BikeApp - Detecting Cyclists Activity and Location using Bluetooth Low Energy Technology

Andriy Zabolotnyy
Instituto Superior Técnico
andriyzabolotnyy@tecnico.ulisboa.pt

ABSTRACT

In urban environments, cycling may help to solve traffic delays, interruptions and parking, while at the same time being a cleaner and healthier mean of urban transportation. Several approaches are being taken to increase cycling in urban areas. One such approach is taking an advantage of the availability of smartphones and developing applications that somehow improve and reward the cycling activity. To promote and motivate cycling in cities we propose *BikeApp*, a cycle rewarding smartphone application. *BikeApp* detects when the user starts cycling and makes him eligible for rewards that can be claimed at the shops. When the user gets close to or enters a shop, *BikeApp* detects it and allows the user to easily claim the benefit. The implementation focuses on cycling and location detection mechanisms, and its integration within the application. Both are implemented using *Bluetooth Low Energy (BLE)* technology. Our *BikeApp* application, implemented targeting the iOS platform, can accurately detect the cycling activity using a BLE bicycle-mounted sensor and determine the shop the user enters using BLE location beacons. The location detection shows to be effective even in worst-case scenarios, with location beacons located in side to side shops. We also tested *BikeApp* in terms of its power consumption, and there is no significant overhead introduced with the BLE communication.

KEYWORDS

BLE, Bluetooth Low Energy, GATT, App, iOS

1 INTRODUCTION

The use of a bicycle as a regular mode of transport in the urban environment is constantly increasing, as it may solve a lot of problems such as traffic delays, interruptions, parking, and also contributes to the reduction of greenhouse gas emissions and brings health benefits. Great effort was put into cycling promotion over the past years, for example, expansion of bicycle lane network and bicycle racks across the city.

Currently, a smartphone is a widely used communication device, with a rich set of built-in sensors such as an accelerometer, digital compass, gyroscope, GPS and camera. These sensors enable new applications across multiple domains, such as healthcare, social networks, safety, etc.

With the availability of powerful smartphones with embedded sensors many personal sensing applications appeared, focused on data collection and analysis. A typical scenario is tracking the user's exercise routines and encourage the user to reach his personal fitness goals. Most of the cycling applications follow the same idea of being a personal fitness trainer, but they also have a goal of contributing to the cycling promotion.

1.1 Goals

The goal of this project is to design, implement and evaluate a smartphone cycling application to encourage citizens to ride their bikes. The application provides users a way of being rewarded for cycling, with benefits (e.g. discounts) at shops that join the rewarding program and also benefit from cycling customers.

To guarantee correct rewards, the application needs to detect the cycling activity, and determine the exact user location, to ensure that a user is within a given shop. Indoor localization is challenging, because accurate positioning using Global Positioning System (GPS) is not possible inside the buildings (indoor spaces), and even for outdoor location the accuracy of GPS is sometimes not enough, so alternative techniques should be used. Activity detection is also problematic and several challenges arise, such as battery consumption, response time or strong requirements of wearable devices.

The solution must fulfill the following functional requirements:

- ***Detect cycling activity***
- ***Detect indoor locations***: detect user's presence within a certain shop
- ***Fully offline***: require no Internet connection, consequently all the computations should be performed locally at the mobile device, without any computation on a centralized server side. Opportunistic Internet connection updates the data stored on the device, required for offline operation.

Regarding the non-functional requirements, our solution must fulfill the following:

- ***High accuracy and precision on cycling activity detection***
- ***High accuracy and precision on indoor location detection***
- ***Scalable location detection***: positioning coverage area should be easily extended and do not limit simultaneous usage by multiple users
- ***Ease of deployment***: system should be easily deployed and without complex and time consuming initial configuration
- ***Minimal maintenance***: require minimal future maintenance or calibration
- ***User-friendly and minimally intrusive***: easy to use and require minimal user input
- ***Low battery consumption***: used technologies should be chosen taking into consideration smartphone's and additional device's power consumption
- ***Responsiveness***: user should receive quick feedback

Our proposed solution named *BikeApp*, is a mobile application intended to be used by cycling users. The application shows the user nearby shops where the user can receive benefits for cycling. To detect the cycling activity, *BikeApp* uses a BLE bicycle-mounted

sensor. After cycling to a given shop, *BikeApp* is capable of detecting the shop and prepares the screen to allow the user easily claim his benefit. The location detection uses BLE location beacons deployed at the shops.

BikeApp implementation uses *Biklio*[1] as its base project. *Biklio* is a production application of the *TRACE*[2] project, with the previously described cycling rewarding capabilites, but with alternative and less accurate cycling and location detection mechanisms. Thus, our main focus is the implementation of the cycling and location detection mechanisms using BLE devices, and its integration within *Biklio*.

1.2 Current Solutions

Currently, there are several indoor positioning techniques. However, many of them are not applicable to our solution, because of their particularities which conflict with our system requirements. The following enumeration specifies some of such techniques, together with the reasons why they are not suitable:

- **Near Field Communication (NFC) -** not all smartphone models support NFC (see Section 2.1.1);
- ***Proximity detection using WiFi networks* -** may have poor accuracy and requires periodic calibration (see Section 2.1.2)
- ***Fingerprinting* -** complex initial setup and high computation complexity (see Section 2.3)

Regarding the cycling detection, there are also some existing solutions, but most of them use the accelerometer data collected from the smartphone or some wearable sensors, and the cycling activity is inferred using supervised learning (see Section 2.4). This approach requires heavy computations in a continuous fashion, which consumes a lot of battery power and does not meet our requirement.

1.3 Contributions

This work makes the following contributions:

- Architectural design of the cycling detection mechanism using BLE bicycle sensors;
- Architectural design of the location detection mechanism using BLE location beacons;
- Implementation of both mechanisms;
- Integration of the mechanisms within *Biklio*, resulting in a new application version called *BikeApp*;
- Evaluation of configurable BLE beacon's parameters to determine the optimal system configuration to meet our requirements;
- Evaluation of the cycling and location detection mechanisms;
- Evaluation of the usability of *BikeApp* with end-users.

1.4 Document Structure

The rest of this document is organized as follows. Section 2 provides a background on existing location and cycling detection techniques, detailedly analyzing its applicability within our solution. Section 3 starts by providing a solution overview, then introduces the architecture of *Biklio* and *BikeApp*, together with the cycling and location detection mechanism explanations. Section 4 addresses the implementation details of our proposed solution. Section 5 covers the taken evaluation process. Finally, Section 6 concludes this work and presents the future work.

