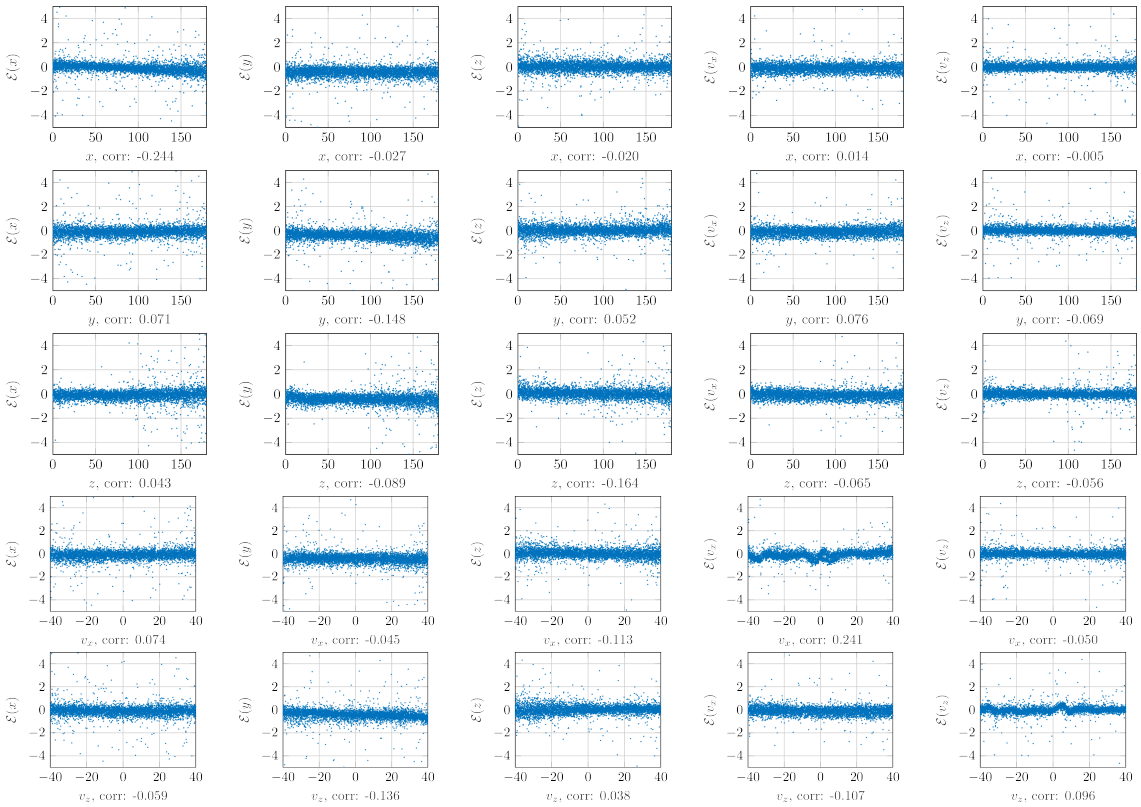


Figure 5.12: Correlation plot for `DB476` using 20 000 novel data points as basis for the statistics. Here the input to the neural network was scaled by the average over all antennas to normalize the gain. The patterns are close to the standard model, confer figure 5.11, but the points are more scattered. Larger versions are available in appendix C, figure C.26.

Conclusion:

While functional, the positional performance was crippled with regards to the p95($\mathcal{E}_r$)-measure. However, the angular recovery had unaffected performance; if such an application is required, the antenna gain normalization could be beneficial since the approach could provide additional robustness.

Remark:

The fact that the ADCs were logarithmic (dB/V) had not been communicated at the time of this test; had this been known, a different

experiment setup would have been used. While theoretically likely, no tests has been performed to ensure that the gain normalization implemented here actually improves robustness.

5.3.2 Altering number of antennas

While one of the benefits of the ANN-based estimator is its ability to utilize a multitude of inputs, experiments were made with a reduced number of antennas. The order in which the antennas were included in the model is shown in Table 5.11 with the numbering of the antennas shown in Figure 5.13. While this is not the optimal order, it is chosen somewhat strategically (i.e. not on a line or on just one side) and the results are nevertheless interesting. In Figure 5.14, the results are shown. The experiment was conducted using the `DB271` database for training; 2000 random data points were used for validation and another 4000 for testing. Further configuration details are available in Table B.5

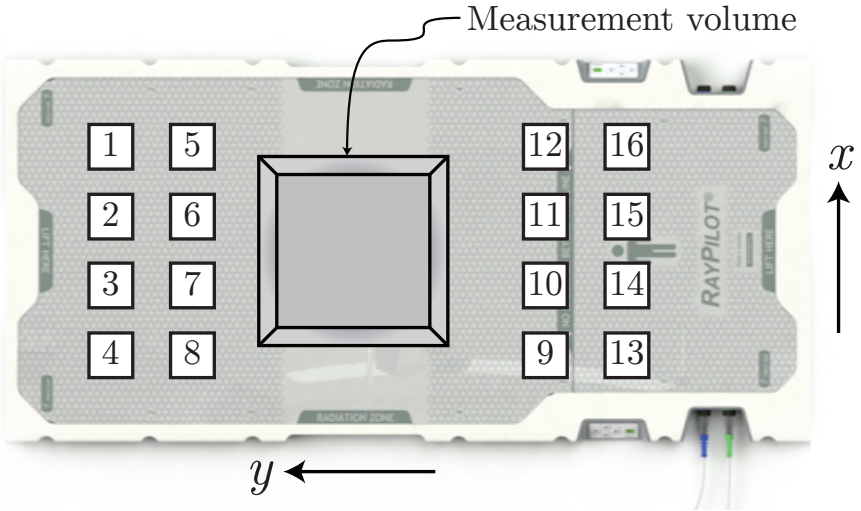


Figure 5.13: Numbering of the antennas and the measurement's placement in relation to the antennas.

Table 5.11: Antenna configurations used during the experiments with altered numbers of antennas.

	Active antennas															
# antennas	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1							✓									
2							✓				✓					
3							✓			✓	✓					
4						✓	✓			✓	✓					
5						✓	✓		✓	✓	✓					
6					✓	✓	✓		✓	✓	✓					
7					✓	✓	✓	✓	✓	✓	✓					
8					✓	✓	✓	✓	✓	✓	✓	✓				
9					✓	✓	✓	✓	✓	✓	✓	✓	✓			
10	✓				✓	✓	✓	✓	✓	✓	✓	✓	✓			
11	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
12	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓
13	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓
14	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
15	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
16	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

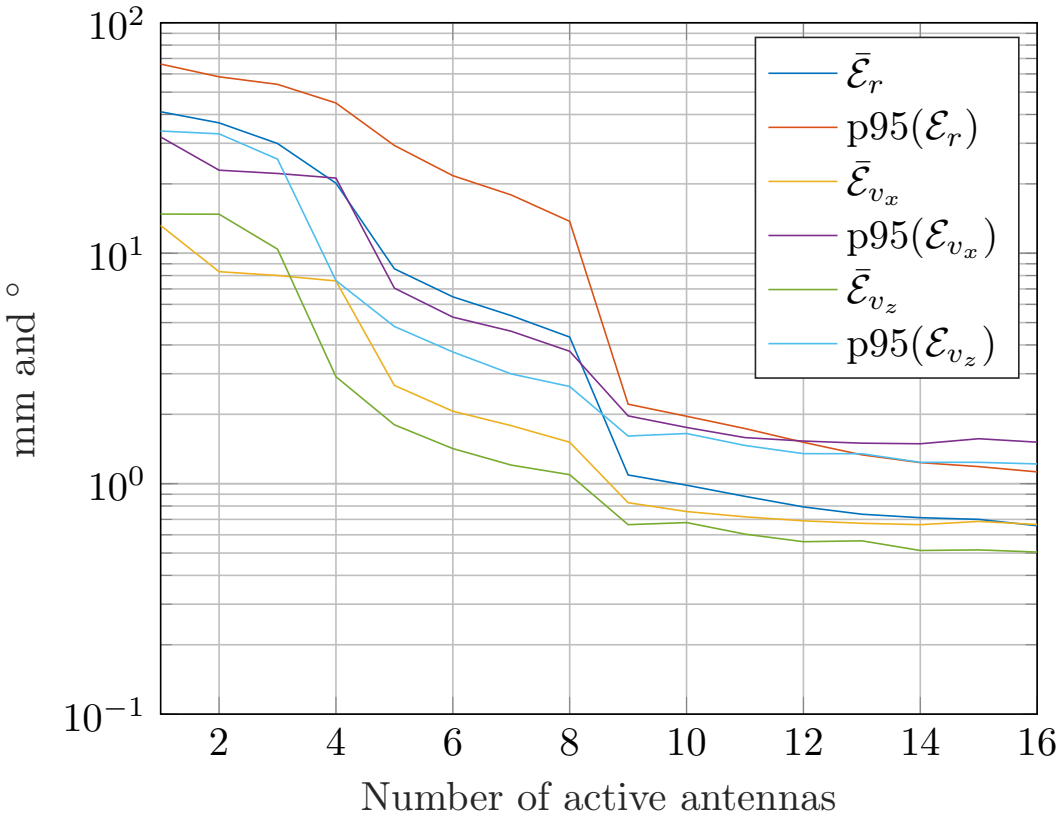


Figure 5.14: Different error-measures with respect to the number of antennas used to train the model. The y-axis is logarithmic to better illustrate the diminishing return of adding more antennas. The results are averaged over 10 runs.

Conclusion:

From Figure 5.14 it is clear that more antennas improve the quality of the estimations. Also, there was a change of improvement rate at nine antennas; the performance gain per antenna was reduced beyond nine antennas. However, this test was performed with one set of antenna combinations. For reference there are

$$\prod_{n=1}^{16} \binom{16}{n} \approx 7.9 \cdot 10^{45}$$

ways to choose 16 antenna combinations with 1–16 antennas! If the configurations are restricted to sets where a chosen antenna remains included in the selection (as in Table 5.11), the expression becomes

$$16! \approx 2.1 \cdot 10^{13}$$

which is way smaller, but still not a feasible search space.

Recommendation:

Given the trend seen in Figure 5.14, using more antennas in combination with a single ANN will likely only provide minor improvements to the

system's accuracy. Another approach is ensemble methods; using different input partitions, this method could be applied to further improve prediction quality and robustness by using committee predictions [40]. Such ensemble systems can benefit from a larger number of antennas, which provide redundant signal information.

5.3.3 Training using random data

The training databases have been of the type introduced in Section 3.1.2.1, where the training examples are placed on a grid. One may wonder how the performance would be affected if the case was reversed; a training set of random positions and a test set with a fixed grid layout, or a random training set combined with a random test set. New databases were generated for the experiment, their properties are listed in Table 5.12.

Table 5.12: Properties for the training databases used in this experiment and the reduced step size experiment.

Identifier	**X** (mm)	**Y** (mm)	**Z** (mm)	$\mathbf{V}_x$ (°)	$\mathbf{V}_z$ (°)	**Datapoints**	**Autosetup**
`Grid Fine`	100–200	100–200	100–200	-45–45	-45–45	133 100	v4
`Grid`	100–200	100–200	100–200	-45–45	-45–45	10 584	v4
`Random 1`	100–200	100–200	100–200	-45–45	-45–45	7776	v4
`Random 2`	100–200	100–200	100–200	-45–45	-45–45	7776	v4
`Validation`	100–200	100–200	100–200	-45–45	-45–45	2500	v4

Table 5.13: Reversed case, training using random data, and random data for both training and test.

Training set	**Test set**	**Conf.**	$\bar{\mathcal{E}}_r$	p95($\mathcal{E}_r$)	max($\mathcal{E}_r$)	p95($\mathcal{E}_{v_x}$)	p95($\mathcal{E}_{v_z}$)
`Grid`	`Random 1`	8×130	0.411 mm	0.689 mm	1.992 mm	1.116°	1.157°
`Grid`	`Random 2`	8×130	0.458 mm	0.824 mm	1.913 mm	1.131°	1.132°
`Random 1`	`Grid`	8×130	0.862 mm	2.380 mm	51.14 mm	1.510°	1.735°
`Random 1`	`Random 2`	8×130	0.286 mm	0.506 mm	1.515 mm	1.182°	1.010°
`Grid` ∪ `Random 1`	`Random 2`	8×130	0.301 mm	0.513 mm	0.880 mm	1.103°	1.097°

Recommendation:

It is obvious from the results of `Grid` ∪ `Random 1` that this diverse data set generates a better performing model. For future use, this type of data set is strongly recommended.

Conclusion:

When training with random data and evaluating against the grid, the max(·)-measure is critically affected; this is likely due to fact that the bounds of the data set are not guaranteed to be present in the random data set. However, a mix of random and grid based training data performed excellently! Additionally, this experiment shows the importance of including edge cases in the validation examples; an alternative would be a training set with a larger extent than the volume intended for use.

5.3.4 Reduced step size, standard volume

Providing more data using a smaller step size is likely to improve the precision of the function approximation. The database use here is `Grid Fine` from Table 5.12 with `Validation` as validation set and `Random 1` as test set. The results are listed in Table 5.14.

Table 5.14: Comparison between grid of 10 584 and 133 100 examples with the same training volume.

Training set	**Test set**	**Conf.**	$\bar{\mathcal{E}}_r$	p95($\mathcal{E}_r$)	p95($\mathcal{E}_{v_x}$)	p95($\mathcal{E}_{v_z}$)
`Grid`	`Random 1`	8×130	0.411 mm	0.689 mm	1.116°	1.157°
`Grid Fine`	`Random 1`	8×130	0.711 mm	1.108 mm	1.109°	1.083°
`Grid Fine`	`Random 1`	8×175	0.431 mm	0.736 mm	1.092°	1.171°

Conclusion:

Tighter example spacing is likely beneficial; the results in Table 5.14 are largely inconclusive since no hyper-parameter search has been performed for `Grid Fine` and 8×175, which was chosen arbitrarily, outperforms the 8×130 model which works well for the regular grid.

5.3.5 A linear sweep through the volume

An interesting question arises; does the network produce continuous predictions given a data set of a linear translation through the volume?

Let $\mathbf{p}_0 = (20, 0, 0)$ be the start point and $\mathbf{p}_{1000} = (280, 300, 300)$ be the end point of a line. This line cuts diagonally through the measurement volume. 1000 test points were generated as

$$\mathbf{p}_i = \mathbf{p}_0 + \frac{i}{1000} \cdot (\mathbf{p}_{1000} - \mathbf{p}_0)$$

where $i \in 1 \dots 1000$. Configuration details are available in Table B.6.

