Gabriel Schanner, BSc

BLE-based Indoor Positioning System using Neural Networks

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Assoc. Prof. Dipl.-Ing. Dr. techn. Robert Legenstein

Institute for Theoretical Computer Science
Head: Assoc. Prof. Dipl.-Ing. Dr. techn. Robert Legenstein

Graz, May 2019

Affidavit

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

Date	Signature

Abstract

While indoor positioning has been an important research topic for many years, no dominant standard has emerged yet. The Bluetooth Low Energy specification offers an interesting possibility for smartphone-based applications and was the chosen wireless technology for this work.

A reference environment of approximately 140 m^2 was equipped with 24 Bluetooth Low Energy beacons and a fingerprint database was created by gathering data at specific locations. A common occurrence during data gathering is the absence of signals from remote beacons, i.e. *missing data*. An Indoor Positioning System using Feedforward Neural Networks was developed with an emphasis on integrating this *missing data* information. The best performing architecture achieves a median test error of 1.624 m, significantly outperforming a k-nearest neighbors approach.

Data gathering was done using three distinct smartphone devices. Furthermore, several ways of aggregating data for network training and inference were investigated. The amount of received beacon signals during data gathering was found to vary significantly between devices, which has a strong impact on the Indoor Positioning System performance for hardware-specific data sets. The implemented Indoor Positioning System achieves promising results, but future work is needed to evaluate performance on non-stationary devices, i.e. moving clients.

Zusammenfassung

Obwohl Indoor Positioning bereits seit vielen Jahren ein wichtiges Forschungsthema ist, gibt es aktuell noch keinen dominanten Standard. Bluetooth Low Energy bietet interessante Verbesserungen zu existierenden Standards im Smartphone-Bereich und wurde für diese Arbeit als Drahtlostechnologie gewählt.

Es wurde eine ca. 140 m^2 große Referenzumgebung mit 24 Bluetooth Low Energy Beacons ausgestattet und eine Fingerprint Database erstellt. Während der Datenaufzeichnung geschieht es häufig, dass von weit entfernten Beacons keine Signale empfangen werden, hierbei handelt es sich um *Missing Data*. Ein Indoor Positioning System, basierend auf Feedforward Neural Networks, wurde entwickelt. Besonderer Fokus wurde hierbei auf die Integration von *Missing Data* in den Lernprozess gelegt. Das beste Modell erreicht einen Fehler Median von 1.624 m auf den Testdatensatz. Dies ist signifikant besser als ein verglichener k-nearest neighbors Ansatz.

Die Datenaufzeichnung erfolgte mit drei unterschiedlichen Geräten. Es wurden verschiedene Arten der Datenaufbereitung und Datenaggregation beim Erstellen von Datensätzen für das Trainieren von Netzwerken untersucht. Es zeigte sich, dass die Anzahl von empfangenen Signalen beim Datenaufzeichnen stark vom verwendeten Gerät abhängt. Dies hatte auch starke Auswirkungen auf die Genauigkeit des Indoor Positioning Systems bei Hardware-spezifischen Datensätzen.

Das Indoor Positioning System erzielt bereits vielversprechende Resultate. Weiterführende Arbeit in der Leistungsevaluierung von Datensätzen, welche aus der Datenaufzeichnung entlang von Pfaden stammen, ist notwendig.

Contents

List of Figures

1 Introduction

Determining the position of mobile devices has been an important research area for many years. Global Navigation Satellite Systems (GNSS), such as the Global Positioning System (GPS), have become part of daily usage habits for many smartphone users. However, GNSS do poorly in indoor environments, as the used signals do not penetrate buildings.

Indoor positioning Systems (IPS) aim to solve localization in these environments using a multitude of approaches. Some of the technologies and media used are: radio (e.g. WiFi, Ultra-Wide Band (UWB) and Bluetooth), light, audio, magnetic fields, orientation and angular velocity sensors. Solutions focusing on WiFi signals are currently among the most commonly used, as infrastructure (i.e. WiFi access points) is already existent in virtually all indoor environments. Nevertheless, positioning is usually rather coarse as increasing accuracy also means increasing the number of comparatively expensive access points. Bluetooth has been growing as an alternative, especially with the advent of the Bluetooth Low Energy (BLE) standard. BLE beacons are tailored specifically for indoor positioning. They are small, easy to deploy and can run on battery power for years.

In this work, an IPS implementation using BLE beacons is examined. The IPS makes use of location fingerprinting, which includes building a database that maps a set of Received Signal Strength (RSS) values of BLE beacons at a point in time to the location of the data-gathering device. This data is used to train a feedforward neural network (FFNN). Gathered signals are grouped together based on time windows, with the chosen window size being a trade-off between positioning accuracy and latency. The effects of varying window sizes is also subject of this work. Furthermore, depending on the window size, the environment layout and the number of deployed beacons, a client device might be missing signals from a set of beacons for

any given time window. Thus, among other traditional hyperparameters, two approaches of incorporating information about missing data into the FFNN are examined.

1.1 Outline

Chapter 2 explains techniques and technologies used in this work. Firstly, general indoor localization approaches are discussed, followed by a brief comparison of WiFi and Bluetooth technologies. Secondly, an introduction to FFNN is given including a brief explanation of the backpropagation algorithm. Finally, an overview of selected articles relevant to this work is given.

Chapter 3 goes into detail concerning the processes of data gathering and aggregation. The reference environment and devices are introduced and the structure of the gathered data is explained in detail.

The implemented IPS is discussed in Chapter 4, starting with a discussion on the approaches of how to incorporate missing data information. Afterwards, the individual modules of IPS are examined.

The stepwise process of evaluating different hyperparameter values for the IPS are discussed in Chapter 5. Furthermore, the consequences of using data from different devices in either concatenated or strictly separated datasets are shown. Lastly, the performance of a simple k-nearest neighbor approach is given when applied to the same datasets as the implemented IPS.

Finally, Chapter 6 offers discussion on the results and provides an outlook concerning future work.

2 Background

2.1 Indoor positioning using BLE

This work investigates the use case of indoor positioning (IP) using smartphones. There are various techniques for IP which utilize multiple data sources available in a typical device. This section gives an overview of the most common approaches for IP (trilateration and location fingerprinting [14]) and the data sources used, and will go into more detail about Bluetooth Low Energy (BLE), which is the technology that is focused on by the IP approach presented in Chapter 4.

Among the data sources that are relevant for IP and most commonly available in consumer smartphones are the following [6]:

- Radio Frequency (RF) Signals such as Wireless Local Area Network (WLAN, often also referred to as "Wi-Fi") and Bluetooth.
- Sensors such as Gyroscope, Accelerometer, Barometer and Magnetometer

Localization using received RF signals mainly rely on the two data points gained by such a signal: the identity of the sender and the Received Signal Strength Indicator (RSSI). The latter is a numeric value measured in decibel-milliwatts (dBm) and is proportional to the distance of sender and receiver [20]. Ideally the distance d is related to the signal power P_r in the inverse square law $P_r \propto d^{-2}$, but real world scenarios introduce various sources of noise, extending the previous formula to include other factors such as a parameter that requires the calibration of each pair of sender and receiver [8]. While these difficulties show that the RSSI is not completely reliable, depending on the actual environment and the level of calibration that was done, it is still a valid basis for IP.

2.1.1 Trilateration

The most direct IP approach is called trilateration and it works by first mapping the sender identity of received signals to its position. The sender id to position mapping needs to be maintained in some form of database and has to be input manually during installation. Then the RSSI is used to approximate the distance to this sender. When at least three distances to senders are known, trilateration is used to approximate the client's position.

2.1.2 Fingerprinting

Another common approach is fingerprinting. This technique works in two steps: an offline and an online step. For the offline part a fingerprint database is build by gathering RF signals for a set of positions (ideally all available positions) in the reference environment and storing the RSSI. These samples are often called fingerprints or calibration points and the database is sometimes referred to as a Radio Map. The core idea of this technique is to determine the "typical" set of signals for each location, usually in the form of RSSI means for each sender and position. This information is then used in the online step, which is the actual location computation. For a given set of gathered data the fingerprint database is searched for the best match using a defined criteria such as the Euclidean distance.

2.1.3 BLE compared to WiFi

The most commonly used RF technologies are WLAN and Bluetooth, with the latter gaining more popularity recently and especially since the introduction of the BLE standard. Some advantages of BLE over WLAN are:

- BLE beacons are generally cheaper than WLAN access points(AP).
- Lower energy consumption (BLE)
- BLE beacons are easily deployable in large numbers and thus provide more potential for higher localization accuracies.

Additionally there are further hindrances that WLAN faces when technical implementation details on smartphones are factored in, such as: lower scan interval, interference with actual network traffic and lack of access to RSSI of non-connected APs [9].

2.2 Neural Networks

The IPS discussed in Chapter 4 makes use of feedforward neural networks (FFNN). Neural networks (NN) in general are used to approximate some function f^* with a function $f(x;\theta)$, where x is the input and θ is a set of adjustable parameters. This is done using pairs of learning examples, which map x to the desired output t (target). The function approximation is improved during the training of the NN by iteratively evaluating the current approximation $f(x)$, computing the approximation error by comparing the output with the given target values t and using the computed error for adjusting the parameters θ [11].

A FFNN consists of multiple layers, where each layer consists of a set of nodes, each being a linear combination of its input applied to a nonlinear activation function. The "feedforward" part specifies that nodes of a layer only have connections to nodes of subsequent layers.

The following is an exemplary definition of a two layer FFNN, with the first layer defined as follows:

$$z_j = h^{(1)}\left(\sum_{i=1}^{D} x_i w_{j,i}^{(1)} + w_{j,0}^{(1)}\right) \tag{2.1}$$

with (1) indicating the layer, x_i with $i = 1,...,D$ indicate the FFNN input variables, $h^{(1)}(\cdot)$ being a nonlinear element-wise activation function, $w_{j,i}^{(1)}$ are weights and $w_{j,0}^{(1)}$ are biases. The set of adjustable parameters θ is given by weights and biases of all layers. $j = 1,...,M$ indicates the number of nodes in the current layer, with z_j being output of the j^{th} node.

The output layer is given analogously by:

$$y_k = h^{(2)} \left(\sum_{j=1}^{M} z_j w_{k,j}^{(2)} + w_{k,0}^{(2)} \right) \tag{2.2}$$

where y_k with $k = 1, ..., K$ indicate the FFNN outputs. This simple network architecture can be extended by adding more layers of the form that is shown in Equation (2.2) [3].

The activation function on the output layer (or simply output function) is chosen based on what kind of output is desired by the network and what error function is used. Common configurations include:

1. Binary classification: sigmoid output function with binary cross-entropy (BCE) error function
2. Classification for multiple classes: softmax output function with cross-entropy (CE) error function
3. Regression: linear output function with mean squared error (MSE) function

In this work the desired outputs are x and y coordinates on a 2-dimensional plane and all investigated FFNN architectures will follow the 3$^{\text{rd}}$ approach.

The total network function can thus be depicted as

$$y_k = h^{(2)} \left(\sum_{j=1}^{M} w_{k,j}^{(2)} h^{(1)} \left(\sum_{i=1}^{D} x_i w_{j,i}^{(1)} + w_{j,0}^{(1)} \right) + w_{k,0}^{(2)} \right) \tag{2.3}$$

A graphical representation of this network is shown in Figure 2.1. Evaluating this function for a given input is called forward propagation.

2.2.1 Activation Functions

While the Rectified Linear Unit (ReLU) is a generally recommended choice for the activation function, there are many other choices and determining which function yields the best results for a given problem set often requires a process of trial and error [11] [10]. Figure 2.2 shows the activation functions discussed in this section.

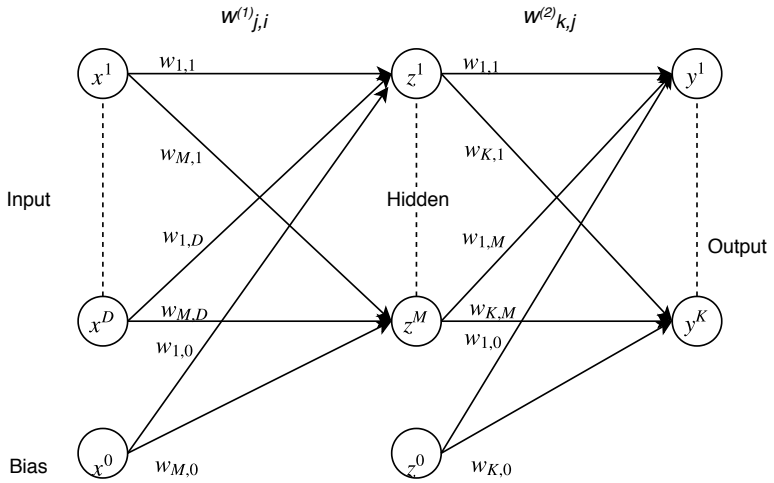

Figure 2.1: Graphical representation of the network function shown in Equation (2.3).

ReLU

The rectified linear unit is given by $f(x) = max\{0, x\}$. The derivative for $x <= 0$ is defined to be 0 and is 1 everywhere else. Its computationally cheap and performs well across a variety of domains [10].

