UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Algoritmul EZ de Localizare în Spații Închise
Implementare și Evaluare

Željko Žužić

Coordonator științific:

Șl.dr.ing. Emilian Rădoi
Dr.ing. Valentin Radu

BUCUREȘTI

2018

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

EZ Algorithm for Indoor Localization
Implementation and Evaluation

Željko Žuržić

Thesis advisor:

Asst. Prof. Dr. Eng. Emilian Rădoi
Dr. Eng. Valentin Radu

BUCHAREST

2018

CONTENTS

SINOPSIS

Localizarea în spații interioare este crucială în unele situații din viața reală. Productivitatea unui depozit sau monitorizarea în domeniul medical sunt puternic influențate de acuratețea și simplitatea localizării unor noduri mobile. În această teză este descrisă și dezbătută o implementare a sistemului EZ [3]. Soluția prezentată folosește semnale ale punctelor de acces WiFi, o tehnologie foarte răspândită și accesibilă în prezent. În practică, punctele de acces dintr-un spațiu interior monitorizat își pot schimba poziționarea sau puncte de acces WiFi noi pot fi montate uneori. Un avantaj important al soluției implementate este lipsa efortului de pre-implementare, EZ având o abordare ce nu necesită informații preliminare despre punctele de acces. Mai mult, localizarea este realizată în mod transparent pentru utilizator, fără a necesita implicarea directă a acestuia. Un singur client mobil este necesar pentru a realiza localizarea în spații interioare folosind EZ. Teza ilustrează o implementare a soluției și rezultatele obținute în două medii de testare diferite - un apartament cu trei camere dintr-o zonă aglomerată și o clădire de birouri situată într-o zonă liniștită.

ABSTRACT

Indoor localization is crucial in some real-life situations. The productivity of a warehouse or healthcare monitoring are highly influenced by accurately and easy locating mobile nodes. An EZ system [3] implementation is described and debated in the current thesis. The presented solution uses WiFi access points signals, nowadays a very spread and accessible tehnology. In real-life situations, the access points from a tracked indoor space may change their position or even new access points may be deployed sometimes. A key advantage of the solution is the lack of pre-deployment effort, since EZ does not require prior information about the access points. Moreover, the localization is performed transparent for the user, without needing any participation from the users. A single mobile client is required for performing indoor localization with EZ. The current thesis illustrates an implementation of the solution and the results obtained in two different environments - a three-rooms flat from a crowded area and an office building situated in a peaceful area.

1 INTRODUCTION

In the last years, the interest for indoor localization encountered a significant growth in the academic world. The progress of industries such as robotics, healthcare, manufacturing sustained the need of localization in indoor spaces like warehouses, airports, hospitals or even homes. As a consequence, tracking assets, patients, personnel or robots‘ indoor navigation are crucial in some cases.

1.1 Motivation

The focus on the topic was also augmented by a technical challenge, since GPS localization proves to be inaccurate indoors. Hence other solutions using fit technologies such as WiFi, bluetooth, infrared, ultrasounds or GSM signals were proposed and analyzed by the scientific world involved in indoor localization.

1.2 Solution

This section stresses on the description of the indoor localization solution implemented and evaluated in the project. Many possible wireless technologies could have stayed at the basis of an indoor localization thesis, but we chose to focus on a solution which uses WiFi access points.

The current thesis presents an implementation and evaluation of EZ Algorithm for indoor localization proposed in [3], identifying and discussing not only the strengths and the use cases of the solution, but also its limitations. The system evaluation takes place in a three-rooms flat in a crowded area and in PRECIS building, situated in a peaceful area. The impact of some limitation factors is also discussed, proposing possible further improvements.

The EZ solution computes real-time indoor position of the tracked devices using WiFi access points. An important prerequisite is the presence of enough WiFi access points to ensure

the coverage in the whole indoor space where the localization takes place, so no matter the position in the indoor space, there are at least three visible access points. We may assume this condition is easily satisfied in the most uses cases.

The other two assumptions made by EZ are that each tracked item is equipped with or carries a device (smartphone, tablet) equipped with WiFi and that, from time to time, the system gets some fixed locations from the clients. The latter may be ensured by reporting GPS locations, when available: entrances, windows, near thin walls.

As other WiFi based indoor localization solutions, a huge advantage of EZ is that it does not require a special preparation of the tracked space or of the tracked nodes, since nowadays WiFi technology is spread enough to assume the indoors presence of enough visible access points and targeted devices equipped with WiFi.

The key feature of EZ in comparison to other systems based on the same technology is the absence of needing to know physical characteristics of the environment. EZ does not imply any information about the location, the transmit power or any other technical specification of the access points. This is a great benefit for EZ, since deploying new access points can happen very often and can be done by various entities in the proximity of any indoor space.

Moreover, in contrast to other prior work, EZ does not involve the presence of several nodes in localization. The solution is not dependent on the distance between the tracked items, so a single node is enough for both generating relevant data for the system and the localization process.

User participation in the whole procedure is also transparent. The client application is designed to collect data in the background, without asking for any position or technical input from the user.

On the other hand, all the advantages mentioned before come up with a significant trade-off: the EZ localization algorithm is less accurate in comparison to prior work when localizing a particular device. However, when localizing a new node, the system remains robust, whilst systems like RADAR [2] or Horus [9] significantly deteriorate their accuracy.

1.3 Summary

Chapter 2 of the thesis introduces prior related work in the field of indoor localization discussing their limitations in contrast to EZ.

Chapter 3 presents the technologies used in our implementation of the EZ system, describing the architecture of the client-side and the server-side applications.

Chapter 4 stresses on the implementation of the solution, the algorithms used and the mathematical and physical concepts underlying the implementation.

Chapter 5 reports the results after evaluating the performance of the algorithm and the system's validation.

Chapter 6 discusses possible further improvements of the system and presents the next steps of our research.

Chapter 7 closes the thesis emphasizing the conclusions of our work.

2 RELATED WORK

This chapter introduces prior work in the field, classifying the discussed indoor localization systems in three categories: solutions which use wireless communication technologies other than WiFi, solutions using WiFi which need prior physical details of the space where the localization takes place and solutions which need more than one node in order to perform the localization.

2.1 Systems Using Communication Technologies besides WiFi

This section presents some indoor localization solutions based on wireless technologies, other than using WiFi access points‘ signals.

2.1.1 Infrared Systems

Some indoor localization solutions rely on infrared technology. This kind of systems use infrared beacons mounted on the tracked nodes which transmit signals to some fix, with known location infrared receptors - as stereo-video cameras. Such a solution is represented by IRIS-LPS [1].

2.1.2 Bluetooth Systems

There are also several indoor tracking systems which use bluetooth tags to transmit and receive signals and then, according to their strength, convert them into distances. An example of a bluetooth-based indoor localization system is BLPA [5], which uses an extended kalman filter to approximate the location of the tracked node. Hybrid bluetooth-infrared indoor localization applications have also been developed - Topaz [7] implies transmission of bluetooth signals from the mobile devices to the bluetooth servers which are interconnected in the same LAN

with another server, where the localization algorithm is running. For error detection and correction, Topaz uses a similar infrared signal transmission-reception mechanism.