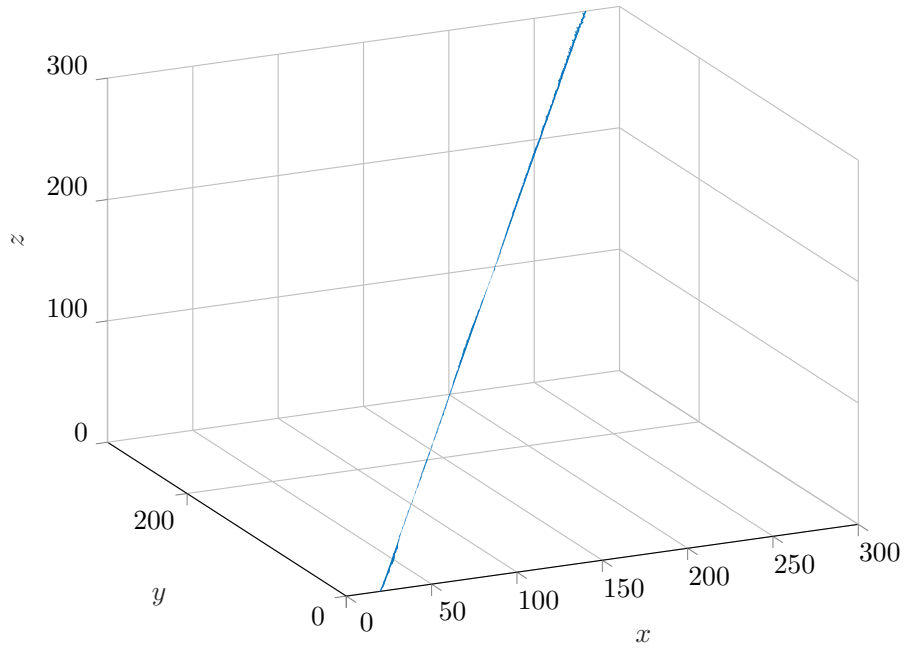


Figure 5.15: Radial deviations plotted as lines between each $(\mathbf{p}, \hat{\mathbf{p}})$ pair for a sweep diagonally through the measurement volume.

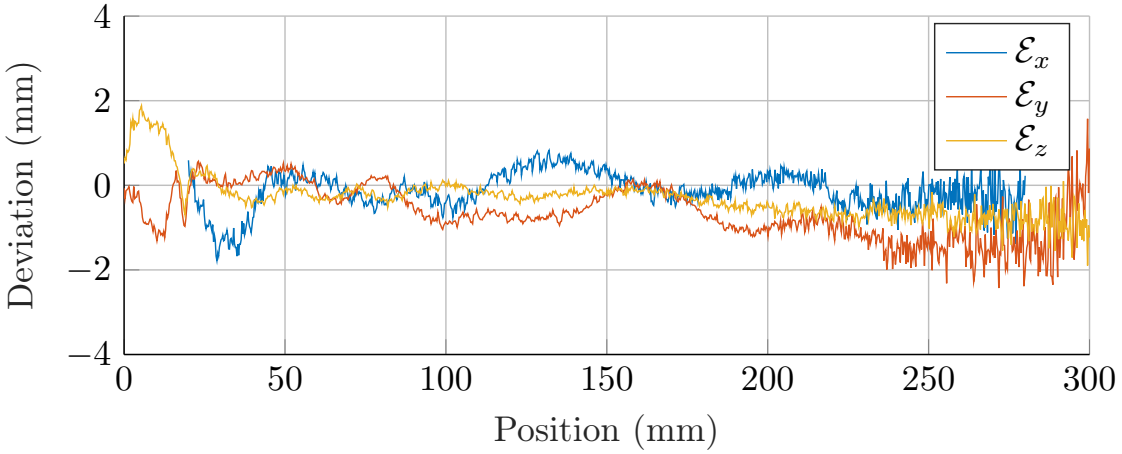


Figure 5.16: Absolute x-, y- and z-deviations from the reference line during a sweep diagonally through the measurement volume, shown in Figure 5.15.

Remark:

This test was performed with a 10×175 model (`DB1071`) which didn't perform as well, but covered the whole volume. There is some noise especially with higher z-values which is expected; the signal to noise ratio is lower since the electric field strength decreases rapidly with distance. The estimator performs better in the interior of the search space. However, the idea with this test is primarily to showcase that the estimator produces a reasonably continuous predictions given approximately continuous input. Since the system is generally used in a relative mode,

rather than absolute, this property is important. The test was performed again with a better model trained in a 100 mm volume trained with `Grid` ∪ `Random 1` from Section 5.3.3, see Figure 5.17. The average radial error over the interval shown is 0.474 mm and the largest radial deviation is 0.808 mm.

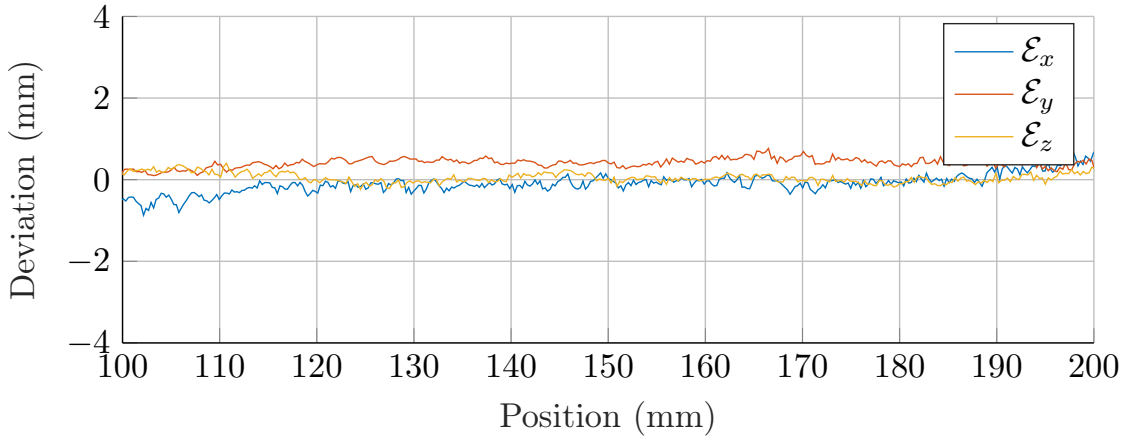
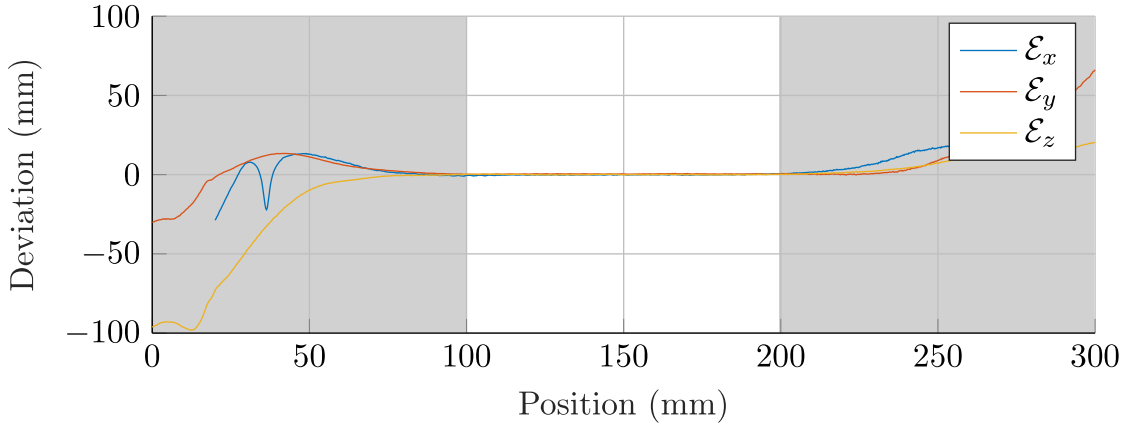


Figure 5.17: The same experiment as in Figure 5.16 estimated with the model trained with `Grid` ∪ `Random 1` from Section 5.3.3 in a smaller volume. The average radial error over the interval shown is 0.474 mm and the largest radial deviation is 0.808 mm.

Remark:
When performing the test in Figure 5.17, there was test data available outside the trained interval. This provided a nice insight into the generalizing property of the network estimator.



This is a fascinating result since the estimates are reasonable ±25 mm outside the volume, per axis. The results for the z-axis are even better, especially above the trained volume—at $z = 300$ mm, i.e. 100 mm outside, $\mathcal{E}_z = 20.4$ mm.

Conclusion:
The estimations are continuous in x-, y- and z-dimensions, as demonstrated in Figure 5.17. The estimator in Figure 5.16 is obviously not

production ready. The estimator has some relevance outside the trained volume, see preceding remark.

5.4 Discussion

In the following sections questions raised on various topics of the works are presented and reflected upon.

5.4.1 Method

The type of exploratory work performed in this thesis, with many parameters and somewhat stochastic performance in the training of the networks, has the inherent downside of an enormous search space. As work progressed, many old tests were invalidated by new discoveries. Ideally, when better performing parameters were found, all previous test would have been re-run. This is however not not always possible, due to resource constraints; personal judgment was used to decide when a change had enough impact to merit a re-run of the affected experiments.

5.4.2 Performance measures

The performance measures were chosen to match the company's previous work. Perhaps other measures, such as the mean squared error, would have been better suited for performance comparisons.

When moving to the larger volumes, there was no performance reference, since the $\mathcal{P}_4$ algorithm was tailored to the standard search space. However, from Table 5.6 is is clear that the ANN-based estimator outperforms the $\mathcal{P}_4$ algorithms in the unfair comparison of 100 mm volume versus 180 mm volume. For the 300 mm volume, the ANN-based estimator is on par with the $\mathcal{P}_4$'s performance in the 100 mm volume.

5.4.3 Hyper-parameter search

The hyper-parameter search experiments generally gave clear results, except for the tests on the $\max(\cdot)$-measure, which would have benefited from an averaging over several runs. However, neighbouring configurations of hidden layers and nodes per layer should have comparable performance and the visual impression from the graphs paints a clear picture. Given more time, the search for a good configuration for the 300 mm model could have covered a larger search space but the evaluations were very time consuming.

The result of the batch size parameter search was not used when training on the GPU, the reason being improved performance per time-unit when running larger batch sizes.

5.4.4 Training data sets

One of the more interesting tests was the training on random data in Section 5.3.3. The results were clear; a more diverse training set improves the performance. This test was done at the very end of the project; had this been known at an earlier stage, combined training data, i.e. grid and random examples, would have been used more often. Additionally, the test indicates that the grid is a tough test which is well suited for validation.

The test on tighter grid spacing in Section 5.3.4 was largely inconclusive. It seems intuitive that tighter spaced data should perform much better. Perhaps a separate hyper-parameter search on the tighter spaced data set would have given more insight.

5.4.5 Linear sweep

The linear sweep test of Section 5.3.5 illustrates some properties of the regular use case of the system. Figure 5.17 is a highlight of the project and demonstrates the desired properties of the estimator; the predictions are continuous which is important for relative positioning, the predictions are accurate with a maximum radial deviation of 0.808 mm and the estimator has a reduced latency. If the company is unable to finalize the 300 mm volume estimator for production, this model is a great improvement over the current estimator.

As mentioned previously, the RayPilot system is commonly used in relative mode where the estimations are relative to a start point within the trained volume, optimally in the center of the trained volume; this means that the repeatability of the Autosetup and the trained model is the most important property. The RayPilot can be software calibrated to interface with other coordinate systems by using a calibration fixture and generating the necessary offsets.

6
Conclusion

6.1 Summary

The two previous attempts to adapt MLPs for use with the RayPilot system have failed to improve on the existing algorithm—third time's a charm.

The project has fulfilled the goals of improving on the accuracy of the positional predictions, extending the recoverable volume to at least 180 mm volume and decreasing the latency of the estimations, with speedups between 15–150×. For the standard volume, the tolerance of rotated implants has been improved from ±20° to ±45° while improving the positional accuracy, the 95th percentile radial error has been reduced to $\sim \frac{1}{3}$.

A set of additional experiments has validated some assumptions:

- By plotting the spatial error distribution, it was shown that the model was performing equally well in all of the test volume, further explored in the Pearson correlation plots.
- Smaller model size for comparable performance; this indicates that the ANN-based estimator is a better fit to the problem than the $\mathcal{P}_4$-model.
- By altering the number of active antennas the project has shown that the quality of the estimations increases with more inputs.
- The test using random data showed that grid training data is suboptimal. The performance is improved when the training data had additional randomly distributed data points.
- The test with a linear sweep concluded that the estimators produced continuous output signals for continuous inputs.

The experimental nature of the work has shown its true colors; many things in the test setups have broken down, cables have been torn off, servo engines driven to the point of failure, a file system corrupted spontaneously, important data were overwritten, an unaviodable operating restart ruined a six day long measurement, et cetera.

6.2 Thesis contribution

Some of the estimators developed in this thesis improve quite dramatically on the best performing, to the company's knowledge, system in the world: the kilovolt

intrafraction monitoring (KIM) system which has an average systematic accuracy of 0.46 mm [41]. The KIM system uses X-ray for the image generation, adding to the radiation dosage of the patient; the system also has a lower refresh rate, 8 Hz.

With the ability to generate different data distributions for the training, validation and test data, the project highlights some properties of the resulting networks when the different distributions are used, as demonstrated in Section 5.3.3. These results are rather intuitive, nevertheless they may provide guidance for data collection tasks.

In the tests with different number of inputs to the network, both during training and inference, it is clear that the network can provide better results with an overdetermined input set. This is an interesting result with applications in the domain of sensor networks. This result is also in line with the observations in [15] where the positional accuracy is plotted as a function of the number of access points.

6.3 Future work

The 300 mm model still needs some tweaking before performing as well as the smaller models. However, performing a hyper-parameter search on this size of data sets is costly with regards to time, even on the GPU. Run-times for the hyper-parameter search in this thesis was 4 days for 44 parameter combinations—that would have been eight months on the MacBook Pro. One possibility is to use cloud based compute servers equipped with multiple powerful GPUs; hyper-parameter search is an example of an "embarrassingly parallel" problem, solved by running multiple instances with different parameters in parallel.

The project resulted in lots of unexplored ideas. One of the delimitations of this project was to treat the hardware in an idealized manner; the company would like to have models which could be calibrated to fit any system. This would reduce the workload in building each system. Perhaps a small regression stage can be built to adapt the antenna responses to some reference responses, thereby calibrating the input features to the ANN-based estimator.

As remarked in the text, better input scaling could be performed to reverse the effects of the logarithmic ADC. Since the electromagnetic field decreases with the distance squared, a first attempt would be $a_{\text{new}} = e^{\frac{1}{2}a}$, since this would make the voltage linearly related to the distance from the transmitter. As a consequence the work on antenna gain normalization initiated in this project needs futher exploration.

One of the initial goals of the project was an FPGA implementation of the forward estimator. This goal was not successful, due to part disinterest from the company, part the large scope of the project; the HNN-solution was dropped mid project.