ELU

The exponential linear unit is given as

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

and the derivative as

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ f(x) + \alpha & \text{if } x \leq 0 \end{cases}$$

It is identical to ReLU for positive x, but allows for negative values, which increases learning speed [4]. In this work the hyperparameter was chosen as $\alpha = 1$.

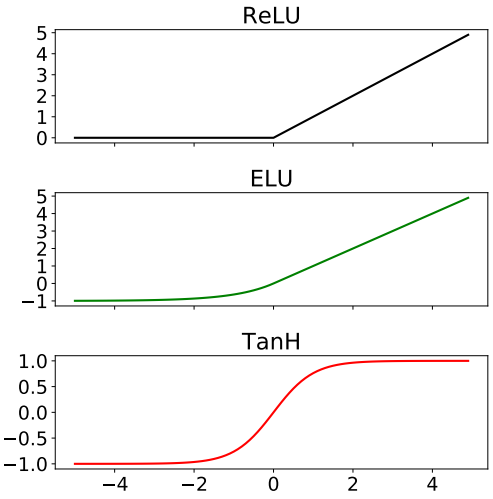

Figure 2.2: Activation functions discussed in Section 2.2.1

TanH

The hyperbolic tangent activation function $f(x) = tanh(x)$ is bound by $(-1, 1)$ and closely related to the sigmoid function, which is also a popular activation function. The derivative is given by $f'(x) = 1 - f(x)^2$ [11].

Backpropagation

The key ingredient in the iterative improvement during FFNN training is the continuous alternation between forward and backpropagation. By executing a forward pass, the network computes an estimation $\mathbf{y}$ for target $\mathbf{t}$ based on inputs $\mathbf{x}$ and adjustable parameters $\boldsymbol{\theta}$. The deviation from the desired output is then measured using some error function $E(\mathbf{w})$. With this information backpropagation is done, consisting of two main parts:

1. Calculate the derivatives of the error function $E(\mathbf{w})$ with respect to each weight.

2. Adjust the weight values based on the individual error derivatives and some update scheme. The simplest technique for doing this is called basic gradient descent.

A detailed description of the full algorithm is given in [3] and [11].

2.3 k-nearest neighbors

A k-nearest neighbors (kNN) algorithm is compared with the IPS developed in this work in Chapter 5. The implementation of the kNN regressor in the Scikit-learn [16] library is used for this purpose[1]. The kNN regressor algorithm works on P training samples, each consisting of multidimensional input and target data. It stores the P training inputs, each having S dimensions, in matrix $\boldsymbol{X}_{\text{Tr}} \in \mathbb{R}^{PxS}$. The P training targets, each having Q dimensions, are stored in matrix $\boldsymbol{Y}_{\text{Tr}} \in \mathbb{R}^{PxQ}$. Regression is done for a new data point (test input) $\boldsymbol{x}_{\text{Te}} \in \mathbb{R}^{S}$ by finding its k nearest neighbors in $\boldsymbol{X}_{\text{Tr}}$. Nearness is calculated using a distance function, e.g. Euclidean distance. The final target prediction $\boldsymbol{y}_{\text{Te}} \in \mathbb{R}^{Q}$ is calculated by using the average of each of the Q dimensions of the rows in $\boldsymbol{Y}_{\text{Tr}}$ that correspond to the k nearest neighbors. The pseudocode of these operations is given in Algorithm 1.

Input: Training inputs $\boldsymbol{X}_{\text{Tr}}$ and corresponding Training targets $\boldsymbol{Y}_{\text{Tr}}$
Test inputs $\boldsymbol{X}_{\text{Te}}$
k neighbors
Result: Test predictions $\boldsymbol{Y}_{\text{Te}}$
foreach $\boldsymbol{x}_{Te} \in \boldsymbol{X}_{Te}$ **do**
  compute Euclidean distances to each $\boldsymbol{x}_{\text{Tr}} \in \boldsymbol{X}_{\text{Tr}}$
  select k data point indexes with lowest distance
  $\boldsymbol{y}_{\text{Te}}$ = featurewise average of the k selected training targets $\boldsymbol{y}_{\text{Tr}}$
end

Algorithm 1: Pseudocode for kNN regression.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html The description in this work assumes the following parameter choices: **weights**:uniform, **n_neighbors**:10, algorithm:brute, **metric**:minkowski, **p**:2

2.4 Related Work

In the literature, many different approaches regarding IPS techniques have been investigated. They generally fall into the two categories explained above: trilateration 2.1.1 and fingerprinting 2.1.2. This section goes into more detail on selected works.

An indoor location-aware system for an IoT-based smart museum has been developed in [1]. The system makes use of wearable devices, which deliver additional information about artworks at landmarks in the environment to the visitor. They are equipped with two modules: position tracking using BLE and image processing for recognizing nearby artworks.
The wearable gathers signal data from various landmarks in the environment, each acting as a designated BLE base station. The positioning is done by computing a proximity index d based on the log distance path loss model $RSSI = -(10nlog_{10}d + A)$. The landmark with the lowest d is then selected as the position. Thus, this represents a proximity-based positioning approach. The image processing module then includes the position information when determining the observed artwork.
It was found that incorporating the position information into the artwork determination phase yielded significant computation performance increase (i.e. shorter processing time) and a slight increase in accuracy.

Bluepass[7] is a Bluetooth-based IPS that uses linear regression on a fingerprinting database for distance estimation and then uses a proximity-based model to determine the position on a room level. In a reference environment, consisting of 4 rooms in a 13m x 15m area, 80% room level accuracy was achieved using three Bluetooth access points.

[9] analyses BLE fingerprinting and compares its BLE-based IPS with a WiFi-based system. It is shown that the three advertising radio channels used in BLE have different RSS mean levels and that all channels suffer of multipath fading. To address these negative effects, fingerprints are build on time windows, so that multiple signals from each BLE beacon are aggregated using either median, mean or maximum.
Position computation in the IPS is done by first discretizing the environment into cells (1m side length) and building a fingerprint database(offline phase). In the online phase, a position is evaluated by computing the Euclidean

distance to each cell and then using the Bayesian Likelihood function to determine the most likely cell.
The testbed consisted of 19 beacons placed in a 600m² environment. An error of $< 2.6m$ for 95% of the time was achieved.

[2] is a fingerprinting approach using Bluetooth. It states the importance of Bluetooth technology for IP, while also acknowledging the inconsistencies when using RSS values for this purpose.
The system they designed has a multilayered architecture that uses multiple NN specialized for various user directions (i.e. the cardinal direction of a user holding the localization device). The NN input vector consists of the last eight RSS values for each of the five deployed Bluetooth base stations. The various positions that a user could be in are discretized to eight distinct 2D coordinates. Thus, the NN output layer consists of eight nodes, each representing one of the possible positions (one-hot encoding). Furthermore, the positions are also stored as a connected graph, indicating the possible position transitions. The graph is used to prevent rapid fluctuations in the predicted position by only reporting a position change if it is possible (according to the graph) or if the change has been reported a configurable amount of times consecutively.
A Recovery Subsystem was implemented that addresses the case of Bluetooth base stations becoming unresponsive. This was done by training NNs with configurations that include "turned off" subsets of Bluetooth bases stations.
Results show significant improvement when including the adaptions for user orientation (using multiple NNs), filtering unlikely positions (using the connection graph) and node failure (using the recovery subsystem). Best reported performance while walking was 90% accuracy.

[21] is a fingerprinting approach using WLAN. It mentions the dependence of RSS-based IPS solutions on the amount of chosen reference points (RP) and their positioning. A NN-based IPS is presented, which focuses on the prediction of positioning errors that occur using a distance dependent algorithm.
Positioning is done by first building an offline radio map which serves as the data source for the chosen distance dependent algorithm kNN. Training data for the NN is gathered by sampling RSS data on a selected set of reference points with coordinates (x_{Tr}, y_{Tr}) and then estimating the posi-

tion for these reference points using kNN $(\hat{x}_{Tr}, \hat{y}_{Tr})$. The estimation errors $\delta x_{Tr} = x_{Tr} - \hat{x}_{Tr}$ and $\delta y_{Tr} = y_{Tr} - \hat{y}_{Tr}$ are then the targets for which the NN is trained upon.
In the online phase the IPS then calculates a position by adding NN-predicted estimation errors $(\delta x_{Te}, \delta y_{Te})$ to kNN-predicted positions $(\hat{x}_{Te}, \hat{y}_{Te})$ to obtain final positioning coordinates $(\hat{x}_P, \hat{y}_P)$.
When compared to basic separate kNN and NN approaches (2.54m and 2.42m mean error respectively), the combined approach showed significant improvement(1.33m mean error).

Another Bluetooth based fingerprinting approach is discussed in [19]. A set of predefined coordinates in a reference environment of 120 m^2 were selected for data gathering with a mobile device. After this offline phase was finalized and the fingerprint database built, positioning accuracy was evaluated in the online phase. Finding target coordinates in the online phase is done using kNN regression and an accuracy of 2.67 m was accomplished.

[13] gives a brief introduction of the positioning solutions for smartphones and explain the basic workflow of the fingerprinting approach. They introduced a PDR (Pedestrian Dead Reckoning) solution that makes use of BLE (Bluetooth low energy), an accelerometer and a digital compass. BLE was used for corridor detection. Two BLE access points were placed in two separate corridors and by finding peaks in the time series of the RSSI measurements, the current corridor was determined. The accelerometer was used for step frequency estimation. Step length was estimated using an empirical model with 3 parameters (pedestrian height, step frequency and an optional personal parameter). Furthermore, a digital compass was used to determine the walking direction of the pedestrian. Initial positioning is done by setting the current position to the same as the nearest BLE access point. Afterwards, the position is determined by tracking the movements of the pedestrian, while also recalibrating each time an access point was passed. The average positioning error was $1.88m$.

A technique called Parallel Multilayer Neural Network (PMNN), which uses two distinct neural networks for the x and y axis respectively, is discussed in [5]. The 144 m^2 reference environment is equipped with numerous WiFi access points and discretized into squares with varying side lengths. A mobile phone is used to build an initial fingerprint database. Neural network

inputs are vectors containing the RSSI value for each beacon and the targets are the x coordinate for one network and the y coordinate for the other. Each network also contains a denoising section in the form of multiple layers of a pretrained autoencoder. The autoencoder was trained by stochastically switching inputs off (i.e. setting the RSSI value to 0) and after training, the same autoencoder is used for denoising the inputs to both networks. The performance is compared with a kNN approach, which is consistently outperformed by the PMNN approach. The best configuration achieves close to 95% accuracy for a grid of squares with a side length of 2 m.

3D indoor positioning, using spectral clustering and a weighted backpropagation network (SWBN), is discussed in [17]. The 3D environment is discretized to grid points (reference points) and a fingerprint database is built by sampling RSSI values at these points (offline phase). Reference points are grouped together in clusters and for each cluster a neural network is trained. In the online phase, coordinates are calculated using a weighted combination of the outputs from the various networks. Evaluation results for the 16 m $\times$ 7.76 m $\times$ 3.25 m reference environment show a median error of 1.48 m.

3 Data Gathering and Aggregation

The goal of the data gathering and aggregation part of this work is to first generate a reference dataset that lists received BLE samples and their RSSI value for a variety of locations (encoded as x and y coordinate) and then apply various transformation to this reference dataset and create aggregations for further use. Furthermore, the reference dataset should also include timestamp information as well as a dataset identifier that allows each sample to be mapped to a specific device with which it was recorded. Datasets created from this reference dataset are then used to train neural networks, with the target of predicting the location for a given sample. Furthermore, these predictions should be possible in a real word scenario, which means that the sampling time of BLE beacon values is limited. To incorporate this constraint into the network training process, the time binned dataset introduced in Section 3.2.3 partitions data based on time frames.

All data was gathered in the office environment depicted in Figure 3.1. The office consists of three rooms and one almost circular hallway. All doors were opened up during data collection and no people other than the data collector were present. During data collection 24 BLE beacons were active in the environment, their location can also be seen in Figure 3.1.

3.1 Data Gathering Pipeline

This section discusses how BLE signal data was gathered and describes the structure of the gathered data.

To be able to log received BLE signals, a custom Android application was created (see screenshot depicted in Figure 3.2). This application was used to create six datasets. Three devices were used and for each device a dataset

Figure 3.1: Map of the reference environment. The dark gray area depicts outside walls, the light gray area depicts inside walls/objects and the medium gray area depicts walkable space. The icons represent the locations of BLE beacons. The origin $0, 0$ is in the left upper corner. The x coordinate runs horizontally and the y coordinate vertically.

was created where the data collector faced towards the northern end of the map and one facing the southern. The following three devices were used:

- One Plus One[1], henceforth the abbreviation **OPO**.
- Huawei P20[2], henceforth the abbreviation **P20**.
- Samsung Galaxy S9[3], henceforth the abbreviation **S9**.

The workflow of creating a dataset with the collection application is as follows:

1. Specify dataset name.
2. Go to a location and enter the X and Y coordinate for this location in the app.
3. Press the "Scan BLE" Button. The application now stores all BLE signals received within the next 5 seconds and labels them with the location specified in the previous step.
4. Repeat Steps 2. and 3. for each location.
5. Send the data to a specified server, which stores the received data in a MongoDB database.