2 RELATED WORK

This section overviews the state-of-the-art of the project's main features: location detection (indoor positioning) and cycling activity detection. Although analyzed separately, the project results in a single system supporting those previously mentioned features. This means that technologies used to implement each of them should be compatible. Related work addressed is based on projects implemented with off-the-shelf equipment, thus excluding any systems deployed with custom-made devices.

Indoor positioning systems are used to locate people and objects within buildings and closed environments. Obtaining indoor position information is not easy, as several issues exist, such as presence of moving people, obstacles that cause high wireless signal attenuation. Consequently, many positioning systems were designed using alternative technologies over the years, and several survey articles were written that overview and provide better understanding about existing wireless indoor location estimation mechanisms [3–6].

Activity recognition plays an important role in context-aware ubiquitous computing and can be applied in many fields such as eldercare, healthcare, tracking applications. By knowing user's activity, a personalized and intelligent service can be provided. When applying inertial sensing, using movement-based sensors, to infer user's activity, accelerometers proved to be more suitable to infer human activity by identifying and classifying movements performed by the subjects [7] [8]. Our main focus are accelerometer-based solutions.

2.1 Proximity detection

Proximity detection is considered the simplest method to estimate mobile device location. This technique determines the position of the mobile device based on existing wireless transmitters at known locations. The idea behind this method is that radio transmitters have a limited range and if a receiver, smartphone in our case, receives signal from a transmitter, it is within the coverage range of that transmitter.

2.1.1 NFC. Reader localization [9] can be a implemented using a NFC enabled mobile device as reader and a covered indoor environment with location information *Radio-Frequency Identification (RFID)* passive tags, but it does not satisfy the requirement of not requiring any user interaction to detect its indoor location. Since NFC is short-ranged, it will require the user to manually read the RFID passive tag. Another drawback is the fact that not all mobile devices have NFC technology, that would limit the positioning system to a set of NFC enabled smartphone models only.

2.1.2 WiFi and Bluetooth. It is also possible to apply this technique using WiFi or Bluetooth wireless transmitters covered within the indoor space. A user scans the available wireless devices, using a receiver, and *Received Signal Strength Indicator (RSSI)* are collected. Users's position is set to the position of the transmitter with the strongest emitting signal, holding the highest RSSI value. Assuming that the wireless signal power decreases as the distance between

transmitter and the receiver increases, the transmitter with the strongest signal is the nearest to the user.

2.2 Triangulation

Triangulation can be defined as the process of determining the location of a point by forming triangles to it from known reference points.

Triangulation using Angle-of-Arrival (AoA) determines the position of a mobile device determining the angle of incidence at which signals arrive at the receiver. The disadvantages of AoA approach is that it requires specialized hardware to measure the angle of arrival at the receiver, by having directional antennas or antenna array. Schüssel [10] evaluated a system of tracking the location of a user with a smartphone using AoA calculation at WiFi *Access Point (AP)*. Quuppa [11] is a commercially available locating system that also uses signal AoA. Quuppa uses advanced antennas capable of measuring the radio signal transmitted by a tag and sends this information to a server, referenced as *Quuppa Positioning Engine* where the tag's location is estimated. However, both solutions are not possible to apply because smartphones only have omni-directional antennas, making impossible to measure the angle of arrival of the signals.

2.3 Fingerprinting

Fingerprinting positioning technique (also known as *pattern matching*) consists in two phases: *offline* and *online*.

During the *offline* phase a *radio map* is created, which stores RSSI from all nearby transmitters at specific *reference points* (or *anchor points*). The *Radio map* consists in a database holding the position of each *reference point* with an unique identifier of each nearby transmitter and their corresponding RSSI values.

In the *online* phase, the mobile device collects the RSSI of nearby transmitters, which represents the current fingerprint. This fingerprint is compared with the stored ones in the *radio map* database using a matching algorithm, resulting in a likeliest position of the mobile device in that indoor space.

Lashkari et al. [12] proposes a mobile application able to estimate the position of a user within a building using WiFI technology, taking advantage of already existing *Wireless Local Area Network (WLAN)* infrastructure. Li et al. [13] evaluates the effect of the granularity of reference points in system's accuracy. Instead of constructing the *radio map* database manually for large number of reference points, this database can be generated using interpolation based on a small number of reference points, simplifying the training phase.

Fingerprinting is the most widely used and with good accuracy technique in indoor positioning. It does not require the exact position of wireless signal transmitters to be known, like it does in *trilateration*. Also, an already existing wireless infrastructure can be used, which reduces the overall cost of the system since no additional hardware is needed. A drawback in this method is periodical calibration required, because indoor environments may change over time due to furniture and moving people, causing changes in propagation environment. Those changes may cause the RSSI, during positioning process, be significantly different from those modeled by the *radio map*, so the *radio map* needs to be periodically updated and even using interpolation, it is still time consuming. Additionally, it has a high computational cost because of the matching algorithm that is executed in the *online phase*.

2.4 Activity Recognition Using Supervised Learning

Most of accelerometer-based activity recognition systems use machine learning approaches. According to Incel et al. [8], the process of activity recognition can be summarized as determining a target set of activities, collecting sensor data and labeling the collected data to the appropriate activities.

Guiry et al. [14] describe a method of detecting human activity using accelerometer data collected from both: mobile device and a wireless sensor band placed on user's chest. It is possible using this approach to detect the cycling activity, but it has to sense accelerometer data in a continuous way and perform complex computations, that would quickly drain smartphone's battery. Since one of our requirements is low power consumption, it would be preferable to have a system which minimizes the computation performed by mobile devices. Additionally, it requires users to wear a wireless sensor band.

2.5 Bicycle-mounted sensor

A more efficient alternative approach of detecting the cycling activity is to detect bicycle pedaling. In cycling, pedaling rate is called *cadence* and represents the number of RPM. Bicycle's cadence is related to wheel speed and consequently cycling speed, but typically all bicycles have many gears making impossible to infer bicycle speed knowing its cadence. The cadence also characterizes cycling style and is used by cyclists to minimize muscular fatigue, maintaining RPM at specific ranges. A system measuring cadence with a bicycle-mounted wireless sensor is discussed in this subsection.

