Technical University of Denmark

Master Thesis

Proximity Door Locking

Author:
Simon Jensen (s146896)

Supervisor:
Christian D. Jensen

September 16, 2016

Abstract

Smart door locks that can unlock doors from a smartphone have recently been introduced to the market. Their main weakness is that they still require the user to press a button on their phone to unlock the door, what if this could be done automatically. This project first analyzes current and upcoming sensors in smartphones that may be able to help unlocking automatically and positioning techniques that do not rely on GPS. Then the Android operating system is analyzed for its capability to provide information from Bluetooth, Wifi, GPS, network location, and inertial sensors. A prototype framework application that can make a choice based on data collected from these sensors is then designed and implemented. Finally the accuracy of sensors and application is tested and evaluated.

Contents

Chapter 1

Introduction

Consumer devices are currently getting smarter and smarter, there are constantly released smart versions of previously “dumb” appliances and other hardware. These smart versions can often communicate with a smartphone in an attempt to provide more information, easier access, or removing the need to carry things that everybody carries around every day. One of the things that have been getting smarter over the past couple of years are door locks. Smart door locks provide the possibility of unlocking doors without having to carry around keys or cards. Smart locks work wirelessly through Bluetooth communication, and can be unlocked by smartphones that are paired with the lock whenever the two are within range of each other. Depending on the smart lock, it is also possible to share the keys wirelessly with friends and family in case they need access to the building, thus eliminating the need for having met up and shared the keys previously.

Currently, the smart locks on the market requires the user to actively unlock the doors themselves by unlocking their phone and pressing a button in the corresponding application. This allows the user to replace keys with their smartphone that they are carrying anyway. However, the process of unlocking the door has not been made any simpler, more likely the process of unlocking the smartphone, launching the application, and pressing the unlock button requires more steps than unlocking the door with a physical key.

Ideally, the process of unlocking should be made easier. As the product is already using smartphones, it makes sense to try to make use of the processing power and vast amount of other hardware available for applications running on said smartphone. This hardware includes sensors that can be used to get information about the surrounding area around the phone, along with the movement of the phone itself.

This report will consist of an analysis of common sensors available for smartphones and the possibility of using each for making a decision to either unlock an encountered door, or keep it locked. Furthermore, the project will analyze the most common phone platforms, and choose one that is best suited for the task, then the platform will be analyzed in order to find existing functionality that can support the project. After that a number of requirements will be identified and a design for a prototype will be created. The prototype will be implemented, and this process will be documented. Finally, the prototype will be tested in a real world environment.

The prototype is intended to be a framework that collects needed data and provides the foundation for future development. The prototype is intended to be further developed in an upcoming master thesis.

1.1 Additional Contributors

The project has been carried out in partnership with BeKey A/S. BeKey have provided the initial idea with a request to develop a prototype that can smartly unlock their locks without unlocking when it is not the intention of the user. The door locks that are used for this project has been provided by BeKey A/S.

Besides BeKey, work has been done in partnership with bachelor student Rasmus Lundsgaard Christiansen. Rasmus did his bachelor project in the unlocking decision making with use of machine learning, a project which relies on the framework created for this project. While this report and the prototype has been written entirely by me, the design and prototype contains features that has been requested by Rasmus in order to facilitate his solution.

1.2 Description of a Door

For the purpose of this project, a door is not just a door in a house or apartment. Instead a door always has an associated BeKey lock, which is Bluetooth enabled and can be detected by a smartphone or other Bluetooth enabled hardware.

The reason for this definition is that the device used to unlock a door must be able to communicate with it. In order to automatically take a decision about unlocking a given door, the lock must be continuously announcing that it exists, such that the smartphone can discover that a known lock is nearby and a decision should be made.

1.3 Why is this Interesting?

In order to see why the problem of automatically unlocking doors is interesting to solve we first look at a naive solution and the problems associated with it.

The naive solution would be to send the unlock signal whenever the smartphone is nearby a lock that it is paired with. However, this for many residential areas where people move around in their home, the smartphone and lock are in range of each other in large parts of the house. This would make the phone send unlock request multiple times when the user is at home.

An example of the range of the smart lock Bluetooth signal can be seen on Figure 1.1. As can be seen, the signal from the lock can be seen from multiple rooms in the house. The naive solution would unlock the front door whenever the user moves around in the green and yellow rooms, or in the hallway around the door. This is likely to unlock the door many times where the user does not want it to unlock, in this case the smart lock would provide more frustration than the problem it tries to solve.

In order to solve this, the application should be smarter than the naive solution, and not unlock the door if the user has not left their house. The application should also not

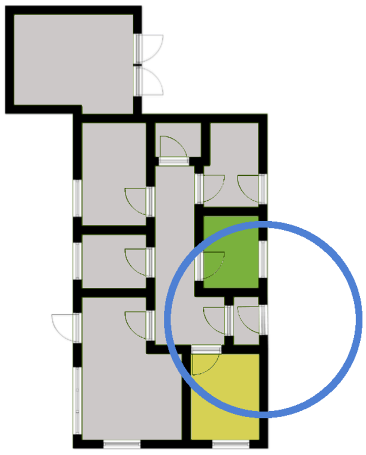


Figure 1.1: An example house with a smart lock on the front door.

unlock the door if it is behind another door that needs to be unlocked first, or if the user is just passing by and does not want to go inside.

1.4 Idea

The idea for the overall project, is that the smartphone should be able to take a decision to unlock a door based on sensor data that it has collected in the time leading up to the phone encountering the door. The decision is taken based on a number of factors. One of the factors is data collected from the accelerometer that will show the path that the smartphone has moved in the time before the door has been encountered.

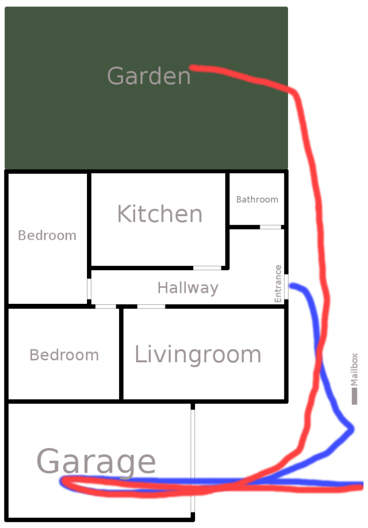


Figure 1.2: An example of two paths taken where the front door should unlock with the blue path and not with the red path.

On Figure 1.2 two different paths from arriving in the garage to moving around the door are shown. The blue line shows the phone moving from the garage to the mailbox, then to the front door, here the door should automatically unlock. The red line shows another path taken, here the phone moves past the door into the garden, in this case the door should stay locked. The same applies for movement inside the home, here the door should stay locked until just before the user is intending to go outside.

1.5 Description of the Solution

The solution for this project will make use of the Android platform, the frameworks it provides, and the available sensors that can be accessed on most high-end smartphones.

For the solution the following sensors will be used to collect data that can be used to make a decision for unlocking a door, as well as general information about when to collect data and when we expect to see a door.

- Inertial sensors (accelerometer, magnetic field sensor, and gyroscope)
- Bluetooth and Wifi
- GPS and network based location

The accelerometer, gyroscope, and magnetic field sensor will be used to measure movement direction and velocity for the person carrying the device, additionally they will be used to measure the phones orientation in relation to magnetic north and when the phone is not moving. Bluetooth and Wifi will be used to find networks nearby, that can indicate where in the world the device is located, and when it is close to a known location. Bluetooth will additionally be used to recognize known doors and unlock them if a decision is made to do so. Network based location will be used to create geofences and will start and stop the collection of data when they are entered or exited. The network based location will assist the GPS because it is much quicker to provide a first location and can be more precise inside buildings, where the GPS might not be able to get a location. When the smartphone is within range of the door, the application will use heuristics to make a decision to unlock or not.

1.6 Lessons Learned

The main takeaway from this project is that the sensors in smartphones are not as accurate as first assumed. The biggest issue is the noise and inaccuracies of the inertial sensors of the accelerometer, gyroscope, and magnetic field sensors. The intention was to use these to record a path of movement but it turns out that they are only somewhat accurate for periods of up to 3 seconds, after which the inaccuracies results in movement patterns that in no way mimic the real world. In addition, the patterns from different measurements differ wildly from each other. GPS and network location are quite stable, but cannot be used for indoor positioning as the number of location updates from the network location is too low and the GPS cannot be expected to work indoors. A decision to unlock the door can be made using the heuristics made, but a number of concessions have had to be made.

Chapter 2

State of the Art

2.1 Sensor Technology in Smartphones

Before making a decision of which sensors to use for unlocking decisions, a survey of the existing sensors have to be made. This survey will cover all the currently existing sensors in smartphones along with their availability and usefulness for the decision making. Additionally, a survey sensors that are currently being developed and are likely to be implemented in future smartphones will be made.

2.1.1 Common Sensors

The sensors that exist in the majority of smartphones are camera, microphone, accelerometer, magnetic field sensor, gyroscope, luxmeter, Global Positioning System (GPS), and a proximity sensor.

For positioning the GPS is quite good, however, it does have a number of issues that makes it unusable as the only source for positioning a smartphone in the case of automatically unlocking a door. Firstly, it requires that the smartphone is able to get a signal from the GPS satellites orbiting the earth, this is a problem when the device is located inside a building, in a metropolitan city with many tall buildings, or if there are other large obstacles in the way. Secondly, the GPS signal does not provide a precise enough position to make a decision exclusively based on the location of a smartphone. The precision of the GPS system has a precision of around 3.5 meter radius with a 95% accuracy in the best case[31, 2].

The accelerometer, gyroscope, and magnetic field sensors are closely related. The accelerometer measures acceleration force applied to the device in three axes, the accelerometer can thus be used to measure movement and direction. The magnetic field sensor measures the strength of the earth's magnetic field in the same three axes making it a three dimensional compass, and can be used to determine the rotation of the device in relation to the magnetic north. Finally the gyroscope measures the rotational force applied to the device, much like the accelerometer measures the acceleration forces. The difference between the accelerometer and gyroscope compared to the magnetic field sensor is that the accelerometer and gyroscope measure change and will thus measure zero if the device is not moving, with the exception of gravity for the accelerometer, where the magnetic field sensor will always measure the magnetic field.

The three sensors all use x, y, and z axes for a reason. The data each sensor measures can be used to manipulate the data from another sensor. For example, it is possible to use the magnetic field sensor and gyroscope data to rotate the accelerometer data depending on the orientation of the device, such as locking the accelerometer data to the world coordinate system. This is desirable because the output of the accelerometer by default is tied to the orientation of the phone. This means that depending on how the phone is carried, the accelerometer data will be different, and data captured with the phone in one orientation is not easily comparable to data captured with the phone in a different orientation. Reorienting the data to the world coordinate system resolves this issue.

The microphone and luxmeter measure sound and light intensity respectively. The microphone is used to record voice and sound for phone calls and videos recorded with the camera, many high-end phones are equipped with multiple microphones to be able to do stereo audio recording and/or noise cancellation. The microphone can also be used to measure the ambient noise around the device. The luxmeter measures the light intensity. The camera can take pictures that can be analyzed with image recognition. The proximity sensor is a binary sensor that measures whether something is close to the screen or not, it is mainly used to detect if the phone is in a pocket and disabling the touchscreen when making phone calls.

In recent years near-field communication (NFC) have become a popular inclusion in many smartphones, and this trend is not likely to stop. Both Google and Apple, along with a number of other large companies, are rolling out mobile payment solutions that use smartphones as credit cards through the use of NFC. This is a large market with money to be earned from every transaction, it is therefore likely that every new smartphone will include NFC.

NFC is a continuation of the radio-frequency identification technology (RFID) that has been used for years. NFC devices can still communicate with passive tags that are powered through electromagnetic induction along with peer-to-peer communication between two NFC-enabled powered devices. NFC differs from RFID by only using one frequency, namely the 13.56 MHz band, that makes the maximum usable distance from NFC tag to a smartphone about 10 cm.

2.1.2 Other Sensors

A number of other sensors have been implemented in smartphones, but are not ubiquitous and exist in as few as one model, while others are beginning to become common in new smartphones, or in high-end models. These sensors are: Barometer, thermometer, humidity sensor, pedometer, heart rate monitor, fingerprint sensor, and radiation sensor.

The barometer is used to measure atmospheric pressure, and can potentially be useful for providing extra data points for positioning. Weather measurements and models can provide a narrow range of expected atmospheric pressure at a location, such as the area near a door, and the barometer can be used to confirm that the atmospheric pressure around the phone is in the correct range. Temperature and humidity sensors can likewise be used to provide more data points for an unlocking decision.

The pedometer (step counter), heart rate monitor, and fingerprint sensors are more focused towards the characteristics of the person carrying the device. The pedometer can count steps taken, however, this is also possible with the accelerometer. The fingerprint sensor could be used to authenticate a device to a person.

Lastly the radiation sensor, which is very rare, can measure the radiation level in the area around the device. Radiation levels in an area do not change unless something drastic happens, the radiation sensor can therefore be used to measure the radiation levels at a door.

2.1.3 Sensors of the Future

For the future of smartphone sensor technology, Google have two interesting projects in their Advanced Technology and Projects (ATAP) group. These two projects are Project Soli and Project Jacquard.

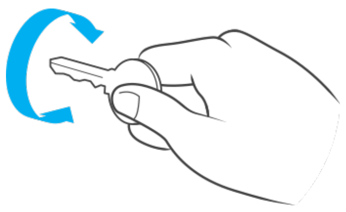


Figure 2.1: Project Soli tries to mimic familiar interactions with gestures.[6]

Project Soli is a project that incorporates a miniaturized radar sensor to detect touchless gestures with very high precision[6]. This is done by emitting electromagnetic waves in a broad beam from the sensor, anything in the beam will reflect electromagnetic waves back toward the radar sensor. By capturing the reflected electromagnetic waves it is possible to recognize size, shape, orientation, material, distance, and velocity of objects in the beam. The information received makes it possible to recognize gestures.

The gestures for Project Soli all follow "Virtual Tool" skeuomorphisms where each gesture tries to mimic a physical tool in order to make it easily understandable for the user. These tools can be buttons, dials, sliders, keys, etc. These gestures could easily be used to communicate with a Bluetooth enabled door, such that locking and unlocking the door only was a matter of gesturing to the phone in your pocket, without any need to take it out and look at it.

Project Soli was shown off at Google I/O 2016 implemented in a smartwatch, using only 0.054 watts of power[52]. Despite this prototype, the project is not ready for the mass market and is likely a couple of years off being available in smartphones.

The other ATAP project, Project Jacquard, has the purpose of weaving touch and gesture enabled surfaces into fabric[5]. By doing this, it is possible to interact with a smartphone placed in the pocket or a bag by touching the clothes you are wearing. The technology is not limited to clothes, but can also be used in furniture and other products made of fabric.

A third Google project, that was formerly a ATAP project but has been moved to its own branch in Google since it is close to release, is Tango[17]. Tango is a combination of hardware and software that can do real-time mapping of 3D spaces. Tango uses computer vision together with depth sensing and motion tracking cameras to track objects, it works much like the Mircosoft Kinect but is integrated in a smartphone.

Moving away from Google, we find a number of other interesting new sensor technologies that are on their way to smartphones and smartwatches. Samsung have a new phone, released in early August of 2016, that includes an iris scanner that is able to provide extra security similarly to a fingerprint scanner by scanning and recognizing an authorized person's iris[3].

