UNIVERSITY OF BERN

BACHELOR THESIS

Cloud-Based Indoor Positioning – ESP32 Client

Handed in by

Stefan SERENA

Supervisor

PROFESSOR DR. TORSTEN BRAUN

Communication and Distributed Systems
Institute of Computer Science

September 16, 2018

UNIVERSITY OF BERN

Faculty of Science
Institute of Computer Science

Bachelor of Science in Computer Science

Cloud-Based Indoor Positioning – ESP32 Client

by Stefan SERENA

Abstract

Due to the growing ubiquity of context aware applications, indoor positioning has become an important research topic. With potential localization clients becoming increasingly widespread, running the required calculations locally is becoming ever more difficult to maintain. This thesis demonstrates an implementation and evaluation for a cloud-based solution. It presents a simple and efficient indoor positioning server, as well as a minimal client capable of uploading ranging- as well as IMU (Inertial Measurement Unit) data. In addition, various approaches to improve ranging accuracy are evaluated: WiFi range-based approaches are often used and convert the measured radio signal strength into range values, which indicate the distance between the target mobile device and a radio transceiver. This process is called ranging. However, non-static environmental conditions in indoor scenarios lead to unpredictable fluctuation in WiFi signals. These fluctuations introduce errors in the localization process. Regression techniques are often used to relate measured radio signal parameters (e.g. Received Signal Strength) into range values. This work suggests that a dynamic selection of regression models and training locations yields a substantially increased accuracy, compared to a predefined, static alternative. Moreover, the sampling rate required for accurate real-time measurements is proportional to the speed of the target. Through combination of one ESP32 and two ESP8266 microcontrollers, 2.5 Hz were achieved. However, due to variations of signal measurements between WiFi controllers, this concept scales only in specific circumstances.

Acknowledgements

I would like to thank Jose Carrera and Zhongliang Zhao for supervising this thesis and their support on various occasions during my work on this thesis.

Contents

List of Figures

List of Tables

List of Abbreviations

GPS	**G**lobal **P**ositioning **S**ystem
AN	**A**nchor **N**ode
MN	**M**obile **N**ode
RSSI	**R**eceived **S**ignal **S**trength **I**ndicator
LOS	**L**ine **o**f **S**ight
NLOS	**N**on **L**ine **o**f **S**ight
PDR	**P**edestrian **D**ead **R**eckoning
IMU	**I**nertial **M**easurement **U**nit
TL	**T**raining **L**ocation
TP	**T**esting **P**oint
CDF	**C**umulative **D**istribution **F**unction
TBTT	**T**arget **B**eacon **T**ransmission **T**ime

Chapter 1

Introduction

1.1 Cloud-based Indoor Positioning

The rapid growth and ubiquity of context aware applications made indoor localization an important research topic. While the Global Positioning System (GPS) is an attractive technology to perform positioning for the outdoor environment[1], it is not a suitable indoor solution due to the limitations of signal propagation through walls. Hence, a multitude of alternative technologies such as magnetic positioning[2], ultrasonic signal processing[3] or WiFi[4] have been proposed, each with their own set of advantages and problems.

This thesis is part of a joint collaboration with contributions in the context of cloud-based indoor positioning solutions. More specifically, we present and evaluate a centralized server for the computationally involving aspects of indoor localization, and introduce various clients, which gather data and optionally display the results from the server.

Our positioning solution features a combination of WiFi ranging as well as pedestrian dead reckoning (PDR). One advantage of this approach is its accessibility, largely due to the widespread availability of both types of sensor, most prominently in smart phones. However, either approach is not without disadvantages. While PDR sensor readings from an accelerometer, gyroscope and magnetometer might feature small errors for short intervals, these individual errors are aggregated and therefore grow over time. WiFi range-based approaches, by contrast, convert the measured radio signal strength into range values, which indicate the distance between the target mobile device and a radio transceiver. Unfortunately, Received Signal Strength Indicator (RSSI) readings suffer from multipath propagation and unpredictable fluctuations due to non-static environmental conditions.

A particle filter was used to fuse information from the PDR sensor readings and range-based localization to provide indoor positions. This is done by "computing or approximating the posterior distribution for the state vector given all available observation at that time."[5] This takes advantage of the strengths of both approaches while correcting their respective weaknesses.

This work in particular focuses on the following aspects:

- It traces and evaluates the basic configuration of a server optimized to provide cloud-based positioning services, as well as the interface between server and clients. Offloading the computationally involving aspects to the server renders indoor positioning accessible to an increased number of mobile devices.

- It demonstrates novel ways to both, increase the accuracy of ranging as well as decrease the time required for the training process. This is achieved by using an optimized subset of training locations to build anchor node specific ranging models.
- It presents a client-implementation of an ESP32 microcontroller with multiple WiFi interfaces connected.
- It evaluates the feasibility of combining multiple WiFi interfaces for high-speed ranging.

1.2 Motivation

Historically, the complexity required to compute an indoor position based on raw data was often performed on the data-gathering device[6]. The redundancy of implementing and supporting the same algorithms on various platforms renders this approach difficult to maintain. We suggest a cloud-based solution with strict separation of concerns: while the computationally involving aspects are accomplished by an optimized server, the responsibilities of client devices can be limited to the gathering and uploading of data. With the computational efforts offloaded, this approach allows many more low-powered devices to be used for indoor positioning, thereby creating new opportunities and flexibility for projects where cost or size matters.

Due to the non line of sight (NLOS) propagation caused by walls and other objects, RSSI (Received Signal Strength Indicator) values cannot directly be converted to distance. Instead, regression is used to obtain a propagation model to relate RSSI readings to distances. However, optimizing these regressions tends to involve a lengthy training process. To boost regression accuracy and reduce the training effort at the same time, we studied the characteristics of various models in relation to the particular training set and specific access points.

Commonly, the configuration of WiFi networks requires about 2 seconds for a full network discovery (see 2.4 for details). This leads to a sampling rate of 0.5 Hz or less, which is not enough for real-time ranging of moving targets, particularly if smoothing of the data through means such as the Hampel- or Median filter is to be performed. To increase the sampling rate, a combination of one ESP32[7] and two ESP8266[8] microcontrollers with built-in WiFi microchips were used. We assessed their sampling rate in terms of speed as well as accuracy and evaluated potential complications.

1.3 Contributions

This work presents a cloud-based localization system which employs clients for data acquisition, and provides means to output computed positions through a web interface. We implemented and tested a working prototype. This lead to the following contributions:

- We provide a simple and efficient server setup to act as a foundation for a cloud-based indoor positioning system.

- We demonstrate that a dynamic selection of regression models and training locations yields a substantially improved ranging accuracy, compared to a predefined, static alternative, even at a considerably reduced training effort.
- We present a C++ based implementation of a minimal client based around the ESP32 microcontroller, capable of uploading ranging- as well as IMU (Inertial Measurement Unit) data.
- We show that combining various WiFi interfaces to increase the WiFi sampling rate is a sensible approach only under specific circumstances.

1.4 Overview

The structure of this thesis is as follows: Chapter 2 reviews the theoretical background required to follow the presentation of the various components of the proposed cloud-based indoor positioning system in chapter 3. Whereas chapter 4 delineates details of our test implementation, chapter 5 shows our evaluation results. Finally, chapter 6 summarizes our findings and concludes with an outlook to possible future work.

Chapter 2

Theoretical Background

The theoretical background covered in this section is required to understand the following chapters. It covers aspects related to the server configuration, information about each phase of the ranging process, as well as implementation details of an ESP32-client.

2.1 Task Scheduling on Multiprocessing Systems

For a cloud-based solution to scale well, managing CPU resources is essential. This section considers means to efficiently schedule tasks on one CPU core, as well as balance the total load over multiple cores.

2.1.1 Asynchronous and Non-Blocking Task Scheduling

Many servers, traditional web-servers included, are fundamentally stateless; they create and destroy one connection per request and instantly forget about the client[9]. Not only is this approach inherently flawed for our use case where states persist over extended periods of time, it is also very inefficient for continuous, fast-paced transmissions. In order to partly overcome this, one full thread can be assigned to each user. However, due to it mostly being idle, this can be very expensive. An alternative approach, partly popularized by Node.js[10], is to provide a single-threaded event loop where every client is placed on[11]. This mandates that application code must be asynchronous and non-blocking whenever possible, so as to not obstruct other clients.

The terms "asynchronous" and "blocking" are often used interchangeably. However, whereas "blocking" refers to a function waiting for something to happen before returning, as is often the case with network or disk I/O, "asynchronous" functions return before they are finished[12]. Care has to be taken that the only blocking functions are those of the CPU performing computations related to localization, and even those should be optimized as far as possible.

2.1.2 Load Balancing

With the computationally heavy aspects being offloaded to the server, load will at some point become too much for one core, or CPU. The possibility to add load balancing is therefore a crucial requirement. Many different configurations are being used, but one common strategy is to create various instances of the server which may run under different IP addresses and/or ports and place them behind a load balancer to provide a unified point of

access for clients. The responsibility of a load balancer is to manage the "efficient utilization of parallel computer systems" by partitioning the job "over the system in an optimal or near optimal fashion"[13]. It supports various mechanisms to assign tasks to servers, such as round-robin, least-connected, or a user-provided functions.

Some computer-language interpreters employ a global interpreter lock (GIL) to synchronize the execution of threads. It guarantees that non thread-safe code cannot be shared with other threads and increases the speed of single-threaded programs. However, it creates a significant barrier to parallelism, which can be circumvented only by spawning multiple processes.[14]

2.2 Data Transmission Strategies Between Servers and Clients

To find a suitable strategy for data transmission, two partly related aspects have to be considered: the communication protocol as well as the data format. Please see 4.1.2 for information about our specific implementation.

2.2.1 Communication Protocol

Many possibilities exist for client-server-communication. A selection has to be based on the following three considerations:

1. **Reliability:** If a certain amount of data corruption is acceptable, UDP might be an option[15]. TCP, on the other hand, guarantees reliable delivery of data. Many text-based data formats require reliable delivery or they cannot be read.

2. **Real-time, full-duplex communication:** Both, UDP and TCP support real-time, full-duplex communication[16]. However, a survey of the available options indicated that many more solutions exist based on TCP.

3. **Ease of implementation:** Particularly when building for as many different platforms as possible, solutions must be easy to implement and therefore standardized.

WebSocket, built on top of TCP and used by this work, is currently the de facto standard for true full-duplex communication, supported by many platforms[17]. It uses the HTTP upgrade header to change from the HTTP to the WebSocket protocol, and adds minimal framing on top of TCP to accomplish the following:

- Support multiple services on one port and multiple host names on one IP address.
- Remove length limits imposed by TCP.
- Add a web origin-based security model for browsers.
- Include an additional closing handshake which works in the presence of proxies.

2.2.2 Data Format

Several factors have to be considered when selecting a suitable data serialization format to transmit data between clients and server:

- **Purpose:** Many formats are created to serve a particular use case.
- **Size:** The message size is proportional to the auxiliary characters/bits required for data separation. Fewer bits result in an enhanced performance.
- **Readability:** A human-readable data format can facilitate the manipulation and debugging of data.
- **Popularity:** Particularly when building for as many different platforms as possible, solutions should be easy to implement. This requires a clear standard.