A set of 52 locations were selected for the creation of the datasets. These locations are scattered throughout the environment and represent positions that are easily accessible for a person walking through the setting. All recordings were done with the mobile phones being held in the same height. While all locations were sampled for the same duration with all devices, the number of collected BLE signals varied strongly as can be seen in Figures 3.3 and 3.4. The exact reason for these differences is hard to determine and was not further researched in this work. Some possible explanations might be differences in the quality of the Bluetooth chipsets and Bluetooth stack implementation differences in different Android versions.

After data was gathered for all devices in this manner, the data was exported from the MongoDB database into a csv file. An excerpt of such a dataset export can be seen in Listing 1. The data in this listing originates from the northwards facing Huawei P20 dataset. To explain further what kind of data is available at this step in the process, a description of all columns follows.

[1] https://www.oneplus.com/at/one
[2] https://consumer.huawei.com/en/phones/p20/
[3] https://www.samsung.com/global/galaxy/galaxy-s9/

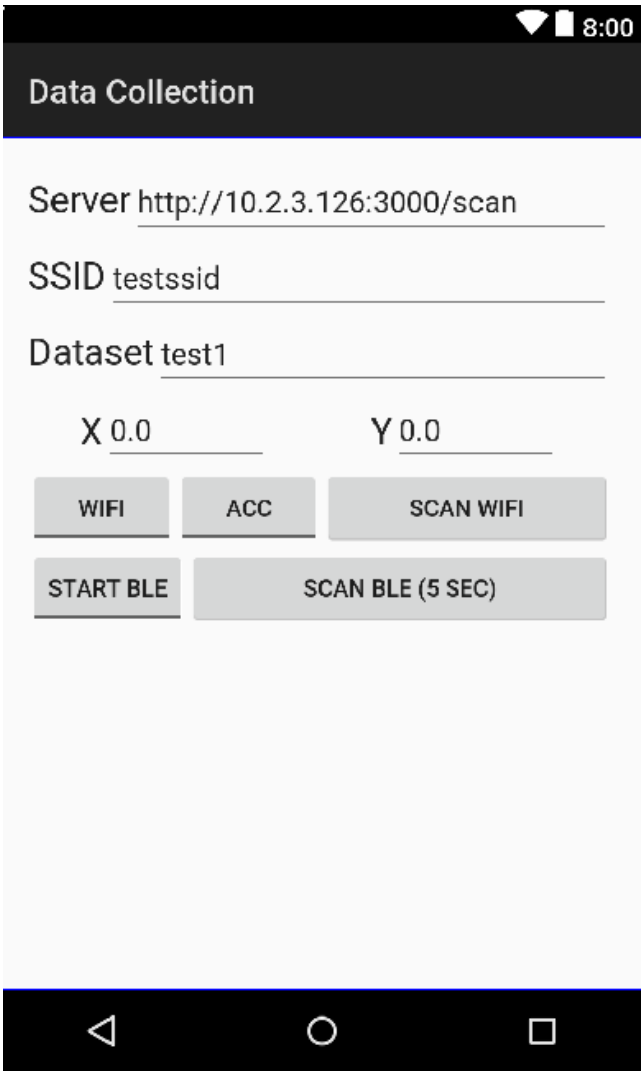

Figure 3.2: The main view of the android data collection application. The "Server" input specifies the location of a REST endpoint that will accept the data collected by the application. The "SSID" field is irrelevant for this work. The "Dataset" field specifies the value of the *data_set* field of the logged data (see Listing 1). The "X" and "Y" fields are manually inputted by the data collector for each location and together they determine the position that will be used for actions triggered by one of the buttons. Of the five depicted buttons only the one labeled "SCAN BLE (5 SEC)" was used for the data collection purposes of this work.

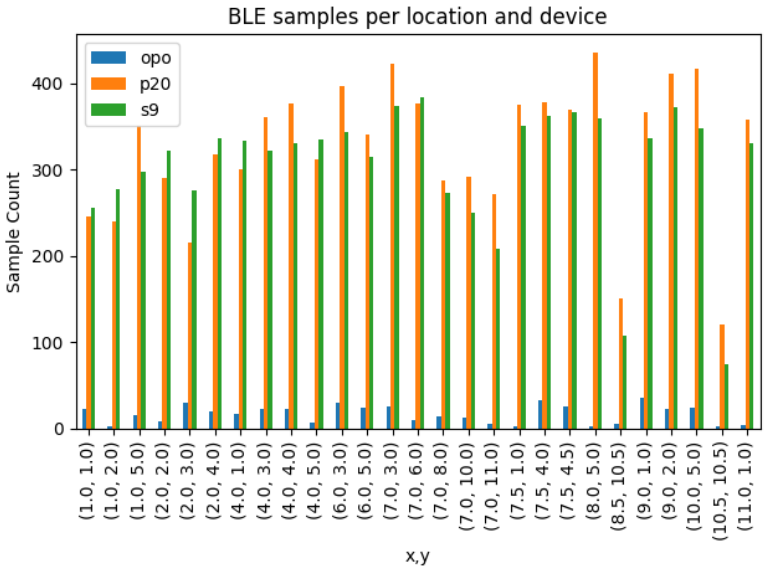

Figure 3.3: Number of BLE samples collected for each device and location. This figure shows the first 26/52 locations.

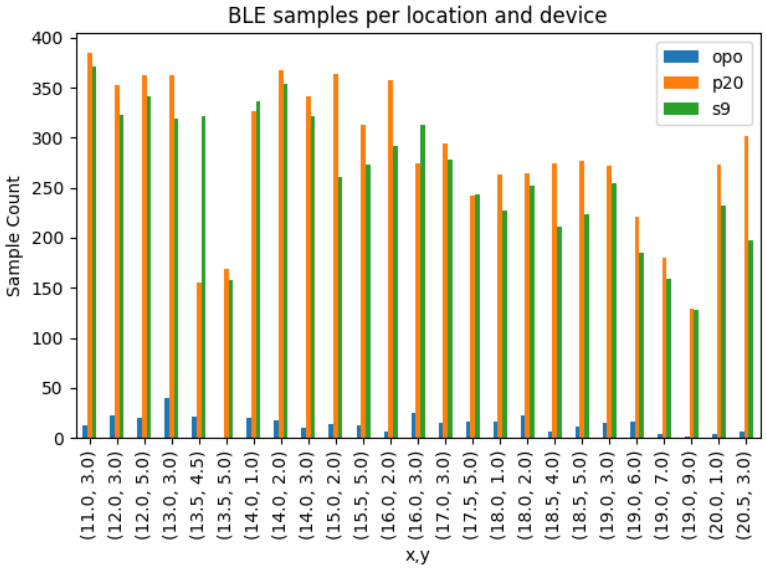

Figure 3.4: Number of BLE samples collected for each device and location. This figure shows the second 26/52 locations.

S.1 **data_set** An id of the dataset that was manually inputted by the collector. The general scheme is <*shortened device name*>_<*cardinal direction*>.

S.2 **device_name** A string expressing information about the BLE beacon manufacturer.

S.3 **signal_type** The data collection android app also supports the collection of WiFi samples. Thus this field indicates if the sample was of signal type "ble" or "wifi".

S.4 **timestamp** The timestamp of collection given in UNIX time in milliseconds.

S.5 **ble_mac** The MAC address of the BLE beacon.

S.6 **ble_major** A read and writable field of the BLE beacon. Manually set. Should be the same for all BLE beacons of the same manufacturer.

S.7 **ble_minor** A read and writable field of the BLE beacon. Used in conjunction with **ble_major** as an alternative way of uniquely identifying beacons.

S.8 **rssi** The RSSI value of the received BLE sample.

S.9 **pos_x** The x-axis coordinate of the position where this sample was gathered. Given in meters. Manually set in data collection application for each location by the collector.

S.10 **pos_y** The y-axis coordinate of the position where this sample was gathered. Given in meters. Manually set in data collection application for each location by the collector.

```
"data_set","device_name","signal_type","timestamp","ble_mac","ble_major","ble_minor","rssi","pos_x","pos_y"
"p20_north","RECO","ble","1529519080792","ED:72:88:FE:FC:7F","501","25542","-93","19","9"
"p20_north","RECO","ble","1529519080808","C9:48:FC:EA:75:AF","501","5277","-94","19","9"
...
"p20_north","RECO","ble","1529519276354","DF:46:A2:D8:93:35","501","25539","-76","7.5","4.5"
"p20_north","RECO","ble","1529519276382","EC:FA:86:B8:9D:57","501","5257","-68","7.5","4.5"
"p20_north","","ble","1529519276384","20:C3:8F:C2:D7:36","4096","8449","-88","7.5","4.5"
"p20_north","","ble","1529519276440","20:C3:8F:C2:C1:07","4096","8450","-69","7.5","4.5"
"p20_north","RECO","ble","1529519276472","E5:ED:2D:4F:F0:96","501","4816","-57","7.5","4.5"
"p20_north","RECO","ble","1529519276532","D6:8A:F1:9D:E0:E0","501","5278","-80","7.5","4.5"
"p20_north","","ble","1529519276550","20:C3:8F:C2:BE:7D","4096","20480","-66","7.5","4.5"
"p20_north","","ble","1529519276580","20:C3:8F:C2:D7:11","4096","4113","-76","7.5","4.5"
"p20_north","RECO","ble","1529519295338","EE:D4:A4:CC:1F:67","501","25544","-74","7","6"
"p20_north","RECO","ble","1529519295399","D6:8A:F1:9D:E0:E0","501","5278","-72","7","6"
"p20_north","","ble","1529519295420","20:C3:8F:C2:D4:B8","4096","8192","-60","7","6"
...
```

Listing 1: Excerpt from the the MongoDB data dump of the dataset of the Huawei P20 device while facing north.

BLE beacon information

While the machine learning approach introduced in Section 4 does not require the actual x and y coordinates of the BLE beacon locations, a reference list of all valid BLE beacons is still used in the data aggregation process (see the following Section 3.2). This BLE beacon information dataset is given in Listing 2 and the columns signify the following:

1. **y** The y-axis coordinate of the beacon. Given in meters.
2. **beaconType** A string containing information about the BLE beacon manufacturer.
3. **minor** The same as described in Item S.7.
4. **major** The same as described in Item S.6.
5. **x** The x-axis coordinate of the beacon. Given in meters.

```
y,beaconType,minor,major,x
1.22,BLE_WELLCORE,4105,4096,18.065
2.455,BLE_WELLCORE,4104,4096,15.52
4.95,BLE_WELLCORE,4113,4096,14.922
2.45,BLE_WELLCORE,4101,4096,18.03
1.22,BLE_WELLCORE,4102,4096,19.9
4.95,BLE_WELLCORE,4103,4096,9.297
2.46,BLE_WELLCORE,4116,4096,19.89
4.95,BLE_WELLCORE,20480,4096,11.797
4.95,BLE_WELLCORE,4114,4096,20.527
10.57,BLE_WELLCORE,8192,4096,7.417
8.07,BLE_WELLCORE,8450,4096,7.417
10.57,BLE_WELLCORE,8449,4096,10.557
4.95,BLE_WELLCORE,8451,4096,18.037
0.58,BLE_WELLCORE,4100,4096,15.52
8.7,BLE_WELLCORE,8448,4096,18.627
0.545,BLE_RECO,5277,501,9.103
0.56,BLE_RECO,5255,501,0.525
3.06,BLE_RECO,5285,501,0.525
5.56,BLE_RECO,5278,501,0.525
0.375,BLE_RECO,5264,501,4.275
3.07,BLE_RECO,5256,501,4.255
5.56,BLE_RECO,5263,501,4.275
2.715,BLE_RECO,5257,501,9.143
3.03,BLE_RECO,22025,501,14.273
5.33,BLE_RECO,25540,501,7.417
2.9,BLE_RECO,25544,501,7.56
0.56,BLE_RECO,25539,501,7.56
2.715,BLE_RECO,25542,501,11.783
0.7,BLE_RECO,25543,501,11.803
0.54,BLE_RECO,99999999,99999999,14.273
```

Listing 2: The BLE beacon information dataset.

3.2 Aggregation

This section discusses the steps necessary to transform the raw data collected in the previous Section 3.1 into datasets that are suitable for the machine learning approaches discussed in Section 4.

3.2.1 Reference dataset

The dataset dubbed as *reference dataset* serves as the source for all further data aggregations discussed in this section. It is created by first concatenating

all six datasets gathered in Section 3.1 and then performing a series of manipulations on it. In detail, the operations done in order to create the reference dataset from the concatenations of the six individual datasets (see e.g. Listing 1) are as follows:

1. Remove all samples whose *ble_minor* and *ble_major* is not listed in the BLE beacon information dataset (see Section 3.1). This is necessary since the android data collection application logs absolutely all Bluetooth signals received. This includes i.e. BLE beacons that are currently not in use (and cannot be turned off) and various Bluetooth based peripherals such as mice and headsets.
2. Map the Bluetooth fields *ble_minor*, *ble_major* and *ble_mac* to a single numerical *ble_id*. This is done by simply enumerating all entries in Listing 2 and using this value as the id.
3. Map the *data_set* field to a numerical value.
4. Remove the columns *device_name* and *signal_type* since they are not required for future use cases.
5. Do a thresholding on the *rssi* values. Samples with $rssi \geq -10$ or ≤ -90 were discarded.