An FPGA solution would have enabled the estimations to be performed inside the RayPilot sensor plate. This would have been desirable for a few reasons:

- Models are physically attached to the hardware which prevents mix-ups.
- Lower latency for real-time gating, since the need for communication with an intermediate PC can be eliminated.

- It lowers the requirements of the associated workstation.

The company might pursue such a solution in the future.

One interesting question that didn't get a resolution during the project was whether it is the data span or the number of data points, in the training set, that governs the suitable number of nodes.

Bibliography

[1] H. Kabir, Y. Wang, M. Yu, and Q.-J. Zhang, "Neural network inverse modeling and applications to microwave filter design", *IEEE Transactions on Microwave Theory and Techniques*, vol. 56, no. 4, pp. 867–879, 2008.

[2] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[3] B. Lennernäs, M. Edgren, and S. Nilsson, "Patient positioning using artificial intelligence neural networks, trained magnetic field sensors and magnetic implants", *Acta Oncologica*, vol. 38, no. 8, pp. 1109–1112, 1999.

[4] S. Haykin, *Neural Networks: A Comprehensive Foundation (2nd Edition)*, 2nd ed. Prentice Hall, Jul. 1998, ISBN: 9780132733502.

[5] A. Y. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance", in *Proceedings of the twenty-first international conference on Machine learning*, ACM, 2004, p. 78.

[6] D. Dearnaley, I. Syndikus, G. Sumo, M. Bidmead, D. Bloomfield, C. Clark, A. Gao, S. Hassan, A. Horwich, R. Huddart, *et al.*, "Conventional versus hypofractionated high-dose intensity-modulated radiotherapy for prostate cancer: Preliminary safety results from the chhip randomised controlled trial", *The lancet oncology*, vol. 13, no. 1, pp. 43–54, 2012.

[7] A. Heidenreich, P. J. Bastian, J. Bellmunt, M. Bolla, S. Joniau, T. van der Kwast, M. Mason, V. Matveev, T. Wiegel, F. Zattoni, *et al.*, "Eau guidelines on prostate cancer. part 1: Screening, diagnosis, and local treatment with curative intent—update 2013", *European urology*, vol. 65, no. 1, pp. 124–137, 2014.

[8] S. Aluwini, F. Pos, E. Schimmel, E. van Lin, S. Krol, P. P. van der Toorn, H. de Jager, M. Dirkx, W. G. Alemayehu, B. Heijmen, *et al.*, "Hypofractionated versus conventionally fractionated radiotherapy for patients with prostate cancer (hypro): Acute toxicity results from a randomised non-inferiority phase 3 trial", *The Lancet Oncology*, vol. 16, no. 3, pp. 274–283, 2015.

[9] J. Kindblom, H. Syrén, R. Iustin, and B. Lennernäs, *Stochastic patter of motion in the prostate*, Conference abstract: Poster, P1077. ESTRO27, Sep. 2008.

[10] P. Juneja, A. Kneebone, J. T. Booth, D. I. Thwaites, R. Kaur, E. Colvill, J. A. Ng, P. J. Keall, and T. Eade, "Prostate motion during radiotherapy of prostate cancer patients with and without application of a hydrogel spacer: A comparative study", *Radiation Oncology*, vol. 10, no. 1, p. 1, 2015.

[11] A. Vanhanen and M. Kapanen, "The effect of rectal retractor on intrafraction motion of the prostate", *Biomedical Physics & Engineering Express*, vol. 2, no. 3, p. 035 021, 2016.

[12] J. Crook, Y. Raymond, D. Salhani, H. Yang, and B. Esche, "Prostate motion during standard radiotherapy as assessed by fiducial markers", *Radiotherapy and Oncology*, vol. 37, no. 1, pp. 35–42, 1995.

[13] D. Cheng, *Field and wave electromagnetics*. Harlow: Pearson Education Limited, 2014, ISBN: 1292026561.

[14] D. Andreasson and O. Nilsson, "From volts to millimeters", Master's thesis, IT University of Gothenburg, Chalmers University of Technology, Gothenburg, Nov. 2006.

[15] M. S. Rahman, Y. Park, and K.-D. Kim, "Localization of wireless sensor network using artificial neural network", in *Communications and Information Technology, 2009. ISCIT 2009. 9th International Symposium on*, IEEE, 2009, pp. 639–642.

[16] A. Payal, C. S. Rai, and B. R. Reddy, "Analysis of some feedforward artificial neural network training algorithms for developing localization framework in wireless sensor networks", *Wireless Personal Communications*, vol. 82, no. 4, pp. 2519–2536, 2015.

[17] I. E. Commission *et al.*, *Iec 61217 radiotherapy equipment—co-ordinates, movements and scales*, 1996.

[18] M. Wahde, *Biologically inspired optimization methods: An introduction*. WIT press, 2008.

[19] J. Snyman, *Practical mathematical optimization: An introduction to basic optimization theory and classical and new gradient-based algorithms*. Springer Science & Business Media, 2005, vol. 97.

[20] N. R. Draper, H. Smith, and E. Pownell, *Applied regression analysis*. Wiley New York, 1966, vol. 3.

[21] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop", in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 9–48.

[22] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, 1989.

[23] M. Riedmiller and H. Braun, "Rprop-a fast adaptive learning algorithm", in *Proc. of ISCIS VII), Universitat*, Citeseer, 1992.

[24] M. D. Zeiler, "Adadelta: An adaptive learning rate method", *ArXiv preprint arXiv:1212.5701*, 2012.

[25] D. Kingma and J. Ba, "Adam: A method for stochastic optimization", *ArXiv preprint arXiv:1412.6980*, 2014.

[26] C. Bishop, "Pattern recognition and machine learning (information science and statistics), 1st edn. 2006. corr. 2nd printing edn", *Springer, New York*, 2007.

[27] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization", *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[28] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude", *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, 2012.

[29] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *Tensorflow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `http://tensorflow.org/`.

[30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning", in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, GA: USENIX Association, 2016, pp. 265–283, ISBN: 978-1-931971-33-1. [Online]. Available: `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi`.

[31] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.", in *Aistats*, vol. 9, 2010, pp. 249–256.

[32] C.-W. Hsu, C.-C. Chang, C.-J. Lin, *et al.*, "A practical guide to support vector classification", 2003.

[33] D. M. Ritchie, "The unix system: A stream input-output system", *AT&T Bell Laboratories Technical Journal*, vol. 63, no. 8, pp. 1897–1910, 1984.

[34] J. Rice, *Mathematical statistics and data analysis*. Belmont, CA: Thomson, Brooks and Cole, 2007, ISBN: 0534399428.

[35] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit", *ArXiv preprint arXiv:1704.04760*, 2017.

[36] NVIDIA, "CUDA API reference manual", *NVIDIA Corporation, Apr*, 2012.

[37] H. Meuer, E. Strohmaier, J. Dongarra, and H. D. Simon, "Top500 supercomputer sites", *Proceedings of SC*, pp. 10–16, 2001.

[38] W.-c. Feng and K. Cameron, "The green500 list: Encouraging sustainable supercomputing", *Computer*, vol. 40, no. 12, 2007.

[39] I. Foster, "Internet computing and the emerging grid", *Nature web matters*, vol. 7, 2000.

[40] T. G. Dietterich, "Ensemble methods in machine learning", in *International workshop on multiple classifier systems*, Springer, 2000, pp. 1–15.

[41] J. A. Ng, J. T. Booth, P. R. Poulsen, W. Fledelius, E. S. Worm, T. Eade, F. Hegi, A. Kneebone, Z. Kuncic, and P. J. Keall, "Kilovoltage intrafraction monitoring for prostate intensity modulated arc therapy: First clinical results", *International Journal of Radiation Oncology* Biology* Physics*, vol. 84, no. 5, e655–e661, 2012.

A

Hyper-parameter search data

A.1 Layers and nodes

Table A.1: The hyper-parameter search with respect to nodes and layers indicated that 8 layers of 130 nodes each performed the best.

Config.	$\bar{\mathcal{E}}_r$	**p95**$(\mathcal{E}_r)$	**max**$(\mathcal{E}_r)$	$\bar{\mathcal{E}}_{v_x}$	**p95**$(\mathcal{E}_{v_x})$	$\bar{\mathcal{E}}_{v_z}$	**p95**$(\mathcal{E}_{v_z})$
1×10	5.02	9.22	19.9	2.13	4.75	1.89	4.81
1×20	2.22	4.34	17.3	1.36	3.28	1.24	3.09
1×30	1.74	3.35	13.4	1.15	2.64	1.08	2.83
1×40	1.41	2.95	10.7	1.11	2.64	1.05	2.6
1×50	1.34	2.71	9.85	0.974	2.38	0.877	2.23
1×60	1.21	2.47	10.5	0.996	2.44	0.909	2.18
1×70	1.2	2.46	10.5	0.947	2.3	0.894	2.21
1×80	1.24	2.52	7.71	0.932	2.21	0.888	2.17
1×90	1.23	2.49	10.1	0.954	2.35	0.893	2.2
1×100	1.13	2.33	10.8	0.891	2.17	0.881	2.2
1×110	1.12	2.32	8.17	0.925	2.23	0.912	2.29
1×120	1.23	2.36	8.55	0.858	2.01	0.87	2.12
1×130	1.18	2.35	8.1	0.812	1.88	0.79	1.98
1×140	1.22	2.37	6.46	0.778	1.86	0.834	2.07
1×150	1.12	2.24	6.58	0.776	1.83	0.762	1.98
1×160	1.13	2.29	8.01	0.835	1.99	0.824	2.14
1×170	1.1	2.12	8.58	0.886	2.09	0.854	2.16
1×180	1.3	2.56	9.68	0.816	1.95	0.862	2.17
1×190	1.14	2.21	9.74	0.86	2.09	0.794	2.04
1×200	1.27	2.4	10.1	0.845	1.95	0.848	2.1
2×10	2.44	4.42	12.2	1.48	3.47	1.37	3.36
2×20	1.24	2.57	7.92	1.02	2.42	0.929	2.31
2×30	1.03	2.02	12	0.867	2.1	0.716	1.86
2×40	0.923	1.7	9.87	0.69	1.5	0.687	1.71
2×50	0.903	1.67	5.05	0.699	1.67	0.674	1.75
2×60	0.915	1.73	6.1	0.703	1.62	0.653	1.61
2×70	0.852	1.52	6.91	0.722	1.71	0.669	1.65
2×80	0.805	1.47	5	0.707	1.7	0.593	1.46
2×90	0.798	1.43	4	0.684	1.45	0.565	1.4
2×100	0.818	1.49	5.14	0.686	1.39	0.592	1.44
2×110	0.742	1.38	5.46	0.68	1.47	0.562	1.4
2×120	0.867	1.44	3.34	0.689	1.54	0.578	1.4
2×130	0.715	1.3	3.86	0.664	1.34	0.553	1.38
2×140	0.745	1.32	6.37	0.667	1.41	0.62	1.52
2×150	0.803	1.41	5.24	0.661	1.4	0.601	1.45
2×160	0.668	1.22	3.79	0.691	1.36	0.564	1.38
2×170	0.695	1.24	7.79	0.678	1.31	0.597	1.43
2×180	0.66	1.18	5.09	0.68	1.62	0.538	1.33
2×190	0.704	1.27	4.7	0.671	1.5	0.615	1.48
2×200	0.693	1.27	5.15	0.669	1.45	0.539	1.29
3×10	1.99	3.56	13	1.26	2.89	0.996	2.52
3×20	1.22	2.34	6.12	0.807	1.88	0.719	1.79
3×30	0.887	1.69	5.78	0.737	1.61	0.681	1.69

Config.	$\bar{\mathcal{E}}_r$	**p95**($\mathcal{E}_r$)	**max**($\mathcal{E}_r$)	$\bar{\mathcal{E}}_{v_x}$	**p95**($\mathcal{E}_{v_x}$)	$\bar{\mathcal{E}}_{v_z}$	**p95**($\mathcal{E}_{v_z}$)
3 × 40	0.874	1.64	4.33	0.704	1.53	0.609	1.49
3 × 50	0.754	1.42	3.48	0.676	1.46	0.595	1.41
3 × 60	0.731	1.3	3.85	0.675	1.51	0.601	1.44
3 × 70	0.729	1.31	6.27	0.678	1.34	0.576	1.44
3 × 80	0.709	1.24	3.76	0.687	1.59	0.543	1.28
3 × 90	0.73	1.23	3.24	0.699	1.6	0.526	1.31
3 × 100	0.642	1.2	4.24	0.685	1.5	0.574	1.35
3 × 110	0.675	1.22	5.51	0.66	1.34	0.509	1.23
3 × 120	0.632	1.15	3.88	0.67	1.49	0.538	1.33
3 × 130	0.576	1.09	3.45	0.696	1.34	0.515	1.25
3 × 140	0.621	1.11	2.87	0.68	1.52	0.508	1.28
3 × 150	0.571	1.03	5.57	0.675	1.54	0.497	1.21
3 × 160	0.558	1.01	2.48	0.644	1.46	0.524	1.33
3 × 170	0.661	1.16	4.28	0.67	1.42	0.51	1.27
3 × 180	0.613	1.16	2.88	0.672	1.28	0.482	1.17
3 × 190	0.651	1.15	3.76	0.622	1.41	0.492	1.23
3 × 200	0.708	1.21	3.09	0.627	1.2	0.477	1.15
4 × 10	1.6	3.09	9.67	0.981	2.52	0.991	2.38
4 × 20	0.992	1.9	5.95	0.863	2.15	0.657	1.65
4 × 30	0.906	1.68	9.13	0.756	1.77	0.607	1.5
4 × 40	0.855	1.52	5.21	0.698	1.62	0.564	1.37
4 × 50	0.862	1.48	6.03	0.693	1.39	0.584	1.41
4 × 60	0.807	1.36	4.12	0.681	1.34	0.558	1.32
4 × 70	0.631	1.13	5	0.664	1.3	0.572	1.32
4 × 80	0.666	1.2	3.42	0.662	1.48	0.529	1.27
4 × 90	0.645	1.16	3.85	0.631	1.26	0.51	1.22
4 × 100	0.689	1.24	2.32	0.675	1.27	0.517	1.25
4 × 110	0.666	1.2	2.79	0.614	1.24	0.518	1.21
4 × 120	0.59	1.05	2.51	0.664	1.31	0.532	1.26
4 × 130	0.598	1.09	5.59	0.658	1.37	0.457	1.1
4 × 140	0.559	0.986	4.22	0.638	1.26	0.507	1.22
4 × 150	0.57	1.07	2.79	0.65	1.54	0.456	1.09
4 × 160	0.525	0.943	4.98	0.618	1.44	0.42	1.02
4 × 170	0.63	1.11	3.87	0.649	1.25	0.437	1.08
4 × 180	0.612	1.08	2.5	0.631	1.32	0.492	1.18
4 × 190	0.732	1.18	3.16	0.6	1.35	0.426	1.04
4 × 200	0.638	1.15	3.43	0.622	1.25	0.466	1.12
5 × 10	1.79	3.44	13.9	1.24	2.9	0.843	2.09
5 × 20	1	1.86	6.56	0.748	1.76	0.636	1.65
5 × 30	0.877	1.55	6.3	0.731	1.75	0.584	1.46
5 × 40	0.811	1.42	4.59	0.675	1.53	0.494	1.23
5 × 50	0.723	1.23	4.07	0.66	1.46	0.521	1.26
5 × 60	0.69	1.18	4.06	0.666	1.43	0.522	1.24
5 × 70	0.612	1.05	2.29	0.651	1.39	0.526	1.25
5 × 80	0.67	1.21	3.14	0.659	1.37	0.518	1.24
5 × 90	0.708	1.26	4.13	0.638	1.33	0.476	1.13
5 × 100	0.645	1.15	3.96	0.631	1.42	0.461	1.14
5 × 110	0.526	0.94	4.42	0.651	1.5	0.482	1.14
5 × 120	0.544	1.03	4.03	0.622	1.38	0.445	1.12
5 × 130	0.533	0.955	3.09	0.652	1.24	0.48	1.15
5 × 140	0.638	1.1	4.03	0.652	1.51	0.43	0.991
5 × 150	0.56	0.994	3.35	0.643	1.52	0.426	1.05
5 × 160	0.524	0.946	3.68	0.656	1.47	0.439	1.03
5 × 170	0.559	0.971	2.34	0.682	1.4	0.472	1.09
5 × 180	0.699	1.13	3.07	**0.581**	1.26	0.465	1.13
5 × 190	0.541	1	3.22	0.624	1.46	0.461	1.11
5 × 200	0.681	1.14	2.97	0.638	1.5	0.43	1.06
6 × 10	1.35	2.69	7.7	0.841	1.95	0.861	2.04
6 × 20	0.985	1.76	5.71	0.697	1.61	0.625	1.5
6 × 30	0.746	1.39	3.55	0.712	1.7	0.56	1.34
6 × 40	0.808	1.45	4.32	0.682	1.53	0.554	1.35
6 × 50	0.637	1.16	7.02	0.689	1.34	0.514	1.23
6 × 60	0.636	1.14	3.5	0.671	1.43	0.537	1.28
6 × 70	0.651	1.15	2.21	0.675	1.57	0.49	1.17
6 × 80	0.541	0.957	2.96	0.69	1.58	0.487	1.17
6 × 90	0.555	1	2.3	0.658	1.36	0.496	1.24
6 × 100	0.53	0.942	4.31	0.668	1.5	0.476	1.09
6 × 110	0.515	0.948	3.72	0.663	1.28	0.43	1.01
6 × 120	0.615	1.12	2.43	0.602	1.37	0.439	1.07