Another upcoming Samsung technology is an advanced laser speckle interferometric sensor that is able to monitor heart rate and blood pressure along with pulse, blood flow, and skin conditions of a user[44]. The technology is a bit further out in terms of implementation and Samsung have only recently applied for a patent. The advanced laser speckle interferometric sensor excels from the current optical heart rate sensors in a number of areas. Firstly the laser technology is more precise, and less susceptible to outside forces such as light and skin pigmentation. Optical heart rate sensors work by shining infrared light on the skin and then recording the amount of light reflected. The reflective properties of oxygenated and deoxygenated blood is different, and the amount of light reflected will spike with every heart beat. This means that for any continuous readings to be correct, the sensor must be completely shielded from sunlight, something that is not really feasible for smartwatches as they will need to be worn very tightly. The second advantage that the laser speckle technology has over the optical is the number of features in addition to heart rate monitoring. The advanced laser speckle interferometric sensor is ideal for smartwatches, as it can be used to do continuous measurements of the vital signs.

Other types of sensors that will possibly be included in future smartphones and smartwatches are blood sugar level sensors for diabetics, body temperature, carbon monoxide, and sensors for other types of gas[42].

All in all, the trend for upcoming smartphones seem to be moving away from sensing details about the smartphone itself, and instead monitor the surroundings around the device. The major area of focus seem to be biometric data and ways of interacting with the device without having to take it out of the pocket or looking at it.

2.1.4 Sensor Overview

Not all of these sensors are usable for the purpose of this project, some cannot be guaranteed to be available and some are not feasible to use while the smartphone resides in the pocket or backpack. This last observation is important as the aim of the project is to create a product that requires less interaction than the current solution of pressing a button in an application on the phone. Table 2.1 shows each sensor type and their availability as well as their usefulness when the device is pocketed.

From Table 2.1 we can conclude that the sensors most likely to be useful for this project are the GPS, accelerometer, gyroscope, magnetic field sensor, and possibly the microphone and the barometer. The GPS, accelerometer, gyroscope, and magnetic field sensor are good as they do not require any human interaction in order to function properly, they are available in most smartphones, and they work well while pocketed. The Microphone is also available and does not require interaction, but it may be muffled by being placed in a pocket. The barometer is not as available, but it does not require any interaction, and it works well while pocketed.

The thermometer and humidity sensor might also be useful, they have the same problems as the barometer and might be influenced by the body temperature or humidity.

Sensor Type	**Availability**	**Usable while Pocketed?**
GPS	Most Smartphones	Yes
Wifi	Most Smartphones	Yes
Bluetooth	Most Smartphones	Yes
Accelerometer	Most Smartphones	Yes
Gyroscope	Most Smartphones	Yes
Magnetic Field	Most Smartphones	Yes
Microphone	All Smartphones	Partially Might be muffled Use sensor on smartwatch instead
Luxmeter	Most Smartphones	No Use sensor on smartwatch instead
Camera	Most Smartphones	No
Proximity	Most Smartphones	No
Barometer	High-end Smartphones	Yes
Thermometer	Few Smartphones	Partially Influenced by body temperature
Humidity	Few Smartphones	Partially Influenced by the body humidity
Pedometer	High-end Smartphones	Yes
Heart Rate	Few Smartphones	Yes While touching the phone
Fingerprint	Newer High-end Smartphones	Yes While touching the phone
Radiation	Very Few Smartphones	Yes
NFC	Most Newer Smartphones	Yes
Future Smartphone Sensors		
Radar	Future Smartwatch Prototype Exists	Yes
Touch Textiles	Expected 2017	Yes
Real-time 3D Mapping	Consumer Prototypes	No
Iris Scanner	Recently Available	No
Advanced Laser Speckle	Future	No Ideally placed on smartwatch
Gas	Future	Yes

Table 2.1: Overview and availability of sensors in smartphones.

However, if the temperature difference is great enough, the thermometer may be helpful to decide whether the device is located inside or outside.

2.2 Sensor Usage

The sensors that have been identified and are usable for positioning can be incorporated in a wireless positioning system in one of two different ways.

The first is called self-positioning in which the smartphone will use the sensors in the phone and signals provided by external hardware, such as GPS and Wifi access points, to determine its own position in the world. The other type of positioning is called remote-positioning which happens when the smartphones position is calculated by a remote system based on signals originating from the smartphone. In this case, the smartphone is not responsible for determining its own position, instead the lock or hardware attached to it will make the decision.[49]

Both types of positioning have their pros and cons. Self positioning is often cheaper as can use existing infrastructure such as nearby Wifi access points and Bluetooth devices and the GPS system. Remote positioning needs multiple nodes that capture signals from the smartphone in order to calculate its position. This means additional cost.

Once the position of the smartphone is determined, it can be used in two different ways. Like with the positioning it can either be used by the system that calculated the position, this is called self-consumption. The position can also be sent to a remote system that can use it, this is called remote-consumption. This means that there are four different combinations of positioning and consumption, these can be seen in Table 2.2. The four combinations are: self-positioning with self-consumption, self-positioning with remote-consumption, remote-positioning with self-consumption, and remote-positioning with remote-consumption.

To get a better understanding of the four types, we define them in a system of a smartphone and a lock. The view is seen from the smartphone with the lock as a remote system.

- Self-positioning with self-consumption: The smartphone captures information from sensors on the device and external signals and calculates its own position. The smartphone then uses this information itself.
- Self-positioning with remote-consumption: The smartphone captures information from sensors on the device and external signals and calculates its own position. The calculated position is sent to the lock which uses the information.
- Remote-positioning with self-consumption: The lock calculates the position of the smartphone based on signals originating from the smartphone. The calculated position is sent to the smartphone which uses it.
- The lock calculates the position of the smartphone based on signals originating from the smartphone. The lock uses the calculated position itself.

In order for the two parts of the system to trust the results they get from each other or themselves, the information provided by sensors used and the communication between

	Self Positioning	**Remote Positioning**
Self Consumption	Integrity of Sensors	Auth of Self Auth of Sensors Integrity of Sensors
Remote Consumption	Auth of Self Auth of Sensors Integrity of Sensors Witnesses	Integrity of Sensors Auth of Node

Table 2.2: Generation and consumption of positioning data.

the devices should first be trusted. The requirements for trust depends on the type of positioning and consumption, this is shown in Table 2.2.

In order for self-positioning with self-consumption the system inherently trusts itself, the only part that needs to prove trust is the integrity of the sensors. If the sensors provide incorrect data then the positioning will be wrong. In the case of self-positioning with remote-consumption, the system calculating the position will still need to trust the integrity of the sensors. The remote system needs to trust both the sensors and the smartphone, here authentication of both the sensors and the smartphone is needed. Additionally other remote systems that also authenticate the smartphone can be used as witnesses that provide confidence that it can be trusted.

Remote-positioning with self-consumption is similar to self-positioning with remote-consumption in the way that the two systems need to trust each other before the data can be trusted. If the calculated position is also consumed remotely, the remote system need to trust the integrity of the sensors an the other nodes that help with the positioning will need to be authenticated.

2.3 Positioning Techniques

A number of different techniques for positioning exist, each can make use of different types of data from measurements. The techniques are trilateration, triangulation, and fingerprinting.

The data is dependent of the type of signal measured. One type of signal data is the strength of a signal, which can be either continuous, discrete, or binary. Another is the propagation time of the signal, and lastly the angle of the signal can be used.

2.3.1 Trilateration

Trilateration is a technique that can position a device based on the signal propagation strength and signal propagation time of wireless signals. The distance to a wireless access point can be calculated using the Received Signal Strength Indicator (RSSI) which is the power of the received signal in dBm, in this case the signal strength is used. Distance calculation with signal propagation time can be done using time synchronized clocks, here the received time will differ based on the location of the sender. This is how the GPS system works.

If there is a line of sight between the antenna and the receiving device, the distance can be calculated from the signal strength based on the free-space path loss, which is the loss of signal strength through air with a line of sight. The formula for this calculation is

$$FSPL(dB) = 20log_{10}(d) + 20log_{10}(f) - 27.55$$

Where d is the distance and f is the frequency of the signal[43]. The constant -27.55 changes based on the units used for distance and frequency, in this case meters and megahertz are used respectively. With this information we can find the distance with a line of sight from the RSSI and the frequency of the signal, which is typically 2.4 or 5 GHz.

In order to position a device, a minimum of three wireless signals are required, the reason for this is seen in Figure 2.2. Here three wireless signals can be seen, in the case where only one signal is available, the device will know that it is somewhere on the perimeter of the circle created by the wireless signal. As soon as two signals are available, the possible locations are reduced to two, these are the places where the two signal perimeters intersect. It is not possible to determine which of these two intersections is the correct one with only two signals. The third wireless signal then intersects one of the two intersections created by the two previous signals, this is the location of the device.

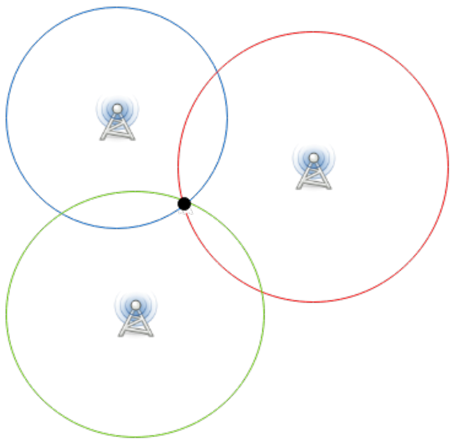


Figure 2.2: Trilateration with three wireless signals.

Trilateration works with any signal that is broadcast, this includes Wifi, Bluetooth, and the GPS system. GPS is a little different as it works in 3D space and thus requires four satellites before it is possible to determine a position on the earth.

2.3.2 Triangulation

Triangulation is a different concept from trilateration, as it is based on the angles rather than the distance to the device broadcasting the signal. Triangulation requires that the positions of two reference points are known. The position of a device is calculated from the measured angles between the current position and each of the two reference points. The angle to each reference point allows to draw a straight line from each of the reference point positions with the measured angle. As seen on Figure 2.3 the two straight lines will intersect, this intersection is the calculated position of your device.

In order to know the angle of a signal, the broadcaster of the signal must provide a way to measure it. The angle to a signal is not possible to measure if only one antenna is

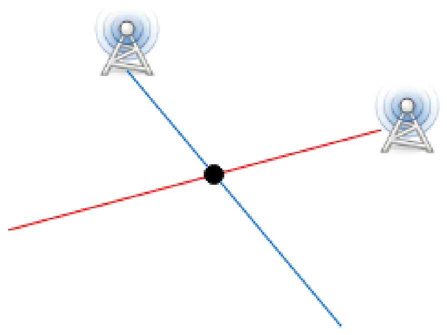


Figure 2.3: Triangulation with three wireless signals.

used, instead a number of antennas placed side by side can be used. The difference in signal strength can be used to calculate the angle. Another option is to use directional antennas which each provide their angle with the signal, the antenna with the strongest signal will be the angle.

As with trilateration, another signal needs to be added for positioning in 3D for a total of three angles, this is one less signal than the four needed for trilateration.

2.3.3 Fingerprinting

Fingerprinting is a bit different from trilateration and triangulation, as it does not necessarily require any broadcast signal. While broadcast signals from Wifi and Bluetooth are helpful in fingerprinting, sensors on the smartphone itself can also be used. The point of fingerprinting is to record a number of signals and tie them to a location.

An example of a fingerprint at a home could be the signal strength and MAC address of nearby Wifi access points, the BeKey Bluetooth lock, the smartphones IP address, time of day or timezone, even signals such as light intensity, noise, temperature, and pressure can be used. The combination of the collected data results in a fingerprint that is unique. For each position in the house, the fingerprint will be slightly different, making it possible to determine positions based on previous fingerprints.

2.3.4 Dead Reckoning

Dead reckoning is a type of positioning which does not rely on any external signals from GPS, Wifi, etc. Instead dead reckoning relies on internal sensors to calculate movement from a known initial position.[49] The initial position can be determined using external sensors, or can be entered manually if known. Dead reckoning typically uses accelerometers to determine movement speed and direction, and gyroscopes to measure the rotation.

Dead reckoning is typically used on ships and in aircrafts where it acts as a backup system in case the GPS system fails. The estimated position that can be provided with dead reckoning is prone to drift because sensor inaccuracies accumulate over time. The estimated position can therefore be quite far off from the actual position which is why most dead reckoning systems make use of GPS to continuously correct it.

2.3.5 Techniques for Types of Sensors

Now that we know the types of sensors that exist in smartphones today and in the future, and we know about the techniques we can use to position a device without using GPS, it is possible to make an overview of which sensors are usable for each technique.

Sensor Type	**Positioning Techniques**
GPS	Trilateration, Fingerprinting
Wifi	Trilateration, Fingerprinting, Triangulation (Requires extra antennas)
Bluetooth	Trilateration, Fingerprinting, Triangulation (Requires extra antennas)
Accelerometer	Fingerprinting, Dead reckoning
Gyroscope	Fingerprinting, Dead reckoning
Magnetic Field	Fingerprinting, Dead reckoning
Microphone	Fingerprinting
Luxmeter	Fingerprinting
Camera	Triangulation
Barometer	Fingerprinting
Thermometer	Fingerprinting
Humidity	Fingerprinting
Radiation	Fingerprinting
Gas	Fingerprinting

Table 2.3: Overview of usable positioning techniques for each sensor type.

Table 2.3 shows that most sensor types can be used for fingerprinting, which supports the theory that fingerprinting is possible with many different types of sensors. The GPS system can be used for trilateration, as it is by every device using GPS for positioning.

Bluetooth and Wifi technology is the only other technology the emits a signal which can be used for trilateration with a distance measurement from each signal. If additional antennas are added to each device, or if additional signals are added such that it is possible to measure the angle of the signals, it is also possible to use these technologies for trilateration.

Finally, dead reckoning techniques can be used by the accelerometer, gyroscope, and magnetic field sensors.

2.4 Indoor Positioning

Determining the location of a device quite easy, reasonably precise, and reliable with the use of GPS positioning. Once the device is moved inside a building, GPS signals are not well suited. This is partly because the signals can be blocked by the building, preventing the device from getting enough information in order to determine the precise location, and partly because more precision is needed indoors. For some purposes, it can be necessary to be able to distinguish between floors and rooms, and which way the device is moving past a certain door. This is where indoor localization techniques take over from GPS, many of these techniques rely on Wifi access points.

Determining the location of a device indoors through the use of Wifi access points is not an easy task. The access points use either the 2.4 or 5 GHz wireless spectrum which are easily blocked by the human body and other objects inside buildings. This means that the signals fluctuate, and as a result many algorithms require maintenance and recalibration.

2.4.1 Attacks and Prevention

In order to create a secure localization implementation, it is important to know the types of attacks there exist such that the implementation can prevent these.

The Wormhole Attack

The wormhole attack consists of an attacker that receives one or more packets from one network and tunnels them to another network where they are replayed into[36].

In the case of a door unlocking, an example of a wormhole attack would be an attacker could receive Bluetooth and Wifi packets near the lock, and tunnel them to the owners location in an attempt to fool their phone into thinking it is close to the door. A potential unlocking attempt could then be tunneled back to the door and replayed, thus unlocking the door while the owner is nowhere nearby. Due to tunneling the signal, the replay can be carried out as bits are captured and tunneled, there is no need to wait for complete packets.

