NTNU

Norwegian University of Science and Technology

Navigation using Bluetooth Low Energy Beacons

Håkon Espeland

Master of Science in Cybernetics and Robotics

Submission date:	June 2018
Supervisor:	Geir Mathisen, ITK
Co-supervisor:	Bjørn Magne Elnes, Aventi

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

MASTER THESIS DESCRIPTION SHEET

Name:	Håkon Espeland
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Navigasjon ved bruk av Bluetooth Low Energy Beacons
Thesis title (English):	Navigation using Bluetooth Low Energy Beacons

Thesis Description: All around the world companies, local authorities and street performers, install and use Bluetooth Beacons. These Beacons transmits a hexadecimal string or readable URL, based on how they are configured. Frameworks from Apple (iBeacon) and Google (Eddystone) are designed to interact with mobile applications on smartphones or tablets, having a Bluetooth antenna.

In this thesis, we want to explore the possibility to use this technology for positioning, applied in a mobile application for Car Parking, providing indoor navigation inside a parking garage.

The following subtasks should be considered:

1. Performing a litterature-study regarding Bluetooth Beacon Protocols and relatable algorithms for positioning.

2. Design a navigation system that uses positioning principals like Cell Identification and Trilateration through Received Signal Strength Indicator (RSSI) from BLE Beacons.

3. Implement the navigation system as an extension to the mobile application, ePark, developed during the related specialization project. Perform extensive testing to evaluate navigation using signals from BLE Beacons.

The subtasks will result in answering the following questions:

Q1: What level of accuracy could a Bluetooth based Navigation System expect to achieve?

Q2: Which positioning concept provide the best results?

Start date:	08. January, 2018
Due date:	04. June, 2018

Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Professor Geir Mathisen, Dept. Of Eng. Cybernetics
Co-supervisor:	Bjørn Magne Elnes, Aventi Technology

Abstract

The wireless techonology Bluetooth has over the years become more commonly supported by personal electronic devices, especially in smart-devices such as smartphones, tablets and smart-watches. Earlier, Bluetooth was mostly used as wireless communication between personal devices, and not between personal and public Bluetooth peripherals. When Bluetooth version 4.0, also called Bluetooth Low Energy (BLE) was introduced in 2010, the development of light-weight, easy-deployable Bluetooth Beacons emerged. This development opened up for using public Bluetooth peripherals to interact with personal devices, broadcasting information to compatible units within close proximity. This report is a result of researching the functionality found within the Bluetooth Low Energy protocol, and by using available frameworks, design and implement a navigation system into an Android mobile application that uses interaction with Bluetooth Beacons for determining its position.

Integrating Android, BLE Beacon libraries, Cloud Services and relevant positioning principals, a navigation system is developed and implemented in the Android application, ePark, as a part of a parking service, providing navigation inside a parking garage. Using the Eddystone UID framework developed by Google, a smartdevice running ePark continously interacts with nearby BLE Beacons. The calculated Received Signal Strength Indication (RSSI) of each beacon are passed into a position algorithm along with the known coordinates of each beacon, having the algorithm calculate the smartdevice's current position.

Positioning principals like Cell Identification and Trilateration are implemented into the position algorithm to evaluate the performance of a bluetooth navigation system which is based on RSSI-measurements. Imprecision in RSSI measurements are seen to directly affect the positioning algorithm, and the achieved accuracy is approximately 5 meters when using trilateration.

Sammendrag

(Norwegian translation of the abstract)

Den trådløse teknologien, Bluetooth, har de siste årene i større grad blitt tilgjengeliggjort i det meste av personlig elektronikk, spesielt i smarte enheter som smart-telefon, nettbrett og smart-klokker. Tidligere har Bluetooth for det meste blitt brukt innenfor enhet-til-enhet kommunikasjon mellom private enheter, og ikke mellom private og offentlige enheter. Når Bluetooth versjon 4.0 ble lansert i 2010 under navnet Bluetooth Low Energy (BLE), iverksatte det utviklingen av en ny måte å bruke teknologien på, ved bruk av BLE Beacons. Denne utviklingen åpnet opp og skapte et behov for kommunikasjon mellom private og offentlige enheter ved å bruke BLE Beacons til å kringkaste informasjon til kompatible enheter innenfor rekkevidde. Denne rapporten er et resultat av undersøkelser rundt funksjonaliteten til Bluetooth Low Energy protokollen, og hvordan man kan bruke tilgjengelige rammeverk for å utvikle et navigasjonssystem og implementere det i en mobilapplikasjon som tar i bruk Bluetooth teknologi til å fastsette posisjonen sin.

Ved å integrere Android, tilgjengelige software biblioteker for BLE Beacons, sky-tjenester og relevante posisjonerings prinsipper, er ePark utviklet som en parkerings app for Android hvor navigasjon inne i et parkeringshus er noe av det appen tilbyr. Ved å bruke Eddystone UID rammeverket som tidligere har blitt utviklet av Google, kan en smart enhet som kjører ePark applikasjonen måle signalstyrken til BLE Beacons innenfor rekkevidde og bruke disse målingene sammen med koordinatene for hver beacon til å kalkulere posisjonen til enheten gjennom en posisjonerings algoritme.

Posisjonerings prinsipper som celle-identifisering og trilaterering er implementert i posisjonerings algoritmen for å kunne vurdere ytelsen av et navigasjonssystem som baserer seg på signalstyrke fra bluetooth sendere. Unøyaktighet i RSSI målinger fra en enkel BLE Beacon viser seg å påvirke posisjoneringsalgoritmen direkte, og oppnåd nøyaktighet er rundt 5 meter når algoritmen baserer seg på trilaterering.

Preface

This report is the result of the Master of Science in Engineering Cybernetics, TTK4900. This thesis is part of a two-year masters program in Cybernetics and Robotics with specialization in Embedded Systems at the Norwegian University of Science and Technology.

The master's thesis was carried out during the spring semester of 2018 in cooperation with Aventi Technology, as further work related to the specialization project which was carried out during the fall semester of 2017. The work focuses Bluetooth Low Energy (BLE) and how signals from BLE Beacons can be used in a navigation system. The practical tasks and development are performed as further work on the ePark application developed during the related specialization project, meaning that all of the developed functionality has been implemented as extensions to the former work.

There has not been performed a lot of serious research regarding this subject. The basis of the master thesis takes stand in knowledge acquired during the specialization project together with applied mathematics and discovered theory of relevance for making this project feasible. A full review of the background material and what distinguishes the specialization project from the master thesis is given in Section 1.1.

I would like to thank my supervisor, Professor Geir Mathisen for valuable feedback and guidance. A special thanks to my co-supervisor, Bjørn Magne Elnes and Aventi Technology for the endless support and guidance throughout the project.

Trondheim, 2018-06-03

Håkon Espeland

Contents

Master Thesis Description Sheet i
Abstract ii
Sammendrag iii
Preface iv
List of Tables ix
List of Figures 1

1 Introduction **1**
1.1 Background and Motivation 1
1.2 Objective and scope 2
1.3 Outline 3

2 Introduction to ePark **5**
2.1 System description 5
2.2 BLE Beacon Technology 7
2.2.1 Bluetooth Low Energy - Bluetooth v4.0 7
2.2.2 Bluetooth Beacons 8
2.2.3 iBeacon 9
2.2.4 Eddystone 10
2.3 Android 12
2.3.1 Architechture 12
2.3.2 Activities 14
2.3.3 Services 16
2.3.4 Manifest and permissions 17
2.3.5 Android Beacon Library 17
2.4 Cloud Storage 19
2.4.1 Server connection 19
2.4.2 Server requests 20

3 Positioning System **21**
3.1 Introduction to positioning systems 21
3.2 Positioning Concepts 23
3.2.1 Cell Identity 23
3.2.2 Triangulation 24
3.2.3 Trilateration 25
3.2.4 Angle of Arrival (AoA) 28
3.2.5 Time of Arrival/Time Difference of Arrival (ToA/TDoA) 29
3.3 Positioning using BLE Beacons 31
3.3.1 Related Work 32
3.3.2 Trilateration with BLE Beacons 33

4 Design **37**
4.1 Introduction 38
4.1.1 Functionality 38
4.1.2 Preliminary decisions on design 38
4.2 System Design 39
4.3 Hardware Design 41
4.3.1 Hardware Requirements 41
4.3.2 Hardware Choices based on design 42
4.4 Software Design 43
4.4.1 Application design 44
4.4.2 Services 47
4.4.3 Database Design 48

5 Implementation **51**
5.1 Hardware Implementation 51
5.1.1 RadBeacon Bluetooth Beacons 52
5.1.2 Smartdevices 53
5.1.3 Hardware Deployment 53
5.2 Software Implementation 54
5.2.1 ePark GUI 54
5.2.2 Activities 55
5.2.3 Bluetooth Service 61
5.2.4 Network Service 65
5.2.5 Positioning Algorithm 68

6 Results **71**
6.1 System Testing . . . 71
6.1.1 Single Beacon Range Test . . . 72
6.1.2 Trilateration Test . . . 73
6.2 Subtask Results . . . 75
6.2.1 ST1: Theory . . . 75
6.2.2 ST2: Navigation System . . . 76
6.2.3 ST3: Android Development and Integration . . . 76

7 Discussion **79**
7.1 Accuracy of distance measurements . . . 79
7.2 Navigation System Accuracy . . . 80
7.3 Research Questions . . . 81
7.4 Integrations . . . 83

8 Conclusion **85**

9 Future Work **87**

References **89**

List of Tables

2.1 Transmission power values 8

4.1 ePark' layered architecture 43

5.1 Comparison of different BLE Beacons 52
5.2 Bluetooth Service Methods 62
5.3 Network Service Requests 67

6.1 Single Beacon Range Test 72
6.2 Trilateration Test 73

List of Figures

2.1 Information flow in the system 6
2.2 iBeacon Data Overview 9
2.3 Eddystone Data Overview 11
2.4 The Android Software Stack from [1] 13
2.5 Activity Lifecycle from [2] 15

3.1 Cell Identification with BLE Beacons 23
3.2 Triangulation 24
3.3 Trilateration 25
3.4 Illustration of trilateration symbols 26
3.5 Angle of Arrival 28
3.6 Time Difference of Arrival (TDOA) 30
3.7 Trilateration using BLE Beacons 31

4.1 Functionality Design 40
4.2 Application flowchart 44
4.3 Database design 48

5.1 Android Studio Graphics Development with XML 54
5.2 Java development in Android Studio 55
5.3 Main Activity in ePark 57
5.4 Map Activity of testarea 58
5.5 Park Activity in ePark 59
5.6 History Activity in ePark 60
5.7 Scheduling the Bluetooth Service 64
5.8 Network Service Communication Flow 65
5.9 Cell Identification 3D illustration 68
5.10 Trilateration 3D illustration 69
5.11 Map Activity Graphical User Interface 70

6.1 Single Beacon Range Test with error 72
6.2 Trilateration Test showing real position versus calculated position 74

Chapter 1

Introduction

1.1 Background and Motivation

Bluetooth Low Energy (BLE) hit the market in 2011 as Bluetooth 4.0. To establish a Machine-to-Machine (M2M) connection using Standard Bluetooth[1], pairing is required before use. This was one of the most comprehensive changes made in v4.0 [3]; using BLE, it is not required to pair devices to establish a M2M connection. Until this release Bluetooth was mostly used as a private network between personal devices, but along with BLE, a market for digital public advertising evolved.

A Beacon is a BLE-based radio transmitter, repeatedly transmitting a single signal that other devices can read. It consists of a CPU, radio and a power source[2]. Beacons are transmitting hex-strings which is basically unique identification of each beacon along with sensor-data dependant on the protocol used. Through different applications, the ID of a beacon can be used to prompt the user with notifications, provide an advertisement or determine their proximity.

The proximity to a BLE Beacon can be measured by a smartdevice, i.e a smartphone or a tablet. Measuring the RSSI (Received Signal Strength Indication), distance between a smartdevice and a beacon can be converted into meters using Eq. 2.2. Aventi Technology desired to explore the possibilities of using the RSSI measurements for navigation, applied

[1]Bluetooth Core Specification 3.0 and lower.

[2]Beacons often use small lithium chip batteries, or a plugged power source.

through conventional positioning principals. Being a company developing systems within ITS (Intelligent Transportation Systems), Aventi Technology came up with the idea of using RSSI measurements from BLE Beacons to provide indoor navigation inside a parking garage through a mobile application.

Conventional GNSS technology (Global Navigation Satellite System), including GPS, relies on the line of sight of satellites. These kind of technologies are not suited nor intended for indoor applications. A parking garage typically has poor GPS coverage, which motivates for the use of a local positioning system, for the use RSSI measurements from locally deployed BLE Beacons. Such a positioning system allows the use of many of the same positioning principals as conventional GNSS technology, which of will be covered through this report.

During the fall of 2017, a specialization project was carried out for Aventi Technology. The work resultet in the development of a parking application named **ePark**, which created the basis for this master thesis. ePark was developed as an Android mobile application, interacting with BLE Beacons. Conceptually, running ePark in a parking garage with deployed BLE Beacons, the user could request and reserve parking spots after parking by having ePark determine the closest beacon (in meters) and reserve the ID of the related parking spot.

1.2 Objective and scope

The main objective of this master thesis is to develop a local navigation system using BLE Beacons. Based on the measured RSSI (Received Signal Strength Indication), positioning algorithms based on conventional positioning principals are to be developed and implemented as extensions into the former developed Android application, ePark. The navigation system is going to be a part of a parking service, featuring a map-interface, reservation possibilities and parking log.

The following subtasks are proposed for the project:

1. Performing a litterature-study regarding Bluetooth Beacon Protocols and relatable algorithms for positioning.
2. Design a navigation system that uses positioning principals like Cell Identification and Trilateration through Received Signal Strength Indicator (RSSI) from BLE Beacons.
3. Implement the navigation system as an extension to the mobile application, ePark, developed during the related specialization project. Perform extensive testing to evaluate navigation using signals from BLE Beacons.

Through these subtasks it is desirable to evaluate the accuracy of a bluetooth-based navigation system, the system performance using different concepts and algorithms, and which use-cases that the system is applicable for.

1.3 Outline

The outline of this report is meant to reflect the subtasks of the thesis along with relevant previous work that were performed during the fall of 2017.

Chapter 2 summarizes the project thesis that was finished in December 2017, bringing forward preliminary theory and results, making the foundation for the subtasks of the master's thesis. Chapter 3 introduces relevant theory regarding positioning technologies and principals. Chapter 4 presents the design of both software and hardware, reasoning the design choices, followed by Chapter 5 presenting the system implementation, with focus on software- and hardware-implementation, software integration and important system modules. The results are presented in Chapter 6. Chapter 7 discusses some important measures in the project, followed by some concluding remarks in Chapter 8. The report finishes off with suggestions for future work in Chapter 9.

Chapter 2

Introduction to ePark

This chapter will introduce the Android mobile application, ePark, developed during the specialization project which was performed as preliminary work for this master thesis. As ePark is described in this chapter, Section 2.2 and Section 2.3 summarizes some of the theory acquired during the specialization project, which is seen as highly relevant for the understanding of this report.

2.1 System description

ePark is an Android Mobile Application developed during the associated specialization project carried out during the fall of 2017. The application is meant to deliever several services when parking in a public garage, such as:

- Parking reservation
- Display real-time elapsed parking time and cost
- Indoor navigation inside the parking garage, with status over available spots
- Parking log of previous parkings

Being further work on an already existing application, almost all of the listed services are supposed to be developed during this project. From earlier on, ePark has the ability to determine the closest BLE Beacon, and reserve the related parking ID. Parking timer, indoor navigation services, and parking log are all features that needs to be developed in conjunction with the proposed work. One comprehensive change in the software structure of ePark, is moving the cloud storage and its related modules from the Microsoft Azure Cloud platform to Cloud Services provided by Digital Ocean. The new structure is presented in Section 2.4.

Installing and running ePark on an Android device (smartphone, tablet), fascilitates interaction between the smart-device, bluetooth beacons deployed in a public parking garage and cloud storage. Requirements for the Android device are:

- Minimum API level 21 (Android Lollipop)
- Supporting Bluetooth v4.0 or higher
- Wi-Fi or 4G LTE enabled
- Granted permissions for Internet, Bluetooth and Location Services in Android

Running ePark as a registered user, the Android device functions as a scanner, scanning for Bluetooth devices within range, based on the Eddystone Protocol. The Android device performs duplex communication with a server (Ubuntu 16.04 MySQL-server), polling beacon and parking information from the cloud-plattform, as shown in Figure 2.1.

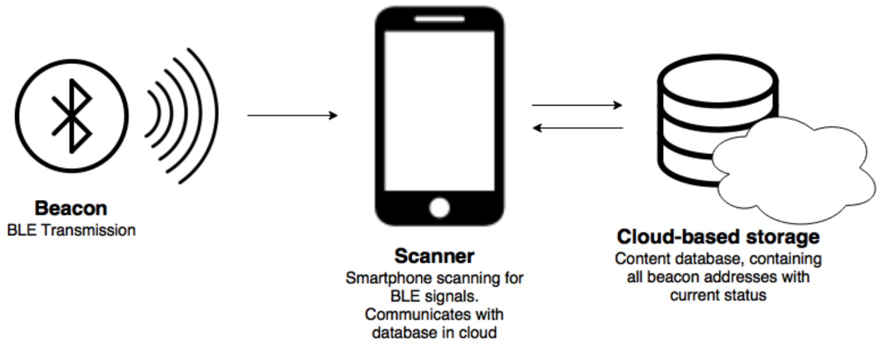

Beacon
BLE Transmission
Scanner
Smartphone scanning for BLE signals. Communicates with database in cloud
Cloud-based storage
Content database, containing all beacon addresses with current status

Figure 2.1: Information flow in the system

The following sections will enlighten the basic theory and concepts which make up the system of ePark.

2.2 BLE Beacon Technology

The beacons used in this project are based on the Bluetooth v4.0 [3], also referred to as BLE (Bluetooth Low Energy) or Bluetooth Smart™. The standard itself will be presented in this section, along with Beacon Standards available today.

2.2.1 Bluetooth Low Energy - Bluetooth v4.0

Bluetooth is a short range Radio Frequency (RF) communication technology that operates in the 2.4 GHz ISM band, and is a global wireless standard for simple connectivity. The technology can be divided into two main categories; Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) and Bluetooth Low Energy (BLE).

Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR):
Used for continous connection in a point-to-point (1:1) connection. Typically used for audio streaming e.g wireless headsets, wireless speakers and in-car audio.

Bluetooth Low Energy (BLE):
Used for various short burst connections such as point-to-point (1:1), broadcast (1:m) and mesh (m:m). Point-to-Point is used in data transfer between a central unit (e.g smartphone) and peripherals like fitness devices and welness monitors. Broadcast focuses on localized information, using beacons to transmit information within a configured range. The last type of connection, Mesh, is used in conjuction with larger device networks such as building automation, wireless sensor networks and asset tracking.

Bluetooth Low Energy (BLE) was incorporated into the Bluetooth 4.0 Core Specification in 2010 and experienced rapid market growth, including all the major operating systems, most of the smartphones and tablets, along with a new breed of devices like fitness bands and simple RF tags. Support for Bluetooth 4.0 on some platforms are listed below:

- iOS 5+ (iOS 7+ preferred)
- Android 4.3+ (numerous bug fixes in 4.4+)
- Apple OS X 10.6+ (named macOS today)
- Windows 8 (XP, Vista and Windows 7 only support Bluetooth 2.1)

2.2.2 Bluetooth Beacons

Bluetooth Beacons are BLE-based devices that broadcasts small packets of data within a radius based on their configuration. The configuration comprises a Beacon Standard, transmit-power determining the range, advertising-rate and configuration of the packets of data broadcasted.