While there is a plethora of available options[18], this work uses JSON (JavaScript Object Notation)[19]. It encodes numbers, strings, booleans, arrays and objects into human-readable text consisting of attribute-value pairs. It is very common, has little overhead, and is built into the core of many language interpreters.

2.3 Signal Propagation Models

2.3.1 Signal Propagation

Ranging in our context denotes the process of deriving a range based on RSSI readings. Anchor Nodes (AN), in this case WiFi access points, which are placed throughout the floor, send out a low powered signal, which is received by a Mobile Node (MN) such as a smart phone. The strength of this signal decreases with distance. This process is predictable in a pure line of sight (LOS) situation such as an open field. Consequently, the range between MN and any given AN can immediately be computed.[20]

Walls and other non line of sight situations (NLOS) however cause the signal to behave unpredictably[21]. Multipath-propagation, caused by signal absorption, reflection and diffraction, is often modeled by using regression.

2.3.2 Regression Models for Multipath Propagation

Regression establishes relationships between variables[22]. The aim is to find an equation which maps measured RSSI (independent variable) to a range (dependent variable) as accurately as possible. Many different models exist and the best model for any specific purpose is often not directly evident.

This work evaluated the general characteristics of five regression models:

- Linear: $d = \alpha x + \beta$
- Exponential: $d = \alpha * e^{\beta x}$

- Power: $d = \alpha * x^{\beta}$
- Quadratic polynomial: $d = \alpha + \beta x + \gamma x^2$
- Cubic polynomial: $d = \alpha + \beta x + \gamma x^2 + \delta x^3$

were d represents the range between the target object and access point, x is the RSSI measured at each training position. α, β, γ and δ are environmental variables to be determined in the regression process. Note that all models except the linear one are forms of nonlinear regression with varying properties[23].

Quality Assessment

Regression requires RSSI values from various known locations to be measured. This process results in tuples consisting of location, as well as signal strength and distance in relation to each AN. These tuples can be used for both, creation as well as evaluation of regression models (cf. section 5.2.3). It is evident that the distribution of tuples among these two purposes affects the quality of the regression model as well as its evaluation. In order to distinguish their function, tuples used for creating a model are called Training Locations (TLs), whereas those for evaluating models are called Testing Points (TPs).

2.4 Network Discovery Frequency

Assessing the RSSIs for all ANs in a network commonly takes about 2 seconds[24]. While this may be enough for non-moving objects, a higher frequency is required the faster an object moves. This is particularly true if algorithms such as the Hampel- or Median filter are used to remove outliers. The Hampel-filter, for instance, works by considering the previous n (typically 4-6) samples, removing those samples differing considerably from the mean and averaging the rest[25]. Therefore, a sampling rate of 0.5 Hz quickly leads to out-of-date positions.

The slow network discovery frequency is due to the fact that according to the IEEE802.11 specification[24], RSSI information is attached specifically to beacon frames. Beacons are sent at an interval defined as the Target Beacon Transmission Time (TBTT), which by most manufacturers is set to 102ms. However, if the wireless medium is busy, the AN will have to contend for access to the medium[26]. It is therefore a sensible approach of most MN drivers to listen for slightly over 100ms on one channel and repeat this process for every channel available – 11 to 14 for a 2.4 GHz network, depending on the geographical location[27].

There are thus three major options to considerably increase ranging frequency:

1. Limit all ANs to one channel. This will allow a single MN to achieve a frequency of close to 10 Hz, which constitutes a hard limit.
2. Change the TBTT to a value smaller than 102ms.
3. Employ more than one MN concurrently. Each one could for instance be assigned a specific set of channels to query.

Even though a combination of option 1 and 2 would be expected to accelerate network discovery considerably, this is often not possible due to additional network requirements. This thesis therefore focuses on option 3, which is implemented by means of one ESP32[7] and two ESP8266[8] WiFi controllers.

2.5 Related Work

Task Scheduling on Multiprocessing Systems. Due to the importance of efficient task scheduling on multiprocessing systems, many papers offering a multitude of solutions for various environments have been written[28]. The subject of event loops however, popularized largely by Node.js[29], lags behind. Still, [30] suggests that Node.js with its event-driven model compares favorably to IIS, which represents a traditional web server. This is true particularly for I/O driven applications.

WebSocket Performance. More work has been done in regard to WebSocket performance, which is compared mainly to its closest but older competitor, long-polling, as well as raw TCP sockets. [31] claims a 500-1000x reduction in unnecessary network traffic and a 3x reduction in latency, compared to long-polling. Less dramatic, but still profound improvements of WebSocket over long-polling were found by [32], citing up to 2.5x less overhead per packet, and 48x more round trips per second. [33] claims a dramatic reduction of network bandwidth by eliminating the unnecessary HTTP header data, whereas [34] asserts a reduction in header data from kilobytes to 2 bytes. When compared to data transmission through a TCP socket, [35] found performance penalties of up to 5x with respect to protocol overhead, as well as up to 3x with respect to payload delivery delay and throughput, respectively. This is contested by [36]. While the overhead in regard to handshaking is indicated as up to 3.7x compared to raw TCP, it is only around 1% for general traffic.

Indoor Positioning. Indoor positioning has been investigated for a long time. A multitude of technologies have been proposed, each with their own set of advantages and problems. According to [37], any solution will be a trade-off between accuracy, precision, complexity, robustness, scalability and cost. [38] and [39] rely on RFID signal strength readings, which [39] claims to result in a 50 percentile error distance of around 1 m. [40] suggests GSM based fingerprinting and achieves a median accuracy of 5 meters by not only relying on the 6 strongest cells traditionally used in the GSM standard, but additional cells that are strong enough to be detected, but too weak to be used for efficient communication. Ultra-wideband technology is used by [41] and [42] to overcome some of the limitations of other technologies and achieve a very high indoor location accuracy of 20 cm. [43] and [44] suggest Bluetooth-based indoor positioning and achieve an error of 3.4 m and 3.76 m, respectively.

Because of its ubiquity, using WiFi radio signals for indoor positioning has been studied extensively. Commonly used parameters for target localization are RSSI[45] and time information[46]. Fingerprinting is often used because of its robustness to multipath propagation[6] and can achieve an

80-percentile error of 2 m[47] or even less than 1 m with a 72% probability and less than 2.6 m with a 95% probability by using a neural network based classifier[48]. Range-based methods on the other hand, which convert RSS values to distance and locate the target based on techniques such as lateration, require much less training effort, compared to fingerprinting[49].

To eliminate the ranging error, caused particularly because of multipath effects, regression analysis is often used. Whereas [50] and [51] rely on linear regression, [52] finds that an exponential model yields optimal results.

Not much work has been done on the subject of increasing the WiFi sampling rates. However, by studying the influence of beacon frames transmitted at 1 Mbps, [53] finds that their effect on network performance is negligible.

Chapter 3

Architecture

Chapter 2 presented a theoretical background which will now find practical application. In this chapter, a centralized, cloud-based localization system is proposed, which accepts and manages connections from various devices. Further, a MN is being presented as a means to increase sampling frequency. Figure 3.1 shows the relevant architecture.

Clients, in this context, are portable devices capable of gathering data required for the computation of a position and/or displaying computed locations. This raw data is sent to the server, which is in charge of data storage and well as data processing. The position found can optionally be sent back to the client for display.

3.1 Server: Cloud Storage and Processing

The server is divided into three sections: Interface, Storage and Processing (cf. Figure 3.1).

3.1.1 Interface

Communication between clients and server is handled by a WebSocket implementation. Optionally, basic flow control can be enabled, which causes the server to respond to every transmission, expecting the client to hold further transmissions until successful reception of its response. The response of the server consists of an integer transmitted from the client; the client is expected to resend the original message if no response was received from the server for a defined period of time.

3.1.2 Storage

The server handles and stores three types of data:

1. **Experiment Data** is the raw data uploaded from clients (see 4.1.2 for details). It is forwarded to the ranging and rotation algorithms for processing.

2. **System Data** consists of environmental variables required for the ranging and rotation algorithms to function. It is composed of the following information:

    - **The location and its GPS position.** GPS coordinates can be transmitted from the client during initialization as a method to identify the active location of the client.

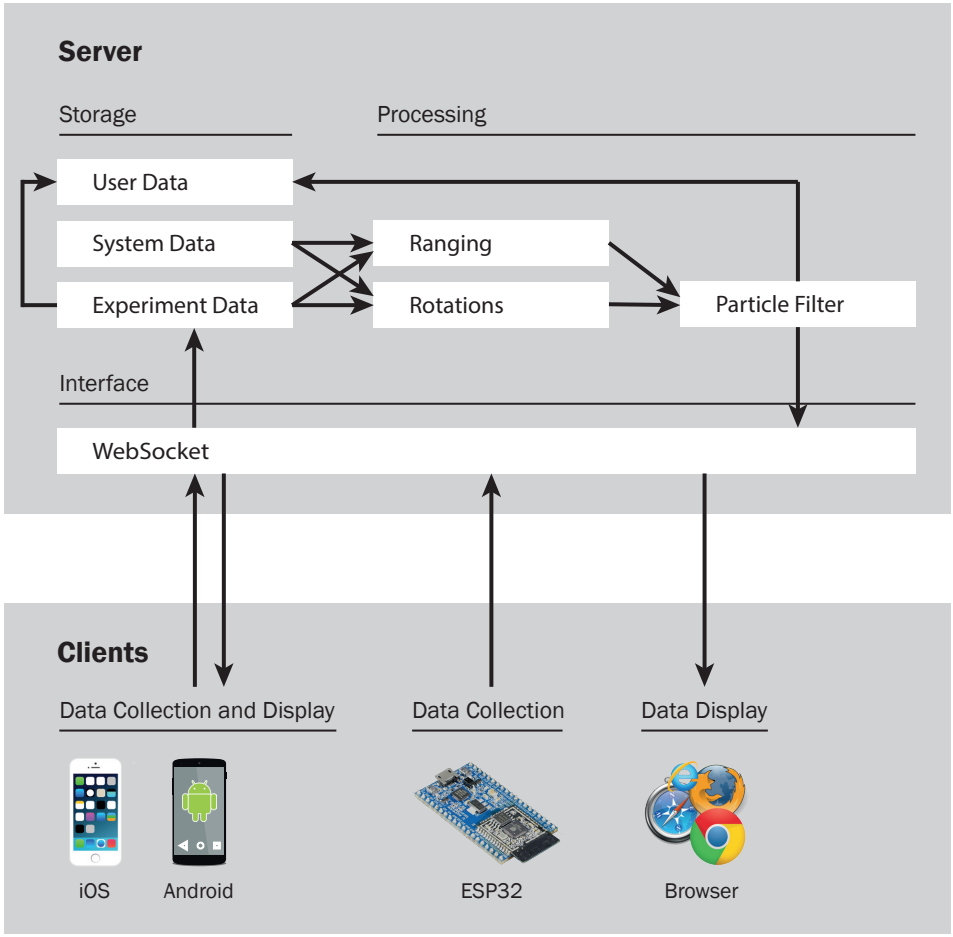

FIGURE 3.1: Architecture of the cloud-based indoor positioning system.