Detecting a wormhole attack can be done by using a packet leash. A packet leash is added to a packet in order to restrict the distance that the packet can travel. Each leash only works for the direct transmission between two devices, if the packet is sent through multiple devices from source to destination, each hop will need a new leash. Two types of packet leashes exist, geographical and temporal leashes. Both are based on distance, but geographical leashes require each node to know their own location and compare it with the location included with the packet. Temporal leashes use the fact that packets are traveling at the speed of light to limit their lifetime, this limit ensures that packets with a temporal leash cannot travel too far from the source. Geographical leashes will restrict the packets to a geographic location.

If we want packets to be restricted to the area around a door with a Bluetooth lock, a range limit could be in the range of 25 meters. Light will travel about 30 cm in a nanosecond, giving us a temporal leash time limit of around 84 nanoseconds. This very low time limit means that the two devices communicating with each other, in this example the lock and a smartphone, will need to have very tightly synchronized clocks. The geographical leash does not require that the clocks of the two devices are that closely synchronized.

For our case the lock will always have the same location and the smartphone is able to determine its own location. Synchronizing the clocks of the two devices will not work if they are not near each other, and furthermore the two clocks are likely to drift apart over time. Instead, the distance between the two could be calculated with communication between them.

Distance Bounding Protocols

Preventing relay attacks can in addition to packet leashes be done using distance bounding protocols.[11] Distance bounding protocols do not require the clocks of two separate devices to be perfectly synchronized, instead these protocols rely on the fact that the electromagnetic signals used by Bluetooth and Wifi networks travel near the speed of light and cannot travel faster than that.

Scenarios where a distance bounding protocol is useful, are scenarios where a system consists of two entities where one has to prove to the other that it is nearby[12]. The system will thus consist of a prover and a verifier. The idea behind distance bounding protocols is to use challenge-response authentication where the verifier sends a challenge to the prover, the response to the challenge is ideally very quick to calculate such that the processing time has as little impact as possible. The challenge is issued such that the prover cannot try to send a response prematurely and reduce the calculated distance between the two entities. This means that the challenge must be random and unpredictable such that the prover cannot guess the response.

Once the prover has completed the challenge, it must immediately send a reply back to the verifier with the response. The minimal calculation time of the response means that the majority of the time spent between the verifier sending the challenge out and receiving the response back is the time it takes the electromagnetic signals moving through the air between the two entities.

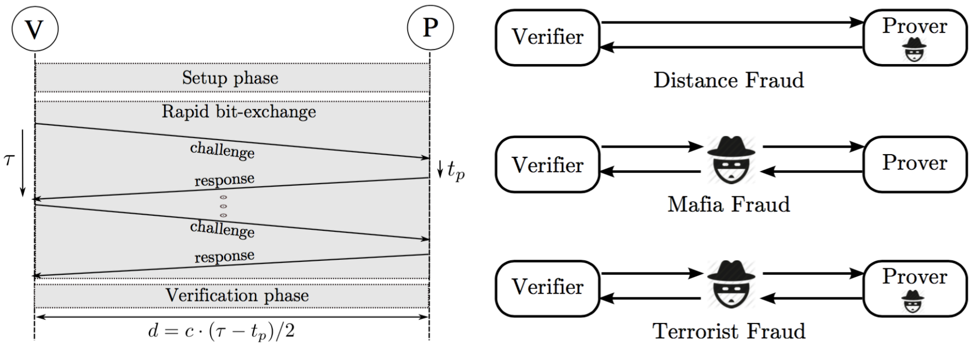


Figure 2.4: Left: The three phases of a distance bounding protocol, Right: Three types of attacks on distance bounding systems.[50]

On the left in Figure 2.4 is the three phases of a distance bounding protocol. First phase is the setup, where the two entities initiate communication between each other. The second phase is the challenge-response phase, more than one challenge and response is sometime used in order to reduce the calculation time of the response. A single bit response is a good way of lowering the calculation time, but it requires a number of challenges and responses as one challenge would be too easily guessable. The third phase is the verification phase, where the verifier calculates the distance to the prover using the formula

$$d = \frac{c * (\tau - t_p)}{2}$$

where d is the distance, c is the speed of light ($3 * 10^8 m/s$), τ is the delay between the verifier sending the challenge and receiving the response, and t_p is the processing time of the prover.[50] If the processing time of the prover is 1 ns, the distance accuracy will be 15 cm.

On the right of Figure 2.4 are three types of attacks on distance bounding protocols. The first attack is the distance fraud, where a dishonest prover tries to lie about their distance to the verifier. It is trivial for the prover to convince the verifier that it is further away than it really is, all it needs to do is to delay the response and the distance calculated by the verifier will be longer than the actual distance between the two entities. What the dishonest prover cannot do is to lie itself closer to the verifier than it is, the reason is that the prover only can influence the processing time, not the time of flight for the signals, and the distance bounding protocols have already minimized the processing time.

The second attack, the mafia fraud attack, is also known as the relay attack which was described in **??**, here both prover and verifier are honest. The relay will add additional distance between the prover and verifier, and with distance comes a longer time of flight for the signals. It is not possible for the relay to speed up the signals as they are already traveling close to the speed of light, the relay cannot lie the prover closer to the verifier than it is. This means that distance bounding protocols prevent relay attacks.

The third and last type of attack on Figure 2.4 is the terrorist fraud attack, which is a relay attack where the prover is dishonest and collaborates with an external attacker. If the prover does not share his secret key with the attacker, then the prover is still responsible for responding to the challenges sent by the verifier. Thus, for the same reasons as the relay attack, the distance bounding protocols will protect against the terrorist fraud attack.

The Sybil Attack

The sybil attack is an attack where one node in the wireless network claims to have multiple identities and thus claim to be multiple different nodes.[45] These sybil nodes can either have stolen or fabricated identities. Stolen identities are identities where the sybil node impersonates an already existing node and either destroys or temporarily disables it. A fabricated identity is a random identity generated by the sybil node that mimics already existing nodes.

Communication between a sybil node and a legitimate node can be either direct or indirect. Direct communication is, as the name suggests, when a sybil node manages to send and receive messages from a legitimate node. Indirect communication happens when one or more malicious nodes pretend to be able to reach sybil nodes.

2.4.2 Techniques for Indoor Positioning

A number of techniques for indoor positioning exist. These techniques make use of Bluetooth/Wifi, cellular networks, GPS, or RFID technologies with fingerprinting or trilateration for determining the position inside a house or office building.[41] While GPS works well outdoors, it is often blocked by buildings and is thus not very suitable for positioning indoors, the same can be said for cellular signals as they also originate from outside buildings.

Some techniques attempt to do positioning without knowledge of the layout of the building while others require a mapping of the building to be loaded before any positioning can be done. The mapping of a building is traditionally not something everybody has access to, and it is not likely that a costumer would map their house and transfer it to their phone for an application, but with Google Tango being available in phones it might be possible to do this quite easily in the very near future.

RFID based techniques can be used to estimate a position with an error rate of 1-2 meters. The LANDMARC[46] and SpotON[35] projects both use a number of active RFID transmitters to position a passive RFID tag by trilateration. Unfortunately both of these projects suffer from a slow pinpointing of the position. SpotON requires between 10 and 20 seconds estimate a position while LANDMARC requires up to one minute to estimate signal strength and have a delay of 7.5 seconds between two consecutive readings from an active RFID tag. This means that these can estimate the position of a stationary object quite well but are unsuited for estimating the position of a moving object.

Techniques using Wifi and Bluetooth often use fingerprinting of nearby wireless access points. The fingerprinting is done by using the received signal strength (RSSI) from each access point. The RSSI measured is dependent on the distance to each access point, material between the access point and the phone such as walls or people, and it fluctuates even when there is an unobstructed line of sight between the two devices. Still, indoor positioning with Wifi seems to be quite accurate. An example of fluctuations and the probability of the correct RSSI reading can be seen on Figure 2.5 from the Horus[57] positioning system.

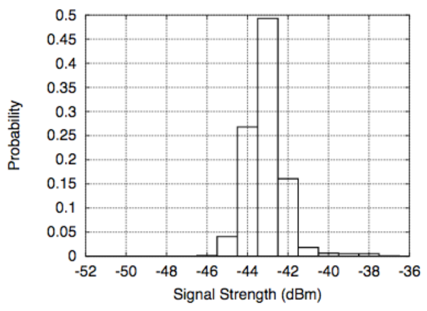


Figure 2.5: Variations in received signals strength and their probability.[57]

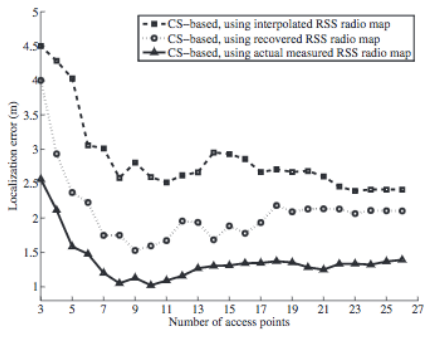


Figure 2.6: Precision of indoor positioning with respect to the number of available access points.[26]

The RSSI fluctuates with up to 10 dBm the overwhelming probability is within 3 dBm. These figures come from a sample of 300 measurements over 5 minutes. The average dBm for a single position does not necessarily correlate with the measured RSSI for other positions as small-scale variations can change the RSSI by up to 10 dBm with a movement of as little as 7.6 cm.

Another project measured the precision of their positioning with respect to the number of access points available, seen on Figure 2.6.[26] Here the estimated position error decreases with the number of access points, from an error of 2.5 meters with 3 available access points down to 1 meter with 8 and a small increase with more. The Horus project also points out that the precision of the positioning decreases with fewer samples. The latency of the positioning increases with the accuracy as the number of samples increase.

This means that if the true position is moving, the estimated position necessarily have a lower accuracy. A good positioning accuracy of a stationary device is between 1 and 5 meters.[41]

Chapter 3

Analysis

3.1 The BeKey System

BeKey currently provides two types of smart locks. One is intended to be mounted on the inside of doors and unlock them by hooking into the existing unlocking mechanism. The locking and unlocking is made possible by a motor that turns the thumb turn that already exists on the inside of the door as seen on Figure 3.1.

The other type of smart lock that BeKey provides is designed to be installed in the front door of apartment buildings with an electronic buzzer that unlocks the door. Many apartment buildings have an intercom system that allows the residents to open the front door from their own apartment when guests arrive to the building. The device on Figure 3.2 is designed to hook into this buzzer system and will send the unlocking signal similarly to the smart lock.

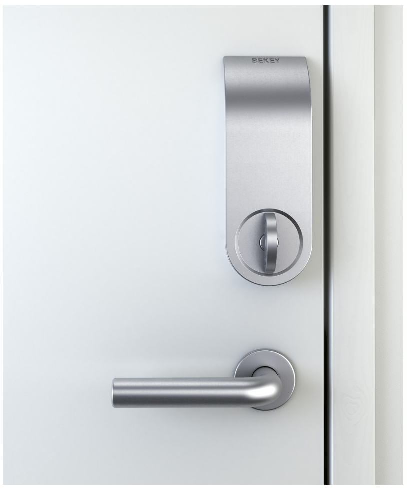


Figure 3.1: The BeKey smart lock.[8]

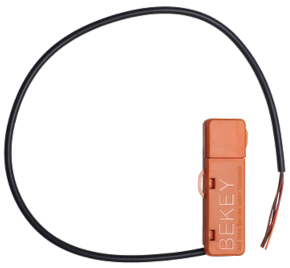


Figure 3.2: The BeKey that hooks into existing electronic unlocking systems.[9]

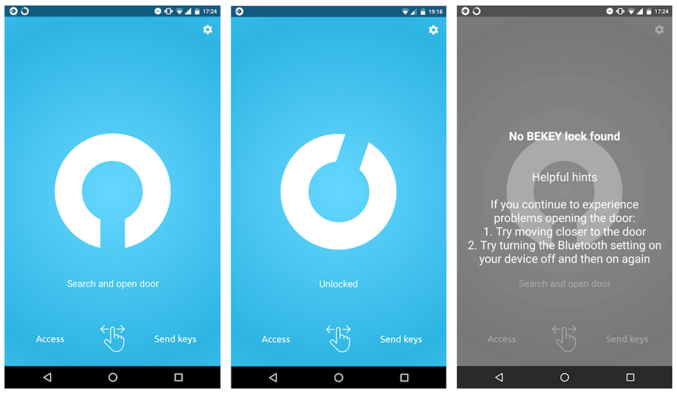
For the smartphone, these two types of devices are the exact same. Both use Bluetooth to communicate, and they both send an announcement signal twice a second that lets the smartphone find the lock when it is nearby.

3.2 Unlocking Today

Making an application that can unlock a Bluetooth door is not a new thing, the BeKey application can do exactly that and applications for other smart locks can do the same. Making an application that can do it automatically without errors is another problem entirely, but that does not mean that it has not been attempted before. The automotive industry has had keyless entry for years, and a number of homebrew solutions for doors and offices also exist.

In order to get an idea of how these solutions work, a number of them will be analyzed. Starting with the BeKey application.

3.2.1 BeKey Application



No BEKEY lock found

Helpful hints

If you continue to experience problems opening the door:
1. Try moving closer to the door
2. Try turning the Bluetooth setting on your device off and then on again

Search and open door

Figure 3.3: Initiating, successful, and unsuccessful unlock in the BeKey application.

The current BeKey application for interacting with the BeKey Bluetooth lock is quite simple. Once a lock is setup, the main user interface consists on one button that initiates a search for a known BeKey lock and unlocks if it is found. If no BeKey lock is found nearby, the application will provide the user with a message informing them that no nearby lock was found that also includes a couple of helpful hints. If more than one lock is nearby, the application will provide a list of nearby locks and ask the user to choose which one to unlock. Examples of the BeKey application can be found in Figure 3.3.

Other functionality in the application consists of adding and managing locks, as well as sharing keys with other people.

3.2.2 Automotive Remote Keyless Entry and Smart Keys

Remote keyless entry has been available in the automotive world for decades now. For much of this time it has consisted of one or more buttons on the key to the car that can remotely lock or unlock the doors. Much like the current smart locks for doors, this has required the user to press this button for the desired action.

In the world of cars you look to the Mercedes S-Class for new technology. The 1998 model S-Class introduced a smart key developed by Siemens, this key allows the user to unlock, start, and stop the engine without the key leaving their pocket. Unlocking the car is done by pulling the door handle when the smart key is nearby, starting and stopping the engine is done with the press of a button located on the gear shifter. Locking the door required a press on a button on the door handle[23].

Since then many if not most other car manufacturers offer models with smart keys. Volkswagen, for example, offers keyless go as an add-on in their Golf model, which means that smart keys are not exclusive to luxury cars. Another change since the first S-Class is the way locking and unlocking works, some car models manage this completely automatically based on the users distance from the car. If the smart key is close enough to the car, the doors will unlock, and when the smart key is no longer in range, the car will lock the doors again. There is no need to pull out the key or press any buttons unless the car is to be started.

Figure 3.4: A smart key in the usual key fob form factor[55].

Figure 3.5: A smart key in the credit card form factor[38].

The car manufacturers still require a separate key for the car. This key can be in the form of the car keys people are used to carry, such as Nissan's offer seen on Figure 3.4, or they can be in the form factor of a credit card, like the Kia smart key in Figure 3.5 that can be placed in a wallet. Bluetooth enabled smart keys that target the car manufacturers are being developed by companies such as Qualcomm[16], but their solution still requires a physical key to enable communication between smartphone and car. Third-party solutions that promise keyless entry without any physical key exist[54].