Okugawa et al. [15] implement a system that generates feedback sound accordingly to the pedaling rate. The system uses a wireless sensor containing a 3-axis accelerometer and 3-axis gyroscope attached to the right pedal crank of the bicycle. The sensor sends accelerometer and gyroscope data via Bluetooth to a portable computer, that calculates the crank angle and the pedaling speed. This approach could be applied in our solution, since we can determine the pedaling speed, however the commercially available BLE devices with built-in accelerometer sensors (e.g., *motion beacons* from Estimote[16]), are not prepared to be mounted on a bicycle.

2.6 Summary

As we can see, there are many approaches for both indoor positioning and cycling detection. These can be applied in many use-cases, however, for the particular case of our application and the initially set system requirements, the presented techniques are not suitable.

3 SOLUTION

There are two entities within our system: cyclists and shops. Our solution contains a set of shops that give rewards to users that cycle to their shop. The communication between these two entities is achieved through a *web application*, a *mobile application*, and a *backend*.

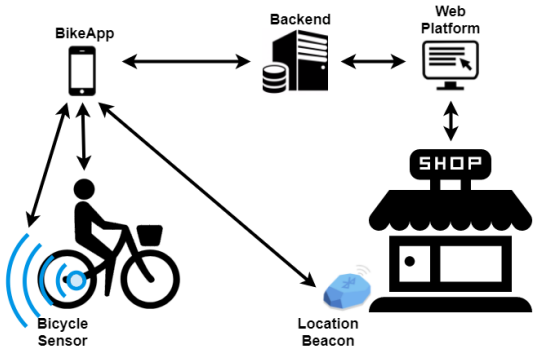

Figure 1: Global overview of BikeApp

The *web application* allows shop owners to register their shops and become part of the *rewarding program*.

BikeApp is a mobile application intended to be used by cyclists, provides information about nearby spots that joined the *rewarding program* and the benefits they offer. One type of benefit is named *cycle-to-spot*, and is given to the users when they cycle to the shop.

When *BikeApp* detects the cycling activity, the user becomes eligible to *cycle-to-spot* rewards. After this, if the user becomes close to or enters one of the shops, *BikeApp* notifies him that there is an available *benefit* to *claim*. Upon opening the application a benefit claim screen pops-up. This screen contains information about the detected shop and the reward description. To claim the benefit, all the user needs to do is press a button and show the claimed screen to the spot cashier, who will give the user the benefit.

The *backend* plays an intermediary role, since it handles requests from both *web application* and *mobile application* sources. It stores all the application relevant data, such as its users, registered spots information, etc. *BikeApp* oportunistically synchronizes with the *backend*, when there is an active Internet connection.

The cycling and location detection are two crucial features, since they are required for making the user eligible for rewards, and preparing the benefit claim upon entering the spot. *BikeApp* detects the cycling activity and the user's location with an introduction of two types of BLE devices: a bicycle sensor and a location beacon. Entities that interact within our system are represented in Figure 1.

3.1 Biklio

As mentioned in Section 1.1, *BikeApp* is a production application with the previously described rewarding capabilities.

The cycling detection of *Biklio* uses an algorithm that relies on the average and exceeded above a given threshold speeds during a trip, to infer whether or not the user is cycling. However, this mechanism may introduce false positives, if certain conditions are met, even if the user is not cycling, making it not accurate.

The location detection of *Biklio* uses the GPS adapter to fetch the user's position and considers that the user is within a spot area if the distance to the spot is less than a certain value. Because of the GPS accuracy known problems, this mechanism is also not accurate and does not work in indoor scenarios.

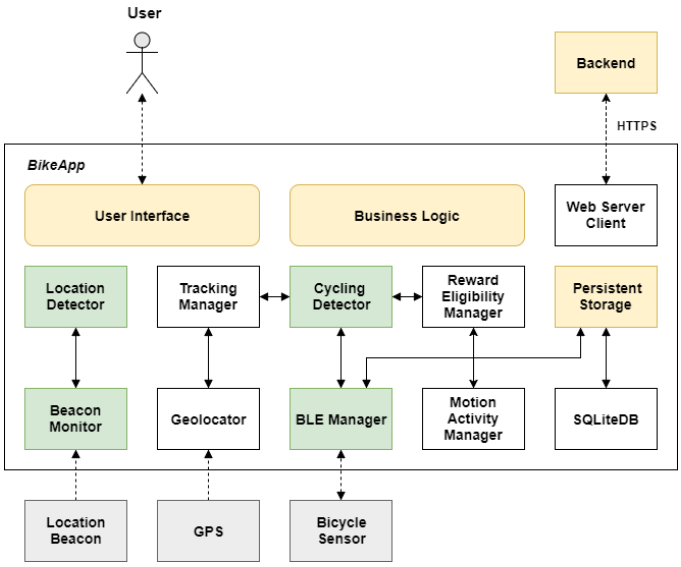

Figure 2: BikeApp modules

3.2 Architecture of BikeApp

BikeApp solves the previously mentioned problems in *Biklio*'s cycling and location detection by introducing alternative accurate cycling and location detection mechanisms. Figure 2 shows the modules of *BikeApp*. Since *Biklio* served as its base project, some of its modules were already implemented, and these are represented as uncolored boxes. The yellow boxes represent the modules that required some changes, and the green boxes represent the new or replaced modules.

The *Cycling Detector* module was completely replaced, and detects the user's cycling state using bicycle mounted sensor data. The communication with the sensor is handled by the newly added *BLE Manager* module. Similarly, the *Location Detector* module was also completely replaced. It determines the user's location considering BLE location beacons in range. The beacons are monitored by the newly added *Beacon Monitor* module. These modules are our main focus, therefore the following sections explain how the detection mechanisms work and the modules are involved.

3.3 Cycling detection

As previously mentioned, *BikeApp*'s cycling detection mechanism uses a BLE bicycle sensor. The sensor is attached to the bicycle frame and contains two components: one magnet attached to the pedal and another attached to the rear wheel spoke.

To enable the cycling detection mechanism it is required that the user pairs the bicycle sensor. The sensor pairing is done at the Bluetooth grouping of *BikeApp*'s Settings, and is only required to be done once. *BikeApp* maintains a state of paired devices and the sensor is automatically connected as soon as it becomes available.