Transmit Power
The Transmit Power setting determines the strength of the transmitted beacon signal, measured in dBm (decibel-milliwatts). A beacon can normally be configured within a range of -30 dBm to +4 dBm, and by measuring the RSSI (Received-Signal-Strength-Indication) from a central device (e.g smartphone), it is possible to calculate the distance in meters.

The RSSI is calculated by the following equation:

$$RSSI_{@1m} = -20 \cdot \log d + MeasuredPower \tag{2.1}$$

which provides:

$$d = 10^{\frac{MeasuredPower - RSSI_{@1m}}{20}} \tag{2.2}$$

where d is the distance between the central device and the beacon.

Table 2.1: Transmission power values

TX Power level	RSSI @ 1m	Range (meters)
-30 dBm	-115 dBm	2
-20 dBm	-84 dBm	4
-16 dBm	-81 dBm	10
-12 dBm	-77 dBm	20
-8 dBm	-72 dBm	30
-4 dBm	-69 dBm	40
0 dBm	-65 dBm	60
4 dBm	-59 dBm	70

Each Beacon is calibrated by measuring the RSSI at a distance of 1 meter. The calibrated $RSSI_{@1m}$ is sent along with the Beacon data, making it possible to perform calculations based on equation 2.2 on the central device.

Advertising Rate

The advertising rate setting determines how often the beacon transmits the signal [1Hz - 10Hz]. A higher advertising rate will consume more power, which often is the most decisive setting configuring a beacon (desirable advertising rate vs. battery life).

2.2.3 iBeacon

Apple announced the iBeacon protocol in December 2013, a Bluetooth Low Energy Beacon profile which officially only works with iOS devices, although some Android devices are capable of receiving the iBeacon signals.

The iBeacon protocol requires an app to receive, process and/or track the beacon, meaning they only work with apps installed on a device. iBeacons interact with apps by broadcasting a single **unique ID**. The broadcasted signal is used by the app to prompt an action, such as providing indoor navigation, keyless access, notifying the user of a nearby deal.

The broadcasted **unique ID** consists of:

- UUID (Universally Unique Identifier) - 16 bytes/128bits BLE UUID
- Major - unsigned integer value between 0 and 65535
- Minor - unsigned integer value between 0 and 65535

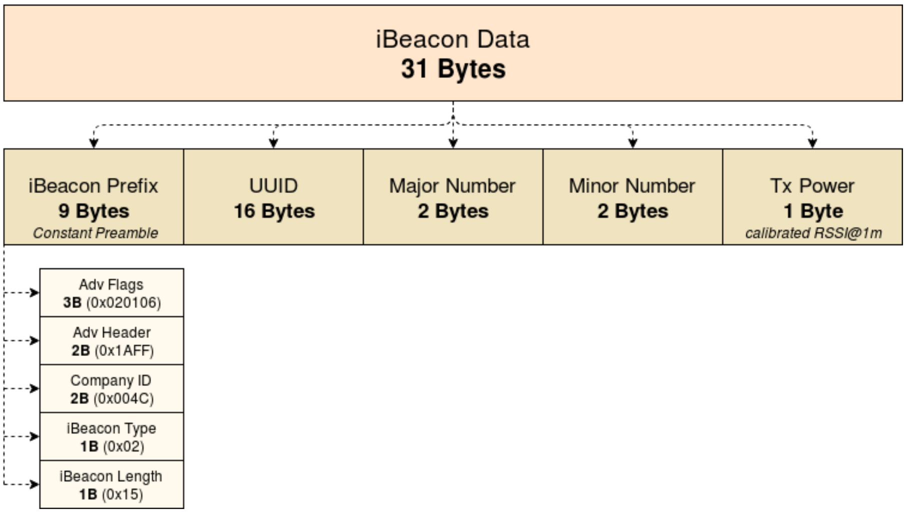


Figure 2.2: iBeacon Data Overview

To put the different values into context, a museum is used as an example scenario. The museum has invested in 1000 iBeacons. All 1000 beacons are configured with the same **UUID**, telling the end-user that the received signal comes from a beacon owned by the museum. Inside the museum, there are several paintings and sculptures spread over 5 floors. It would then be natural to assign the floor number as **Major** (1-5), and assigning the exhibit number as **Minor**. If each floor has 200 exhibits, then each beacon would be configured with the same **UUID**, a **Major** representing the floor (1-5), and a **Minor** representing the exhibit number (1-200).

The iBeacon API makes it possible to develop applications for both monitoring and ranging. By monitoring, functions from the library returns the **UUID**, **Major** and **Minor** values from all beacons that the end-user receives signals from. Ranging uses the **Tx Power** sent along with the rest of the iBeacon data to compute the distance to each beacon based on actual RSSI and Equation 2.2. The accuracy values may fluctuate due to RF interference.

2.2.4 Eddystone

Eddystone is a Bluetooth Low Energy Beacon profile released by Google in July 2015, as a cross-plattform protocol containing multiple frame types. The protocol is already implemented in several native apps, including Google Maps and Nearby.

The Eddystone protocol has 4 frame types:

- **Eddystone-UID** broadcasts an identifying code containing namespace ID, instance ID, making it possible to distinguish between beacons when interacting with apps.
- **Eddystone-EID** broadcasts an encrypted identifier which is continously rotated to increase the security of the protocol, but otherwise functions as the UID frame.
- **Eddystone-TLM** broadcasts information about the beacon such as sensor data, battery level and other relevant information to the beacon admininistrators. TLM must be accompanied by another frame to be used.
- **Eddystone-URL** broadcasts a URL, redirecting the end-user to a website that is secured using SSL.

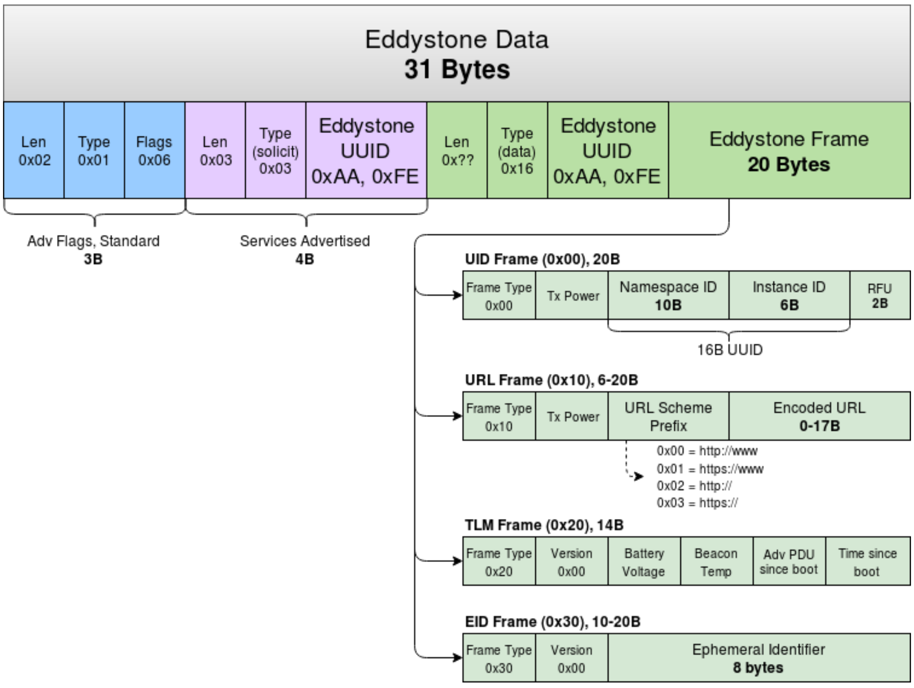


Figure 2.3: Eddystone Data Overview

An Eddystone Beacon can be configured with several frames, swapping between different GAP advertising packets that is being transmitted. The **UUID** consisting of **namespace ID** and **instance ID** transmitted through the Eddystone-UID frame, provides the same 16Bytes of UUID namespace that iBeacon does. The **namespace ID** is unique and assigned to a group of beacons, along with the **namespace ID** assigned to each beacon within the group, making it possible to distinguish between different beacons.

2.3 Android

Android is a mobile operating system developed by Google, primarily for touchscreen devices such as smartphones and tablets. Initially, the system was developed by Android Inc. which was bought by Google in 2005, before the mobile platform was released in 2007. Through Android API's (Application Programming Interface), a set of core libraries and functionalities can be accessed, extended and customized for development of applications for modern mobile devices.

Applications ("apps"), extending the functionality of a device, are written using the Android SDK, often accompanied by the Java programming language. The Android plattform provides full C++ support when using Java, and development is also available with the use of Google's own programming language Go (Golang) and Kotlin. The below subsections describe the basics of the Android Platform and how Bluetooth Beacon libraries can be implemented in an application.

2.3.1 Architechture

The Android Platform is an open source, Linux-based software stack, developed primarily for mobile operating system, but also for televisions (Android TV), for cars (Android Auto) and for wrist watches (Android Wear). Figure 2.4 below shows the major components of the Android Platform Architechture [1].

The well-known **Linux Kernel** lies on the bottom of the Android Architechture, providing drivers for the different parts of an Android-running device, along with the power management. On top of the Linux Kernel, the **Hardware Abstraction Layer (HAL)** provides standard interfaces between the hardware and the rest of the stack. **Android runtime (ART)** is the managed runtime used by applications and system services on Android. ART replaced its predecessor Dalvik in Android version 5.0 (API Level 25), and is developed to run multiple virtual machines on low-memory devices, executing DEX files, a bytecode format designed specially for Android considering minimal memory footprint. ART includes the following features; Ahead-of-time (AOT) nad Just-in-time (JIT) compilation, Optimized garbage collection (GC) and better debugging support with detailed diagnostic exceptions and crash reporting.

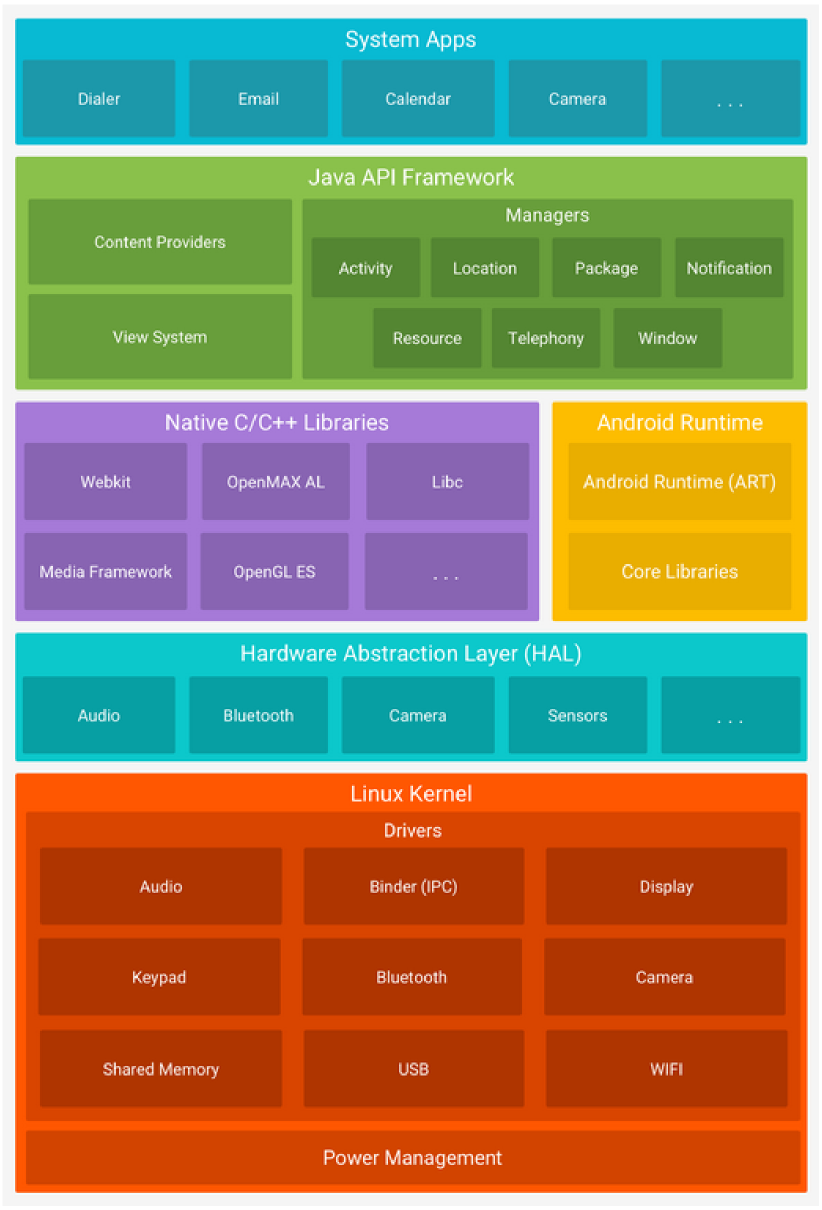


Figure 2.4: The Android Software Stack from [1]

Java API Framework:
All core features in Android OS is available through APIs written in the Java language, simplifying the development by modular system components and services, which include

the following:

- A **View System** used to build the application User Interface (UI), including lists, text boxes, buttons, menus and general graphics and animation.
- A **Resource Manager** providing access to non-code resources such as imported graphics, strings and layout files.
- A **Notification Manager** that enables all applications to show customized notifications in the status bar.
- An **Activity Manager** that manages the lifecycle of applications and provides a common navigation back stack.
- **Content Providers** that enables applications to share data among eachother.

2.3.2 Activities

An activity is a common element in Android Applications. It is basically a Java class which extends the *Activity* class that takes care of creating a window where the User Interface (UI) can be placed by using `setContentView(View)`. Almost all activities interact with the user, presented as full-screen windows, floating windows or embedded inside of another activity.

It is important to note that one activity usually is related to a single view in the application, thus every view is related to its own Java class. Android Applications normally run each activity on the UI thread, but also support multiple threads running in the background.

Android uses an activity stack to manage activities. An activity has to be triggered to start. Starting an activity can be done as default start-up activity, after a given amount of *time*, by user-input (touch, voice-control, keyboard etc.) or by external hardware (signal detection). A new activity is put on top of the stack and comes to the foreground, while the current activity is paused in the background until the new activity exits.

Figure 2.5 displays the lifecycle of an activity with all its methods being called sequentially.

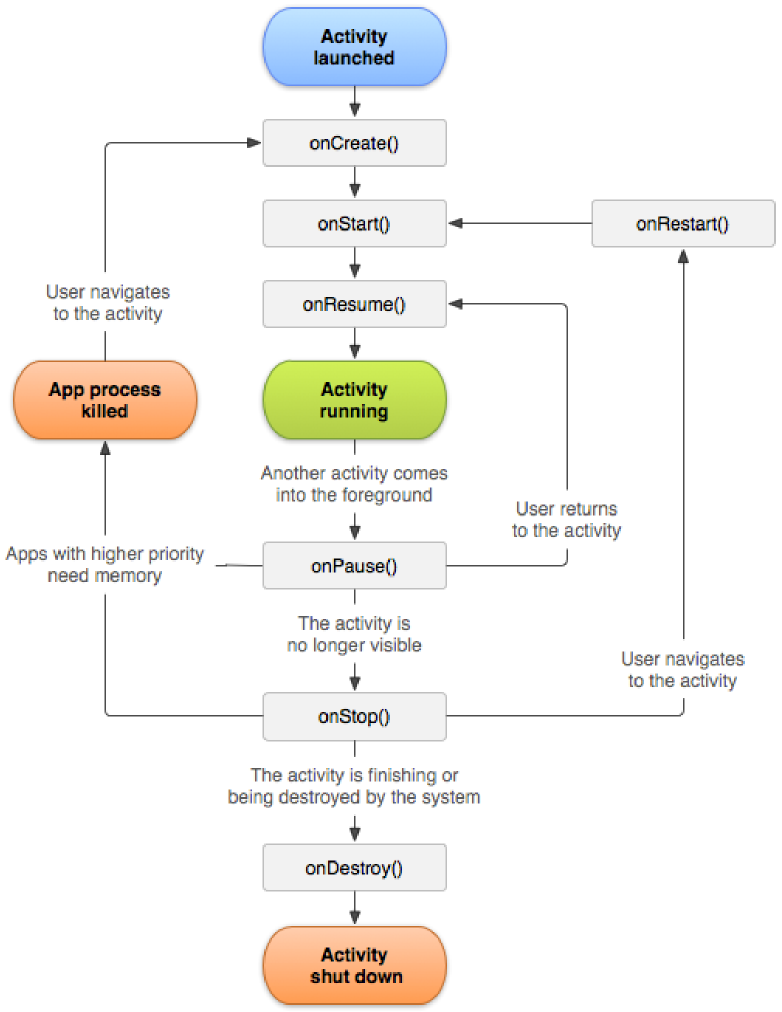

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 2.5: Activity Lifecycle from [2]

There are three loops regarding an activity:

- The **entire lifetime** is from the first call to `onCreate()` through to a single final call to `onDestroy()`. The activity will perform setup of the global state in the `onCreate`-method, and release all remaining resources in the `onDestroy()`-method.
- The **visible lifetime** is everything that happens between a call to `onStart()` until a corresponding call to `onStop()`, and it is during this time that the user can see

the activity on-screen.

- The **foreground lifetime** happens between corresponding calls between `onResume()` and `onPause()`. During this time, the activity is on top of the stack, in front of all other activities. Certain calls are frequent, and are caused by for example when the device goes to sleep or when background tasks deliever results.

The entire lifecycle of an activity is defined by the methods shown in Figure 2.5, and as explained in [2], they can be modified through an `@Override` to do the desirable work when the current activity changes state.

An activity has four states, which are:

- **Running:** The activity is active and runs in the foreground
- **Paused:** The activity has lost focus, but is still alive and displayed
- **Stopped:** The activity saves state information, but is obscured by another activity and is exposed for being killed if the system requires memory
- **Terminated:** The activity is being shut down or killed by the system, freeing system memory

2.3.3 Services

As described in Section 2.3.2, only one activity is allowed to run on the UI-thread at a time. If it is desirable to run tasks in the background without disturbing the user interacting with the app through the UI-thread, *Services*[4] can be used.

A service is Java code that is running in the background, independent of the UI-thread and with no connection to anything directly visible to the user. Typical standard Android Services are services listening for location, incoming phone calls or SMS. These services are part of the Android API, and can not be altered, but it is possible to develop *User Defined Services* to achieve the desirable result from tasks running in the background.

2.3.4 Manifest and permissions

To make an Android Application into an executable, it needs to obtain a detailed description of the application and how it is composed. This is provided through the manifest, `AndroidManifest.xml`. The manifest contains important information regarding:

- Including all app components, and describe the intent-filter of activities set to be `MAIN` and `LAUNCHER` (launcher activity)
- Determining which permissions the application should use. This is done by the `<uses-permission>` command, used to grant permissions for `LOCATION_SERVICES`, `BLUETOOTH`, `INTERNET` and more.
- Setting the app icon, label and theme.

To be able to implement compatibility for Bluetooth Low Energy Beacons in Android, permission for `BLUETOOTH` has to be granted. After the launch of Android version 6.0 (API Level 23), permission for `LOCATION SERVICES` has to be granted in order to request permission for `BLUETOOTH`. As a security measure, all permissions related to location services have to prompt the user with a dialog box, asking for permission. The permission prompt has to be implemented as an abstract Java class, gathering permissions from the `AndroidManifest.xml` that has to be asked permission for.