2.1.3 Ultra wideband Systems

The usage of radio signals on UWB is also a popular wireless approach in indoor localization. A good performing UWB system is proposed and analyzed in [10]. This solution resembles the GPS localization system, but in this case there are some mobile transmitting tags with some fixed base stations, working as receivers. As increasing the number of satellites improves the GPS accuracy, so does the increase of the number of base stations in the proposed UWB-based solution in [10].

The accuracy of some infrared, bluetooth or ultra wideband signal transmitting indoor localization systems may reach centimeters. However, they imply a perceptible pre-deployment effort in terms of cost and implementation.

2.2 Systems Needing Prior Context Information

This section presents solutions which use WiFi signals, but need prior physical details of the space where the localization takes place or prior information about the access points.

2.2.1 Systems Building Radio Maps

A state of the art indoor localization solution is Horus[3]. The system is designed to work in two different phases: an offline phase, when a RSS map is build by associating each possible location to the signal strength of each visible access point and an online phase, when user queries are solved based on the live strength of each signal received from the access points and the map build during the offline phase. Horus proved to have a good accuracy. On the other hand, the trade-offs of the systems are the pre-deployment effort of building a RF map and the loss of accuracy when using different devices as nodes to be located indoors.

2.2.2 Systems Needing Prior Information about Access Points

TIX [4] is a good example of an indoor localization system which needs prior information about the access points. In this solution, a specialized server provides information about the location of the access points and their signal strength, measured by each access point implied in the localization.

Systems which need prior context information as the location of access points or their signal strength may not be a fit solution in the indoor spaces where many entities can often deploy new access points, for example in office buildings or shopping malls.

2.3 Ad-hoc systems

Ad-hoc indoor localization systems imply communication between the tracked nodes. Some of these solutions build graphs where nodes‘ location may be known from before. An ad-hoc localization system which uses geometrical approaches where the tracked items can estimate the distance to other nodes is SpotON [6]. Besides needing more than one client in order to perform the localization, another disadvantage of SpotON is that its accuracy is strongly dependent on the cluster size.

There is a variety of possibilities when talking about indoor localization architectures or used technologies, each of them relying on a different trade-off.

EZ localization proposed in [3] comes up with some key-advantages in comparison to prior related work. Although the solution trades-off in terms of accuracy, it does not need pre-deployment of specialized infrastructure. Moreover, it does not require prior information about the location or technical specifications of the access points, neither the simultaneous presence of more than one node in order to perform the localization.

3 ARCHITECTURAL OVERVIEW

The current chapter gives an overview of the software technologies used in our implementation and illustrates the solution's workflow. It also discusses the modules and the classes used both in the client-side and server-side applications.

3.1 Technologies Used

In our implementation of EZ localization system, various technologies were used depending on their fitness and purposes.

As mentioned before, the EZ solution implies being in an indoor environment with enough WiFi access points. When available, EZ also receives GPS locations.

The client application runs on Android OS [8] devices and is implemented in Android Studio, since according to a study in regard of the second quarter of 2016, mobile devices which ran Android OS had a worldwide market share of 86.2% [1].

The communication between the client-side application and the server-side application is established using TCP/IP sockets, as TCP communication protocol is stream-based, reliable and connection-oriented, fit for our scope: to send WiFi and GPS fingerprints collected on the client application to the application running on a server in the same LAN.

Python was chosen as the programming language for the implementation of the server-side application since it offered us both standard libraries and mathematic-related libraries useful for our computations. The oriented-object support was another influencing factor of our choice. The readability of the code syntax was also a key argument for Python, as it would be simple to improve the implementation by further work.

For storing the data collected by the clients, the server-side application uses a SQLite database. The choice was motivated by the cross-platform file format of the database, the low memory

[1] https://android.jlelse.eu/apple-vs-android-a-comparative-study-2017-c5799a0a1683

consumption - since the collected data can be very large and the availability of sqlite3 library in Python, which made the interaction with the database easier.

3.2 System Workflow

A mobile device situated in an indoor space collects data from all the visible access points - MAC addresses, received signal strengths and GPS locations - when available and sends this fingerprints to the EZServer. The server queries a SQLite database in order to get the available data about the access points - location, transmit power, path loss exponent and then uses trilateration to estimate the client's position. Further mathematical and physical proofs will be discussed in the implementation chapter.

Before being able to answer to location queries, the server should compute a model for the relevant access points. To ensure this computation, the system needs to collect data from the environment - WiFi signals and GPS locations in a training phase.

The database with the collected data is also updated while serving clients with real-time estimated positions. The algorithm which estimates the relevant access points specifications and locations should be run once in a while, because the location of the old access points may change or new access points may be deployed in the area. This procedure should not always run on the server because, depending on the data set, on an average machine the computation may last for several hours.

The whole system's working flow is illustrated in Figure 1.

3.3 Client Application Overview

The client application is implemented in Android Studio using Java. In the MainActivity, the user may press the start button when he wants to start collecting data/using the localization system and the stop button when he wants to disconnect. The onCreate() method call sets the associated click listeners for the buttons.

When the start button is pressed, its click listener is triggered: the Scanner service is started and two broadcast receivers are created and registered (if not already being instantiated before)

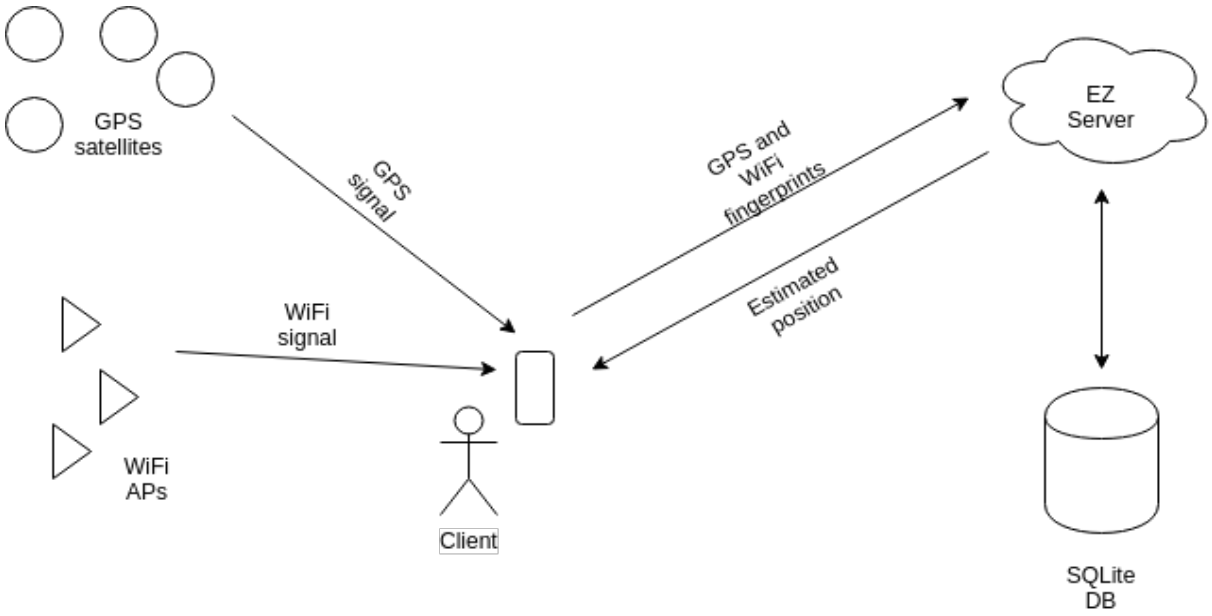

Figure 1: System Workflow

- one for receiving intents associated with GPS fingerprints and the other one for receiving WiFi fingerprints.