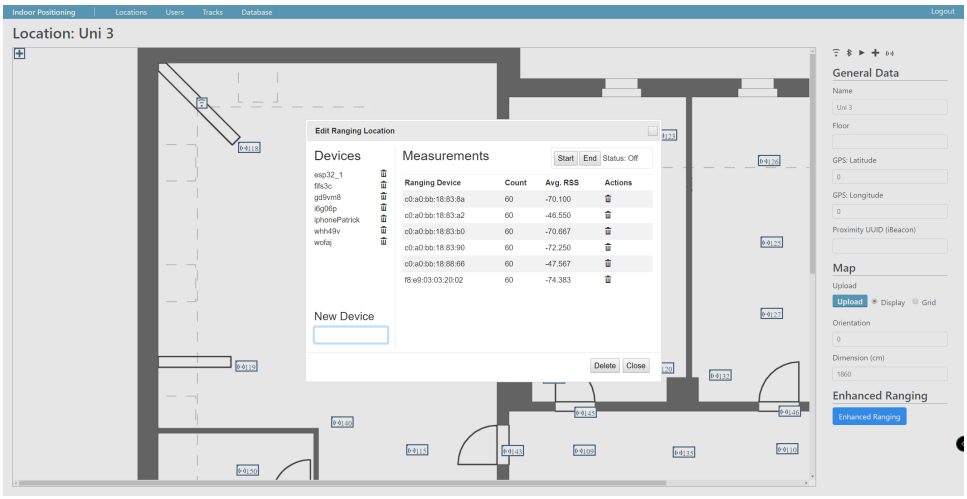

FIGURE 3.2: Administration interface to update environmental data

- **The ANs of the active location and their position.** The ANs can be used during the initialization process as another means to identify the active client location. This works by comparing all ANs visible to the client to those stored for that particular location. During the training phase, the position of the ANs is used to derive the distances to the TL, which constitute the dependent variable of the regression. During the testing phase, the position is required as the origin of the computed range.
- **The client specific regression model for each AN at any given location.** Specifically, this data consists of the environmental variables α, β, γ and δ derived at during the regression process. It is used during the recall phase to acquire the distance between AN and MN based on RSSI values.
- **The TLs and their position.** This information is used only during the training process, together with the ANs, and constitutes the dependent variable of the regression.
- **The signal strengths measured from any given TL.** This information is used only during the training process and forms the independent variable of the regression.

This data can be managed through an administration interface that is exposed through a regular web server running on a port different from the one used for WebSocket (see Figure 3.2).

3. **User Data** includes the raw data sent by the clients as well as the corresponding calculated positions. This information allows a track to be replayed at a later time. Architecturally, user data differs from experiment data in that user data is stored permanently, whereas experiment data is merely forwarded to the appropriate methods for processing.

3.1.3 Processing

General Overview

Ranging and Rotations depend on both, experiment data and system data. While the ranging algorithm returns ranges to each access point, Rotations return a vector indicating the relative movement of the MN. Lastly, this information is consumed by the particle filter to derive the most likely position of the MN. This position is stored for each user and, if required, can be sent back through the WebSocket connection to be displayed in real time.

This means that every time the server receives new data from a connected client, the following primary steps are placed on the event loop:

1. Perform rotation update if rotation data was received
2. Perform ranging update if ranging data was received
3. Perform localization update with particle filter
4. Return location to client

Consequently, it is mandatory for these algorithms to perform efficiently so as to not block other clients.

Additionally, a few auxiliary processes for session management or data storage are implemented. Because WebSocket does not always send a formal command to terminate the connection[17], the server checks for inactive connections once a minute. Both their raw data as well as computed positions are then stored in a persistent database and removed from memory. This database access could be offloaded to a dedicated database server, and therefore scales easily if required.

Ranging

Due to this work being a joint project, the rotation update and particle filter are described elsewhere. Ranging is performed in two separate steps:

Training. Before indoor positioning is possible, suitable regression models are created for each AN of any given location (cf. sections 2.3.2 and 5.2.1). In this case, only the ranging module is called and no positions are calculated by the particle filter.

Localization. To compute the range in respect to various ANs, the regression models created during the training phase are applied to the RSSI data received from the clients. These ranges are then forwarded to the particle filter, which calculates the indoor position.

3.2 Clients

3.2.1 Display of Calculated Positions

Positions can be displayed by means of a web server exposed on a port different from the one used for the WebSocket connection. This setup allows for much flexibility, because the position of any MN can be displayed in

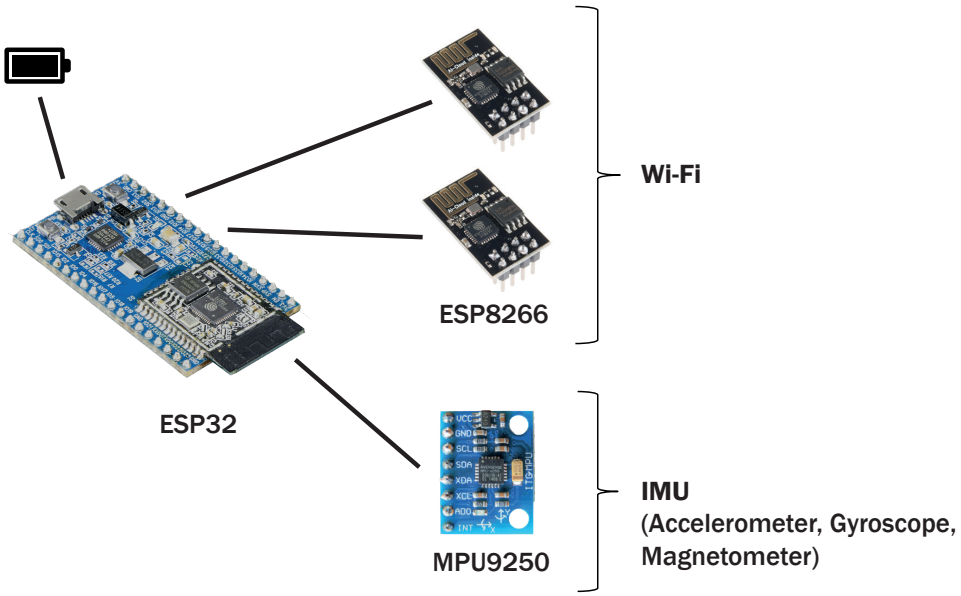

FIGURE 3.3: Setup of the ESP32 ranging client

any web browser by passing its identity during the connection. Real-time updates are broadcast to the browser through WebSocket.

3.2.2 ESP32

This work attempts to combine RSSI data and accelerometer, gyroscope and magnetometer readings to determine the position of a MN. System on a chip (SoC) microcontrollers were used, because they allow for much flexibility in connecting additional modules. In particular, a ESP32[7] served as a controller, to which two ESP8266s[8] and one MPU9250[54] were connected to acquire WiFi-, and inertial measurement data, respectively (see section 4.2 for implementation details).

This work attempts to increase sampling frequency by employing more than one MN concurrently. Effective ranging relies on as much of an unambiguous relationship between signal strength and distance as possible. This implies that all RSSI values gathered from a set position should be as similar as possible. In order to avoid errors because of varying calibrations between models, only the two ESP8266s were used for data acquisition, whereas the WiFi controller built into the ESP32 was used for communication with the server. The two ESP8266s were positioned approximately one cm apart, were equally aligned, and their orientation was locked. The ESP32 served as a controller, managing the flow of data between connected devices and the server. Figure 3.3 illustrates this setup.

It is evident that no real time location can be displayed with this setup. However, monitors could be connected to an ESP32. Alternatively, any browser on another device can be coupled with the ESP32 to display the position in real time (cf. 3.2.1).

Chapter 4

System Implementation

This chapters details the implementation of the system outlined in chapter 3. See [55] and [56] for code access.

4.1 Server Configuration

There are two aspects to the server: Hardware and Software

4.1.1 Hardware

The server instance has the following specifications:

- **Processor:** Intel Xeon E312xx @ 2.5 GHz
- **Cores:** 4
- **Memory:** 8 GB
- **HD:** 80 GB

4.1.2 Software

After a listing of existing software packages installed for the initial configuration, this section will detail various core aspects of the implementation of the indoor positioning system.

Initial Configuration

Initially, the server was configured as follows:

- **OpenStack** was used as a virtualization platform[57].
- **Ubuntu 16.4** was used as operating system.
- **Python 3.6** was the programming language used for the implementation. No code was directly written in any other language.
- **Tornado 4.5.2** was used as web framework and asynchronous networking library.
- **SQLite 3.11.0** was used for persistent data storage.
- **Peewee 2.10.2** was used as ORM (object-relational mapping).
- **Nginx 1.10.3** was used as a load balancer.

Database Implementation

All data was saved in an SQLite database because of its simplicity and responsiveness for few users. Peewee, which maps object manipulations in Python to database queries, supports SQLite, MySQL as well as PostgreSQL. Changing database server would therefore be easy, should requirements change.

Data model Please refer to Figure 4.1 in respect to the following section. What follows is a short introduction to the relevant components:

- **Location:** Contains basic information about each location.
  - **name:** Name of the location
  - **gps_latitude, gps_longitude:** GPS coordinates can be transmitted from the client during initialization as a method to identify the active location.
  - **map_display, map_grid:** Contains the file extension of the images of the floor plan for display and grid calculation.
  - **map_orientation:** In degrees from north. Used for rotations.
  - **map_width, map_scale_start_x, map_scale_start_y, map_scale_end_x, map_scale_end_y:** Whereas map_scale_* defines two points on the map, map_width expresses the distance in cm between them. Used for conversion between pixel and cm.
- **RangingDevice:** Contanis information about ANs.
  - **identifier:** MAC-address for WiFi access points. Has to match string sent by MNs.
  - **signal_strength:** RSSI-values for WiFi access points. Used for regression.
  - **position_x, position_y:** Defines position on the map.
  - **type:** Typically "bluetooth" or "wifi".
  - **location_id:** Foreign key. Indicates location the AN belongs to.
- **RangingLocation:** Contains information about TLs/TPs.
  - **position_x, position_y:** Defines position on the map.
  - **location_id:** Foreign key. Indicates location the TL/TP belongs to.
- **RangingModel:** Contains information about the regression model.
  - **name:** Name of the regression model.
  - **client_id:** Foreign key: Regression models are specific to clients.
  - **ranging_device_id:** Foreign key: Regression models are specific to ANs.
  - **data:** Contains the model-specific variables α, β, γ or δ as a JSON string.