2.3.5 Android Beacon Library

To detect BLE Beacons in Android, there are several external libraries available. In this project Eddystone UID will be implemented, hence the library **Android Beacon Library** developed by AltBeacon, providing full support for all Eddystone frames [5].

As AltBeacon describes in their documentation, any device with Android 4.3+ and a Bluetooth Low Energy chipset can detect Beacons, using their library.

The package `org.altbeacon.beacon` [6] contains several interfaces and classes, as described below:

`BeaconConsumer`
An interface for an Android `Acticity` or `Service` that wants to interact with beacons. The interface is used in conjunction with `BeaconManager` and provides a callback when the `BeaconService` is ready to detect and range beacons.

`BeaconManager`
A class used to set up the interaction with the beacons. The `BeaconManager` includes and manages all methods for both monitoring and ranging beacons. As mentioned above, the `BeaconManager` requires a callback from the `BeaconConsumer` to use its methods.

`RangeNotifier`
An interface for ranging the distance to beacons. The `RangeNotifier` provides both identification and ranging of visible beacons. It is implemented by the `BeaconManager` class, and calculates the distance through the method:
`void  didRangeBeaconsInRegion(Collection<Beacon> beacons, Region region)`

`MonitorNotifier`
An interface for monitoring a `Region`. Like the `RangeNotifier`, the `MonitorNotifier` provides identification of the beacons in a region, but it does not range them.

`Region`
A class that represents several criterias to match beacons. Through instances of this class, it is possible to filter the region which is desirable to monitor or range, based on Beacon Standard, UUID and other ID parameters (dependable on Beacon Standard).

`Identifier`
A class that encapsulates a beacon identifier of arbitrary byte length, and stores it as a byte array.

2.4 Cloud Storage

Developing an application sometimes requires some form of cloud-based storage. ePark uses an Ubuntu 16.04 server provided by DigitalOcean. The server is configured as a MySQL-server with phpMyAdmin, holding important information such as:

- **User database** - Application Registering and Login
- **Beacon database** - Information and status of all beacons in the system
- **Ticket database** - All ongoing and completed parking requests

2.4.1 Server connection

The connection to the server is done through the language PHP, along with Volley-requests [7] from Android (URL-based requests).

```
<?php
    $host="localhost";
    $port=3306;
    $socket="";
    $user="********"; # Server username
    $password="********"; # Server password
    $dbname="mysql";

    $con = new mysqli($host, $user, $password, $dbname, $port, $socket)
           or die ('Could not connect to the database server'
               . mysqli_connect_error());

    $sql = "SELECT * FROM tickets WHERE user_id = ? AND completed = 1";

    $statement = mysqli_prepare($con, $sql);

/>
```

2.4.2 Server requests

Server requests are requests from Android through the Volley-library [7], to either acquire data, insert data or manipulate existing data. Based on the request, different php-scripts are initiated through their designated URL (https://67.205.190.1/<*php_script_name*>.php).

A php-script could contain an executable MySQL-query. This concept along with the Volley-library [7] in Android allows parameters to be passed to the php-script and used in the MySQL-query. The Volley-library then handles the response from the server, whether it returns a string, a JSON Object or a JSON Array.

JSON (JavaScript Object Notation) [8] is a lightweight data-interchange format. The notation makes it easily readable for humans, in addition to making parsing and generation for computers easy. JSON is mainly built on two structures:

- **JSON Object:** A collection of name/value pairs
- **JSON Array:** A list containing JSON Objects

The notation for an object begins (**{**) and ends (**}**) with curly-brackets. Inside these curly-brackets, name/value pairs are seperated by a colon (**:**) and different pairs can be distinguished by each comma (**,**). This notation is shown through the example JSON Object displayed below:

```
{"ID":5, "name":"George", "age":43}
```

A JSON Array is basically only a list of JSON Objects. The notation for a JSON Array is encapsulated by square-brackets (**[ ]**). Inside the brackets, the JSON Objects are separated by a comma (**,**) as shown below:

```
[{"ID":5, "name":"George", "age":43},
 {"ID":6, "name":"Lisa", "age":19},
 {"ID":7, "name":"Mark", "age":22}]
```

JSON supports data-types such as *string, integer, float, object/array*[1]*, true/false, null.* Missing data-types can be handled through convertion back and forth to the string data-type or other compatable data-types.

[1]Represented as JSON Object/Array as in [9]

Chapter 3

Positioning System

This chapter will introduce some relevant surveying concepts based on signal measurements, applicable for positioning applications. The belonging theory and mathematic expressions will be derived in each section.

3.1 Introduction to positioning systems

Conventional GNSS technology (Global Navigation Satellite System), including GPS, relies on the line of sight of satellites. These kind of technologies are not suited nor intended for indoor applications. This section will introduce several surveying concepts that are applicable for indoor use.

An indoor positioning system requires some form of technology that can determine the relative position of an object. There are several technologies that can be used for this purpose.

- **Signal measurements:** By having signal-anchors of known coordinates, signal measurements between an object and an anchor could determine the position of the object relative to the anchor(s) (distance calculations or time-based concepts). This implies knowing the position of an anchor and the distance to an object, you know the possible position(s) of the object.
- **Magnetic Positioning:** Deploying magnetic sensors obtaining a three-dimensional magnetic field, mapping of this magnetic field can be used for positioning. Through

calculations and filters, changes in the magnetic field can determine the position of an object that interferes with the magnetic field.

- **Dead Reckoning:** By measuring the heading and movement (e.g speed) and time of an object, the position can be estimated. The position then become an estimate based on previous measurements, and will be estimated once a new measurement is available.

This thesis will focus positioning concepts based on signal measurements, having a signal-sources of known coordinates, broadcasting a signal that can be measured by the object (e.g a smartphone), determining the relative position relative to the signal-source. Some of the most common technologies to accomplish this kind of positioning system are:

- **Wi-Fi:** The localization technique is based on measuring the RSSI (Received Signal Strength Indicator) from wireless access points. A WPS (Wi-Fi Positioning System) can be utilized where it is inadequate for conventional GPS, having the position of each wi-fi access point stored in a database, including their SSID and unique MAC-address. WPS is suitable for locations that already has several access points installed, making it a cost-effective solution.

- **Ultra-Wide-Band:** Unlike Wi-Fi, Ultra-Wide-Band (UWB) uses time-synchronization for positioning purposes. UWB operates with pulse-based transmission in the from 3.1 to 10.6 GHz frequency range. By measuring the time it takes for the signal to travel from the transmitter to the receiver, UWB enables higher accuracy than RSSI-based concepts. The concept is based on transmitting a signal from a so-called tag that the user is carrying. The signal from the tag is transmitted to UWB-anchors of known coordinates. Using concepts like Time of Arrival (ToA) or Time Difference of Arrival (TDoA) implemented in a RTLS-server (Real-time location system), the position of the tag can be determined and feeded back to the tag[1] or by Wi-Fi to a smartdevice (e.g smartphone, tablet). Some of the drawbacks by using UWB is that each UWB-anchor requires a synchronization signal, mostly obtained through a ethernet-connection, in addition to deploying a RTLS-server that handles the positioning calculations. This introduces a relative high cost for installing a positioning system based on UWB.

- **Bluetooth:** Like Wi-Fi, a bluetooth-based positioning system uses RSSI (Received Signal Strength Indicator) to determine the position of a bluetooth device. Protocols and frameworks developed by Google (Eddystone) and Apple (iBeacon) enables

[1]Some UWB tags support bluetooth, and can transmit the latest position update over bluetooth to a desired smartdevice for use by installed apps

continous data-transmissions from Bluetooth Beacons of known coordinates, broadcasting their identity along with their calibration parameters. As described in Section 2.2, these measurements can be used in several positioning concepts to determine the position of a bluetooth-device. Due to the Bluetooth Low Energy support, bluetooth beacons results in a positioning system of high mobility due to long battery-life and of relative low cost.

In the next section bluetooth will be focused as the choice of technology, but the concepts are also applicable for both Wi-Fi and UWB.

3.2 Positioning Concepts

3.2.1 Cell Identity

The map is divided into cells, having a beacon deployed in the center of each cell. Measuring the RSSI, the closest/actual cell is determined as the position of the measuring device (e.g smartphone).

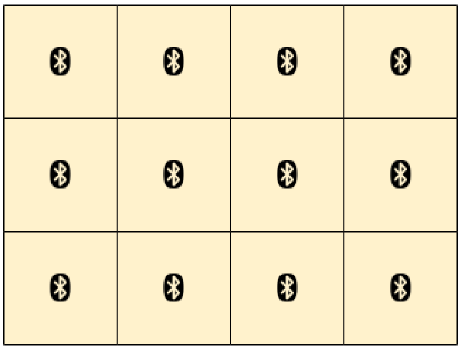


Figure 3.1: Cell Identification with BLE Beacons

Cell Identity achieves a precision based on the size of each cell. Reasonable sizing of the cells are done based on cost-per-cell (1 beacon for each cell) along with expected accuracy of the RSSI. Normally, Cell Identification is used for applications that can tolerate an accuracy exceeding 10 meters. Increasing the accuracy (< 10 meters) can be done by implementing other positioning concepts within the cell, either by installing more equipment or using the signals from the beacons deployed in the neighbour cells.

3.2.2 Triangulation

Triangulation is a surveying technique in which unknown distances between beacons and the smartdevice can be determined by trigonometric applications of one or several triangles [10]. This requires that the antenna's are capable of measuring the angle of the incoming signal, which bluetooth antennas are not capable of doing. With antennas that are capable of measuring the angle of the incoming signal, the object that is desirable to track, sends a signal to surrounding anchors that measure the angle of the incoming signal. As shown in Figure 3.2, this creates 4 triangles.

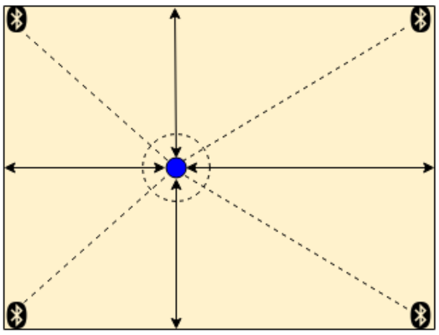
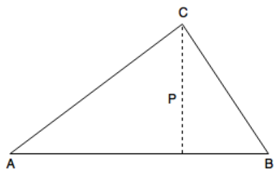


Figure 3.2: Triangulation

Triangulating one triangle provide either a x- og y-coordinate, so at least three anchors are required to perform the following equation:



The length of $\overrightarrow{AB}$ is known from the coordinates of the anchors placed in A and B. $\angle A$ and $\angle B$ are measured based on the incoming signals in A and B. The y-coordinate can then be calculated by finding P:

$$
\begin{aligned}
&\text{Trigonometry provides: } P = \left|\overrightarrow{BC}\right| \sin(\angle B)\\
&\angle C = 180^o - (\angle A + \angle B)\\
&\text{The law of sines gives: } \frac{\sin(\angle B)}{\left|\overrightarrow{AC}\right|} = \frac{\sin(\angle A)}{\left|\overrightarrow{BC}\right|} = \frac{\sin(180^o - (\angle A + \angle B))}{\left|\overrightarrow{AB}\right|}\\
&\left|\overrightarrow{BC}\right| = \frac{\left|\overrightarrow{AB}\right| \cdot \sin(\angle A)}{\sin(180^o - (\angle A + \angle B))}\\
&\\
&\text{We obtain: } P = \frac{\left|\overrightarrow{AB}\right| \cdot \sin(\angle A) \cdot \sin(\angle B)}{\sin(180^o - (\angle A + \angle B))}
\end{aligned} \tag{3.1}
$$

This proves that two anchors could provide either an x- or an y-coordinate. To determine the position of the object, at least 3 anchors are needed (R^2) and 4 anchors if the system operates with x-y-z coordinates (R^3).

3.2.3 Trilateration

Trilateration is a surveying technique based on measuring lengths creating a triangle [11], rather than angles like in triangulation. In trilateration, the angles are calculated by the law of cosines from the measured distances. Figure 3.3 shows, purely theoretical, how 1 distance measurement form a curve, 2 distances obtains two intersections and how the third distance measurement determines the correct intersection.

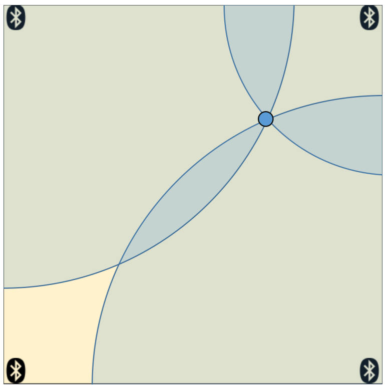


Figure 3.3: Trilateration

In practice, signals fluctuate, and the measurements are not accurate enough to re-create the scenario displayed in Figure 3.3. Lets consider the following scenario in Figure 3.4

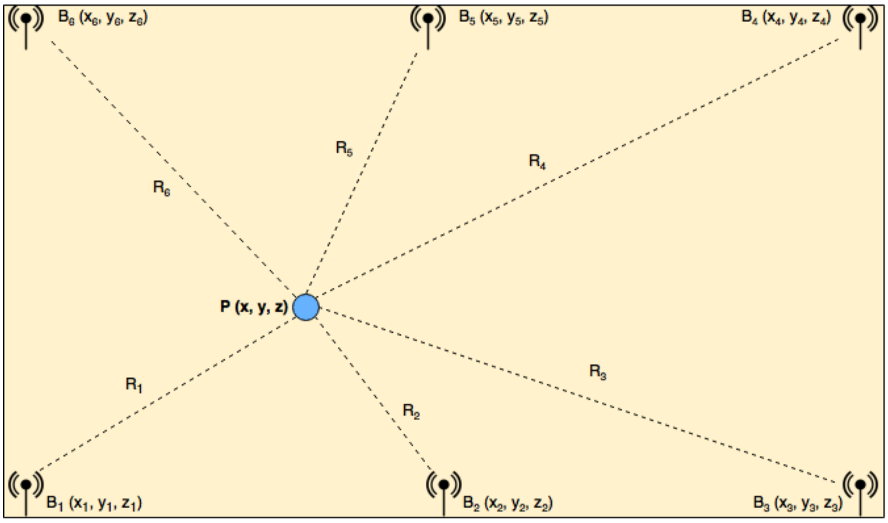


Figure 3.4: Illustration of trilateration symbols

Notations:

- Coordinates of target point P is denoted $\theta = (x, y, z)$
- The known location of beacons/anchors are denoted $B_i = (x_i, y_i, z_i)$
- The distance (d) between a beacon/anchor and the target θ can be derived as:

$$d_i(\theta) = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \tag{3.2}$$

- Choosing a known beacon/anchor as a reference point of coordinates (x_r, y_r, z_r), the distance between this reference point and an arbitrary beacon/anchor can be derived as:

$$d_i r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2} \tag{3.3}$$

- The distance between a reference and the target θ can then be derived as:

$$d_r(\theta) = \sqrt{(x - x_r)^2 + (y - y_r)^2 + (z - z_r)^2} \tag{3.4}$$

Derivation of a Linear Model:

Using the reference, adding a zero-sum into the equation, the **cosine-rule** can be applied to simplify the expression for the linear model:

$$d_i(\theta)^2 = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2$$

Adding the zero-sum:

$$\begin{aligned} d_i(\theta)^2 &= (x - \mathbf{x_r} + \mathbf{x_r} - x_i)^2 + (y - \mathbf{y_r} + \mathbf{y_r} - y_i)^2 + (z - \mathbf{z_r} + \mathbf{z_r} - z_i)^2 \\ d_i(\theta)^2 &= (x - x_r)^2 + 2(x - x_r)(x_r - x_i) + (x_r - x_i)^2 \\ &+(y - y_r)^2 + 2(y - y_r)(y_r - y_i) + (y_r - y_i)^2 \\ &+(z - z_r)^2 + 2(z - z_r)(z_r - z_i) + (z_r - z_i)^2 \end{aligned}$$

Combined with previous expressions the following is obtained:

$$d_r(\theta)^2 + {d_{ir}}^2 - d_i(\theta)^2 = 2((x_i - x_r)(x - x_r) + (y_i - y_r)(y - y_r) + (z_i - z_r)(z - z_r))$$

where $i = 1, 2, ..., n$ with $n \geq 4$

Using any beacon as a reference point (e.g B1), inserting known coordinates results in a **Linear system** by the following equation:

$$(x_n - x_1)(x - x_1) + (y_n - y_1)(y - y_1) + (z_n - z_1)(z - z_1) = \frac{1}{2}[{r_1}^2 - {r_n}^2 + {d_{n1}}^2] = b_{n1}$$

When one beacon is used as reference, the number of equations for the linear system is (n-1) where n is number of beacons[2]. And as we know, there are 3 unknowns in the system; coordinate **x, y** and **z**. The system can be written in matrix-form (**Ax=b**) as:

$$A = \begin{bmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ \vdots & \vdots & \vdots \\ x_n - x_1 & y_n - y_1 & z_n - z_1 \end{bmatrix}, x = \begin{bmatrix} x - x_1 \\ y - y_1 \\ z - z_1 \end{bmatrix}, b = \begin{bmatrix} b_{21} \\ b_{31} \\ \vdots \\ b_{n1} \end{bmatrix} \quad (3.5)$$

[2]This indicates that a system of nD-space requires n+1 beacons to determine the position of an object

3.2.4 Angle of Arrival (AoA)

Angle of Arrival (AoA) measurement is a method for determining the direction of an incoming signal. In positioning systems, the object that is preferable to track, functions as an emitter while deployed sensors/antennas measure the angle relative to an antenna array. Having multiple sensors determining angles, intersections are obtained to determine the position of an object.

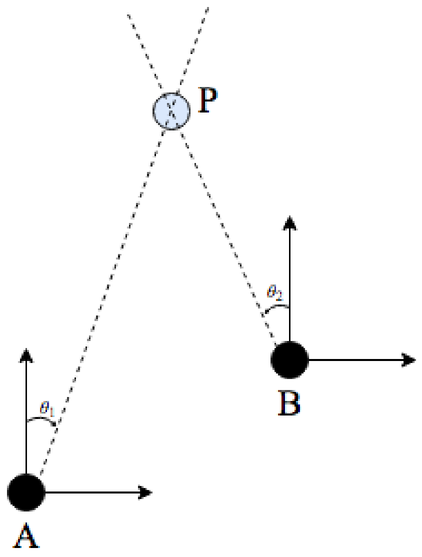


Figure 3.5: Angle of Arrival

This method is suitable for applications where there is line-of-sight between the emitter (P) and the sensors (A and B), and require large and complex hardware. The angle measurement is normally performed in one out of two ways:

- Mechanically-agile directional antennas which automatically adjust themselves towards the highest signal strength.
- An antenna-array which is made by measuring the TDOA (Time difference of Arrival) at individual elements of the antenna array.

Angle of Arrival is not a preferable method for indoor positioning as it is highly sensitive to signal reflections (multipath), decreasing both accuracy and precision.

3.2.5 Time of Arrival/Time Difference of Arrival (ToA/TDoA)

Time of Arrival is the most common surveying technique, and is also used in Global Positioning System (GPS). The method is based on knowing the exact time of transmission from the target object, the exact time of arrival at a reference point, and the speed at which the signal travels. Once these parameters are known, the distance between the target and the reference point can be calculated using the following equation:

$$d = c * (t_{arrival} - t_{sent}) \tag{3.6}$$

where c is the speed of the signal, usually the speed of light.