The ScannerService implements the interface IOnWifiDataCallback. This service uses a locationManager and a locationListener to request GPS locations. After computing a mean location (using an algorithm which will be explained in the next section), the ScannerService creates a GPSFingerprint and passes it to an intent which is broadcast by the service and received by the gpsBroadcastReceiver from MainActivity.

On the onCreate() method the ScannerService also instantiates a WifiScanner, which searches for the visible access points and collects their signal strengths into a WifiFingerprint (by some rules also explained in the next section). After a WifiFingerprint is constructed, the onWifiSample() callback is called and passes the fingerprint to an intent, being broadcast from the service and received by wifiBroadcastReceiver, also in MainActivity.

When one of the two broadcast receivers gets an intent, it launches a separate MessageSender, which is an AsyncTask responsible for sending collected data to the server - WifiFingerprints or GPSFingerprints.

A WifiFingerprint consists of more WifiRecords and a timestamp. A WifiRecord is given by an access point and the its signal strength. GpsFingeprints are given by a GPS collected location and a timestamp. Both fingerprints are served in JSON format to facilitate the data

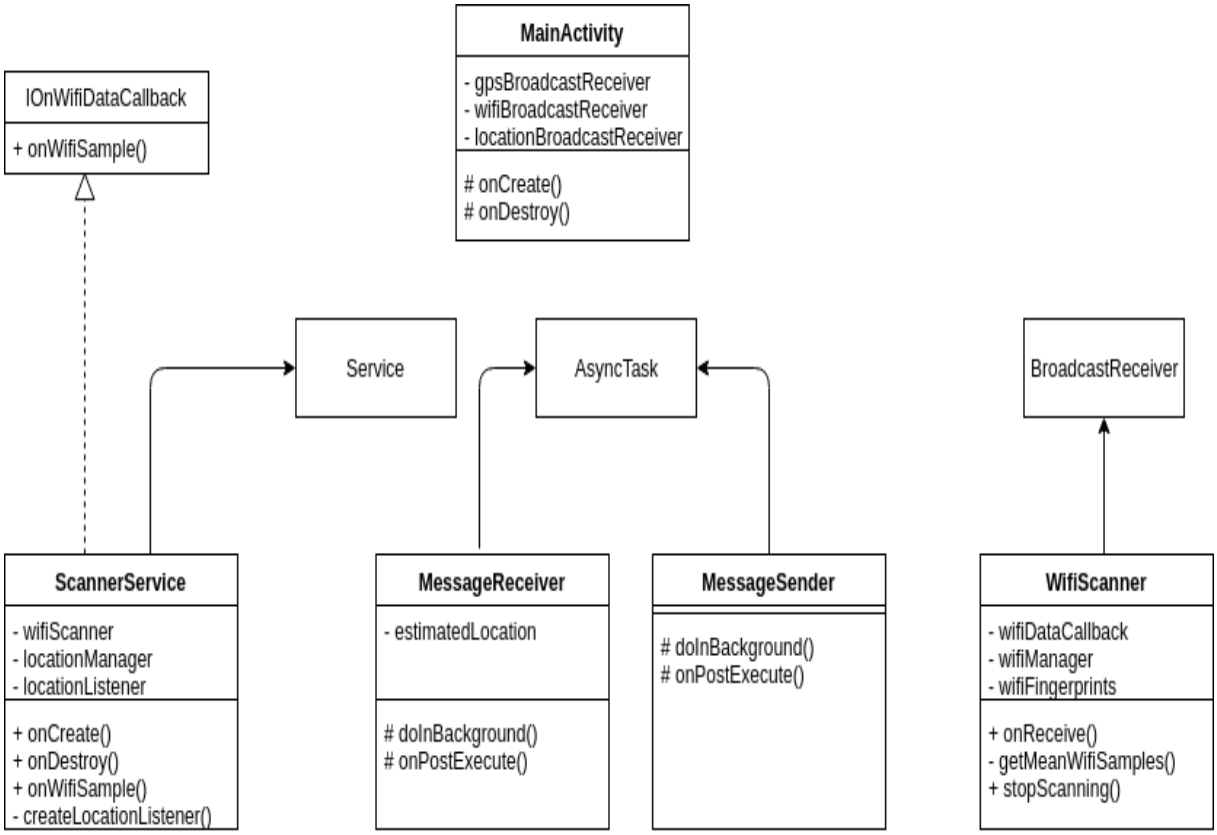

Figure 2: Client Application UML Diagram

processing on the server side.

The MessageSenderAsyncTask uses TCP/IP sockets for the communication with the server, sending both the content of the fingerprint and the type of the sent message - WiFi or GPS.

The MessageReceiver async task waits in the background for a message from the EZServer, consisting of the estimated current location. After receiving the coordinates, it makes an intent with the data which is broadcast and then received in the MainActivity which gives the location feedback to the user.

A diagram of the client application structure can be consulted in Figure 2.

3.4 Server Application Overview

On the server side, the system consists of two Python modules: locationServer.py and computeAPParam.py.

The locationServer module creates a TCP/IP socket and listens for client connections on port

8888. After the connection is established, two messages are received: the type of the content (GPS/WiFi) and the concrete content. Depending on the message type, two different objects may be created, based on a JSON decoder: WifiScan or GpsScan.

A WifiScan consists of the client who sent the fingerprint, a timestamp and a list of the access points - wifiAPList which were fingerprinted at the respective timestamp. The wifiAPList is composed of WifiAP objects, each of them encapsulating the MAC address (bssId) and the strength of the received signal (signal) for the access points which were available when the fingerprint was taken. A GpsScan encapsulates the client who got the GPS location, a timestamp and the coordinates of the location - latitude and longitude. All the details from the scans are decoded from the received JSON format message using a jsonDecoder available through json library.

Two SQL tables (Figure 3) are created to store the two types of information - wifi_samples for WifiScan objects and gps_samples for GpsScan objects. The locationServer calls a addToDB function to insert records in the corresponding table. A wifi_samples record contains details about an access point - its bssId, signal strength and the timestamp when the signal was captured. A gps_samples record encapsulates information regarding the GPS location coordinates -latitude and longitude and the associated timestamp of the collected GPS signal.