The connection and the communication with the BLE sensors are managed by the *BLE Manager* module. After the connection is established, the *BLE Manager* starts receiving sensor data and

routes it to the *Cycling Detector* module. The latter interprets the sensor data and determines the cycling state.

The sensor data contain four cumulative values: the total counts of crank and wheel revolutions, and the times the last crank and wheel revolutions occurred. Let's designate them *Crank Revolutions*, *Wheel Revolutions*, *Last Crank Event Time*, and *Last Wheel Event Time*. All the counters start at 0 when the sensor is activated for the first time after purchasing, and are always incremented. When the user cycles, each pedal and wheel revolution increment the *Crank Revolutions* and *Wheel Revolutions* counters by 1, respectively. The *Last Crank Event Time* and *Last Wheel Event Time* are incremented with the elapsed time, in units of 1/1024 of a second, everytime there is a crank and wheel revolution, respectively. For example, if a revolution takes 2 seconds, the value is incremented by 2048.

Upon receiving the previous counters, *Cycling Detector* module determines instantaneous wheel and crank revolution speeds. This is done using simple calculations involving the elapsed time between two sequential revolutions and the number of revolutions that occurred during that time. Then, it uses the determined instantaneous speeds to determine the cycling activity, considering that the cycling activity is initiated when the instantaneous speed and cadence values exceed 30 RPM and 11 km/h during a period of 5 seconds. When *Cycling Detector* does not receive updated sensor data for a period of 30 seconds, it considers that the cycling activity was terminated.

3.4 Location detection

BLE location beacons are devices that broadcasts its identifier to nearby devices at regular periods. Our solution requires a location beacon to be placed within each shop. The beacons are installed by the shop owners, and configured with the parameters and generated identifiers given by the *web application* during the shop registration. Each location beacon is configured to the *iBeacon* [17] protocol, with unique to each shop *Major* and *Minor* identifiers, and a global UUID.

The *Beacon Monitor* module is responsible for monitoring BLE beacons, and can detect whenever a mobile device appears in the signal range of a beacon. When this happens, the *Beacon Monitor* module triggers an event and provides information about the beacons in range to the subscribed entities. This information contains beacon's UUID, *Major* and *Minor* identifiers, and also an estimated distance to the beacon.

The *Location Detector* module encapsulates all the location detection logic, which is simply based on a correspondence table between each beacon and the shop it is placed in. By knowing the identifiers of the closest in range beacon, it determines the visited shop through a table lookup.

To determine the closest in range beacon, the *Location Detector* module uses the *Beacon Monitor*. It requests *Beacon Monitor* to monitor all the shop beacons, and subscribes to its event to be notified and receive beacon information, whenever a shop beacon appears in range. If multiple beacons are in range, the *Location Detector* module uses the estimated distance at which each beacon is located to determine the closest beacon. Finally, *Location Detector* just consults its beacon-shop correspondence table and determines the visited shop.

4 IMPLEMENTATION

This section presents the implementation details and important decisions done during the implementation process of the two core features we mainly focused: cycling and location detection mechanisms.

We start by introducing the development environment. Then, we discuss the implementation aspects of each of the previously introduced modules, namely *BLE Manager*, *Cycling Detector*, *Beacon Monitor* and *Location Detection*.

4.1 Development environment

BikeApp is developed in a cross-platform fashion using Xamarin [18], follows the *Portable Class Library (PCL)* project type, and is entirely developed using C# programming language. PCL contains two types of project: *core project* and *platform-specific project*. The *core project* is a shared code-base containing the greater part of the implementation. The *platform-specific project* only contains the native part of the implementation, which requires the use of *Software Development Kit (SDK)*.

During the implementation, we only developed the application targeting the iOS platform. However, because the implementation was done as a cross-platform project, we abstracted our solution in a way that is possible and requires little implementation effort to expand it targeting also the Android platform.

4.2 Module communication

To achieve high cohesion and low coupling, each of our modules have a well-defined responsibility and a set of exposed events, providing a way of interaction to the other modules in a *event-driven* fashion.

Since Xamarin projects are implemented in C#, we use the event handling feature available within the .NET framework. This feature allows an entity to *listen* (or *subscribe*) to a specific *event*, and be notified by the *sender* (or *publisher*) when the event is raised, executing its own custom code. The sender can also send custom data to its subscribers when the event triggers.

4.3 BLE Manager module

The *BLE Manager* module is responsible for discovery and communication with BLE devices. It provides a set of operations to request an action, more specifically to start/stop scanning for bicycle sensors and connect/disconnect a given sensor.

Additionally, it provides a set of events that other modules can subscribe to, to receive important data. One such event is *SensorDataReceived*, which provides updated sensor received data to the subscribing entities.

To increase code sharing between the platforms and avoid code duplication, part of the module is implemented within the *core project* and the other part within the *platform specific project*. The implementation from the platform side is required because native SDK must be used to implement the scanning and the communication with the sensor.

4.3.1 Scanning. The scanning for the sensors is implemented using the *Core Bluetooth* [19] framework, more specifically using

ScanForPeripherals and *StopScan* operations contained within its *CBCentralManager* class.

An important particularity of background scanning in iOS, is that it is only possible if we declare the *Uses Bluetooth LE accessories background execution mode* [20], and keep the scanning continuously running when the application is in the background. This sounds highly inefficient in terms of battery consumption, but iOS automatically manages its Bluetooth scanning, adjusting its *scan window* and *scan interval* parameters when the application goes into the background, to avoid the smartphone's battery draining. We analyze the overhead introduced with the background continuous scanning in Section 5.5.

4.3.2 Communication with the sensor. BLE sensors share information with the smartphone through *Generic Attribute Profile (GATT)* communication. The sensor acts as a GATT server, and defines a hierarchical data structure that is exposed to connected GATT clients (smartphone). Its hierarchy is composed of a single *profile* containing a mandatory *Cycling Speed and Cadence (CSC)*[21] *service*. The CSC service contains a *CSC Measurement*[22], which contains provides crank and wheel revolutions data.

The sensor only sends its data to connected devices that subscribes to its *CSC Measurement* characteristic. Once the subscription to the characteristic is done, the sensor starts periodically sending its *value*. The communication with the sensor also uses the *Core Bluetooth* framework. *Core Bluetooth Programming Guide* [23] describes the required steps to start receiving sensor's data.