Further on, concepts like Trilateration can be used to determine the position of the target, requiring three reference points for 2-dimensional positioning, and four reference points for positioning in 3 dimensions.

Synchronization is an important measure when it comes to ToA, and can be achieved by:

- Exact synchronous clock on both sides (target and reference). Inaccuracy of clock synchronization directly leads to imprecise positioning.
- Using a common reference point on both sides.

Having synchronized time opens up for the use of **Time Difference of Arrival (TDoA)**, measuring the time difference between the signals received at two reference points. The difference can be calculated by the following equation:

$$\Delta d = c(\Delta t) \tag{3.7}$$

where c is the speed of light and Δt is the difference of arrival times at the reference points.

In 2D, Eq. 3.7 can be used together with

$$\Delta d = \sqrt{(x_2 - x)^2 - (y_2 - y)^2} - \sqrt{(x_1 - x)^2 - (y_1 - y)^2} \tag{3.8}$$

where (x_1, y_1) and (x_2, y_2) are known coordinates of beacon 1 and 2, Δd is the distance relationship based on the time difference. Using nonlinear regression, hyberbolas are created through the equations and the position of the target can be determined by finding the intersections as shown in Figure 3.6 below.

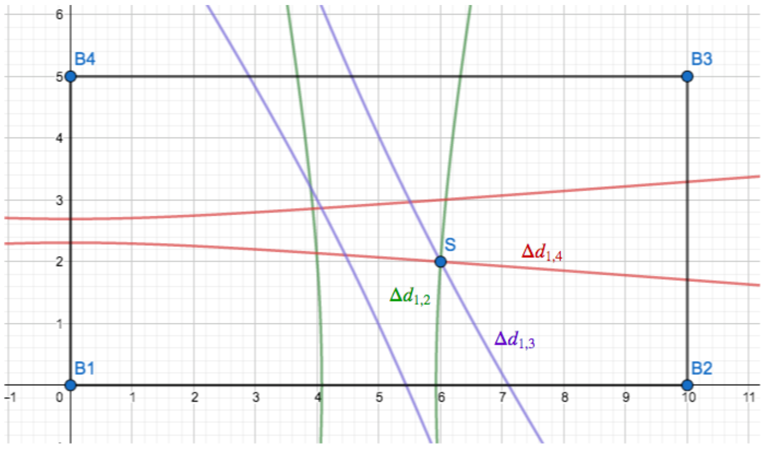


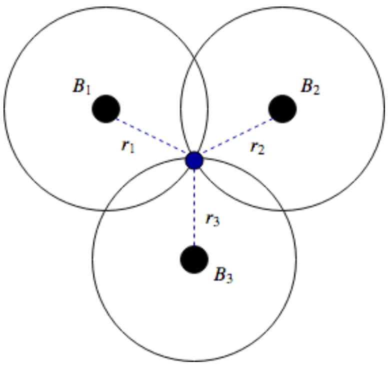
Figure 3.6: Time Difference of Arrival (TDOA)

3.3 Positioning using BLE Beacons

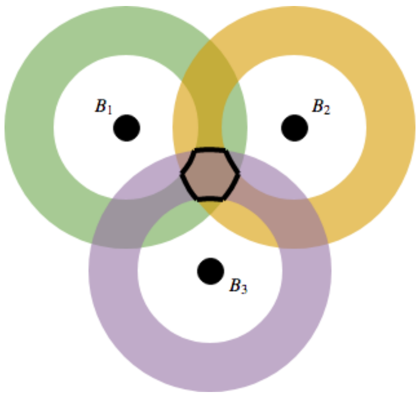
Based on the positioning concepts presented in the latter sections, only a few would be feasible together with Bluetooth Low Energy (BLE) Beacons. As presented in Section 2.2, BLE Beacons transmits data (varies with framework[3]) that is received by a smart-device such as a smartphone or a tablet. Due to limitations of the general antennas that are found in these smart-devices today, the only direct surveying technique of a single signal is calculating the distance through the RSSI-equation (Eq. 2.2). This limits bluetooth-based positioning systems to **Cell identification** and **Trilateration**.

In a bluetooth-based positioning system, the smartdevice performs a scan cycle, discovering and ranging the distance to all nearby BLE beacons. Based on the implemented software in the running application, the position of the target can be obtained. In some applications Bluetooth is also used as a supplement to other technologies to improve location services, e.g together with GPS and Wi-Fi positioning.

Using BLE Beacons, the distance measurements are essential for making a concept like Trilateration work. A theoretical approach with perfectly accurate distance measurements would result in a point position, but due to disturbances, a bluetooth positioning systems need an implementation of trilateration that handles uncertainties. As shown in Figure 3.7, three BLE Beacons are used to determine the position of the smartdevice by trilateration. The first, 3.7a, displays a theoretical approach, resulting in a point position. The second, 3.7b, shows how the possible solution is defined as an area instead of as a point.



(a) Trilateration



(b) Trilateration with uncertainty

Figure 3.7: Trilateration using BLE Beacons

[3]Eddystone by Google and iBeacon by Apple are examples of different frameworks

3.3.1 Related Work

The need of Positioning Systems for indoor applications has already been mentioned due to poor coverage of conventional GPS signals based on line-of-sight. When it comes to using RSSI measurements for positioning, there has been some former research regarding the subject using different wireless technologies.

An acoustic positioning system was developed and tested in [12]. The achieved accuracy resulted in deviances below 1cm, but due to directional transducers with relative low range, the coverage area was limited to approximately $3m^3$.

A wireless technology found in the same band as Bluetooth Low Energy, is WLAN IEEE 802.11[4], also called Wi-Fi. A real-time positioning system based on the WLAN IEEE 802.11 using Trilateration is presented in [13]. One of the most comprehensive advantages with the suggested system is that it can be implemented into existing Wi-Fi infrastructure, but the estimates of the calculated position is off by several meters.

In [14], a bluetooth positioning system is developed and tested, tracking a smartphone by using four other smartphones as reference, using the Bluetooth 2.1 standard. The accuracy of the system is not specified, but the authors suggests the use of other algorithms to solve the positioning problem.

This thesis is based on navigation with BLE Beacons, having data broadcasted from beacons to a smartphone, allowing tracking of several objects simultaneously. In [15] a Bluetooth sensor network with inverted communication flow achieves a positioning accuracy of 3 meters using triangulation. This system only allow tracking of one device at a time, and explains to what the degree the human body blocks the signal and lowers the accuracy considerably.

The related work presented above is using technology that can be related to Bluetooth Low Energy (BLE) and positioning based on signal strength measurements, which creates the basis for this thesis. BLE has advantage over other technologies due to being low-cost, highly supported, low-power consumption and light-weight for easy deployment, making it interesting to see if BLE could be a reliable technology for indoor positioning.

[4]Normally found in the frequency bands 2.4, 3.6 and 5GHz. Both version 802.11b and 802.11g are located in the ISM band along with BLE

3.3.2 Trilateration with BLE Beacons

Due to the uncertainties in the distance measurements when using BLE Beacons for positioning, it is not straightforward to solve the mathematical equations to determine the position of a smartdevice. Although there is derived a linear model for trilateration in Section 3.2.3, only obtaining estimates for the distances due to interference, it results in nonlinear equations of higher degree. Solving these with distance estimates instead of accurate distances can be performed satisfactorily through **linear least squares**, and **nonlinear least squares** solution techniques. The linear system of the trilateration problem is defined in matrix form as:

$$A\vec{x} = \vec{b}$$

$$A = \begin{bmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ \vdots & \vdots & \vdots \\ x_n - x_1 & y_n - y_1 & z_n - z_1 \end{bmatrix}, x = \begin{bmatrix} x - x_1 \\ y - y_1 \\ z - z_1 \end{bmatrix}, b = \begin{bmatrix} b_{21} \\ b_{31} \\ \vdots \\ b_{n1} \end{bmatrix} \tag{3.9}$$

The linear system above has (n−1) equations in three unknowns (x , y , z). The system could also be reduced into a two-dimensional (R^2) system, resulting in (n-1) equations in two unknowns (x , y). This requires the presence of four beacons (n=4) in a three-dimensional system (R^3), and three beacons (n=3) in a two-dimensional system (R^2).

Linear Least Squares

As the distance measurements used as input to solve the trilateration problem only are estimates, it results in a solution containing several points (area) instead of a point as shown in Figure 3.7. Using Linear Least Squares as solution technique, the sum of the squared differences between the data values and their corresponding modeled values is minimized, providing a position approximation. Since all measured distances to each beacon are estimates, the problem requires the determination of $\vec{x}$ such that $A\vec{x} \approx \vec{b}$. As described in the technical report [16], this is done through minimizing the sum of the squares of the residuals,

$$S = \vec{r}^T\vec{r} = (\vec{b} - A\vec{x})^T(\vec{b} - A\vec{x}) \tag{3.10}$$

which leads to

$$A^T A\vec{x} = A^T\vec{b} \tag{3.11}$$

Equation 3.11 can be solved for $\vec{x}$ in several ways. The condition number of $\mathbf{A^T A}$ determines which method is best. If $\mathbf{A^T A}$ is non singular and well-conditioned then

$$\vec{x} = (A^T A)^{-1} A^T \vec{b} \tag{3.12}$$

If $A^T A$ is singular or poorly conditioned, QR Factorization of matrix A can be used to decompose it into the product $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix. From the QR Factorization, the solution for $\vec{x}$ can be found by back substitution when $\mathbf{A}$ is full rank from Equation 3.13:

$$R\vec{x} = Q^T \vec{b} \tag{3.13}$$

Although matrix $\mathbf{A}$ is not close to singular, it may happen that $\mathbf{A^T A}$ is. This could result in a problem that is not possible to solve through QR Factorization and has to be solved through Singular Value Decomposition (SVD).

Nonlinear Least Squares

When using estimates for the beacon distances in trilateration, Nonlinear Least Squares method provides higher accuracy than the Linear Least Squares method [16]. One of the main differences between Nonlinear Least Squares (NLSQ) and Linear Least Squares (LLSQ) is that while LLSQ can be solved through direct methods, NLSQ is an iterative process. In NLSQ, the least square estimate needs to be obtained through successive approximation, starting with linearization of the model by approximation to a first-order Taylor-series about β^k, where β is the least square estimate and k is the iteration number. Using the same linear model as in LLSQ from (3.9), NLSQ can be used with the following restrictions for achieving higher accuracy than LLSQ:

- Positioning with NLSQ restricts the smartdevice of only being positioned at one side of the plane that the BLE beacons is deployed. In practice this means only determining z-coordinates which are below the z-coordinate of the beacons. This is due to NLSQ may have multiple minima in the sum of squares, while in LLSQ the solution is unique.
- The smartdevice will have to be inside the perimeter of the installed beacons. Both elevation changes and moving further away from the perimeter will decrease the accuracy.

These restrictions are mentioned in [16] as measures for conserving high accuracy, although the Nonlinear Least Square method **will** provide results if these restrictions are violated.

In NLSQ, the sum of the squares of the errors on the estimated distances is minimized. The estimated (measured) distances are denoted r_i, exact distance is denoted d_i, i.e

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = d_i^2 \tag{3.14}$$

To minimize the sum of the squares of the errors on the estimated distances, one must minimize the following function:

$$F(x, y, z) = \sum_{i=1}^{n} (d_i - r_i)^2 = \sum_{i=1}^{n} f_i(x, y, z)^2, \tag{3.15}$$

where

$$f_i(x, y, z) = d_i - r_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} - r_i \tag{3.16}$$

The next step is differentiating (3.15) with respect to x yields

$$\frac{\partial F(\theta)}{\partial x} = 2\sum_{i=1}^{n} f_i \frac{\partial f_i(\theta)}{\partial x} = 2\sum_{i=1}^{n} f_i \frac{\partial d_i(\theta)}{\partial x} \tag{3.17}$$

The equations for the partials with respect to y and z are similar. Let two vectors be defined as:

$$f(\theta) = \begin{pmatrix} f_1(\theta) \\ f_2(\theta) \\ \vdots \\ f_n(\theta) \end{pmatrix}, \nabla F(\theta) = \begin{pmatrix} \frac{\partial F(\theta)}{\partial x} \\ \frac{\partial F(\theta)}{\partial y} \\ \frac{\partial F(\theta)}{\partial z} \end{pmatrix} \tag{3.18}$$

and define the Jacobian matrix **J** as:

$$\mathbf{J}(\theta) = \begin{pmatrix} \frac{\partial d_1(\theta)}{\partial x} & \frac{\partial d_1(\theta)}{\partial y} & \frac{\partial d_1(\theta)}{\partial z} \\ \frac{\partial d_2(\theta)}{\partial x} & \frac{\partial d_2(\theta)}{\partial y} & \frac{\partial d_2(\theta)}{\partial z} \\ \vdots & \vdots & \vdots \\ \frac{\partial d_n(\theta)}{\partial x} & \frac{\partial d_n(\theta)}{\partial y} & \frac{\partial d_n(\theta)}{\partial z} \end{pmatrix} \tag{3.19}$$

Based on (3.18) and (3.19) the following equation needs to be solved:

$$\triangledown F(\theta) = 2\mathbf{J}(\theta)^T f(\theta) = 0 \tag{3.20}$$

where

$$\mathbf{J}(\theta)^T f(\theta) = \begin{pmatrix} \sum_{i=1}^{n} \frac{(x-x_i)f_i(\theta)}{d_i(\theta)} \\ \sum_{i=1}^{n} \frac{(y-y_i)f_i(\theta)}{d_i(\theta)} \\ \sum_{i=1}^{n} \frac{(z-z_i)f_i(\theta)}{d_i(\theta)} \end{pmatrix} \tag{3.21}$$

There are several algorithms available for minimizing the sum of the square errors, such as Newton iteration. The **Newton Algorithm** uses an initial guess for the vector, $\vec{R}$, containing the position to search for the true position. The Newton algorithm is given by

$$\vec{R}_{\{k+1\}} = \vec{R}_{\{k\}} - (J_{\{k\}}^T J_{\{k\}} + A_{\{k\}})^{-1} J_{\{k\}}^T \vec{f}_{\{k\}} \tag{3.22}$$

As calculating the Hessian matrix **A** is computionally intensive, the **Gauss-Newton Algorithm** can be used if **A** is small compared to $\mathbf{J^T J}$. Solving positioning using trilaterating allows the use of Gauss-Newton, ignoring the Hessian matrix **A**, making the Gauss-Newton algorithm equivalent to the Newton Algorithm giving

$$\vec{R}_{\{k+1\}} = \vec{R}_{\{k\}} - (J_{\{k\}}^T J_{\{k\}})^{-1} J_{\{k\}}^T \vec{f}_{\{k\}} \tag{3.23}$$

If the matrix-product $\mathbf{J}_{\{\mathbf{k}\}}^{\mathbf{T}} \mathbf{J}_{\{\mathbf{k}\}}$ can occur as singular or ill-conditioned, (3.23) can be modified to

$$\vec{R}_{\{k+1\}} = \vec{R}_{\{k\}} - (J_{\{k\}}^T J_{\{k\}} + \lambda_{\{k\}} D_{\{k\}})^{-1} J_{\{k\}}^T \vec{f}_{\{k\}} \tag{3.24}$$

known as the **Levenberg-Marquardt Algorithm**. $\lambda_{\{k\}}$ is the Levenberg-Marquardt parameter and $D_{\{k\}}$ is a diagonal matrix with positive elements along the diagonal.

Chapter 4

Design

This chapter will present the design of the system and all concepts that needs to be realized to cover all key-aspects of the problem formulation. The design is basically an software-extension to the **ePark** application, enabling location services through bluetooth technology.

To clarify some important terms:

- **Scanner:** A smart device scanning for Bluetooth Low Energy Beacons, receiving their unique ID. In this case, it is represented as a smartphone running the ePark application.
- **Advertiser:** A Bluetooth Low Energy Beacon that is transmitting data based on the configured framework, as mentioned in Section 2.2.
- **Cloud Storage:** As mentioned in Section 2.4, ePark runs an Ubuntu 16.04 Server as cloud storage. Through a MySQL database, the application is user-dependant instead of device-dependant, meaning that a user could bring the account forward to e.g a new phone or similar.

4.1 Introduction

As described in the problem formulation, one of the main objectives for this work is to develop an application that provides location services inside a parking garage. Due to poor coverage of daily-basis technologies like GPS, the location service will be achieved through Bluetooth technology.

The initial development of **ePark** carried out during the fall of 2017 will be the foundation for both the design and implementation of the location service. Below is a scenario which substantiates the design of the system.

4.1.1 Functionality

1. ePark is downloaded and installed on the users smartphone and configured with personal login.
2. The user enters a public parking garage, ePark should then provide:
    - Location service with map-preview of the parking garage and current position of the user
    - Interactive status coloring of available parking spots
3. The user then chooses parking based on the information provided by the application, reserves and pays for the parking, everything in one application
4. The application should also contain history of previous parkings, making it simple for the user to keep track of parkings and costs.

4.1.2 Preliminary decisions on design

To create a framework for the design, the following decisions are made:

- **Development platform:** Android
- **BLE Beacon Framework:** Eddystone UID
- **Beacon Provider:** Radius Networks
- **Cloud Storage Provider:** Digital Ocean

These decisions are embracing the system design with important boundaries.

4.2 System Design

The system design is based on a smartphone (scanner) interacting with BLE Beacons (advertisers). The interaction is handled within the Android application ePark, which also performs all computation of position data. Interaction occurs in the following ways:

- BLE Beacons $\longrightarrow$ Smartphone: Every beacon broadcasts its identifiers which is received by the smartphone. Distance-calculations can be performed on the received signal.
- Smartphone $\leftrightarrow$ Cloud Storage: The smartphone can poll, manipulate and insert data into the MySQL database functioning as Cloud Storage.

BLE Beacons can as described in Section 2.2 be configured with one or several frameworks (iBeacon, Eddystone). As this work is intended as an extension to ePark, it will keep its initial specifications, using **Eddystone UID** as the preferable chosen framework. This design choice implies that the following data is received in a transmission between a beacon and a smartphone:

- **UUID:** Set as a 16-bit hexadecimal value by the beacon provider (Radius Networks)
- **namespace ID:** Grouping beacons in larger groups, for instance the floor they are installed
- **instance ID:** Distinguishing between beacons within a namespace group, giving all beacons an unique ID combination

The combination of UUID, namespace ID and instance ID is referred to as the address of a beacon. This address should be used by the smartphone to poll the physical installation coordinates from the MySQL database. This design allows the smartphone to calculate its position based on known locations of beacons along with distance-calculations to these beacons. Based on the positioning concepts outlayed in Section 3.1, this design makes it possible to implement positioning algorithms for both **Cell Identification** (Section 3.2.1) and **Trilateration** (Section 3.2.3).

The figure below displays how the transmissions of data from beacons can be used to determine the position of the smartphone (represented as the blue dot). Independentantly of the chosen algorithm, the computed position can be put as an overlay over a layout drawing of the parking garage. Trilateration and Cell Identification can also be used combined. Then the Trilateration algorithm will determine the position of the smartphone, and when parking, Cell Identification determines the correct parking spot (e.g cell P8).

