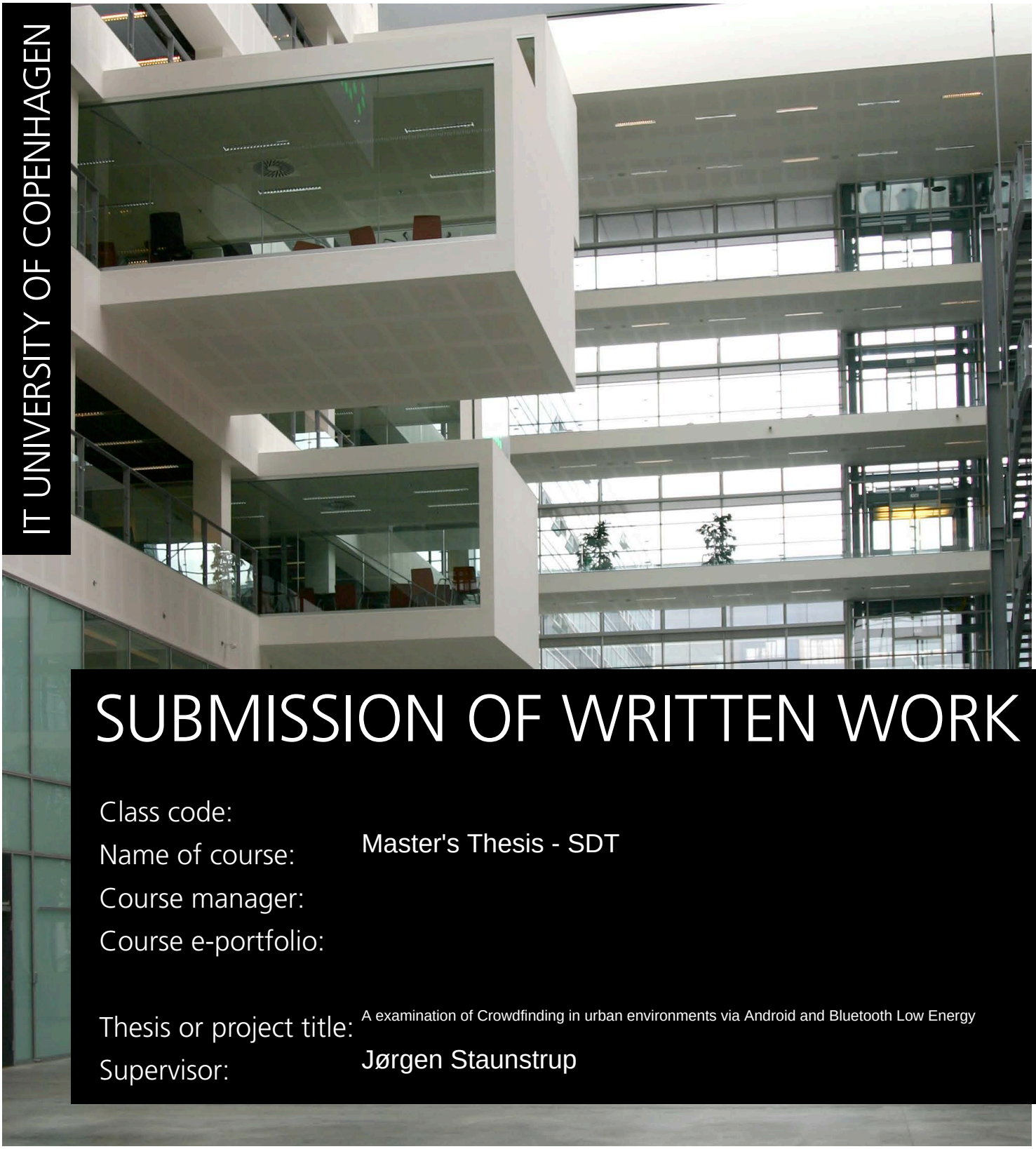
IT UNIVERSITY OF COPENHAGEN



SUBMISSION OF WRITTEN WORK

Class code:

Name of course: Master's Thesis - SDT

Course manager:

Course e-portfolio:

Thesis or project title: A examination of Crowdfinding in urban environments via Android and Bluetooth Low Energy

Supervisor: Jørgen Staunstrup

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Johnni Hested	03/09-1985	jhes@itu.dk
2.	Thomas Charles Rohleder	10/12-1985	tcha@itu.dk
3.			@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

A examination of Crowdfinding in urban environments via Android and Bluetooth Low Energy

Master's thesis 2018

Authors

Johnni Hested
Thomas C. Rohleder

IT UNIVERSITY OF COPENHAGEN

IT UNIVERSITY OF COPENHAGEN

Department of Computer Science

A examination of Crowdfinding in urban environments via Android and Bluetooth Low Energy

Authors

Johnni Hested - jhes@itu.dk

Thomas C. Rohleder - tcha@itu.dk

Supervisor

Jørgen Staunstrup

Master's thesis

Software Development

May 31, 2018

Version: 1.0

IT University of Copenhagen
Department of Computer Science
Rued Langgaards Vej 7
2300 Copenhagen
Denmark

Abstract

Recovering lost bicycles in a city can be a difficult task to overcome and it can be compared to finding a needle in a haystack. Owners, as well as bicycle rental companies like Donkey Republic, can benefit from technologies which can aid in this task. This thesis will examine how smartphones automatically can assist in locating bicycles augmented with Bluetooth. The location of the bicycle is defined as a circular area. To locate a bicycle, an Android prototype and an algorithm is developed. These combines location data from multiple smartphones in order to estimate the smallest area possible. Each location data point is generated with two different methods, FIXED_m and RSSI_m, which is developed for the purpose. FIXED_m uses a distance between the smartphone and the bicycle of 20 meters and RSSI_m calculates a distance based on Bluetooth signal strength. The prototype is used to analyze which of the two methods perform the best in locating bicycles in the city. The results show that FIXED_m is a better option for locating bicycles because it estimates areas down to 20 meters of radius in size where the bicycle is certain to be inside. The thesis suggests that the Crowdfinding algorithm can be implemented on third-party platforms. This will increase the size of the crowd which will increase the chance of finding a lost bicycle.

Keywords: Crowdfinding, Localization, Android, Crowdsourcing, Pervasive Computing, Bluetooth Low Energy, BLE, RSSI, Network-assisted location, Smartphones.

Acknowledgement

We would like to express our gratitude to our supervisor Jørgen Staunstrup of the Computer Science department at the IT University of Copenhagen. During our Mobile App Development course, Staunstrup introduced us to the possibilities of designing context aware applications with of BLE beacons. He lectured with passion and enthusiasm which inspired us to research Android and BLE for pervasive systems. We have developed and analyzed a pervasive Crowdfinding prototype for bicycles which would not have been possible without his advice and guidance. Staunstrup's academic experience helped us achieve a high quality thesis.

We would also like to extend a thanks Jens Kjærby Frandsen, CTO of Donkey Republic, for taking interest in Crowdfinding and for discussing the issue of locating bicycles. We are also thankful for lending an AXA Bluetooth lock of the type currently used by the company.

Source code

The experiments of this thesis were conducted with two Android prototype applications. The source code for the applications is publicly available in our GitHub repository.

Repository: `https://git.io/vhIE6`

Contents

1 Introduction

In many cities, the bicycle has become a sustainable alternative to motorized transportation such as cars and busses. This is because of the bicycle's many advantages, e.g. it is a healthy alternative to cars and busses, it is a faster form of transportation in cities during rush hour, and it has a small carbon emission footprint compared to a car. Despite the advantages, owning a bicycle presents new challenges for the cyclist.

Losing a bicycle is one of the main challenges cyclists are faced with. It is an economic loss that in worst case can be very difficult to deal with because bicycles can be very expensive. A bicycle can be lost in many different ways; sometimes the owner simply forgets where the bicycle is parked. Other times external factors such as theft or displacement by law enforcement is the cause of the loss.

Several companies have come up with solutions for solving the issues of losing a bicycle. One solution is to augment bicycles with location technology such as GPS or by using Bluetooth transmitters that can be detected. The main issue with GPS is its large power consumption which means the batteries will have to be changed or charged often. An issue with the existing Bluetooth based solutions is that the details about methodology, technique, and accuracy is not known to neither researchers or the public because the software is proprietary. Other companies, like Donkey Republic, have a radically different approach. Instead of helping cyclist locating their bicycle, they offer bicycle rental via mobile applications. The bicycle

can be picked up and parked anywhere in the city. This moves the challenges of locating the bicycles from the cyclists to the companies.

1.1 Goal of the Thesis

This thesis examines a method for locating lost bicycles which is a major challenge for both private owners as well as the bicycle rental companies. We propose a system which utilizes the movement of citizens in a city and the sensors on their smartphones to find lost bicycles. This system is an implementation of Crowdfinding which we will elaborate on in Chapter 3. Our implementation combines techniques from Pervasive computing, Crowdsourcing and localization theory, in order to locate bicycles in a city in a cost and energy efficient way. It uses GPS location data together with Bluetooth RSSI measurements from smartphones in order to estimate an area of location for each lost bicycle. The bicycles have to be augmented with Bluetooth transmitters. We will analyze two methods for creating Location data points that we call $RSSI_m$ and $FIXED_m$. Then we will develop an algorithm which uses the location data points to estimate an area which a lost bicycle is within. The algorithm combines Proximity between Bluetooth transmitters and receivers with geometry to calculate the most probable area possible while reducing the size of the area.

We will develop:

1. An algorithm for our Crowdfinding implementation which can narrow down the size of the estimate area by combining location data points.
2. An Android prototype which can test the Crowdfinding algorithm in a city environment using the $RSSI_m$ and $FIXED_m$ methods.

Much research has been conducted on using Bluetooth RSSI for indoor location [29] [7] [25] [40] [21]. Hested, Andreasen and Rohleder showed how LoPy Bluetooth

receivers could track Estimote beacons in an indoor environment. Furthermore, they showed how hardware specific RSSI calibrations could improve the accuracy [15]. Due to the large array of use cases where GPS is a sufficient tool for outdoor localization, there is a gap in the literature on how Bluetooth can be used as an outdoor localization technology in the city. Determining the location outside based on Bluetooth RSSI presents many challenges when it comes to accuracy and reliability and therefore the subject still needs to be researched. Furthermore, no research has been conducted on how to estimate an area based on GPS and Bluetooth and reducing its size with the geometry of intersecting circles.

This thesis will contribute by examining how circular location data points from a combination of GPS and Bluetooth can be used to estimate an area and reduce its size as more data points become available. Additionally, the thesis examines how Bluetooth RSSI can be used as a measure of distance on different smartphones in cities in order to increase the accuracy of the location data points. The goal of this thesis is to answer three main questions:

1. Is Crowdfinding useful for finding lost bicycles?
2. How small and accurate can an estimated area get?
3. Is RSSI_m more suitable for Crowdfinding than FIXED_m?

1.2 Thesis Structure

The thesis consists of 6 Chapters including the Introduction. Chapter 2, Theory, introduces the theoretical foundation. It includes a description of pervasive computing in relation to Crowdfinding, how Crowdfinding relates to crowdsourcing, and the relevant localization methods.

Chapter 3, Crowdfinding, provides a deeper understanding of the current state of location technologies which can be used to locate bicycles. This is followed by the our vision and the requirement for designing a Crowdfinding system.

The remainder of the thesis is dedicated to experiments and analysis. Chapter 4, Relation between RSSI and distance, shows how an Android prototype is developed for testing the relationship between distance and RSSI. It examines the difference between two different Samsung smartphones in order to find coefficients for calibrating a distance equation.

Chapter 5, Finding the location of the bicycle, presents an algorithm for locating bicycles and juxtaposes the $RSSI_m$ and $FIXED_m$ methods.

Lastly, in Chapter 6 Conclusion, the thesis will answer the three main questions presented in Chapter 1. Furthermore, it will discuss Crowdfinding in a larger context.

Compiled lists of figures, code listings, tables, and glossary can be found in the end of the thesis.

The prerequisites for reading this thesis is an understanding of computer science, programming concepts, and geometry.

2 Theory

The theoretical framework is laid out in the following section. We dive into pervasive computing and how the distribution of computation is opening up new possibilities for localization systems. Then we describe crowdsourcing which is an important element in Crowdfinding. Crowdsourcing presents a theoretical framework for outsourcing tasks to a crowd which tie closely to how pervasive computing distribute workload on a multitude of computers and sensors. Finally, we look at localization theory which describes methods for locating objects in the physical world. This will highlight how pervasive computing and crowdsourcing can aid in building low energy localization systems for Crowdfinding.

2.1 Pervasive Computing

Pervasive computing seeks to weave computation into everyday objects in unobtrusive ways. Back in 1988, Mark Weiser, chief scientist at the Xerox PARC (Palo Alto Research Centre), envisioned a future where computers would converge with everyday object to a point where they would be invisible [10]. He called this vision ubiquitous computing (ubicomp). For Weiser, the goal of ubicomp was to build large Computer-Supported Cooperative Work systems (CSCW) which would aid the office worker by having unobtrusive interaction with the CSCW system in the office environment. He called this invisible computing [1]. Other researchers in ubicomp has later argued that the real goal is to provide many single-activity

systems that promote a unified and continuous interaction between humans and services[1]. IBM coined the term pervasive computing in the mid-1990s which had many similarities with ubicomp but with a focus on mobility and people's everyday life [2]. They worked on a pervasive system with Swissair that enabled travelers to check in and get information about the flight via a mobile browser. This can be seen as a precursor to how modern humans interact with services through their smartphone. The smartphone can be seen as a part of both Weiser's and IBM's vision and nowadays the terms ubicomp and pervasive computing are used interchangeable. This is in part because laptops, tablets, and smartphones has made office work mobile. Furthermore, the work sphere and private sphere have become more intertwined because of this change. E.g. people use an e-mail client on their smartphones for both private- and work-related tasks. The focus of our thesis is to locate lost bicycles augmented with Bluetooth by having smartphones automatically detect the Bluetooth signal in an unobtrusive way. This makes The system pervasive.

Smartphones is the convergence of many different technologies which make them good pervasive systems. Smartphones are relatively cheap, they are small, they are good at solving many different tasks, and they have a multitude of sensors. Jakob Bardram and Adrian Friday elaborates on the paradigm by stating that pervasive systems should strive to solve problems by using small and inexpensive devices which can communicate wirelessly [2]. Pervasive systems have an inherent need to be context aware [1] which means that the systems need sensing capabilities to adapt to the user's needs. Context information can be acquired via sensors embedded in the smartphones and can relate to hardware, nearby Bluetooth devices, and geographical location. Location is a commonly used context which can be provided by GPS [1]. GPS is widely used but it comes with a cost. It has a large power consumption which has a limiting effect on mobile devices [8]. This is usually not an issue for devices that are charged daily but for devices that are not, this becomes a problem because it requires attention and therefore becomes obtrusive and breaks the illusion of invisible computing. We address this issue

by having the smartphone handle the GPS location instead of the bicycle and we place a Bluetooth transmitter on the bicycle that the smartphone can detect. This combination has a lower power consumption on the bicycle which means the user rarely has to recharge or change the battery on the transmitter. Now the smartphone can detect the bicycle in an unobtrusive manner which is a goal for pervasive systems.