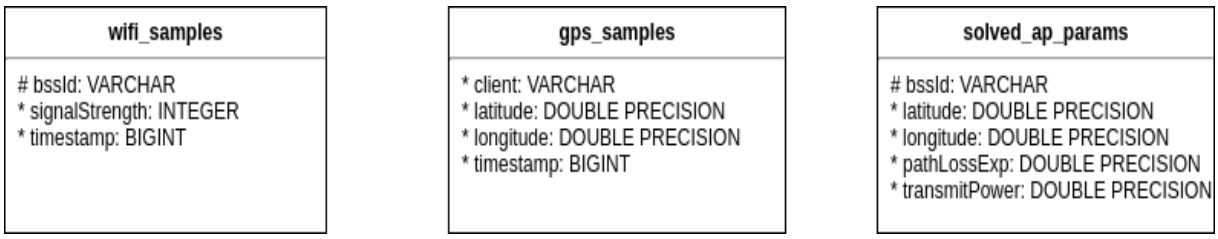

Figure 3: SQL Tables

On a separate execution thread, the locationServer interrogates the solved_ap_params table (Figure 3) to get the necessary information to compute the client's location. The relevant records for the estimation of the position are the ones with an access point which was also available for the client and sent in the query. For each visible access point, the path loss exponent and the transmit power are extracted from the record which, in combination with the signal strength of the respective access point received by the user, make the computation of the distance between the client and the access point possible (in a manner which will be described in the next section). When all the distances between the client and his visible access

points are computed, the client's estimated position is calculated using trilateration and served to the client application.

In order to populate the solved_ap_params table, the computeAPParam module should be run. The program forms two lists: one of the received observations and one containing the relevant access points for computation, with data accessed from wifi_samples and gps_samples tables (in a manner presented in the Implementation chapter). The observation list contains DeviceObservation objects - the timestamp of the observation, the initial supposed coordinates - generated random or GPS-based, a flag indicating whether the coordinates where GPS collected and a list of APFingerPrints. An APFingerprint gives information about each received signal strength of the visible access points for the corresponding observation.

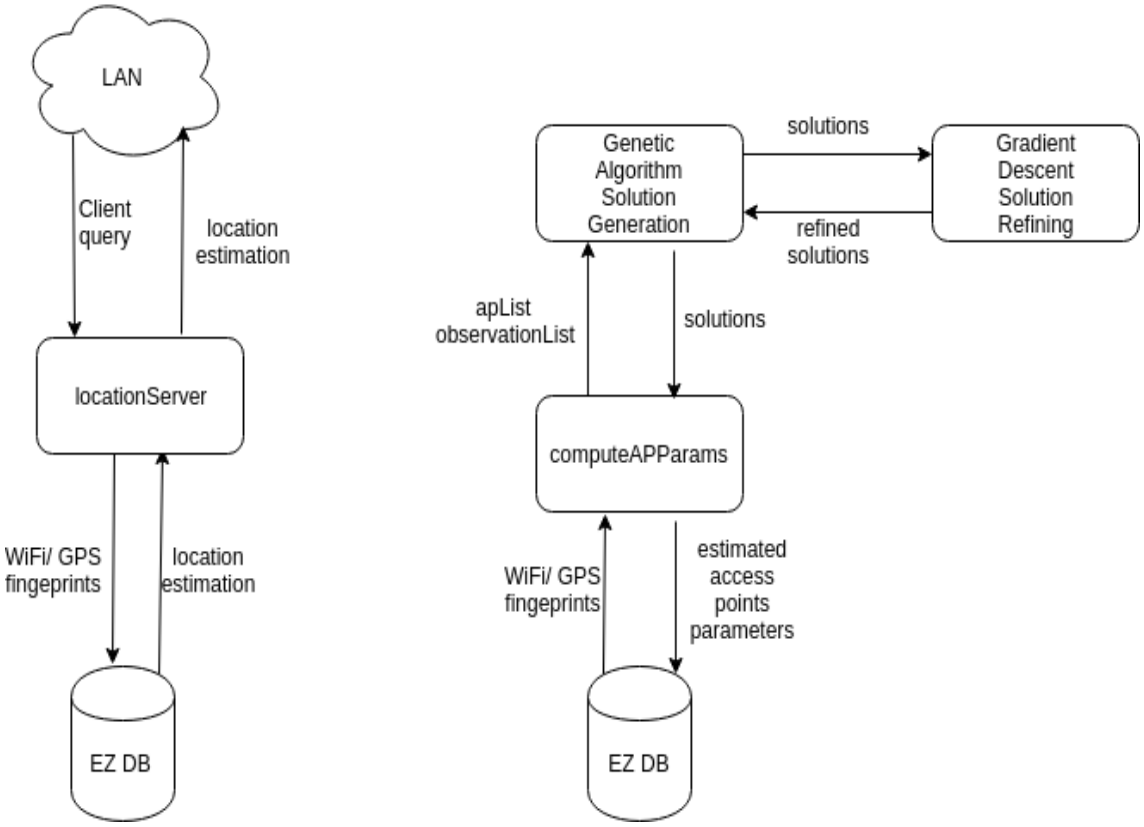

Figure 4: Server Application Workflow

Based on this two formed lists, a genetic algorithm computes possible solutions for the access points parameters. Each generated Solution has a fitness inversely proportional with last mean absolute error. Besides the fitness score, a Solution also contains the observationList and the apList associated with the score computation. During the genetic algorithm, some solutions are refined using the gradient descent method, in order to minimize the last mean

absolute error. After passing ten consecutive generations without any improvement, the genetic algorithm stops and the solved_ap_params table is updated. All this process will be explained in detail in the next chapter - Figure 4.

The classes used for modeling the data on the server-side application for the computeAPParam program are written in a separate module - models.py, whereas the help function in the process can be found in the utilities.py module. The gradient descent algorithm is written in a separate module, grad.py.

Since the genetic algorithm may take several hours to compute the relevant data, it is recommended to be rarely run. Still, this should not affect the performance of the system, assuming that the configuration of the access points would not change often.

4 IMPLEMENTATION

This chapter presents the mathematical and physical basis of the used algorithm. During this chapter, the algorithms used for both collecting data on the client-side application and the server-side clients‘ location estimation/access points‘ parameters estimation are described in detail.

4.1 Theoretical Basis

The current section gives a short overview of the theoretical mathematical and physical concepts applied in our implementation.

4.1.1 Trilateration

Trilateration represents the process of determining an unknown position by knowing the distance between the respective point and other known locations. For the purpose of the current thesis only 2D trilateration will be explained.

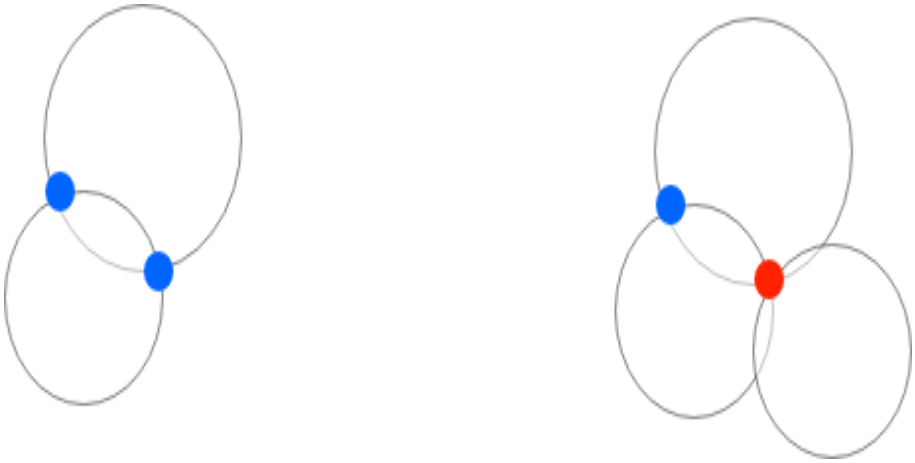

Figure 5: Triangulation

If a single distance x is available, the unknown location can be anywhere on a circle of radius

x. With an additional known distance *y*, the unknown position can also be anywhere on a circle of radius *y*, so the location we are searching for is at the intersection of the two formed circles - see the blue dots in Figure 5. Thus, we have two possible solutions. In order to uniquely determine the searched coordinates, a third known distance *z* is enough. Applying the same principle as we did before, our unknown position is given by the coordinates of the intersection of the three circles.