Security Concerns

The security of remote keyless entry systems has been questioned for a long time with new vulnerabilities being discovered periodically. A recent report shows how the system used by the Volkswagen group and the system used by a large number of other manufacturers are vulnerable to attacks that can be used by third-parties to unlock cars.[28]

The passive remote keyless entry systems, which do not require a button press on the key, use a challenge-response protocol that is used to secure the unlocking. They are often limited in distance by the power of the signals transmitted by the car. However, these systems are often vulnerable to relay attacks. The remote keyless entry systems that require a button press were in the beginning not using any cryptography to secure the communication and were thus vulnerable to replay attacks. Later systems have incorporates a number of different cryptography schemes and rolling codes for each unlock. Many of the systems have proven to use weak cryptography that have since been broken.

The remote keyless entry system used by the Volkswagen group from 1995 until today has been found to rely on a single master key for their cryptographic system. The system makes use of a rolling code that can be used in combination with the master key to unlock a large part of the vehicles produced by the Volkswagen group in the time period. The system used by many other car manufacturers does not have a master key, however, they use the Hitag2 cipher, which is cryptographically weak and the key can therefore be cloned by eavesdropping four to eight of the rolling codes used.

All in all, these systems are not very secure, but they are widely used and most people do not seem concerned by the security issues mentioned.

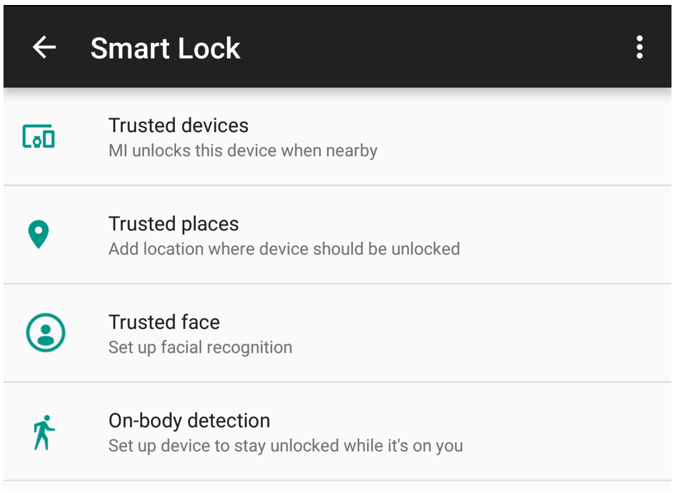
3.2.3 The Homebrewed Solutions

Though there does not seem to be any commercial products, a number of homebrewed solutions for automatic unlocking of a door with a smartphone exist. As with all homebrewed solutions they vary highly in solution, success, and completeness, but a couple of them are interesting for this project.

One project by Matthew Carlson on hackaday.io[13], attempts to solve the problem by unlocking the door by sniffing authentication packets when a new device authenticates on the Wifi network. The reasoning is that smartphones will often try to connect to the Wifi network before the door is reached, meaning that the door will be unlocked when the people living in the house are coming home.

3.2.4 Automatic Unlocking of Computers and Smartphones

Automatic unlocking is also present in the world of computers and smartphones. Google has a “Smart Lock” feature that works on Android, the Chrome browser, and Chromebooks. The Android Smart Lock, which can be seen on Figure 3.6, includes a number of different ways to automatically unlock a password protected smartphone.


Smart Lock

Trusted devices
MI unlocks this device when nearby

Trusted places
Add location where device should be unlocked

Trusted face
Set up facial recognition

On-body detection
Set up device to stay unlocked while it's on you

Figure 3.6: Smart lock options on an Android device.

The main interesting feature of the Android Smart Lock is “Trusted devices” which allows the smartphone to automatically unlock when a trusted Bluetooth or NFC device is nearby. The Bluetooth devices can be anything from smart watches to a key fob, the only requirement is that the two devices have been paired.

Apple has a similar feature in their macOS 10.12 called Auto Unlock that will unlock the computer if a trusted Apple Watch is nearby[1]. The Apple Watch will need to be unlocked, and it needs to be worn before it will unlock the computer. An Apple Watch lying besides the computer will thus not trigger an unlocking.

Both of these are examples of zero-interaction authentication where a physical authentication token is used to wirelessly unlock the device.[15] Zero-interaction authentication is useful for laptops and mobile devices because it allows the device to automatically lock itself in case of theft by locking an attacker out before they can do anything. It does this without inconveniencing the user, and in cases where it is used to unlock, the user will not need to enter a password.

3.3 Types of Door Locks

Different types of door locks will have an impact on how they can be controlled by an add-on Bluetooth device. The two most widely used are the dead bolt and the latch bolt, therefore these are the ones that will be described.

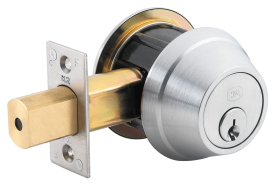


Figure 3.7: Dead bolt lock type.[51]

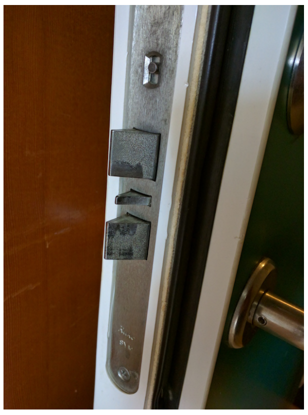


Figure 3.8: Latch bolt lock type.

Dead Bolt

The dead bolt (Figure 3.7) is a type of lock that needs to be explicitly locked and unlocked with a key, or with a possible twist knob on the inside of the door. This means that the bolt in the lock cannot be moved by putting force on the bolt itself. Some doors with dead bolt locks have additional security that requires the door handle to be lifted before they can be locked, such that additional bolts are extended at the top and bottom of the door.

Automatic locking and unlocking of this type of door could potentially be done in the same way as done on cars with smart keys today. This would require additional processing on the lock that has the ability to detect if a trusted device is nearby or not.

Latch Bolt

The latch bolt (Figure 3.8) relies on a spring to extend the bolt which locks the door. The spring mechanism means that the latch bolt always locks the door once it is closed, unless it is explicitly disabled. It is common that doors with latch bolt locks will open once unlocked, as the unlocking will also retract the separate latch attached to the door handle.

3.4 Intensions

When a door is being unlocked with a key or a press on a button in an app, the person unlocking the door shows a clear intension of doing so. In order to enable automatic unlocking by observing the movement of a person, it is important that the intent of that person is captured. If there is no intent to unlock a door, and the movement of the person carrying the smartphone making the decision is moving past the door without needing to enter it, the door should not open by itself.

3.5 Types of Entrances

Entrances to homes are very different depending on the type of building it is placed in and where in the world they are built. Apartments in the city are very different from houses in the suburbs, dormitories are different, and town houses with more than one family per building is again another type. In order to get an overview of the types of entrances, and how to recognize them with the smartphone, we will look at the most common types of entrances.

3.5.1 Apartments and Dormitories

In Copenhagen it is very common that an apartment staircase has multiple apartments per floor. Here, doors are often either right next to each other, or they face each other on each side of the staircase. The staircase itself rotates on itself, with a plateau every half floor. This means that there is a plateau with entrances to apartments followed by a plateau without apartments. This pattern should be easily recognizable by the accelerometer, with the possibility to recognize the exact number of plateaus before the correct entrance is reached.

A thing to strive for with these types of entries, would be to avoid unlocking the door before the user is within half a floor. Opening the door while the user is on the floor below could give the impression that the lock is unsafe or has unlocked the door by itself at some point during the day. Furthermore, it should be avoided that the door is unlocked on the way down the staircase, if the Bluetooth signal from the lock was out of range for a short period while the user is exiting the building.

Another type of entrance is the one typically seen in dormitories, where a lot of small rooms are located very close to each other. The entrances to these rooms are often within a few meters of each other, some times they are even situated on both sides of a narrow hallway on multiple floors. Some dormitories have a separate kitchen that is shared between multiple rooms. This means that the user will often leave the room and walk to the kitchen, which might be very close. Ideally, the door should not unlock while the user is in the kitchen, but should unlock when they leave the kitchen again and go to their room. Examples of these types of entrances can be seen on Figure 3.9 and Figure 3.10.

Figure 3.9: Rooms at the Professor Ostenfeld dormitory.

Figure 3.10: Student housing in Roskilde.

3.5.2 Suburbs and Town Houses

Houses in the suburbs often have a yard with a spot to park cars and an entrance that is most commonly used. Unlike most apartments, houses in the suburb will often have more than one entrance, sometimes close to each other. The people living there will, in addition to staying inside the house, sometimes move around in the yard. While moving about in the yard it is not ideal if the door is continuously unlocked as the smartphone passes by the door. Instead, the door should only unlock when the intention is to go inside the house again.

Figure 3.11: A suburban house.

Multiple doors potentially mean multiple Bluetooth locks. If a house has multiple Bluetooth enabled doors, it is important that the correct door is opened. In some suburban houses, the garage is located right next to the utility room which has its own entrance. The main entrance, the one that is mainly used to enter and exit the house, is located further away. For this case, the door to the utility room should not be the one opening when the residents arrive home.

Suburban houses and town houses typically have a yard, which means that the distance to neighbors is typically larger than in apartments and dormitories. This means that the amount of discoverable Wifi and Bluetooth devices will likely be a good amount lower than in apartment buildings. Coupled with the fact that there are often more than one entrance, and that the residents are likely to move past these doors without wanting to enter, suburban houses and town houses are likely to be more difficult to correctly predict unlocking for.

3.6 Network Location

Most smartphones automatically keep track of their location without the user needing to do anything. This gives the possibility of features such as location based weather forecasting, the ability of providing the user with suggestions for things nearby, and a number of other things.

As phones are battery powered, there is an incentive for this location tracking to use as little power as possible. Normally, the built-in GPS is used for getting an accurate location of the phone. However, using the GPS in a phone uses a lot of power and most phones would run out of charge if they had to rely on the GPS signal constantly. While the GPS gives a very accurate location, it is not always necessary needed to know the exact location in order to give the user any information needed most of the time it is good enough to just have an approximate location.

In order to get an approximate location, without using too much power, smartphones make use of multiple low power network signals. These signals consist of cellular signal, Wi-Fi, and Bluetooth, all of which are often turned on anyway. As this technique relies on cellular towers and Wi-Fi networks, it will be more precise the more of these are nearby. In cities with many cell towers and Wi-Fi networks, the location accuracy of this technique can be as low as 10 meters, while areas that have low or no cellular towers or Wi-Fi networks have lower accuracy of hundreds of meters to not being able to use the network for location of the phone. In these cases the phone will have to rely on the GPS signal.

3.6.1 Geofence

A geofence is the ability to specify a virtual perimeter for a real-world location that the smartphone will know when it enters and exits. The geofence can be represented as a point with a specified radius around it, it can also be defined as an area of any shape. Whenever the phone detects that it has entered a specified geofence, it is able to

The result of smartphones always keeping track of their location is that they are able to notify apps when the phone is near a specified location. For this, apps can specify a geofence.

The circumference of the circle specifies the geofence, any location inside the circle will be in the geofence while any location outside of the circle will not. Geofencing means that unless an app is used for navigation or needs very accurate location tracking, it will not need to continuously keep track of the location and can instead rely on the operating system to notify the app when it needs to take an action.

3.7 Choice of Platform

When choosing the platform for the project has to consider a number of factors in order to create the best possible product. The market share of the possible platforms will make a large difference for the usability of the project in the end. If the platform has a low market share, then only few people will be able to use the end product. Some of the important functionality of the project will require features to be supported by the operating system, if the OS does not support features like background tasks or geofencing then the end product will not be usable.

3.7.1 Market Share

The three most popular smartphone operating systems are Android (∼85%), iOS (∼13%), and Windows Phone (∼2%)[29, 37]. The usage of other operating systems are essentially non-existent, Windows Phone never really gained traction and has had falling market share for a while. Thus the decision is between using Android and iOS.

The share between iOS and Android is not the same around the world. In most European and North American countries the market share between iOS and Android is more equal than the global numbers suggest, here the market share of the two platforms are more 50/50, with iOS even beating Android in some countries[18]. In Denmark iOS seems to be the most popular smartphone operating system, and thus the choice is not necessarily easy based on market share alone.

3.7.2 Geofencing

Both Android and iOS support geofencing and have the ability for apps to let the system know which geofences they need to know about. Whenever the system detects that the phone has entered a geofence, apps that have registered with the system will be notified about the location and can begin any background task needed.

3.7.3 Background Tasks

The intent of the project is that it will be continuously running in the background without user interaction, therefore it is vital to the project that the application can be running in the background and monitor the users location and movement.

Android has the possibility for applications to run a service[21] that will be running in the background without providing a user interface. Android services are used for long-running operations that will continue to run in the background even when the main application is closed. Android services can be killed if the system is low on memory, but they are generally running at all times when they have been started.

iOS handles applications differently than Android and does not allow the same freedom for applications to run in the background. iOS has the possibility for apps to implement background execution[40] for finite-length tasks, and for specific functionality it is possible to implement long-running tasks. This functionality includes tracking the location of the device, thus it should be possible to create a long-running task for the purpose of this project.

3.7.4 Programming Language

The programming language used to create applications on Android and iOS are partly decided by the operating system itself. iOS applications are mostly written Objective-C or the new language Swift which is the replacement language for Objective-C. Android is mostly built using Java, and applications are generally written in Java as well.

A number of projects exist that provide the ability to use other languages than the ones typically used. One such project is Xamarin[56], a project that promises the ability to create applications on iOS, Android, and Windows Phone using a shared codebase written in C#. Xamarin promises that applications will look and run like applications written for each operating system separately, all while sharing 75% of the codebase on average.

As the three operating systems handle background tasks very differently, each operating system will need its own implementation for background tasks. Given the nature of our application, the benefits of Xamarin are somewhat negated.

3.7.5 Choice

The prototype will be developed on the Android platform using Java. The reason for this is that Android allows more freedom for long running services and seems to be the more popular operating system. Java has been chosen over C# because the focus is only on one platform.

3.8 Android

Android provides a number of functions and services that might be helpful when creating an app that relies on sensor information and location data. Furthermore, Android allows an app to run as a service in the background that will not be killed unless the phone is low on memory.

3.8.1 Bluetooth

In order to communicate with and unlock the door, the application will need to make use of Bluetooth. Android provides methods to turn the Bluetooth adapter on and off programmatically, meaning that Bluetooth does not need to be enabled at all times in order for the app to function. Instead the app can turn the Bluetooth adapter on when the phone enters a geofence, and turn it off when the door has been unlocked or based on a timeout.

When the Bluetooth adapter is on, the application can continuously discover new phones, or query already paired phones in order to find the door sensors. The method used will depend on if there is a need to require that the phone and door sensors are paired before allowing unlocking, or if the phone just needs to be aware of the signal strength of the door sensors.

The connection between two Classic Bluetooth phones consists of a client and a server. In order to connect the two, one of the phones must act as a server that opens a server socket and the client must then initiate the connection by using the servers MAC address. To reduce the power consumption of the door sensors and make their batteries last longer there exists a newer Bluetooth standard called Bluetooth Low Energy which is designed to use as little power as possible.

Bluetooth Low Energy provides the same range of discoverability as Classic Bluetooth and even provides a much lower latency when sending data from a non-connected state. In order to reach a lower power consumption, Bluetooth Low Energy has a much lower data transfer speed and throughput, meaning that it is not usable for applications that transfer large amounts of data. However, the BeKey locks do not require a lot of data to be transferred in order to unlock, which makes the Bluetooth Low Energy well suited.