2.2 Crowdsourcing

Crowdsourcing (CS) is a process for connecting organizations with online crowds such that information and problems can be shared between the organization and the crowd. As computation becomes pervasive, new forms of CS is become possible [4]. For example, smartphones can collect real-time data about the vicinity of bicycles while the owner of the phone conducts other business around the city. This circumvent many of the problems related to motivation since the crowd can perform the sourced task passively. A common problem with CS is how an organization motivates the crowd to perform tasks. One way is by providing an economic incentive by paying the crowd for conducting tasks. This is known as extrinsic motivation which can be defined as external sources of motivation [17] . Another approach is intrinsic motivation which appeal to the crowd's internal drivers, such as the internal satisfaction from helping others or mastery of a skill [26]. Crowdfinding utilizes the crowd by distributing the task for mutual benefit. It can be difficult for an organization to augment a whole city with sensing capabilities. Crowdfinding distributes this task to the crowd which is acting as moving Bluetooth receivers. It can be difficult and time consuming for individuals to find the location of a lost bicycle and that is where Crowdfinding can assist.

CS allows Crowdfinding to utilize an online community in order to perform a mutual beneficial task; finding a bicycle across a large area. CS amplifies the range of the reach of Crowdfinding as well as the individual. Traditionally, CS relates to

what has been coined Human Computation, according to Brabham [4]. Human Computation relates to tasks which can be difficult for a computer to solve such as ideation and design. Often CS is used in areas such as design and production of consumer goods, media content, science, and policies. Some companies are using CS in relation to gathering geographical information. Traditionally, this is done by having users manually inputting data via a website or mobile application. One of these applications is Ushahidi [32] which is a CS platform for social activism and crisis response. The application allows users to manually input hazards on a map such that responders can act on it and civilians can avoid a dangerous situation. In Crowdfinding, we use CS to distribute the task of moving around in the environment to the users while having their smartphones search and collect Bluetooth data from nearby bicycles. This happens automatically without interaction from the user.

2.3 Localization

Many methods have been developed for providing location context on smartphones. The most common metric for location is geographical coordinates based on longitude and latitude [33]. Location technology is divided into three sub infrastructures; client-based, network-based, and network-assisted [33]. In client-based systems, the client receives a signal from transmitters such that the client can calculate its own position. One of the advantages of client-based systems is how well they scale since the device only has to calculate its own position as opposed to network-based systems. A problem in relation to client-based localization is battery drainage due to the many calculations, according to Varhavsky and Patel [33]. A good example of a client-based system is GPS. An alternative to client-based localization is network-based localization which has a network of receivers which calculates the position of the transmitters in the environment. An issue with network-based localization is the scalability. Tracking many objects puts pressure on the system as a whole, because all the calculation is done at once. The advantage of network-based systems is the low power consumption on the client side. The last category

is network-assisted, which is a combination of the two aforementioned categories. Network-assisted is a good solution for locating bicycles because it shifts the power consuming calculations away from the bicycle and onto the smartphone. This can be done by augmenting the bicycle with a Bluetooth transmitter and designing a mobile application which is capable of measuring the signal strength. Then the position of the bicycle can be calculated in relation to the smartphone.

2.3.1 Determining location in network-assisted systems

One method for determining the location of a device is by proximity. If the device is in proximity of a sensor then the device must be at a certain reference point [33] [22] [21]. With Bluetooth, this means within range of the transmitter. In 2007/2008, a team of researchers tested how RFID and Bluetooth could be used for tracking passengers inside Copenhagen Airport. They used proximity-zones by dividing the airport into areas equipped with receivers. In their implementation, Bluetooth was used to track the general flow of users based on the minority which had Bluetooth advertisement turned on. They chose to use RFID tags for tracking passengers because they could associate each tag to a passenger's travel information. Another argument was that RFID, at the time, was more energy efficient than Bluetooth [14]. Whenever the tags entered a zone where a reference point was in range, the system would conclude that the tag was in the respective area.

Smartphones can be used as reference points for Crowdfinding and then use Bluetooth for detecting the proximity of a bicycle. To do that, the geographical position of the smartphone has to be known. Android's FusedLocationProvider, which is part of Google Play Services, can be used to obtain the GPS coordinates of the smartphone. GPS coordinates comes with some uncertainty when it is not under ideal circumstances [36]. For that reason, Android provides a calculated

accuracy in meters. The minimum possible GPS accuracy is 10 meters. According to Android, there is a 68% probability that the smartphone is within a radius of the given accuracy from the GPS coordinates [19].

2.3.2 Determining the distance to the bicycle

One method to determine the distance is via RSSI (Received Signal Strength Indication) [37]. Bluetooth signals can be used with this method because the signal strength can be measured and translated into distance. Bluetooth is a technology that provides wireless communication between devices. The technology was invented in 1994 by the Swedish telecommunication company Ericsson as an alternative to wired communication [12]. In 2010, Bluetooth version 4.0 was released and it introduced BLE (Bluetooth Low Energy) which made it possible for devices to communicate with a lower energy consumption. This paved the way for small, battery powered devices such as heart rate monitors, wireless headphones, beacons, etc., which potentially can last for months or even years without being recharged. The BLE protocol was implemented in Android 5.0 in 2014 [19]. This allowed for detection of BLE devices via Android's BluetoothLeScanner. Furthermore, it allowed for different BLE specific callbacks such as filters on UUID, device name, mac address etc. In 2016, Bluetooth 5 was revealed which is the latest version. It provides, among other things, increased range and data transmissions which increases Bluetooth's viability in Internet of Things [28].

The reason why this development in Bluetooth is important is because it can enable a device to act as a beacon that can be discovered by other devices. For this to happen, the two devices have to be relatively close to each other; often less than 100 meters. An analogy of this can be the playful game Marco Polo where players has to find each other without the use of vision but only by repeatedly getting a sound from the other players. The louder the sound is, the closer the players are to each other. The same is true for Bluetooth. A beacon device has to send out

small advertisement packets with its own unique id similar to the player who needs to be found in Marco Polo. A Bluetooth receiver device is now able to discover it and determine that it is within vicinity but it cannot determine the exact distance. To get an approximation of the distance to the beacon, the receiver has to listen for the RSSI signal on different distances. Then the distance can be determined by using a reference map with RSSI to distance relations. This method has some limitations since radio signals interact with the environment in unpredictable ways since it bounces and deflects off of other objects [35]. The RSSI-to-distance map is dependent on the hardware that measures the signal and therefore, each map has to be calibrated to the hardware it runs on [13]. Because Bluetooth signals are omnidirectional, the RSSI decreases logarithmically when the distance increases which makes it even more difficult to calculate the distance accurately [11].

3 Crowdfinding

Crowdfinding is a concept where a crowd of users contribute with information about specified item's position with the goal of locating the items. This enables a single user to locate items across large distances without searching for it themselves. The concept is inspired by crowdsourcing where an organization cooperates with an online community in order to complete predefined tasks that otherwise would be impossible. In Crowdfinding, the predefined task is to locate items distributed over a large area. It is important to specify what type of item to locate and by which technology to use to document the location. Documentation can be achieved manually, for instance with the use of pen and paper, RFID tags, QR codes, or cameras, or it can be achieved automatically. One of the challenges in crowdsourcing is to provide incentive for users to conduct work manually because it can be time consuming or trivial and therefore it is better to automate the process when it is possible. With the technology of smartphones, it is possible to automate the data collection by utilizing sensors, artificial intelligence, or radio antennas. A requirement is that users carry sensors which can locate the items and the items can be detected by the sensors.

One way of automate Crowdfinding is with Bluetooth transmitters and receivers. The company Diims offers Bluetooth transmitters that customers can install on their items [9]. Diims has a collaboration with the postal service which has implemented Bluetooth receivers on their mail vans. The company uses a combination of Bluetooth proximity and GPS position from the mail van to locate items with the

Diims Bluetooth transmitter installed. This approach only covers areas where the postal service drives which limits its range.

Two other interesting companies, which have similar approaches to finding items, are TrackR [31] and Tile [30]. Both companies provide small and inexpensive Bluetooth transmitters as well as a mobile platform for users to interact with their service. The services combine Bluetooth RSSI data from many users' smartphones in order to cover a larger area. Tile has a service they call Community Find which allows the users to set up search parties to find a lost item. TrackR has a similar service they call Crowd Locate where all users on the service can cooperate to find an item. Both companies use their own proprietary algorithms which is a problem from a research point of view.

3.1 Crowdfinding for bicycles

In a modern city, citizens are connected to the internet as never before. This allows for new ways of designing pervasive systems like automated Crowdfinding. Many pervasive systems rely heavily on a sensor infrastructure build into the environment which require a large setup and increases the overall cost and complexity of the system.

Bicycles are becoming popular in modern cities because they help solving some of the challenges we are faced with today [39] [5]. Among these challenges are the increasing inner-city traffic which puts pressure on the road infrastructure, obesity and health concerns because citizens are moving around less, and pollution from cars which is an environmental concern. Owning a bicycle presents new challenges. Due to their mobility, they can easily be stolen, moved, or lost in some other fashion. This is an economical expense which people have to be insured against. If lost bicycles are not returned to their owners they can end up as scrap in the city which leads to expenses for the city which have to clean up.

Donkey Republic is a global bicycle rental company which has deployed bicycles with Bluetooth locks in many modern cities, e.g. Amsterdam and Copenhagen (Figure 3.1). A bicycle can be rented via an application on the smartphone which solves some of the challenges associated with private bicycle ownership. Cyclist no longer have to rely on insurance companies and they can park the bicycle virtually everywhere without remembering the specific location. However, having a global bicycle rental service moves many of the challenges onto the rental company that needs to have exact knowledge of the location of each bicycle. This requires a large infrastructure that needs to be setup and maintained. One of the ways Donkey Republic have tried to solve this problem is by having specific zones for parking [24]. Donkey Republic is not the only company faced with these challenges. Bikeshare Danmark is another bicycle rental company which focus on motor assisted bicycles that are equipped with GPS and internet. These bicycles have to be returned and parked in special racks that the company already know the location of. In April 2018, the Bikeshare Danmark was exposed to a hacker attack and they lost the location of all their bicycles [6]. The company asked citizens for help via a Facebook post where they encouraged citizens to take a picture of a lost bicycle and its location and send it to the company [3]. This method relies heavily on manual work despite the bicycle's own capabilities. Another company, Sherlock, uses GPS which can locate a bicycle accurately but it is also a large power consumer [27]. Because of that, the user has to charge the device regularly otherwise the GPS will not work and the bicycle cannot be found.

Figure 3.1: Donkey Republic bicycle with a Bluetooth lock

In the light of these challenges, we propose Crowdfinding as a pervasive localization system built on the existing infrastructure provided by a Crowd of smartphones. We envision a pervasive system which utilizes the mobility of the crowd and their smartphone sensors to automatically solve the task of locating lost bicycles. This is a relatively inexpensive solution to a problem many people face. Utilizing the sensors provided by each smartphone in a crowd is an effective way of gaining access to a large infrastructure of sensors in a city. The advantage of a crowd of sensors is that each sensor is mobile and therefore it will cover a larger area over time. This does not guarantee that an area will be covered by a sensor at any given time but if we assume that the bicycle we want to locate is stationary, then it will be discovered at some point in time. In Crowdfinding, citizens can be utilized to create a pervasive infrastructure to find lost bicycles which in return will benefit the citizens and the city. A requirement for Crowdfinding is that the bicycles we want to find have the ability to advertise their presence. This will be achieved by installing a Bluetooth transmitter on the bicycle. Bluetooth is an inexpensive and reliable way of deploying radio transmitters for prolonged periods of time and it is compatible with the majority of smartphones.

Another requirement is the algorithm that narrows down the estimated area of the bicycle based on multiple data points from a multitude of smartphones. The Estimated area are calculated on a server, such that the computation on the smartphones is kept at a minimum. Each data point is marked with a bicycle id and a timestamp such that the system can detect if a bicycle has changed its position. If new data points arrive far away from the previous, then we assume that the bicycle has been moved and the system updates the bicycle location to the new area. Lastly, the participating users are required to have a smartphone which are able to handle client-side calculation and data collection from Bluetooth and GPS. The client side of Crowdfinding comes as a standalone application as well as an API that integrates into third-party applications such as Donkey Republic's bicycle rental application.

This thesis focus on a Crowdfinding prototype build on Android that can accurately locate lost bicycles by combing techniques from pervasive computing, crowdsourcing, and localization. We provide an algorithm that combines multiple location data points from different smartphones in order to estimate the smallest and most probable area possible. The prototype examines the algorithm and how accurate the estimated area can get. Furthermore, the thesis shows how to improve the estimated area by examining the relationship between distance and Bluetooth RSSI. We develop two methods, RSSI_m and FIXED_m, which is analyzed through the prototype.

4 Relation between RSSI and distance

The following Chapter focuses on distance estimation based on RSSI. A data collection, followed by an analysis, is conducted in order to see how well RSSI can be utilized to estimate distance. This is important because the more accurate we are able to measure the distance between the phone and the bicycle, the better our estimated area will be in Chapter 5. We collect and analyze RSSI data in order to derive smartphone specific functions that can calculate the distance based on RSSI values. The subsequent evaluation of these functions will elucidate how well this method performs in estimating distance.

We have developed an Android prototype for the purpose of data collection and evaluation. The system design Chapter will elaborate on the prototype. Throughout this project, all data collections and experiments will be conducted on a Samsung Galaxy S6 (API 24) and a Samsung Galaxy S8 (API 24). These will be referred to as S6 and S8 throughout the thesis. The bicycle used for the experiments in this thesis has an AXA Bluetooth lock attached. The lock is provided by Donkey Republic. It broadcasts advertisements each second which allows smartphones to measure the signal strength. We initially measured the maximum distance for receiving a Bluetooth signal from the bicycle. Both smartphones have a maximum receiving distance of 20 meters. Above this distance, the phones are not able to detect the Bluetooth signal from the bicycle. We define the FIXED_{m} method as

detecting a bicycle inside a proximity zone of 20 meters. The $RSSI_m$ method is defined as determining distances between 1 and 20 meters based on RSSI values. This Chapter will cover the $RSSI_m$ method in more detail.

4.1 Methodology

By using the RSSI as a measurement of distance, we evaluate how accurate we can estimate the distance between a bicycle augmented with Bluetooth and a smartphone. Our approach consists of three parts. First is to collect the RSSI on different distances. Secondly, based on the data collection, we create an exponential function where RSSI is mapped to a distance. Lastly, we evaluate the function to examine how accurately it can measure distance.

In order to collect the RSSI data from the bicycle, we create an Android application that can scan for Bluetooth advertisements and log the advertisement name along with the measured RSSI and the known distance. RSSI can fluctuate even when the transmitter and receiver are stationary. Therefore, we decide to take 12 measurements on each distance and then use the median RSSI value. In this particular case, we use the median to get the most accurate value because it is less susceptible to the influence of outliers from a fluctuating signal than the mean value [18].

After all the data points are collected, we generate the RSSI-to-distance function by using logarithmic regression analysis. We use logarithmic regression because the RSSI value decreases logarithmically over distance. To generate the function, we download the data from the phone as a CSV file and import it into a spreadsheet. Then we use logarithmic regression on the data set which will generate two coefficients a and b (Equation 4.1). This is a general RSSI-to-distance function for a specific phone model where a and b are the coefficients generated from the logarithmic regression.

$$dist(RSSI) = e^{(\frac{RSSI-a}{b})}$$

Equation 4.1: RSSI-to-distance function

To conduct the data collection, it requires a smartphone that acts as the Controller and one or more smartphones that are the Collectors. The Controller is used for starting and stopping experiments and displaying the results. The Collectors measure the RSSI from the bicycle, calculate the median for each collection, and upload the result to the server. Furthermore, it requires a track with known distances to conduct the RSSI collection. In our setup, the track is marked at each meter up to a maximum of 20 meters. The bicycle is placed on mark 0. The data is collected on predetermined distances. Figure 4.1 shows the distances we use for collecting the initial RSSI data which is used to create the RSSI-to-distance function. It also shows the distances used to evaluate the function afterwards. We measure the RSSI at each distance on two different Collectors, the S6 and S8. The Collectors are placed in two chest pockets on the researcher with the screen facing away from the body. The researcher is standing on the predetermined positions on the track facing towards the bicycle. Then the data collection is started from the Controller. The collection is conducted outdoor in a vacated street which resembles a real situation to get more realistic data [16] [23]. The process is thorough in order to maintain consistency in the data.