4.1.2 LDPL equations

A key theoretical model for the computation of the access points' parameters are the log-distance path loss (LDPL) equations. This model predicts the path loss of a signal in an indoor space. The formal description of the model is illustrated by Equation 1, where PL represents the total path loss (measured in db), $P_{Tx_{dBm}}$ the transmitted power (in dBm), $P_{Rx_{dBm}}$ the received power (in dBm), PL_0 the path loss considered at the distance d_0, d the length of the signal's path, γ the path loss exponent and X_g a gaussian random variable which describes the attenuation of the signal.

$$PL = P_{Tx_{dBm}} - P_{Rx_{dBm}} = PL_0 + 10\gamma log_{10}\frac{d}{d_0} + X_g \tag{1}$$

Assume we have m visible access points and n mobile tracked nodes. Based on the Pythagorean theorem, the distance between the i-th access point and the j-th mobile node is given by Equation 2.

$$d_{ij} = \sqrt{(i_{latitude} - j_{latitude})^2 + (i_{longitude} - j_{longitude})^2} \tag{2}$$

In the same time, based on the LDPL model described by Equation 1, the distance between the i-th access point and the j-th tracked client may also be computed as illustrated in Equation 3, where P_{i0} is the transmit power of the i-th access point at 1m distance, p_{ij} is the strength of the WiFi signal transmitted by the i-th access point and received by the j-th tracked client node and γ_i is the path loss exponent illustrating the received signal strength (RSS) falls in the proximity of the i-th access point.

$$d_{ij} = 10^{(\frac{P_{i0} - p_{ij}}{10\gamma_i})} \tag{3}$$

4.1.3 Equation Solving

The out-of-the box approach proposed for EZ implementation [3] in comparison with other indoor localization solutions is to consider not only the coordinates of the access points unknown of the equations, but also P_0 and γ.

Using trilateration and supposing the distances between the access points and the tracked nodes known, three RSS measurements at known locations would be enough to uniquely determine an access point's location - Equation 2. As EZ considers P_0 and γ also as unknown parameters, d_{ij} also becomes unknown. As a consequence, two additional equations are needed in order to solve the equation system. As a result, five RSS measurements are needed to achieve the parameters unique solution for a single access point.

As collinear locations cannot be used in trilateration to determine a location, LDPL equations cannot be solved to uniquely determine P_0 and γ if the observations were taken at the same distance from the access point. To ensure that, observations which record the same signal strength value for signals received from the same access point should be discarded.

When knowing all the visible access points' parameters, a client's location query is served using trilateration, where p_{ij} is taken from the received fingerprint and the other parameters of the equations (besides $j_{latitude}$ and $j_{longitude}$) are known.

4.1.4 Gradient Descent

Gradient descent is a very popular method in artificial intelligence and machine learning. For our purpose, the application of this method stresses on finding the local minimum of a function by using the first derivative. Finding a local minimum for J_{EZ}, we minimize the error of the solution - see Equation 7.

The gradient descent initializes the unknown variables with random values and then refines them in the sense of minimizing the value of the function. To do so, the method subtracts the value of the partial derivative with respect to the involved variable multiplied with the learning rate from its initial value. This operations repeats for each random generated value which needs to be modified - see Equation 4.

$$x = x - \alpha * \frac{\delta f}{\delta x} \tag{4}$$

The learning rate α should be carefully picked. If it is too small, the gradient descent converges too slow. If it is too big, the method may miss the local minimum.

The algorithm ends when the euclidian norm of the gradient vector - see Equations 5 - multiplied by the learning rate is lower or equal to a fixed value ϵ.

$$\nabla f(x, y) = [\frac{\delta f}{\delta x} \frac{\delta f}{\delta y}] \tag{5}$$

4.2 Client Implementation

The previous chapter presented an architectural overview of the client application. In this section a detailed implementation will be described, focusing on the logics of the data collection.

Algorithm 1 ScannerService Collect Fingerprint Data

```
1: requestForGPSFingerprint()
2: requestForWifiFingerprint()
```

As mentioned in the previous chapter, a ScannerService runs in the background. When the service is created, it starts for requesting GPS location updates and scanning for WiFi signals - Algorithm 1. The algorithm which define the GPSFingerprint creation is implemented when the location listener detects a location change, whereas the logic for forming a new WifiFingerprint is called when the WifiScanner receives a new WiFi signal, triggering a callback function which sends an intent to the wifiBroadcastReceiver. The locations received from the GPS satellites which are less accurate than the accuracy limit of 10 meters are discarded. After taking 10 consecutive GPS samples, a mean location is computed and a new GPSFingerprint is created. A new GPS sampling is considered to have started if the received locations become inaccurate - we assume that the tracked node does not receive GPS signals anymore as moving indoors. The values for the accuracy limit and the number of GPS samples per fingerprint are empirically determined - Algorithm 2.

During a WiFi sampling, a list containing the scan results of the signals received is formed.

Algorithm 2 GPSFingerprint Computation

```
1: procedure requestForGPSFingerprint
2:     if isGPSAvail() then
3:         location = getGPSLocation()
4:         if location.accuracy < ACCURACY_LIMIT then
5:             discard location
6:             counter = 0
7:         else
8:             keepForMeanLocation(location)
9:             counter = counter + 1
10:        if counter == GPS_NO_OF_SAMPLES then
11:            counter = 0
12:            meanLocation = computeMeanLocation()
13:            gpsFingerprint = getGPSFingerprint(meanLocation)
14:            sendGPSIntent(gpsFingerprint)
```

After the WiFi scan interval is exceeded - 3 seconds, a WifiFingerprint is created using the scan results from the current sample. In this sense, the RSS from the visible access points in the WiFi scan interval are recorded - each only once, with the condition of having the standard deviation lower than the standard deviation threshold of 10dB - see Algorithm 3. The standard deviation measures how much does the RSS vary from the average RSS from the same access point during the WiFi scan interval - computed dividing the sum of the squares of differences between the RSS from each scan and the mean RSS by the number of scans(Equation 6). The WiFi scan interval is empirically determined, whereas the deviation threshold's value is the recommended in [3].

$$SD = \sqrt{\frac{\sum_{k=1}^{N}(RSS_k - RSS_{mean})^2}{N}} \tag{6}$$

A GPSFingerprint is sent through an intent to MainActivity directly from the ScannerService when computed. The WifiFingerprints captured by the WifiScanner trigger a callback from the ScannerService when ready. This callback sends the fingerprint to MainActivity. Both intents are broadcast and received in MainActivity by gpsBroadcastReceiver or wifiBroadcastReceiver,

according to the fingerprint's type. When an intent is received, a new MessageSender is run asynchronously.