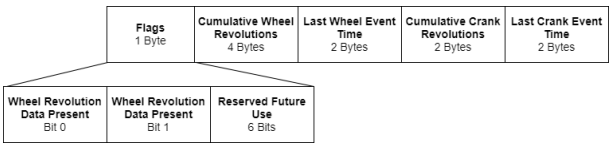

Figure 3: Raw advertised data structure

Figure 3 shows the raw data that is received from the *CSC Measurement* characteristic subscription. The first data byte contains flags field and the first two bits indicate whether the value contains wheel revolution data, crank revolution data or both. Wheel revolution data, if present, consists of a 32-bit cumulative count of wheel revolutions and a 16-bit value representing the time the last wheel event was measured, in units of 1/1024 of a second. Crank revolution data consists of a 16-bit cumulative count of revolutions of the crank, and a similar 16-bit last event time value. Once the *BLE Manager* receives the raw data, it retrieves the meaningful counters and passes them to the subscribed entities through its *SensorDataReceived* event.

4.4 Cycling Detector module

The *Cycling Detector* module is entirely implemented within the core project. This module serves as an entry point to use the cycling detection mechanism. It contains a property *IsCycling* indicating whether or not cycling is being detected. Additionally, it contains *CyclingStarted* and *CyclingStopped* events, which allow other application modules to be notified when the cycling state changes.

As already mentioned in the Section 3.3, *Cycling Detector* uses the received sensor data to determine the cycling activity. The sensor data is obtained through *SensorDataReceived* event subscription of *BLE Manager*, as shown in Listing 4.4. This event starts to be triggered after *BLE Manager* establishes a connection with a sensor. The subscribed callback invokes the *Input* operation, passing it the sensor data, namely the *Crank Revolutions*, *Wheel Revolutions*, *Last Crank Event Time* and *Last Wheel Event Time* counters. Then, the *Input* method applies the logic previously described in Section 3.3 to determine the cycling state.

```
BLEManager.Instance.SensorDataReceived += (s,e) =>{
        Input(e.Data);
}
```

Listing 1: Sensor data subscription

The *CyclingStarted* and *CyclingStopped* events are raised when the cycling state changes. These events allow easy integration of the mechanism within our application through event subscription. Listing 4.4 shows how the automatic tracking (register route using GPS) can be enabled.

```
CyclingDetector.Instance.CyclingStarted += (s,e) =>{
    TrackingManager.Instance.StartTracking();
}

CyclingDetector.Instance.CyclingStopped += (s,e) =>{
    TrackingManager.Instance.StopTracking();
}
```

Listing 2: Automatic tracking enabling

4.5 Beacon Monitor module

The *Beacon Monitor* module is responsible for monitoring BLE location beacons. It provides two operations, to start and to stop monitoring for a given beacon region. Both operations receive an object that represent the target beacons, through its UUID parameter. For example, if we invoke the start monitoring operation and pass it the required object with UUID parameter set to a given value, then all the beacons that share the same UUID are also monitored, not considering their *Major* and *Minor* identifiers. This approach allows us to monitor all the shop beacons, which share the same UUID, by requesting a single monitoring.

The *Beacon Monitor* module provides two events: *BeaconRegionEntered* and *BeaconRegionLeft*. The *BeaconRegionEntered* event is raised when a monitored beacon appears in range. Similarly, *BeaconRegionLeft* event is raised when the monitored beacons becomes out of range.

Similarly to the *BLE Manager* module, *Beacon Monitor* module was also developed within the core and platform-specific projects.

4.5.1 Beacon protocol. Even though *Core Bluetooth* framework scans for BLE wireless devices, the UUID property of the discovered beacons do not match the real beacon's UUID, advertised in the data packets [24]. Instead, it is randomly generated from device to device. This means that the same beacon has different associated UUID values when scanned in different devices. Thus, the platform-specific part of *Beacon Monitor* module is implemented with the only alternative framework capable of detecting beacons, named *Core Location*[25].

With *Core Location* it is possible to detect the proximity to monitored beacons by their real UUID, and obtain a distance estimation to the beacons in range. However, the framework works exclusively with beacons using the *iBeacon* protocol. We do not consider this a big limitation, because this protocol is widely used and many beacons from different manufacturers support it. Addittionally, Android's SDK provides full access to the raw beacon advertised data, making our solution using *iBeacon* also compatible.

4.5.2 Platform-specific implementation. The monitoring functionality provided by the *Beacon Monitor* is implemented using two features available in *Core Location* framework, namely *region monitoring* and *region ranging*.

The *Core Location*'s *region monitoring* has the same monitoring behavior as described previously. When requesting the region to be monitored by its UUID, all beacons that have the same UUID are contained within the region. However, when one of the monitored beacons appear in range, *region monitoring* only provides its UUID, and the beacon's *Major* and *Minor* identifiers are unknown, making impossible to determine which of the monitored beacons is in range. To get these identifiers we use the *beacon ranging*.

Beacon ranging continuously scans for beacons and provides results every second. The ranging result is a list of *CLBeacon* objects, which represent the available in the vicinity beacons. *CLBeacon* object contains a UUID, *Major* and *Minor* identifiers. Additionally, it also contains an *Accuracy* parameter, which represents the estimated to the beacon distance. Just considering the first ranging result is unreliable, because beacon's RSSI fluctuates over time, and the estimated distance directly depends on RSSI. Thus, we keep the ranging running for a few seconds and consider the calculated average *Accuracy* value as beacon's estimated distance. After running the ranging for a couple of seconds, we have information determining the average estimated distance to every in range beacon, we trigger the *BeaconRegionEntered* event of the *Beacon Monitor* module, provide the identifiers and the estimated distance of nearby beacons to its subscribed callbacks.

We performed a set of tests to our location detection mechanism with the ranging being run for 10 seconds and consider this time period enough to correctly identify the closest beacon in situations where two beacons are in range, which happens where there are two shops next to each other. The experiment is detailedly explained in Section 5.3.