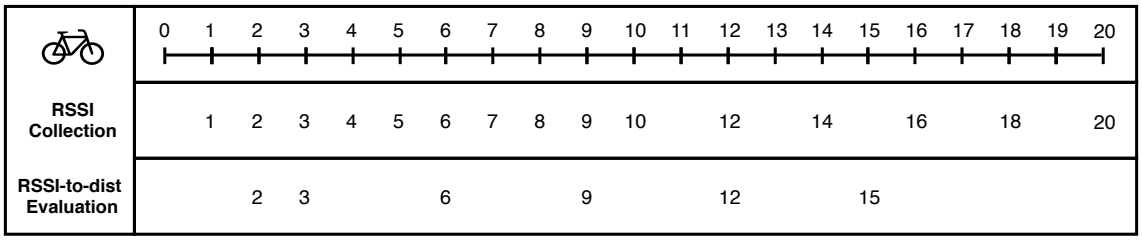


Figure 4.1: Track for RSSI collection and RSSI-to-distance evaluation

Now we have a method to collect RSSI data points from the bicycle on known distances and a method to calculate the coefficients for the RSSI-to-distance function. The next step it to evaluate the function. We evaluate the function by analyzing

the error in meters to the bicycle. To do that, we conduct an experiment where we measure and calculate the distance several times and compare it to the actual distance. Then we can find the mean and the standard deviation (SD). The error in meters is calculated by subtracting the mean from the actual distance.

To collect the distance data, we extend the prototype so it is able to collect RSSI data and calculate the distance by using the RSSI-to-distance function. The prototype collects 40 data points and then sorts the distances from low to high. We do this because Bluetooth signals fluctuate and a single measurement does not give a reliable result. Therefore, we decided that 40 data points are sufficient for the analysis.

Then we remove outliers by using Tukey's fences with a k value of 1.5 in order eliminate errors caused by the experiment (Equation 4.2). This is important because the outliers can have a large influence on the mean and therefore the final results [34]. We use this method and k value because it is efficient at removing data points that skews the mean.

$$[Q_1 - k(Q_3 - Q_1), Q_3 + k(Q_3 - Q_1)]$$

Equation 4.2: Range of Tukey's fences

After that, we calculate the mean, error in meters, percentage error, SD, and confidence interval (CI). Then we upload the values to the database together with the actual distance for later use in the analysis. The CI shows how far the estimated distance is from the actual distance measured in number of SDs. This is useful because it shows how the results compare to one another. A CI of 1 SD means that the estimated distance is within 1 SD and a CI of 2 SDs means it is within 2 SD, etc. The percentage error is calculated in order to examine if there is a constant deviation in our measurements caused by the experiment (Equation 4.3).

$$error_{percent} = \frac{|V_{true} - V_{observed}|}{V_{true}} \times 100$$

Equation 4.3: Equation for percentage error

It is important that the calculated distance is not lower than the actual distance because this will otherwise cause the estimated area in Chapter 5 to be a false positive. We want to eliminate false positives and therefore we chose a coefficient of determination of 95% which we find sufficient for this type of data. According to Zar, the coefficient can be determined by the researcher for each specific case [38]. If we find a correlation in the percentage error, then we can correct for it in our RSSI-to-distance function by multiplying the estimated distance with the average of the percentage errors. If we do not find a correlation, then we use the maximum percentage error as correction in our distance estimation (Equation 4.4).

Figure 4.2 shows how a calculated distance can be either higher or lower than the actual distance and how adding an error correction can prevent false positives in the calculated distance.

$$dist(RSSI) = e^{(\frac{RSSI-a}{b})} \times correction$$

Equation 4.4: RSSI-to-distance function with correction

4.2 Prototype design and implementation

The prototype is designed with emphasis on experimentation and is comprised of two Android applications and a Firebase Database. The first Android application is meant to run on the Collectors which collect RSSI data from bicycles and store it on the server.

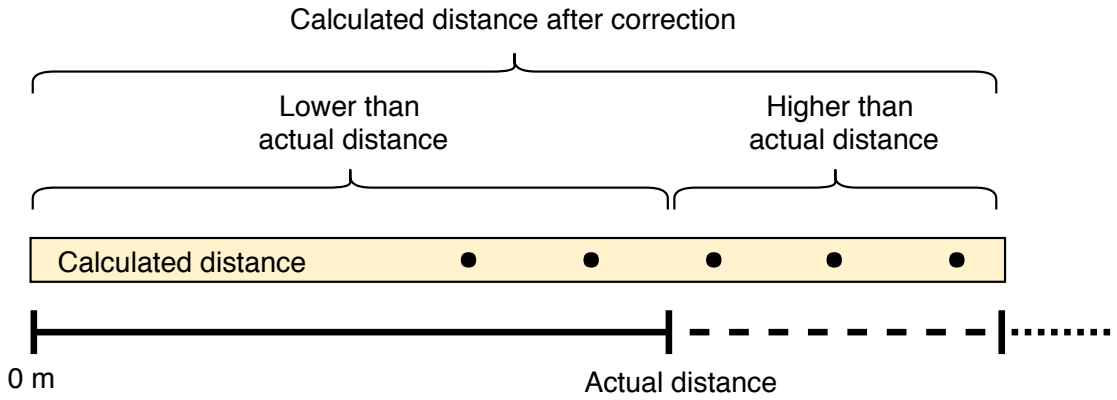


Figure 4.2: Correcting the calculated distance with the percentage error

The second application is the Controller which can start and stop the Collectors remotely. This is necessary because we want to be able to control the experiments without interfering physically with the Collectors which could cause errors in the data collection. We have chosen Android API 21 as the targeted API level because it is the lowest level for which BLE can be used.

4.2.1 Server and data structure

The prototype is distributed across multiple devices and therefore we need to share the data via an online server. We choose to use Firebase Database for storing data because it provides consistent and reliable data storage and it integrates well with the Android platform. Firebase Database is a NoSQL object database that is schemaless and uses JSON as data structure. This is an advantage when working with Java objects because we can change the data structure and data type in our system without having to change the entire database. The data points from each experiment and from each Collector is stored as separate lists in the database (Figure 4.3).

Figure 4.4 shows how single data objects are stored in the database. The *Function coefficients* child can hold multiple Collectors where each has one instance of a data object. The *Running experiment* child holds information that controls the system.

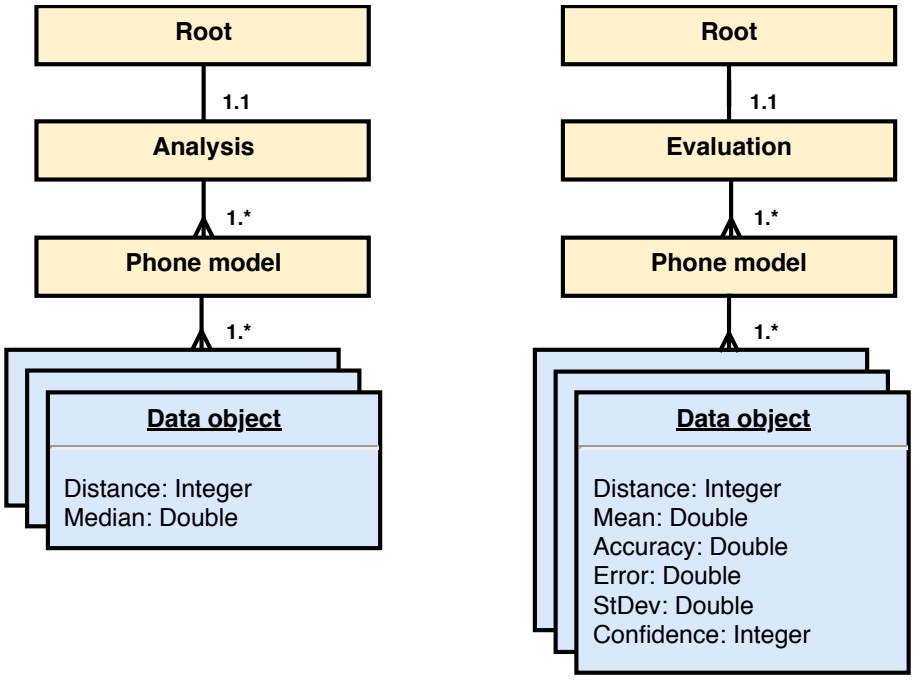


Figure 4.3: Structure for storing lists of data objects

We only need one instance of this object. The value is set from the Controller and then the Collectors listen for the commands.

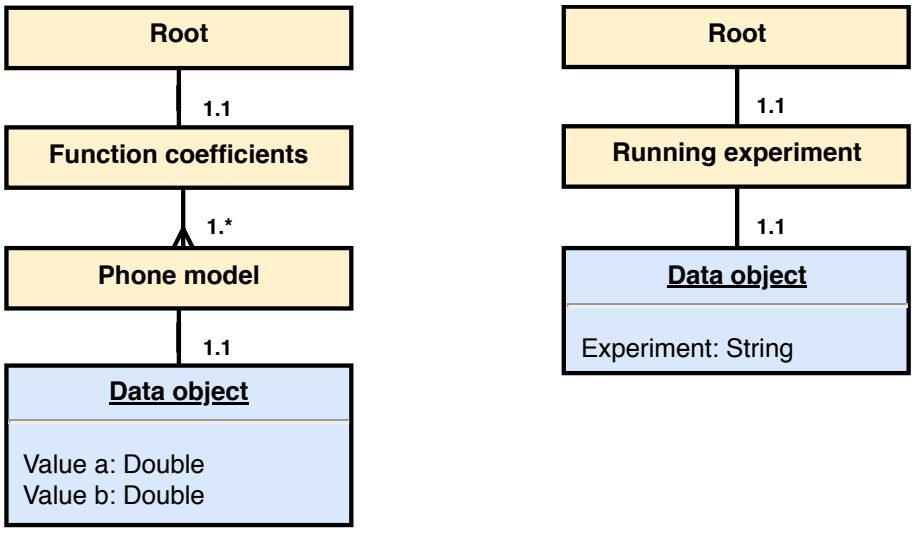


Figure 4.4: Data structure for storing single object values

Listing 4.1 shows how we push data to the server. This is done by creating a data object with the given parameters, as shown on line 1. On line 4 and 5 we get a child reference by using the name of the experiment and Collector. This structure

is shown in Figure 4.3. Then the data object is pushed into a list in the database as shown on line 6 and 7.

```
1 RssiAnalysisModel model = new RssiAnalysisModel(distance, median);
2 FirebaseDatabase.getInstance()
3     .getReference()
4     .child(experimentName)
5     .child(phoneModel)
6     .push()
7     .setValue(model);
```

Listing 4.1: Pushing data objects to a list in Firebase

In Listing 4.2, we show how a listener is set up to listen for value changes in the database. On line 1, we set a reference to the object in the database which we are interested in. This is done by specifying each child. On line 5, we add a value event listener which is responsible for acting when data changes at the specified location in the database. In this case, it listens to the FunctionCoefficientsModel object. When the data changes in the database, it gets pushed to the listener and the Collector can then use the object.

```
1  DatabaseReference database = FirebaseDatabase.getInstance()
2          .getReference()
3          .child(EXP_FUNCTION_COEFFICIENTS)
4          .child(PHONE_MODEL);
5  database.addValueEventListener(new ValueEventListener() {
6      @Override
7      public void onDataChange(DataSnapshot dataSnapshot) {
8          FunctionCoefficientsModel values =
               dataSnapshot.getValue(FunctionCoefficientsModel.class);
9          ...
10     }
11     ...
12 });
```

Listing 4.2: Firebase ValueEventListener on the FunctionCoefficientsModel

4.2.2 Controlling the system

The experiment is started via the Controller by sending a start command with the distance and experiment name to the database. Then the database pushes the start command to the Collectors which start the data collection for the specified experiment. When each Collector has collected enough data, the result is calculated and send to the database. The database receives the result and pushes it to the Controller. This prompt the Controller to update its view with the result so the the researcher can verify it.

4.2.3 Collecting the data

To show how the Collector handles the data collection, we created a flowchart that shows the sequence of operations from when it receives a start command till it has stored the result in the database (Figure 4.5).

Listing 4.3 shows how we handle a scan result from the Android's BluetoothManager. For each result, we get the Bluetooth device name and its RSSI value. If the device name matches the experiment lock, then we store the RSSI value in a list in the Collector's memory. When it has collected enough data points, it calculates the result and stores it on the server.

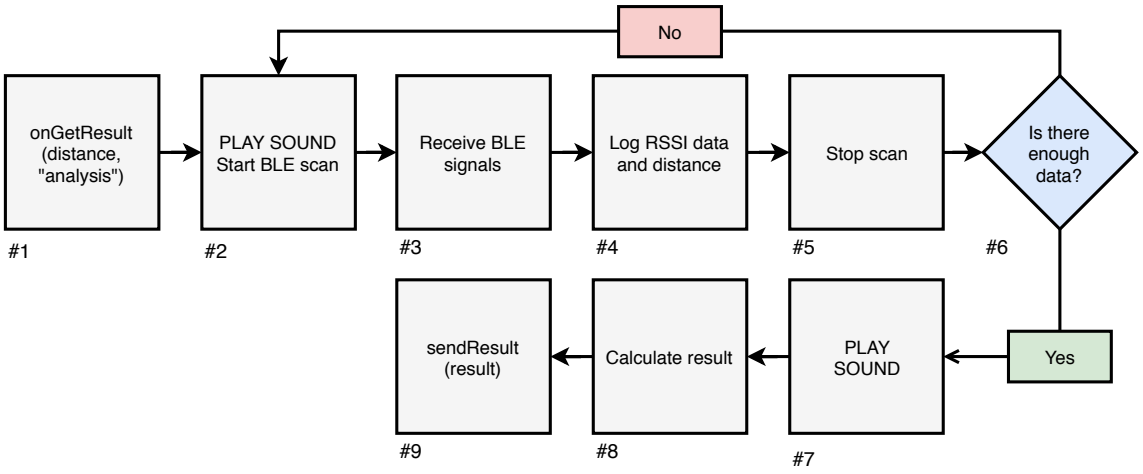


Figure 4.5: Flowchart showing the data collection flow of the Collector

```
private void addScanResult(ScanResult result) {
    String name = result.getScanRecord().getDeviceName();
    Integer rssi = Math.abs(result.getRssi());

    if (name.equals(BIKE_LOCK_NAME)) {
        if (results.containsKey(name)) results.get(name).add(rssi);
        else {
            List<Integer> rssiData = new ArrayList<>();
            rssiData.add(rssi);
            results.put(name, rssiData);
        }
    }
}
```

Listing 4.3: Storing RSSI and name from Bluetooth ScanResult

4.2.4 User interface design

The user interfaces of the prototype are simple and focus on usability and stability. The system has two applications that each has their own activities. The Collector has one activity and its corresponding user interface (Figure 4.6). It shows information about the status of the system in order to make the data collection consistent and correct. It has no buttons to press and the rotation of the screen is disabled to make the application more stable and avoid accidental clicks and swipes.