The MessageSender creates a TCP/IP socket to communicate with the host on port 8888. Firstly, it sends the message type (GPS/WiFi, depending on the fingerprint category) to the server and waits for an ACK response. Finally, it sends the content of the fingerprint to the server. After finishing the data transmission, a MessageSender instance marks a static flag to true, in order to permit other waiting instances to have access to the communication socket.

Another TCP/IP socket is opened by the MessageReceiver, to get the estimated location from the host on port 8889. In an infinite loop, when receiving a location from the server, the MessageReceiver makes an intent with it which is then got in MainActivity by locationBroacastReceiver and displayed to the user.

Algorithm 3 WifiFingerprint Computation

```
1: procedure requestForWifiFingerprint
2:     wifiSampleList = scanForWifiSamples(SAMPLING_TIME_INTERVAL)
3:     discardSamplesWithHighDeviation(wifiSampleList, DEVIATION_LIMIT)
4:     wifiFingerprint = computeWifiFingerprint(wifiSampleList)
5:     sendWifiIntent(wifiFingerprint)
```

4.3 Server Implementation

This section presents in detail the logic and the algorithms used in the implementation of the server-side application. As mentioned before, the server-side application is split into two different programs: the locationServer module which continuously runs in order to respond to the clients' location queries and the computeAPParams module which should be run from time to time in order to update the visible access points' characteristics. In the current section, the two programs will be separately presented.

The locationServer creates a TCP/IP socket to open a communication channel on the port 8888 with the clients. Before listening for connections, the necessary SQL tables are created - wifi_samples, gps_samples and solved_ap_params (if they do not already exist) - see Figure 3. In an infinite loop, the server accepts connections from clients, treating them on separate execution threads.

From each client, the type of the message (GPS/WiFi) is firstly received. The program sends then an ACK message, to announce it is ready to get the fingerprint. If the solved_ap_params has no data (there are no computed access point characteristics), the server announces the client the request could not be solved due to lack of data. Otherwise, the locationServer computes the distances between the client and the access points from the fingerprint based on the RSS observations and the available data from solved_ap_params table - see Equation 3. Finally, using trilateration, the client's latitude and longitude are solved and the location is served to the user.

After serving the query of the client, the locationServer also updates the information about the access points received from the observations. In this sense, a WifiScan object or a GpsScan object are created, depending on the fingerprint type received from the client and the two associated SQL tables - wifi_samples and gps_samples are updated. For details regarding the records from the SQL tables and the data encapsulated by the two objects, see the previous chapter.

The computeAPParams program first needs to extract the relevant data from the SQL tables and form two lists - an observationList and a list of the access points - allAPList. For each record from wifi_samples, the MAC address of the access point, the RSS and the timestamp of the observation are collected. Traversing the observationList, if there is already an observation with the same timestamp, the observation is updated by adding a fingerprint with the MAC address of the access point and the RSS. If the extracted access point does not exist in the allAPList, it is added, otherwise its number of observations is incremented. If the observation does not already exist in the observationList, the gps_samples table is searched to see if there is a recording in the same place interval - a GPS fingerprint taken a second before or a second after the WiFi fingerprint. If there is the case, the observation's coordinates are taken from the respective GPS location, otherwise they are randomly generated and then the observation is added to the list. After inspecting the two database tables to form the two lists, they are filtered in order to keep only the access points which were observed for at least 5 times.

The genetic algorithm - see Algorithm 4 - proposed and described in [3] is then used in order to compute the access points parameters: the coordinates, P_0 and γ. Each generation is made of 100 solutions. The algorithm stops when, for 10 consecutive generations, the best 10 solutions do not improve. A solution is considered to be better than another solution if it has

a greater fitness score. The fitness score is considered to be $\frac{1}{J_{EZ}}$, where J_{EZ} represents the last mean absolute error after solving the LDPL associated system of equations - see Equation 7.

$$J_{EZ} = \sum_{i=1}^{n} \sum_{j=1}^{m} |P_{ij} - P_{i0} + 10\gamma_i \log_{10} d_{ij}| \tag{7}$$

Algorithm 4 Compute Access Points Parameters

```
1:  procedure GENETICALGORITHM
2:      initialSolutionGeneration = generateRandomSolutions()
3:      gradientDescentRefine(initialSolutionGeneration)
4:      computeFitness(initialSolutionGeneration)
5:      betterGeneration = True
6:      oldGeneration = initialGeneration
7:      while betterGeneration do
8:          bestOldGeneration = getBestTenPercentSolutions(oldGeneration)
9:          newGeneration = []
10:         newGeneration.append(bestOldGeneration)
11:         newGeneration.append(randomGenerateTenPercentSolutions())
12:         sixtyPercentSolutions = combineTwoOldSolutions(oldGeneration)
13:         gradientDescentRefine(sixtyPercentSolutions)
14:         newGeneration.append(sixtyPercentSolutions)
15:         twentyPercentSolutions = perturbeSolutions(oldGeneration)
16:         gradientDescentRefine(twentyPercentSolutions)
17:         newGeneration.append(twentyPercentSolutions)
18:         if not newGeneration has better solution than the bestOldGeneration then
19:             betterGeneration = False
20:         oldGeneration = newGeneration
21:     update wifi_ap_params table with the best solution
```

An initial generation of solutions is randomly generated, being refined by applying the gradient descent method. Each next generation evolves from the previous one.

10% of a generation consists of the solutions which have the best fitness score from its

ancestor.

Other 10% of the solutions are randomly generated.

The majority of a generation (60%) is given by merging two randomly picked solution from the previous generation in a random convex linear combination, this solution being then refined using the gradient descent method - see Equation 8, where *a* represents a randomly generated vector with elements in (0, 1), the number of the elements from the vector being equal to the number of the unknowns to be solved, *4m + 2n* and $\mathbf{1}$ is a vector with *4m + 2n* elements, all of them equal to 1.

$$S_{current} = a \cdot S_{1previous} + (\mathbf{1} - a) \cdot S_{2previous} \tag{8}$$

The remaining 20% solutions from a generation are formed by picking random values from an exponential distribution for each unknown variable and adding/subtracting them to the values from a randomly picked solution from the previous generation. This is done for allowing large perturbations of the generated solutions. When generating random values for the access points parameters, they picked in a relevant interval - assuming the conventional possible values: P_0 may be between -50dB and 0dB, γ may be between 1.5 and 6 and the coordinates should be the tracked indoor space's limits, in order to make the algorithm converge faster.

5 IMPLEMENTATION EVALUATION

This chapter presents the evaluation of the current thesis EZ solution implementation discussing the differences between the current results and the state-of-art implementation [3]. The possible factors involved in the implementations' different results are also discussed. In our evaluation we tested the system in two unrelated environments - a three-rooms flat situated in a crowded area and an office building (PRECIS) - Figure 6, placed in a peaceful area.