Bluetooth 5

In a recent announcement the details of Bluetooth 5 have been made public [33]. Some of the improvements to the next version of Bluetooth are likely to make the communication between smartphone and lock better. Most interestingly, the new standard promises a quadruple increase in the range of low energy connections. An updated lock with Bluetooth 5 and a smartphone that can take advantage of the improvements will likely make the product better for the end user. Any unlocking decision will be available to be made further from the lock, and in cases where the door blocks much of the signal from the lock, it is possible that the door can be unlocked in the time between the smartphone detects the lock and the user reaches it.

3.8.2 Wifi

Wifi communicates on the same 2.4 GHz frequency band as Bluetooth, newer smartphones can also use the 5 GHz frequency band. Because of this similarity Wifi and Bluetooth are often handled by the same hardware in the phone. Additionally, the access of both Bluetooth and Wifi is quite similar within Android, and Wifi can like Bluetooth be enabled by applications automatically if needed. This means that Wifi can be used to recognize the surrounding area of a smartphone with fingerprinting even if the user is not actively using it.

3.8.3 Sensors

Due to the large amount of different Android smartphones, it is not possible to guarantee that every phone has all of the sensors needed, and that the data they output are of high enough quality. While high end Android phones have generally good quality sensors, some cheaper phones use sensors of a lower quality in order to save money. This might make the detection of direction and movement of the phone unreliable on the cheaper Android phones.

Android supports three types of sensors, motion, position, and environmental sensors. Motion and position sensors track the phones movement and position using accelerometers, gyroscopes and orientation sensors. These are the two types of sensors that we are interested in for this project. Environmental sensors would provide little value to the project, as they measure the environment around the phone, this can be done using thermometers, barometers, photometers, and more.

Android provides access to the sensor data for any app, some of the interesting sensors for us are acceleration, rotation, and the magnetic field sensor.

Accelerometer

The accelerometer measures the acceleration force in m/s^2 on all three physical axes. The Android API specifies how each axis should be oriented in relation to the phone. Figure 3.12 shows how the axes are specified by the Android API, this specification is the same for any sensor that provides data in relation to the physical orientation of the phone.

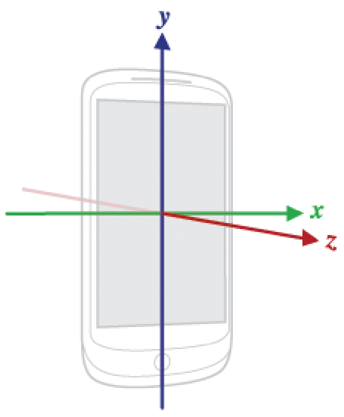


Figure 3.12: Sensor coordinate specification in accordance with the Android API.[20]

The accelerometer is used to detect movement of the phone. Any movement will result in the accelerometer reporting the change in force for the direction of movement. The accelerometer also measures gravity, which means that even when the phone lies completely still the accelerometer will show an acceleration force of 9.8 m/s^2 in the direction perpendicular to the ground. The gravity thus makes it possible to determine the orientation of the phone at all times.

For purposes where the gravity is not needed or is unwanted, the *linear acceleration* sensor is available. It provides the same data as the accelerometer but the gravity is filtered out. This means that a phone lying still will show 0 acceleration on all axes.

Magnetic Field

The magnetic field sensor is used to measure the ambient geomagnetic field and is used to create a compass for the phone to use. The ambient magnetic field is measured in micro-Tesla (μT) in the same three axes as the accelerometer.

The magnetic field sensor can be used to get the orientation of the phone in relation to the earth. The values for each axis will change depending on the phones orientation. In the end, this can be used in combination with the gravity from the accelerometer to rotate the acceleration data independent of the orientation of the phone. The acceleration data can thus be locked to the geographic coordinate system with the x-axis going east/west, the y-axis going north/south, and the z-axis going up/down.

3.8.4 Intents

Android provides a messaging system that applications must use in order to communicate with each other. Messages in Android are called intents, and are used for all communication between both applications and application components such as activities, services, and broadcasts. An intent can either explicitly specify which component it is meant for, but can also be implicit and instead declare a general action that can be used by a component in a separate application.

3.8.5 Location

There are two types of location sensors available in Android. The first one is the GPS sensor, which uses GPS satellites to calculate the current location. The other is network-based location, which uses the Google Play Services to get a location based on nearby Wifi, Bluetooth, and cellular towers.

In most cases the GPS location will be more precise, but it takes longer to start up and might not be able to get contact to the satellites inside buildings. For situations where the GPS is unreliable, the network-based location will be a suitable backup solution to get the current location of the phone.

The location that is returned from the GPS and network-based location contains information including latitude, longitude, accuracy and speed (GPS only). It is also possible to get the distance to a location in meters.

Geofences

Adding a geofence in Android is done by registering a location with the Android system. The geofence registration requires a latitude and longitude as well as a radius and an expiration time. When the geofence is entered or exited, Android generates an intent that will alert the application.

It is thus possible to leave our application waiting without doing any calculations until the phone is nearby a door that it can unlock. This means that the application can activate GPS location for better tracking of the phone where it is useful without using any extra power where it is not.

Once the phone is inside the geofence, it is possible to continuously calculate the distance to the door in order to know when we expect to see the Bluetooth signals from the lock.

3.8.6 Mode of Transportation

The Android class `DetectedActivity` has methods for detecting the activity of the phone, and includes an associated confidence in a range of 0 to 100. The activities that can be detected are in vehicle, on bicycle, and on foot. On foot can be either walking or running, the phone lying still or tilting are also detected.

The activity of the phone can be used to determine if the user is moving in a pattern that is usual in the time preceding the unlocking, in addition to other data from location, Bluetooth/Wifi, accelerometer, and time from entering the geofence to reaching the door.

3.8.7 Permissions

As a security measure Android locks some functionality for applications by default. The application will need to request to use parts of this functionality through the permissions system. The general idea behind this system is to inform the user of a phone about application that may use parts of the phones functionality that may cost money, access private information, or locate the phone.

In order for an application to use these features, the permission must be declared in the Android manifest file. This informs the Android system that the application should be run with the specified privileges. Prior to Android 6.0, all permissions are granted when the user installs the application. The user can see an applications required permissions when they install it, or if new permissions are added in an update. This means that the user will have to either grant all permissions or not install the application. Android 6.0 provides a more fine grained permission system which is closer to the permission system in iOS. The first time an application needs to use a permission, the Android system will ask the user if they want to grant it or not. If the user does not grant a specific permission, then the application will continue to function without the functionality requiring that specific permission.

This fine grained permission control allows the user to use an application without granting it permission to unwanted or unneeded functionality.

A number of permissions are needed in order to access the identified sensors that will be used. These permissions are:

- `ACCESS_FINE_LOCATION` which gives access to the GPS. This permission also implies the `ACCESS_COARSE_LOCATION` permission which grants access to network based location.
- `ACCESS_WIFI_STATE` allows the application to access information about wifi networks
- `BLUETOOT_ADMIN` allows the application to discover and pair Bluetooth phones, additionally `BLUETOOTH` allows an application to connect to already paired phones.

Sensors like accelerometer and magnetic field are available for all Android applications without requesting permission.

3.8.8 Services

Android provides two types of services, bound and unbound. The difference between these is that a bound service can be bound to another component such that the service for example handles the download of a file or playback of music while the bound component handles user interface and can be closed while the service will continue its operation in the background. An unbound service will strictly run in the background and does not provide the ability to create a user interface that can interact with it. It is possible for a bound service to start unbound and later bind to a component.

When a service has been started, it will continue to run until it is stopped by either the parent application or the Android system. It is possible to avoid that Android kills a service, this requires that the service is run as a foreground service. A foreground service must provide a notification in the Android notification drawer at all times, as it is considered to be something the user is actively aware is running. Foreground services is intended for music playback and downloads that the user knows are happening but is running in the background. Many apps do not follow this and simply user foreground services to avoid that their service gets killed by the system.

3.9 Inertial Navigation Systems

In order to know if the inertial sensors in todays smartphones are precise enough to get a movement pattern of the phone while walking around the house, they can be compared with the instruments that are already in use on planes and ships. Here the dead-reckoning technique is used in inertial navigation systems. These use accelerometers and gyroscopes in an inertial measurement unit (IMU) to calculate the current position based on an initially known position.

Inertial systems can be broken down into a number of categories based on their accuracy. These categories can be seen on Table 3.1, systems in each categories can be used for different things based on their accuracy.

	Position Performance	Gyroscope Inaccuracy	Accelerometer Inaccuracy
Military Grade	1 nmi / 24 h	<0.005° / h	<30 µg
Navigation Grade	1 nmi / h	0.01° / h	50 µg
Tactical Grade	>10 nmi / h	1° / h	1 mg
Commercial Grade	N/A	>10° / h	>1 mg

Table 3.1: Types of inertial navigation systmes.[24, 32]

The military and navigational grade are systems that are used for navigation in ships, submarines, and aircrafts in cases where the GPS signal is lost. These systems can replace the GPS system completely for longer periods of time. The tactical grade systems are used for the artificial horizon in aircrafts and aircraft autopilots, they often have integration with the GPS system which is used to correct inaccuracies in the sensors. The commercial grade inertial systems are typically used in smartphones, video game controllers, for camera stabilization, and electronic stability control in cars. These systems cannot be used for navigation using only inertial sensors.[32]

Tactical grade inertial systems can only be used for navigation for very short periods of time, as their inaccuracy in determining are more than 10 nautical miles per hour, or 18.52 kilometers per hour. This results in an inaccuracy per minute of just over 300 meters.

An example tactical grade inertial navigation system is the Systron SDN500, which can be seen on Figure 3.13. This system is integrated with GPS and weighs around 700 grams. It promises a Spherical Error Probable (SEP)[47] Position error of 340 meters over 15 minutes, meaning that the true position is with a 50% probability within a sphere with a radius of 340 meters.

Figure 3.13: Systron SDN500 tactical grade INS.

3.9.1 Calculating Velocity

In order to create a vector for the movement of the device, the velocity for each direction has to be calculated. In order to get the velocity for an axis, the acceleration is integrated, the calculation for the velocity is:

$$v(t) = \int_{t=0}^{t} adt$$

Where $v(t)$ is the velocity at time t and adt is the acceleration over the time difference from the previous data point. The velocity calculation has to be done for each axis separately.

3.9.2 Position Calculation

Once the velocity of the phone has been calculated, the change in position from the original position can be calculated. It is done by once again integrating the velocity data like so:

$$x(t) = \int_{t=0}^{t} vdt$$

Where $x(t)$ is the distance moved at time t and v is the velocity. The integration of the velocity data which itself is the integrated acceleration data, is a double integration. Like the velocity this calculation has to be done for each of the three axes. The total distance moved at time t can then be calculated by combining the distance moved in each axis.[30]

Because the position is calculated by the integration of already integrated data, the noise of the original data increases the inaccuracy of the sensors. In addition, the angle of the gravity that is removed from the accelerometer data in order to get the linear acceleration quite significantly influences the calculated position if it is off by just a little bit.

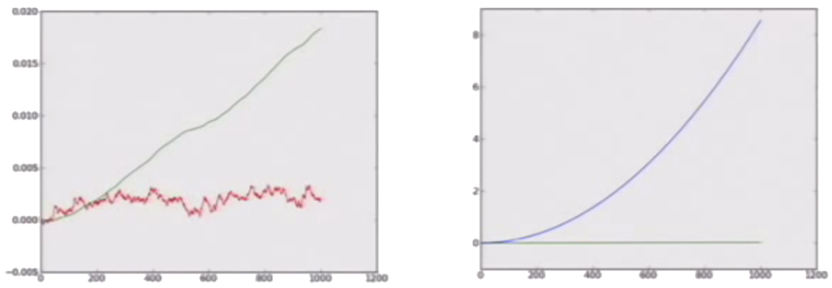
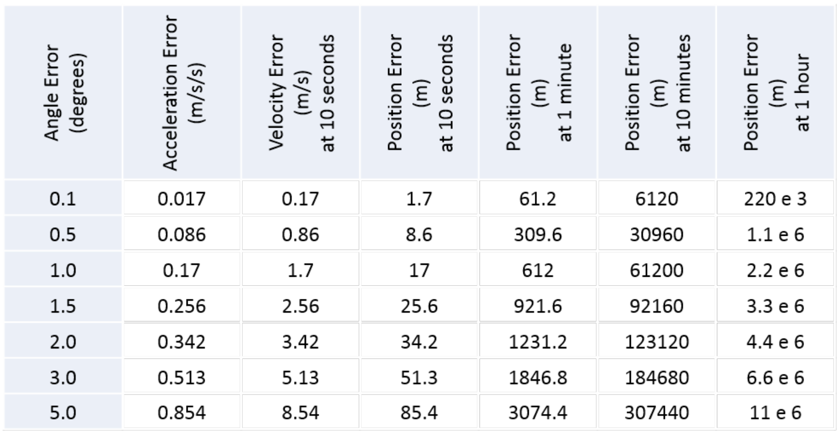


Figure 3.14: Left: Velocity in red and position change in green, Right: Same green position change and position with one degree angle bias in blue.[30]

Figure 3.14 shows on the left the calculated velocity and position of a phone that has been held in the hand. Over a period of one second the calculated position is off by 20 centimeters, this is without any angle bias. On the right is the same calculated position, but here a new line has been introduced, this is the same calculated position with one degree of angular bias. When angular bias is added, the calculation becomes a parabola and the new calculated position has an error of 8.5 meters over one second.



Angle Error (degrees)	Acceleration Error (m/s/s)	Velocity Error (m/s) at 10 seconds	Position Error (m) at 10 seconds	Position Error (m) at 1 minute	Position Error (m) at 10 minutes	Position Error (m) at 1 hour
0.1	0.017	0.17	1.7	61.2	6120	220 e 3
0.5	0.086	0.86	8.6	309.6	30960	1.1 e 6
1.0	0.17	1.7	17	612	61200	2.2 e 6
1.5	0.256	2.56	25.6	921.6	92160	3.3 e 6
2.0	0.342	3.42	34.2	1231.2	123120	4.4 e 6
3.0	0.513	5.13	51.3	1846.8	184680	6.6 e 6
5.0	0.854	8.54	85.4	3074.4	307440	11 e 6

Figure 3.15: Calculated position error based on incorrect angle assumption.[14]

Figure 3.15 shows a table of calculated position and velocity errors at different angles of gravity bias. Already at an incorrect angle assumption of 0.1°, a calculated position will be off by 1.7 meters after 10 seconds given an otherwise perfect signal, once noise from the sensors is included the error will be larger. From this we can gather that while it is theoretically possible to use the accelerometer to estimate velocity and position of a phone, the sensors used in todays phones will need to have minimal noise and drift otherwise the measurements will result in very poor accuracy estimates that are not useful.[14]

3.9.3 Rotating Data to World Coordinates

The accelerometer data is oriented as specified in the Android API, seen on Figure 3.12, which means that a phone will output movement data differently depending on how it is oriented. This makes sense for purposes where the data is used to orient content shown on the screen of a phone, for tracking a users movement it means that the data

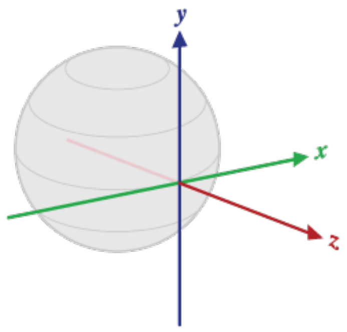


Figure 3.16: Data rotated to world coordinates have their coordinate system defined with the y-axis pointing north, the x-axis pointing east, and the z-axis pointing up.[19]

captured will be incompatible with old data if the phone is not carried the same way in the pocket. In order to combat this, the data can be rotated to use world coordinates instead of device specific coordinates with the help of the magnetic field sensor and the gyroscope.