The Controller is comprised of four activities; Main, RssiAnalysis, RssiEvaluation, and FunctionCoefficients. This is because each analysis and experiment have different requirements in regards to presentation and interaction. The Main activity functions as a navigation menu with buttons to start different activities and to download data (Figure 4.7).

In the RssiAnalysis and RssiEvaluation, we can start a data collection with a specified distance and monitoring the status and result of the experiment. The FunctionCoefficients is for sending coefficient values to the Collectors which is used in the RSSI-to-distance function.

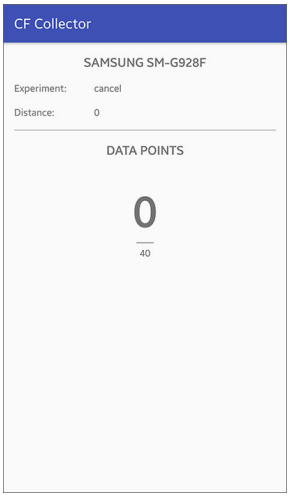


Figure 4.6: Collector user interface

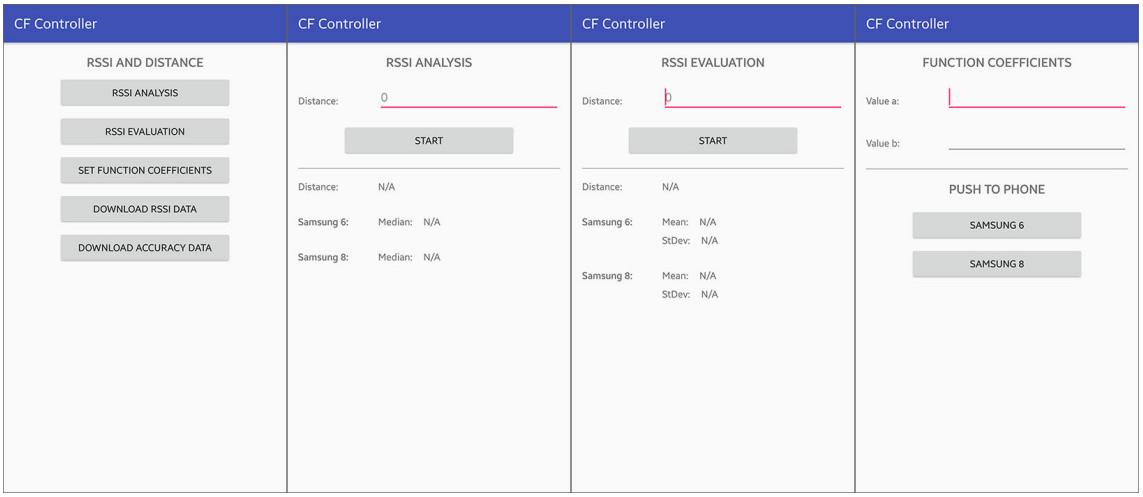


Figure 4.7: Controller user interface

Phone	Value a	Value b	R^2
S6	-73.1	-5.62	0.869
S8	-70.0	-8.46	0.918

Table 4.1: RSSI-to-distance function coefficients for S6 and S8

4.3 Creating the RSSI-to-distance function

In order to create the RSSI-to-distance function, we download the data that we collected and put it into a spreadsheet for analysis. The data we collected is shown in Figure 4.8. Furthermore, it shows that the Collectors measure RSSI differently on the same distance and that is why it is important to create a different RSSI-to-distance function for each smartphone. We used logarithmic regression on the data set to calculate the function coefficients *a* and *b* for each Collector (Table 4.1). The coefficients are used in the RSSI-to-distance function as show in Equation 4.5 and 4.6. We used the functions to map a distance to each RSSI value which shows the spectrum of distances each smartphone can detect (Table 4.2).

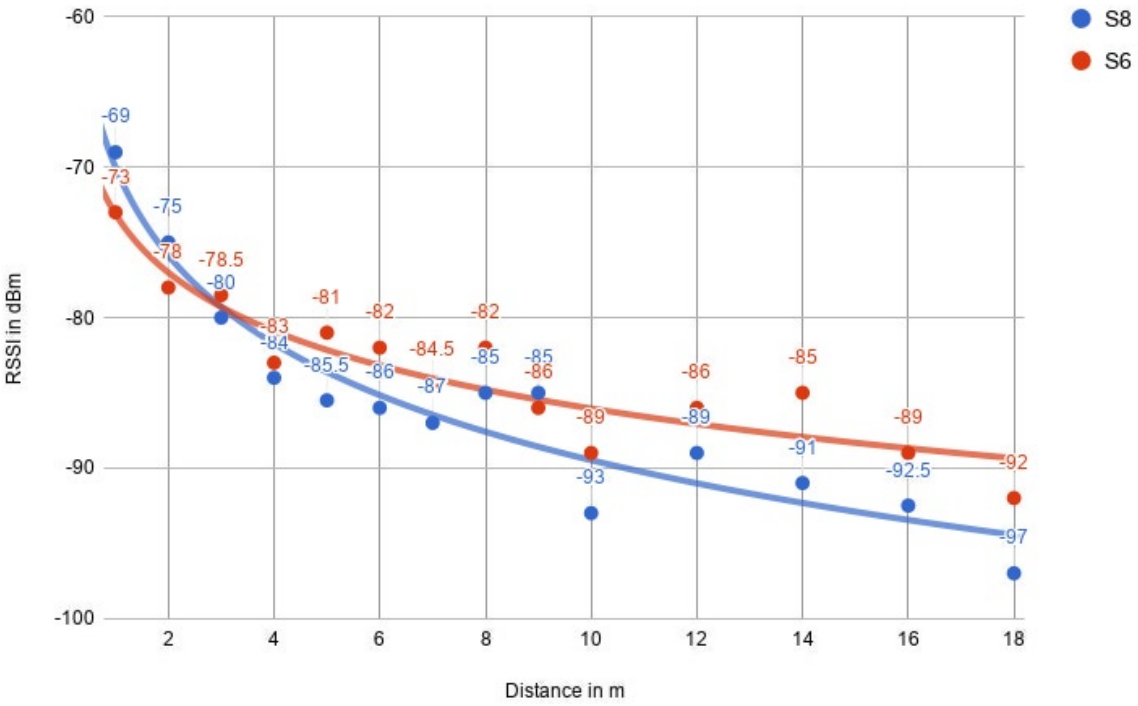


Figure 4.8: RSSI-to-distance graph

RSSI	<-95	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86
S6	20	20	20	20	20	20	20	16	14	11	9
S8	20	19	17	15	13	11	10	9	8	7	6

RSSI	-85	-84	-83	-82	-81	-80	-79	-78	-77	-76	>-76
S6	8	6	5	4	4	3	2	2	2	1	1
S8	5	5	4	4	3	3	2	2	2	2	1

Table 4.2: Mapping of each RSSI to a distance

$$dist(RSSI) = e^{(\frac{RSSI-(-73.1)}{-5.62})}$$

Equation 4.5: RSSI-to-distance function S6

$$dist(rssi) = e^{(\frac{rssi-(-70.0)}{-8.46})}$$

Equation 4.6: RSSI-to-distance function S8

4.4 Evaluation of RSSI-to-distance function

Evaluating the calculated distance against the known distance shows how accurate the RSSI-to-distance function is. The results show a difference between the two Collectors at certain distances but it also shows similarities on others (Table 4.3 and 4.4). The SD and CI in general increases with distance on both Collectors. The data shows that S6 in general calculates distances shorter than the actual distance. This is different from the S8 that calculates distances higher than the actual distance.

The graphs in Figure 4.9 show curves that are tall and narrow on short distances and then get shorter and wider as the distance increases. The width of each curve indicates how precise the estimation is and the peak shows the estimated distance. When the distance increases, the SD increases as well. The results for the

Dist	Mean	Error (m)	Error (%)	SD	CI
2	1.54	0.46	0.23	0.50	1
3	1.42	1.58	0.53	0.86	2
6	5.20	0.80	0.13	0.60	2
9	9.62	0.62	0.07	2.02	1
12	7.00	5.00	0.42	2.13	3
15	9.33	5.67	0.38	2.01	3

Table 4.3: RSSI evaluation data from S6

Dist	Mean	Error (m)	Error (%)	SD	CI
2	2.41	0.41	0.20	0.65	1
3	4.00	1.00	0.33	1.10	1
6	9.17	3.17	0.53	2.67	2
9	13.86	4.86	0.54	4.02	2
12	12.07	0.07	0.01	2.49	1
15	10.92	4.08	0.27	2.84	2

Table 4.4: RSSI evaluation data from S8

estimation is more accurate on short distances. On the S6, it is difficult to get a correct estimation on distances above 10 meters. This is seen on the graph where the curves for 12 and 15 meters is placed below the 10-meter-mark. This is not seen on the S8.

If there is a correlation in the percentage error, then it can be used to correct the RSSI-to-distance function to make it more accurate. To do that, we use linear regression on the data to find the coefficient of determination in the percentage error. The results are shown in Table refpercentage-error-table.

Phone	R^2
S6	0.07
S8	0.051

Table 4.5: Correlation in percentage error

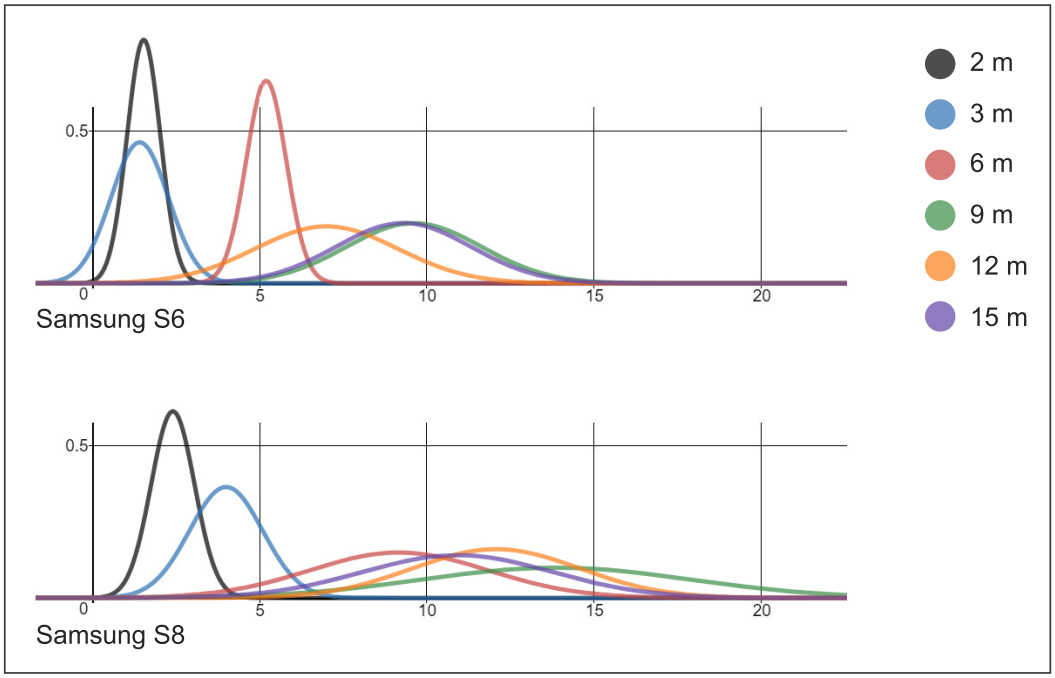


Figure 4.9: Normal distribution of distance estimations

4.5 Discussion of results

The evaluation shows that the RSSI-to-distance function is able to estimate the distance based on RSSI with an error between 0.07 and 5.67 meters or between 1 and 54 percent depending on the distance. The RSSI fluctuations dependent on the environment between the transmitter and receiver. The CI shows that the precision decreases as the distance between the transmitter and the receiver increases. This makes the calculated distance unreliable in many cases. Part of it can be ascribed to the relation between RSSI and distance because incremental changes in the RSSI increases the distance exponentially. Measuring multiple data points on the same distance can result in slightly different RSSI which results in relatively large changes in distance. We are using the median of twelve measurements for each distance. This seems like a sufficient amount, although one could argue that more data points would result in more precise data. RSSI is measured as an integer and it has a low degree of detail. Introducing decimals and calculating a mean in combination with outlier handling could potentially give more accurate results.

Phone	Distances							
S6	7	10	12	13	15	17	18	19
S8	12	14	16	18				

Table 4.6: Distances with no mapped RSSI

This would increase the RSSI spectrum such that the distance equation could be more accurate. The CIs for both devices (Table 4.3 and 4.4) show a dive at distances 9 and 12 meters where both devices have a CI of 1SD. Figure 4.8 shows a similar pattern where the RSSI decreases at the same distances. This suggests that a 3th order polynomial equation have potential for a better fit due to this behavior in RSSI.

The results show that there is a difference between the S6 and the S8. The RSSI spectrum for each Collector is calculated with the RSSI-to-distance function for distances between 1 and 20 meters (Table 4.2). The larger the spectrum is, the more precise the distance can be calculated. The S8 has a minimum and maximum RSSI of -95 and -70, which gives a spectrum of 25 RSSI levels. The S6 has a minimum and maximum RSSI of -90 and -73, which gives a spectrum of 17 RSSI level. Because the spectra are logarithmic, it has the consequence that some distances do not have a corresponding RSSI value. This results in blind spots where the function is not able to calculate the distance. The S6 has a smaller spectrum which results in more blind spots (Table 4.6).

In Chapter 5, the distance between the smartphone and the bicycle is used in order to estimate an area where the bicycle is located. In our prototype, it is important that the bicycle is within the estimated area otherwise it is a false positive. Therefore, a correlation in the percentage error can be used to improve the accuracy of the RSSI-to-distance function.

Table 4.5 shows that the coefficient of determination values are far less than 95% which means there is no satisfying correlation in the percentage error and that the average of the percentage errors is not used as a correcting coefficient. Because of

this, we decide to use the maximum observed percentage error for each Collector to adjust the RSSI-to-distance function. This results in a calculated distance that is longer than the actual distance but not shorter which will make false positives in the estimated area less likely. The maximum percentage error for S6 is 0.53 and for S8 it is 0.54.

5 Finding the location of the bicycle

The purpose of our Crowdfinding system is to use smartphone sensors that are naturally distributed across a city to find the location of a bicycle. In the following Chapter, we will examine how GPS and Bluetooth can be combined to estimate an area where the bicycle is within; and furthermore, we want to examine how small that estimated area can get. Because the system uses GPS, and because Bluetooth signals is affected by the surroundings, we conduct the experiments in the city under authentic conditions. The experiments are conducted at three locations in order to test how the system performs under different circumstances. We use two different methods, $RSSI_m$ and $FIXED_m$, and compare their performance. The performance is defined as how small the estimated area can get while containing the bicycle. $RSSI_m$ calculates a distance between the smartphone and the bicycle which is between 1 and 20 meters. The $FIXED_m$ method uses a distance of 20 meters no matter what the RSSI value is. We examine how these two methods affect the reliability of the estimated area.

5.1 Methodology

In order to calculate the estimated area of the bicycle, we need the phone's GPS location and accuracy together with the distance to the bicycle estimated from the

Bluetooth RSSI. We use the S6 and S8 to collect the location data on three different locations. Then we download the location data and calculate an area for each new location data point. After all the areas are calculated, we evaluate how accurate the estimated area is as the amount of data points increases. The estimation of the area is done with the two different distance methods described in Chapter 4. One where the distance between the phone and the bicycle is 20 meters called FIXED_m and another where the distance is calculated based on the RSSI which can vary from 1 to 20 meters called RSSI_m. We compare the results from the two methods to show how the calculated distance based on RSSI affects our Crowdfinding system.