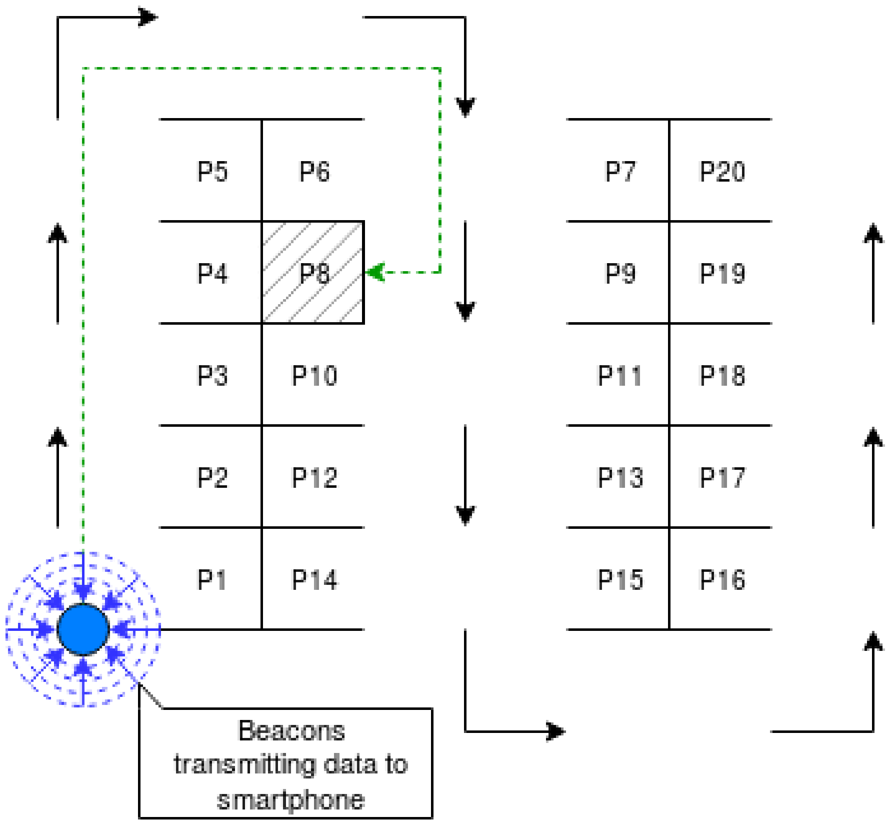


Figure 4.1: Functionality Design

A scenario for the use of the ePark application would consist of:

1. The user arrives at the entrance of the parking garage and runs the application:
    - The application provides a location service with a GUI that displays current position and available spots
    - High refresh rate of the GUI provides a smooth location service
2. The user parks at an available spot and enters the 'parked' state by touching a button:
    - The application registers the parking in the Cloud Storage
    - Based on business-model, payment can be performed at the time of check-out, or as elapsed-time-based micro-transactions

4.3 Hardware Design

This section will cover the hardware requirements for realization of the system and the choices made to accommodate these requirements.

4.3.1 Hardware Requirements

Section 4.2 roughly described the system as a whole. Choosing reliable, compatible and correct hardware is fundamental for the realization of the system. The hardware components necessary for realization are:

BLE Beacons

Bluetooth Low Energy Beacons are to be deployed at the designated area which a location service is to be used. The following measures are important for achieving the bluetooth-coverage needed:

- **Transmission power:** The specified Tx-value for the BLE Beacon determines the possible range of the signal (Table 2.1)
- **Advertising rate:** Determines the system broadcasting latency
- **Power-source:** Plugged power-source increases the cost of the system drastically, therefore battery-power is preferred in most cases. The battery-life of the chosen BLE Beacon affects the maintainability of the system
- **Encapsulation:** The BLE Beacon has to be encapsulated to withstand the environments of deployment
- **Deployment:** The beacons should be deployed at a height of 2.5 - 4 meters to reduce interference from moving cars and humans. Every inch of the area should be covered by at least 4 beacons.

When choosing beacons, Eddystone UID support is required. Before installing the beacons at their location, a calibration routine is necessary to achieve accurate results. This calibration routine is performed by measuring the RSSI@1m, putting the measured dBm into the Eddystone UID framework.

Smartdevice

To run the ePark application and receive signals from the BLE Beacons, an Android smartdevice such as a smartphone or tablet is required. The following requirements are needed for realization of the system:

- Android OS - Minimum API lvl 21 (Android Lollipop)
- Support of Bluetooth Low Energy [3]
- Allowing low-level modification of the software that controls the Bluetooth service hardware.

4.3.2 Hardware Choices based on design

Based on the requirements for the beacons and the smartdevice, both cost and availability affects the design choices. To eliminate the possibility of poor performance due to chosen hardware, each hardware component will run as duplicates - meaning that there will be used at least two models for both beacons and smartdevice. By comparing the requirements to recommended products, the following hardware components are chosen:

- **Smartdevices:**
  1. Huawei Mediapad T3 10 LTE (tablet)
  2. Huawei Honor 7 PLK-L01 (smartphone)
- **Beacons:**
  1. Radius Networks RadBeacon Dot
  2. Nordic Semiconductor nRF51822 Beacon

The components will be a part of the system through the entire development. The performance of each hardware component will be evaluated to discover the need of necessary changes in design to accomplish optimal results.

4.4 Software Design

The software of the system has to work as an interpreter between the smartdevice and the beacons, making use of the bluetooth signals. To make this feasible, the following requirements are needed:

- Software support for Bluetooth Low Energy [3]
- Compatible libraries for using BLE [5], MySQL communication and for solving nonlinear mathematical equations

Meeting these requirements, the software will be developed with great focus on modularity, making it easy to re-use and extend the software and its functionality. Achieving modularity starts of with deriving the software into layers representing the architectural pattern of the **ePark** application; as it is, and with the necessary extensions to accomplish functionality such as location services. The table below displays the layers and briefly about their content:

Table 4.1: ePark' layered architecture

User Layer The User Layer mainly consists of a graphical user interface (GUI), providing the user with useful information, and allowing user-interaction. This is the highest level of abstraction, telling the other layers what to return to the user as some form of informative object.
Resource Managing Layer This layer handles all the resources, mapping the resources to GUI objects. The layer uses raw-data from the underlying layers and translates it into readable information.
Positioning Layer The positioning layer receives beacon-data from the Bluetooth layer including relative distances to each beacon and their ID. Based on running positioning algorithm, the positioning layer uses the related coordinates for each beacon ID to calculate the position.
Bluetooth Layer Handles all communication related to the Bluetooth Low Energy protocol and the Eddystone framework. The layer scans for beacons nearby, stores them along with their distance-calculation, forwards the data to the positioning layer, and deletes them before the next scan.
Networking Layer This layer consists of the Android built-in PHP service handling all communication between the application and the cloud storage. Using the Android Volley library, string requests and JSON requests can be used to access data from the database through public html.

4.4.1 Application design

The GUI itself should run on the UI-thread within the Android application, letting the Bluetooth service, Network service and the Positioning algorithm run on separate threads in the background where they are needed. The Android UI-thread activates functionality whenever events occur (button presses, swipes etc.), and achieves the maximum performance when not interrupted by other tasks and services. Making this design choice provides a good user experience, letting the resource manager of the system pass the requested resources between the different tasks and services that are running.

The flowchart below is intended to illustrate the designed functionality and the required services that have to be run in the background.

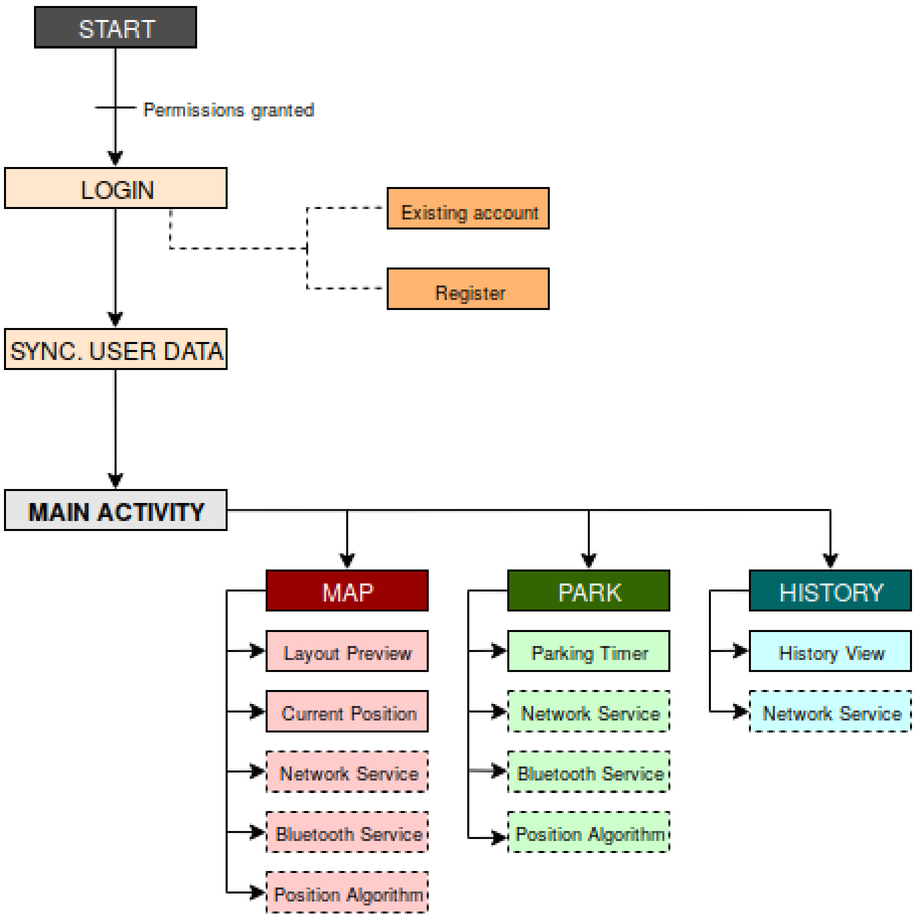


Figure 4.2: Application flowchart

1. **Start:**
    - Starting the application runs through the standard Android Lifecycle [2] displayed in Figure 2.5. The start-up also checks for necessary permissions specified in the Android Manifest. First-time users will be prompted to accept the necessary permissions, while this is remembered by the device, running the application without being prompted in the future.
    - The application should require permissions for *INTERNET, BLUETOOTH, LOCATION SERVICES*[1] and requires an active internet connection as well as bluetooth activated.
2. **Login:**
    - Users are registered in the MySQL database. Login requests determines whether the user is allowed to access the application, along with the access-level (authentication).
    - Existing users are authenticated by username and password.
    - New users has to register before login, and by registering their personal information it is possible to assign owners to the cloud resources (ownership).
    - Login provides personalization to the application, making it possible to access the same data from several devices (user-dependant instead of device-dependant).
3. **Synchronize User Data:**
    - A successfull login will send a request to the MySQL server, retrieving user-data and personal settings. This synchronization makes it possible to poll desirable parameters before entering the main activity and is easy to extend in future development.
    - The synchronization request should be able to poll strings, JSON objects and JSON arrays.
4. **Main Activity:**
    - The Main Activity is going to function as a *home-screen* of the application, providing the user with options for the various features.
    - **Main features:** Map Activity, Park Activity, History Activity.

[1]Android API level 21 and higher requires permission for Location Services to use Bluetooth

5. **Map:**

   - The Map Activity should display a scaled and user-friendly *layout preview* of the parking garage.

   - As an overlay of the preview, *current position* of the smartdevice should be displayed.

   - The interface should be refreshed at the highest rate possible to provide the best service possible. The refresh rate will be dependant on the intervals between each bluetooth scan along with the computation time of the positioning algorithm.

   - The Map Activity will be using *Network Service*, *Bluetooth Service* and the *Positioning Algorithm.*

6. **Park:**

   - The Park Activity should reserve a parking spot in the MySQL database and create a ticket with user data (ownership), parking identifiers and timestamps for both check-in and check-out.

   - Based on the created ticket, a *Parking Timer* will provide the user with the necessary information such as elapsed time, cost and location of parking.

   - The *Parking Timer* should be easy to customize for different payment models.

   - The Park Activity will be using *Network Service*, *Bluetooth Service* and the *Positioning Algorithm.*

7. **History:**

   - The History Activity should provide the user with information regarding previous completed parkings such as time and date, cost, duration and location, represented as a list.

   - The data presented in the list should be requested from the MySQL database as a JSON array.

   - The History Activity will be using *Network Service.*

4.4.2 Services

Based on the theory presented in Section 2.3.3, the Android Services [4] will be running as background tasks apart from the UI-thread. There are three main services presented in this design; *Network Service*, *Bluetooth Service* and the *Positioning Algorithm*.

Network Service:
The network service is placed in the *Networking Layer* as shown in Table 4.1, and handles all communication between the application and the MySQL database. To minimize data-usage, the network service is designed to be request-based, meaning that it will only interact with the database at certain events.

Events that triggers database request should both be programmable as they should be triggered by user interaction (buttons, swipes etc.). Based on the type of request, the network service is designed to retrive strings, JSON objects and JSON arrays from the database, using the Google Volley library [7].

Bluetooth Service:
Having the application interact with BLE Beacons, a Bluetooth Service is required. This service is intended to run in the background apart from the UI-thread, where interaction with BLE Beacons is desired. As described in Section 2.3.5, the Android Beacon Library provide the necessary support for a Bluetooth Service in Android. The service should, when active, read all present BLE signals, filter by framework (e.g Eddystone) and range the beacons - calculating the relative distance to each beacon.

Positioning Algorithm:
The system, as specified in the System Design in Section 4.2, is able to run a implementation of both Cell Identification and Trilateration to solve the positioning. To do this, a Positioning Algorithm will run as a task in the background apart from the UI-thread, solving the conceptual dependant equations. The algorithm is feeded with:

- Beacon IDs and distances (in metres)
- Beacon IDs and coordinates, either from local storage or polled from the MySQL database

Based on the running positioning concept, the algorithm should use the coordinates and distances to calculate and estimate the location of the smartdevice, pass the calculated coordinates of the smartdevice to the resource manager, updating the relevant GUI objects.

4.4.3 Database Design

The database will be configured as a MySQL database, running on the Ubuntu server provided by Digital Ocean [17]. As described in Section 2.4, the MySQL database is configured with phpMyAdmin and holds data in tables, accessible from the application through Volley-requests. To obtain a scalable system capable of handling the functionality specified in the problem formulation, Figure 4.3 shows the necessary tables and their content, designed as a relational database.

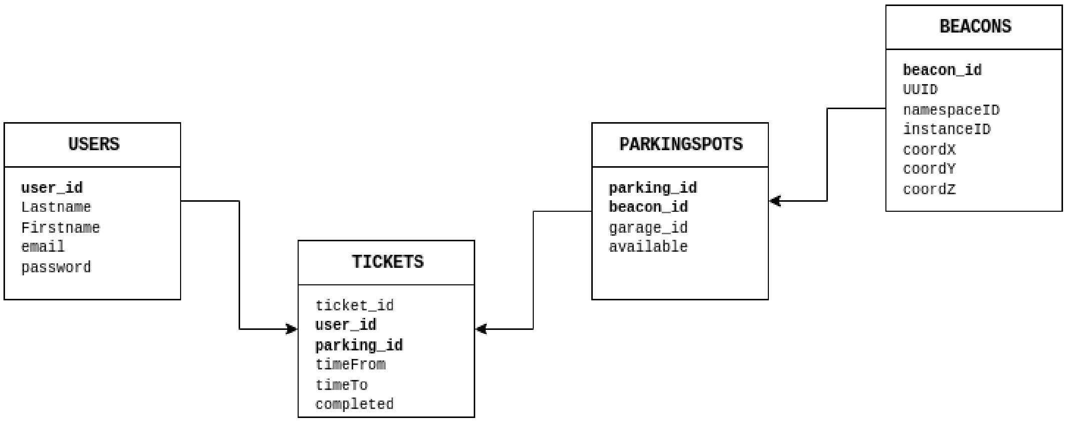


Figure 4.3: Database design

- **USERS:** The users table are required for the login and registration of the application. It also provides user-dependancy for data synchronization to the application, along with personalization and assignment of ownership to cloud resources. The `user_id` will automatically be generated by the system as a unique number for each user.
- **TICKETS:** The tickets table is a relational table, using the `user_id` from *users* and the `parking_id` from *parkingspots*. A parking request should send a MySQL query, creating a new row in the tickets table. The new row should be initialized with `timeFrom` containing the timestamp of the entry, and `completed` as `NULL` until the ticket is completed and paid for. Completing a parking should send a MySQL query that updates the row with `completed` as 1 and with `timeTo` containting the timestamp of completion. This design allows the *tickets* table to actively be used during parking, and as it contains all parkings, it can be used to provide the parking-history feature, filtered by user ID.
- **PARKINGSPOTS:** The *parkingspots* table is intended to function as a higher level of abstracion of the beacon parameters, applying the beacon identifiers with infor-

mation regarding the garage they are installed in, a unique `parking_id` in addition to the `available` parameter, whether the parkingspot is taken or available.

- **BEACONS:** The *beacons* table is intended to contain all the necessary information regarding a beacon. Each beacon should be configured uniqely and will also obtain a unique `beacon_id`. This table also contains the physical installation coordinates of the beacon, which can be polled by the application or be stored locally inside the application.

Chapter 5

Implementation

This chapter will cover the implementation of both hardware and software based on the specified theory and design, along with the necessary changes made along the way to solve all subtasks in the problem formulation.

5.1 Hardware Implementation

On of the most important factors developing a reliable, accurate and precise system for positioning is about choosing the correct hardware. The hardware chosen were listed in Chapter 4 as a part of the planned design. The following subsections will describe the implementation of:

- BLE Beacons
- Smartdevices
- Hardware deployment

5.1.1 RadBeacon Bluetooth Beacons

The chosen BLE Beacons specified in the design are provided by Radius Networks and Nordic Semiconductor. The nRF51822 [18] and the RadBeacon Dot [19] benefits from many other BLE Beacons by being battery-powered in addition to being configurable or programmable to support the desirable Beacon protocols. Table 5.1 compares the features of both the nRF51822 and the RadBeacon Dot, along with the RadBeacon USB [20] - a USB powered beacon provided by Radius Networks.

Table 5.1: Comparison of different BLE Beacons

Feature	RadBeacon Dot	RadBeacon USB	nRF51822 Beacon
Beacon Provider	Radius Networks	Radius Networks	Nordic Semiconductor
Power-source	Battery (CR2032)	USB-powered	Battery (CR1632)
Transmit Power	-20dBm to +4dBm	-20dBm to +0dBm	-20dBm to +4dBm
Advertising Rate	1Hz to 10Hz	1Hz to 10Hz	1Hz to 10Hz
Bluetooth Support	Bluetooth v4.0 (BLE)	Bluetooth v4.0 (BLE)	Bluetooth v4.0 (BLE)
Beacon Protocol Support	Eddystone UID Eddystone URL iBeacon AltBeacon	Eddystone UID Eddystone URL iBeacon AltBeacon	Fully programmable

Using the **RadBeacon Configuration App** for the RadBeacon Dot beacons, and by using the **Android-nRF-Beacon-for-Eddystone App** for the nRF51822 beacons, all beacons in the system were configured with the following parameters:

- **Beacon Protocol:** Eddystone UID
- **Transmit Power:** +4dBm, achieving maximum distance-range
- **Advertising rate:** 3 Hz
- **Namespace ID:** 8AB99F056B0874A7AF25 (generated)
- **Instance ID:** 1 to n (n is the number of beacons in the system)

To achieve accuracy in the distance-calculations, all beacons require calibration. This is also performed inside the configuration applications, completing a calibration-routine that measures the Received Signal Strength Indication (RSSI) at a distance of 1 meter. The calibrated $RSSI_{@1m}$ is put in an own data field and transmitted along with the Bluetooth data package (31 bytes). Due to interference from other communication protocols operating at the same frequency as Bluetooth Low Energy (2.4GHz ISM-band), all beacons were calibrated with approximately identical environmental conditions.