To summarize, the monitoring capability of *Beacon Monitor* works as follows. When we request the *Beacon Monitor* to monitor a region identified by UUID = X, it uses the *beacon monitoring* feature of *Core Location* framework to monitor the requested beacon region. When any beacon with UUID = X appears in range, *beacon monitoring* of *Core Location* detects it, and runs the *beacon ranging* for a period of 10 seconds. During this period *beacon ranging* collects the beacon's identifiers and determines for each beacon in vicinity the average of its object's *Accuracy* parameter. The resulting value represents an estimated to the beacon distance. Finally, the *BeaconRegionEntered* event is triggered and the subscribes entities receive the identifiers and the estimated distance of nearby beacons.

4.6 Location Detector module

The *Location Detector* modules is responsible for detecting when the users approaches or enters any of the shops.

It contains a settable *BeaconSpotTable* property, which is a C# *Dictionary* collection containing beacon identifiers as a *key*, and the corresponding shop object from the application's business model as a *value*.

Location Detector module is entirely implemented within the core project and its implementation uses the *Beacon Monitor* module. *Location Detector* requests the *Beacon Monitor* to start monitoring for all the shop beacons with a given UUID. Additionally, the *Location Detector* subscribes its *BeaconRegionEntered* event, to be notified whenever a shop beacon appears in range. Finally, *Location Detector* determines the visited shop, using its *BeaconSpotTable*, by the steps previously described in Section 3.4.

When *Location Detector* determines the shop, it triggers its *SpotEntered* event, which notifies the subscribed entities and provides the object representing the visited shop. This object can also be consulted with the *CurrentSpot* property.

Similarly to the *Cycling Detector* module, this module also allows easy integration of the mechanism within our application through event subscription.

5 EVALUATION

The present chapter describes the evaluation methodology as well as the experiments performed to understand the viability of our solution and determine optimal conditions to achieve the system requirements listed in Section 1.1.

5.1 Advertising Intervals and Discovery Times

This section analyzes beacon discovery times when scanning beacons with different *advertising intervals*. This is required, to determine an optimal beacon *advertising interval* value, as a trade-off between beacon discovery time and its battery life.

In this experiment we set our smartphone to scan for BLE beacons. The scanning was performed in the background and with a duration of 1 hour for each beacon *advertising interval*. We choose this duration to understand how the discovery time varies over time with different *advertising interval* configurations, because the application can remain in the background for long periods of time and we want to minimize the time to discover a beacon.

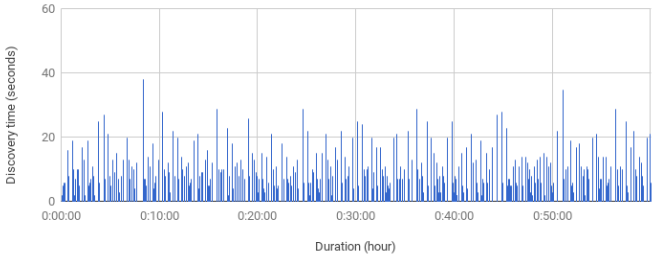

Figure 4: Beacon discovery times with 100 ms Advertising Interval

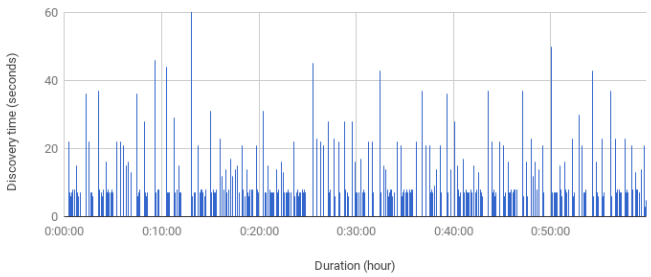

Figure 5: Beacon discovery times with 300 ms Advertising Interval

Figure 4 and Figure 5 show the beacon discovery times over 1 hour background scanning for 100 ms and 300 ms *advertising interval*, respectively. The y-axis contains the time in seconds that each beacon discovery took. The x-axis contains the total scanning duration.

In both interval configurations the majority of discoveries are within the 0-10 seconds range. But we can not observe any pattern or rule by which the discovery time varies. The discoveries with a higher *advertising interval* may sometimes take a little bit longer, as expected. However, both contain unpredictably long discoveries, even during the first minutes of scanning. This is result of Apple's way of adjusting the *scan window* and *scan interval* parameters, not controllable by the developers.

Regarding the question which *advertising interval* should be used in beacons, as we can see on the presented charts, there is no substantial advantage of using the recommended by the Apple 100 ms value over the 300 ms interval. This means that the beacons can be configured to 200 ms or 300 ms and benefit from prolonged battery life.

5.2 Estimated vs real distance

Transmission power is the power with which the beacon broadcasts its signal. Logically, beacon's range directly depends on the *transmission power*, but greater *transmission power* values decrease the beacon's battery lifespan.

We collected the beacon's RSSI and estimated distance at different distances, to understand how the signal strength and the given estimated distance correlate with the real distance.

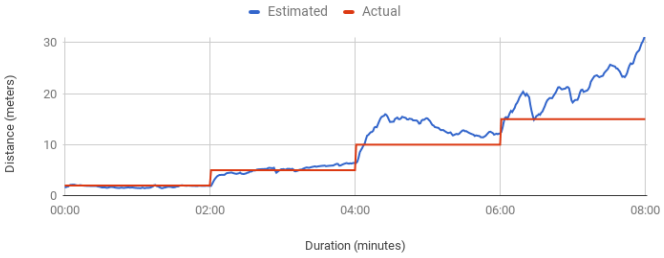

Figure 6: Real and estimated distances at 0 dBm *transmission power*

Figures 6 illustrate the correlation between the estimated and the real distances at which a beacon is located. Y-axis contains the measured distance (in meters), while the x-axis contains the duration of the experiment (in minutes). The red line represents the real distance, at which the beacon's signal was captured, and the blue line represents the *CLBeacon*'s *Accuracy* parameter value. Indeed, we can see that the *Accuracy* parameter does represent a beacon's distance estimation, and the values have a good correlation until transitioning from the 5 to 10 meters distance, which occurred at the 04:00 time. It is important to notice that there is always a considerable difference in the estimated distance value when being closer (0-5m) or further away (10-15m) to the beacon. The same experiment with the *transmission power set to -12 dBm* shows similar result.

5.3 Location Detection

To evaluate the location detection mechanism we decided to collect estimated distance values of two beacons placed in a worst-case scenario, within shops adjacent to each other, and see how the estimated distance varies when entering one of the shops. Each trial consisted in approaching both shops (standing at the same distance from both) and then enter one of the shops, while collecting the estimated distance values of the beacons placed within each shop.