Android provides functionality that can aid the rotation in form of a rotation vector sensor which can be converted into a rotation matrix that the accelerometer data can be multiplied onto. When the data is rotated, the accelerometer will be rotated to world coordinates as shown on Figure 3.16. The positive x-axis is east, the positive y-axis is north, and the positive z-axis is up, away from the earth.

3.9.4 Filtering

Filtering techniques are sometimes used to improve the noise from inaccurate sensors. The most used filtering techniques are the simple high-pass filter and the more effective Kalman filtering.

A high-pass filter is a simple filter that will remove any measurements under a specified threshold. It is used to filter the noise from the sensor while it is stationary, but will also affect measurements under the threshold that are correct. The high-pass filter works best when the sensors have a high precision where the threshold can be as small as possible.

The Kalman filter is used to filter outliers in otherwise accurate measurements as it can filter out single outliers in otherwise good data. It works by recursively predicting states in a system, and finds optimal estimates from noisy data. The Kalman filter can be used in real-time because new measurements can be processed as they arrive and added onto the previously calculated state. If all the noise produced by sensors is Gaussian, the Kalman filter will minimize errors.[25, 39]

3.10 Android Security

Android provides a number of security features limits applications and third parties from accessing and reading sensitive data. Some of these are provided by the Linux kernel which Android runs on top of.

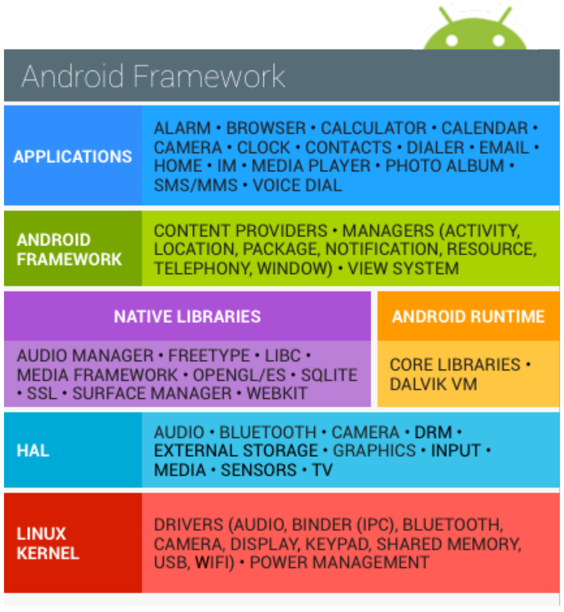

Android Framework
APPLICATIONS
ALARM • BROWSER • CALCULATOR • CALENDAR • CAMERA • CLOCK • CONTACTS • DIALER • EMAIL • HOME • IM • MEDIA PLAYER • PHOTO ALBUM • SMS/MMS • VOICE DIAL
ANDROID FRAMEWORK
CONTENT PROVIDERS • MANAGERS (ACTIVITY, LOCATION, PACKAGE, NOTIFICATION, RESOURCE, TELEPHONY, WINDOW) • VIEW SYSTEM
NATIVE LIBRARIES
AUDIO MANAGER • FREETYPE • LIBC • MEDIA FRAMEWORK • OPENGL/ES • SQLITE • SSL • SURFACE MANAGER • WEBKIT
ANDROID RUNTIME
CORE LIBRARIES • DALVIK VM
HAL
AUDIO • BLUETOOTH • CAMERA • DRM • EXTERNAL STORAGE • GRAPHICS • INPUT • MEDIA • SENSORS • TV
LINUX KERNEL
DRIVERS (AUDIO, BINDER (IPC), BLUETOOTH, CAMERA, DISPLAY, KEYPAD, SHARED MEMORY, USB, WIFI) • POWER MANAGEMENT

Figure 3.17: The Android software stack.

3.10.1 Application Sandbox

Any application running on Android is running in a sandbox. A sandbox is a means of protecting applications and their resources from other applications running on the device. This is done by running each application with their own unique user id, and restricting access to their data to users with that user id.

As Figure 3.17 shows, there are a large amount of other things running on top of the Linux kernel that make up Android. Many of these things are needed by applications in order to provide functionality for Bluetooth, Wifi, and more. All of the software running on top of the Linux kernel is likewise sandboxed.

The application sandbox means that in order to gain access to files created by other applications, an application would either need to break out of the sandbox or run as the root user. Breaking out of the sandbox is unlikely but possible. As the sandbox is controlled by the Linux kernel, and the Linux kernel is both open and widely used, any new vulnerabilities will likely be patched quickly.

Running an application as the root user is more likely, mostly because of the phenomenon called rooting. Many users modify their Android phones to make it possible for them to grant applications access to the root user. This means that anything stored on the phone is freely available to read and modify by an application with root access. As such it is not possible to rely on the sandboxing alone to prevent unauthorized access to saved data on rooted phones.

3.10.2 Physical Device Access

Limiting access to parts of the file system for applications make no security difference if the intruder has physical access to the device. For these scenarios Android provides the ability to encrypt all user data on the device, a so called Full Disk Encryption

(FDE). The encryption algorithm used is AES128 with Cipher Block Chaining (CBC), the encryption is done by the Linux Kernel subsystem ‘dm-crypt’. The password for the FDE is a 128-bit master key, which is generated when the device first boots. The master key is composed of the hash of a password with a stored salt.ăThe master key is then encrypted by a Trusted Execution Environment (TEE) that has unique keys for each device[22]. The ARM implementation of the TEE is called the TrustZone[4].

By default the password used for the generation of the master key is ‘default_password’, as the master key is encrypted with a device specific key, the data is secure to brute-force attacks in cases where it is copied off the device. When the user sets a pin or password on the device, the master key is re-encrypted such that the pin or password is needed to decrypt it. The data stored on the device is thus not re-encrypted, only the master key.

In order for any of this to create securely encrypted data, the TEE must be secure and unavailable for third-parties, which it unfortunately is not. One of the largest manufacturers of System on Chips (SoC) in Android devices, Qualcomm, has multiple times had their implementation of the ARM TrustZone compromised[7, 10]. In these cases, the supposedly secure keys stored in the TrustZone have successfully been extracted. With the keys extracted from the TrustZone, the FDE in Android is quickly brute-forced.

Even if the FDE was fully secure, encrypting the filesystem will only prevent third parties from accessing the stored data in case they get physical access to the phone, it will not hinder any applications running on the phone from accessing everything as any data will be decrypted once accessed. Thus, any private and sensitive data generated by the application should be stored in an encrypted format on the device.

3.11 Biometric Security

A possible security measure for preventing unauthorized persons from using the automatic unlocking on a stolen phone, is to make use of biometrics of the person carrying the phone. The types of biometric systems that are interesting for this use case are quite limited. As the phone is likely to be carried in a pocket, face or iris recognition should not be used. Likewise, fingerprint recognition would require the user to touch the phone whenever they wanted to unlock their door, making an automatic unlocking meaningless.

However, the use of gait recognition may provide improved security without requiring the user to do any conscious interaction with the device.

3.11.1 Gait Recognition

The walking style, or gait, of people is quite unique. So much so that it can be used to identify a person from video of their movement, but also likely from the use of the accelerometer and gyroscope that is present in most smartphones [48].

The gait of people varies because a number of factors influence it. Some of these factors are: The length of a persons legs and stride, the speed they move, the angle of feet and hips, and the weight of the person. A number of external forces also influence peoples gait, some of these are the footwear worn by the person and the surface they are walking on. Wearing different shoes makes a persons gait so different that gait recognition algorithms have a significantly higher error rate when the reference has been captured with a different pair of shoes than the probe data [27].

The gait can be used to correctly identify the person with a 95% accuracy [48] by using a Nintendo Wii Remote as accelerometer and the Nintendo Wii Motion Plus attachment as a gyroscope, the accuracy of which is in the range of the sensors in current smartphones. For the identification to work, the sensors ideally will need to be placed in the pants pocket of the person, and must at least be carried on the body.

For this purpose, the requirement for needing to carry the device on the body is not a hinderance, as many people will be carrying their phone on their body at all times, the issue with recognizing people because they change their shoes is a much larger concern. If the automatic unlocking does not work because the user has changed shoes, the user is likely to be irritated or switch the gait recognition off if it is possible.

Before gait recognition can be used to verify the person carrying the phone before an unlocking attempt is carried out, there are a number of obstacles that must be overcome. Firstly the gait recognition must work independently of the shoes that the user is wearing, next it should work even if the reference data is taken while the phone was carried in the pocket, and the probe was carried differently. Lastly the reference data should be taken from the immediate vicinity of the lock, as the ground will be similar when an unlocking attempt is made.

3.12 Location Spoofing

Before unlocking a door automatically, the application should ensure that the phone is at the location it expects the door to be at. If the location were to be spoofed, it would be possible to forward the signal from the lock through the internet and create a connection that could unlock the door remotely, while still acting as a valid unlock from the owner. This would enable unauthorized third-parties to enter a building without leaving any evidence of a break-in.

3.12.1 Mock Locations

Android has a number of settings that exist in order to help developers create applications easily. One of these settings is the support for mock locations. Mock locations enable a third party application to override the location information that is gathered from network and GPS location, and instead input its own location coordinates and update them as needed. When mock locations are used, there should not be made any attempt to unlock a door automatically. Luckily, it is possible for an application to ask the Android system if mock locations are used, allowing it to act accordingly. Ideally, the application should also notify the user if it is not working because mock locations are turned on.

3.12.2 GPS Spoofing

Another, more difficult, threat is the act of GPS spoofing. The GPS system relies on satellite signals that are used to calculate the location. The signals sent from these satellites are not encrypted for civilian use, making it possible for an attacker to create their own signals that look authentic.

In order to prevent these attacks, we must understand how it can be done, and the likeliness of an attack. In order to spoof a GPS signal, an attacker can set up an antenna which sends out a fake GPS signal. If the attacker can ensure that they have full control over the received signal, for example by blocking all legitimate GPS signals, the spoofing antennas can send signals that can spoof a single location [53]. This however, requires that the attackers antenna is close to the victim in order to ensure that no legitimate GPS signals are received.

In cases where it is not feasible to completely block legitimate GPS signals, it is possible to influence the location calculated by a targeted device. By comparing the GPS location with the network location it is possible to detect discrepancies if the two locations are very far apart.

3.13 Battery Saving Techniques

In order to have the best possible battery life, Android phones will apply a number of battery saving techniques. The first of which is that the phone will enter a low power state that puts the CPU into a deep sleep state where no calculations are performed. The CPU will wake up if needed, but will spend most of the time where the screen is off in this state. Sensors on the phone will often also either lower their output of measurements or completely turn off, this means that applications using these sensors will stop receiving data when the screen is off unless they explicitly request updates from the specific sensors.

Turning off sensors when the screen is off is handled by the Android PowerManager, which is where applications will need to request what is called a wake lock if they need sensors to stay active. The Android developer documentation states that the use of wake locks will affect the battery life of a device significantly, and requests that wake locks are not acquired unless they are actually needed and that they should be released as soon as possible.

Because of the effect that collecting data from the accelerometer, Bluetooth, Wifi, and location will have on the battery, it should only be collected if it is actually needed. This means that the data collection should ideally not be running for most of the time, allowing the smartphone to save as much power as possible.

Chapter 4

Problem Analysis

The requirements for the prototype have in part been determined from the analysis and in part been requested from Rasmus for what he needed for the machine learning part of the project.

The prototype should have the ability to:

- Collect data from Bluetooth, Wifi, GPS, network location, and inertial sensors on an Android smartphone. The inertial sensors should output data rotated to the world coordinates
- Start and stop collection of data from each sensor type should be possible independently of other sensors.
- Collect relevant data and automatically start/stop the collection based on the physical location of the device by using geofences.
- Store important data about paired locks such as location and lock MAC address in a database, along with data about nearby Wifi and Bluetooth devices.
- Store recently recorded data temporarily in a data buffer for comparison with stored data from the database.
- Automatically make a decision about whether a nearby lock should be unlocked based on collected data and predefined heuristics.

Chapter 5

Design

The application design tries to fulfill the requirements specified in the problem analysis. In order to do this, a number of data collection services will be created and used in combination with geofences and a database that can be used to store the data that can be used to make a decision for unlocking a door that is known by the application.

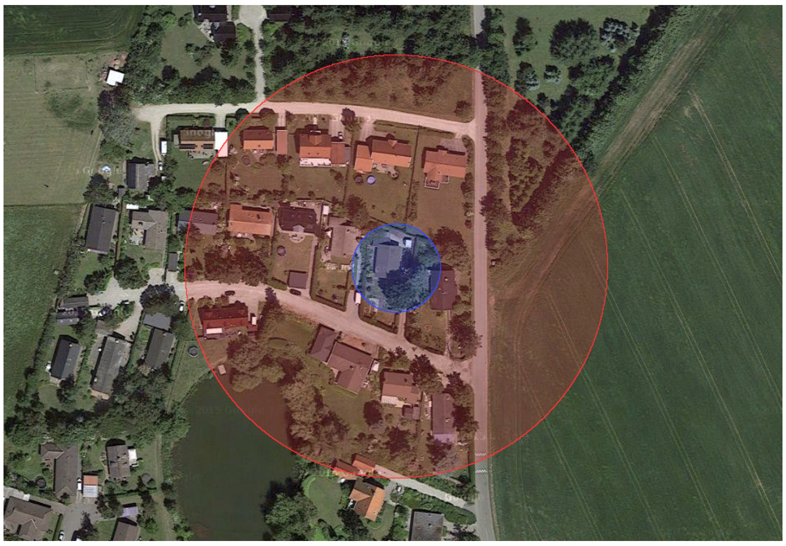


Figure 5.1: The inner and outer geofence zones around a house.

The general idea is to make use of geofences to control the collection of data such that is only collected if it is expected to be needed shortly. This saves battery by letting the application be idle for most of the time. The focus of this prototype is to be able to make a correct unlocking decision when the user is coming home from a period of being away from the house. This means that no unlocking decision will be made if the user is moving around inside the house or garden.

Figure 5.1 shows two geofences with a center in the front door of a house. The general idea behind this is to have an inner and an outer geofence for each known lock. When the smartphone is in a location outside of the outer geofence, the application will be idle. Once the outer geofence is triggered, the application will start collecting the location from GPS and the network location provider. When the inner geofences is triggered, the application will start collection of additional data. This data comes from Bluetooth, Wifi, accelerometer, gyroscope, and magnetic field sensor. Whenever a decision has been made to unlock the door, the data collection will stop and the application will go to the idle state. No new data will be collected until the smartphone exits the inner geofence, at which point the collection of location data will start again.

Making a decision about unlocking a door cannot be done the first time the door is encountered. The application will need to encounter the door a few times before enough information has been collected. The steps before the application can start to make a decision are as follows:

- The user manually unlocks the door. At this point, the application will record the location of the lock, nearby Wifi and Bluetooth devices, and set up initial geofences.
- The second encounter of the door, after the smartphone has left both geofences, the user will need to stand still in front of the door for two seconds before unlocking it. Here, the application will record the direction of approach to the door, and the orientation of the phone with the magnetic field sensor.
- Following this, the application will attempt to make a decision but will not send the unlock signal. Instead the user will be notified and their response will help tune parameters such as geofence sizes, signal strength of nearby Wifi and Bluetooth devices, orientation, direction of approach, and average time in the inner geofence before the unlocking should happen.