5.1.2 Smartdevices

Based on the requirements for the smartdevice presented in Chapter 4, a **Huawei Honor 7** [21] smartphone meets all the requirements and was used throughout the development. The Honor 7 features Bluetooth Low Energy [3] along with running Android version 5.0.2 (API level 21), which made the smartphone a reasonable choice during development.

During development, the ePark application is loaded and installed on the Honor 7 by using Android Studio and the supplied USB cable. Due to interaction with BLE Beacons, the functionality actually had to be tested by running the full application, making *unit-testing*[1] difficult.

5.1.3 Hardware Deployment

To make sure that signal-blocking and interference is kept at minimum, the deployment of the BLE Beacons is important. Normal conditions of use would include having moving cars and humans, making it optimal to install the BLE Beacons at a height of 3-4 meters, eliminating some signal-blocking from cars and humans.

When deploying beacons, the first step is to accurately measure the area that they are being installed in. These measurements will function both as a reference for the coordinates of each beacon, and when drawing the scaled layout preview for the Android application.

[1]Unit Testing is a way of testing code snippets and functions by using pre-defined tests, testing the functionality isolated from the rest of the system.

5.2 Software Implementation

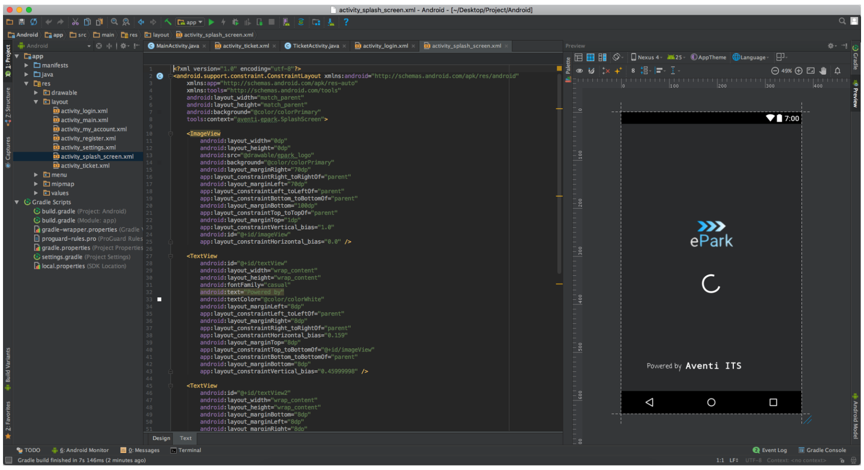
This section will describe the core of the system, the software implementation. ePark is designed and developed to meet all requirements specified in the problem formulation. The software implementation will be divided into the following modules:

- ePark GUI
- Activities
- Bluetooth Service
- Network Service
- Positioning Algorithm

The implementation of the different modules is described in the following sections.

5.2.1 ePark GUI

The graphical user interface (GUI) of ePark is written as XML-files in the Android Studio IDE. Developing the graphical user interface can be done directly in each XML-file, but also by using the built-in object oriented graphical tool in Android Studio, which automatically generates the XML-code. Through the standard Android libraries, ePark is filled with push-buttons, views, progress bars and many other customizable objects providing user-friendly interactive graphics.



```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorPrimary"
    tools:context="aventi.epark.SplashScreen">

    <ImageView
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:src="@drawable/epark_logo"
        android:background="@color/colorPrimary"
        android:layout_marginRight="70dp"
        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginLeft="70dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="100dp"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="1dp"
        app:layout_constraintVertical_bias="1.0"
        android:id="@+id/imageView"
        app:layout_constraintHorizontal_bias="0.0" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="casual"
        android:text="Powered by"
        android:textColor="@color/colorWhite"
        android:layout_marginLeft="8dp"
        app:layout_constraintLeft_toLeftOf="parent"
        android:layout_marginRight="8dp"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintHorizontal_bias="0.159"
        android:layout_marginTop="8dp"
        app:layout_constraintTop_toBottomOf="@+id/imageView"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="8dp"
        app:layout_constraintVertical_bias="0.45999998" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
```

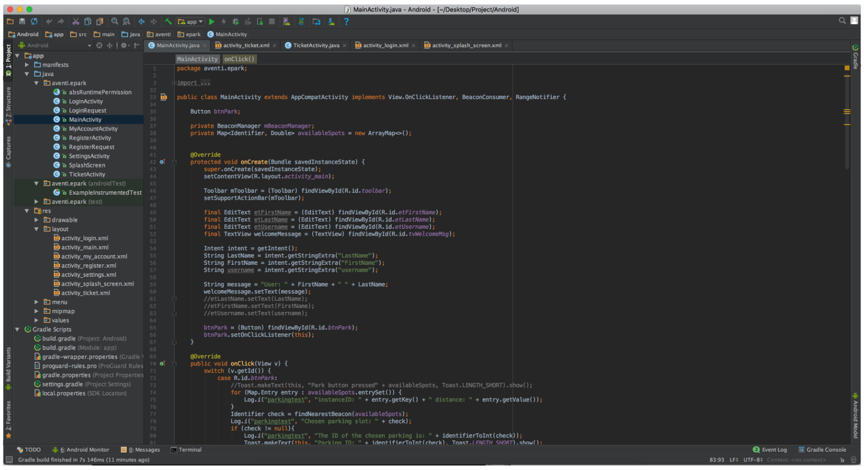
Figure 5.1: Android Studio Graphics Development with XML

Figure 5.1 displays the Android Studio IDE when developing graphics in XML. The left-side displays the editable XML code, while the right-side contains the graphics tool - letting the developer drag and drop objects, having the XML code automatically generated. Assigning a string-ID to each object makes all the objects accessible from the back-end of the application which is referred to in Section 2.3.2 as Activities.

The functionality in **ePark**, represented by the Activities, is written in Java - the standard programming language of Android Studio. The Java classes uses the ID of each XML object to achieve what is known as the front-end functionality of the application.

5.2.2 Activities

The back-end of ePark mainly consists of Activities and Services, which firstly were presented in Section 2.3. These are developed through the Java programming language as Java classes, and their behaviour is what separates them. A service normally runs in the background apart from the UI-thread, while one activity run at a time on the UI-thread connecting the back-end to the front-end.



```java
package aventi.epark;

import ...

public class MainActivity extends AppCompatActivity implements View.OnClickListener, BeaconConsumer, RangeNotifier {

    Button btnPark;

    private BeaconManager mBeaconManager;
    private Map<Identifier, Double> availableSpots = new ArrayMap<>();


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Toolbar mToolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(mToolbar);

        final EditText etFirstName = (EditText) findViewById(R.id.etFirstName);
        final EditText etLastName = (EditText) findViewById(R.id.etLastName);
        final EditText etUsername = (EditText) findViewById(R.id.etUsername);
        final TextView welcomeMessage = (TextView) findViewById(R.id.tvWelcomeMsg);

        Intent intent = getIntent();
        String LastName = intent.getStringExtra("LastName");
        String FirstName = intent.getStringExtra("FirstName");
        String username = intent.getStringExtra("username");

        String message = "User: " + FirstName + " " + LastName;
        welcomeMessage.setText(message);
        //etLastName.setText(LastName);
        //etFirstName.setText(FirstName);
        //etUsername.setText(username);

        btnPark = (Button) findViewById(R.id.btnPark);
        btnPark.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.btnPark:
                //Toast.makeText(this, "Park button pressed" + availableSpots, Toast.LENGTH_SHORT).show();
                for (Map.Entry entry : availableSpots.entrySet()) {
                    Log.i("parkingtest", "instanceID: " + entry.getKey() + " distance: " + entry.getValue());
                }
                Identifier check = findNearestBeacon(availableSpots);
                Log.i("parkingtest", "Chosen parking slot: " + check);
                if (check != null){
                    Log.i("parkingtest", "The ID of the chosen parking is: " + identifierToInt(check));
                    Toast.makeText(this, "Parking ID: " + identifierToInt(check), Toast.LENGTH_SHORT).show();
```

Figure 5.2: Java development in Android Studio

Figure 5.2 shows the Java development in Android Studio. Based on ePark' design, the Activies are implemented as Java classes for the purpose of:

- Looping through the Android Activity Lifecycle vizualised in Figure 2.5, Section 2.3.2
- Connecting a XML view, displaying information to the user, where specified in the `@Override`

- Running background tasks such as Services
- Handling user-interaction, updating the GUI

By writing a `@Override` on of the methods in the Activity Lifecycle, event-handling and functionality can be bound to certain points in the lifecycle. By overriding the `OnCreate`-method, it is possible to program the first actions of each Activity.

Code Implementation example:

```
@Override
protected void OnCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ...
}
@Override
protected void onPause() {
        beaconManager.unbind(this);
}

@Override
protected void onResume() {
    beaconManager = BeaconManager
    .getInstanceForApplication(this.getApplicationContext());

    beaconManager.getBeaconParsers().add(new
       BeaconParser().setBeaconLayout(EDDYSTONE_UID_LAYOUT));

    beaconManager.bind(this);
    ...
}
```

In the example above, the `OnCreate`, the `OnPause` and the `OnResume` methods are overridden. In the `OnCreate`-method, the selected view that is displayed to user is set through the `setContentView(R.layout...)` function. The Bluetooth Service in ePark is controlled by a beacon manager, described in Section 2.3.5. The beacon manager binds the Bluetooth Service to an activity and starts ranging for BLE Beacons. Placing this in the `OnResume` and `OnPause` methods prevents the Bluetooth Service of running when it is not needed.

Implementation of the activities is done according to the specified design in Section 4.4, with Main Activity, Map Activity, Park Activity and History Activity.

Main Activity:
The Main Activity is runned after a successful passing of the login and user-synchronization. It is developed as a home-screen of the application, featuring options for running other activities, logout and settings. Synchronization of user-data makes it possible for the Main Activity to pass data used for ownership to other activities.

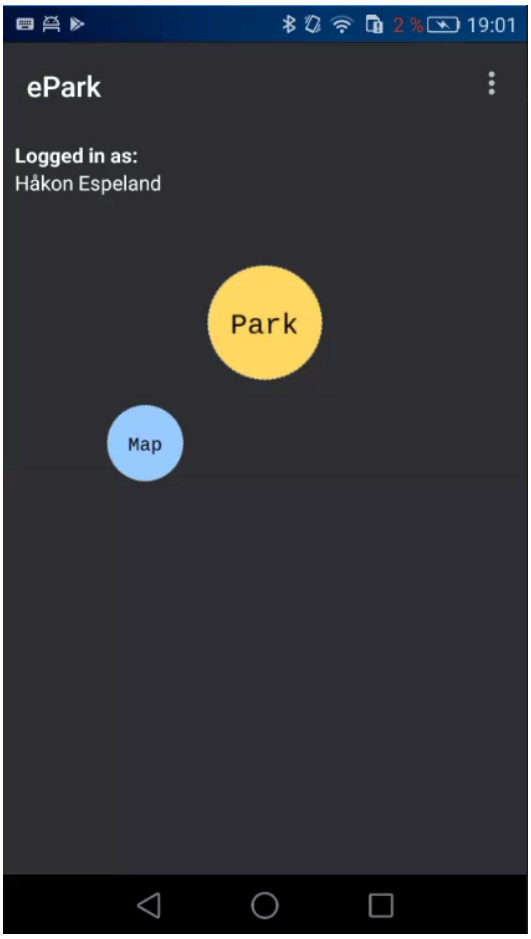


Figure 5.3: Main Activity in ePark

Map Activity:

The Map Activity is implemented containing a scaled layout preview (map) of the parking garage, a moving dot representing the current position and a button which will perform a parking request, reserving the actual parking spot in the *parkingsspots* MySQL table. As previously illustrated in Figure 4.1, the Map Activity uses:

- Bluetooth Service - For interacting with BLE Beacons, calculating the relative distance to each beacon
- Network Service - Request beacon coordinates from the cloud storage if not stored locally
- Positioning Algorithm - For calculating and updating the current position on the layout preview

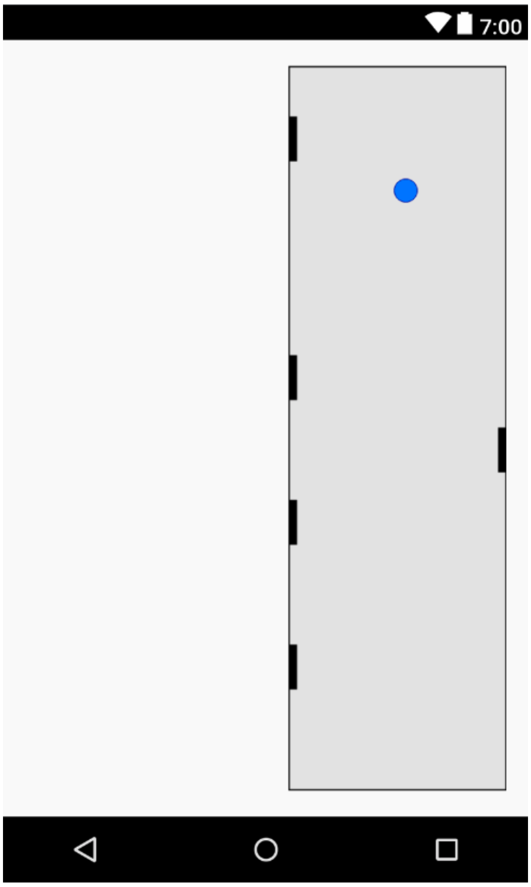


Figure 5.4: Map Activity of testarea

Park Activity:
The Park Activity is run as a result of a park request from the user. By using the positioning algorithm, it determines which parking spot to be put along in the request, changing the *available* status in the MySQL table from 1 to 0. The Park Activity also features a parking clock interface, displaying elapsed time, timestamp and cost. The implementation is done according to the specified design, having the activity use the same services as the Map Activity.

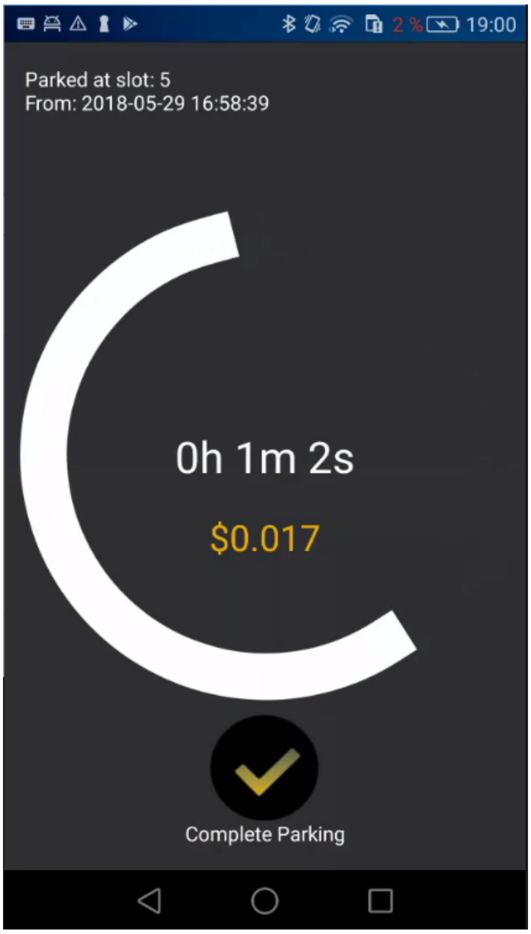
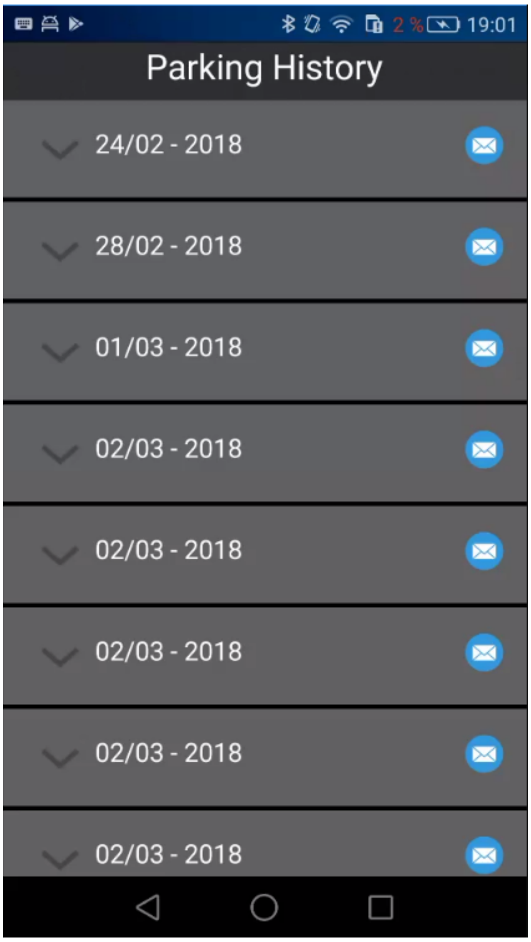


Figure 5.5: Park Activity in ePark

History Activity:
The History Activity is implemented as a list, displaying all previous parkings to the user. Each listobject contains timestamps, elapsed time and cost. The functionality is implemented by using the Network Service' JSON Array Requst - A request that polls a JSON Array from the MySQL table *tickets*, containing JSON Objects that each represents a row. The JSON Array filled with JSON Objects is filtered by the `user_id` that was synchronized preliminary to the Main Activity.



24/02 - 2018

28/02 - 2018

01/03 - 2018

02/03 - 2018

02/03 - 2018

02/03 - 2018

02/03 - 2018

02/03 - 2018

Figure 5.6: History Activity in ePark

5.2.3 Bluetooth Service

One of the main features for solving all subtasks in this project is having a background service that interacts with BLE Beacons. The Bluetooth Service is implemented through the Android Beacon Library described in Section 2.3.5.

The library is included in the **gradle build file**:

```
compile 'org.altbeacon:android-beacon-library:2.3.+'
```

Making the classes and interfaces of the library available for the application. As described in Section 2.3.5, the `Beacon Manager` manages all methods for both monitoring and ranging beacons, thus it is the only private member variable of the Java class that represents an Activity that needs to be initialized to achieve the desired functionality.

```
        private BeaconManager beaconManager;
```

Having a `Beacon Manager` placed in an activity allows modification of the classes, methods and interfaces located in the Android Beacon Library by performing an `@Override` on one of the methods. Together with the Beacon Manager, the Bluetooth Service is implemented through the `BeaconConsumer` and the `RangeNotifier` interface.

```
        public class [CLASS NAME] implements
            BeaconConsumer, RangeNotifier {...}
```

Through this implementation, the Bluetooth Service can be bound to an activity by using the methods displayed in the table below.

By overriding these methods, filters for Eddystone UID can be set through either defining a beacon-layout or by defining a Region. `onBeaconServiceConnect()` defines the RangeNotifier interface for the beacon manager, before the `didRangeBeaconsInRegion()`-method calculates the distances to the discovered BLE Beacons.

Since the `didRangeBeaconsInRegion(...)`-method gets called periodically with a period of 1s, this allows the rest of the system to handle the calculated distances as valid measurements for 1 second. All computations based on these measurements have to finish within 1 second.

Table 5.2: Bluetooth Service Methods

BeaconConsumer

Type	Method	Description
void	`onBeaconServiceConnect()`	Called when the beacon service is running and ready to accept the programmed commands through the BeaconManager.
boolean	`bindService(...)`	Called by the BeaconManager to get the context of the Activity, and bind the bluetooth.
void	`unBindService(...)`	Called by the BeaconManager to unbind the bluetooth service from the activity.