An excerpt of the resulting reference dataset can be seen in Listing 3. This dataset now lists all received BLE signals for each hardware, location and BLE beacon. However, to be able to train FFNNs with this data, a single training sample should have a target position and the RSSI values for every BLE beacon.

```
timestamp,data_set,rssi,pos_x,pos_y,ble_id
1529517369996,3,-80,19.0,9.0,14.0
1529517370029,3,-78,19.0,9.0,12.0
1529517370056,3,-77,19.0,9.0,4.0
1529517370076,3,-70,19.0,9.0,8.0
1529517370308,3,-75,19.0,9.0,6.0
...
1529517675778,3,-87,8.0,5.0,19.0
1529517675870,3,-84,8.0,5.0,26.0
1529517701787,3,-75,7.5,4.5,17.0
1529517701847,3,-80,7.5,4.5,26.0
...
```

Listing 3: Excerpt from the the reference dataset. The full file has 31539 samples.

3.2.2 Location Binned Dataset

The naive approach to creating a dataset that lists the RSSI values for each position is to average all RSSI values by position and BLE beacon (the creation process is depicted in Figure 3.5). This results in a dataset having as many samples (rows) as there are distinct positions in the dataset and having as many features (columns) as there are beacons. Information gained by incorporating the timestamp data of each received signal is ignored with this approach, but it has the advantage of having the most accurate RSSI values that are available for each position. While the value in this approach is purely theoretical, as it ignores the real world constraint of limited time frame length for BLE signal collection, it serves as a comparative reference for further approaches.

```
Beacon 2 AVG RSSI,... ,Beacon 28 AVG RSSI,X,Y
-89.75,... ,-85.48,1.0,1.0
-89.50,... ,-84.42,1.0,2.0
-86.48,... ,-81.71,1.0,5.0
...
-76.72,... ,-78.87,8.0,5.0
 00.00,... ,-89.27,8.5,10.5
-84.24,... ,-74.77,9.0,1.0
...
```

Listing 4: Excerpt from the the location binned dataset. The original file has one sample per location for a total of 52 samples. Note that while there were 24 beacons in use during data gathering, the original beacon meta data file (see Listing 2) lists 6 older beacons that cause the offset in the beacon ids shown in the header of this listing.

3.2.3 Time Binned Dataset

To more accurately reflect the real world scenario of limited time data sampling, this approach groups data by time bins. The procedure of doing so is as follows:

1. Sort the reference dataset by the timestamp column

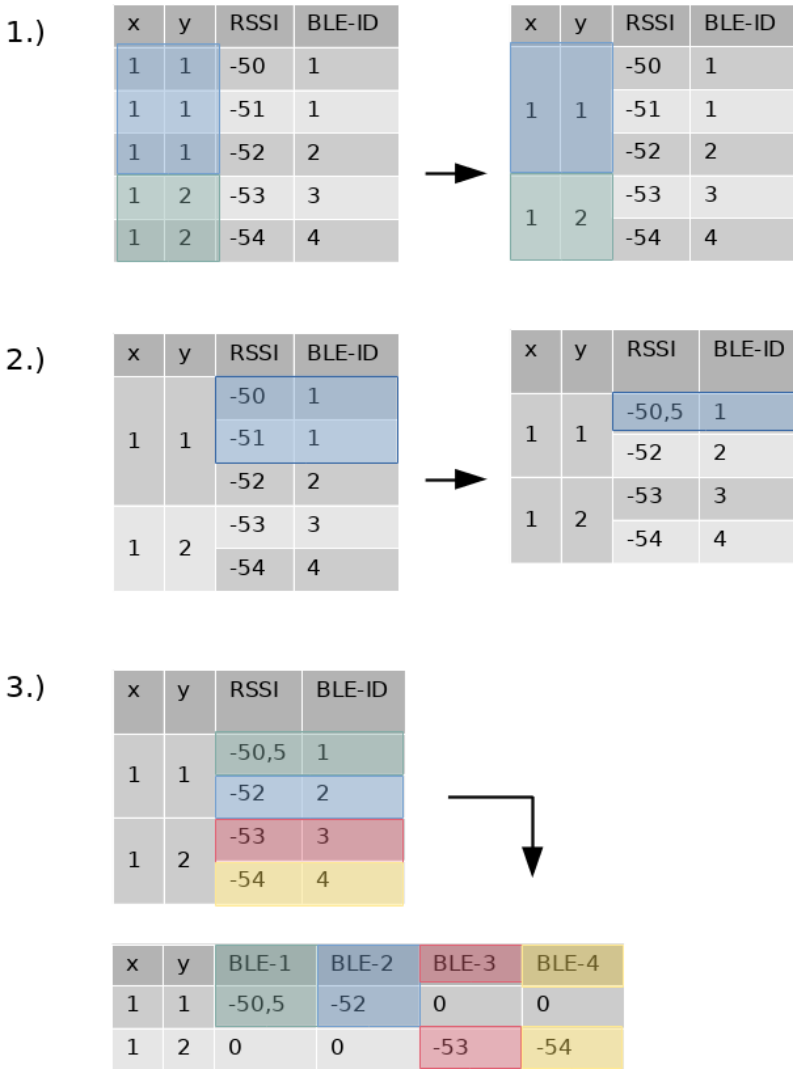

Figure 3.5: Creating the location binned dataset from the reference dataset. 1.) Group by location. 2.) Average RSSI for same BLE-ID in location groups. 3.) Flatten to one entry per position by moving BLE-IDs into columns.

2. Normalize the timestamp column by subtracting t_{min} from each timestamp t, where t_{min} is the minimum of all timestamp values. Thus after this operation the timestamp value cannot be mapped to an actual date and time, instead it specifies the difference in milliseconds to the earliest timestamp in the reference dataset (with the earliest timestamp now having the value 0).
3. Divide timestamp field by *<time-bin-size>* (a time bin size of 500 milliseconds was chosen) and round down the result to the nearest integer. The resulting number is the "time-bin-id" and indicates for each sample the time bin it belongs to.
4. Group the data by the position and the "time-bin-id". Each group is a sample (i.e. row) in the resulting time binned dataset.
5. Average the RSSI values for each beacon in a group.

This process is also depicted in Figure 3.6. An excerpt of the resulting dataset is shown in Listing 5.

The test dataset was created by randomly (uniform) selecting 20% of the samples for each location. This was done to ensure that each location occurs both in the training and test dataset.

```
Beacon 2 AVG RSSI,... ,Beacon 28 AVG RSSI,X,Y
 ...
-80.0,... ,-89.0,17.5,5.0
 00.0,... , 00.0,17.5,5.0
 00.0,... , 00.0,15.5,5.0
-67.0,... ,-87.0,15.5,5.0
...
 00.0,... ,-71.0,7.0,6.0
 00.0,... ,-80.0,7.0,6.0
 -85.0,... ,-79.0,7.0,8.0
...
```

Listing 5: Excerpt from the time binned dataset which was created using a bin size of 500 ms. The original file has 2475 samples with a single location having an average of 46.7 samples..

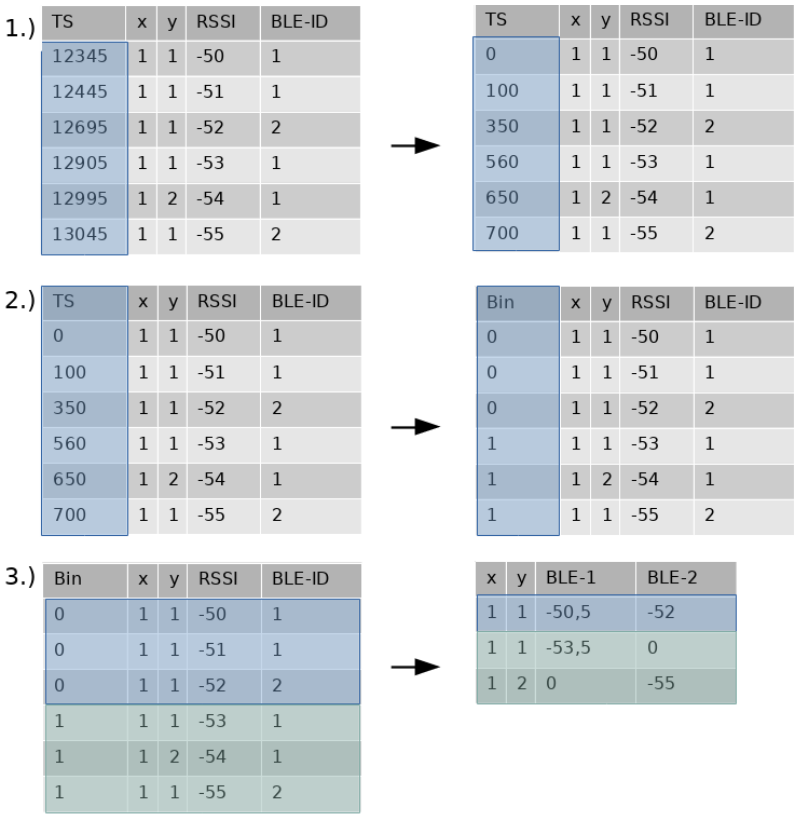

Figure 3.6: Creating the time binned dataset from the reference dataset. 1.) The timestamp column is normalized by subtracting the minimum value. 2.) Each sample is assigned a time bin by calculating $bin = \lfloor TS/bin\_size \rfloor$ with $bin\_size = 500$. 3.) All samples within a time bin are processed as shown for the location binned dataset in Figure 3.5.

3.2.4 Time Binned Dataset with Time Shifts

In order to augment the time binned dataset by boosting the number of samples, a technique hereby defined as "time shifting" was used. This technique builds on the fact that the time binned dataset is created from the reference dataset by grouping the samples by time. As described in the previous section (see list Item 2), the timestamp column is initially normalized before grouping so that $t_{min} = 0$. However, if an offset is added to all timestamp values after normalization, the resulting groups are potentially made up of a different set of samples from the reference dataset. An example for creating such a time binned dataset with an offset of 250 ms is depicted in Figure 3.7. When talking about a dataset with a time shift of n, where n is a positive integer or zero, n specifies the amount of time shifted datasets added to the original time binned dataset. Thus, a dataset with time shift 0 is equivalent to the original time binned dataset. More specifically, a dataset with time shift n consists of datasets $(||_{i=1}^{n} ds_i)||ds_{orig}$, where ds stands for dataset, $||$ represents matrix concatenation along the row (sample) axis, ds_{orig} is the original time binned dataset without an offset and ds_i is a time binned dataset created with offset o_i. The offset o_i in milliseconds is given by $o_i = i * \frac{bs}{n+1}$, where bs is the time bin size in milliseconds, which was defined as 500 ms in this work(see list Item 3 in Section 3.2.3). For example, a dataset with timeshift 4 is the concatenation of a time binned dataset with offset 0 ms (the original time binned dataset ds_{orig}), one with offset 100 ms, 200 ms, 300 ms and 400 ms.

In theory this approach multiplies the sample size of the original time binned dataset by factor n (the amount of timeshifts). In practice, however, an increasing number of time shifts provides diminishing returns, due to following factors:

- **Data duplication.** Depending on the frequency with which BLE beacons were received and thus the time density of the collected data (which varies by device, see Figures 3.3 and 3.4), a time offset will often not make a difference in the reference dataset sample to time bin mapping. I.e. when looking at the example of Figure 3.7, it is relatively easy to see that any offset values between $[0, 149]$ ms would produce the same time bins and thus duplicate data.

- **Test dataset.** Initially the test dataset was created in a similar way as in Section 3.2.3, by selecting 20% of samples for each location. However, the way time shifted datasets are created allows for overlaps of test and training information, since the same sample from the reference dataset could be included in up to n samples in the time shifted dataset. Thus it is necessary to make sure that all samples from the reference dataset, which were selected for the test dataset, are not included in any way in the training dataset. This can only occur after the initial 20% sampling per location has been done and causes the final ratio of #samples in the test dataset to #samples in the training dataset to be skewered towards a larger test dataset than intended. To retain the 80 : 20 ratio of training to test data, the percentage for test data point selection had to be adjusted manually. The lookup table for these percentages is given in Table 3.1 and the values were found by trial and error. For example, when a dataset with time shift 1 is created, 13,5% of data points are selected for the test dataset. All other data points are assigned to the training dataset. After the training dataset has been cleaned of overlaps as described above, the ratio of training to test data becomes approximately 80 : 20.

Time Shifts	Test Data Percentage
1	13,5
2	11,5
3	9,9
4	8,8
5	7,95
10	5,5
20	3,47

Table 3.1: Percentages used to generate test dataset for various time shifts.