Config.	$\bar{\mathcal{E}}_r$	$\mathbf{p95}(\mathcal{E}_r)$	$\mathbf{max}(\mathcal{E}_r)$	$\bar{\mathcal{E}}_{v_x}$	$\mathbf{p95}(\mathcal{E}_{v_x})$	$\bar{\mathcal{E}}_{v_z}$	$\mathbf{p95}(\mathcal{E}_{v_z})$
6 × 130	0.507	0.885	3.19	0.65	1.28	0.479	1.13
6 × 140	0.535	0.97	3.87	0.663	1.6	0.434	1.01
6 × 150	0.512	0.916	4.16	0.659	1.3	0.462	1.1
6 × 160	0.574	1.01	3.96	0.635	1.36	0.43	1.04
6 × 170	0.653	1.17	3.26	0.6	1.23	0.407	0.977
6 × 180	0.565	1.02	5.57	0.619	1.46	0.432	1.03
6 × 190	0.641	1.08	4.25	0.669	1.26	0.416	0.982
6 × 200	0.488	0.881	4.52	0.618	1.26	0.516	1.19
7 × 10	1.46	2.65	7.89	0.942	2.12	0.798	1.97
7 × 20	0.897	1.71	5.26	0.789	1.9	0.675	1.64
7 × 30	0.839	1.52	5.59	0.699	1.7	0.555	1.34
7 × 40	0.69	1.17	2.7	0.691	1.68	0.527	1.28
7 × 50	0.736	1.26	2.98	0.647	1.46	0.51	1.23
7 × 60	0.606	1.08	3.03	0.685	1.49	0.477	1.13
7 × 70	0.605	1.07	4.58	0.674	1.62	0.485	1.15
7 × 80	0.549	0.962	5.16	0.645	1.35	0.44	1.04
7 × 90	0.589	1	2.69	0.665	1.25	0.465	1.12
7 × 100	0.627	1.16	2.9	0.66	1.25	0.497	1.19
7 × 110	0.625	1.09	2.94	0.638	1.34	0.415	0.977
7 × 120	0.491	0.884	5.75	0.635	1.43	0.422	1.01
7 × 130	0.466	0.841	2.86	0.627	1.33	0.426	0.986
7 × 140	0.531	0.94	4.5	0.652	1.39	0.468	1.12
7 × 150	0.535	0.982	2.87	0.657	1.22	0.459	1.06
7 × 160	0.621	1.07	5.33	0.648	**1.19**	0.442	1.07
7 × 170	0.548	0.963	3.1	0.651	1.46	0.425	1
7 × 180	0.494	0.879	3.59	0.686	1.61	0.46	1.08
7 × 190	0.608	1.07	3.28	0.689	1.44	0.424	1.04
7 × 200	0.592	0.992	4.34	0.641	1.43	0.435	1.06
8 × 10	1.4	2.62	9.93	0.932	2.22	0.696	1.75
8 × 20	0.885	1.65	4.51	0.757	1.87	0.664	1.57
8 × 30	0.823	1.47	4.39	0.661	1.52	0.552	1.31
8 × 40	0.802	1.4	5.25	0.646	1.42	0.566	1.33
8 × 50	0.623	1.12	3.59	0.635	1.37	0.505	1.24
8 × 60	0.563	1.02	3.28	0.652	1.49	0.468	1.1
8 × 70	0.629	1.1	2.25	0.648	1.49	0.498	1.15
8 × 80	0.608	1.1	4.35	0.677	1.27	0.514	1.2
8 × 90	0.608	1.08	3.89	0.644	1.48	0.465	1.1
8 × 100	0.578	0.969	3.75	0.673	1.2	0.494	1.14
8 × 110	0.53	0.95	5.93	0.688	1.42	0.448	1.07
8 × 120	0.482	0.869	2.16	0.665	1.22	0.483	1.14
8 × 130	**0.452**	**0.801**	4.25	0.669	1.5	0.453	1.05
8 × 140	0.511	0.935	4.73	0.656	1.38	0.455	1.08
8 × 150	0.507	0.91	2.59	0.666	1.28	0.418	0.988
8 × 160	0.601	1.09	2.71	0.653	1.24	0.42	0.968
8 × 170	0.563	1.04	3.31	0.637	1.22	0.437	1.04
8 × 180	0.507	0.869	4.54	0.643	1.36	0.472	1.13
8 × 190	0.532	1.01	4.31	0.649	1.45	0.457	1.09
8 × 200	0.569	1.04	3.81	0.631	1.36	0.437	1.01
9 × 10	1.6	3	10.8	0.959	2.33	0.775	2
9 × 20	0.935	1.69	6.84	0.743	1.67	0.675	1.63
9 × 30	0.795	1.4	4.24	0.694	1.66	0.563	1.34
9 × 40	0.715	1.23	3.25	0.656	1.45	0.507	1.21
9 × 50	0.64	1.13	3.65	0.669	1.5	0.517	1.23
9 × 60	0.577	1.03	3.07	0.668	1.22	0.495	1.2
9 × 70	0.689	1.24	2.87	0.666	1.25	0.483	1.17
9 × 80	0.547	0.984	2.2	0.668	1.28	0.448	1.08
9 × 90	0.54	0.927	2.63	0.665	1.45	0.437	1.02
9 × 100	0.573	1.04	2.6	0.66	1.33	0.443	1.08
9 × 110	0.502	0.873	3.83	0.692	1.25	0.445	1.05
9 × 120	0.525	0.904	3.41	0.655	1.54	0.453	1.04
9 × 130	0.518	0.927	3.08	0.672	1.44	0.449	1.05
9 × 140	0.604	1.1	2.73	0.629	1.23	0.418	0.976
9 × 150	0.566	1.06	3.46	0.659	1.23	0.442	1.09
9 × 160	0.499	0.867	2.44	0.668	1.29	0.46	1.09
9 × 170	0.533	0.966	3.48	0.683	1.33	0.444	1.04
9 × 180	0.66	1.14	2.8	0.668	1.42	0.405	0.965
9 × 190	0.638	1.05	5.6	0.622	1.44	0.444	1.05
9 × 200	0.499	0.918	5.09	0.644	1.53	0.435	0.995
10 × 10	1.68	3.11	11.5	0.887	2.13	0.818	1.91

Config.	$\bar{\mathcal{E}}_r$	$\mathbf{p95}(\mathcal{E}_r)$	$\mathbf{max}(\mathcal{E}_r)$	$\bar{\mathcal{E}}_{v_x}$	$\mathbf{p95}(\mathcal{E}_{v_x})$	$\bar{\mathcal{E}}_{v_z}$	$\mathbf{p95}(\mathcal{E}_{v_z})$
10×20	1.01	1.82	7.64	0.739	1.63	0.619	1.54
10×30	0.769	1.43	4.65	0.715	1.69	0.516	1.24
10×40	0.731	1.24	4.91	0.672	1.37	0.525	1.25
10×50	0.618	1.03	3.68	0.674	1.54	0.478	1.1
10×60	0.607	1.05	4.15	0.657	1.3	0.472	1.13
10×70	0.559	1.06	2.22	0.672	1.54	0.451	1.08
10×80	0.5	0.909	3.68	0.669	1.42	0.451	1.09
10×90	0.638	1.18	3.47	0.669	1.35	0.413	1.03
10×100	0.602	1.04	3.88	0.656	1.2	0.475	1.09
10×110	0.498	0.869	5.02	0.674	1.37	0.505	1.18
10×120	0.565	1.02	3.78	0.64	1.3	0.429	1.04
10×130	0.534	0.965	2.21	0.684	1.34	0.433	1.01
10×140	0.527	0.979	3.67	0.674	1.48	0.448	1.07
10×150	0.559	0.989	5.01	0.684	1.35	0.457	1.09
10×160	0.577	1.08	3.22	0.656	1.25	0.443	1.06
10×170	0.527	0.942	3.83	0.637	1.49	0.472	1.09
10×180	0.537	0.936	2.39	0.684	1.43	0.46	1.07
10×190	0.577	1.01	4.4	0.645	1.24	0.495	1.15
10×200	0.659	1.23	5.48	0.663	1.3	0.448	1.04
11×10	1.59	3.04	9.01	0.976	2.4	0.861	2.13
11×20	0.906	1.69	5.77	0.747	1.75	0.624	1.51
11×30	0.7	1.26	4.13	0.676	1.59	0.504	1.21
11×40	0.702	1.24	4.47	0.668	1.52	0.532	1.26
11×50	0.642	1.1	2.78	0.672	1.42	0.486	1.19
11×60	0.568	1.04	4.06	0.67	1.53	0.456	1.09
11×70	0.482	0.844	3.95	0.649	1.42	0.444	1.04
11×80	0.516	0.941	3.92	0.675	1.37	0.483	1.14
11×90	0.51	0.904	5.46	0.681	1.25	0.467	1.08
11×100	0.495	0.872	3.01	0.681	1.48	0.446	1.05
11×110	0.556	1.02	2.64	0.702	1.46	0.463	1.13
11×120	0.586	1.05	4.04	0.693	1.32	0.42	1
11×130	0.591	1.07	2.53	0.664	1.36	0.418	1.01
11×140	0.643	1.12	2.96	0.645	1.41	0.424	1.02
11×150	0.523	0.93	3.92	0.675	1.56	0.422	0.986
11×160	0.611	1.04	2.64	0.672	1.45	0.469	1.12
11×170	0.561	1.01	2.26	0.676	1.41	0.484	1.15
11×180	0.583	1.07	2.3	0.639	1.46	0.426	1.02
11×190	0.583	1.02	3.33	0.67	1.38	0.471	1.11
11×200	0.528	0.914	4.78	0.659	1.42	0.469	1.12
12×10	1.68	3.11	7.67	0.923	2.14	0.82	1.99
12×20	0.914	1.73	5.82	0.694	1.52	0.599	1.46
12×30	0.784	1.37	4.02	0.704	1.61	0.553	1.32
12×40	0.631	1.1	13.5	0.686	1.65	0.5	1.21
12×50	0.694	1.22	4.69	0.633	1.42	0.49	1.16
12×60	0.61	1.13	2.82	0.664	1.4	0.503	1.19
12×70	0.587	1.07	3.38	0.657	1.37	0.45	1.09
12×80	0.546	0.969	2.44	0.669	1.48	0.449	1.06
12×90	0.541	0.982	2.95	0.659	1.44	0.427	1.06
12×100	0.556	1.01	5.1	0.673	1.38	0.459	1.1
12×110	0.505	0.911	2.13	0.668	1.26	0.434	1.02
12×120	0.577	1.06	3.68	0.658	1.45	0.472	1.12
12×130	0.62	1.15	4.84	0.645	1.25	0.426	1.02
12×140	0.534	0.953	2.74	0.655	1.3	0.439	1.05
12×150	0.639	1.09	5.05	0.651	1.34	0.476	1.15
12×160	0.608	1.05	4.24	0.677	1.39	0.443	1.04
12×170	0.651	1.12	3.45	0.652	1.4	0.404	0.99
12×180	0.502	0.923	2.84	0.654	1.5	0.447	1.08
12×190	0.601	1.09	5.34	0.659	1.42	0.424	1.01
12×200	0.626	1.1	3.68	0.68	1.5	0.467	1.12
13×10	1.65	3.07	7.41	0.805	1.98	0.796	1.97
13×20	0.967	1.77	5.46	0.736	1.8	0.661	1.56
13×30	0.733	1.34	3.02	0.71	1.69	0.547	1.29
13×40	0.58	1.09	2.62	0.68	1.54	0.49	1.21
13×50	0.607	1.08	2.64	0.641	1.41	0.495	1.21
13×60	0.514	0.932	3.07	0.69	1.53	0.524	1.24
13×70	0.575	0.987	2.26	0.671	1.44	0.441	1.06
13×80	0.568	1	2.53	0.667	1.23	0.441	1.08
13×90	0.631	1.1	3.57	0.67	1.47	0.436	1.03
13×100	0.666	1.22	3.02	0.652	1.39	0.437	1.05