Android's FusedLocationProvider provides the position of the smartphone in geographical coordinates and GPS accuracy which is defined in meters. The Bluetooth RSSI from the bicycle is used to estimate the distance to it. Combined, this results in a circular area that we define as a location data point. The location data point can be described as a circle with a center point and a radius. Figure 5.1 shows a location data point as a circle where the center point is the GPS coordinates, the inner circle corresponds to the GPS accuracy, and the distance between the inner and outer circle is the distance between the smartphone and the bicycle calculated by RSSI_m or FIXED_m. Each location data point is independent which means that data points from different smartphones can be used in the same analysis to provide a more accurate estimated area. This is a central concept in our Crowdfinding system.

In a real Crowdfinding system, a server is used to calculate areas as new location data points become available. Our prototype uses the Controller to calculate the areas instead of a server because it gives us the control we need to perform the experiments correctly.

We collect data by placing the bicycle on three predetermined locations in Copenhagen city and write down the coordinates of the bicycle (Table 5.1). The streets are chosen to examine how our Crowdfinding system performs under real life con-

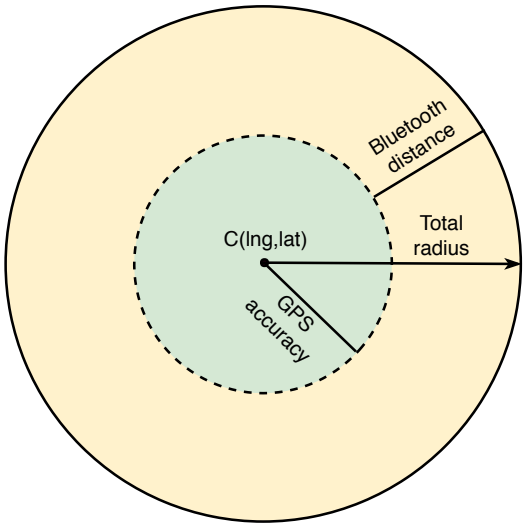


Figure 5.1: Visual representation of a location data point

Address (Copenhagen, Denmark)	Coordinates (Lat, Lng)	Description
Amagertorv	55.678842, 12.579377	Large, busy square
Havnegade 47	55.677984, 12.592026	Average street
Sværtegade 5	55.681083, 12.580567	Narrow, busy street

Table 5.1: Experiment locations

ditions. The radio signals from GPS and Bluetooth is affected by the environment which influence how the system performs. The first street is Sværtegade 5 which is a narrow street in the city center that is busy with people. The conditions of this location should have an effect on the radio signals we receive both because of the large surrounding buildings and because of the many people in the street. The second place is on a square in the city, Amagertorv, which is an open area with a lot of people walking around. In this case, surrounding buildings should have less effect on the radio signals but they can still be affected by the crowd. The last location is Havnegade 47. This street has an average width for Copenhagen and few people. The chosen part of the street has buildings on each side. We choose this street as representative of average conditions where we expect the radio signals to be less affected by the surrounding environment. We expect these results to be more accurate than those from the narrow street but similar to the once from the square.

After the location data is collected, it is downloaded to the Controller in order to create the estimated areas. A new area is calculated each time there is a new location data point. This is because we want to show how the estimated area changes as new data points are added.

Every data point represents an area which the bicycle can be within. This means that the area where the bicycle can be within gets larger the more data points we collect. Figure 5.2 shows six data points as circles and the bicycle's location marked as a red dot. In a naive approach, the area covered by all the circles is the area where the bicycle could be. The goal is to reduce the area by combining the information from all the circles. This presents several challenges. First is to find which circles are more likely to contain the bicycle than others. In Figure 5.2, the circle in the top left corner does not intersect with any other circles and it does not contain the bicycle. Also, some circles have less intersections than others as with the circle closest to the bottom. We want to find the area where most circles intersect and then use two intersection points inside that area to create an estimated area where the bicycle is within. In the area where most circles intersect, we assume this is the area that has the highest probability of enclosing the bicycle. We base this on the assumption that each location data point encloses the bicycle. If a data point does not intersect with other data points, then we assume that the specific data point is an outlier, as seen in Figure 5.2.

No algorithm exists for finding the intersection area of circles where each circle can have any given position, any given radius, and any given amount of intersections. Therefore, we design our own algorithm to solve this issue which will be described in this Chapter.

The calculation starts with a list of circles created from the location data points. If there is only one circle then the estimated area is equal to the area of that circle. If there are more than one circle, then we use our algorithm to calculate an estimated area. The algorithm is divided into eight steps (Listing 5.1).

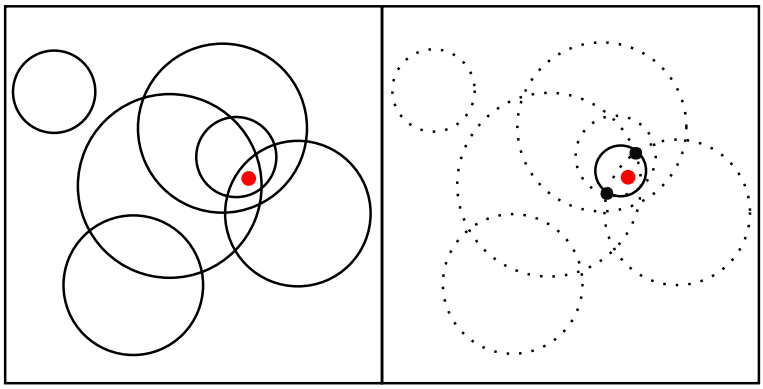


Figure 5.2: Location circles and a bicycle

```
1  private AreaEstModel calculateArea(List<Circle> circles) {
2    ...
3    // Step 1
4    final List<CircleWithIntersections> intersections1 =
         createIntersections(circles);
5    // Step 2
6    final CircleWithIntersections cwi1 =
         findCircleWithHighestIntersections(intersections1);
7    // Step 3
8    final List<Point> points1 = generatLisePointsToEvaluate(cwi1.self);
9    // Step 4
10   final List<Circle> circles1 = generateCirclesWithMostCommonIntersections(cwi1,
         points1);
11   // Step 5
12   final List<Circle> circles2 = removeEnclosingCircles(circles1);
13   // Step 6
14   final List<Point> points2 = findAllIntersectionPoints(circles2);
15   // Step 7
16   final List<Point> points3 = findPointsWithinAllIntersections(circles2,
         points2);
17   // Step 8
18   final AreaEstModel area1 = generateEstArea(points3);
19 }
```

Listing 5.1: Algorithm to estimate an area for a lost bicycle

Step 1: First it loops through the list of circles. For each circle, it calculates which other circles it intersects with. All the circles it intersects with are stored in a list (Listing 5.2).

```
for (Circle self : circles) {
    for (Circle circle : circles) {
        if (!(self == circle)) {
            if (intersecting(self, circle)) {
                result.add(circle);
   ...
}
```

Listing 5.2: Finding intersecting circles

Step 2: After finding all the intersecting circles, it finds the circle that has the most intersections with other circles (Figure 5.3). This is because the area with most overlapping circles is included in the circle that has the most intersections and that is why we want to find this circle. In the scenario where it finds multiple circles with the same intersection count or where no circles intersect, it chooses the first one (Listing 5.3). This is mainly for optimizing the algorithm such that the next step uses less computation.

```
for (int i = 0; i < intersections.size(); i++) {
    CircleWithIntersections cwi = intersections.get(i);
    if (highest.intersections < cwi.intersections) highest = cwi;
}
```

Listing 5.3: Comparing intersection count of circles

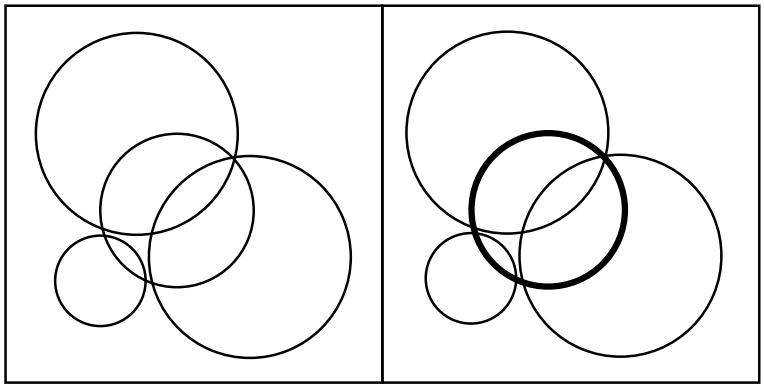


Figure 5.3: Finding the circle with most intersections

Step 3: The goal is to find a single point inside the area where most circles overlap. That point is important to find because it is used to find the circles that overlap the point and therefore the area we are looking for. We know that the point has to be inside the circle with most intersections. This circle is found in step 2. To find

that point, it will first create a certain amount of points inside the circle and then evaluate each one to find which point has the most circles enclosing it, which is the goal.

In this step, it generates a list of points inside the circle. The points are generated from the circle with most intersections. First it creates a square that circumscribes the circle. Then it creates a list of points inside the square that are evenly spread out with a distance of one-hundredth of the square length apart . For example, if the radius of the circle is 50 meters then the square is 100 by 100 meters and the points generated will be spread out evenly 1 meter apart from each other. After that, it evaluates each point to see if it is inside the circle. The points that are inside the circle are stored in a list and returned (Listing 5.4).

```
int factor = 100;

double xStart = circle.getC().x - circle.getDistInDegrees();
double xEnd = circle.getC().x + circle.getDistInDegrees();
double xIncrement = Math.abs(xEnd - xStart) / factor;

double yStart = circle.getC().y - circle.getDistInDegrees();
double yEnd = circle.getC().y + circle.getDistInDegrees();
double yIncrement = Math.abs(yEnd - yStart) / factor;

for (double x = xStart; x < xEnd; x += xIncrement) {
    for (double y = yStart; y < yEnd; y += yIncrement) {
        if (isPointInsideCircle(x, y, circle)) result.add(new Point(x, y));
    }
}
```

Listing 5.4: Looping through coordinates and evaluating each point

Step 4: In this step, the goal is to find the circles with most common intersections because these circles will be used to find the area where most circles overlap. It starts with the circle with the most intersections and the circles it intersects with (Figure 5.3). Then it takes the list of points calculated in step 3 (Figure 5.4) and evaluates each point to see how many circles that encloses it. Figure 5.5 shows the number of circles intersecting at each given area. Each area has multiple points which evaluate to the same value (Figure 5.4). In Figure 5.5, we write the value

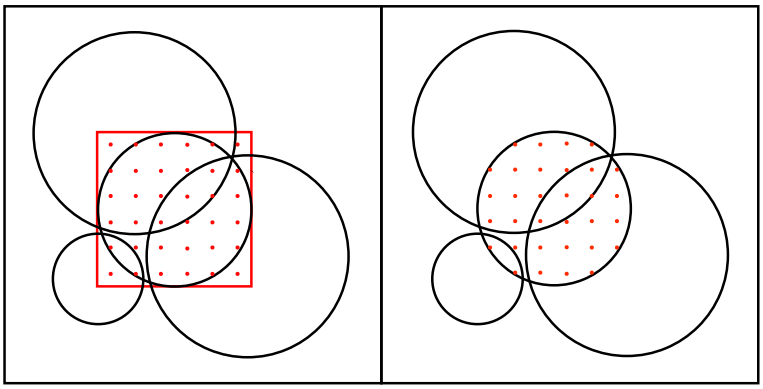


Figure 5.4: Generated points to evaluate

for one of these points for each area. In this example, the highest count is 3. When it has found the point with the highest count, it finds the circles that contain the point and stores them in a list (Listing 5.5). If multiply points evaluate to the same value but do not have the same circles enclosing them, it chooses the first point.

```
1  int highestCount = 0;
2  Point bestPoint = points.get(0);
3
4  for (Point point : points) {
5      int count = 0;
6      for (Circle circle : circles) {
7          if (isPointInsideCircle(point, circle)) count++;
8      }
9      if (highestCount < count) {
10         highestCount = count;
11         bestPoint = point;
12     }
13 }
14
15 for (Circle circle : circles) {
16     if (isPointInsideCircle(bestPoint, circle)) result.add(circle);
17 }
```

Listing 5.5: Finding the circles with most shared intersections

Step 5: We still want to find the area with most overlapping circles. This step removes enclosing circles. With the list of circles from step 4, there is a change that some of the circles entirely enclose another circle. Figure 5.6 shows that when a circle surrounds another circle, it does not add any information to the final area we want to calculate and therefore we can ignore it or remove it. The algorithm removes these circles from the list before continuing (Listing 5.6).

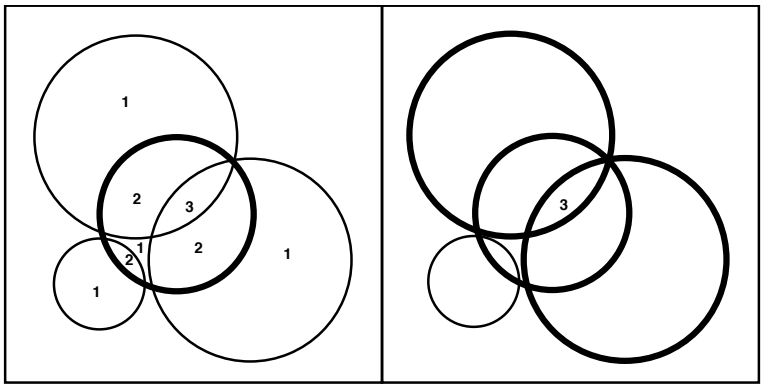


Figure 5.5: Counting the amount of circles overlapping each area

```
for (Circle c1 : circles) {
    for (Circle c2 : circles) {
        if (isOneCircleInsideAnother(c1, c2)) {
            if (c1.getDistInDegrees() < c2.getDistInDegrees()) circles.remove(c2);
            else circles.remove(c1);
            return removeEnclosingCircles(circles);
...
}
```

Listing 5.6: Removeing enclosing circles

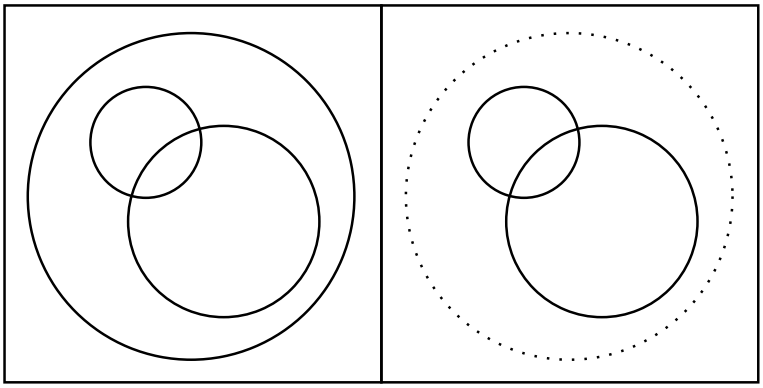


Figure 5.6: Removing circles that enclose another circle

Step 6: Now we have the final list of circles that we can use to estimate the area for the bicycle's location. In this step, it finds the intersection points for all pairs of circles and stores them in a list (Figure 5.7). It does that because the intersection points are used to find and calculate the smallest area possible. To find the intersection points of two circles, it uses the steps in Listing 5.7 [20].

```java
final double x1 = c1.getC().x;
final double y1 = c1.getC().y;
final double r1 = c1.getDistInDegrees();

final double x2 = c2.getC().x;
final double y2 = c2.getC().y;
final double r2 = c2.getDistInDegrees();

final double d = distanceBetweenTwoPoints(x1, y1, x2, y2);
final double d1 = (r1 * r1 - r2 * r2 + d * d) / (2 * d);
final double h = Math.sqrt(r1 * r1 - d1 * d1);

final double x3 = x1 + ((d1 * (x2 - x1)) / d);
final double y3 = y1 + ((d1 * (y2 - y1)) / d);

final double x4 = x3 + ((h * (y2 - y1)) / d);
final double y4 = y3 - ((h * (x2 - x1)) / d);
final double x5 = x3 - ((h * (y2 - y1)) / d);
final double y5 = y3 + ((h * (x2 - x1)) / d);

final Point p4 = new Point(x4, y4);
final Point p5 = new Point(x5, y5);
```