When the application unlocking a door automatically, it will continue to update the data used to make the decision if suitably large changes happen.

5.1 Domain Analysis

From the domain analysis, a domain model has been created. This domain model describes the overall structure for the door unlocking application, and will be the base for the class diagram to be created.

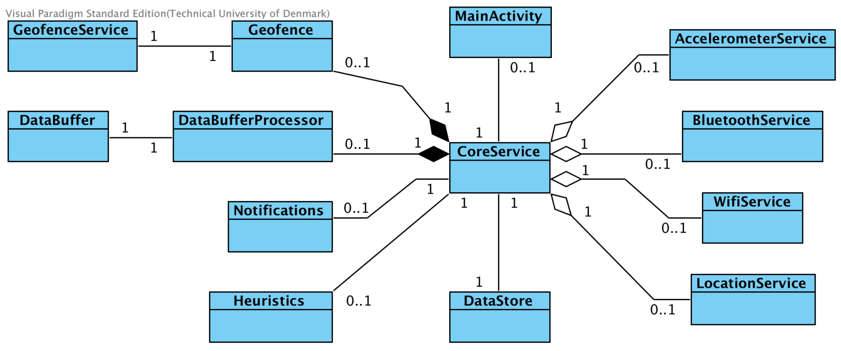


Figure 5.2: Domain model for the door unlocking application.

The domain model seen on Figure 5.2 shows the overall design of the unlock application.

The user interface will be contained in the `MainActivity` class, and will control the users interaction with the application. As soon as the user interface is started the first time, it will start the `CoreService`, which will be responsible for controlling collection of data, starting the decision process for a potential unlock of a known door, as well as unlocking the door if the decision to do so is made. The `CoreService` will always be running in the background, independent of the other services and the user interface.

The four services seen on the right of the domain model in Figure 5.2 are the services responsible for collecting relevant data for the `CoreService`. These four services will handle the collection of the following:

- `AccelerometerService` will handle the collection of accelerometer, gyroscope, and magnetic field sensor data and rotate it to a fixed coordinate system with north at the magnetic north.
- `BluetoothService` will handle discovery of nearby Bluetooth devices.
- `WifiService` will handle the discovery of nearby Wifi access points.
- `LocationService` will get the location of the device from GPS satellites or network based locationing if GPS is not available.

The data collected from these four services will be saved to a buffer that the `Heuristics` class can use to make a decision whether to unlock a nearby known door or not. Data used to train the `Heuristics` will be saved to the `DataStore` such that it can be used in the decision process later.

Finally the starting and stopping of the data collection will be governed in part by the `Geofence` class, which will notify the `CoreService` when the device is close to a known door.

5.2 Activities

A number of activities are needed for the application to make correct decisions and be usable. This section describes the identified activities.

5.2.1 Unlocking Activity

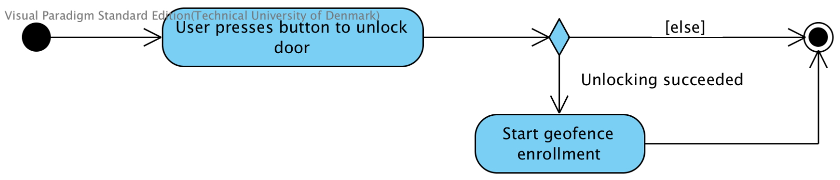

Visual Paradigm Standard Edition(Technical University of Denmark)
User presses button to unlock door
[else]
Unlocking succeeded
Start geofence enrollment

Figure 5.3: Activity diagram for unlocking without existing trained data.

The activity for unlocking a door is different, and depends on existing knowledge of the lock in the form of stored data. If no stored data exists, then the application will not do any data collection in the background automatically. Instead it will wait for the user to do a manual unlock of a door, at which point the activity of Figure 5.3 will be carried out.

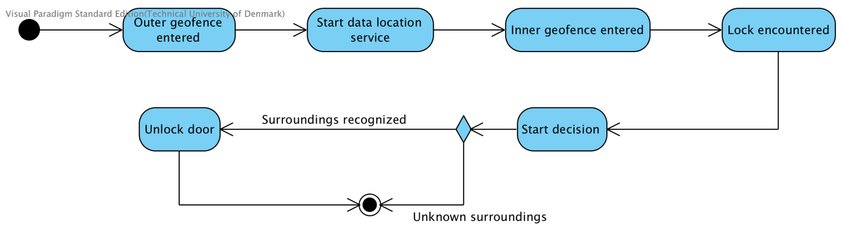


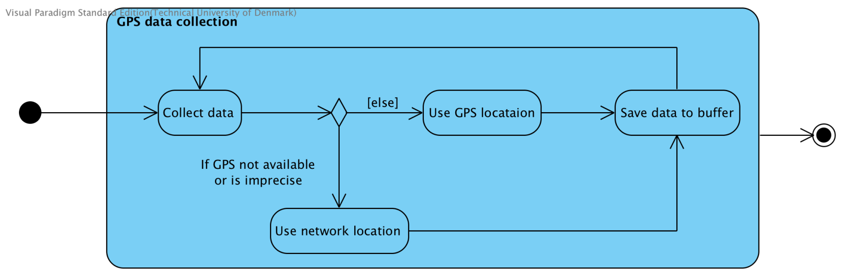
Figure 5.4: Activity diagram for unlocking with existing trained data.

At this point, the application will get the location of the device, which will be in the immediate vicinity of the door, then it will register a geofence.

The unlocking activity will start once the outer geofence is penetrated, and the device is located inside of it. Figure 5.4 shows the unlocking activity when it is started by entering the geofence. The data collection for the location service is immediately started and continues until a paired lock is encountered, the remaining data collection services are started when the inner geofence is entered. The earlier start of the location service served two purposes, firstly the GPS is likely to need time to be accurate and the active GPS signal makes positioning more accurate for when the inner geofence is entered. Once the lock is encountered, the activity starts deciding whether to unlock or not. If it recognizes the surroundings, such as nearby Wifi and Bluetooth devices, and has entered the inner geofence, the door unlock command will be sent. Otherwise the decision process and data collection will stop.

5.2.2 Data Collection Activities

Data collection from different sensors is similar but not the same. These activities consist of the Wifi, Bluetooth, network location, GPS, and accelerometer data collection activities. The similarity between these data collection activity is that they continuously collect the data until they are stopped.

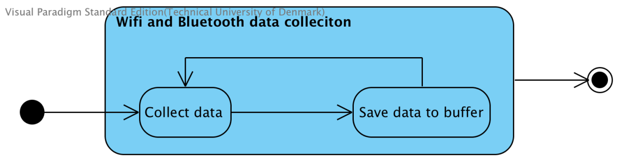


Visual Paradigm Standard Edition(Technical University of Denmark)
GPS data collection
Collect data
[else]
Use GPS locataion
Save data to buffer
If GPS not available or is imprecise
Use network location

Figure 5.5: Activity diagram for location data collection.

The location service, seen on Figure 5.5, will receive data from both the GPS and the network location service. The data from the two sources will be compared, in cases where the GPS data is unavailable, significantly less precise or outdated, the data from the network location will be saved to the data buffer. In any other case, the GPS data will be saved to the data buffer.

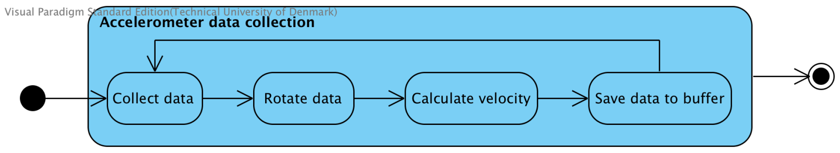
Data from the GPS is generally more precise and consistent and is therefore preferred. However, because it takes time to get the initial fix and because GPS often does not work indoors, the network location will be used as fallback location data.



Visual Paradigm Standard Edition(Technical University of Denmark)
Wifi and Bluetooth data colleciton
Collect data
Save data to buffer

Figure 5.6: Activity diagram for Wifi and Bluetooth data collection.

The data collection activities for both Wifi and Bluetooth are quite straight forward and identical. The data collected from nearby Wifi and Bluetooth devices does not need to be processed. The activity diagram for these two activities can be seen on Figure 5.6.



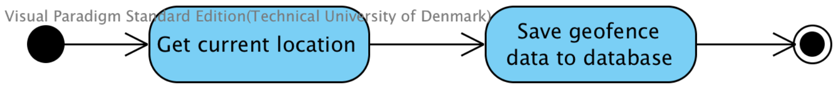
Visual Paradigm Standard Edition(Technical University of Denmark)
Accelerometer data collection
Collect data
Rotate data
Calculate velocity
Save data to buffer

Figure 5.7: Activity diagram for accelerometer data collection.

The accelerometer data collection activity on Figure 5.7 needs to process the data before saving it to the data buffer. Specifically, the data needs to be rotated from the device specific coordinates to geographic coordinates as specified in **??**.

5.2.3 Geofence Activity

The purpose of the geofence activity is to register and unregister geofences with the Google Play Services, which is used to inform the controller activity when to start and stop data collection. Before registering a geofence, the location of the associated lock is needed.



Visual Paradigm Standard Edition(Technical University of Denmark)
Get current location
Save geofence data to database

Figure 5.8: Activity diagram for adding new geofences.

Figure 5.8 shows the activity for creating a new geofence. The new geofence activity will be called when a successful unlock has been performed on a previously unknown lock without trained data. When creating a new geofence, the application will get the current location of the device which will be tied to the specific nearby locks MAC address. Once the location of the device is known, the activity will save the new geofence data in the database.

Once a new geofence has been created, it will need to be registered with the Google Play Services, which will inform the application when the device enters or exits a registered geofence. This will be done immediately after creating a new geofence, and for all geofences when the application is restarted. Figure 5.9 shows the register geofence activity. If a geofence is unneeded, or needs to be updated, it will have to be unregistered from the Google Play Services before it is deleted or updated.

Lastly, a geofence can be updated, as seen on Figure 5.10. The update can be a change of location, but it can also be a change of the radius for each of the two geofences associated with a lock. This radius is tuned by the heuristics based on user input.

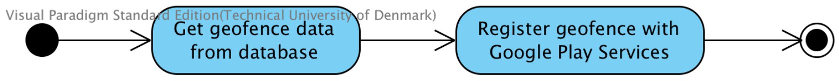

Visual Paradigm Standard Edition(Technical University of Denmark)
Get geofence data from database
Register geofence with Google Play Services

Figure 5.9: Activity diagram for registering geofences.

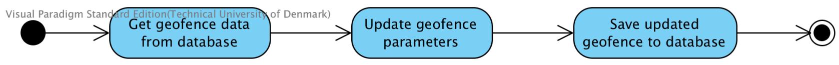

Visual Paradigm Standard Edition(Technical University of Denmark)
Get geofence data from database
Update geofence parameters
Save updated geofence to database

Figure 5.10: Activity diagram for updating geofences.

Geofence Service

Once the geofences have been registered with the Google Play Services, the geofence service will be started. As can be seen on Figure 5.11, the geofence service has the purpose of waiting for the Google Play Services to inform that a registered geofence has been entered or exited. When receiving the information, the geofence service will decide if the geofence in question was an inner or outer geofence, and will send an intent informing the core service about the change.

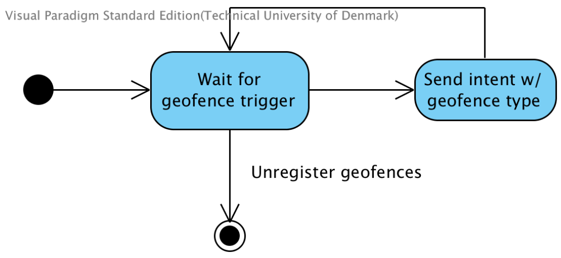

Visual Paradigm Standard Edition(Technical University of Denmark)
Wait for geofence trigger
Send intent w/ geofence type
Unregister geofences

Figure 5.11: Activity diagram for the geofence service.

5.2.4 Heuristics Activity

The heuristics activity has the purpose of deciding whether to unlock or not whenever a known lock is encountered. This is choice is based on a number of factors such as location, nearby Bluetooth and Wifi access points, and orientation of the smartphone. The heuristics activity also has the responsibility of tuning the parameters that is used for the decision. The tuning of these parameters is done based on user input that is presented whenever a decision has been made.

Figure 5.12 shows the heuristics activity. Whenever it is started, it first gets the stored decision data for the corresponding lock in the database. The stored data is then compared with current data collected by the smartphone sensors. The match can either be close, or not. If it is close enough then the decision to unlock will be made. The door will not be unlocked the first five times, instead the user will be notified to unlock. This is done to give the application time to learn the surrounding areas around the lock, after this, the application will start to unlock automatically.

In cases where the decision is to keep the door locked, the application will still provide a notification the first five times. This is done such that the user can guide the application, after that, the user will not be notified. After a decision has been made, the application will tune the heuristics based on the newly collected data. If there has been a change, the updated data will be stored in the database.

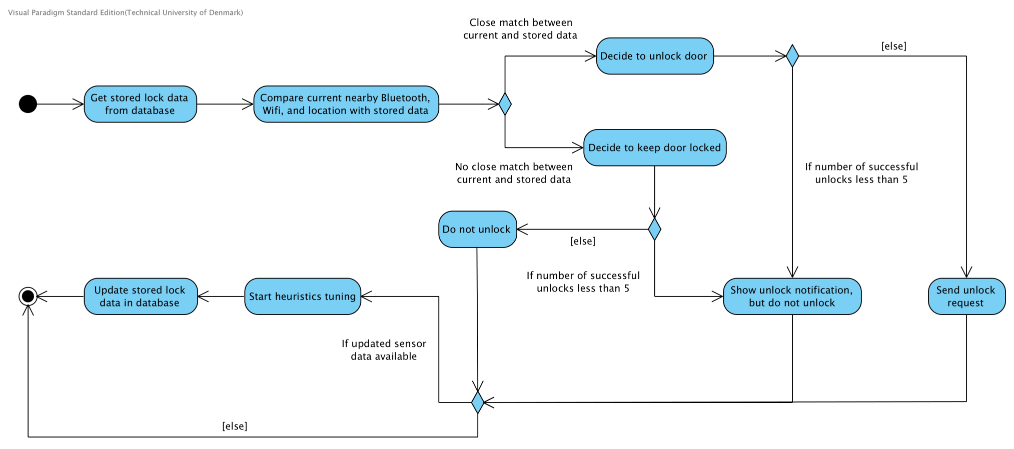

Visual Paradigm Standard Edition(Technical University of Denmark)
Get stored lock data from database
Compare current nearby Bluetooth, Wifi, and location with stored data
Close match between current and stored data
Decide to unlock door
[else]
No close match between current and stored data
Decide to keep door locked
If number of successful unlocks less than 5
Do not unlock
[else]
If number of successful unlocks less than 5
Show unlock notification, but do not unlock
Send unlock request
Update stored lock data in database
Start heuristics tuning
If updated sensor data available
[else]

Figure 5.12: Heuristics activity for decisions and tuning of unlocking parameters.

5.3 Use Cases

Here a number of use cases for the automatic unlocking interaction between a smartphone and a BeKey door lock have been identified. These use cases each describe scenarios where they are relevant.

Table 5.1 describes the use case where the user of a house or apartment arrives home after being away for an extended period of time. The reasons for being away are manyfold, and can include work, shopping, visiting friends, etc. There are two success scenarios for this use case, either the smartphone will recognize the door lock and surroundings and unlock, or it will recognize the door lock but not the surroundings and not unlock.