Config.	$\bar{\mathcal{E}}_r$	**p95**($\mathcal{E}_r$)	**max**($\mathcal{E}_r$)	$\bar{\mathcal{E}}_{v_x}$	**p95**($\mathcal{E}_{v_x}$)	$\bar{\mathcal{E}}_{v_z}$	**p95**($\mathcal{E}_{v_z}$)
13 × 110	0.542	0.98	3.62	0.645	1.32	0.466	1.08
13 × 120	0.593	1.04	3.39	0.68	1.33	0.474	1.11
13 × 130	0.567	1	4.83	0.685	1.49	0.45	1.07
13 × 140	0.711	1.29	2.93	0.65	1.39	0.462	1.08
13 × 150	0.703	1.26	4.58	0.661	1.34	0.447	1.08
13 × 160	0.631	1.12	5	0.658	1.52	0.436	1.04
13 × 170	0.612	1.11	2.83	0.658	1.4	0.457	1.09
13 × 180	0.668	1.16	4.1	0.647	1.41	0.448	1.09
13 × 190	0.672	1.2	6.22	0.644	1.27	0.454	1.09
13 × 200	0.684	1.24	5.37	0.653	1.33	0.426	1
14 × 10	1.75	3.19	10.4	0.908	2.12	0.908	2.24
14 × 20	0.979	1.76	9.95	0.716	1.68	0.618	1.44
14 × 30	0.7	1.3	4.03	0.714	1.7	0.592	1.37
14 × 40	0.596	1.08	2.84	0.7	1.61	0.482	1.16
14 × 50	0.645	1.12	4.2	0.649	1.38	0.435	1.04
14 × 60	0.578	1.07	2.44	0.686	1.55	0.474	1.11
14 × 70	0.522	0.968	2.58	0.672	1.45	0.457	1.07
14 × 80	0.565	1.01	7.63	0.655	1.44	0.472	1.13
14 × 90	0.622	1.06	3.43	0.67	1.42	0.447	1.04
14 × 100	0.57	0.991	2.78	0.676	1.32	0.445	1.04
14 × 110	0.568	1.05	3.99	0.643	1.41	0.414	1.02
14 × 120	0.62	1.07	3.56	0.663	1.46	0.439	1.04
14 × 130	0.734	1.39	2.5	0.658	1.34	**0.4**	**0.964**
14 × 140	0.597	1.08	3.86	0.658	1.4	0.459	1.09
14 × 150	0.655	1.21	4.17	0.679	1.32	0.447	1.08
14 × 160	0.668	1.22	4.45	0.656	1.35	0.476	1.13
14 × 170	0.694	1.24	3.89	0.652	1.39	0.467	1.14
14 × 180	0.654	1.24	4.65	0.69	1.37	0.496	1.18
14 × 190	0.664	1.18	3.33	0.679	1.48	0.438	1.04
14 × 200	0.856	1.52	5.2	0.672	1.39	0.465	1.1
15 × 10	1.82	3.39	7.67	0.977	2.45	0.754	1.99
15 × 20	1.04	1.96	5.79	0.753	1.67	0.659	1.55
15 × 30	0.707	1.3	3.97	0.739	1.44	0.58	1.35
15 × 40	0.61	1.09	3.5	0.656	1.36	0.524	1.28
15 × 50	0.66	1.17	2.85	0.655	1.34	0.477	1.18
15 × 60	0.695	1.24	2.6	0.643	1.41	0.451	1.07
15 × 70	0.522	0.967	2.56	0.686	1.49	0.45	1.07
15 × 80	0.631	1.15	6.45	0.663	1.29	0.426	0.985
15 × 90	0.522	0.931	**2.11**	0.667	1.49	0.422	0.992
15 × 100	0.57	1.01	3.83	0.668	1.41	0.466	1.08
15 × 110	0.597	1.05	3.94	0.668	1.47	0.455	1.12
15 × 120	0.645	1.15	3.78	0.674	1.47	0.455	1.06
15 × 130	0.709	1.23	4.82	0.677	1.48	0.426	1.01
15 × 140	0.659	1.16	3.46	0.677	1.56	0.431	1.03
15 × 150	0.585	1.07	4.27	0.676	1.37	0.454	1.08
15 × 160	0.64	1.17	4.39	0.693	1.55	0.433	1.02
15 × 170	0.691	1.26	5.19	0.684	1.55	0.433	1.05
15 × 180	0.75	1.36	7.32	0.68	1.48	0.48	1.13
15 × 190	0.827	1.42	5.07	0.683	1.53	0.491	1.18
15 × 200	0.759	1.38	4.94	0.675	1.45	0.501	1.19

Table A.2: The hyper-parameter search for the 300 mm volume. With regards to the radial errors the best performing model was 8 × 225 and 10 × 175.

Config.	$\bar{\mathcal{E}}_r$	**p95**($\mathcal{E}_r$)	**max**($\mathcal{E}_r$)	$\bar{\mathcal{E}}_{v_x}$	**p95**($\mathcal{E}_{v_x}$)	$\bar{\mathcal{E}}_{v_z}$	**p95**($\mathcal{E}_{v_z}$)
8 × 100	1.51	3.14	11.9	0.645	1.51	0.646	1.52
8 × 125	1.74	3.05	12.8	0.607	1.38	0.62	1.46
8 × 150	1.25	2.55	8.49	0.686	1.65	0.615	1.42
8 × 175	1.34	2.57	13.4	0.597	1.34	0.611	1.39
8 × 200	1.23	2.44	9.91	0.618	1.4	0.625	1.46
8 × 225	**1.14**	2.33	10.6	0.704	1.64	0.618	1.37
8 × 250	1.36	2.68	14.5	0.632	1.48	0.651	1.53
8 × 275	1.37	2.73	13.1	0.63	1.44	0.619	1.4
8 × 300	1.58	3.01	11.5	0.665	1.55	0.64	1.48
8 × 325	1.3	2.62	16.2	0.753	1.69	0.706	1.62
8 × 350	1.77	3.2	11.4	0.667	1.55	0.678	1.62
10 × 100	1.47	3.09	17.7	0.629	1.46	0.641	1.48

Config.	$\bar{\mathcal{E}}_r$	**p95**($\mathcal{E}_r$)	**max**($\mathcal{E}_r$)	$\bar{\mathcal{E}}_{v_x}$	**p95**($\mathcal{E}_{v_x}$)	$\bar{\mathcal{E}}_{v_z}$	**p95**($\mathcal{E}_{v_z}$)
10×125	1.34	2.51	14.5	0.64	1.48	0.627	1.46
10×150	1.33	2.61	9.17	**0.587**	1.33	0.645	1.48
10×175	1.18	**2.29**	**7.4**	0.663	1.52	0.617	1.39
10×200	1.37	2.53	13.2	0.624	1.41	0.625	1.42
10×225	1.28	2.48	8.82	0.649	1.45	0.669	1.55
10×250	1.56	2.89	11.3	0.66	1.51	0.668	1.54
10×275	1.81	3.43	15.3	0.694	1.65	0.658	1.52
10×300	1.69	3.26	12.5	0.703	1.64	0.694	1.61
10×325	1.89	3.67	13.4	0.733	1.72	0.698	1.63
10×350	1.84	3.62	19.8	0.734	1.67	0.746	1.76
12×100	1.36	2.65	9.25	0.621	1.47	0.636	1.48
12×125	1.25	2.6	9.68	0.595	**1.31**	0.643	1.53
12×150	1.3	2.47	12.8	0.609	1.37	0.631	1.46
12×175	1.31	2.43	14.3	0.658	1.49	0.643	1.48
12×200	1.61	3.04	12.7	0.633	1.47	0.641	1.55
12×225	1.76	3.34	11.8	0.664	1.49	0.662	1.5
12×250	1.66	3.3	7.73	0.687	1.57	0.739	1.76
12×275	2.09	3.74	8.87	0.691	1.58	0.749	1.73
12×300	2.27	4.21	11.6	0.686	1.57	0.755	1.81
12×325	2.15	4.18	21.5	0.729	1.74	0.777	1.84
12×350	2.1	4.03	13.4	0.781	1.82	0.796	1.86
14×100	1.32	2.59	9.08	0.604	1.38	**0.603**	**1.35**
14×125	1.41	2.62	12.8	0.6	1.38	0.62	1.48
14×150	1.55	2.86	13.8	0.608	1.39	0.619	1.43
14×175	1.61	3.09	10.2	0.675	1.61	0.654	1.53
14×200	1.8	3.25	9.3	0.668	1.54	0.703	1.69
14×225	1.8	3.4	10.4	0.677	1.6	0.746	1.73
14×250	1.78	3.5	9.33	0.749	1.72	0.763	1.79
14×275	2.02	3.84	10.9	0.73	1.66	0.789	1.84
14×300	2.34	4.48	15.8	0.782	1.8	0.742	1.77
14×325	2.49	4.68	27.3	0.74	1.78	0.836	2.01
14×350	2.12	4.22	13.2	0.766	1.76	0.824	1.98

B

ANN Configurations

This appendix contains additional configuration details for the experiments performed.

Table B.1: Common parameters through all tests.

Parameter	Setting
Optimization algorithm	ADAM-optimizer
β_1, β_2	0.9, 0.999 (ADAM defauls)
Weight initialization	Xavier initialization
Bias initialization	Zeros
Activation function	$\varphi(x) = 1.7159 \tanh\left(\frac{2}{3}x\right)$
Cost function	L2-norm

Table B.2: Hyper-parameter search standard volume.

Parameter	Setting
Database	`DB271`
Learning rate	0.0005
Batch size	4000
Training examples	7776
Validation examples	2000
Test examples	4000
Epochs	25 000
Post-processing	None

Table B.3: Hyper-parameter search 300 mm volume.

Parameter	Setting
Database	DB1071
Learning rate	0.0005
Batch size	10 000
Training examples	175 616
Validation examples	10 000
Test examples	10 000
Epochs	10 000
Post-processing	None

Table B.4: Antenna gain normalization.

Parameter	Setting
Database	DB476
Learning rate	0.0001
Batch size	10 000
Training examples	171 051
Validation examples	10 000
Test examples	10 000
Epochs	500 000
Post-processing	None

Table B.5: Altered number of antennas.

Parameter	Setting
Database	DB271
Learning rate	0.0001
Batch size	4000
Training examples	7776
Validation examples	2000
Test examples	4000
Epochs	25 000
Post-processing	Averaged over 10 runs

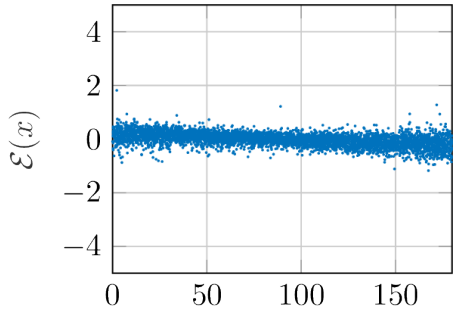
Table B.6: Linear sweep.

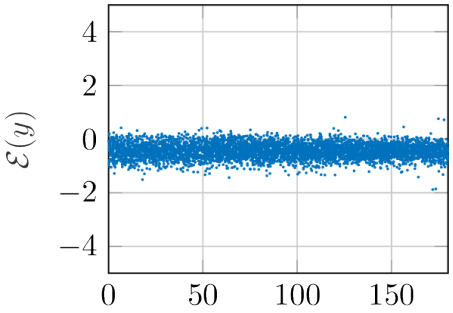
Parameter	Setting
Database	DB1071
Learning rate	0.0001
Batch size	4000
Training examples	175 616
Validation examples	10 000
Test examples	10 000
Epochs	15 000
Post-processing	None

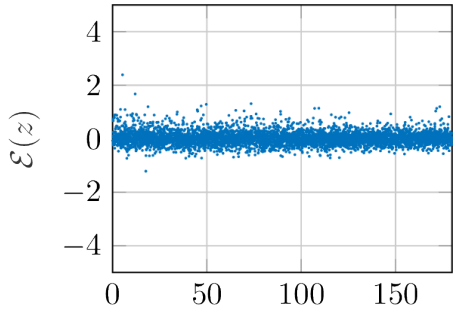
C

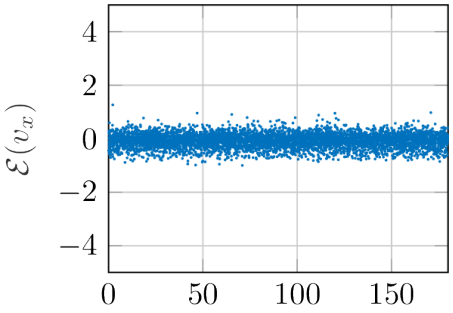
Graphs and plots

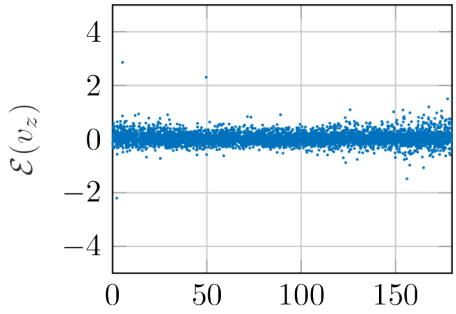
This appendix provides larger versions, for readability, of selected graphs and plots from the text.