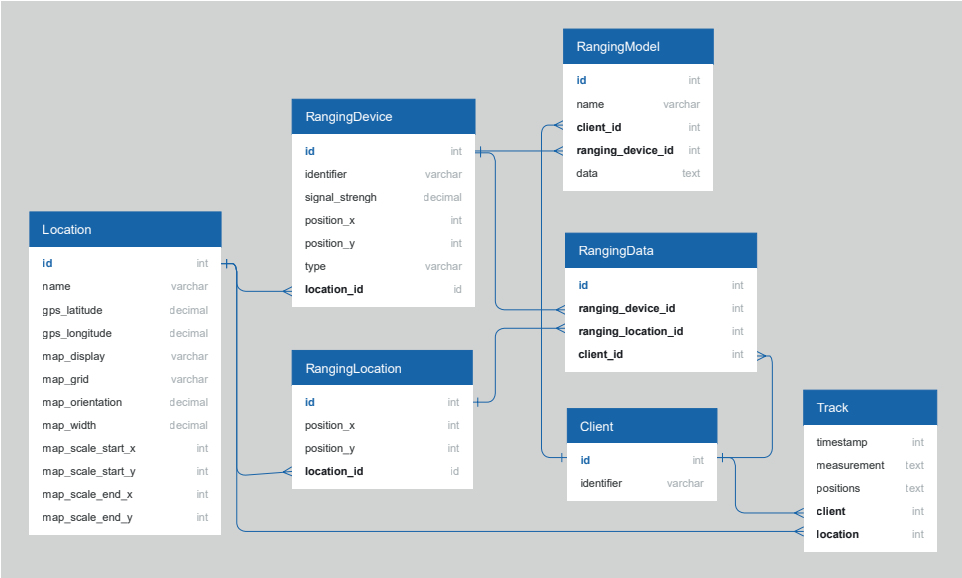

FIGURE 4.1: Database schema

- **RangingData:** Contains the signal strength measured during the training process (see 5.2.1 for more information).
  - **ranging_device_id:** Foreign key: Ranging data is specific to ANs.
  - **ranging_location_id:** Foreign key: Ranging data is specific to TLs.
  - **client_id:** Foreign key: Ranging data is specific to clients.
- **Client:** Contains information about the MN.
  - **identifier:** Unique identifier of the client transmitted during initialization and used to identify the MN in the administration area.
- **Track:** Contains information on MN data uploads.
  - **timestamp:** Unix timestamp to denote the time of the capure. Seconds since epoch.
  - **measurement:** Contains raw data in JSON format.
  - **positions:** Contains computed positions in JSON format.
  - **client:** Foreign key to MN.
  - **location:** Foreign key to location.

Communication

Tornado[58] was used to provide both, a general web server as well as a WebSocket server. There are various existing Python-based server implementations with capabilities required for this project to choose from. A decision for Tornado was based on the following reasons:

- It has a relatively small footprint and aligns well with our needs.

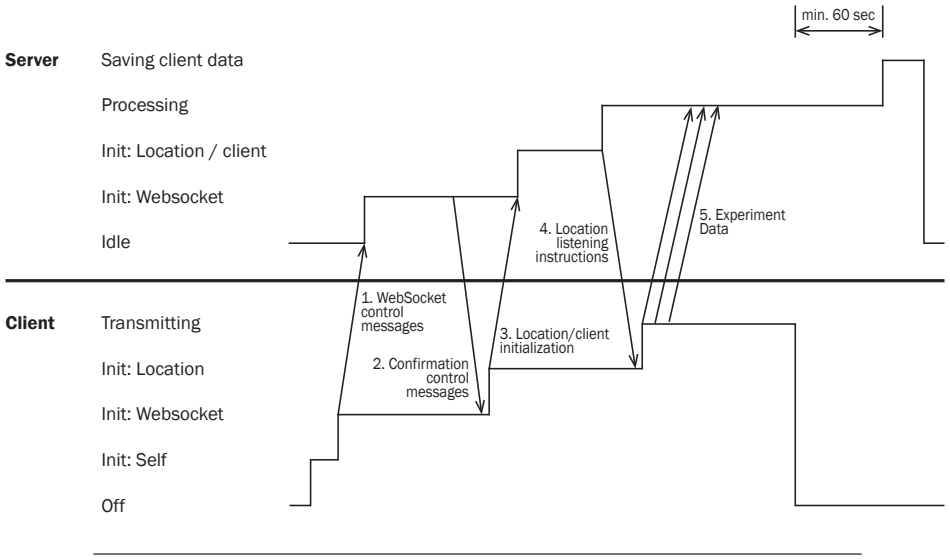

FIGURE 4.2: Data Flow Time Diagram

- It provides stable implementations of booth, a general web server as well as a WebSocket server, both of which are required.
- It scored high when comparing the speed of various frameworks.
- It is an active project with regular updates.
- It is well documented.
- It supports load balancing, and can easily be combined with a third party load balancer.

Data Flow

The controller ensures that uploaded data is handled appropriately. This section will introduce the basic communication scheme between client and server. Please compare Figure 4.2 for the following explanation.

When a MN connects for the first time, it can optionally transmit a number of minimal WebSocket messages which are instantly returned by the server (cf. 1. and 2. in Figure 4.2). This is to ensure that the channel is ready before the client is expected to send an initialization message, shown below (cf. 3. in Figure 4.2). Find inline comments for documentation.

```
{
    // Type: Init
    "t": "i",

    // Data from here
    "d": {
        "location": {

            // Option 1: Location to be determined by ANs found
            "type": "wifi",
```

```
            // Two ANs found
            "devices": ["abc", "def"]

            // Option 2: Location to be determined by GPS
            "type": "gps",

            // GPS position of MN
            "latitude": 46.897382,
            "longitude": 6.325643
        },

        // Identifier of MN
        "upload_id": "abc_123",

        // This MN will upload raw track data
        // ("download" is for those devices displaying positions
        // through the browser)
        "type": "upload",

        // This MN supports both, wifi and bluetooth
        "capabilities": ["wifi", "bluetooth"]
    }
}
```

If the controller is capable of determining the location of the MN, it will respond with a message containing the ANs to listen to (cf. 4. in Figure 4.2):

```
{
    "t": "i",
    "d": {

        // Listen to "def" and "abc" WiFi access points.
        "wifi": ["def", "abc"],

        // Equivalent for bluetooth.
        "bluetooth": ["ghi", "jkl"]
    }
}
```

Both, client and server are now ready to transmit raw data updates (cf. 5. in Figure 4.2):

```
{
    // Type is "Data"
    "t": "d",

    // Optional, used for flow control, is being returned to sender
    "c": 123,
    "d": [
        {
            // Timestamp. Precision: milliseconds
            "t": 20171018205530567,
```

```
            // Accelerometer, gyroscope and magnetometer data
            "a": [0.12, 1.23, 2.34],
            "g": [3.45, 4.56, 5.67],
            "m": [6.78, 7.89, 8.90],

            // Ranging data
            "r": {
                "abc": [40, 41, 42.3],
                "def": [50, 49.8]
            }
        },
        {...}
    }]
}
```

Not all data is required during transmission. For each transmission, the controller does these things:

1. Save measurements for later use.
2. If IMU-data was submitted: Send data to rotation algorithm. Returns vector.
3. If ranging-data was submitted: Send data to ranging algorithm. Returns distances to ANs.
4. Send results to particle filter. Returns position.
5. Save calculated position for later use.
6. If display clients are connected: Send calculated position to them through WebSocket (not part of this thesis).

Ranging

The central aspect of the ranging process is in the application of various regression models. To this end, every model is implemented as its own class, which extends another class *Regression*, containing general helper methods. The most relevant are:

- **get_data**. Aggregates and returns the data gathered during the training phase to create regression models.
- **get_regression_data**. Returns environmental variables for one particular regression model.

The model classes themselves also contain two principal methods:

- **compute_data**. Computes the environmental variables of the regression model based on data received from *get_data*.
- **get_distance**. Computes the range to one AN based on signal strength and the variables received from *get_regression_data*.

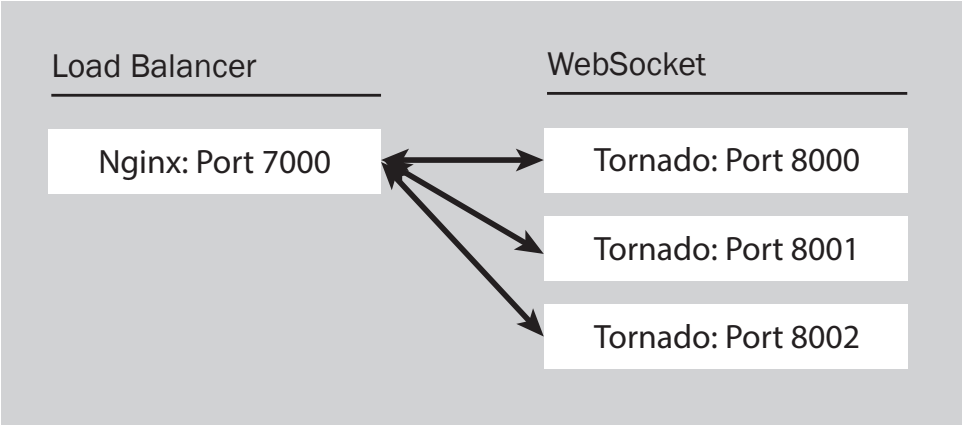

FIGURE 4.3: Schematic overview of port forwarding during load balancing

Whereas the data flow indicated above is the same for training as well as localization, the internal routing is not. The data uploaded from the client is sent to *compute_data* whenever the training process is started through the administration area (cf. section 5.2.1 for further details). Otherwise, it is sent to *get_distance*.

Load Balancing

Tornado provides a built in load balancer which is easy to enable for basic deployments[59]. However, due to the Python GIL (Global Interpreter Lock), taking advantage of multi-CPU machines is possible only if multiple Python processes are run[14]. Each process will be exposed on a different address/port. Consequently, a load balancer like Nginx[60] is required to present one common address and port to outside users. See Figure 4.3.

What follows is a listing of the most relevant components of a suitable Nginx configuration. Please see inline comments for further information. Note that in order to achieve compatibility with HTTP, WebSocket uses the HTTP Upgrade header to be activated. This handshake expects at least two additional headers, seen at the bottom of the configuration[61].

```
http {

    # Enumerate all the Tornado servers
    upstream frontends {
        server 127.0.0.1:8000;
        server 127.0.0.1:8001;
        server 127.0.0.1:8002;
    }

    # Set the variable $connection_update to be in compliance
    # with the WebSocket standard.
    map $http_upgrade $connection_upgrade {
        default upgrade;
        ''      close;
    }
```

```
    server {

        # Listen to port 7000
        listen 7000;

        location / {

            # Sets addresses of proxied servers
            proxy_pass http://frontends;

            # Use of WebSocket requires HTTP version 1.1 or higher
            proxy_http_version 1.1;

            # Set WebSocket upgrade headers
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection $connection_upgrade;
        }
    }
}
```

Measuring Execution Time

To measure execution time of the computationally heavy aspects of this architecture within Python, the `time.perf_counter()` method is used because it is the most precise timer for short periods and is capable to measure microseconds[62]. These aspects include steps 1 to 3 from 3.1.3 as well as various minor administrative processes like session management. With this being a joint project, the rotation update and particle filter was not yet implemented at the time of this writing and could therefore not be measured. Any benchmarks therefore apply only to the ranging update as well as all auxiliary functions of the server.

4.2 ESP32 Microcontroller as Client

This section first details the setup of the ESP32 with its connected devices, followed by a description of the implementation of the evaluations in regard to its reliability and performance.

4.2.1 Setup

Hardware

The ESP32 used has the following specifications:

- **Processor:** Dual-core Tensilica LX6 microprocessor @ 240MHz.
- **SRAM:** 520kB
- **Flash memory:** 4MB
- **WiFi:** Integrated 802.11 BGN transceiver

- **Connectivity:** 28 GPIO, 4 x SPI, 2 x I^2C, 3 x UART (among others not used here)

Through these interfaces, multiple external WiFi controllers can be connected, to be controlled by the ESP32. Development boards also feature a USB interface to simplify transferring the compiled code from a PC.