RangeNotifier

Type	Method	Description
void	`didRangeBeaconsInRegion(...)`	Called once per second to calculate the distance to each beacon in the collection.

Methods with (...) indicates that arguments are required, found in the Android Beacon Library [5]

Data Manipulation of received Bluetooth packages:

- The received identifiers from the Eddystone UID Bluetooth package was of the data type `byte-array` (Identifier)
- The instanceID of each beacon was received on the form 0xFFFFFFFFFFFF, making it necessary to convert them into integers

Having the instanceID of each beacon as integers, using the same data-type in application as in the database, made the comparison of beacons inside the application simpler. The Android Beacon Library does feature a `stringToInt()`-method, not being supported by Eddystone UID, a convertion from byte-array to integer had to be implemented as shown below:

```
    public int identifierToInt(Identifier id) {
        int sum = 0;
        String str = id.toString();
        int index = 13;
        int p = 0;
        while (index > 1 && str.charAt(index) != '0'){
            int pow = (int) Math.pow(16.0,(double) p);
            if (str.charAt(index) > '0' &&
                str.charAt(index) <= '9'){
                 sum += (((int)str.charAt(index))-48)*pow;
            }else{
                 sum += (((int)str.charAt(index))-87)*pow;
            }
            index--;
            p++;
        }
        return sum;
    }
}
```

Scheduling - Bluetooth Service:
The implementation of the Bluetooth Service intend to always provide other software modules with valid data instead of deprecated data. To guarantee this, the following criterias are implemented in the service:

- Deleting previous measurements always occurs first - Clearing the map of beacon IDs and distance-measurements
- Beacon identifiers and distance-measurements are put into the map in pairs - Making sure that the ID corresponds with the distance

Since the `didRangeBeaconsInRegion`-method is called with a period of 1 second, scheduling the Bluetooth Service is implemented as shown in Figure 5.7

- **Event 1:** Deletes the previous data and starts scanning for Bluetooth Beacons
- **Event 2:** One by one, ranges the discovered beacons, converts their ID from byte array (Identifier) to integer with the `identifierToInt(...)`-method and stores it in a map
- **Available:** The time left of a range cycle, equivalent with the time the data is valid

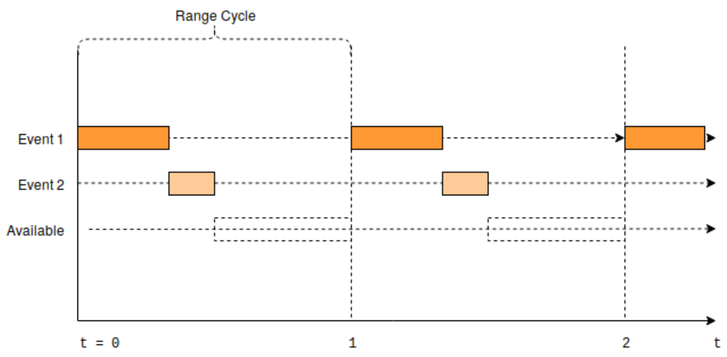


Figure 5.7: Scheduling the Bluetooth Service

This implementation makes valid data available for other software modules as frequently as possible, since the Android Beacon Library does not offer any further adjustment of the 1 second period of the `didRangeBeaconsInRegion()`-method. Other software modules using the Bluetooth Service, copies the map containing the beacon IDs and distance-measurements, making them independent of the range cycle when it comes to execution time.

5.2.4 Network Service

The Network Service is implemented according to the specified design, as a request-based service. The communication between the application and the MySQL database is triggered by events, running the desired php-request that is implemented in Android through the Google Volley library [7]. The php-request is written in Java, accessing an URL that is specified within the request.

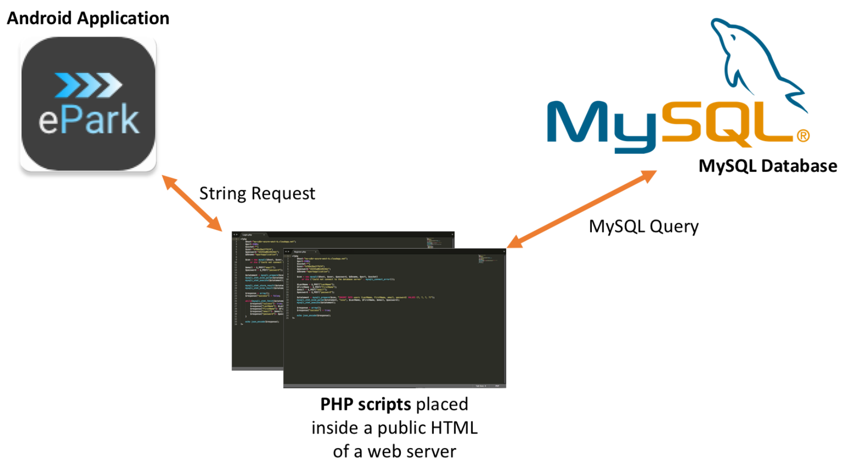


Figure 5.8: Network Service Communication Flow

A php-request could contain parameters - data to be inserted into the MySQL database. The URL that is specified within the request, contains a PHP script that runs a MySQL query, accessing the database, gaining the functionality specified in the design.

PHP scripts are stored in the server through public HTML. This implementation makes it possible to execute PHP-scripts through the URL along with the name of the file that contains the script. An example of this is the Login.php that handles the system login:

- Accessible through `https://SERVER_IP_ADDRESS/Login.php`
- The Google Volley Library passes the username and password
- Username and password are compared to the entry in the users table and returns a JSON Object
- Successful login is made if the JSON Object variable `success` contains the string "success"

Placing the files in public html could vary from server to server. This implementation runs a Ubuntu 16.04 server where public html is done by placing the 6 php files in the path: `/var/www/html`. Each file is used by a request in Android. The different request are described in Table 5.3 on the next page.

Each request returns either a JSON Object or a JSON Array. These responses are received in the Android application through a `ResponseListener`. The `ResponseListener` listens for responses from the specified URLs, and decodes the received JSON Objects.

Table 5.3: Network Service Requests

LoginRequest

Parameters	Description	Return Type
username, password	Parameters are passed to Login.php. A MySQL query compares the parameters with data in the users table.	JSON Object

RegisterRequest

Parameters	Description	Return Type
Last name, first name, email, password	Parameters are passed to Register.php. A MySQL query inserts the parameters into the users table.	JSON Object

TicketRequest

Parameters	Description	Return Type
user_id, parking_id	Parameters are passed to RequestParking.php. A MySQL query inserts the parameters into the tickets table, initialized with completed = 0 and timeFrom = CURRENT_TIMESTAMP	JSON Object

CompleteRequest

Parameters	Description	Return Type
user_id	Parameters are passed to ValidateParking.php. A MySQL query updates the entry for the current user_id where completed = 0 and alters timeTo in the tickets table with CURRENT_TIMESTAMP	JSON Object

TimingRequest

Parameters	Description	Return Type
user_id	Parameters are passed to getTime.php. A MySQL query returns a JSON Object containg the timstamps for the current user_id where completed = 0 in the tickets table.	JSON Object

HistoryRequest

Parameters	Description	Return Type
user_id	Parameters are passed to getHistory.php. A MySQL query returns a JSON Array containing JSON Objects for all complete entries for the current user_id in the tickets table.	JSON Array

5.2.5 Positioning Algorithm

The Positioning Algorithm is implemented as two Java classes, one for each implemented positioning concept, Cell Identification and Multilateration, a trilateration algorithm that can use measurements from more than 3 beacons at a time. As described in the design, the positioning algorithm, regardlessly of positioning concept, uses the beacon ID, known coordinates of each beacon and distance measurements to determine the position of the smartdevice.

Cell Identification

As illustrated in Figure 5.9 below, Cell Identification is implemented for determining the cell in the grid where the smartphone is located. Each cell has a related beacon, meaning that if the smartdevice calculates a beacon to be the closest one, it inherits the coordinates of that beacon. Through the Bluetooth Service, the `didRangeBeaconsInRegion`-method is called once every second, providing a maximum refresh rate of 1Hz. The accuracy of this implementation would be directly related to the size of the cells - greater cells equals greater accuracy.

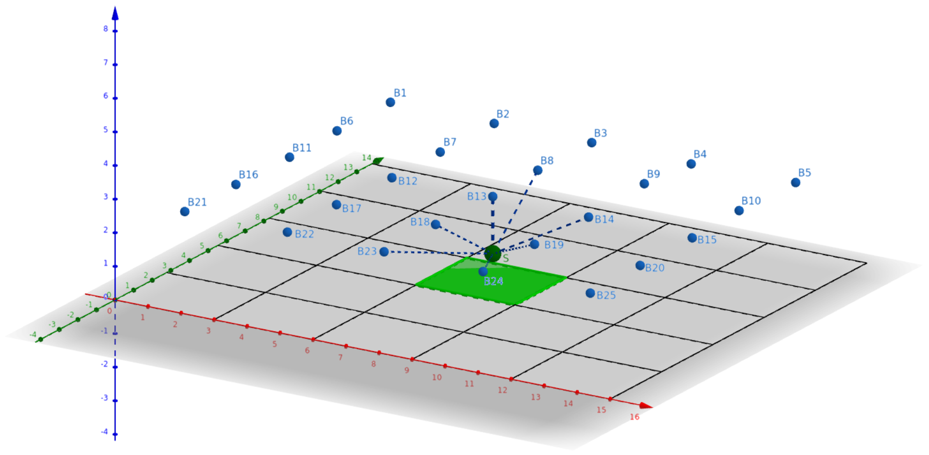


Figure 5.9: Cell Identification 3D illustration

Multilateration

As illustrated in Figure 5.10, the trilateration implementation, called Multilateration, require far less beacons than Cell Identification to cover an area. This implementation also provides a maximum refresh rate of 1Hz due to the constraints on the Bluetooth Service. The output of the algorithm is `x-y-z` coordinates, preventing the jump from cell to cell, obtaining higher positioning resolution inside the application.

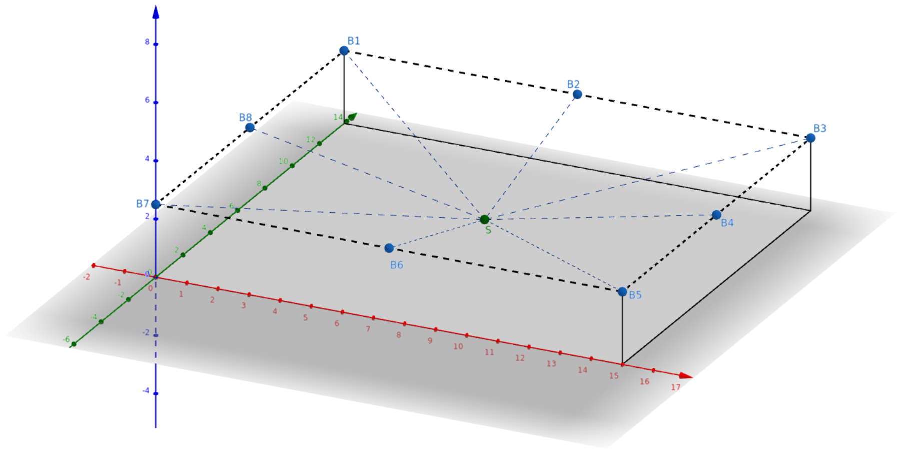


Figure 5.10: Trilateration 3D illustration

To accomodate the uncertainties of distance measurements that follows with bluetooth positioning, described in Section 3.3, the Multilateration algorithm is implemented based on nonlienar regression, fitting the distance measurements from the beacons through successive approximation. As described in Section 3.3.2, the Multilateration algorithm is implemented through a Non-linear Least Square Solver using the Levenberg-Marquardt algorithm to calculate the sum of the square errors.

The implementation was performed through the Apache Commons Math library [22], which supports solving Non-linear Least Square problems in Android using the Levenberg-Marquardt algorithm. In Java, it is implemented as three classes; `TrilaterationFunction`, `NonLinearLeastSquaresSolver` and `Multilateration`.

The Jacobian Matrix for a given point is calculated and initialized in the `TrilaterationFunction` Java class, before it is passed to the `NonLinearLeastSquaresSolver`. This is based on the equations presented in Section 3.3.2. Through non-linear regression, solvers in the `NonLinearLeastSquaresSolver` Java class are implemented to solve the Trilateration problem. A Java class named `Multilateration` is added to create a higher level of abstraction for solving the trilateration problem with methods for determining the validity of the measurements.

Positioning in the Map Activity

The Map Activity of ePark continously scans for nearby beacons, and stores their IDs together with the measured distance. Initializing instances of both `Cell Identification` and `Multilateration`, the beacon ID along with the coordinates and distance measurement are passed in as argument to get the returned value representing the current location of the target.

A marker on the layout preview updates continously along with the new coordinates received from the positioning algorithm, inheriting the coordinates according to the layout preview, as shown in Figure 5.11 below.

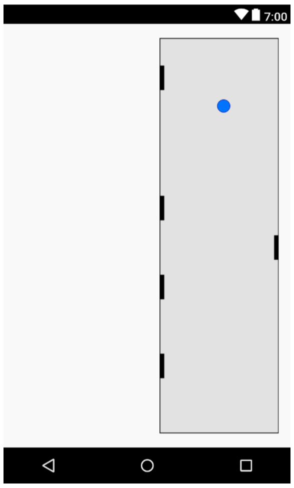


Figure 5.11: Map Activity Graphical User Interface

Chapter 6

Results

This chapter describes the results based on the subtasks and the implementation. At first, some different tests will be presented to substantiate the results of each subtask. The subtasks are denoted and referred to as **ST**.

6.1 System Testing

This section will present some tests carried out for evaluating the Bluetooth Low Energy standard as technology for use in navigation systems. The results of the system testing will provide useful information for both determining the accuracy in Received Signal Strength Indication (RSSI) measurements, and how these measurements affect the trilateration calculations in the positioning algorithm.

6.1.1 Single Beacon Range Test

The Single Beacon Range Test is intended to evaluate the accuracy of the RSSI signal received by the smartphone, transmitted by a Bluetooth Low Energy Beacon using the Eddystone UID framework. The test is based on placing a beacon in a low-interference environment at different distances, logging the measured RSSI.

Table 6.1: Single Beacon Range Test

Distance [m]	1	2	3	4	5	6	7	8	9	10
Expected RSSI [dBm]	-59	-65	-69	-71	-73	-75	-76	-77	-78	-79
Average RSSI [dBm]	-59	-73	-64	-73	-71	-70	-80	-76	-75	-79

Table 6.1 displays both the measured average and the theoretical expected Received Signal Strength Indication (RSSI). Including the errors, Figure 6.1 displays the average with error versus the expected theoretical values.

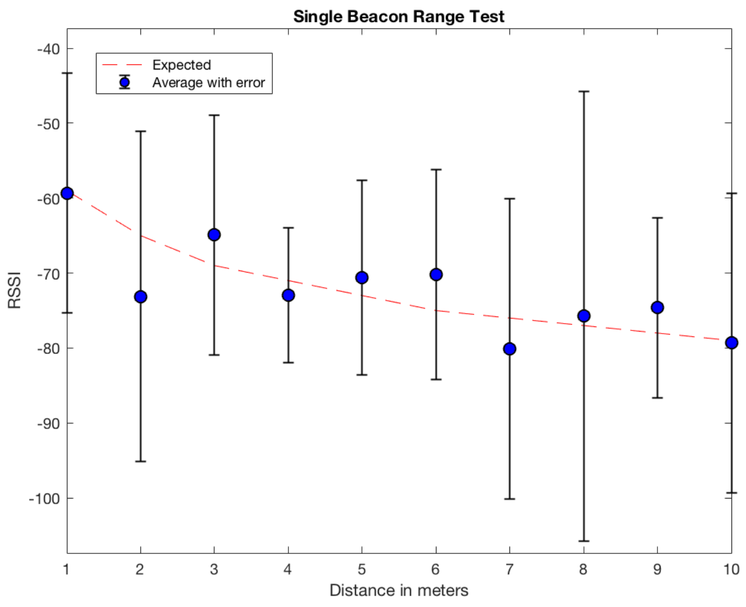


Figure 6.1: Single Beacon Range Test with error

Due to the navigation system intends to provide interactive tracking of current position, unpredictable measurements with a remarkable amount of error would create a poor basis for positioning algorithms.

6.1.2 Trilateration Test

This test is based on the input/output relationship of the positioning algorithm when Trilateration is the running positioning concept. Four beacons [B1, B2 ... B4] are installed respectively at coordinates {[0,0],[10,0],[0,10],[10,10]} forming a square as shown in Figure 6.2. The beacons are installed at a height of 1.5 meters, which allows the test to run in two dimensions (2D) when holding the smartphone at the same height. Table 6.2 displays the output of the algorithm both when the input is fixed and by using real-time measurements from the BLE Beacons.

Table 6.2: Trilateration Test

Smartdevice Coordinates [x,y]	**Fixed Input**		**Measured Input**	
	Fixed distances	**Output**	**Measured distances**	**Output**
[0, 0]	[0, 10, 10, 14.1]	[0, 0]	[1.4, 8.3, 5.4, 15.9]	[0.2, 2.3]
[5, 0]	[5, 5, 11.2, 11.2]	[5, 0]	[1.8, 5.6, 13.1, 16]	[2.6, -1.7]
[10, 0]	[10, 0, 14.1, 10]	[10, 0]	[8.9, 2.4, 13.3, 9.5]	[8.7, 0.5]
[10, 5]	[11.2, 5, 11.2, 5]	[10, 5]	[17.3, 8.9, 7.3, 12.1]	[8.3, 10.5]
[10, 10]	[14.1, 10, 10, 0]	[10, 10]	[9.5, 4.1, 9.8, 4.3]	[8.7, 4.9]
[5, 10]	[11.2, 11.2, 5, 5]	[5, 10]	[21.3, 14.9, 3.2, 6.7]	[3.2, 13.8]
[5, 5]	[7.1, 7.1, 7.1, 7.1]	[5, 5]	[3.2, 4.9, 11.4, 10.2]	[4.3, 0.4]
[0, 5]	[5, 11.2, 5, 11.2]	[0, 5]	[6.7, 10.8, 4.6, 14.5]	[-1.2, 5.1]
[0, 10]	[10, 14.1, 0, 10]	[0, 10]	[5.6, 19.4, 3.1, 9.1]	[-0.8, 7.2]

Table 6.2 shows that with a perfect theoretical approach, with fixed distances, the ouput from the algorithm corresponds 100% to the coordinates of the smartdevice. Measuring the RSSI signals and putting them into the algorithm, the calculated position is off by nearly 6 meters in several cases. Figure 6.2 displays the smartdevice's real position by the blue dots, and the calculated position by the orange dots. The number represents the relative deviance in meters, ranging from 1.2 to 5.76 meters.