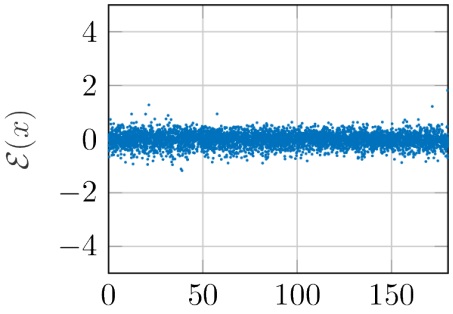


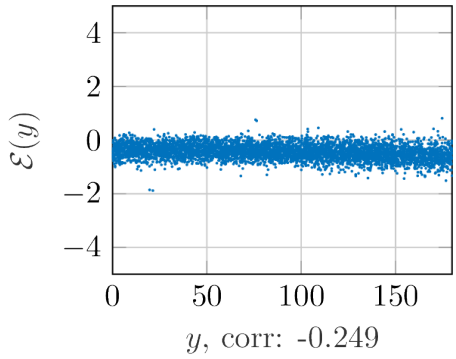


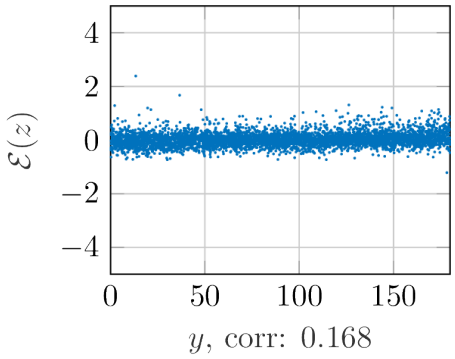


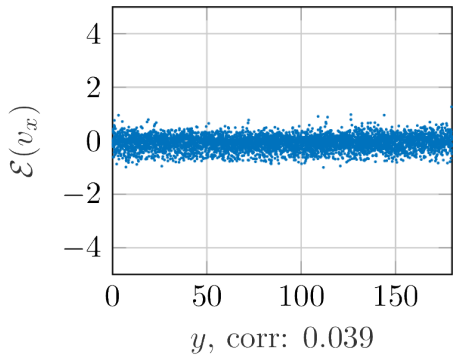


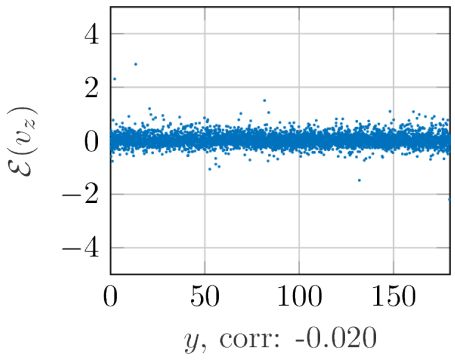


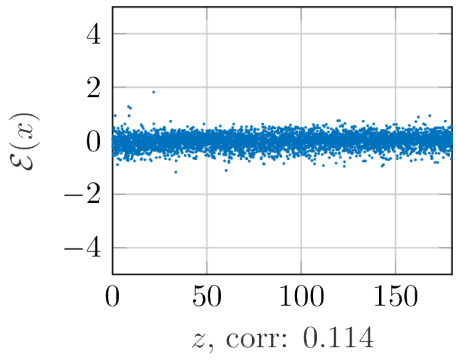


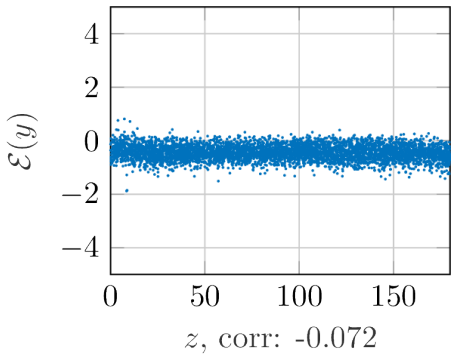

$\mathcal{E}(y)$
4
2
0
−2
−4
0
50
100
150
y, corr: -0.249

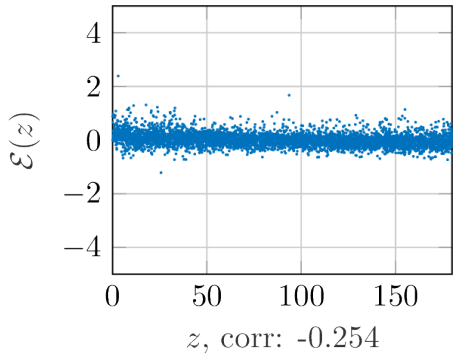

$\mathcal{E}(z)$
4
2
0
−2
−4
0
50
100
150
y, corr: 0.168

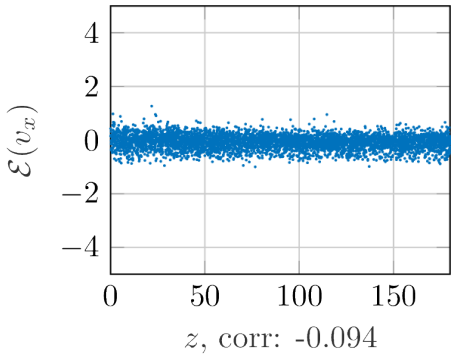

$\mathcal{E}(v_x)$
4
2
0
−2
−4
0
50
100
150
y, corr: 0.039

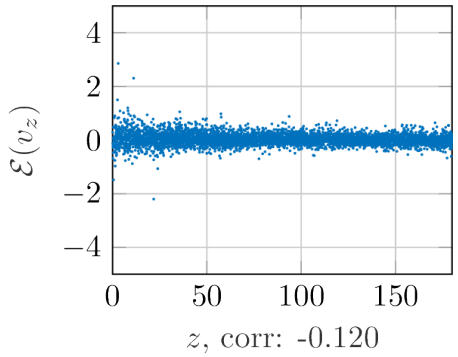

$\mathcal{E}(v_z)$
4
2
0
−2
−4
0
50
100
150
y, corr: -0.020

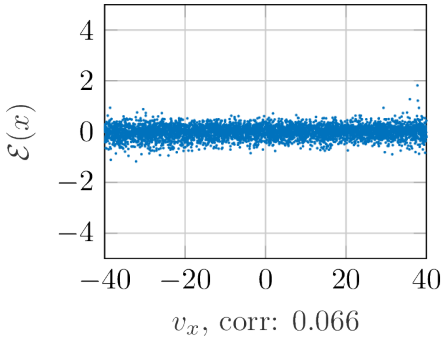

$\mathcal{E}(x)$
4
2
0
−2
−4
0
50
100
150
z, corr: 0.114

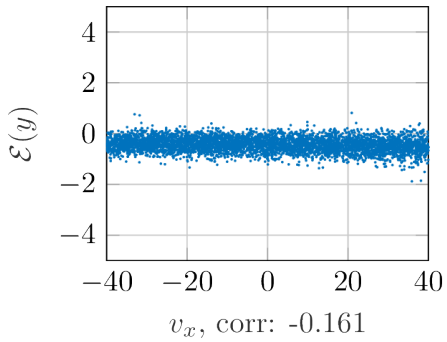

$\mathcal{E}(y)$
4
2
0
−2
−4
0
50
100
150
z, corr: -0.072

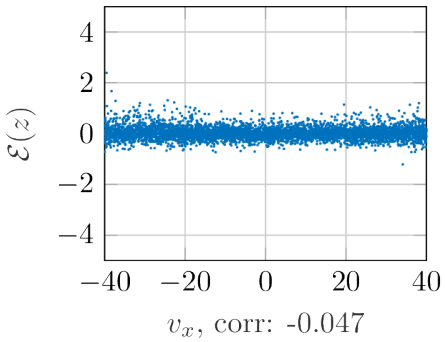

$\mathcal{E}(z)$
4
2
0
−2
−4
0
50
100
150
z, corr: -0.254

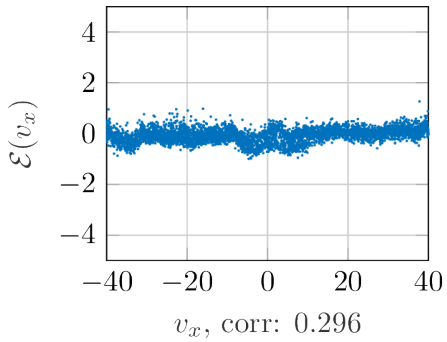

$\mathcal{E}(v_x)$
4
2
0
−2
−4
0
50
100
150
z, corr: -0.094

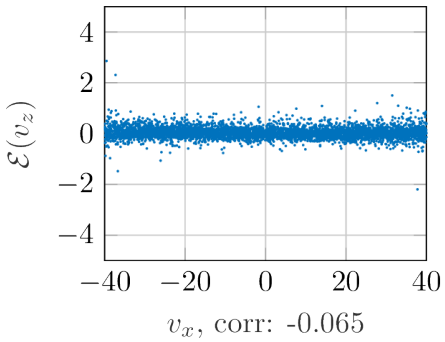

$\mathcal{E}(v_z)$
4
2
0
−2
−4
0
50
100
150
z, corr: -0.120

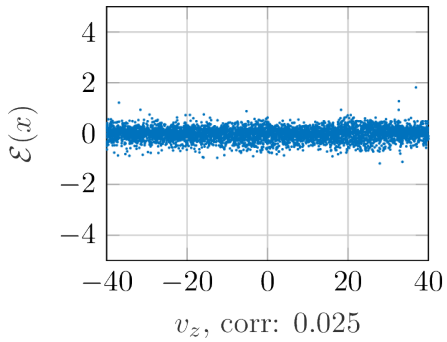

$\mathcal{E}(x)$
4
2
0
−2
−4
−40
−20
0
20
40
v_x, corr: 0.066

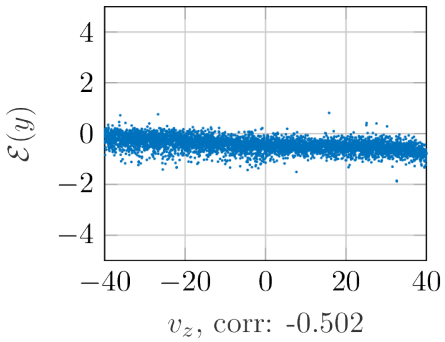

$\mathcal{E}(y)$
4
2
0
−2
−4
−40
−20
0
20
40
v_x, corr: -0.161


$\mathcal{E}(z)$
4
2
0
−2
−4
−40
−20
0
20
40
v_x, corr: -0.047


$\mathcal{E}(v_x)$
4
2
0
−2
−4
−40
−20
0
20
40
v_x, corr: 0.296


$\mathcal{E}(v_z)$
4
2
0
−2
−4
−40
−20
0
20
40
v_x, corr: -0.065


$\mathcal{E}(x)$
4
2
0
−2
−4
−40
−20
0
20
40
v_z, corr: 0.025


$\mathcal{E}(y)$
4
2
0
−2
−4
−40
−20
0
20
40
v_z, corr: -0.502

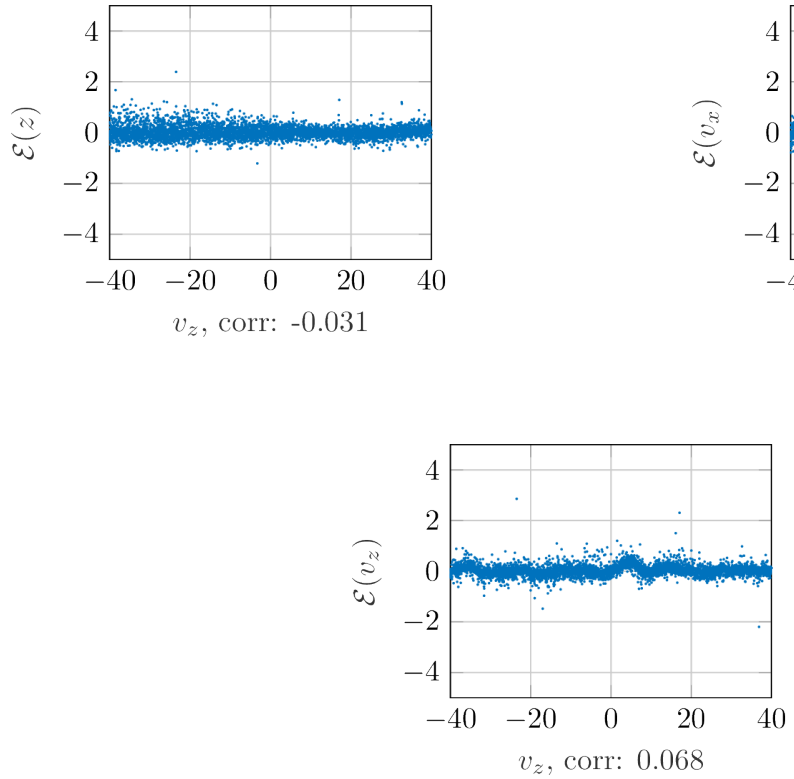
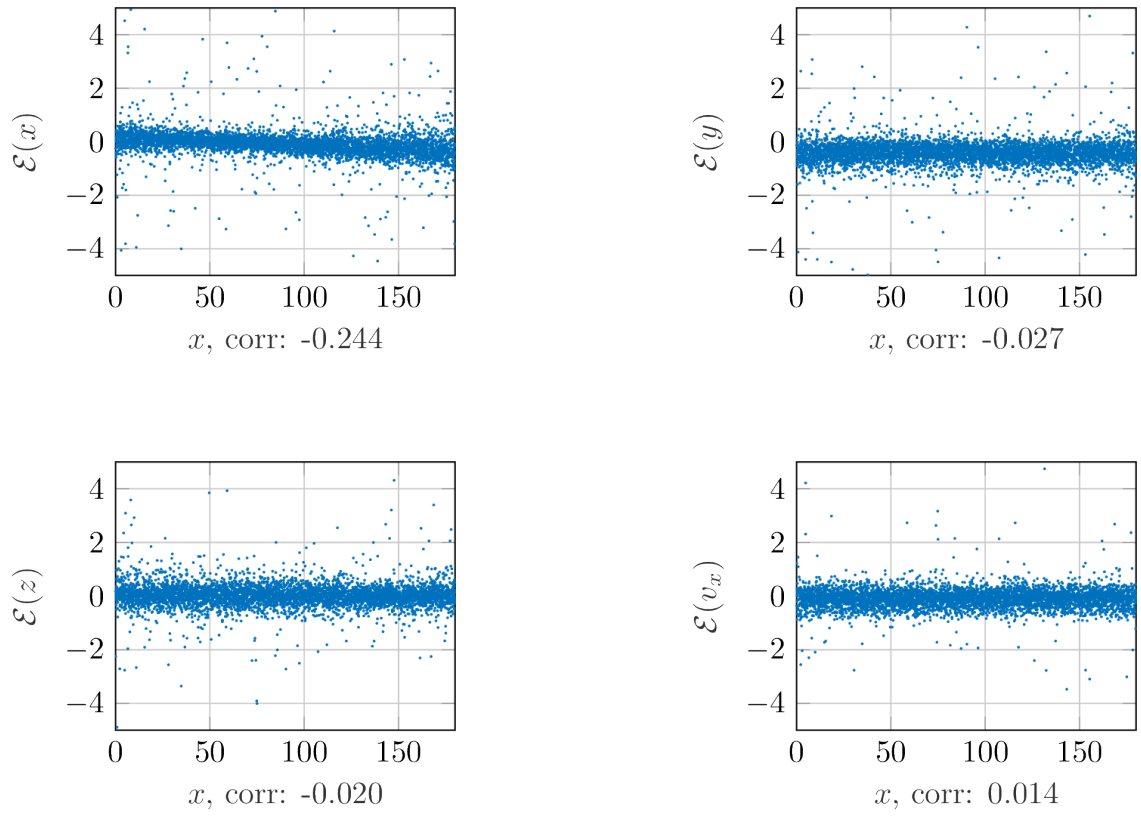
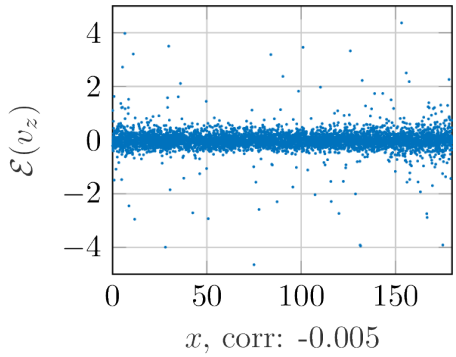
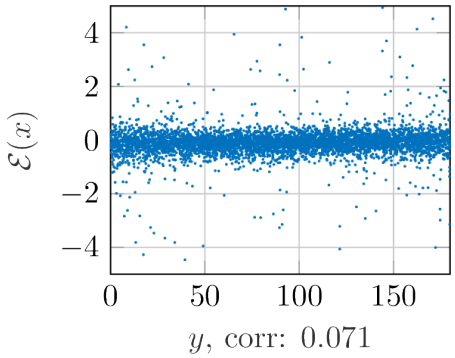
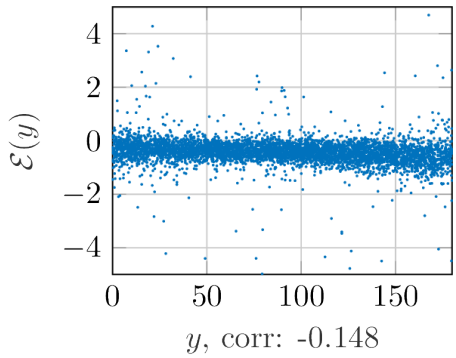


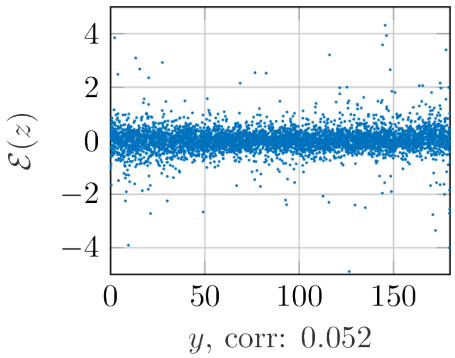
Figure C.13: Larger versions of the correlation plot for `DB476` using 20 000 novel datapoints as basis for the statistics.