The connected ESP8266-01 WiFi-controllers have the following specifications:

- **WiFi:** 802.11 BGN transceiver
- **Flash memory:** 1MB
- **Connectivity:** UART

The connected Grove IMU 9DOF v2.0 has the following specifications:

- **Connectivity:** I^2C, SPI
- **Capabilities:**
  - **Gyroscope:** 3-axis with range of ±250°/sec, ±500°/sec, ±1000°/sec and ±2000°/sec.
  - **Accelerometer:** 3-axis with range of ±2g, ±4g, ±8g and ±16g.
  - **Magnetometer:** 3-axis.

Softare

The ESP32 exposes an integrated API to be used through C. Due to the ubiquity of the Arduino framework, an integration from Espressif Systems, the developer of the ESP32, and most notably the availability of many libraries covering a vast amount of use cases, many use the Arduino IDE to program for the ESP32.

Libraries The following libraries were used to implement various aspects of the ESP32:

- **WiFi:** WiFi for ESP32[63]
- **WebSocket:** Websocket client for Arduino, with fast data send[64]
- **UART:** HardwareSerial for ESP32[63]
- **MPU9250:** MPU9250_asukiaaa[65]
- **JSON:** ArduinoJson[66]
- **Queue:** QueueList Library for Arduino[67]
- **Time:** Time for ESP32[63]

Even though the entire implementation is close to 1,500 lines in length, the two main functions provide a good overview over the basic setup. Please note the inline comments.

```
// This function is executed only once upon powering on the ESP32
void setup() {

  // Initialize various debugging functions
  debugInit();

  // Establish connection to server.
  // Includes WiFi-connection, server-connection and upgrade
  // to WebSocket.
  // All functions are synchronous.
  serverConnectionEstablish();

  // Retrieves time from NTP server.
  timeConfig();

  // Initializes I2C connection for MPU9250
  imuInit();

  // Initializes serial connection to ESP8266s
  rangingInit();

}

// The loop function is constantly executed.
// All containing functions are anynchronous and keep track of
// their own state.
void loop() {

  // Use connectionStatus to keep track of status
  if (connectionStatus == WEBSOCKET_INITIALIZED) {

    // Send a few empty messages and expect response
    // to ensure that connection is indeed online.
    bool websocketReady = serverCommunicationWebsocketStart();
    if (websocketReady) connectionStatus = WEBSOCKET_READY;
  }
  else if (connectionStatus == WEBSOCKET_READY) {

    // Put the ESP8266s in station mode
    rangingResetDevices();
  }
  else if (connectionStatus == RANGING_DEVICES_READY) {

    // Find WiFi access points in area for autodetection of location
    rangingFindAccessPoints();
  }
  else if (connectionStatus == SERVER_CONNECTION_INIT) {

    // Send list of access points to server and wait for response
    serverCommunicationInitProcess();
  }
```

```
  else if (connectionStatus == FIND_CHANNELS) {

    // Find specific access points after first response from server,
    // or update channels in case of change.
    rangingFindAccessPoints();
  }
  else if (connectionStatus == READY) {

    // Get current IMU data
    if (!rangingMeasurements) {
      imuUpdate();
    }

    // Get current ranging data
    rangingUpdateProcess();

    // Send gathered data to server
    serverCommunicationProcess();
  }
}
```

While these functions may be executed several hundred times every second, internal timers are used to limit activities to a predefined interval. Typically, *imuUpdate()* and *serverCommunicationProcess()* is allowed to run 10 times per second, whereas *rangingUpdateProcess()* is executed as often as possible to minimize the delays related to the beacon frames (cf. 2.4 for details).

Synchronization between the two ESP8266 happens through an internal variable which keeps track of the last access point queried. Whenever one of the WiFi-controllers is free, this variable is increased by one, or reset to zero, thereby instructing it to query the subsequent AN.

4.2.2 WiFi Network Scanning: Sampling Rate

Being that performance is useless without reliability, both aspects will be evaluated:

Reliability

There are various concerns when using multiple WiFi interfaces:

1. Even though they are the same hardware model, they might not be calibrated the same way.
2. Even though they are positioned as similarly as possible, the difference might suffice to return divergent data.
3. Because they are positioned as closely as possible, interference might distort the quality of the data.

To test for these potential problems, the WiFi-controller of the ESP32 was disabled and data between one location and one access point in a static environment was gathered with both devices separately, as well as with

both of them combined. The duration for each test was approximately 5 minutes. Divergent averages between both devices would indicate problem one or two, while high deviations in any of the individual data sets would point to problem three.

Performance

Espressif Systems, the creator of the ESP8266 WiFi controller, provides an AT instruction set[68] to control its various aspects. There are two distinct options to list signal strengths:

1. Detect all available access points
2. Detect a single access point, given a specific SSID, MAC-address and channel

In our particular testing environment, only 6 of the approximately 20 ANs were to be considered. Based on the theory presented in section 2.4 and with channels being flexible, it is to be expected that the second option would perform better. However, because channel numbers can change at arbitrary intervals, it is necessary to periodically update channel numbers by using option 1.

4.3 Communication Technology: WebSocket

For the rotational updates to function well, approximately 10 measurements from the accelerometer, gyroscope and magnetometer should be transmitted to the server every second. Efficiency, particularly of the communication interface, is therefore an important consideration.

To assess the behavior or the WebSocket interface under load, benchmarks for various message sizes as well as various numbers of concurrent connections were run. The focus of this test was on the capabilities of the server, not the network. Therefore, both, client and server were run on the same computer, powered by an AMD Ryzen 1600x CPU @ 3.6 GHz with six cores and 16 GB of memory. A browser-based WebSocket client, capable of initiating up to 250 concurrent WebSocket connections, was used to simulate mobile nodes. It was executed within Chrome version 67. This test was not executed on the default server described above, because only terminal access was available which renders running Chrome impossible.

To evaluate whether the message size would affect overall throughput, 100,000 bytes were split into chunks of 50, 100, 300, 500 and 1000 bytes respectively. These chunks were then transmitted sequentially and the time needed was measured. Similarly, to establish the scalability of the interface, 10, 100 as well as 250 concurrent connections to the server were opened, each of which sending 1,000,000 bytes. Ideally, the total time required was to be strictly proportional to the amount of concurrent connections.

Chapter 5

Evaluation

This chapter presents the evaluation of the testing environment introduced in the two previous chapters.

5.1 Server

With the computationally heavy aspects being offloaded to a server, a quick processing time is important because it directly translates to the number of clients that can be served. As indicated in 4.1.2, only the ranging update as well as time required for auxiliary functions (see 3.1.3) could be measured. Average processing time for one request was 0.000099 seconds, or approximately 0.01 ms. This implies that 100,000 clients could be served from one thread, irrespective of the WebSocket performance. However, this number will undoubtedly decrease as the rotation update and particle filter are implemented.

5.1.1 Data Transmission: WebSocket Performance

This section measures the maximum number of concurrent WebSocket connections one thread can handle. Due to the assumption that WebSocket performance might be affected by message size, the average size for our use case was first evaluated to be approximately 293 characters per message (see 4.1.2). Even though UTF8 supports variable length encoding, 1 byte covers all characters used in our case. 300 bytes per message is therefore a good approximation.

On the basis that some modifications to the message format used would have been possible, various message sizes around the 300 byte mark were evaluated for their efficiency. However, the results (see Figure 5.1) show that very little could be gained by adjusting for throughput, partly because message size would not heavily affect it, and partly because at 11.25 seconds for 100,000 characters, 300 byte chunks were surpassed only slightly in speed.

When evaluating the effects of concurrency, each connection would send 1,000,000 bytes in chunks of 300 bytes. The results indicate that at least up to 250 concurrent connections, concurrency does not affect throughput. Table 5.1 shows that it takes approximately 0.12 ms to send one message. It follows that at 10 messages per second, approximately 800 concurrent connections could be handled per server ($\frac{1sec}{0.00012sec/message*10messages/sec} = 833$). Even though this only amounts to 2.5 MB/sec, whether or not a locally restricted wireless network would be able to sustain this rate with over 800 simultaneously connected clients, is doubtful[69].

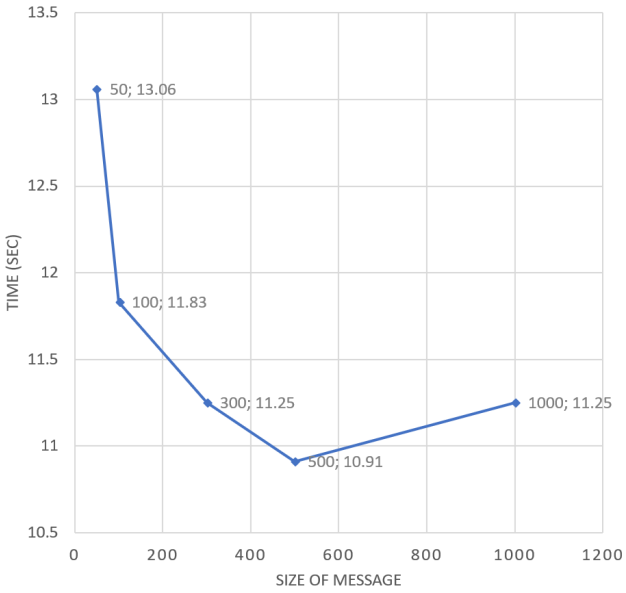

FIGURE 5.1: Time required to send 100,000 bytes in various chunk sizes through a WebSocket connection.

TABLE 5.1: WebSocket throughput for concurrent connections

Concurrent connections	**10**	**100**	**250**
Time total (ms)	4,264	38,960	100,778
Time per message (ms)	0.12792	0.11688	0.1209336

All this suggests that the actual WiFi implementation is likely the major bottleneck, followed by the Tornado WebSocket implementation and lastly by the computational power of the server itself (which may be subject to change).

5.2 Ranging Accuracy

5.2.1 Experimental Setup

To assess whether or not ranging accuracy could be enhanced, two steps were performed:

- Acquire TLs from various positions throughout the testing area.
- Process the data to determine the optimal regression model.

Data Acquisition

Experiments were conducted in an office area of nine separate rooms on 288 m^2, located at the Institute of Computer Science at the University of Bern (see Figure 5.2). 2880 samples from 48 predefined locations, each consisting of RSSI values to six WiFi access points, were gathered in around one hour. At each location, readings to each of the six ANs were collected at four directions, each 90 degrees offset from the other, in order to minimize the effect of the subject conducting the measurements. While the two ESP8266 micro-controllers were used to gather the data, the WiFi controller of the ESP32 was used to simultaneously communicate with the server.

Data Processing

To find the optimal regression model, data was evaluated in three steps. All computations were performed with Python.

Step 1. Evaluate general characteristics of regression models: The general characteristics of all five regression models were evaluated, which included behavior at lower TL counts. Given that any number of TLs could be selected from the 48 locations measured, the process was as followed for every model: For each number of TLs between 5 and 40, 1000 combinations of TLs were randomly chosen to create a regression model. To assess the error of the model at every iteration, the measured signal strengths from various TPs (see 5.2.3 for information on their selection) were applied to the respective regression model. The differences between the resulting ranges and the equivalent ground truth ranges were averaged to derive the error of one iteration. Eventually, all iteration errors were again averaged to derive the median error for any particular TL count for one regression model. This represents the expected error when statically creating a regression model based on all collected TLs. Thus:

Error per iteration:

$$e_i = \frac{\sum_{j=1}^{t} |\hat{d}_j - d_j|}{J},$$

FIGURE 5.2: Floorplan of the testing environment. Blue dots denote WiFi access points, red dots locations at which samples were taken.

where e_i is the error of one iteration, i is the index of the iteration, j is the index of a testing location, J is the amount of testing locations, d_j is the ground truth and $\hat{d}_j$ is the estimated range.