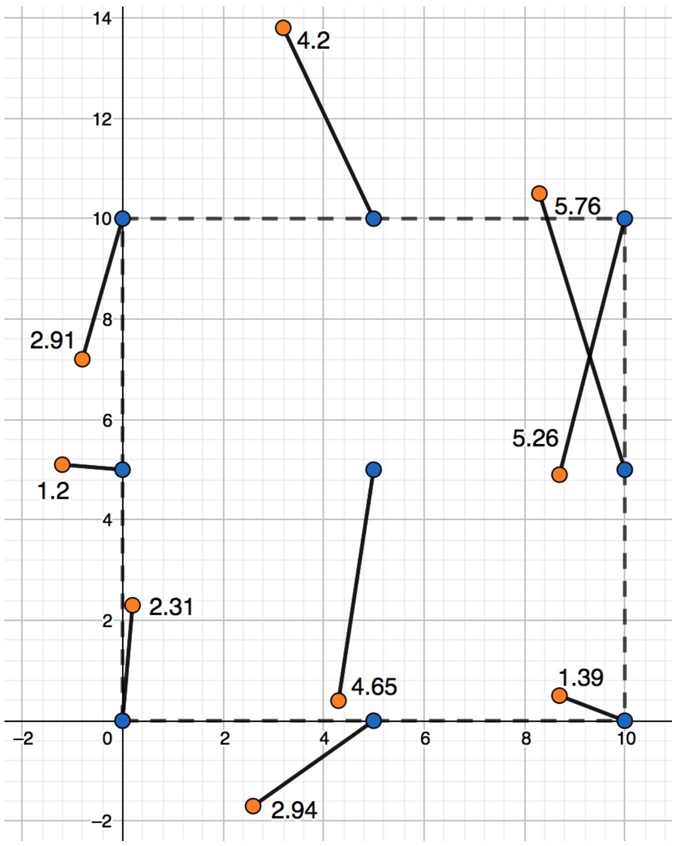


Figure 6.2: Trilateration Test showing real position versus calculated position

Both Table 6.2 and Figure 6.2 indicates that imprecise RSSI measurements directly affects the output of the positioning algorithm.

6.2 Subtask Results

The subtasks were formulated as:

1. Performing a litterature-study regarding Bluetooth Beacon Protocols and relatable algorithms for positioning.
2. Design a navigation system that uses positioning principals like Cell Identification and Trilateration through Received Signal Strength Indicator (RSSI) from BLE Beacons.
3. Implement the navigation system as an extension to the mobile application, ePark, developed during the related specialization project. Perform extensive testing to evaluate navigation using signals from BLE Beacons.

6.2.1 ST1: Theory

The litterature-study along with the forwarded theory from the preliminary development on the ePark application provided useful information regarding the Bluetooth Low Energy (BLE) standard, BLE Beacons and algorithms/concepts that can be used to determine the location of a smartdevice based on the strength of the received bluetooth signals (RSSI).

Chapter 2 provided the necessary theory and results that were obtained during the project thesis, where the ePark application first was developed. In this Chapter, the system is described as a whole, before Section 2.2 and Section 2.3 directly sums up the theory and experience acquired during the project thesis.

As of February 2018, ePark's cloud storage was moved from Microsoft Azure to Digital Ocean due to a resignation of the former membership Aventi had with Microsoft. How the cloud storage provided by Digital Ocean could be connected to ePark and what kind of server requests that were possible was described in Section 2.4, implemented as designed, providing a stable and reliable cloud solution for the application.

Chapter 3 enlightened some of the most used and widespread technologies for positioning, different positioning concepts and outlined the mathematics between each concept where necessary. Section 3.3 described why Cell Identification and Trilateration were the only two positioning concepts feasible to implement using Bluetooth Low Energy Beacons due to technological limitations, not having antennas in smartdevices capable of measuring angles.

As intended, Chapter 2 and Chapter 3 provide the information and theory required to re-create the positioning system developed during this project.

6.2.2 ST2: Navigation System

Based on the theory in Chapter 3, the following positioning concepts were deduced and illustrated:

- Cell Identity
- Triangulation
- Trilateration
- Angle of Arrival
- Time of Arrival/Time Difference of Arrival

Due to the data between the Bluetooth Low Energy Beacons and the smartdevice only flowing in one direction, the Received Signal Strength Indicator (RSSI) was the only measurement that could be brought into a positioning concept, only allowing the use of Cell Identity and Trilateration as positioning concept for the system specified in Chapter 4.

Based on the design presented in Section 4.2, the implementation of both Cell Identity and Trilateration described in Section 5.2 and the tests developed and completed as described in Section 6.1, a local navigation system was developed, divided into several Java classes. The navigation system is able to run both Cell Identity and Trilateration as positioning concept based on the build configuration, although the accuracy and precision of both concepts is below a satisfying level.

6.2.3 ST3: Android Development and Integration

Based on the work carried out during the fall of 2017, **ePark** was further developed, from providing a parking service choosing the nearest beacon, to an application featuring navigation functionality inside a parking garage.

All of the functionality specified in the application design was completed. Interacting with Bluetooth Low Energy Beacons, the Android application uses the implemented positioning algorithm to determine the position of the smartdevice by measuring the relative distance to each beacon. Using the PHP-based cloud storage, the application is able to poll data from the MySQL database such as beacon coordinates, user information during login, and

JSON Arrays to display the parking history to the user. The implemented cloud solution also allows the application to insert data into the MySQL database, creating entries for new users, new parking requests and so on.

Chapter 7

Discussion

In the previous chapters, a Navigation System using Bluetooth Low Energy beacons has been designed and implemented into the Android mobile application, ePark. Based on tests, results and experience, this chapter will discuss the system as a whole, limitations regarding performance and use the results as arguments to answer the research questions specified in the problem definition.

7.1 Accuracy of distance measurements

Developing a Navigation System that uses the Bluetooth Low Energy technology together with the Eddystone UID framework, the output containing the target coordinates is shown to be highly dependent on the input, being the distance measurements of each BLE Beacon. The distance measurements that are based on Received Signal Strength Indicator (RSSI), have through testing proven themselves to be inaccurate and extremely sensitive to interference from other wireless technologies operating at the same frequency, and interference due to blocking objects and signal reflecting materials.

Based on the **Single Beacon Range Test** in Section 6.1.1, using the RSSI equation (2.2), the accuracy of a distance measurement deviates with several meters from the real distance. Being unpredictable, it is difficult to determine whether a measured distance is valid or not. It is possible to define maximum and minimum values for each distance measurement, creating boundaries for the area where the navigation system is to be used, but having this high deviance between the real distance (Expected RSSI) and measured distance (Average RSSI), it is expected that neither of the presented positioning concepts presented in Chapter 3 would be able to produce accurate results.

As mentioned in the preliminary specialization project carried out during the fall semester of 2017, the RSSI equation (2.2) does not take into account that smartphones have different casing, and measures different RSSI at the same distance due to signal loss through the material between the antenna and the environment. To increase the accuracy for RSSI measurements, the RSSI equation (2.2) could be altered to use calculated constants for each specific smartphone, which is found through measurements and non-linear regression and stored inside the Android Beacon Library [5]. Currently, there are no more than a couple of devices supported, and the Huawei Honor 7 used in development is not supported. A supported smartphone would potentially use the specific RSSI equation with constants that are received from the library by checking the unique build number of the smartphone. If supported, the accuracy of a distance measurement might be increased a little, but due to Bluetooth Low Energy being highly sensitive to interference, determining the position of a target device through a positioning algorithm would not be remarkable of higher precision.

7.2 Navigation System Accuracy

As presented in Table 6.2, Trilateration produces accurate output when using a fixed input. This indicates that the distance measurements performed for each beacon, directly affects the output coordinates of the Navigation System. Due to inaccuracy of BLE RSSI calculations, imprecise results are produced through the positioning algorithm.

Deploying different wireless technologies to achieve higher accuracy in each distance measurement, would lead to a more precise Navigation System. The drawback with other wireless technologies is that the lightweight of the system when using Bluetooth might be lost. BLE Beacons are cheap, small in physical size, completely wireless through battery-power and easy to deploy. This makes BLE Beacons applicable where the Navigation System intend to be temporary, and the accuracy demands are reasonable.

Using Bluetooth Low Energy for positioning could still be done more accurate by using other positioning principals. The BLE Beacons are bound by the beacon protocols such as Eddystone or iBeacon. This implies that the data only flows in one direction, from the beacon to a smartdevice that only has one opportunity with the antennas used today, calculating RSSI. Ignoring beacons, using Bluetooth Low Energy featured SoCs (System on a chip) where angle measurements are supported or through accurate time synchronization, the data flow could be reverted, flowing from the smartdevice to a SoC. Then positioning principals such as Angle of Arrival (AoA) and Time of Arrival/Time Difference of Arrival (ToA/TDoA) could be used. These principals would make the Navigation System independant of the design properties of the smartdevice, i.e the signal broadcasted from the smartphone and achieved by SoCs have the same base brought into calculations, and in theory, the casing of each smartphone would not affect the end result.

7.3 Research Questions

As described in the problem formulation, the subtasks would help answering the following research questions:

1. What level of accuracy could a Bluetooth based Navigation System expect to achieve?
2. Which positioning concept provide the best results?

This section will bring forward the results and discuss how they should be interpreted to answer these questions.

Q1: What level of accuracy could a Bluetooth based Navigation System expect to achieve?

The level of accuracy for a Bluetooth based Navigation System is dependant on system design properties for data flow direction, and applicable positioning concepts. This report has focused on the use of BLE Beacons, allowing only data flow in one direction, making Cell Identification and Trilateration the only applicable positioning concepts due the RSSI measurements being the only measurements possible.

As discussed in the two previous sections, the distance measurement performed at a single beacon is quite unpredictable and is off by several meters. Using these imprecise measurements as input for Trilateration produce imprecise output as displayed in Table 6.2, and displayed in Figure 6.2 with deviances exceeding 5 meters.

The level of accuracy related to the output of the algorithm is determined by:

- Total number of beacons used in calculations
- Total sum of error in measurements compared to total sum of valid measurements

Being directly affected by the distance measurements, the **expected accuracy** for a Bluetooth based Navigation System would be dependant of several measures. Some of these measures are possible to alter for better or worse, some are difficult to handle in form of accuracy. Achieving accuracy well within 5 meters is difficult, but can possibly be achieved through the following suggestions:

- **Reasonable beacon deployment:** Making sure that the area where the beacons are deployed has minimum interference from both objects and other wireless technologies.

- **Testing for optimal number of beacons:** Due to the Bluetooth signals behaving different in different areas, extensive testing with different numbers of active beacons in the system will produce different result. Adapt the number of beacons based on performed tests at the desired location.

- **Reasonable beacon configuration:** Tweaking the signal strength of each beacon to maximum could lead to having a lot of imprecise measurements from irrelevant beacons that are far away from the target position. Dividing the area into "zones", setting the transmit power of a group of beacons to cover only that zone, could improve accuracy.

- **Signal filtering:** Filtering the signals might remove some interference and can be achieved through for instance a Kalman Filter. Being able to weight each signal, signals that we know are imprecise (i.e greatest distance), can be excluded in calculations. Machine Learning could also be used for the principal of "fingerprinting" - allowing the "machine" to learn the error image at given locations.

These suggestions are related to the use of BLE Beacons. Reverting the data flow as mentioned in Section 7.2, making the data flow from the smartdevice to a Bluetooth Low Energy featured SoC, enabling positioning concepts like Angle of Arrival (AoA) and Time Difference of Arrival (TDOA), higher accuracy could be expected, as such concepts do not determine position based on inaccurate RSSI measurements.

Q2: Which positioning concept provide the best results?

In this report, both Cell Identification and Trilateration has been explained, designed, implemented and tested. Neither of them excels at any point, but there are certain differences that imply that the choice of positioning concept should be based on the purpose of use. In a Navigation System where the intention is to achieve highest accuracy possible, Trilateration is solely the positioning concept that has potential to produce the best results, when using RSSI measurements. Lowering the accuracy demands, Cell Identification with quadratic cells that exceeds the inaccuracy in each RSSI measurement, the Navigation System would achieve higher reliability, and guarantees that the result is valid in a higher manner than by Trilateration. Based on the results of the specialization project preliminary to this master thesis, ePark was able to determine the closest beacon 9 out of 10 times, argumenting for the reliability of Cell Identification.

Each positioning concepts will provide the best results based on the purpose for where it should be applied. The main cause of determining which concept that provide the best result is the level of accuracy demanded. High prevision navigation is not achievable through BLE Beacons that uses RSSI measurements. The accuracy might increase through Angle of Arrival and Time Difference of Arrival, but the consensus is that using a technology that could produce accurate RSSI measurements, Trilateration would be a highly appropriate positioning concept to achieve high accuracy.

7.4 Integrations

When designing a system based on several platforms and interfaces, integrations are essential to the end result. Throughout the work carried out, ePark is integrating:

- Bluetooth Low Energy (BLE) Interaction through the BLE protocol using the Eddystone UID framework
- Android OS
- Cloud Storage through a MySQL Database delievered by Digital Ocean

The application developed matches the specified design with all functionality. The experience acquired during testing of the application has proved that integrations has been carried out satisfactory even though the Navigation System did not provide accuracy of an acceptable level. Software integrations has been implemented as modules, making it easy to exchange parts in the future, i.e exchanging the Bluetooth Low Energy Service with other wireless technologies that hopefully would improve the accuracy of the system. For instance, a module implementing Ultra-wide Band (UWB) could be developed as long as it passes coordinates and distances to the positioning algorithm.

Chapter 8

Conclusion

The goal of this thesis was to develop a navigation system, and implement it into a Android Parking Service providing the user with indoor navigation inside a parking garage. Along with the navigation, the parking service was intended to provide parking reservation and parking log implemented through cloud-based storage.

A navigation system was develop based on distance measurements from Bluetooth Low Energy (BLE) Beacons, radio transceivers using the BLE protocol for broadcasting specific data. Using the Eddystone UID framework, the broadcasted identifiers of each beacon made it possible to solve the positioning problem through both Cell Identification and Trilateration when measuring the Received Signal Strength Indicator (RSSI), measured by a smartphone running the ePark application. Extensive testing with the BLE Beacons was conducted, ensuring that the setup and configuration of the system provided the best results possible.

Testing both distance measurements from single beacons and positioning data ouput from the positioning algorithm revealed great weaknesses with the use of BLE Beacons for navigation. Due to imprecise and unpredictable distance measurements, deviating with several meters from the real distance, the location determination of the target device did not enter within the desirable demands when it comes to accuracy. Bluetooth Low Energy should not be written off as a potential technology to be used for navigation systems, but using lightweight beacons featuring standard beacon frameworks as Eddystone UID limits to positioning concepts that is based on RSSI measurements, and therefore high accuracy can not be expected. Using hardware that supports duplex communication, other positioning principals like Angle of Arrival and Time Difference of Arrival can be implemented, removing some uncertainty and most likely result in higher accuracy.

Chapter 9

Future Work

The navigation system developed has mainly two aspects for future work, developing the system further,

1. Exchange the current hardware with components that would potentially increase navigation accuracy through other wireless technologies or positioning concepts
2. Including more functionality, improving user-interface adding enhanced features

A suggestion for future work, improving accuracy, is to first implement hardware supporting Bluetooth Low Energy Angle of Arrival, or time synchronization making it possible to implement Time Difference of Arrival with Bluetooth Low Energy duplex communication. Re-using the concepts developed throughout this thesis, the results of these concepts can be compared to Cell Identification and Trilateration to conclude which positioning concept providing the best results. Another suggestion would be to exchange the entire system with wireless components that possibly could achieve greater accuracy in single measurements, through another wireless technology such as Ultra-wide band. Due to the results produced through fixed input in the positioning algorithm, more accurate signle measurements would improve higher accuracy.

Adding functionality inside the ePark application could for instance include:

- Algorithms for calculating the shortest path to a desirable parking spot
- Positioning data from other vehicles in the parking garage to avoid collisions

- Payment solutions, either virtual or real, either based on normal currencies or crypto-currency
- Enhanced ICT security - Scouring the system design for flaws regarding ICT security

It is clear that it is several aspects that could be improved to increase navigation accuracy as well as functionality is only limited by the imagination of the developer, and could also be improved through user-feedback.

References

[1] "Android Developer - Platform Architecture," last visited 2018-03-08. [Online]. Available: https://developer.android.com/guide/platform/index.html

[2] "Android Documentation - Android Activity," last visited 2018-02-24. [Online]. Available: https://developer.android.com/guide/components/activities/activity-lifecycle.html

[3] "Bluetooth Core Specification Version 4.0," last visited 2018-02-23. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification/legacy-specifications

[4] "Android Documentation - Android Service," last visited 2018-02-24. [Online]. Available: https://developer.android.com/reference/android/app/Service.html

[5] "Android Beacon Library version 2.3," last visited 2018-02-24. [Online]. Available: https://altbeacon.github.io/android-beacon-library/index.html

[6] "Package org.altbeacon.beacon - Android Beacon Library," last visited 2018-03-16. [Online]. Available: https://altbeacon.github.io/android-beacon-library/javadoc/index.html

[7] "Google Volley - an HTTP library for Android Networking," last visited 2018-04-27. [Online]. Available: https://github.com/google/volley

[8] "Introduction to JSON," last visited 2018-03-18. [Online]. Available: https://www.json.org

[9] "JSON Data Types," last visited 2018-03-18. [Online]. Available: https://www.w3schools.com/js/js_json_datatypes.asp

[10] R. C. Brinker and R. Minnick, *The Surveying Handbook, Ch.11 - Triangulation.* Chapman and Hall, 1995.

[11] ——, *The Surveying Handbook, Ch.12 - Trilateration.* Chapman and Hall, 1995.

[12] J. C. S. C. Medina and A. D. la Torre, "Ultrasound indoor positioning system based on a low-power wireless sensor network providing sub-centimeter accuracy," *Sensors*, vol. 13, no. 3, pp. 3501–3526, 2013.

[13] S. Mazuelas, A. Bahillo, R. M. Lorenzo, P. Fernandez, F. A. Lago, E. Garcia, J. Blas, and E. J. Abril, "Robust indoor positioning provided by real-time rssi values in unmodified WLAN networks," *IEEE Journal on Selected Topics in Signal Processing*, 2009.

[14] Y. Wang, X. Yang, Y. Zhao, Y. Liu, and L. Cuthbert, "Bluetooth positioning using RSSI and triangulation methods," in *2013 IEEE 10th Consumer Communications and Networking Conference, CCNC 2013*, 2013.

[15] J. Castano, M. Svensson, and M. Ekstrom, "Local positioning for wireless sensors based on Bluetooth/spl trade/," *Proceedings. 2004 IEEE Radio and Wireless Conference (IEEE Cat. No.04TH8746)*, 2004.

[16] W. S. Murphy, Jr., and W. Hereman, "Determination of a position in three dimensions using trilateration and approximate distances," Tech. Rep., 1995.

[17] "Digital Ocean - Cloud Computing," last visited 2018-04-28. [Online]. Available: https://www.digitalocean.com

[18] "nRF51822 Beacon by Nordic Semiconductor," last visited 2018-04-29. [Online]. Available: https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF51822-Bluetooth-Smart-Beacon-Kit/(language)/eng-GB

[19] "RadBeacon Dot by Radius Networks," last visited 2018-04-09. [Online]. Available: http://downloads.radiusnetworks.com.s3.amazonaws.com/datasheets/datasheet-radbeacon-dot.pdf

[20] "RadBeacon USB by Radius Networks," last visited 2018-04-09. [Online]. Available: http://downloads.radiusnetworks.com.s3.amazonaws.com/datasheets/datasheet-radbeacon-usb.pdf

[21] "Huawei Honor 7 - model PLK-L01," last visited 2018-04-09. [Online]. Available: https://www.gsmarena.com/huawei_honor_7-7269.php

[22] "Apache Commons Math 3.6.1 API," last visited 2018-05-05. [Online]. Available: http://commons.apache.org/proper/commons-math/javadocs/api-3.6.1/index.html