Analyzing the estimated distances collected from the beacons located within each shop, we observe two situations. The first situation happens when the *region entered* event is triggered when being few meters away from both shops, where signal from both beacons is received. At this time our location mechanism starts the *beacon ranging* and collects the estimated distance values during a period of 10 seconds. Figure 7 represents one of this trials and we can see that the estimated distance from both beacons is approximately the same when the *beacon ranging* starts, but as long as we walk towards shop B, its beacon's estimated distance considerably decreases, while the estimated distance of shop A slightly increases. Thus, the calculated average of the estimated distance from both beacons indicates the shop we enter.

Figure 7: Location detection starts few meters away of both shops before entering shop B

The first situation happens because the chosen *transmission power* of -12 dBm has a theoretical range of 15 m and since the shops were of small dimensions, the beacon's signal was received even when being outside and few meters away from both shops. It shows the importance of selecting an appropriate *transmission*

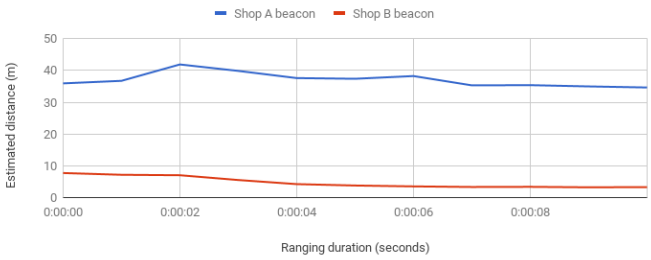

Figure 8: Location detection starts upon entering shop B

power value and the beacon's placement within the shop on a way that the shop indoor area is within the beacon's range and the beacon signal coverage outside the shop is minimized. For example, a slightly lower *transmission power* would be a better fit for shops A and B, because the beacon's signal will be mainly detected when entering the destination shop, thus decreasing the first presented situation, were the location detection starts outside of the both shops.

The second situation happens when the *region entered* event is only triggered upon entering the destination shop. We can see from the estimated distance values shown in Figure 8, that during the whole period of 10 seconds *beacon ranging* always reported estimated distances with values less than approximately 8 meters for the beacon of the shop we entered, and values greater than approximately 35 meters for the beacon of the neighboring shop. Logically, the average of the estimated distance values have a considerable difference, which makes the location detection mechanism correctly determine the entered shop.

It is also important to note that the signal strength is highly attenuated because of the wall that separates the shops, as we can see from the presented estimated distance variation charts.

Our mechanism always correctly determined the entered shop, in all initial 10 trials. Thus, we consider that the *beacon ranging* period of 10 seconds is a good candidate for our solution.

5.4 Cycling Detection

When *BikeApp* is in the foreground, the sensor is always instantly connected as soon as it becomes available, i.e., with a crank or wheel revolution, and starts sending its data.

We decided to test also the connection time in an alternative scenario, in which the application is scanning in the background and the sensor becomes activated.

Thus, to evaluate the cycling detection mechanism we take into consideration the time it takes to connect to a deactivated sensor when starting cycling, the time the mechanism takes to detect the cycling activity and the accuracy of detection.

During the first 10 trips we observed an average and maximum connection times of 9 and 17 seconds, respectively. The connection times showed to be very similar to the beacon discovery times, where the majority of discoveries are within the first 10 seconds, but sometimes this time may be greater (see Section 5.1).

Table 1: Battery consumption when smartphone is idle

Running	Battery consumption per hour
No application	0.40%
BikeApp	0.74%
Biklio	2.75%

Table 2: Battery consumption when cycling

Running	2% battery drop	Battery consumption per hour
BikeApp	≈45 minutes	≈2.6%
Biklio	≈37 minutes	≈3.2%

This cycling detection time represents the elapsed time until the instantaneous cadence and speed values exceed determined thresholds during a period of 5 seconds. After the established connection with the sensor, our mechanism took on average 8 seconds to detect the cycling activity over the 10 trips.

Regarding the accuracy, our mechanism proved to be 100% accurate on detecting the cycling activity. The sensor was always connected and the instantaneous cadence and speed values reached the required thresholds, resulting in cycling detections.

5.5 Smartphone Battery Consumption

One of our system requirements is the low battery consumption. Because our mobile application uses the Bluetooth adapter, it is important to analyze the power consumption overhead introduced with the required background continuous scanning, and the communication with the sensor.

First, we determine the power consumption of the smartphone without running any application (standby), to have a reference point. Then, we determine the power consumption of *BikeApp* when the smartphone is idle and when cycling. Similarly, we determine the power consumption of *Biklio* (which uses GPS instead of BLE), when the smartphone is idle and when cycling. in both cycling and location detection mechanisms

We left the smartphone overnight registering the battery level changes into the logfile for the following scenarios: not running any application, running *BikeApp* when smartphone is idle, and running *Biklio* when the smartphone is idle. When cycling, we determined the amount of time required to drop the battery level by 2% on both applications, and estimated a consumption per hour.

In all three scenarios, we consider the battery consumption over a period of approximately 12-13 hours, and obtain the following consumptions represented in Figure 1. The results show that when the smartphone is idle, *BikeApp* has considerably less power consumption than *Biklio*, with a difference of ≈ 2% consumption per hour. Additionally, the continuous background BLE scanning does not introduce a big power consumption overhead, the consumption just increases by 0.34%.

Regarding the battery consumption when cycling, as we can see in Table 2, our solutions has a slightly lower battery consumption than *Biklio*'s. The communication with the sensor introduces an overhead of ≈2.2% per hour. However, since the majority of cycling

trips in urban environments do not last hours, a power consumption from an occasional trip of 10-15 minutes is unnoticeable from the user perspective.

5.6 Usability Evaluation

To finish the *BikeApp* evaluation, we evaluate the usability of our solution with end-users, by asking them to interact with the application and answer a survey. The questionnaire answers from the user testing sessions show that the users did not experience any problem in finding relevant information or pair a sensor within the application settings. By following the bicycle sensor's manufacturer installation instructions, the majority of users rate the installation as easy. The reward claims are also considered easy, and the majority of users find the automatic shop detection as very useful. However, without knowing the particularities of BLE in terms of power consumption, many users consider the fact that the application requires Bluetooth connection as negative, thinking that it drains the smartphone's battery.