The median error for one particular TL count is the average of all iteration errors:

$$e = \frac{\sum_{i=1}^{I} e_i}{I},$$

where e is the median error for one particular TL count and I is the number of iterations per TL count.

In addition to the median error for any TL count, the data from the individual iterations reveals different behaviors in respect to the standard deviation as well as distances between MN and AN. This data served as baseline to compare step 2 and 3 (see below) against.

Step 2. Evaluate optimized selection of TLs: The assumption that a specific, optimized selection of TLs might produce a regression model with a higher accuracy than a random set of TLs, even at a much higher count, lies at the foundation of step 2. To assess this assumption, step 1 was repeated, again for various combinations of TLs. However, instead of calculating the average, the one with the minimum error and, therefore, optimal accuracy was selected. The gain of this process was assessed by comparing this minimum to the average error from step 1.

Step 3. Produce composite minimum: To determine whether the error could be reduced further by assigning individual regression models to access points, the respective minimum error for each access point out of all TL counts was computed by extending step 2. This process resulted in a composite minimum with optimized models per access point. It served as final, optimized error and, therefore, provided optimal accuracy.

Ranging Optimization at lower TL counts

Up to this point, this process assumed the availability of 48 TLs, from which any set resulting in an optimized, minimal error could be chosen. However, for it to be beneficial in reducing the training effort, the final, composite error should yield substantially improved results, even at much lower total TL counts. Therefore, to ascertain that this process does not depend on a high number of total TLs, the computations were repeated for sets with fewer than 48 total locations.

5.2.2 Experimental Results

Figure 5.3 shows the average error by number of TLs used to create the regression. Whereas the power model is the most stable overall, it is slightly surpassed by the polynomial models at high numbers of locations. The exponential model surpasses the linear model only slightly and is otherwise inferior in these tests[49]. The best average error at 40 locations is 135cm, achieved by both polynomial models. The abysmal error of over 3 m of

TABLE 5.2: Performance vs number of locations

Configuration	Avg. error	50% Acc.	90% Acc.	S.D.
Linear, 10 TL	158cm	130cm	331cm	131cm
Linear, 20 TL	149cm	125cm	312cm	117cm
Linear, 30 TL	146cm	124cm	308cm	114cm
Linear, 40 TL	145cm	123cm	308cm	112cm
Exponential, 10 TL	159cm	126cm	336cm	135cm
Exponential, 20 TL	148cm	119cm	316cm	148cm
Exponential, 30 TL	145cm	115cm	311cm	119cm
Exponential, 40 TL	144cm	114cm	311cm	117cm
Power, 10 TL	148cm	122cm	313cm	119cm
Power, 20 TL	140cm	116cm	296cm	111cm
Power, 30 TL	137cm	114cm	293cm	108cm
Power, 40 TL	136cm	113cm	294cm	107cm
Poly2, 10 TL	172cm	129cm	351cm	218cm
Poly2, 20 TL	143cm	118cm	304cm	117cm
Poly2, 30 TL	137cm	115cm	294cm	110cm
Poly2, 40 TL	135cm	115cm	290cm	108cm
Poly3, 10 TL	324cm	137cm	422cm	2001cm
Poly3, 20 TL	155cm	118cm	316cm	211cm
Poly3, 30 TL	140cm	115cm	297cm	129cm
Poly3, 40 TL	135cm	114cm	289cm	110cm

the third degree polynomial model (and, to a lesser extend, of the second degree polynomial model) is due to huge outliers, as demonstrated by the cumulative distribution function (CDF) in Figure 5.4; the standard deviation of the third degree polynomial model is 11278 cm at 5 location, 372 cm at 10 locations, 63 cm at 15 locations and 18 cm at 20 locations respectively. This increased standard deviation affects the average much more than the mean. Please refer to table 5.2 for more details.

Figure 5.5 exemplarily contrasts the average errors from step 1 with the minimum error from step 2 and the composite minimum from step 3 for the exponential model. It demonstrates that a search for the optimal combination of TLs (step two) can yield substantial gains, particularly in the absence of many TLs. This general behavior is the same for other regression models.

Table 5.3 lists the minimum error for each access point based on an evaluation of all regression models and TL counts. While the polynomial models have a large standard deviation at lower TL counts, their flexibility appears to be an asset, provided that optimization processes are performed. Step 3 reduces the error by 4 cm to 131cm, compared to the best model at a TL count of 40.

Steps two and three are noticeable particularly at lower total TL counts. Note that this process does not depend on a high number of total TLs to choose the optimal combination of TLs from. The composite minimum is 133.37 cm for 10 locations, 132.25 cm for 20 and 132.04 cm for 30 locations respectively. Table 5.4 shows the improvement of the composite minimum over the static application of any given regression model for various TL counts, and therefore the final contribution of this approach. Since only

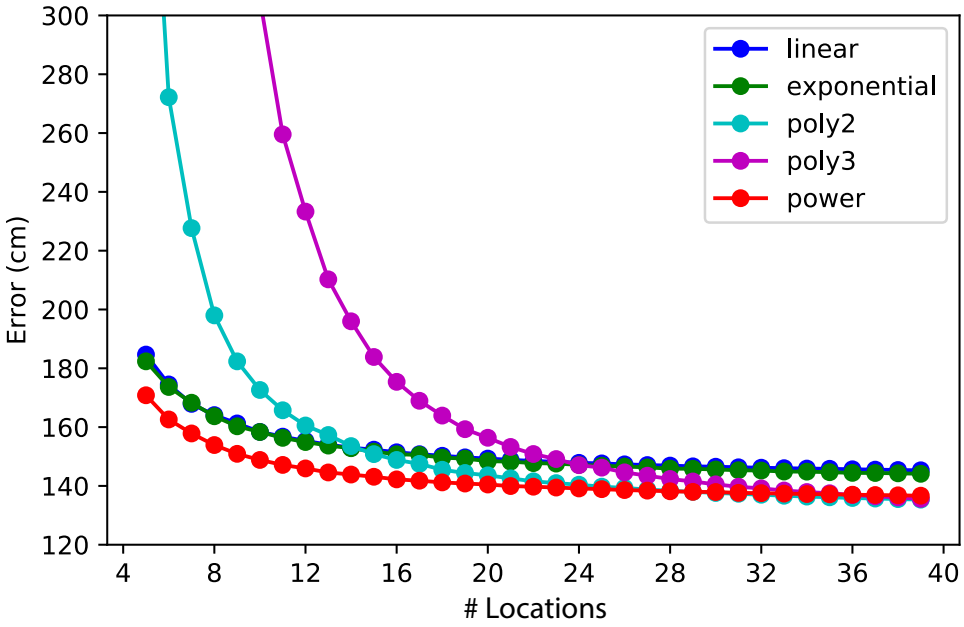

FIGURE 5.3: Average error vs. number of locations for each regression model.

TABLE 5.3: Composite minima

Access Point	Model	TLs	Minimum Value
1	Poly3	8	137.08 cm
2	Poly2	37	116.67 cm
3	Poly3	19	127.56 cm
4	Poly2	30	155.85 cm
5	Poly3	40	120.16 cm
6	Poly2	35	130.54 cm
Average			**131.31 cm**

TABLE 5.4: Improvements of composite minima over static model

TL Count	Linear	Exponential	Power	Poly2	Poly3
10	25cm	26cm	15cm	39cm	191cm
20	17cm	16cm	8cm	11cm	23cm
30	14cm	13cm	5cm	5cm	8cm
40	14cm	13cm	5cm	4cm	4cm

10 locations are required for almost optimal results, the training effort can substantially be minimized.

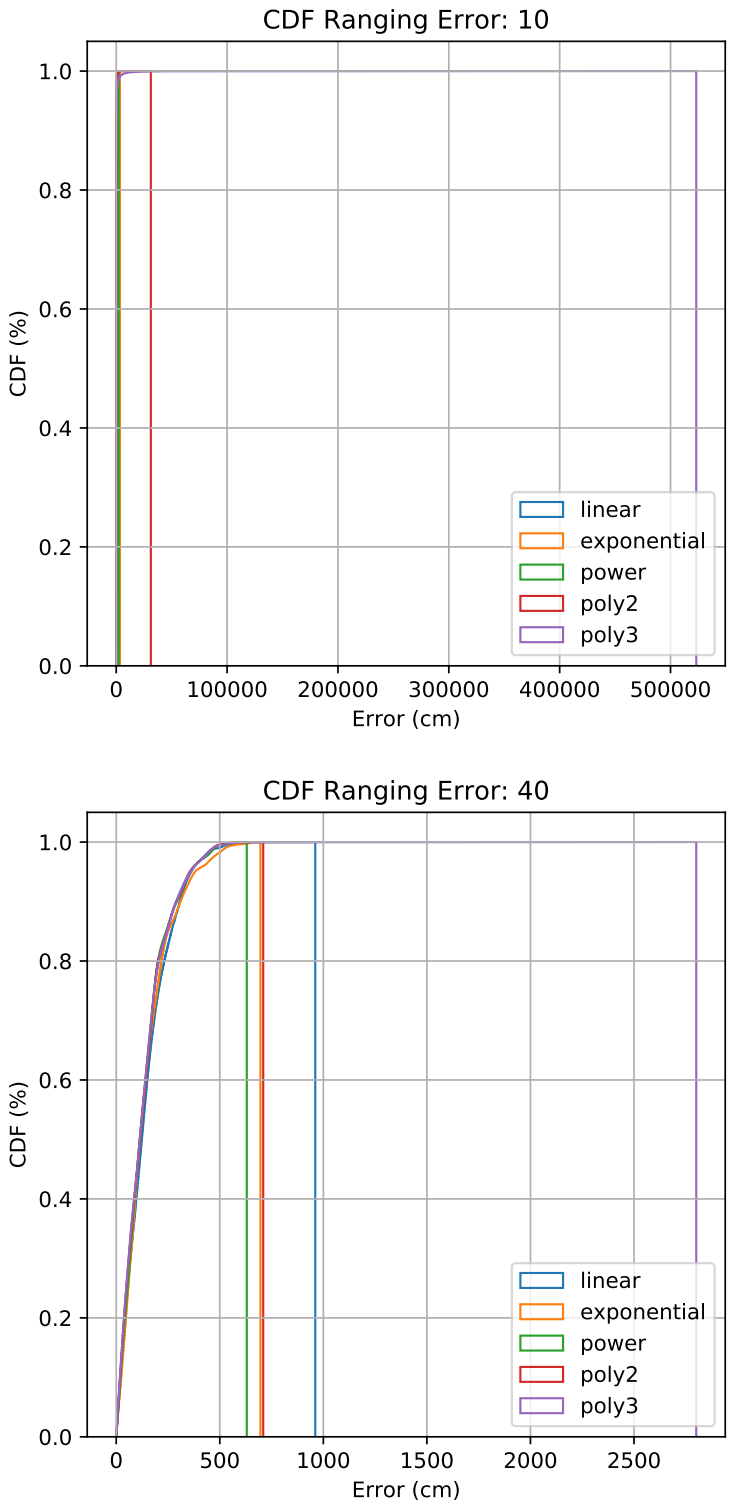

FIGURE 5.4: CDF for 10 and 40 TLs and all regression models. Note the varying scale of the error.