3.2.5 Hardware specific Datasets

Two types of hardware (meaning mobile phone) specific time binned datasets were created. The general procedure for doing so is the same

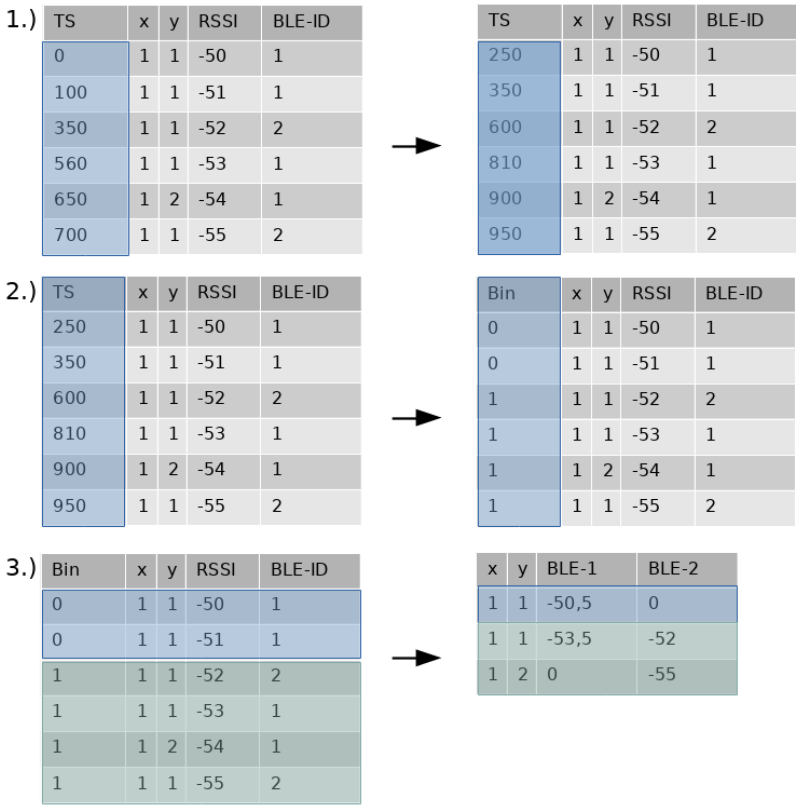

Figure 3.7: Creating a time binned dataset with a time shift of 1 from the reference dataset. This is essentially the same process as depicted for the time binned dataset in Figure 3.6. In this figure the timestamp normalization has already been done and is not depicted. Step 1.) is the step that illustrates the time shift by adding a time offset of 250 ms to the timestamp column. Steps 2.) and 3.) are the same as in Figure 3.6, except that the outcome differs because of the offset.

as described in Section 3.2.3, except that the samples from the reference dataset were filtered by hardware for creating time binned datasets.

1. **Specific hardware only.** With this approach only samples from the same hardware device were used for creating the test and training dataset. This results in a test and training dataset for each of the mobile phones listed in Section 3.1.
2. **Test on specific hardware.** With this approach, the test dataset was created using all the data of one specific device, while the data of the other two devices was used for the training dataset. Again three pairs of test and training datasets were created.

4 Indoor Positioning System Implementation

In this chapter the actual implementation of the Indoor Positioning System (IPS) is discussed.

The input is given by the datasets introduced in Chapter 3: location binned, time binned, time binned with time shifts and hardware specific. A sample of such a dataset always consists of 24 RSSI values (features) as well as x and y coordinates (targets).

Datasets obtained from parsing, aggregating and scaling the raw input from data gathering (see Section 3.1), are used to train a FFNN.

The goal of this IPS is to find a robust mapping between the signal strength of nearby BLE beacons to a specific location, which consists of an x and y coordinate in a 2D environment..

See Figure 4.1 for a general overview of the whole implementation and its workflow.

4.1 Missing data and Masking

The shorter the interval for data gathering, the less BLE signals will be received. For example, the time binned dataset in Listing 5 has 55% missing entries. If a sample has no value for a feature, because no BLE signal was received during the selected timeframe, it is treated as missing data. Fields of missing data are set to 0 before data scaling. However, with this approach features that have the lowest RSSI value are treated equivalent to

features representing missing data (for details see the following data scaling Section 4.2.1).

To incorporate this information into the IPS the following three approaches have been investigated:

1. **Binarized Dataset:** Set input data features to 0 if it is missing data and 1 otherwise. This serves as a baseline approach and is useful when comparing to the other two approaches, which should, logically, both perform better.
2. **Input Layer Masking:** The **Binarized Dataset** described above is treated as a missing data matrix X_m. In the first layer of the FFNN (and thus after any data scaling has been done) this matrix is used to set all missing data fields of the input matrix X to 0 by $X' = X \circ X_m$ where $\circ$ is the Hadamard product.
3. **Append Missing Data Mask:** The missing data matrix X_m is simply concatenated to the original input X, thus doubling the amount of features. In contrast to X, X_m does not undergo data scaling as described in Section 4.2.1.

4.2 Implementation

Prototyping was done in MATLAB using the Neural Network Toolbox. The actual implementation was then done in Python using the Tensorflow [15] machine learning library. For data exploration and plotting evaluation results Jupyter Notebooks [18][12] was used.

4.2.1 Data Module

This module handles data parsing, aggregation and scaling.

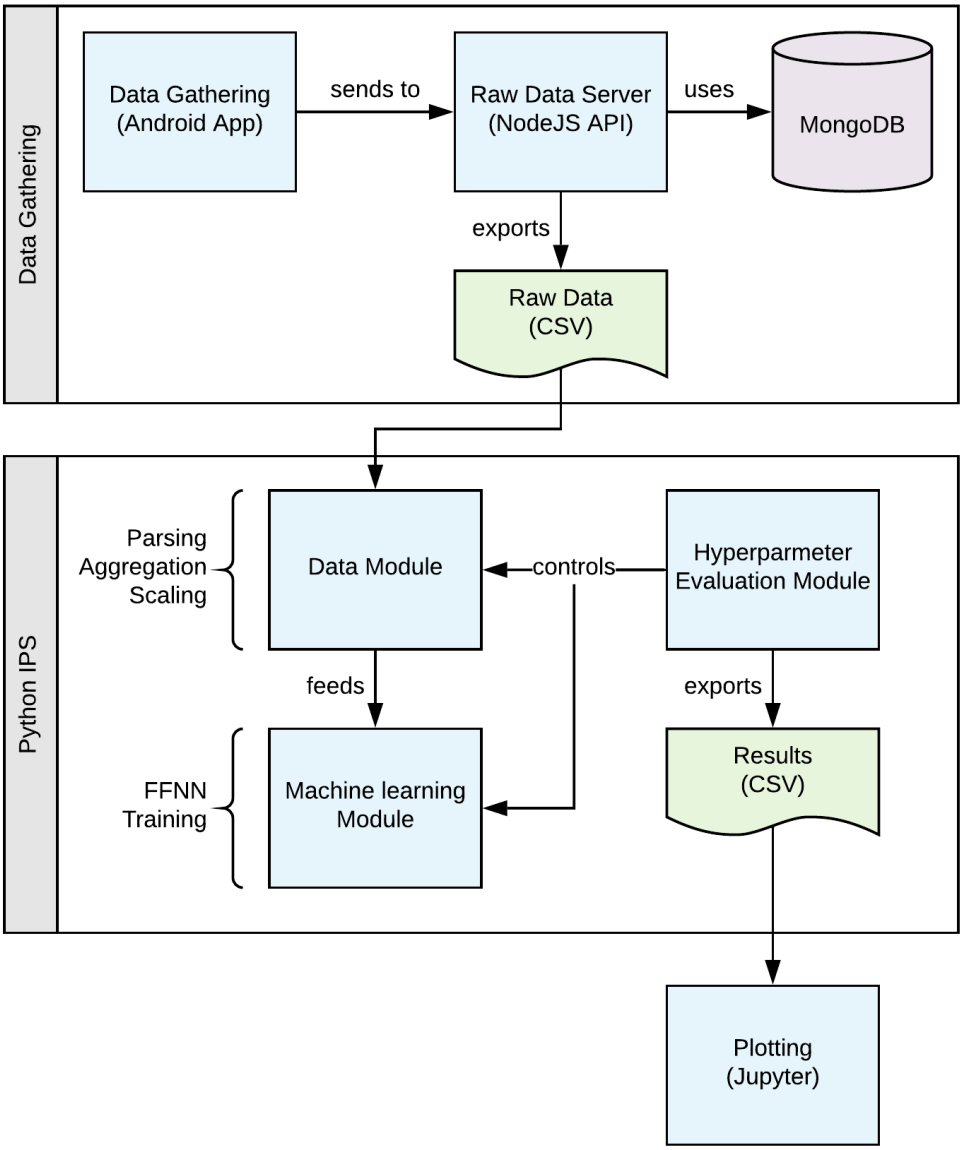

Figure 4.1: Full workflow of all systems used in this work. Components grouped in *Data Gathering* have been discussed in Chapter 3. Components grouped in *Python IPS* are discussed in this chapter. More specifically, *Data Module* is discussed in Section 4.2.1, *Machine learning Module* in Section 4.2.2 and *Hyperparameter Evaluation Module* in Section 4.2.3.

Data Parsing

This submodule reads in all raw datasets obtained from data gathering (see example in Listing 1) and BLE information as depicted in Listing 2.

Data Aggregation

The data aggregation submodule handles the transformation of the parsed data into the datasets described in the Aggregation Section 3.2:

- Location Binned Dataset (Section 3.2.2)
- Time Binned Dataset (Section 3.2.3)
- Time Binned Dataset with Time Shifts (Section 3.2.4)
- Hardware specific Datasets (Section 3.2.5)

Data scaling

This submodule takes the aggregated data as an input and provides three ways of data scaling. It is also responsible for storing the missing data matrix along with the scaled data.

RSSI values were first thresholded to the range $[-90, -10]$ and then mapped to positive values by $x_{mapped} = x_{orig} + 90$. If a RSSI value was not available for a given feature for a sample, the value is kept at 0.

While the previous operations were always performed, the following data scaling operations were treated as an optimizable hyperparameter. These options were being investigated separately for features and targets:

1. **Standardization** Values are mapped to have 0 mean and 1 standard deviation by

$$x_{scaled} = \frac{x - \mu}{\sigma} \tag{4.1}$$

2. **Normalization** Values are mapped into the range $[0;1]$ by

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.2}$$

3. **Binary** This is the implementation of the **Binarized Dataset** described in Section 4.1 Item 1. It only applies to features. Values are set to either 0 or 1 by

$$x_{scaled} = \begin{cases} 0 & \text{if } x = 0 \\ 1 & otherwise \end{cases}$$

4.2.2 Machine learning module

The FFNN architecture used is based on the definition given in equation (2.3). The following network properties are parameterized and are evaluated in Chapter 5:

1. **Input Masking** is either turned on or off. If it is active, the input to the FFNN will be set to 0 for missing data fields, as described in Section 4.1 Item 2.
2. **Layers**. The number of hidden layers. For each layer the amount of hidden nodes and the activation function are specified individually.
3. The **learning rate** for the chosen optimization algorithm.
4. **Weight Decay/Regularization**. Specifies whether or not regularization is done. If regularization is done, the l2 loss for all weights is computed based on the Tensorflow function *tf.nn.l2_loss*[1] with scaling term β being another parameter:

$$l2loss = \frac{\beta}{2} \sum_{t=1}^{T} \left\| W_t^{\circ 2} \right\|_1 \tag{4.3}$$

with T being the total number of hidden layers plus the output layer, W_t the matrix containing all weight vectors connecting the nodes from layer $t-1$ to layer t (the input layer is layer 0), $\circ$ indicating Hadamard exponentiation and $\|W\|_1$ is the matrix 1-norm.

[1] https://www.tensorflow.org/api_docs/python/tf/nn/l2_loss

Following network properties were fixed:

- The Mean Squared Error (MSE) was used as the **loss function**:

$$MSE = \frac{1}{U}\sum_{u}^{U}(\boldsymbol{y_u} - \hat{\boldsymbol{y}}_u)^2 \tag{4.4}$$

  with U being the total number of targets. Furthermore, $\boldsymbol{y_u}$ indicates the u^{th} row of the test dataset target matrix $\boldsymbol{Y} \in \mathbb{R}^{Ux2}$ which has two columns representing a 2D position in the form of x and y coordinates. Analogously, $\hat{\boldsymbol{y}}_u$ represents the corresponding row in the prediction target matrix $\hat{\boldsymbol{Y}} \in \mathbb{R}^{Ux2}$.
- The **bias** nodes were initialized with 1
- **Weights** were initialized by sampling from a normal distribution with $\mu = 0$ and $\sigma = 0.1$.
- Following activation functions were used: ReLU, ELU and TanH (see Section 2.2.1).
- Adam was used as the **Optimization Algorithm**.

Training

Training is done by reading in a train and test dataset and then performing k-Fold cross validation on the train dataset. Actual train and test dataset generation (handled by the data aggregation submodule 4.2.1) is done prior to training and thus which dataset is used is indicated by which dataset files are specified for training. The following properties concerning training are available:

- Which **dataset** files are used.
- The k parameter for **kFold** cross validation.
- **Training epochs**
- **Maximum validation epochs** for early stopping.