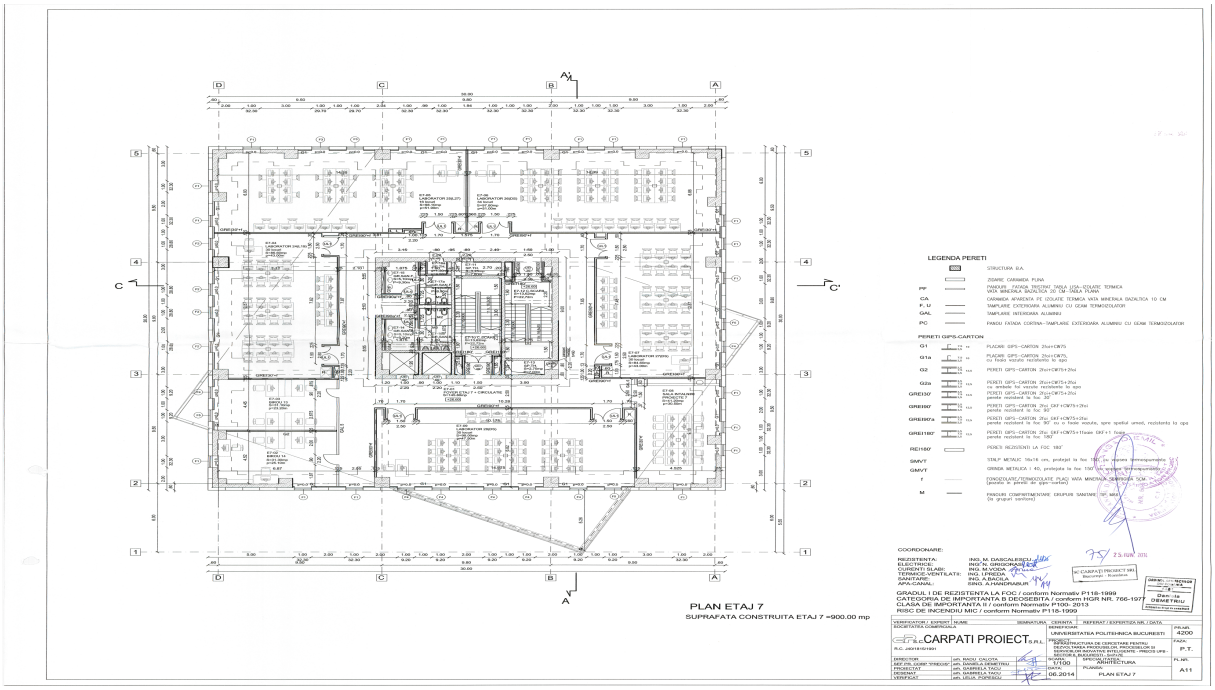

Figure 6: PRECIS Floor Plan

5.1 Three-rooms Flat Evaluation

In this section a description of how the evaluation in a three-rooms flat was done and what were the results.

The data set taken for this experiment consisted of 276 WiFi fingerprints taken from all the three rooms from the flat, the kitchen and the balcony and 9 GPS fingerprints. The 9 known

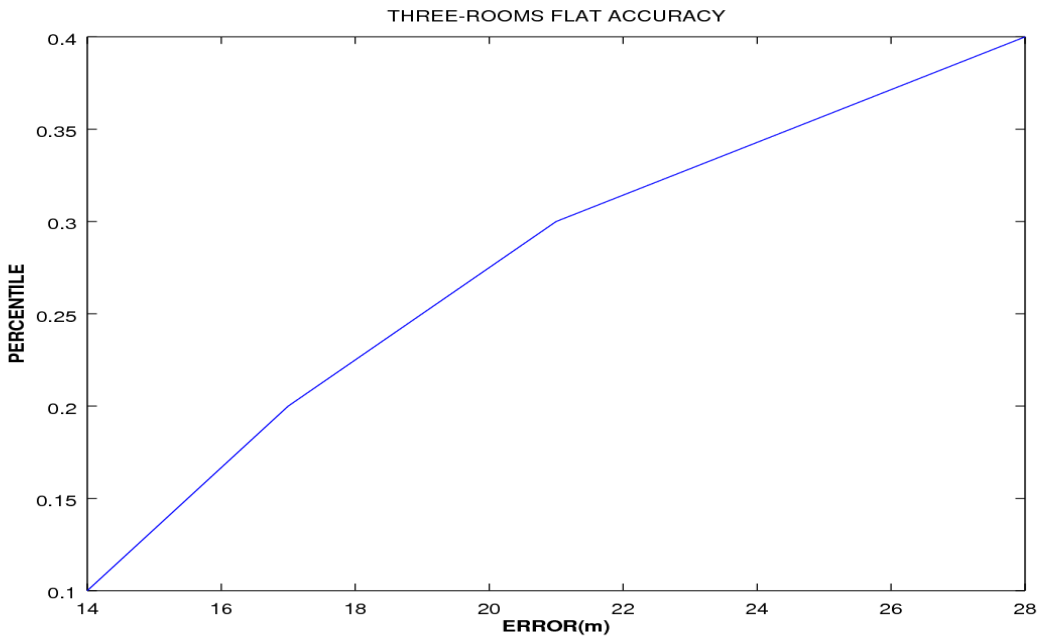

Figure 7: Performance in Three-rooms Flat

locations‘ fingerprints were taken near the windows of each room, in the balcony and at the entrance of the flat, near the building‘s hall windows.

The results were far from the expectations. The evaluation of the implementation in the state-of-the-art proposal [3] in a small building concludes in having a maximum error of 3.3m with a 0.8 probability, whereas the current implementation has a 0.1 probability of having an error of maximum 8m - Figure 7.

The bad performance of the implementation in this scenario may be motivated by the crowded environment. Also, being in a building with several floors of flats, there are many entities deploying WiFi access points at several altitudes - the average number of WiFi records per fingerprint was around 50 RSS from different access points. The evaluation was held during the day, in medium to high traffic conditions. However, this factors should slightly affect the localization in comparison to this situation‘s results and the deployment of the state-of-the art system in a small building [3].

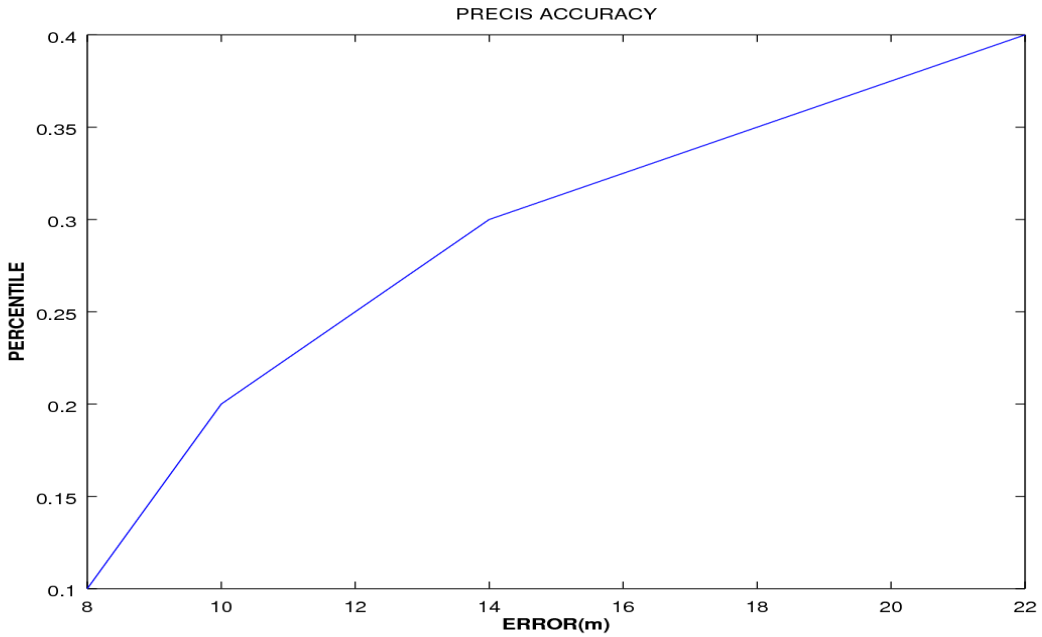

Figure 8: Performance in PRECIS Office Building

5.2 PRECIS Office Building Evaluation

This section presents the results after the deployment of the implemented solution in PRECIS Office Building. Discussions regarding the performance of the system are also introduced.