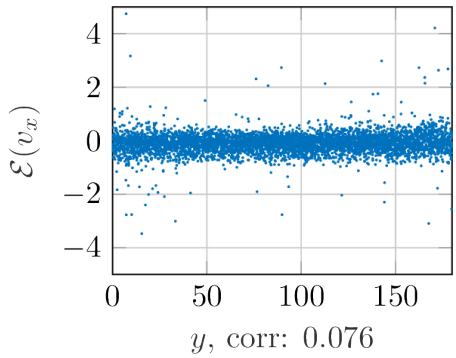


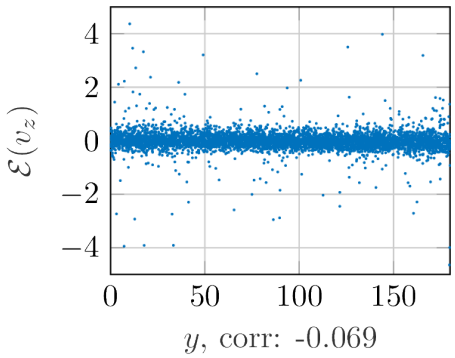

$\mathcal{E}(v_z)$
4
2
0
−2
−4
0
50
100
150
x, corr: -0.005

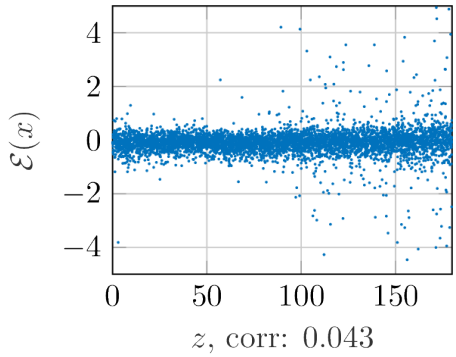

$\mathcal{E}(x)$
4
2
0
−2
−4
0
50
100
150
y, corr: 0.071

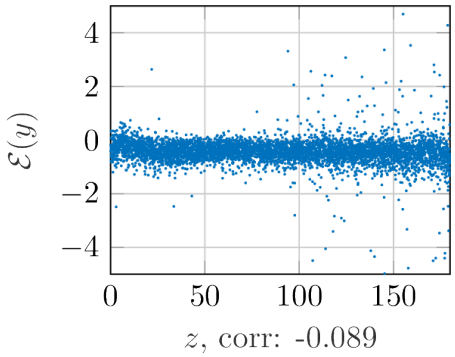

$\mathcal{E}(y)$
4
2
0
−2
−4
0
50
100
150
y, corr: -0.148

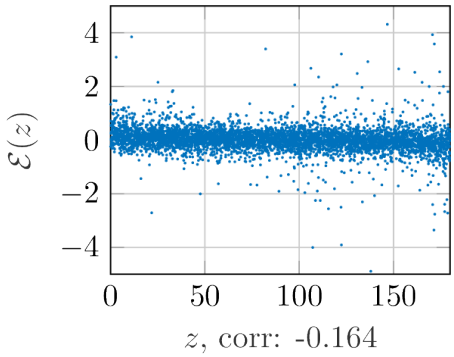

$\mathcal{E}(z)$
4
2
0
−2
−4
0
50
100
150
y, corr: 0.052

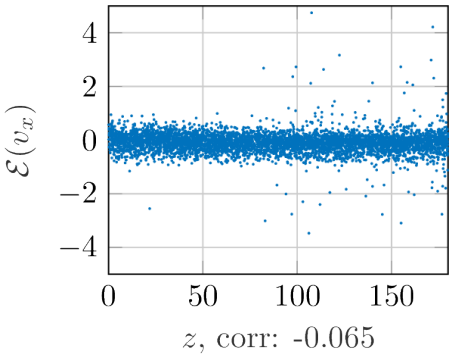

$\mathcal{E}(v_x)$
4
2
0
−2
−4
0
50
100
150
y, corr: 0.076

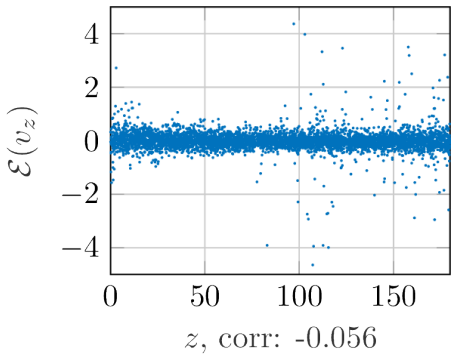

$\mathcal{E}(v_z)$
4
2
0
−2
−4
0
50
100
150
y, corr: -0.069

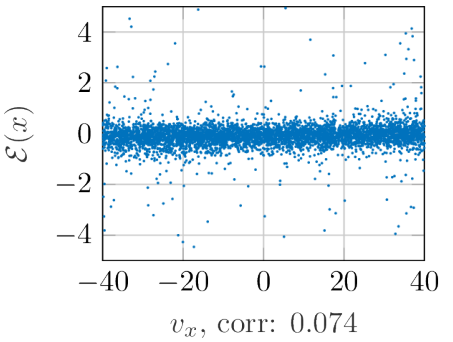

$\mathcal{E}(x)$
4
2
0
−2
−4
0
50
100
150
z, corr: 0.043

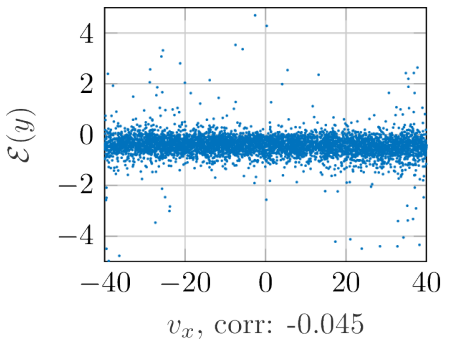

$\mathcal{E}(y)$
4
2
0
−2
−4
0
50
100
150
z, corr: -0.089

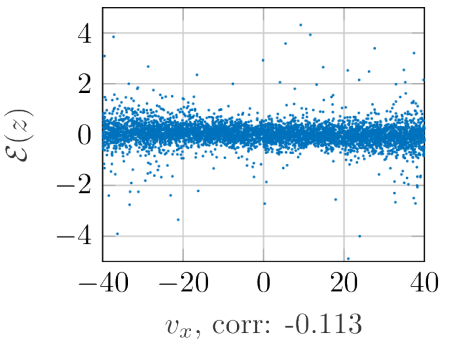

$\mathcal{E}(z)$
4
2
0
−2
−4
0
50
100
150
z, corr: -0.164

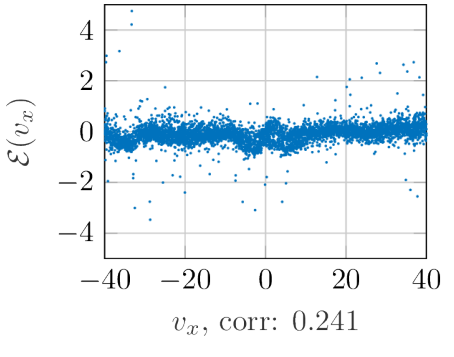

$\mathcal{E}(v_x)$
4
2
0
−2
−4
0
50
100
150
z, corr: -0.065

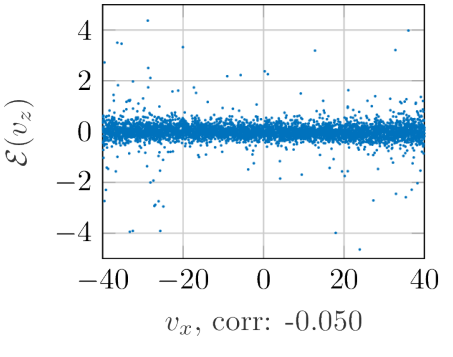

$\mathcal{E}(v_z)$
4
2
0
−2
−4
0
50
100
150
z, corr: -0.056


$\mathcal{E}(x)$
4
2
0
−2
−4
−40
−20
0
20
40
v_x, corr: 0.074


$\mathcal{E}(y)$
4
2
0
−2
−4
−40
−20
0
20
40
v_x, corr: -0.045


$\mathcal{E}(z)$
4
2
0
−2
−4
−40
−20
0
20
40
v_x, corr: -0.113


$\mathcal{E}(v_x)$
4
2
0
−2
−4
−40
−20
0
20
40
v_x, corr: 0.241


$\mathcal{E}(v_z)$
4
2
0
−2
−4
−40
−20
0
20
40
v_x, corr: -0.050

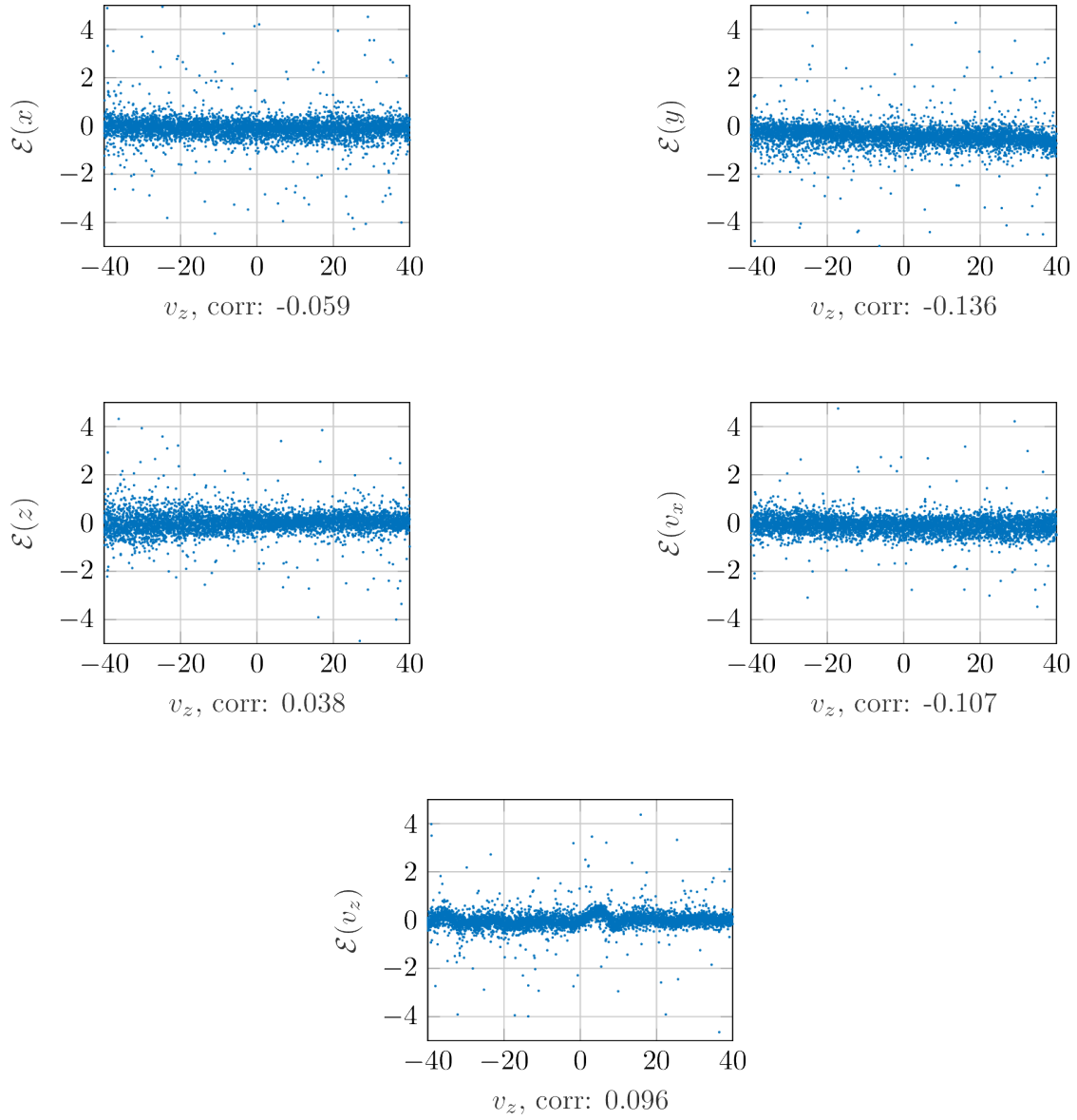


Figure C.26: Larger verions of correlation plot for `DB476` using 20 000 novel data points as basis for the statistics. Here the input to the neural network was divided by the average over all antennas to normalize the gain. The patterns are close to the standard model, confer Figure 5.11, but the points are more scattered.

D

Code listings

This appendix contains listings for the different ANN-implementations developed during this project.

D.1 Shell script for the TensorFlow implementation

Shell scripting and communication via environment variables were used to separate configuration and function in the TensorFlow script.