4.2.3 Hyperparameter Evaluation Module

The Hyperparameter Evaluation Module generates a set of concrete configurations and then applies and executes them automatically. This is done

```
base_config = {...
  "cfg.input.data-file": ['tb-opo-500ms-shift5', 'tb-p20-500ms-shift5'],
  "cfg.data.append-mask-to-input": True,
  "cfg.nn.train.max-training-epochs": [15000, 20000],
...
}
```

Listing 6: Excerpt of an example base configurations. The values of properties "cfg.input.data-file" and "cfg.nn.train.max-training-epochs" are lists and thus will be flatted out when transformed into concrete configurations.

by first specifying a base configuration, which includes lists of possible values for any given configuration parameter. All parameters discussed in Sections data scaling 4.2.1 and machine learning module 4.2.2 are specifiable, as well as some additional implementation specific parameters. The base configuration is then flattened into concrete configurations, for which training is performed. The results of the trained network for each concrete configuration are collected and written to a CSV file. An example of a base configuration and the resulting concrete configurations is given in Listings 6 and 7.

```
generated_configs = [{...
  "cfg.input.data-file": 'tb-opo-500ms-shift5',
  "cfg.data.append-mask-to-input": True,
  "cfg.nn.train.max-training-epochs": 15000,
...},{...
  "cfg.input.data-file": 'tb-p20-500ms-shift5',
  "cfg.data.append-mask-to-input": True,
  "cfg.nn.train.max-training-epochs": 15000,
...},{...
  "cfg.input.data-file": 'tb-opo-500ms-shift5',
  "cfg.data.append-mask-to-input": True,
  "cfg.nn.train.max-training-epochs": 20000,
...},{...
  "cfg.input.data-file": 'tb-p20-500ms-shift5',
  "cfg.data.append-mask-to-input": True,
  "cfg.nn.train.max-training-epochs": 20000,
...}]
```

Listing 7: The four concrete configurations which result from the example base configuration in Listing 6.

5 Evaluation

The full evaluation of the various approaches discussed in Sections 3.2 and 4 was executed in a sequence of individual evaluation steps, where each focused on a subset of the evaluable IPS parameters. This reduces computational complexity when compared to a more brute force exhaustive hyperparameter search approach. Performing a smaller evaluation in these cases serves as additional support in selecting the logically favored parameters.

Where not specified otherwise, the following settings were used for all evaluations:

- $k = 10$ for **kFold** cross validation
- The **test error** was computed using the average Euclidean distance (in meters) between the target and the prediction:

$$E_{test} = \frac{1}{N} \sum^{N} \sqrt{(y_x - \hat{y}_x)^2 + (y_y - \hat{y}_y)^2} \tag{5.1}$$

  where subscript x and y denote the x and y coordinate of the target $\boldsymbol{y}$ and prediction $\hat{\boldsymbol{y}}$ and N is the total amount of samples in the test dataset.
- **Maximum training epochs** were set to 20000
- **Maximum validation epochs** were set to 2000, i.e. after 2000 epochs of increasing validation error, training is aborted.

While the best performing values of following parameters were found through the evaluations discussed in the next sections, their initial values were:

- **Dataset**. The time binned dataset without time shifts was used. Bin size was 500ms. See Listing 5 for an excerpt.

- **Layers**. Two hidden layers with 40 and 20 hidden nodes and hyperbolic tangent (TanH, see Section 2.2.1) activation function were used. The output layer had two nodes and used linear output function.
- **Learning rate** was set to 0.05.
- No **regularization** was used.

5.1 Hyperparameter evaluation for general use case

The evaluations done in this section led to the best performing model for datasets using all hardware models.

5.1.1 Missing data incorporation

Different ways of incorporating the information of missing data into the learning process were evaluated. Figure 5.1 shows the results of comparing the following techniques:

- The **Original Dataset**.
- Using a **Binarized Dataset** as input, as described in Section 4.1 Item 1.
- **FFNN Masking** used masking on the first layer of the FFNN, as described in Section 4.1 Item 2.
- **Missing Mask Appended** uses the original dataset concatenated with the missing data mask, as described in Section 4.1 Item 3.

As expected, the **Binarized Dataset** performed worst. While the performance difference between the other three options did not vary significantly, the **Missing Mask Appended** approach performed slightly better than both **FFNN Masking** and the **Original Dataset** and was thus chosen as a fixed parameter for further evaluation steps.

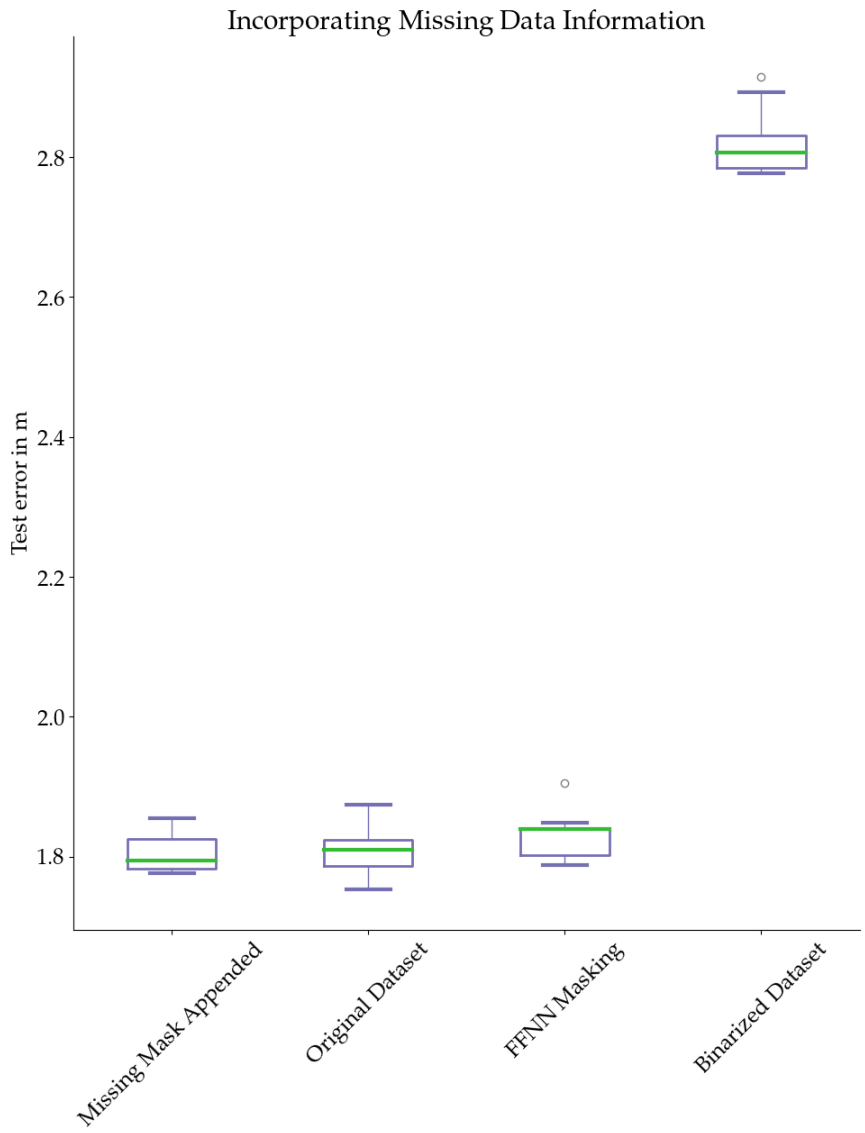

Figure 5.1: Performance of various approaches on how to incorporate missing data information. Depicted are the test errors from the 10 folds done during training. The boxes are spanned from the first (Q_1) to the third quartile (Q_3), with the second quartile (Q_2, which is also the median) being represented as a green line. The whiskers indicate the lowest and highest datum that is still within the first and third quartile plus 1.5 times the interquartile range (IQR). More specifically, the lower whisker extends to the lowest datum that is $\geq Q_1 - 1.5 * IQR$ and the upper whisker to the highest datum that is $\leq Q_3 + 1.5 * IQR$. Data points that do not lie within $[Q_1 - 1.5 * IQR; Q_3 + 1.5 * IQR]$ are considered outliers and depicted as "flier points" above or below the whiskers. Furthermore, the x-axis is sorted by the median.

5.1.2 Data scaling

The evaluation of the following data scaling options for both features and targets are depicted in Figure 5.2:

- **Standardization** as described in Section 4.2.1 Item 1. Labelled in the Figure 5.2 with "T standardization" for target standardization and "F standardization" for feature standardization.
- **Normalization** as described in Section 4.2.1 Item 2. Labelled in the Figure 5.2 with "T normalization" for target normalization and "F normalization" for feature normalization.

Differences in performance were not significant, but feature normalization and target standardization were chosen as fixed parameters for future evaluations as they performed best.

5.1.3 Time shifted datasets and regularization

The following parameters were evaluated, with the results being depicted in Figure 5.3:

- **Time Shift**. While all examined datasets were time binned, the amount of time shifts(see Section 3.2.4) varies as follows: $0, 1, 3, 5, 10, 20$ (thus time shift 0 is equivalent to the original time binned dataset as seen in Listing 5). They are depicted in the x-axis.
- L2 Regularization as described in Section 4.2.2 Item 4. The selected value for the $\boldsymbol{\beta}$ parameter is given in the title of each subplot.

It can be seen that a too large value for β severely hurts the performance, while a small β performs best. Furthermore, a single time shift outperforms the other datasets. Thus regularization with $\beta = 0.00001$ and the dataset with 1 time shifts were chosen as fixed values for further evaluations.

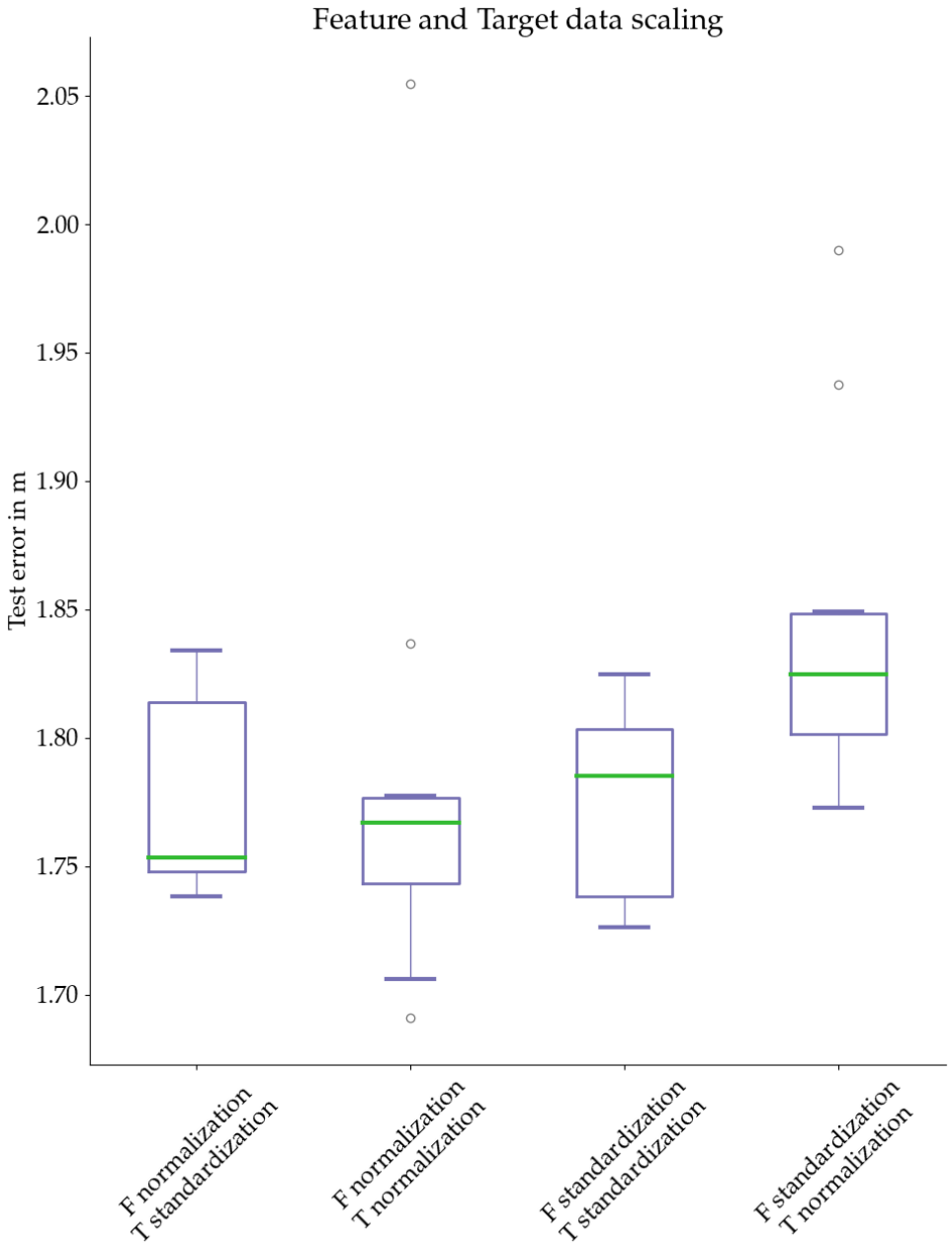

Figure 5.2: Performance of various data scaling approaches. An "F" prefix indicates feature data and "T" indicates target data. "Normalization" means normalizing the data to target range $[0, 1]$. "Standardization" standardizes the data to mean 0 and standard deviation 1. See Figure 5.1 for a general definition of the methods used to generate this plot.