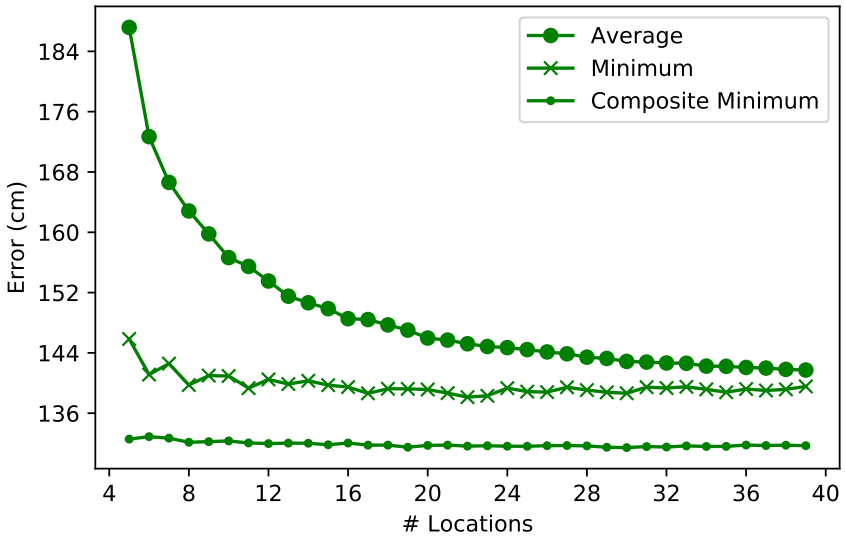

FIGURE 5.5: Average, minimum and composite error vs. number of locations for the exponential regression model.

5.2.3 Testing Points

To produce an error e, a set of TPs is required. Given that a fixed set of total locations is available, the question of differentiation between TLs and TPs arises, because the choice of TPs will ultimately affect the computation of the error e. To find an optimal set, three approaches were considered:

1. Use a dynamic set of TPs consisting of those locations not used as TLs.
2. Define a fixed (over all TL counts) set of TPs never to be used as TLs.
3. Always use all 48 locations as TPs.

The quality of a regression can only be evaluated by comparison with a set of known locations, which is the premise of all three options. This implies that no external yardstick is available to objectively compare these three options. Therefore, the expectation is that properties would emerge by simply applying every option, which would make a reasonable case for certain conclusions.

To find the optimal set of testing points (TP), experiments described above were run for all three configurations. The following reasons indicate that the third approach, using all 48 locations as TPs, is preferable:

- Whereas the minimum error e of approach 1 is 10cm smaller than approach 3 for the non-polynomial models at low TL counts, the gap increases to 27-30 cm at higher TL counts (see Figure 5.6), returning an absolute error of approximately 100 cm. These unrealistic characteristics confirm that it gets progressively easier to find combinations with minimum errors for ever decreasing amounts of TPs. Therefore, static selections of TPs are preferable.
- The experimental results for approach 2 and 3 are similar.

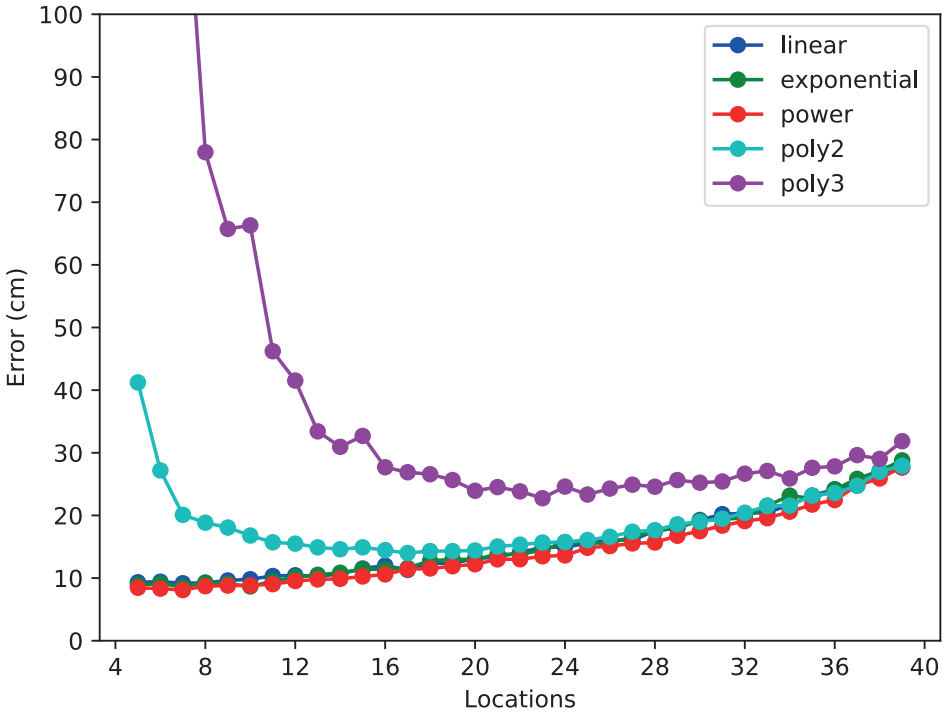

FIGURE 5.6: The error between a dynamic set of TPs not used as TLs and a static selection of TPs grows as the TL count increases.

- Approach 3 enables us to utilize all locations as TLs, whereas approach 2 is restricted by those not used as TPs. Having a bigger dataset increases its predictive powers. A higher number of TPs will always render a more complete assessment.
- The purpose of the TPs is in evaluating the regression model, not the TLs. Therefore, the argument that this process be responsible for predetermining the outcome is no more true than it would be when applied to the coefficient of determination r^2 to express the "fit" of a regression.

The results in this thesis thus rely on method 3, or all 48 TPs, for error assessment. While not completely without possible reproach, method 2 could also be criticized for presenting too incomplete an assessment. Gathering even more samples from additional locations would indeed mitigate this critique because of even more accurate results; it would do nothing, however, to remedy the fundamental problem of not having a flawless way to evaluate the regression models.

5.3 WiFi Sampling Rate

Every WiFi interface requires approximately 2.3 seconds to measure the entire WiFi landscape, which includes approximately 20 ANs. Because only 6 ANs are relevant, this translates to 383ms per AN. By contrast, measuring the signal strength of one individual AN takes around 130 ms. With one

sample consisting of six such readings, one ESP8266 requires 780 ms per sample and therefore provides 1.28 samples / sec. Consequently, the sampling rate of both ESP8266 is about 2.5 samples / sec. This is in agreement with the theory presented in 2.4.

Whereas the standard deviation for both interfaces individually was relatively small at 0.6 and 1.0 dBm, their averages differed by 3.44 dBm at -40.95 and -37.51 dBm, respectively. This data suggests a problem due to their offset positions or different calibrations. With both devices active concurrently, the average signal strength is -39.43 dBm and the deviation is 2.35 dBm. This average is only 0.2 dBm from the average of the individual results, at -39.23 dBm, and can therefore be explained by their individual differences; thus, there are likely no problems due to interference.

These tests could be resumed with more WiFi devices to gain a more complete picture. However, a difference of 3.44 dBm, coupled with the unpredictability of this behavior, suggests, that combining several WiFi controllers is generally no sensible method to achieve higher sampling rates. It follows, however, that by assigning each WiFi-interface to a specific and unchanging set of ANs, the process of regression would automatically correct for this difference.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we presented and evaluated four core aspects of a centralized, server-based indoor positioning system:

- Server performance.
- WebSocket as a protocol for real-time exchange of positioning data.
- Sampling accuracy when dynamically evaluating training data.
- WiFi ranging with multiple radio interfaces to enhance the sampling rate.

Server performance While current results will have to be adjusted based on the implementation of the rotation update and particle filter, they currently suggest that approximately *100,000 simultaneous clients* could be handled per core, which is many more than any other aspect of the system. With sophisticated load balancing, implemented in combination with Nginx, this can be scaled as required.

WebSocket as a protocol for real-time exchange of positioning data. WebSocket was chosen as a communication protocol, not least because of its low overhead in full-duplex communication. Given our average message size of 300 bytes and a target of 10 messages per second for real-time updates, it was concluded that the server could handle approximately *800 concurrent connections*. While this amounts to only 2.4 MB/sec in actual data, whether or not the WiFi network would be able to sustain this transfer rate for this many simultaneous connections is questionable. The WebSocket protocol fulfills its purpose; any potential bottleneck would likely be due to other factors.

Sampling accuracy when dynamically evaluating training data. A dynamic selection of both, training locations as well as AN specific ranging model was used to enhance the sampling accuracy. The result was an optimized average error of *131.31 cm*. Compared to the exponential ranging model, this constitutes an improvement of approximately 13 cm at 30 TLs, or 26 cm at 10 TL. Remarkably, the total amount of TLs does not fundamentally affect this result, resulting in an even larger improvement at low TL counts.

WiFi ranging with multiple radio interfaces to enhance the sampling rate. With one sample consisting of 6 individual, sequential readings to WiFi access points, one WiFi interface is capable of approximately *1.25 samples per second* in our testing environment. This could be multiplied by connecting multiple WiFi interfaces to the ESP32, up to a theoretical limit of about 10hz. However, experiments have shown that this will result in different RSSI values being returned from each interface, thus skewing the final data. Whether this is due to different sensor calibrations, or to their slightly offset position, is unclear. The benefit and scalability of this approach is therefore rendered questionable, unless a set attribution of WiFi-interfaces to ANs is possible.

6.2 Future Work

It seems reasonable to assume that more sophisticated machine learning algorithms might be better prepared to deal with the irregularities of multipath-effects than relatively simple regression models. This would also provide a basis for more advanced algorithms such as floor-detection, which simple regression is unable to provide.

Bibliography

[1] Krzysztof W. Kolodziej and Johan Hjelm. *Local Positioning Systems*. 1st ed. Taylor & Francis, May 2006.

[2] J. Song et al. "Improved indoor position estimation algorithm based on geo-magnetism intensity". In: *2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. Oct. 2014, pp. 741–744. DOI: 10.1109/IPIN.2014.7275555.

[3] Y. Itagaki, A. Suzuki, and T. Iyota. "Indoor positioning for moving objects using a hardware device with spread spectrum ultrasonic waves". In: *2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. Nov. 2012, pp. 1–6. DOI: 10.1109/IPIN.2012.6418850.

[4] Jose Luis V. Carrera et al. "Discriminative Learning-based Smartphone Indoor Localization". In: *CoRR* abs/1804.03961 (2018). arXiv: 1804.03961. URL: http://arxiv.org/abs/1804.03961.

[5] F. Gustafsson. "Particle filter theory and practice with positioning applications". In: *IEEE Aerospace and Electronic Systems Magazine* 25.7 (July 2010), pp. 53–82. ISSN: 0885-8985. DOI: 10.1109/MAES.2010.5546308.

[6] José Luis Carrera et al. "A Real-time Indoor Tracking System in Smartphones". In: *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (2016), pp. 292–301.

[7] *ESP32*. https://www.sparkfun.com/products/13907. Accessed June 8, 2018.

[8] *ESP8266*. https://www.espressif.com/en/products/hardware/esp8266ex/overview. Accessed June 8, 2018.