Since in this scenario the tracked indoor space was situated in a peaceful area and the majority of the available access points were situated on the same floor, better results than in the three-rooms flat space were expected. Although, the results were even worse from some causes presented in the next paragraph.

For this scenario, a data set of 238 WiFi samples and 6 known locations obtained through GPS queries was used. The fingerprints were taken in 4 different rooms and on the floor‘s hall.

The original implementation [3] has an error of 7m in 50% of the cases and 10m in 80% of the tests performed in a large building. The current solution‘s performance is illustrated in Figure 8.

A possible reason for having a great error rate in the PRECIS testing scenario is the limited number of the visible access points in the area. Only 5 access points had coverage in all the

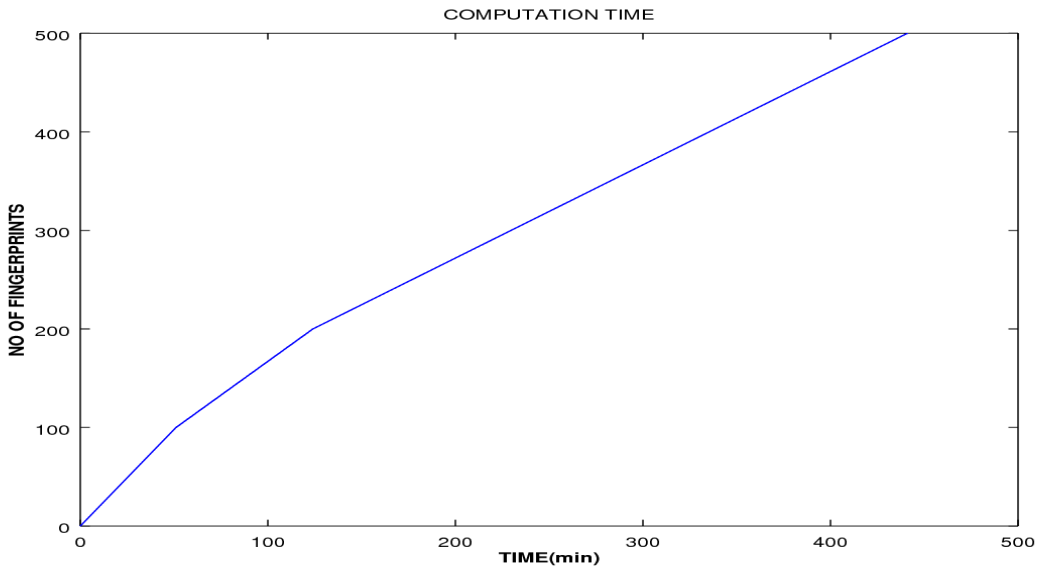

Figure 9: Computation Time

places from the floor.

5.3 Access Point Computation Evaluation

In the evaluation process no prerequisite data about the available access points was known in neither of the scenarios.

However, the computation time for the access points parameters was measured according to the number of RSS WiFi records - Figure 9.

5.4 Possible Implementation Enhancements and Bugs

In this section information about the possible sources of error from the implementation are presented.

The gradient descent is used to refine the solutions by minimizing the last mean error. Although, the J_{EZ} scores after sending the solutions to the gradient descent method improve less than expected. As a consequence, the algorithm should be improved to have a greater impact on the solutions.

Some of the access points used in the solutions may be unreliable. A better selection of the

relevant access points may improve the performance of the system [3].

The known locations gained from the GPS satellites are inaccurate. An error of knowing certain locations with an error of 10m may drastically influence the performance of the algorithm. The GPS fingerprints data collection should be improved.

6 CONCLUSIONS

This chapter discusses conclusions after implementing a state-of-the-art indoor localization system [3] and proposes further work on the present solution.

6.1 Further work

Besides the possible further work proposed in the *Implementation Evaluation* chapter, there are also some possible enhancements for the present system.

The dynamics of a traffic from an indoor space may drastically influence the computed parameters of the access points. To reduce the impact, dedicated data sets may be used - for example day localization/night localization, using accordingly data sets.

The current implementation uses a distributed solution only for gradient descent solution refining. Further parallelization of other modules - like solution generation may be implemented.

The current thesis main purpose to implement a state-of-the-art solution for indoor localization was achieved. The decrease in pre-deployment effort comes with an accuracy trade-off. The implementation presented in the thesis is still less accurate than expected due to possible reasons described in the previous chapter.

A WiFi signal-based system localizing mobile nodes indoors without knowing any prior information about the access points is highly desired solution, since many entities may deploy various access points in a building and parameters prerequisite may not be available. Making the indoor localization of mobile nodes without any user participation and with a single node participation are also key advantages of the EZ solution.

REFERENCES

[1] E. Aitenbichler and M. Muhlhauser. An IR Positioning System for Smart Items and Devices. *Distributed Computing Systems Workshops*, 2003.

[2] P. Bahl and V. N. Padmanabhan. RADAR: An Inbuilding RF-based User Location and Tracking System. *INFOCOM*, 2000.

[3] Krishna Chintalapudi, Anand Padmanabha Iyer, and Venkata N. Padmanabhan. Indoor Localization Without the Pain. *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 173–184, 2010.

[4] Y. Gwon and R. Jain. An IR Positioning System for Smart Items and Devices. *Mobiwac*, 2004.

[5] T.D. Hamalainen, H. Leppakoski, M. Hannikainen, and A. Kotanen. Experiments on local positioning with Bluetooth. *Proceedings ITCC 2003. International Conference on Information Technology: Coding and Computing*, 2003.

[6] J. Hightower, G. Borriello, and R. Want. An Indoor 3D Location Sensing Technology Based on RF Signal Strength. *University of Washington*, 2000.

[7] Dr. Zeev Weissman. Indoor location. `http://www.tadlys.co.il/media/downloads/Indoor_location_Systems.pdf`. Last accessed: 24 June 2018.

[8] Karim Yaghmour. *Embedded Android*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2003.

[9] M. Youssef and A. Agrawala. The Horus WLAN Location Determination System. *MobiSys*, 2005.

[10] Cemin Zhang, Michael Kuhn, Brandon Merkl, Aly E. Fathy, and Mohamed Mahfouz. Accuracy enhancement of uwb indoor localization system via arrangement of base stations. `http://www.ursi.org/proceedings/procGA08/papers/C05bp1.pdf`. Last accessed: 24 June 2018.