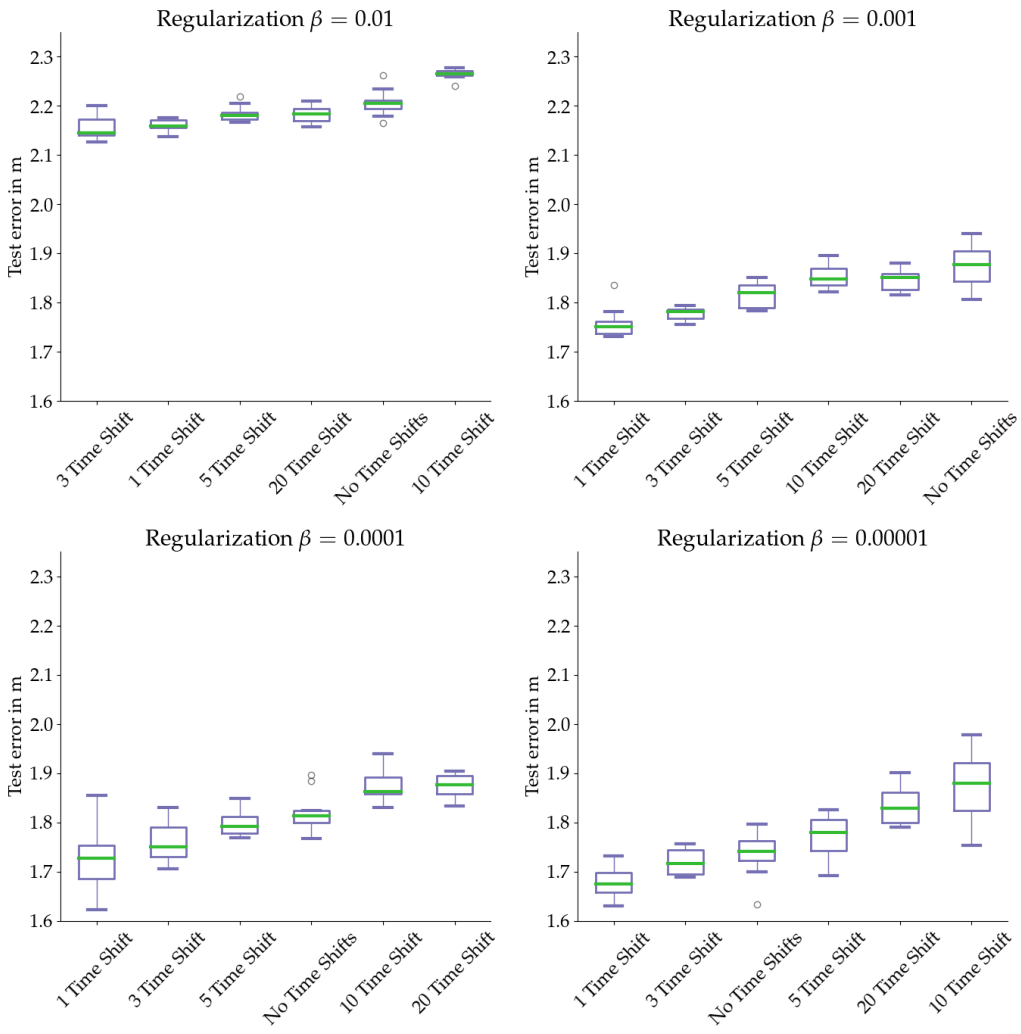

Figure 5.3: Performance of regularization and varying dataset time shifts. Each subplot depicts the chosen value for the regularization parameter β in the title. The time shift values of the used datasets are given in the x-axis. See Figure 5.1 for a general definition of the methods used to generate this plot.

5.1.4 Hidden layers and learning rate

Finally, the actual network architecture and the learning rate of the optimizer algorithm were evaluated, with results depicted in Figure 5.4:

- **Hidden Layers** The following hidden layer configurations were examined:
  - Two hidden layers with 20 and 10 nodes.
  - Two hidden layers with 40 and 20 nodes.
  - Three hidden layers with 80, 40 and 20 nodes.
- The **Learning rate** used for the ADAM optimizer. Based on the **activation function** (see Section 2.2.1), the following learning rates were evaluated:
  - **TanH**: Learning rate either 0.01, 0.05 or 0.1
  - **ELU** and **ReLU**: Learning rate either 0.001 or 0.0001.

Surprisingly, the **TanH** activation function seemed to outperform both **ReLU** and **ELU**. Learning rate differences had a minor impact with the exception of more complex networks that used **TanH**, for which a very small learning rate led to worse results. Network complexity (i.e. count of hidden layers and their nodes) yielded similar results as the learning rate comparison, showing little variance when using the same activation function and learning rate. The one exception proved to be the best performing variant: three hidden layers (with 80, 40 and 20 nodes), learning rate 0.01 and **TanH** activation function. This architecture achieved a 5 cm improvement when compared to the next best architecture and was thus chosen as the final parameter. The median and standard deviation of the test error for this architecture were 1.624 m and 0.036 m, respectively.

5.2 Hardware specific evaluation

To investigate the differences between mobile phones, further evaluation was done on datasets that were filtered by hardware type. The IPS parameters used in this section are the ones that were determined in the previous sections, with the exception of the input datasets.

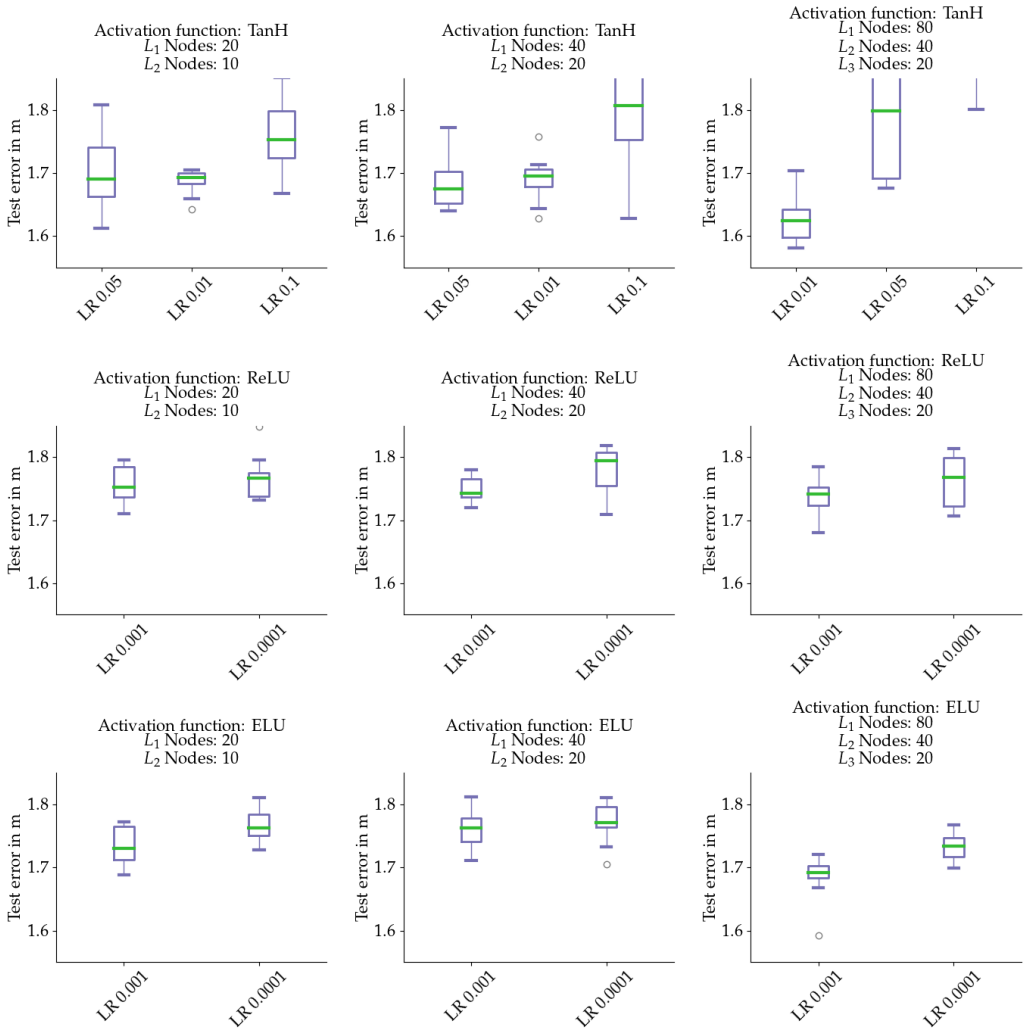

Figure 5.4: Performance of hidden layer architectures and learning rates. Each subplot depicts a fixed architecture consisting of an activation function (given in the first line of the subplot title) as well as the number of hidden layers and their node count (given in the subsequent lines of the subplot title). Results are given for each architecture based on a chosen learning rate (LR, depicted in the x-axis). See Figure 5.1 for a general definition of the methods used to generate this plot.

5.2.1 Testing on dataset of exclusively one device

For this evaluation the network was trained on the data samples originating from two devices, while the test dataset was comprised of samples from the third device (see Section 3.2.5 Item 1). The results of the trained networks on the three possible permutations of hardware datasets is depicted in Figure 5.5.

The increased error when testing on the OPO dataset is significant and very likely related to the fact that the OPO gathered far less data then the other devices (see Figures 3.3 and 3.4). The median/std for the test errors for the "P20 test dataset" and "S9 test dataset" evaluations were 1.92 m / 0.2 m and 2.026 m / 0.058 m, respectively.

5.2.2 Training and testing on data of same device

For this evaluation, both training and test dataset were created with data originating solely from one device(see Section 3.2.5 Item 2). The results are depicted in Figure 5.6.

The "OPO" dataset is the worst performing one by far, while the test error for both the "P20" (median 1.66 m std 0.064 m) and "S9" (median 1.706 m std 0.06 m) dataset is slightly worse than the best non-hardware-specific solution discussed in Section 5.1.4.

5.3 kNN

For comparison, selected datasets were trained and tested using a k-nearest neighbors algorithm (see Section 2.3). The results (see Figure 5.7) are significantly worse than all other evaluated FFNN-based approaches.

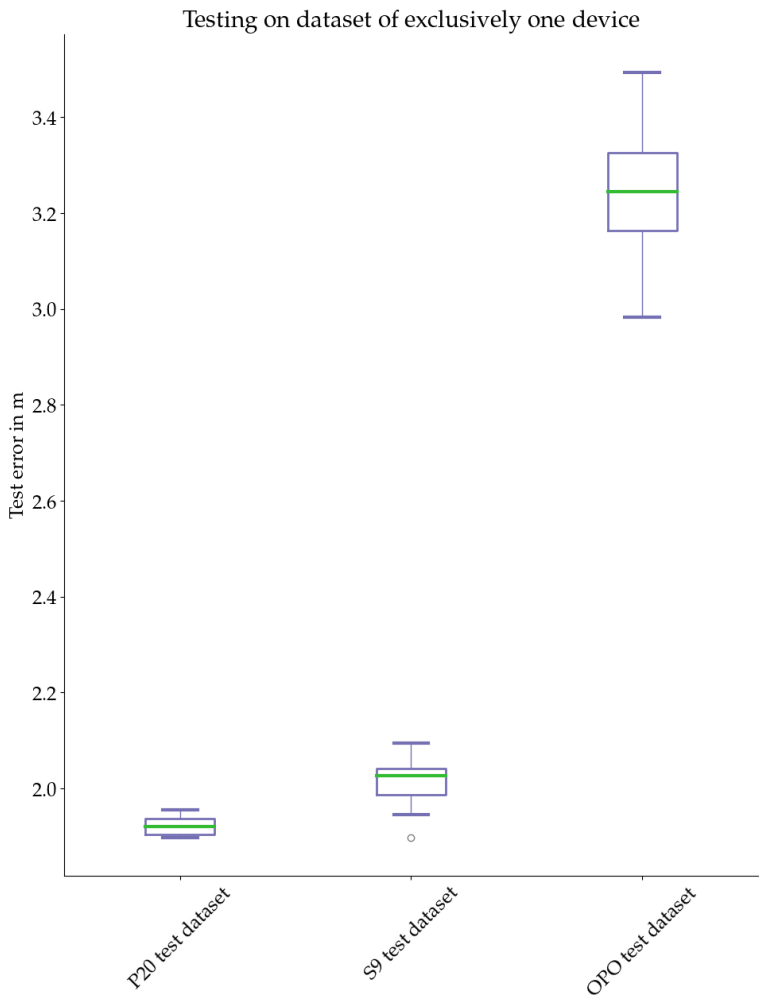

Figure 5.5: The x-axis label indicates which dataset was used for testing. E.g. "P20 test dataset" means that network training was done using data of the S9 and OPO while testing was done on data originating from the P20. See Figure 5.1 for a general definition of the methods used to generate this plot.