Listing 5.7: Finding the two intersection points of two intersecting circles

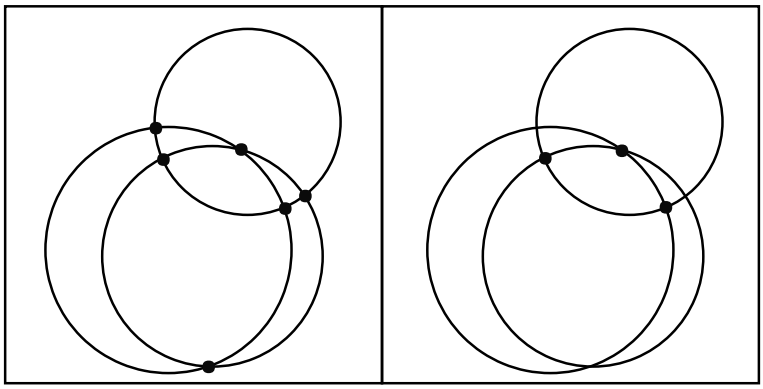


Figure 5.7: Intersection points of circles

Step 7: With the list of all intersection points, it finds the points that are within all the circles (Listing 5.8). These points can be described as boundaries of the final area we want to calculate (Figure 5.7).

```
for (Point point : points) {
    boolean withinAll = true;
    for (Circle circle : circles) {
        if (!isPointInsideCircle(point, circle)) {
            withinAll = false;
            break;
    ...
    if (withinAll) result.add(point);
}
```

Listing 5.8: Finding the intersection points that are within all circles

Step 8: The final step is to calculate the estimated area where we think the bicycle is within. The algorithm uses the list of points from step 7 and finds the two points furthest apart as depicted in Figure 5.8. Then it chooses the points farthest apart in order to ensure that the estimated area covers the whole area of the overlapping circles. To calculate the area, it uses the distance between the two points as diameter and the midpoint as centrum (Listing 5.9). Now the final estimated area is calculated and this is where the bicycle is within.

```
Point firstPoint = points.get(0);
Point secondPoint = points.get(1);
double highestDistance = 0.0;

for (int i = 0; i < points.size() - 1; i++) {
    Point p1 = points.get(i);

    for (int j = i + 1; j < points.size(); j++) {
        Point p2 = points.get(j);
        double distance = distanceBetweenTwoPoints(p1, p2);

        if (highestDistance < distance) {
            highestDistance = distance;
            firstPoint = p1;
            secondPoint = p2;
...
double estLng = (firstPoint.x + secondPoint.x) / 2;
double estLat = (firstPoint.y + secondPoint.y) / 2;
double radius = highestDistance / 2;
Circle circle = Circle.newCircleFromDegrees(new Point(estLng, estLat), radius);
```

Listing 5.9: Generating the estimated area from a list of points

The correctness of the estimated area can be evaluated by examine if the bicycle is inside the area or not. This is done by comparing the radius of the area to the

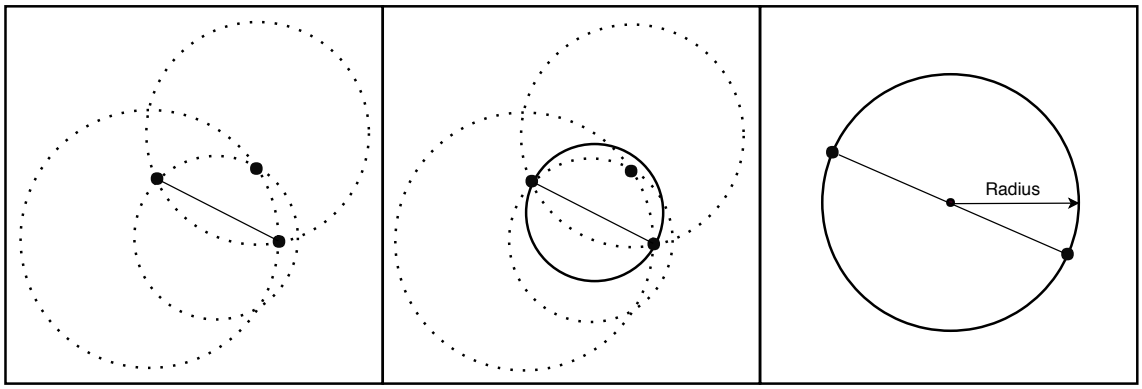


Figure 5.8: Estimated area generated from two intersection points

distance between the center and the bicycle. The bicycle is inside the area if the distance is smaller than the radius. The estimated area tends to get smaller as the amount of data points grow. The larger the area is, the more difficult it is to find the bicycle inside of it. The smaller it gets, the less likely is it that the bicycle is inside of it. We want to examine how small the estimated area can get while ensuring that the bicycle is inside. It is a false positive when the bicycle is not inside the area. The smallest area is found by examining the estimated areas over time. A new estimated area is calculated each time a new location data point is added. Each area is evaluated against the bicycle's position and stored. We expect estimated areas based on RSSI_m will be smaller in size than those based on FIXED_m.

5.2 System design and implementation

The prototype is an extension to the prototype developed in Chapter 4. New activities have been added to the Controller application and we have implemented new methods in the Collector application. Furthermore, the database has been extended with two new branches. The following Chapterwill elaborate on these additions.

5.2.1 Database and data structure

We have added two new branches to the database structure (Figure 5.9). The first is the *Location* branch which is where we store location data points collected from the smartphones. Each data point contains the GPS coordinates and the distance between the smartphone and the bicycle. The position of the bicycle is known beforehand. It also contains the phone model and a server timestamp which is used in the analysis. The other branch is the *Area* branch where the estimated areas are stored. Each area contains a timestamp which is used for ordering. Furthermore, they contain the coordinates of the bicycle which is used to evaluate if the bicycle is inside the area. This is stored in the *InArea* boolean.

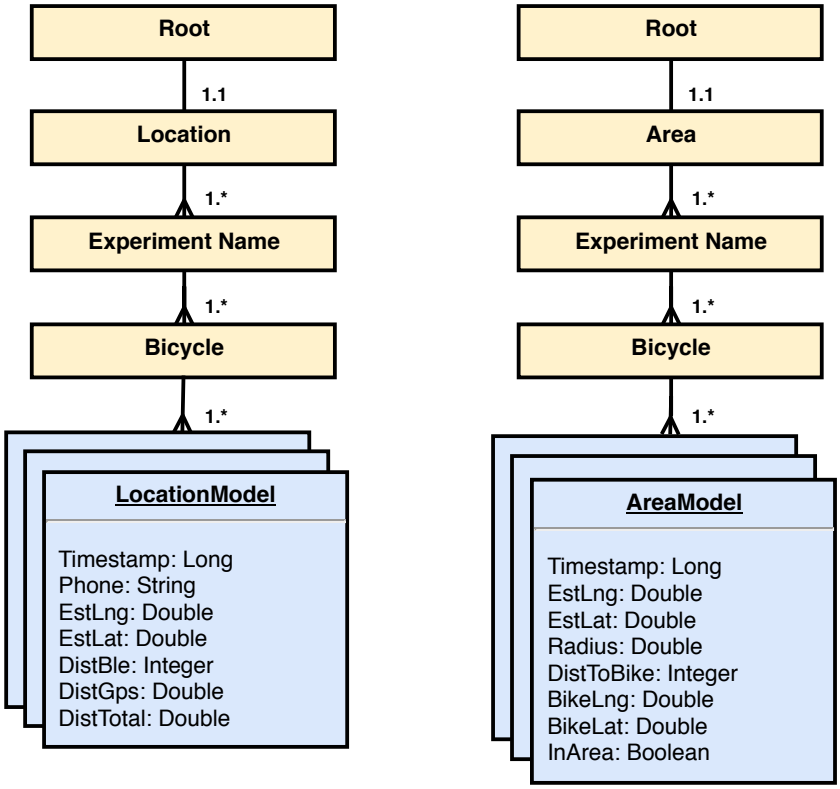


Figure 5.9: Structure for storing location data points and estimated areas

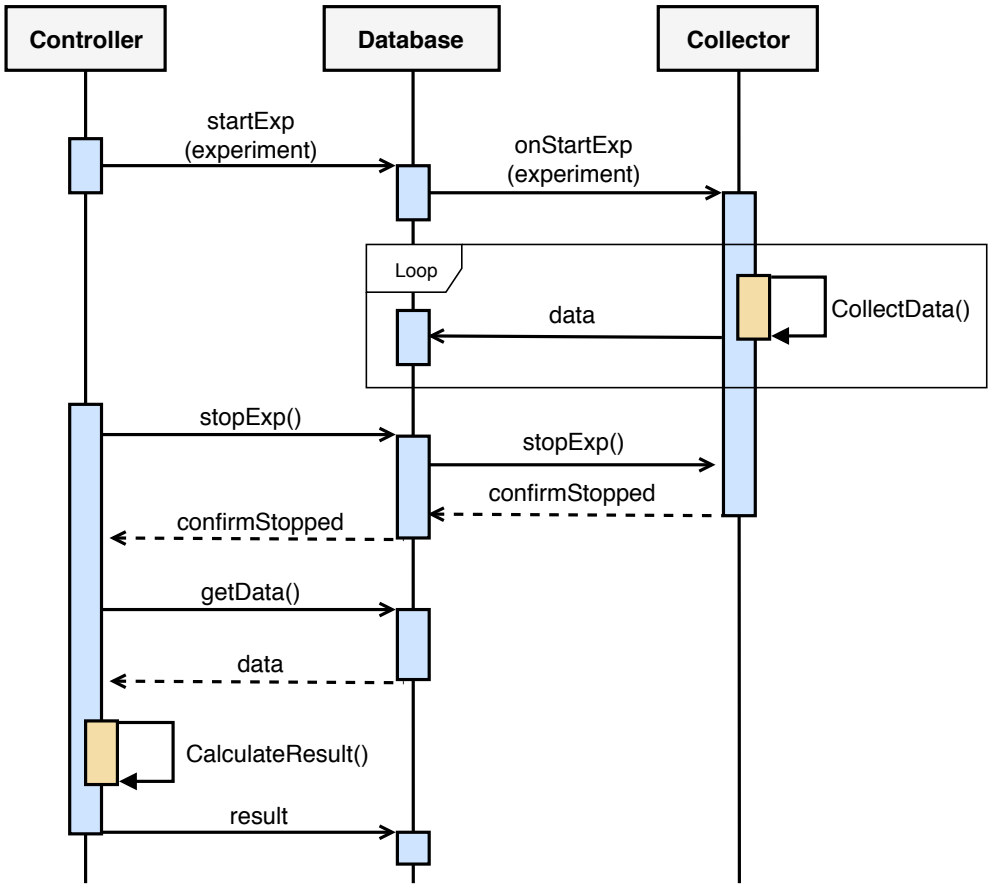


Figure 5.10: Sequence diagram of the data collection and calculation of area

5.2.2 Controlling the system and collecting data

The prototype is controlled from the Controller which can start and stop experiments on the two Collectors. This is different from previously where the Collector could stop the experiment on its own. As seen in Figure 5.10, when the Controller starts the experiment, the Collector starts locating its position using GPS and listening for Bluetooth signals from the bicycle. This happens in a loop and each time a data point is collected, it is send directly to the database. When we decide that we have enough data, the experiment is stopped from the Controller.

The Collector checks the name of the device each time it gets a Bluetooth scan result. If the name matches our bicycle, then the Collector gets the GPS coordinates and accuracy as well as the RSSI value from the scan result (Listing 5.10). Two distances

are calculated, one based on $RSSI_m$ and one based on $FIXED_m$ which is used to create two different location data points. These are stored in the database.

```
private void addScanResult(ScanResult result) {
    String name = result.getScanRecord().getDeviceName();
    Integer rssi = Math.abs(result.getRssi());

    if (name.equals(BIKE_LOCK_NAME)) {
        Location location = mLocationCallback.getLocation();

        double distRssi = rssiDistanceValues[rssi];
        double distProx = 20.0;
        double distGps = location.getAccuracy();
        double x = location.getLongitude();
        double y = location.getLatitude();

        LocationModel rssiModel = new LocationModel(
        ServerValue.TIMESTAMP, PHONE_MODEL, x, y, distGps, distRssi);
        LocationModel proxModel = new LocationModel(
        ServerValue.TIMESTAMP, PHONE_MODEL, x, y, distGps, distProx);

        sendLocationToServer("RSSI-" + locationExpName, rssiModel);
        sendLocationToServer("PROX-" + locationExpName, proxModel);
    }
}
```

Listing 5.10: Handeling of the ScanResult

After conducting the experiments, the location data is downloaded to the Controller and stored. Listing 5.11 shows how the Controller calculates an estimated area for each new data point. When all the areas are calculated, they are stored in the database.

```
for (int i = 1; i < mLocationModels.size(); i++) {
    List<LocationModel> tempList = null;

    if(i == mLocationModels.size() - 1) tempList=mLocationModels;
    else tempList=mLocationModels.subList(0, i);

    List<Circle> circles = new ArrayList<>();

    for (int j = 0; j < tempList.size(); j++) {
      circles.add(tempList.get(j).toCircle());
    }

    final AreaEstModel area = calculateArea(circles);
    mAreaEstModels.add(area);
    sendAreaToServer(mExpName, area);
}
```

Listing 5.11: Calculating multiple estimated areas

5.2.3 User interface design

The Collector keeps the same UI as previously because it conforms to the same functionality as before. The Controller has two new activities. The Main menu has been extended with extra buttons and the FunctionCoefficients has been extended with an input field for the function error correction. The first new activity is for controlling the experiments. It has an input field for the experiment name and a start/stop button. The second new activity is for calculating and analyzing the data. It contains a drop-down menu for selecting experiments and a map that displays location data and estimated areas from the database (Figure 5.11). Listing 5.12 shows how location data from the two Collectors are plotted on the map and how the data points get a color that matches their Collector.