Listing 8: Shell script for invocation of TensorFlow.

```
#! /usr/bin/env bash
# Hyper parameter search using enviornment variables
export HPS_COMMENT='Layers and nodes for db1071'
export HPS_DATABASE='./MPOS_data/DB1071/DB1071+Eval.csv.scaled.csv'
export HPS_PATH='./layer_nodes_db1071'

for layers in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do
  for nodes in 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200
  do
    export HPS_NODES_PER_LAYER=$nodes
    export HPS_NUMBER_OF_LAYERS=$layers
    python tf_script.py
  done
done
```

D.2 TensorFlow implementation

The TensorFlow program was implemented using the Python API. In Listing 9 is a basic version of the program used to train the ANNs. Many experiments used alterations or additions of this code to test different aspects of the training.

Listing 9: A sample TensorFlow program in Python.

```
## -*- coding: UTF-8 -*-
import tensorflow as tf
import numpy as np
import math, random
import csv
import json
import os
```

```
import datetime
import time
import sys

COMMENT = os.environ['HPS_COMMENT']
NODES_PER_LAYER = int(os.environ['HPS_NODES_PER_LAYER'])
NUMBER_OF_LAYERS = int(os.environ['HPS_NUMBER_OF_LAYERS'])
HIDDEN_NODES = map(lambda x: NODES_PER_LAYER, range(0, NUMBER_OF_LAYERS))
NUM_FEATURES = 16
NUM_OUTPUTS = 5
NUM_EXAMPLES = 358400+10000
TRAIN_SIZE = 358400
VALID_SIZE = 5000

TRAIN_END = TRAIN_SIZE
VALID_START = TRAIN_END
VALID_END = VALID_START + VALID_SIZE
TEST_START = VALID_END

MINI_BATCH_SIZE = 4000
LEARNING_RATE = 0.0005
NUM_EPOCHS = 25000
weights = [] # Used to store best weights
biases = [] # Used to store best biases

mpos_data = []
database = os.environ['HPS_DATABASE']
with open(database, 'rt') as csvfile:
  mpos_reader = csv.reader(csvfile, delimiter=';')
  for row in mpos_reader:
    mpos_data.append(map(float, row))

path = os.environ['HPS_PATH']
if not os.path.exists(path):
  os.makedirs(path)

train = mpos_data[:TRAIN_SIZE]
np.random.shuffle(train) # Shuffle training data
mpos_eval = mpos_data[TRAIN_SIZE:]
np.random.shuffle(mpos_eval) # Shuffle evaluation
valid = mpos_eval[0:VALID_SIZE]
test  = mpos_eval[VALID_SIZE:]

get_features = lambda row: row[5:21] # Acceces the inputs/features in a CSV-row
get_values   = lambda row: row[0:5]  # Acceces the outputs/values in a CSV-row

def init_weights(shape, init_method='xavier', name=None, xavier_params = (None,
None)):
  if init_method == 'zeros':
    return tf.Variable(tf.zeros(shape, name=name, dtype=tf.float32))
  elif init_method == 'uniform':
    return tf.Variable(tf.random_normal(shape, name=name, stddev=0.01, dtype=tf.float32))
  elif init_method == 'xavier':
    (fan_in, fan_out) = xavier_params
    low = -np.sqrt(6.0/(fan_in + fan_out))
    high = np.sqrt(6.0/(fan_in + fan_out))
    return tf.Variable(tf.random_uniform(shape, name=name, minval=low, maxval=high,
 ↪   dtype=tf.float32))
  else:
    raise ValueError('Undefined init_method')

def model(X, hidden_nodes=[10]):
  prev_layer = X # Input layer is first "previous layer"
  ns = [NUM_FEATURES] + hidden_nodes

  # phi(x) = a tanh(bx)
  a = tf.constant(1.7159, dtype=tf.float32)
  b = tf.constant(2.0/3.0, dtype=tf.float32)

  # Append all hidden layers: use xavier initialization and zero bias
```

```
  for i in range(0,len(ns)-1):
    weights.append(init_weights([ns[i], ns[i+1]], 'xavier', name='weight_' + str(i),
↪     xavier_params=(ns[i], ns[i+1])))
    biases.append(init_weights([ns[i+1]], 'zeros', name='bias_' + str(i)))
    prev_layer = tf.scalar_mul(a, tf.nn.tanh(tf.scalar_mul(b, tf.add(tf.matmul(prev_layer,
↪     weights[-1]), biases[-1]))))

  # Finally add the output layer
  weights.append(init_weights([ns[-1], NUM_OUTPUTS], 'xavier', xavier_params=(ns[-1], 1)))
  biases.append(init_weights([NUM_OUTPUTS], 'zeros'))
  return tf.add(tf.matmul(prev_layer, weights[-1]), biases[-1])

def run_evaluation():
  file_name = path + "/%d_%d_%s.txt" % (NUMBER_OF_LAYERS, NODES_PER_LAYER,
↪     datetime.datetime.utcnow().isoformat())
  print("starting on file %s" % file_name)
  with tf.device('/gpu:0'):
    X = tf.placeholder(tf.float32, [None, NUM_FEATURES], name="X")
    Y = tf.placeholder(tf.float32, [None, NUM_OUTPUTS], name="Y")

    yhat = model(X, HIDDEN_NODES)
    loss = tf.nn.l2_loss(tf.subtract(Y, yhat))
    train_op = tf.train.AdamOptimizer(learning_rate=LEARNING_RATE).minimize(loss)

  sess = tf.Session(config=tf.ConfigProto(log_device_placement=False, allow_soft_placement=False))
  sess.run(tf.global_variables_initializer())

  with tf.device('/gpu:0'):
    #Initiate constant data
    trainx = sess.run(tf.constant(np.array(list(map(get_features, train)), dtype=np.float32),
↪     dtype=tf.float32))
    trainy = sess.run(tf.constant(np.array(list(map(get_values, train)), dtype=np.float32),
↪     dtype=tf.float32))

    validx = sess.run(tf.constant(np.array(list(map(get_features, valid)), dtype=np.float32),
↪     dtype=tf.float32))
    validy = sess.run(tf.constant(np.array(list(map(get_values, valid)), dtype=np.float32),
↪     dtype=tf.float32))
    testx = sess.run(tf.constant(np.array(list(map(get_features, test)), dtype=np.float32),
↪     dtype=tf.float32))
    testy = sess.run(tf.constant(np.array(list(map(get_values, test)), dtype=np.float32),
↪     dtype=tf.float32))

    RESCALE = tf.constant([300/2.0, 300/2.0, 300/2.0,    0.0,    0.0], dtype=tf.float32)
    ALPHA   = tf.constant([    0.0,     0.0,     0.0, 90/2.0,    0.0], dtype=tf.float32)
    BETA    = tf.constant([    0.0,     0.0,     0.0,    0.0, 90/2.0], dtype=tf.float32)
    # Define computations performed on GPU:0
    # Train #
    mean_train_radial_err = tf.reduce_mean(tf.sqrt(tf.reduce_sum(tf.square(tf.multiply(RESCALE,
↪     tf.subtract(trainy, yhat))), axis=1)))
    # Validation #
    mean_valid_rad_err = tf.reduce_mean(tf.sqrt(tf.reduce_sum(tf.square(tf.multiply(RESCALE,
↪     tf.subtract(validy, yhat))), axis=1)))
    # Test #
    test_rad_error = tf.sqrt(tf.reduce_sum(tf.square(tf.multiply(RESCALE, tf.subtract(testy,
↪     yhat))), axis=1))
    test_mean_radial_error = tf.reduce_mean(test_rad_error)
    test_max_radial_error = tf.reduce_max(test_rad_error)
    test_alpha_err = tf.sqrt(tf.reduce_sum(tf.square(tf.multiply(ALPHA, tf.subtract(testy, yhat))),
↪     axis=1))
    test_mean_alpha_err = tf.reduce_mean(test_alpha_err)
    test_beta_err = tf.sqrt(tf.reduce_sum(tf.square(tf.multiply(BETA, tf.subtract(testy, yhat))),
↪     axis=1))
    test_mean_beta_err = tf.reduce_mean(test_beta_err)

  errors = []
  radial_errors = []
  alpha_errors = []
  beta_errors = []
  mean_radial_errors = []
```

```
costs = []
min_validation_cost = 1.0E6
best_weights = []
best_biases  = []
mean_rad_err = 0
max_rad_err = 0
p95 = 0
train_rad_err = 0
mean_alpha = 0
mean_beta = 0
mean_train_rad_err = 0
start_time = time.time()
for i in range(NUM_EPOCHS):
  try:
    # Batch training
    permutation = np.random.permutation(TRAIN_SIZE) # Shuffle
    for start, end in zip(range(0, TRAIN_SIZE, MINI_BATCH_SIZE), range(MINI_BATCH_SIZE, TRAIN_SIZE,
↪   MINI_BATCH_SIZE)):
      sess.run(train_op, feed_dict={X: trainx[permutation[start:end]], Y:
↪   trainy[permutation[start:end]]})
    validation_cost = sess.run(loss, feed_dict={X: validx, Y: validy})
    if validation_cost < min_validation_cost:
      min_validation_cost = validation_cost
      # Log best weights
      best_weights = []
      for w in weights:
        best_weights.append(sess.run(w))
      # Log best biases
      best_biases = []
      for b in biases:
        best_biases.append(sess.run(b))
      # Train
      train_mean_rad_err = sess.run(mean_train_radial_err, feed_dict={X: trainx})
      # Test
      radial_errors = sess.run(test_rad_error, feed_dict={X: testx})
      alpha_errors = sess.run(test_alpha_err, feed_dict={X: testx})
      beta_errors = sess.run(test_beta_err, feed_dict={X: testx})
      mean_rad_err = sess.run(test_mean_radial_error, feed_dict={X: testx})
      max_rad_err  = sess.run(test_max_radial_error, feed_dict={X: testx})
      mean_alpha = sess.run(test_mean_alpha_err, feed_dict={X: testx})
      mean_beta = sess.run(test_mean_beta_err, feed_dict={X: testx})
      # Valid
      valid_mean_rad_err = sess.run(mean_valid_rad_err, feed_dict={X: validx})
    if i%1000 == 0: # Print online results
      duration = time.time() - start_time
      print("%d\tcost: %g\trad: %g \talpha: %g\tbeta: %g\tdur: %ds" % (i,min_validation_cost,
↪   mean_rad_err, mean_alpha, mean_beta, duration))
      mean_radial_errors.append(float(sess.run(mean_valid_rad_err, feed_dict={X: validx})))
      costs.append(float(sess.run(loss, feed_dict={X: validx, Y: validy})))
      start_time = time.time()
  except KeyboardInterrupt:
    # Allow early interrupt while still printing results!
    break # Exits training and writes session results

# Calculate p95
radial_errors.sort()
p95 = radial_errors[int(0.95*len(radial_errors))]
alpha_errors.sort()
p95a = alpha_errors[int(0.95*len(alpha_errors))]
beta_errors.sort()
p95b = beta_errors[int(0.95*len(beta_errors))]
# Write session results to file
f = open(file_name, 'a')
f.write("comment\t%s\n" % (COMMENT))
f.write("nodes\t%d\n" % (NODES_PER_LAYER))
f.write("layers\t%d\n" % (NUMBER_OF_LAYERS))
f.write("mini batch size\t%d\n" % (MINI_BATCH_SIZE))
f.write("learning rate\t%d\n" % (LEARNING_RATE))
f.write("database\t%s\n" % (database))
f.write("validation Radial (mm)\t%g\n" % (mean_rad_err))
```

```
  f.write("mean Alpha (°)\t%g\n" % (mean_alpha))
  f.write("mean Beta(°)\t%g\n" % (mean_beta))
  f.write("train Radial (mm)\t%g\n" % (mean_train_rad_err))
  f.write("max Radial (mm)\t%g\n" % (max_rad_err))
  f.write("p95 Radial (mm)\t%g\n" % (p95))
  f.write("p95 Vx (°)\t%g\n" % (p95a))
  f.write("p95 Vz (°)\t%g\n" % (p95b))
  f.write("mean radial errors\t%s\n" % (json.dumps(mean_radial_errors)))
  f.write("final radial error distribution\t%s\n" % (json.dumps(radial_errors.tolist())))
  log_biases(best_biases, file_name)
  log_weights(best_weights, file_name)
  sess.close()

def log_biases(biases, file_name):
  f = open(file_name + '_biases', 'w')
  for i, bias in enumerate(biases):
    f.write("bias_%d\t%s\n" % (i, json.dumps(bias.tolist())))

def log_weights(weights, file_name):
  f = open(file_name + '_weights', 'w')
  for i, weight in enumerate(weights):
    f.write("weight_%d\t%s\n" % (i, json.dumps(weight.tolist())))

run_evaluation()
```

E

Ethical concerns

The motivation for the system is clear; real-time local positioning during external beam radiotherapy has an instrumental value for patients by potentially lowering the damage done to healthy tissue, thereby avoiding side-effects such as bowel, urinary and potency problems. In this section some ethical implications of the proposed work are discussed.

E.1 Estimation of positions

The proposed system will produce estimates of the position and angles. Systematically wrong estimations will misguide other equipment and potentially cause severe harm to a patient undergoing radiotherapy. Estimation is necessary in the case of RayPilot, the physical problem requires approximations at several levels: voltage measurements, external interference and the positional recovery-process.

These facts are well known within the company and among customers. Anyhow, as a contributor to such a system it becomes important to acknowledge the limitations of the estimator in documentation and in communication with colleagues.

E.2 Pushing hypofractionation

Similar systems like Calypso (Varian Medical Systems) and Clarity (Elekta) are promoted as tools that enable hypofractionated *Stereotactic Body Radiation Therapy* (SBRT). The hypofractionation method uses fewer fractions but with a higher dose; SBRT indicates the use of positional tracking or fixtures in order to obtain higher precision.

In a randomized trial of hypofractionation and conventional fractionation of 168 patients, no difference in late[1] radiation toxicity were found, in another indicator of treatment efficiency hypofractionation performed slightly better [1]. There are

[1] as opposed to acute; months to years after radiation therapy

large ongoing studies (HyPro, CHHiP) to further explore the area of hypofractionation. An ethical question arises; Are equipment retailers pushing the change in methodology or is the transition fundamentally based on medical progress?

A sales argument for hypofractionation is the possibility to shorten health care waiting lists, since the number of factions are lower which allows a higher throughput of patients. This fact creates an economic incentive for the change in methodology which might challenge the patients individual interest in minimization of side effects. The best combination may very well be improved positioning with conventional fractionation.

Appendix bibliography

[1] G. Arcangeli, B. Saracino, S. Gomellini, M. G. Petrongari, S. Arcangeli, S. Sentinelli, S. Marzi, V. Landoni, J. Fowler, and L. Strigari, "A prospective phase iii randomized trial of hypofractionation versus conventional fractionation in patients with high-risk prostate cancer", *International Journal of Radiation Oncology* Biology* Physics*, vol. 78, no. 1, pp. 11–18, 2010.