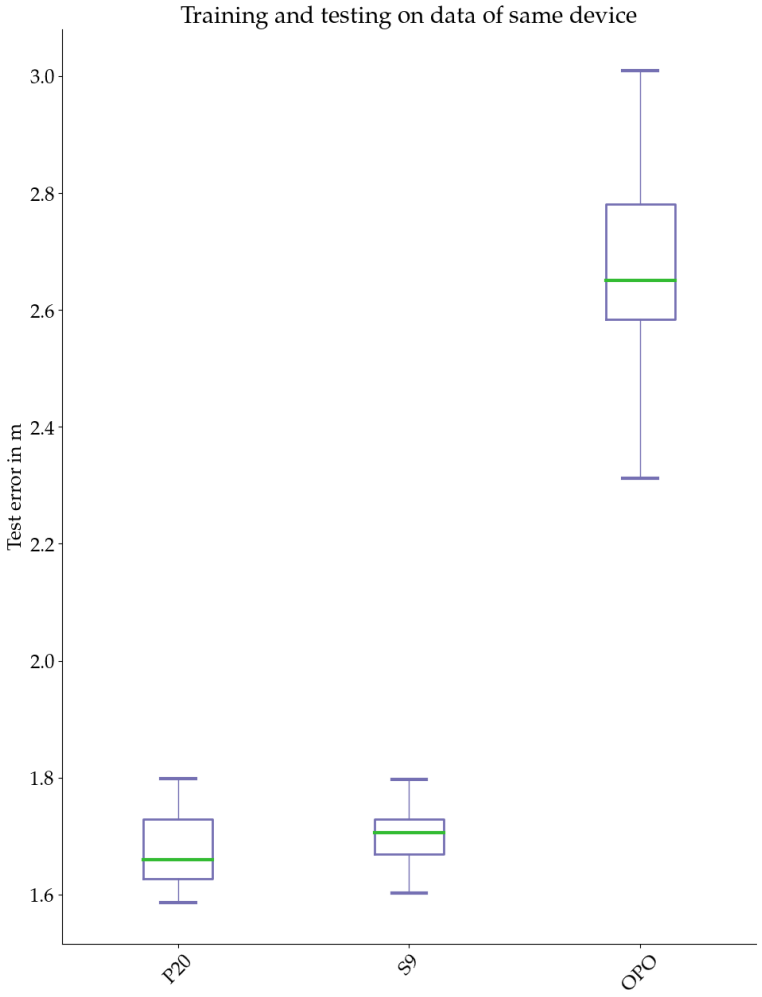

Figure 5.6: The x-axis label indicates which data was used for the training and test set, i.e. "P20" means that both training and test data originated from the P20 device. See Figure 5.1 for a general definition of the methods used to generate this plot.

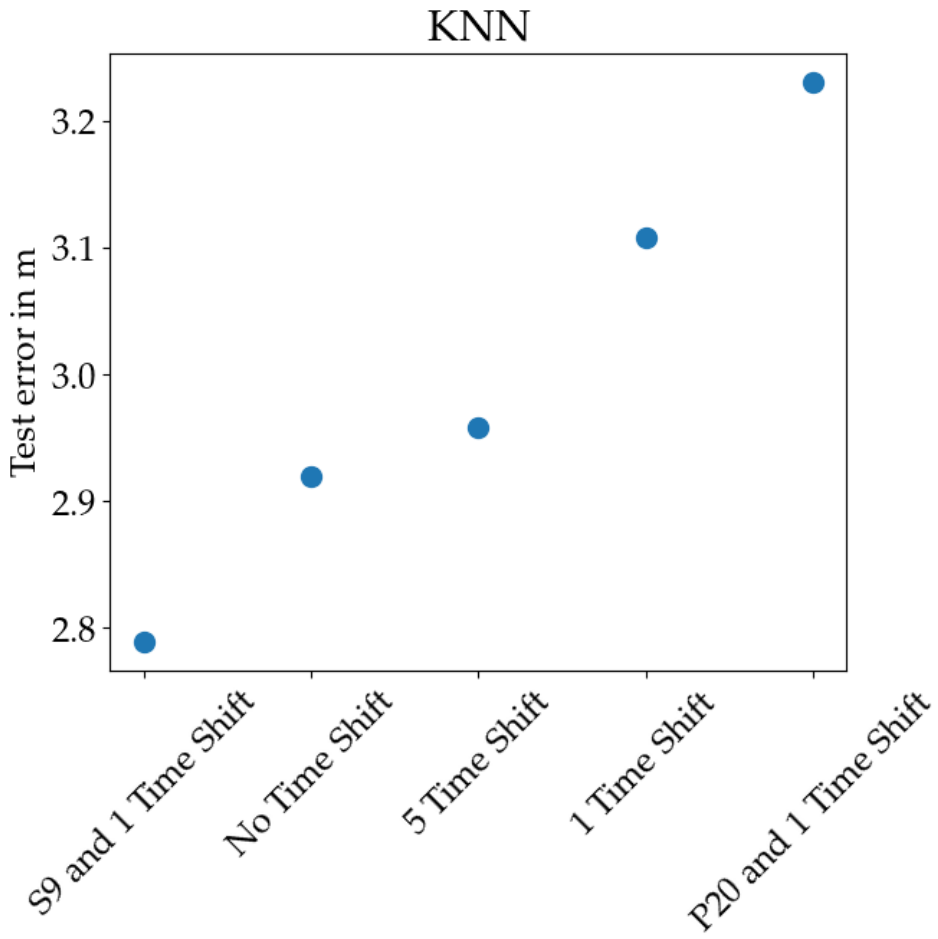

Figure 5.7: The x-axis label indicates which dataset was used. "No/1/5 Time Shift" are the same datasets as used in Section 5.1.3. "S9/P20 and 1 Time Shift" are the same datasets as used in Section 5.2.2

6 Conclusion

In this work an IPS based on BLE beacons was implemented. This was done by setting up a reference environment, outfitting it with beacons and gathering signal data. This data was refined and aggregated and used to build a fingerprint database. The IPS used this data to train a FFNN and perform hyperparameter evaluation.

It has been shown that increasing the number of training samples by utilizing time shifts in dataset creation, leads to increased performance up to certain degree of time shifts (Section 5.1.3). The methods demonstrated for incorporating missing data information into the FFNN model did not yield significant improvement (Section 5.1.1). The best found FFNN model for the general use case (i.e. using data of all hardware devices) achieved a median test error of 1.624 m (Section 5.1.4).

When looking at the results concerning datasets grouped by hardware, it is important to acknowledge the differences in the data gathering capabilities of the devices. While the S9 and and P20 devices displayed a comparable amount of BLE signals received for each location, the OPO generally managed to sample only a fifth of the amount (see figures 3.3 and 3.4). Thus, if the IPS is used on datasets separated strictly by device (Section 5.2.2), both the S9 (median test error 1.706 m) and the P20 (median test error 1.66 m) come close to the general use case model, while the performance of the OPO is far worse (median test error 2.65 m). When using the data of two devices for training and then testing the model on data of the third device (Section 5.2.1), the median test error for testing on the S9 (2.026 m) and P20 (1.92 m) datasets was still somewhat competitive, while testing on the OPO dataset yielded a significantly worse performance (median test error 3.245 m). These two hardware-specific use cases might indicate that using datasets consisting of data from different devices (i.e. the general use

case mentioned above) for the IPS could be advantageous for both well and badly performing (in terms of data gathering) devices, when compared to hardware specific data separation. However, the low number of tested devices has to be taken into account.

A naive kNN implementation (Section 5.3) performs worse than the IPS examined in this work for selected datasets: 3.108 m error on the dataset used for the IPS general use case. 2.789 m and 3.231 m for datasets containing only data from the S9 and P20 devices, respectively.

6.1 Future Work

In this work, gathered BLE signals were grouped together into time bins using various window sizes, in order to assign a 2D position to this set of signals. An alternative approach could be treating incoming signals as a time series and modeling the task of positioning as a time series prediction using a long short-term memory (LSTM) model.

Real world applications are also likely to see client devices moving during the process of position estimation. Thus, data gathering and aggregation could be extended to allow for capturing and labeling predefined walks across a set of positions.

A total of three smartphone devices were used in this work, which is far from being an exhaustive comparison to the real world. Extending the fingerprinting database with further devices is very likely to yield better generalization and more insights in hardware differences. Furthermore, the process of gathering data for the fingerprinting database is currently time-consuming, tedious and its consistency is highly dependent on the person performing the task. Therefore, automated methods for performing this task would be a valuable asset to the IPS training pipeline. Solutions using free moving agents (e.g. drones or other autonomously moving robots) in conjunction with some form of ground truth verification (e.g. an already existing IPS as done in [9] or using QR-code stickers on designated positions) are thinkable.

This work focused on using solely the RSS of received BLE signals as input to the IPS. Presumably, incorporating other information available to devices will improve the general performance. A non-exhaustive list of candidates are:

- Accelerometer and gyroscope information to determine device orientation and if the user is currently walking or standing still.
- Magnetometer to determine cardinal direction
- Ambient light sensors could help with detecting room changes

Bibliography

[1] S. Alletto et al. "An Indoor Location-Aware System for an IoT-Based Smart Museum." In: *IEEE Internet of Things Journal* 3.2 (Apr. 2016), pp. 244–253. ISSN: 2327-4662. DOI: 10.1109/JIOT.2015.2506258 (cit. on p. 10).

[2] M. Altini et al. "Bluetooth indoor localization with multiple neural networks." In: *Wireless Pervasive Computing (ISWPC), 2010 5th IEEE International Symposium on*. May 2010, pp. 295–300. DOI: 10.1109/ISWPC.2010.5483748 (cit. on p. 11).

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738 (cit. on pp. 6, 9).

[4] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)." In: *CoRR* abs/1511.07289 (2015). arXiv: 1511.07289. URL: http://arxiv.org/abs/1511.07289 (cit. on p. 7).

[5] H. Dai et al. "Indoor Positioning Algorithm Based on Parallel Multilayer Neural Network." In: *2016 International Conference on Information System and Artificial Intelligence (ISAI)*. June 2016, pp. 356–360. DOI: 10.1109/ISAI.2016.0082 (cit. on p. 12).

[6] P. Davidson and R. Piché. "A Survey of Selected Indoor Positioning Methods for Smartphones." In: *IEEE Communications Surveys Tutorials* 19.2 (Secondquarter 2017), pp. 1347–1370. ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2637663 (cit. on p. 3).

[7] J. J. M. Diaz et al. "Bluepass: An indoor Bluetooth-based localization system for mobile applications." In: *The IEEE symposium on Computers and Communications*. June 2010, pp. 778–783. DOI: 10.1109/ISCC.2010.5546506 (cit. on p. 10).

[8] Qian Dong and W. Dargie. "Evaluation of the reliability of RSSI for indoor localization." In: *2012 International Conference on Wireless Communications in Underground and Confined Areas*. Aug. 2012, pp. 1–6. DOI: 10.1109/ICWCUCA.2012.6402492 (cit. on p. 3).

[9] R. Faragher and R. Harle. "Location Fingerprinting With Bluetooth Low Energy Beacons." In: *IEEE Journal on Selected Areas in Communications* 33.11 (Nov. 2015), pp. 2418–2428. ISSN: 0733-8716. DOI: 10.1109/JSAC.2015.2430281 (cit. on pp. 5, 10, 54).

[10] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks." In: *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*. Apr. 2011 (cit. on pp. 6, 7).

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pp. 5, 6, 8, 9).

[12] Thomas Kluyver et al. "Jupyter Notebooks – a publishing format for reproducible computational workflows." In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87–90 (cit. on p. 34).

[13] T. Kröger et al. "Method of pedestrian dead reckoning using speed recognition." In: *Ubiquitous Positioning Indoor Navigation and Location Based Service (UPINLBS), 2010*. Oct. 2010, pp. 1–8. DOI: 10.1109/UPINLBS.2010.5653994 (cit. on p. 12).

[14] Binghao Li et al. "Indoor positioning techniques based on wireless LAN." In: *LAN, FIRST IEEE INTERNATIONAL CONFERENCE ON WIRELESS BROADBAND AND ULTRA WIDEBAND COMMUNICATIONS*, pp. 13–16 (cit. on p. 3).

[15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 34).

[16] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 9).

[17] L. Peng et al. "3D Indoor localization based on spectral clustering and weighted backpropagation neural networks." In: *2017 IEEE/CIC International Conference on Communications in China (ICCC)*. Oct. 2017, pp. 1–6. DOI: 10.1109/ICCChina.2017.8330353 (cit. on p. 13).

[18] Fernando Pérez and Brian E. Granger. "IPython: a System for Interactive Scientific Computing." In: *Computing in Science and Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: https://ipython.org (cit. on p. 34).

[19] Fazli Subhan et al. "Indoor positioning in Bluetooth networks using fingerprinting and lateration approach." In: *2011 International Conference on Information Science and Applications* (2011), pp. 1–9 (cit. on p. 12).

[20] Ambili Thottam Parameswaran, Mohammad Iftekhar Husain, and Shambhu Upadhyaya. "Is RSSI a reliable parameter in sensor localization algorithms: an experimental study." In: (Jan. 2009) (cit. on p. 3).

[21] Y. Xu and Y. Sun. "Neural Network-Based Accuracy Enhancement Method for WLAN Indoor Positioning." In: *2012 IEEE Vehicular Technology Conference (VTC Fall)*. Sept. 2012, pp. 1–5. DOI: 10.1109/VTCFall.2012.6399107 (cit. on p. 11).