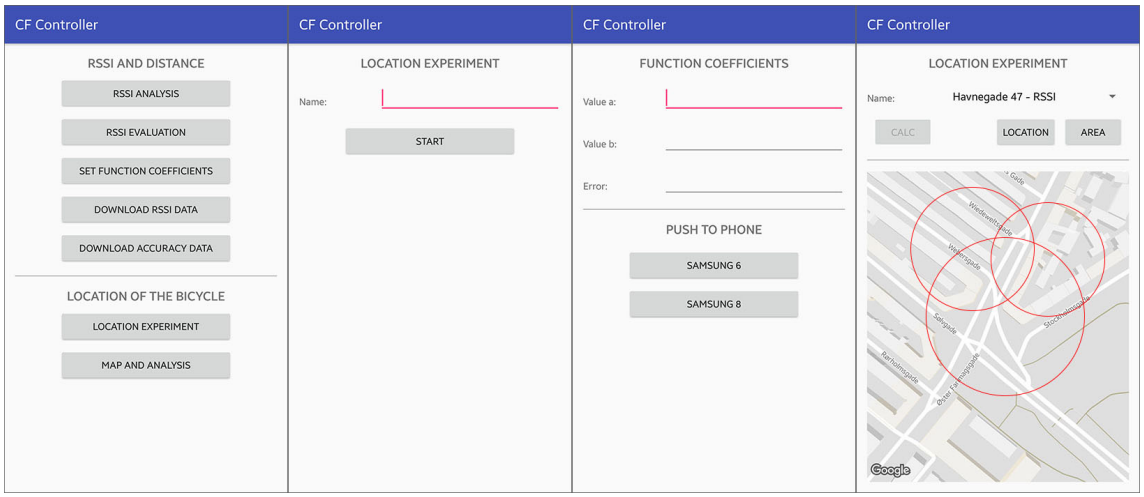


Figure 5.11: Extended Controller user interface

```
for (LocationModel location : locations) {
    int color = Color.LTGRAY;
    if (location.getPhone().equals(PHONE_NAME_SAMSUNG_6)) color = Color.MAGENTA;
    if (location.getPhone().equals(PHONE_NAME_SAMSUNG_8)) color = Color.GREEN;

    mGoogleMap.addCircle(new CircleOptions()
            .center(new LatLng(location.getCoordinateY(),
                location.getCoordinateX()))
            .radius(location.getDistTotal())
            .strokeColor(color)
            .strokeWidth(3f));
}
```

Listing 5.12: Drawing location data points on Google Maps

5.3 Area analysis

We divide the analysis into two parts; one about $RSSI_m$ and the other about $FIXED_m$. The data we analyze are the estimated areas calculated from the location data points on the three different locations. In Figure 5.13 and 5.12, the data are plotted on a map and divided into location data and estimated areas for each of the experiments. The two Collectors, S6 and S8, are highlighted with different colors. Each estimated area is colored according to its order in logical time. Green color represents the first estimate and pink represents the last. Normally, the Crowdfinding system only produces one estimate which changed each time a new

Street	Lowest radius, bicycle inside	Highest radius, bicycle outside
Amagertorv	9.7	N/A
Havnegade 47	10.1	6.8
Sværtegade 5	22.4	10.7

Table 5.2: $RSSI_m$ area radius limits

data point is available. In this prototype, we show the estimated areas over time to give an insight in how the system works. The actual bicycle position is depicted as a red dot.

The estimated area is evaluated by analyzing if the bicycle is inside the area. Table 5.2 and 5.3 show the data from the three experiments. For each of the experiments, we look at the smallest area radius where the bicycle is inside the area and the largest radius where the bicycle is not inside. These values are interpreted as the limits for how small and accurate we can estimate a correct area.

5.3.1 Crowdfinding based on $RSSI_m$

In Figure 5.12, we see a pattern in the three experiments. When the estimated area gets smaller, it is less likely that the bicycle is inside. This method is able to accurately estimate an area down to a radius of 10 meters in two of the experiments but on Sværtegade, this limit is 22 meters (Table 5.2). At the city square, Amagertorv, all estimates are correct. We see that the initial estimates are large and covers the whole square. Later estimates are smaller and on target. At Havnegade, we see the same behavior but the last estimates do not include the bicycle. Here the initial estimates are smaller than what we see on Amagertorv and Sværtegade.

5.3.2 Crowdfinding based on $FIXED_m$

With $FIXED_m$, we see that the estimated areas are similar across the different experiments. All of the location data points include the position of the bicycle.

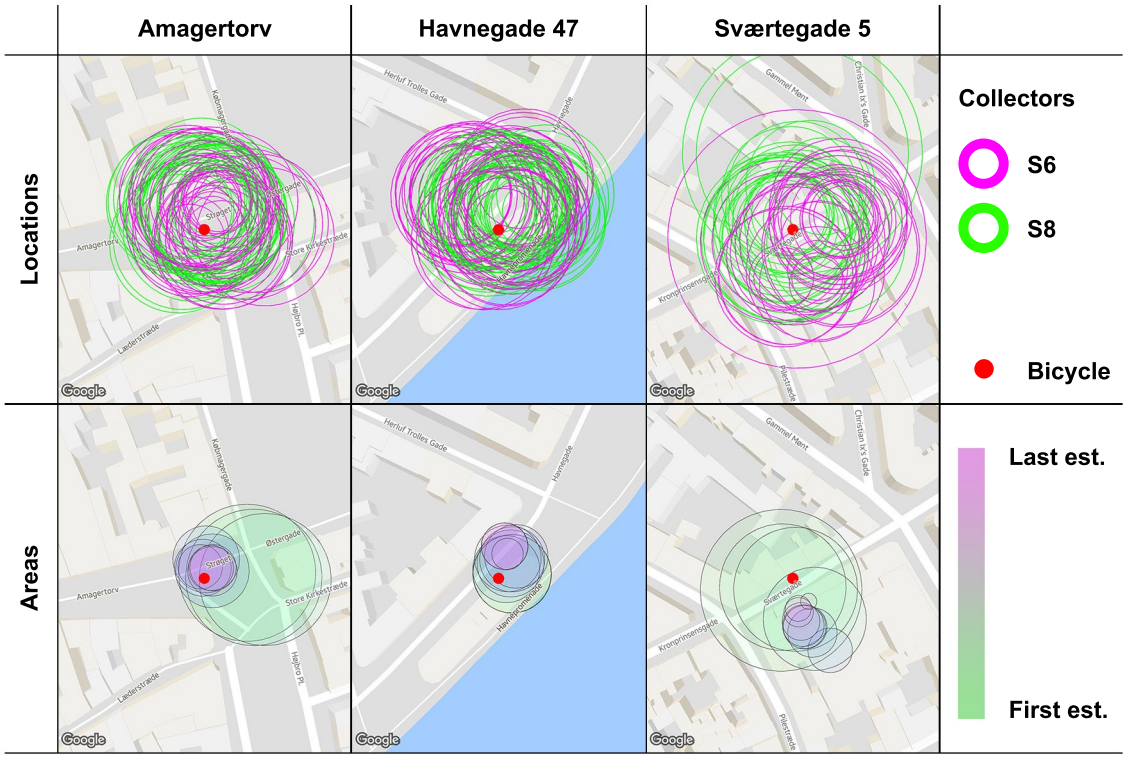


Figure 5.12: Location data and estimated areas with $RSSI_m$

Street	Lowest radius, bicycle inside	Highest radius, bicycle outside
Amagertorv	20.3	N/A
Havnegade 47	20.0	N/A
Sværtegade 5	17.8	N/A

Table 5.3: $FIXED_m$ area radius limits

Furthermore, in all three experiments, the system successfully locates the area where the bicycle is (Figure 5.13). The smallest radius for an estimated area is 17.8 meters which is twice the size of the one from $RSSI_m$ which had a radius of 9.7 meter. In general, it seems that this system will estimate an area of about 20 meters in radius no matter the location (Table 5.3).

5.4 Discussion of results

The prototype is designed to continuously calculate a new area when new data points are added. With each new data point, the estimated area either stays the

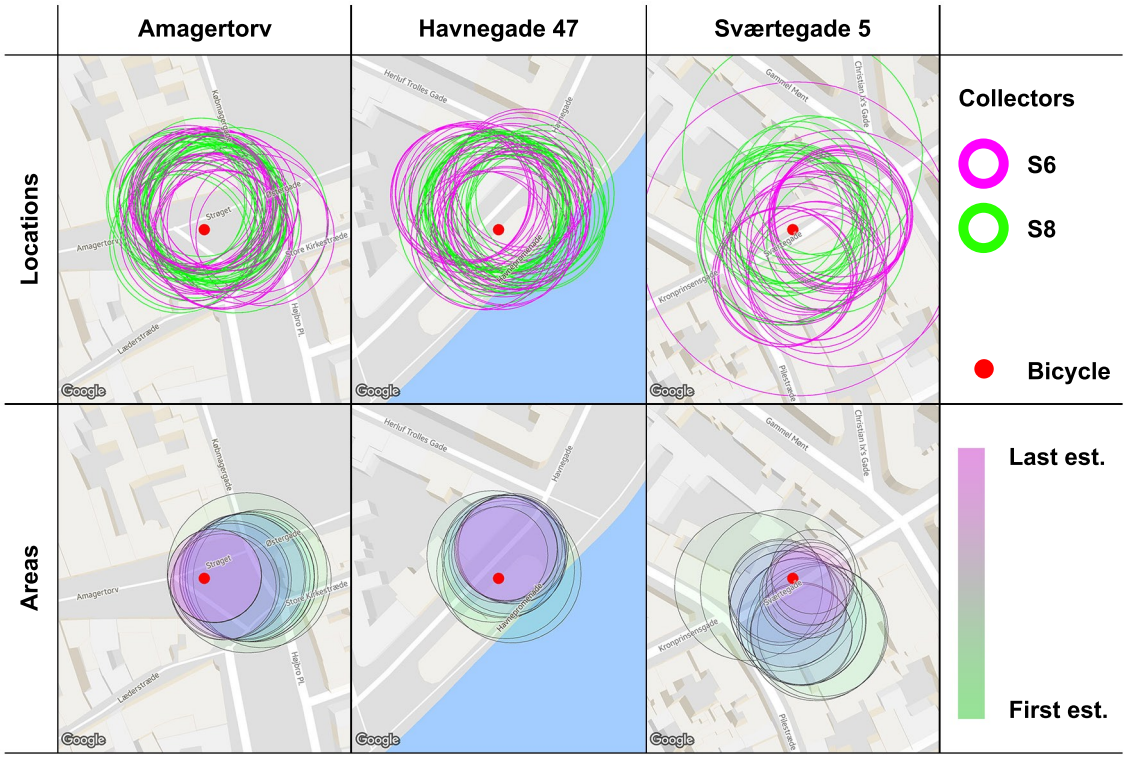


Figure 5.13: Location data and estimated areas with FIXED_m

same or gets smaller. The results show that this is what happens. When we get more data points it is more likely that the circles are spread across a larger area. Intersecting circles which are spread apart result in a smaller estimated area than circles which are closely aligned. When using RSSI_m, this has a larger effect because the location data points can get significantly smaller. This results in smaller estimated areas compared to FIXED_m. Calculating a smaller area is good because then it is easier to find the lost bicycle but only as long as the bicycle is inside that area. The problem with RSSI_m is that it is very susceptible to the surrounding environment. As we see on Havnegade and Sværtegade, the estimated areas do not contain the bicycle when the area is smaller than 10 and 20 meters respectively. It is known that Bluetooth signals are affected by the surrounding environment, especially if there are buildings and people that can reflect or block the radio waves. At Amagertorv, there were no buildings in close vicinity of the bicycle and here we see the best results. Both methods, RSSI_m and FIXED_m, uses the same GPS signal which means that the difference in accuracy can be ascribed to how we use RSSI to determine distance.

The $FIXED_m$ method is more reliable than $RSSI_m$ because the bicycle position is always inside the estimated area. This is partly due to the larger location data points, which gives a higher probability of containing the bicycle. The main reason this method is less affected by the environment is that is does not utilize RSSI as a measure of distance. The minimum radius of a location data point in this method is 30 meters; 10 meters from GPS accuracy and 20 meters from the maximum Bluetooth distance. Larger location data points will often result in larger estimated areas because the circles will overlap to a larger degree. This is seen on Figure 5.12 at Havnegade. Based on our results, we recommend that the $FIXED_m$ method is the best for Crowdfinding.

We have identified two cases where the system does not work as intended. In the first case, the system does not calculate an area if the location circles are aligned as shown in Figure 5.14. This case is characterized by having an area in the middle of the circles that is not covered by any circle. This is because of step 7 in our algorithm where it finds the intersection points that are inside all of the circles. No point is inside all of the circles which results in no area. We have not found any occurrences of this in our three experiments.

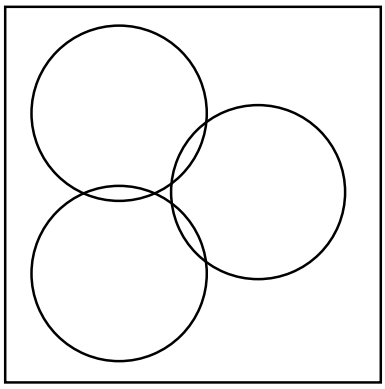


Figure 5.14: Undetectable area due to arrangement of circles

In the second case, the system calculates an area that is smaller than expected which can lead to a false positive. As we see in Figure 5.15, the two dotted circles are location data points and the solid circle is the estimated area. When the location circles are arranged like this, the estimated area only covers a small portion of the

area inside the two circles. This results in a wrong estimated area. We see one case of this scenario on Sværtegade with $RSSI_m$.

A future iteration of the algorithm should be able to handle the two aforementioned cases to make it more reliable.

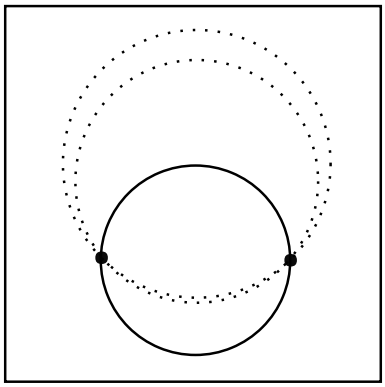


Figure 5.15: Similar location circles causing too small area estimate

6 Conclusion

In this Chapter, we will answer the three main questions of the thesis which are:

1. Is Crowdfinding useful for finding lost bicycles?
2. How small and accurate can an estimated area get?
3. Is $RSSI_m$ more suitable for Crowdfinding than $FIXED_m$?

To answer question 2 and 3, we will look at the results from Chapter 5. Here we found that $FIXED_m$ is more reliable than $RSSI_m$ when it comes to accurately estimating an area where the bicycle is located. $RSSI_m$ was able to produce smaller estimated areas but the bicycle would not always be inside of them. The smallest estimated areas with $FIXED_m$ are around 20 meters in radius and we saw that they always contained the bicycle. It is important that the area always contains the bicycle because otherwise, it will be difficult to find bicycles using the system. With $RSSI_m$ we saw areas which had a radius of 10 meters but they did not always contain the bicycle.

When we take a look at our implementation of Crowdfinding and want to answer whether or not it is useful for finding lost bicycles, we have to take a closer look at our algorithm and the two methods for positioning a bicycle.

In Chapter 4, we examine how RSSI can be used to measure the distance between a smartphone and a bicycle. The results showed that RSSI can be unreliable because

of fluctuation in the Bluetooth signals. To account for the fluctuation, the RSSI-to-distance function was correct by multiplying the distance with the maximum percentage error from our evaluation. This should result in a distance larger than the actual distance and prevent false positives in the area estimation. The results from $RSSI_m$ in Chapter 5 are different than what was expected. The estimated areas had false positives because the bicycle's position was not included in some of the areas. Each area is estimated from the location data points and here we see that some of the points do not include the bicycle in the first place. This is a problem because it affects the outcome of the estimated area. Because we already examined the RSSI at various distances and corrected for the fluctuations, we question the reliability of the GPS accuracy provided by Android. Furthermore, we did not examine the GPS the same way we examined the Bluetooth. The GPS accuracy provided by Android's FusedLocationProvider comes with a confidence of 68% which could be the reason why some of the location data points were inaccurate. This confidence of 68% can be sufficient in many use cases where the position gets replaced by a new one every time it becomes available but in a Crowdfinding system like ours, we see that it results in unreliable data points. Additionally, we know that GPS is highly susceptible to buildings and other objects in the surrounding environment. GPS requires a direct line of sight to four satellites. If buildings block the reception, then the GPS will cease to work accurately. The FusedLocationProvider combines location information from many different sources to estimate the most accurate position. When the GPS signal is not available, it uses the last known position together with the accelerometer to predict a new position. This method is called *Dead reckoning* and it comes with uncertainty because it is difficult to accurately account for small differences in speed and direction. At Sværtegade, some location data points are far from the locations we collected data on. This could be because the GPS is not receiving a signal and therefore uses dead reckoning. Generally, $RSSI_m$ is unreliable to use for Crowdfinding because the inaccuracy in RSSI measurements together with the inaccuracy of GPS result in false positives when creating location data points.