6 CONCLUSION

This project introduces new cycling and location detection mechanisms using BLE devices. We integrated those mechanisms in *Biklio*, replacing its less accurate cycling and location detection mechanisms. The new version of the application (*BikeApp*) requires users to pair their bicycle's sensor in the application Settings, and the shop owners to install a location beacon within their shops. *BikeApp* detects when the user starts cycling and makes him eligible for rewards that can be claimed at the shops. When the user gets close to or enters a shop, *BikeApp* detects the shop and shows a screen to allow the user claim the benefit.

After the implementation, we evaluated the system. The implemented detection mechanisms show to be accurate and introduce a small overhead of 0.34% when the smartphone is idle, and an acceptable overhead of 2.2% when the smartphone keeps the communication with the sensor while cycling. In both scenarios *BikeApp* has less power consumption than *Biklio*, especially whe the smartphone is idle, with a difference of $\approx$ 2% consumption per hour. The usability of *BikeApp* was also evaluated with end-users and a questionnaire. The results allow us to conclude that users do not consider difficult the bicycle sensor installation and pairing within the application. Users consider useful the location detection mechanism and consider very easy the process of claiming the benefit.

6.1 Future Work

To achieve a public release, there are some aspects that need to be checked and implemented. We suggest to add an introductory tutorial to appear upon opening *BikeApp* for the first time, to clarify the use of bicycle-sensor and location beacons and briefly mention the advantages of BLE in power consumption. Since the application was implemented in cross-platform, we also suggest as future work to expand its implementation to the Android platform. Because the tested version of iOS (10.3.2) has a bug in its *Core Location* framework [26]. It is important to check that in the newer versions of iOS, the bug is fixed, to guarantee correct application functioning. The *web application* is already implemented for the *Biklio* project. However, it is required to update its shop registration page, to include a section about location beacons, that should explain how the location beacons work and provide configuration instructions.

REFERENCES

[1] Biklio. Accessed: 28-12-2016. [Online]. Available: https://www.biklio.com
[2] Trace - Walking and Cycling Tracking Services. Accessed: 28-12-2016. [Online]. Available: http://h2020-trace.eu
[3] H. Koyuncu and S. H. Yang, "A Survey of Indoor Positioning and Object Locating Systems," *International Journal of Computer Science and Network Security (IJCSNS '10)*, vol. 10, no. 5, pp. 121–128, 2010.
[4] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," pp. 1067–1080, 2007.
[5] Z. Farid, R. Nordin, and M. Ismail, "Recent advances in wireless indoor localization techniques and system," *Journal of Computer Networks and Communications*, vol. 2013, 2013.
[6] K. Al Nuaimi and H. Kamel, "A survey of indoor positioning systems and algorithms," *2011 International Conference on Innovations in Information Technology, IIT 2011*, no. May 2011, pp. 185–190, 2011.
[7] B. G. Mathie, M. J., Coster, A. C., Lovell, N. H., & Celler, "Accelerometry : providing an integrated , practical method for long-term , ambulatory monitoring of human movement," *Physiological measurement*, vol. 25, no. 2, p. R1, 2004.
[8] O. D. Incel, M. Kose, and C. Ersoy, "A Review and Taxonomy of Activity Recognition on Mobile Phones," *BioNanoScience*, vol. 3, no. 2, pp. 145–171, 2013.
[9] L. Mainetti, L. Patrono, and I. Sergi, "A survey on indoor positioning systems," *2014 22nd International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pp. 111–120, 2014.
[10] M. Schüssel, "Angle of Arrival Estimation using WiFi and Smartphones," no. October, pp. 4–7, 2016, accessed: 26-11-2016. [Online]. Available: http://www3.uah.es/ipin2016/usb/app/descargas/223_WIP.pdf
[11] Quuppa Intelligent Locating System. Accessed: 26-11-2016. [Online]. Available: http://quuppa.com/
[12] A. H. Lashkari, B. Parhizkar, and M. N. A. Ngan, "WIFI-based indoor positioning system," *2nd International Conference on Computer and Network Technology, ICCNT 2010*, no. November 2016, pp. 76–78, 2010.
[13] B. Li, J. Salter, A. Dempster, and C. Rizos, "Indoor positioning techniques based on wireless LAN," *Conference on Wireless*, pp. 13–16, 2006.
[14] J. J. Guiry, P. van de Ven, J. Nelson, L. Warmerdam, and H. Riper, "Activity recognition with smartphone support," *Medical Engineering and Physics*, vol. 36, no. 6, pp. 670–675, 2014.
[15] R. Okugawa, K. Murao, T. Terada, and M. Tsukamoto, "Training system of bicycle pedaling using auditory feedback," *Proceedings of the 12th International Conference on Advances in Computer Entertainment Technology - ACE '15*, pp. 1–4, 2015.
[16] Estimote. Accessed: 26-7-2017. [Online]. Available: https://www.estimote.com
[17] Apple Developer - iBeacon. Accessed: 2-8-2017. [Online]. Available: https://developer.apple.com/ibeacon
[18] Xamarin. Accessed: 26-7-2017. [Online]. Available: https://www.xamarin.com
[19] Apple Developer Documentation - Core Bluetooth. Accessed: 28-7-2017. [Online]. Available: https://developer.apple.com/documentation/corebluetooth
[20] iOS Programming Guide - Background Execution. Accessed: 31-7-2017. [Online]. Available: https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html
[21] GATT - Cycling Speed and Cadence Service. Accessed: 28-7-2017. [Online]. Available: https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.cycling_speed_and_cadence.xml
[22] GATT - CSC Measurement Characteristic. Accessed: 28-7-2017. [Online]. Available: https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.csc_measurement.xml
[23] Core Bluetooth Programming Guide. Accessed: 28-7-2017. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts
[24] Radius Networks Developer Blog - CoreBluetooth Doesn't Let You See Beacons. Accessed: 29-7-2017. [Online]. Available: http://developer.radiusnetworks.com/2013/10/21/corebluetooth-doesnt-let-you-see-ibeacons.html
[25] Apple Developer Documentation - Core Location. Accessed: 28-7-2017. [Online]. Available: https://developer.apple.com/documentation/corelocation
[26] iOS Core Location Bug Report. Accessed: 16-9-2017. [Online]. Available: http://www.openradar.me/31886860