[9] V. Pimentel and B. G. Nickerson. "Communicating and Displaying Real-Time Data with WebSocket". In: *IEEE Internet Computing* 16.4 (July 2012), pp. 45–53. ISSN: 1089-7801. DOI: 10.1109/MIC.2012.64.

[10] *The Node.js Event Loop, Timers, and process.nextTick()*. https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick. Accessed August 28, 2018.

[11] L. P. Chitra and R. Satapathy. "Performance comparison and evaluation of Node.js and traditional web server (IIS)". In: *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*. Feb. 2017, pp. 1–4. DOI: 10.1109/ICAMMAET.2017.8186633.

[12] *I/O Concept – Blocking/Non-Blocking VS Sync/Async*. https://blogs.msdn.microsoft.com/csliu/2009/08/27/io-concept-blockingnon-blocking-vs-syncasync/. Accessed August 2, 2018.

[13] M.A. Iqbal, J.H. Saltz, and S.H. Bokhart. *Performance tradeoffs in static and dynamic load balancing strategies*. Mar. 1986.

[14] *GlobalInterpreterLock*. https://wiki.python.org/moin/GlobalInterpreterLock. Accessed August 7, 2018.

[15] *User Datagram Protocol*. https://tools.ietf.org/html/rfc768. Accessed August 26, 2018.

[16] *Transmission Control Protocol*. https://tools.ietf.org/html/rfc793. Accessed August 26, 2018.

[17] I. Fette and A. Melnikov. *The WebSocket Protocol*. Internet Engineering Task Force (IETF). Dec. 2011.

[18] *Wikipedia: Comparison of data serialization formats*. https://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats. Accessed August 26, 2018.

[19] *The JavaScript Object Notation (JSON) Data Interchange Format*. https://tools.ietf.org/html/rfc8259. Accessed August 26, 2018.

[20] J. Koo and H. Cha. "Localizing WiFi Access Points Using Signal Strength". In: *IEEE Communications Letters* 15.2 (Feb. 2011), pp. 187–189. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2011.121410.101379.

[21] C. Feng et al. "Compressive Sensing Based Positioning Using RSS of WLAN Access Points". In: *2010 Proceedings IEEE INFOCOM*. Mar. 2010, pp. 1–9. DOI: 10.1109/INFCOM.2010.5461981.

[22] *Wikipedia: Regression analysis*. https://en.wikipedia.org/wiki/Regression_analysis. Accessed August 5, 2018.

[23] *Wikipedia: Nonlinear regression*. https://en.wikipedia.org/wiki/Nonlinear_regression. Accessed August 5, 2018.

[24] *IEEE Std 802.11-2016*. http://standards.ieee.org/findstds/standard/802.11-2016.html. Accessed August 2, 2018.

[25] *Hampel Filter*. https://www.mathworks.com/help/dsp/ref/hampelfilter.html. Accessed August 5, 2018.

[26] William Stallings. *Data and Computer Communications*. 10th ed. Pearson Education Inc., 2014, pp. 404–412.

[27] *Wikipedia: List of WLAN channels*. https://en.wikipedia.org/wiki/List_of_WLAN_channels. Accessed August 3, 2018.

[28] D. Jain and S. C. Jain. "Load balancing real-time periodic task scheduling algorithm for multiprocessor enviornment". In: *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*. Mar. 2015, pp. 1–5. DOI: 10.1109/ICCPCT.2015.7159407.

[29] X. Gu, L. Yang, and S. Wu. "A real-time stream system based on node.js". In: *2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing(ICCWAMTIP)*. Dec. 2014, pp. 479–482. DOI: 10.1109/ICCWAMTIP.2014.7073454.

[30] L. P. Chitra and R. Satapathy. "Performance comparison and evaluation of Node.js and traditional web server (IIS)". In: *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*. Feb. 2017, pp. 1–4. DOI: 10.1109/ICAMMAET.2017.8186633.

[31] *HTML5 WebSocket: A Quantum Leap in Scalability for the Web*. https://websocket.org/quantum.html. Accessed September 11, 2018.

[32] Markku Laine and Kalle Säilä. *Performance Evaluation of XMPP on the Web*. Apr. 2012.

[33] Amir Almasi and Yohanes Kuma. *Evaluation of WebSocket Communication in Enterprise Architecture*. June 2013.

[34] Peter Lubbers, Frank Salim, and Brian Albers. "Pro HTML5 Programming". In: Apress, 2011, p. 137.

[35] S. Agarwal. "Real-time web application roadblock: Performance penalty of HTML sockets". In: *2012 IEEE International Conference on Communications (ICC)*. June 2012, pp. 1225–1229. DOI: 10.1109/ICC.2012.6364271.

[36] D. Skvorc, M. Horvat, and S. Srbljic. "Performance evaluation of Websocket protocol for implementation of full-duplex web streams". In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. May 2014, pp. 1003–1008. DOI: 10.1109/MIPRO.2014.6859715.

[37] H. Liu et al. "Survey of Wireless Indoor Positioning Techniques and Systems". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.6 (Nov. 2007), pp. 1067–1080. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2007.905750.

[38] Jeffrey Hightower, Gaetano Borriello, and Roy Want. "SpotON: An Indoor 3D Location Sensing Technology Based on RF Signal Strength". In: *UW CSE Technical Report #2000-02-02* (Feb. 2000).

[39] Lionel M. Ni et al. "LANDMARC: Indoor Location Sensing Using Active RFID". In: *Wireless Networks* 10.6 (Nov. 2004), pp. 701–710. ISSN: 1572-8196. DOI: 10.1023/B:WINE.0000044029.06344.dd. URL: https://doi.org/10.1023/B:WINE.0000044029.06344.dd.

[40] Veljo Otsason et al. "Accurate GSM Indoor Localization". In: *UbiComp 2005: Ubiquitous Computing*. Ed. by Michael Beigl et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 141–158. ISBN: 978-3-540-31941-2.

[41] S. Gezici et al. "Localization via ultra-wideband radios: a look at positioning aspects for future sensor networks". In: *IEEE Signal Processing Magazine* 22.4 (July 2005), pp. 70–84. ISSN: 1053-5888. DOI: 10.1109/MSP.2005.1458289.

[42] R. J. Fontana. "Recent system applications of short-pulse ultra-wideband (UWB) technology". In: *IEEE Transactions on Microwave Theory and Techniques* 52.9 (Sept. 2004), pp. 2087–2104. ISSN: 0018-9480. DOI: 10.1109/TMTT.2004.834186.

[43] J. Hallberg, M. Nilsson, and K. Synnes. "Positioning with Bluetooth". In: *10th International Conference on Telecommunications, 2003. ICT 2003.* Vol. 2. Feb. 2003, 954–958 vol.2. DOI: 10.1109/ICTEL.2003.1191568.

[44] A. Kotanen et al. "Experiments on local positioning with Bluetooth". In: *Proceedings ITCC 2003. International Conference on Information Technology: Coding and Computing*. Apr. 2003, pp. 297–303. DOI: 10.1109/ITCC.2003.1197544.

[45] M. Youssef and A. Agrawala. "The horus wlan location determination system". In: *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys05)*. 2005, pp. 205–218.

[46] Z. Li, T. Braun, and D. C. Dimitrova. "A time-based passive source localization system for narrow-band signal". In: *2015 IEEE International Conference on Communications (ICC)*. June 2015, pp. 4599–4605. DOI: 10.1109/ICC.2015.7249048.

[47] Hongbo Liu et al. "Push the Limit of WiFi Based Localization for Smartphones". In: *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*. Mobicom '12. ACM, 2012, pp. 305–316. ISBN: 978-1-4503-1159-5. DOI: 10.1145/2348543.2348581. URL: http://doi.acm.org/10.1145/2348543.2348581.

[48] S. Saha et al. "Location determination of a mobile device using IEEE 802.11b access point signals". In: *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.* Vol. 3. Mar. 2003, 1987–1992 vol.3. DOI: 10.1109/WCNC.2003.1200692.

[49] Zan Li, Torsten Braun, and Desislava C. Dimitrova. "A passive WiFi source localization system based on fine-grained power-based trilateration". In: *2015 IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. June 2015, pp. 1–9. DOI: 10.1109/WoWMoM.2015.7158147.

[50] Zhu Minghui and Zhang Huiqing. "Research on model of indoor distance measurement based on receiving signal strength". In: *2010 International Conference On Computer Design and Applications*. Vol. 5. June 2010, pp. V5–54–V5–58. DOI: 10.1109/ICCDA.2010.5540847.

[51] Kaibi Zhang, Yangchuan Zhang, and Subo Wan. "Research of RSSI indoor ranging algorithm based on Gaussian - Kalman linear filtering". In: *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. Oct. 2016, pp. 1628–1632. DOI: 10.1109/IMCEC.2016.7867493.

[52] Z. Li and T. Braun. "Passively Track WiFi Users With an Enhanced Particle Filter Using Power-Based Ranging". In: *IEEE Transactions on Wireless Communications* 16.11 (Nov. 2017), pp. 7305–7318. ISSN: 1536-1276. DOI: 10.1109/TWC.2017.2746598.

[53] E. Lopez-Aguilera, J. Casademont, and J. Cotrina. "IEEE 802.11g performance in presence of beacon control frames". In: *2004 IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE Cat. No.04TH8754)*. Vol. 1. Sept. 2004, 318–322 Vol.1. DOI: 10.1109/PIMRC.2004.1370886.

[54] *MPU9250*. https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250. Accessed September 13, 2018.

[55] *Bitbucket: Server Code*. https://bitbucket.org/indoorpositioning/localization. Accessed August 28, 2018.

[56] *Bitbucket: ESP32 Code*. https://bitbucket.org/indoorpositioning/esp32. Accessed August 28, 2018.

[57] *OpenStack*. https://www.openstack.org/. Accessed August 5, 2018.

[58] *Tornado Web Server*. http://www.tornadoweb.org. Accessed September 11, 2018.

[59] *Tornado: Running and Deploying*. http://www.tornadoweb.org/en/stable/guide/running.html. Accessed August 5, 2018.

[60] *Using nginx as HTTP load balancer*. http://nginx.org/en/docs/http/load_balancing.html. Accessed August 5, 2018.

[61] *The WebSocket Protocol*. https://tools.ietf.org/html/rfc6455. Accessed August 5, 2018.

[62] *PEP 418 – Add monotonic time, performance counter, and process time functions*. https://legacy.python.org/dev/peps/pep-0418/. Accessed August 3, 2018.

[63] *Github: Arduino core for the ESP32*. https://github.com/espressif/arduino-esp32. Accessed August 5, 2018.

[64] *Github: Websocket client for Arduino, with fast data send*. https://github.com/bhagman/Arduino-Websocket-Fast. Accessed August 5, 2018.

[65] *Github: A library for arduino to read value of MPU9250*. https://github.com/asukiaaa/MPU9250_asukiaaa. Accessed August 5, 2018.

[66] *ArduinoJson*. https://arduinojson.org/. Accessed August 5, 2018.

[67] *QueueList Library For Arduino*. https://playground.arduino.cc/Code/QueueList. Accessed August 5, 2018.

[68] *ESP8266 AT Instruction Set*. https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf. 2018 (accessed June 8, 2018).

[69] Fred Halsall. *Computer Networking and the Internet*. 5th ed. Pearson Education Limited, 2005, pp. 240–252.