$FIXED_m$ used a Bluetooth distance of 20 meters no matter the RSSI value. This resulted in location data points that were accurate in all experiments conducted. The method is less susceptible to fluctuations in the Bluetooth signal. An advantage with $FIXED_m$ is that it is not necessary to calibrate each device in order to find the hardware specific RSSI-to-distance function which otherwise is necessary with the $RSSI_m$ implementation. Therefore, $FIXED_m$ is better suited for a crowd and to accurately collect location data points for our Crowdfinding system.

Our Crowdfinding algorithm is a proposed solution to how multiple circular data points from many different smartphones can be combined to increase the accuracy of a location. The algorithm uses circles in order to calculate the area where most location data points overlap. This area can be interpreted to be the most likely location of the bicycle and therefore an estimate. Crowdfinding is useful for finding lost bicycles because it can successfully locate a bicycle in a large city down to an area with a radius of 20 meters. When we compare the size of the area to the average size of a bicycle, we believe, this is a sufficient precision for finding a bicycle.

6.1 Discussion and future research

This thesis has proposed an algorithm and an Android prototype for Crowdfinding. Generally, this works well for finding lost bicycles but if we take a critical look at our implementation we find that the prototype has some limitations that could be improved. One part is how the distances are corrected in the RSSI-to-distance function. In our implementation, we calculate the absolute percentage error for all the calculated distances in relation to the actual distance (Table 4.3 and 4.4). Then we use the largest percentage error as the correcting factor without differentiate between positive and negative values of the percentage error. This could be improved by using the percentage errors that are positive. Positive percentage

errors represent calculated distances that are lower than the actual distance and these are the distances we want to correct for.

Another part is how we use Bluetooth RSSI to calculate a distance. Bluetooth can be used in different ways to calculate the distance between a receiver and a transmitter. One way is by measuring the time it takes to send a packet between two devices. Radio signals travel with the speed of light and if we know how long time it takes for the signal to travel between the devices, then the traveled distance can be calculated. Another way is by measuring the absolute decrease in signal strength. This requires information of the signal strength emitted by the transmitter in combination with RSSI. If we know the exact drop in signal strength from the source transmitter, then we can calculate the distance which increases exponentially similar to our own RSSI calculations in Figure 4.8.

Lastly, the GPS was not as accurate as we hoped for. When we look at the results in Chapter 5, we see that $RSSI_m$ does not have the most accurate estimated areas and we ascribe this to the Bluetooth measurements (Figure 5.12 and Table 5.2). Part of the problem could come from the GPS accuracy which we did not examine. One way of improving the GPS accuracy for our Crowdfinding system could be by analyzing to which degree it needs to be corrected. This could increase the accuracy of the $RSSI_m$ method.

The current prototype focuses on implementing an algorithm for Crowdfinding. Due to the limited scope of the thesis, it remains unknown how long time it will take to find a bicycle in a city. Many factors affect how fast Crowdfinding can locate a bicycle, among which are the crowd size, activity level of the crowd, and the size of the city. Future prototypes can focus on how long time it takes to locate bicycles using Crowdfinding. The time can be the measure for performance and the amount of participating people and the area to cover could be independent variables.

The future of our Crowdfinding system relies on a crowd of users participating. One way of reaching a crowd that has a critical mass is by implementing the algorithms on third-party platforms such as Donkey Republic's. However, using a single platform limits the crowd size. If the algorithm can be implemented on multiple platforms it would increase the crowd size and therefore the effectiveness of the system. This would benefit everyone participating. There are several stakeholders, bicycle rental companies, insurance companies, private citizens, municipalities, etc., that can benefit from a shared Crowdfinding platform for finding lost bicycles. This would increase the size of the crowd and increase the chance of finding a bicycle.

References

[1] Gregory D. Abowd and Elizabeth D. Mynatt. „Charting Past, Present, and Future Research in Ubiquitous Computing“. In: *ACM Trans. Comput.-Hum. Interact.* 7.1 (Mar. 2000), pp. 29–58 (cit. on pp. 5, 6).

[2] Jakob Bardram and Adrian Friday. „Ubiquitous Computing Systems“. In: *Ubiquitous Computing Fundamentals*. Ed. by John Krumm. CRC Press, 2009, pp. 37–94 (cit. on p. 6).

[3] *Bikeshare Denmark ask citiizens for help*. May 2018. URL: `https://www.facebook.com/BycykFslen/posts/838102409728497` (cit. on p. 14).

[4] Daren C. Brabham. *Crowdsourcing*. 1st. Cambridge, Massachusetts: MIT Press., 2013 (cit. on pp. 7, 8).

[5] 2017.04.28 By The Danish Cycling Federation. *NEW NETWORK OF CYCLE SUPER HIGHWAYS IN DENMARK*. 2017. URL: `https://stateofgreen.com/en/profiles/state-of-green/news/new-cycling-super-highways-will-open-in-may` (visited on May 24, 2018) (cit. on p. 13).

[6] *Bycyklen has been hacked*. May 2018. URL: `www.redstargroup.dk` (cit. on p. 14).

[7] E. Cabrera-Goyes and D. Ordóñez-Camacho. „Towards a Bluetooth Indoor Positioning System with Android Consumer Devices“. In: *2017 International Conference on Information Systems and Computer Science (INCISCOS)*. Nov. 2017, pp. 56–59 (cit. on p. 2).

[8] Aaron Carroll and Gernot Heiser. „An Analysis of Power Consumption in a Smartphone“. In: *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*. USENIXATC'10. Boston, MA: USENIX Association, 2010, pp. 21–21 (cit. on p. 6).

[9] *Diims*. May 2018. URL: `http://www.diims.dk/` (cit. on p. 12).

[10] Paul Dourish. *Where the Action Is: The Foundations of Embodied Interaction (MIT Press)*. The MIT Press, 2004 (cit. on p. 5).

[11] Chris Downey. *Understanding Wireless Range Calculations*. May 2018. URL: http://www.electronicdesign.com/communications/understanding-wireless-range-calculations (cit. on p. 11).

[12] Ericsson.com. *BLUETOOTH INVENTOR NOMINATED FOR TOP EUROPEAN HONOR*. May 2018. URL: https://www.ericsson.com/en/news/2012/6/bluetooth-inventor-nominated-for-top-european-honor (cit. on p. 10).

[13] Jonathan Fürst, Kaifei Chen, Hyung-sin Kim, and Philippe Bonnet. „Evaluating Bluetooth Low Energy for IoT“. In: *Proceedings of the 1st Workshop on Benchmarking Cyber-Physical Networks and Systems (CPSBench'18)*. Apr. 2018 (cit. on p. 11).

[14] John Paulin Hansen, Alexandre Alapetite, Henning Boje Andersen, Lone Malmborg, and Jacob Thommesen. „Location-Based Services and Privacy in Airports“. In: *Human-Computer Interaction – INTERACT 2009*. Ed. by Tom Gross, Jan Gulliksen, Paula Kotzé, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 168–181 (cit. on p. 9).

[15] Johnni Hested, Søren Andreasen, and Thomas Rohleder. „Indoor Navigation in Grocery Stores using Bluetooth Low Energy Beacons“. In: *Unpublished manuscript, IT University of Copenhagen* (2017) (cit. on p. 3).

[16] Stephanie Houde and Charles Hill. „What do prototypes prototype?“ In: *Handbook of Human-Computer Interaction (Second Edition)*. Elsevier, 1997, pp. 367–381 (cit. on p. 19).

[17] Mark R. Lepper and Jennifer Henderlong. „Chapter 10 - Turning “play” into “work” and “work” into “play”: 25 Years of research on intrinsic versus extrinsic motivation“. In: *Intrinsic and Extrinsic Motivation*. Ed. by Carol Sansone and Judith M. Harackiewicz. Educational Psychology. San Diego: Academic Press, 2000, pp. 257–307 (cit. on p. 7).

[18] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. „Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median“. In: *Journal of Experimental Social Psychology* 49.4 (2013), pp. 764–766 (cit. on p. 18).

[19] *LOCATION | ANDROID DEVELOPERS*. May 2018. URL: https://developer.android.com/reference/android/location/Location.html#getAccuracy() (cit. on p. 10).

[20] A. E. Middleditch, T. W. Stacey, and S. B. Tor. „Intersection Algorithms for Lines and Circles“. In: *ACM Trans. Graph.* 8.1 (Nov. 1988), pp. 25–40 (cit. on p. 42).

[21] Y. Miyagawa and N. Segawa. „Construction of Indoor Location Search System Using Bluetooth Low Energy“. In: *2017 Nicograph International (NicoInt)*. June 2017, pp. 33–36 (cit. on pp. 2, 9).

[22] F. Naya, H. Noma, R. Ohmura, and K. Kogure. „Bluetooth-based indoor proximity sensing for nursing context awareness“. In: *Ninth IEEE International Symposium on Wearable Computers (ISWC'05)*. Oct. 2005, pp. 212–213 (cit. on p. 9).

[23] Antti Oulasvirta. „Field Experiments in HCI: Promises and Challenges“. In: *Future Interaction Design II*. Ed. by Hannakaisa Isomäki and Pertti Saariluoma. London: Springer London, 2009, pp. 87–116 (cit. on p. 19).

[24] *PRINCIPLES FOR A SUSTAINABLE BIKE SHARE - DONKEY REPUBLIC BIKE-SHARE FOR CITIES*. May 2018. URL: https://cities.donkey.bike/4-principles-sustainable-bike-share/ (cit. on p. 14).

[25] M. E. Rida, F. Liu, Y. Jadi, A. A. A. Algawhari, and A. Askourih. „Indoor Location Position Based on Bluetooth Signal Strength“. In: *2015 2nd International Conference on Information Science and Control Engineering*. Apr. 2015, pp. 769–773 (cit. on p. 2).

[26] Richard M Ryan and Edward L Deci. „Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being.“ In: *American psychologist* 55.1 (2000), p. 68 (cit. on p. 7).

[27] *Sherlock.bike lock*. May 2018. URL: https://www.sherlock.bike/en/ (cit. on p. 14).

[28] Gary Sims. *THE TRUTH ABOUT BLUETOOTH 5 - GARY EXPLAINS*. May 2018. URL: https://www.androidauthority.com/bluetooth-5-speed-range-762369/ (cit. on p. 10).

[29] F. Subhan, H. Hasbullah, A. Rozyyev, and S. T. Bakhsh. „Indoor positioning in Bluetooth networks using fingerprinting and lateration approach“. In: *2011 International Conference on Information Science and Applications*. Apr. 2011, pp. 1–9 (cit. on p. 2).

[30] *Tile*. May 2018. URL: https://www.thetileapp.com/en-us/ (cit. on p. 13).

[31] *TrackR - find more, search less*. May 2018. URL: https://secure.thetrackr.com/ (cit. on p. 13).

[32] *Ushahidi*. May 2018. URL: https://www.ushahidi.com/ (cit. on p. 8).

[33] Alexander Varhavsky and Shwetak Patel. „Location in Ubiquitous Computing“. In: *Ubiquitous Computing Fundamentals*. Ed. by John Krumm. CRC Press, 2009, pp. 285–320 (cit. on pp. 8, 9).

[34] John W. Tukey. *Exploratory Data Analysis*. Pearson, 1977 (cit. on p. 20).

[35] Kamin Whitehouse, Chris Karlof, and David Culler. „A Practical Evaluation of Radio Signal Strength for Ranging-based Localization“. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 11.1 (Jan. 2007), pp. 41–52 (cit. on p. 11).

[36] NJ 08405 William J. Hughes Technical Center WAAS TE Team Atlantic City International Airport. *Global Positioning System (GPS) Standard Positioning Service (SPS) Performance Analysis Report*. 2017. URL: http://www.nstb.tc.faa.gov/reports/PAN96_0117.pdf#page=22 (visited on May 24, 2018) (cit. on p. 9).

[37] Jiuqiang Xu, Wei Liu, Fenggao Lang, Yuanyuan Zhang, and Chenglong Wang. „Distance measurement model based on RSSI in WSN". In: *Wireless Sensor Network* 2.08 (2010), p. 606 (cit. on p. 10).

[38] Jerrold H Zar. *Biostatistical Analysis*. eng. 5th ed. Harlow, United Kingdom: Pearson Education Limited, 2013 (cit. on p. 21).

[39] Lihong Zhang, Jun Zhang, Zheng-yu Duan, and David Bryde. „Sustainable bike-sharing systems: characteristics and commonalities across cases in urban China". In: *Journal of Cleaner Production* 97 (2015). Special Volume: Why have 'Sustainable Product-Service Systems' not been widely implemented?, pp. 124–133 (cit. on p. 13).

[40] Yuan Zhuang, Jun Yang, You Li, Longning Qi, and Naser El-Sheimy. „Smartphone-Based Indoor Localization with Bluetooth Low Energy Beacons". In: *Sensors* 16.5 (Apr. 2016), p. 596 (cit. on p. 2).

List of Figures

List of Listings

List of Tables

List of Equations

Glossary

API	Application Programming Interface.
CI	Confidence interval.
Collector	A smartphone with the prototype Collector application installed and used for collecting data in experiments.
Controller	A smartphone with the prototype Controller application installed used for controlling experiments.
Crowd	A online community which participates in a Crowdsourcing project.
Crowdfinding	When an organization utilizes a crowd in order to locate items. This can be achieved in many different ways.
Crowdsourcing	This is a sourcing technique which allows organizations to source tasks to a crowd of individuals for mutual benefits.
CS	Crowdsourcing.
CSCW	Computer-Supported Cooperative Work is an area of HCI where cooperation in the office is enabled by computation.
CSV	Comma-separated values.
Dead reckoning	A method for calculating position by using a previously determined position combined with velocity and direction.
Donkey Republic	Donkey Republic is a global bicycle rental service.
Estimated area	The area which the algorithm estimates the bicycle to be within.
Firebase Database	NoSQL object database provided by Google.

$FIXED_m$	A method used to create location data points. It uses a predetermined Bluetooth distance of 20 meter.
FusedLocationProvider	This is a location API provided by Google Play services.
Google Play Services	Proprietary services and APIs bundled for Android devices from Google.
HCI	Human–Computer Interaction examines the design and implementation of technology focused on the interfaces between humans and computers.
JSON	JavaScript Object Notation.
Location data points	A data point comprised of GPS coordinates, GPS accuracy, and BLE distance which is used by the algorithm.
NoSQL	Nonrelational database that use an alternative data model such as object, document, or graph.
PARC	Palo Alto Research Center.
Pervasive computing	This is a vision within computer science based on ubicomp. Pervasive computing originated from IBM and focuses on mobility. The goal is to solve task with an multitude of ubiquitous sensors and computers.
Proximity	A localization method which utilizes proximity zones.
RSSI	Received Signal Strength Indication.
RSSI-to-distance function	This function calculates a distance based on RSSI.
$RSSI_m$	A method used to create location data points. It calculates distance based on a RSSI-to-distance function.
SD	Standard deviation.
The algorithm	This takes a list of location data points created by a crowd and returns an estimated area for a bicycle.

The prototype	Our first iteration of Crowdfinding which include Collectors, a Controller, database, the algorithm, and the two distance methods.
The system	See *prototype*.
ubicomp	Ubiquitous Computing is a term referring to the research field started by Mark Weiser at Xerox PARC.