If Bluetooth is disabled, the lock can not be recognized. The Wifi being disabled, highly restricts the applications ability to recognize the surrounding Wifi access points. Blocked GPS and unavailable data connection will prevent the application from getting the devices location from both GPS and network. Without a location, the geofences are unable to function. The inner geofence is used to prevent that the application continuously tries to unlock the door while the smartphone is home, if the user is still at home they should not expect the door to unlock when walking up to it.

Table 5.2 shows the use case where the user manually unlocks the door lock. If the smartphone is already paired with the lock, the door will be unlocked. In the case where the two devices are not paired, the user will be presented with a request for the passphrase for the nearby lock. Once the user has provided the correct passphrase, the lock will be unlocked. If Bluetooth is disabled on the smartphone, or the smartphone is not in range, communication between lock and smartphone is not possible.

Table 5.3 shows the tuning of heuristics. The decision data is updated whenever the application successfully predicts the unlocking of a specific lock, or when an incorrect prediction is made that is corrected by the user. When an incorrect decision is made, the user is presented with a list of reasons explaining why the prediction was incorrect. The user will then choose reasons that fit, and the application will tune the heuristics based on this feedback. If the feedback from the user is incorrect, the tuning of the decision data will also be incorrect.

Use Case Name	Arriving home
Description	A user arrives home to their house after being away for a while. The paired BeKey door lock automatically unlocks the door when the user is nearby.
Actors	User, Smartphone, BeKey lock.
Preconditions	Smartphone has moved far enough away before returning to the vicinity of the lock.
Main Scenario	[Door unlocks as user arrives home] 1. User arrives home 2. Smartphone recognizes door lock and surroundings 3. Smartphone decides to unlock the door
Alternate Scenarios	[Door remains closed because the surroundings are not recognized] 1. User arrives home 2. Smartphone recognizes door lock and but not the surroundings 3. Smartphone decides not to unlock the door [Failure points] - Bluetooth or Wifi is disabled on the smartphone - GPS is blocked and data connection is unavailable - User tries to enter door without having left the inner geofence, thus not triggering the unlock decision
Post conditions	A decision has been made by the smartphone to either unlock or not take any action.

Table 5.1: Use case: Arriving home

Use Case Name	Manual unlock
Description	A user is nearby a lock, they open the unlocking application and manually initiate the unlocking.
Actors	User, Smartphone, BeKey lock.
Preconditions	Smartphone is turned on.
Main Scenario	[Manual unlocking successful] 1. User initiates unlocking 2. Smartphone is already paired with lock 3. The door unlocks
Alternate Scenarios	[Smartphone and lock not paired] 1. User initiates unlocking 2. Smartphone and lock are not paired 3. Smartphone initiates pairing and requests the passcode for the lock 4. User inputs passcode and the two devices are paired 5. The door unlocks [Failure points] - Bluetooth is disabled on the smartphone - The smartphone is not in range of a BeKey lock
Post conditions	The door has been unlocked and can be opened.

Table 5.2: Use case: Manual unlock

Use Case Name	Tuning heuristics
Description	When an unlocking decision has been made, the user will be presented with a notification that asks if the unlocking was correct. If it was not, inquiries will be made for the reason and the data for the heuristics will be updated.
Actors	User, Smartphone.
Preconditions	An unlocking decision has been made.
Main Scenario	[Unlocking decision was correct] 1. User indicates that the unlocking was correct 2. Smartphone updates decision data if new data is available
Alternate Scenarios	[Unlocking decision was not correct] 1. User indicates that the unlocking decision was incorrect 2. User is presented with a list of reasons why the decision was incorrect 3. Trained data is updated based on user input [Failure points] - User provides incorrect data
Post conditions	The trained data is updated.

Table 5.3: Use case: Tuning heuristics

5.4 Class Design

The classes in Figure 5.13 each have their own responsibility that combined makes the application function successfully.

MainActivity

The `MainActivity` class is responsible for the user interface that is presented when the application is launched. It is also responsible for launching the `CoreService` if it is not already running. This means that buttons presented to the user will need to call a method in this class.

CoreService

The `CoreService` is intended to always be running, and handle the starting and stopping of other services that handle data collection and processing. The database is initialized in the `CoreService` class because it should always be readily available for any other class that needs to access or change stored data. The `CoreService` class does not itself process any data, but handles the starting and stopping of both the `DataBuffer`, `DataProcessorService`, and the `ScannerService` which handle the temporary storage and processing of newly collected data. The `CoreService` will also connect to the Google Play Services such that the needed geofences can be registered.

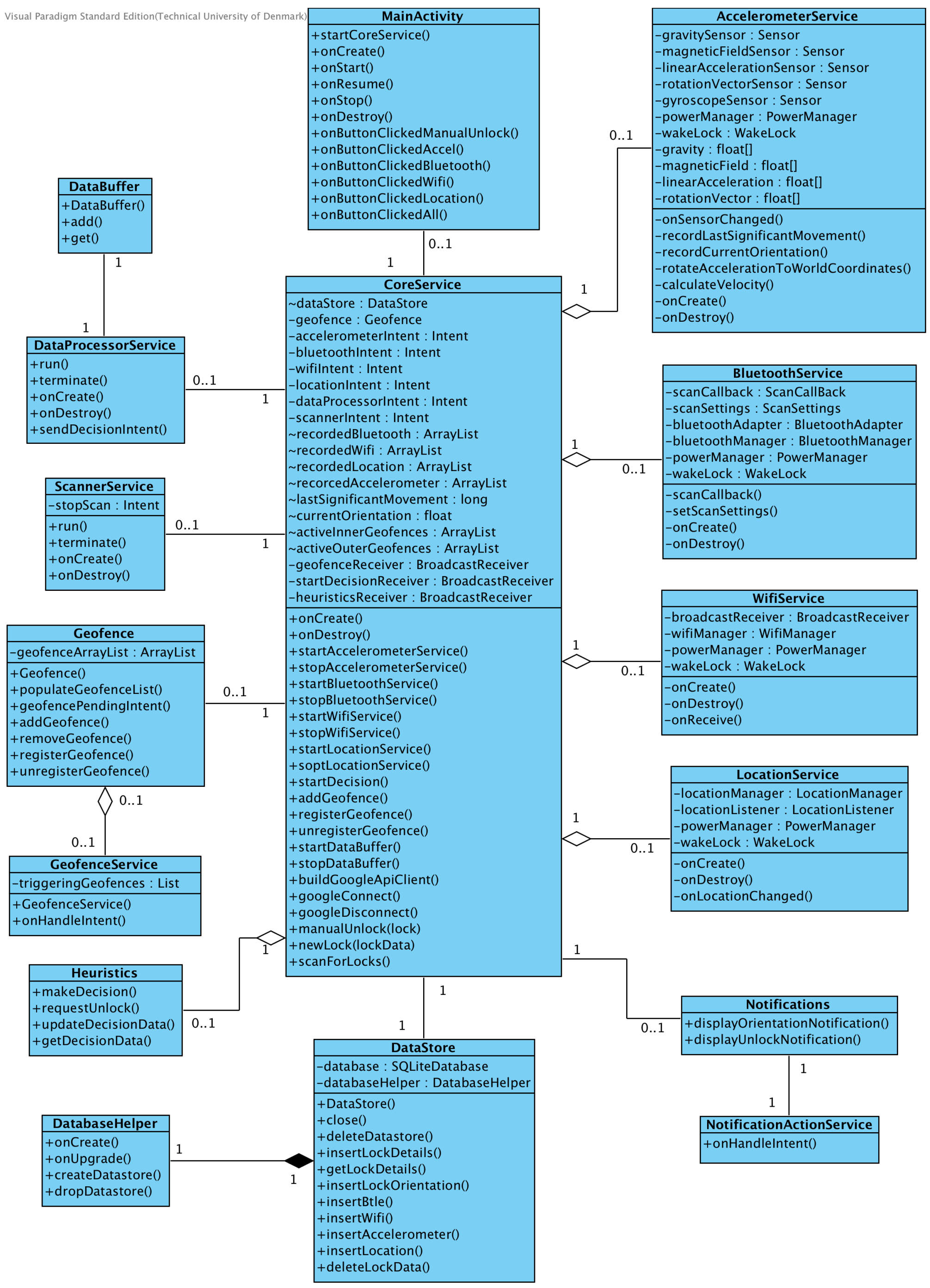


Figure 5.13: Class diagram for autounlock application.

Data Collection Services

The four data collection services, `AccelerometerService`, `BluetoothService`, `WifiService`, and `LocationService` are very similar in their design. They are kept separate to easily add or remove new services for collecting different data, to start and stop them independently, and so they each can process the collected data in their own way. Each service will be started in their own thread, such that they do not block the user interface or each other.

Geofence and GeofenceService

The `Geofence` class handles registering and unregistering geofences with the Google Play Services. The `GeofenceService` is started by the `Geofence` class, it is responsible for handling the responses that the Google Play Services returns when a geofence has been entered or exited.

DataStore

The `DataStore` class relies on the `DatabaseHelper` class to provide a SQLite database that stores decision data about known locks, and passcodes used for unlocking. The `DataStore` provides methods for inserting and updating data in the database such that no other class needs to handle SQLite directly. The methods will themselves handle correct SQLite creation and prevent that the database is accessed by multiple methods at the same time, preventing errors.

DataBuffer, DataProcessorService, and ScannerService

The `DataBuffer` class is the circular buffer that stores temporary data used by Rasmus' machine learning decision project. The data is inserted into the buffer by the `DataProcessorService`, which will also detect nearby known locks. The `ScannerService` is similar as it recognizes expected locks, but is used to aid the heuristics by starting the decision.

Heuristics

The `makeDecision()` method in the `Heuristics` class is called whenever a known lock is encountered. A decision is then taken based on the most recent recorded data and the stored data in the `DataStore`. Once a decision is made, the decision data will be updated based on the recent data and user input.

NotificationUtility

The `NotificationUtility` handles notifying the user about new and known locks. When a notification is displayed the user will be able to choose a number of options that are used to tune the heuristics data.

5.5 Behavior

The behavior of the classes defined in Section 5.4 is represented by state machines in this section.

Core Service

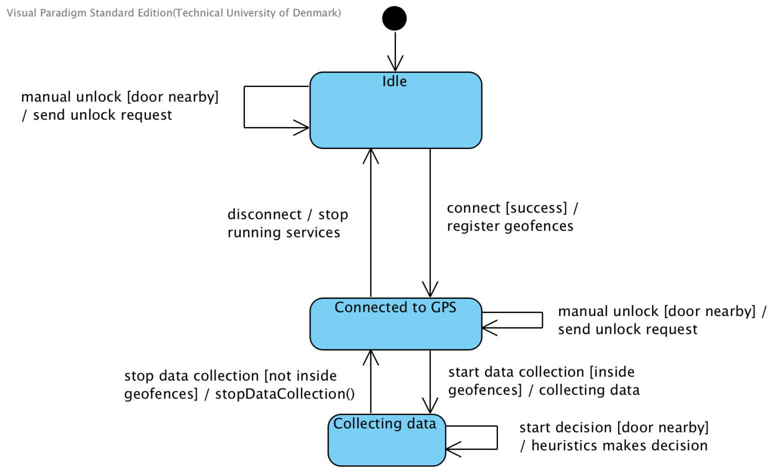

Visual Paradigm Standard Edition(Technical University of Denmark)
Idle
manual unlock [door nearby] / send unlock request
disconnect / stop running services
connect [success] / register geofences
Connected to GPS
manual unlock [door nearby] / send unlock request
stop data collection [not inside geofences] / stopDataCollection()
start data collection [inside geofences] / collecting data
Collecting data
start decision [door nearby] / heuristics makes decision

Figure 5.14: State machine for the CoreService class.

The core service is started in an idle state where it is only possible to trigger an unlock manually and initiate a connection to the Google Play Services. Once the connection to the Google Play Services is successful, the geofences of locks stored in the database will be registered, and the core service will enter a new state where it is waiting for geofences to be triggered. Once an outer geofence has been entered, the core service will start the data collection services and enter a new state. While collecting data, the core service will start the heuristics decision whenever a known lock is encountered and the location of the device is known. The state diagram can be seen in Figure 5.14.

Heuristics

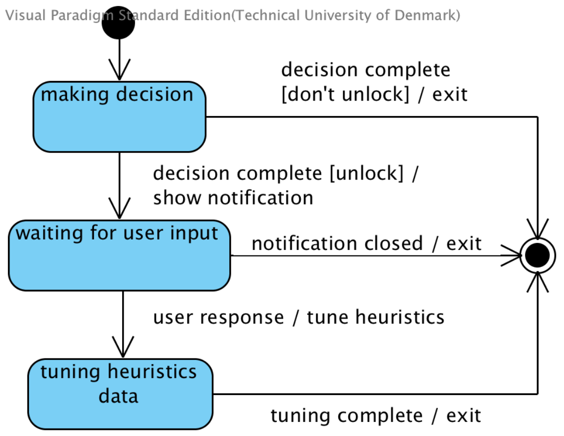

Visual Paradigm Standard Edition(Technical University of Denmark)
making decision
decision complete [don't unlock] / exit
decision complete [unlock] / show notification
waiting for user input
notification closed / exit
user response / tune heuristics
tuning heuristics data
tuning complete / exit

Figure 5.15: State machine for the Heuristics class.

The heuristics on Figure 5.15 will start in a decision making state. If a decision has been made to unlock, a notification will be shown to the user. Otherwise the heuristics will exit. The heuristics will temporarily save the data used to make the decision until the user has responded to the notification, at which point the data can be used to tune the heuristics. If the notification is closed without a response, or when the tuning is complete, the heuristics will have completed and stop.

Geofence

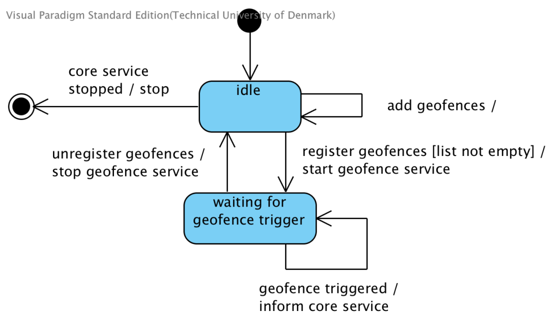

Visual Paradigm Standard Edition(Technical University of Denmark)
idle
core service stopped / stop
add geofences /
unregister geofences / stop geofence service
register geofences [list not empty] / start geofence service
waiting for geofence trigger
geofence triggered / inform core service

Figure 5.16: State machine for the Geofence class.

The geofence in Figure 5.16 will be started once the core service is created. When the core service is connected to the Google Play Services, geofences registered with known locks will be added and registered. At this point the geofence service will be started and wait for the registered geofences to be triggered. When a geofence has been triggered, the geofence service will inform the core service of the trigger type, associated lock, and whether it was the inner or outer geofence.

Data Collection Services

The behavior for the data collection services is very simple. Once they are started, they will be collecting data as often as possible and save this data to the core service. They will continue to do so until they are stopped, at which point the services will exit, data collection will stop, and the data buffer will be cleared.

5.6 Data Storage

Data is stored in two different ways, temporarily in a buffer and permanently in the database.

Database

The database is using the Android SQLite implementation for saving data. The data stored in the database can be seen on Figure 5.17.

The tables contained in the database are `LOCK_DATA`, `ACCELEROMETER_TABLE`, `WIFI-_TABLE`, and `BLUETOOTH_TABLE`. Data tied to a specific lock is stored in the `LOCK_DATA`

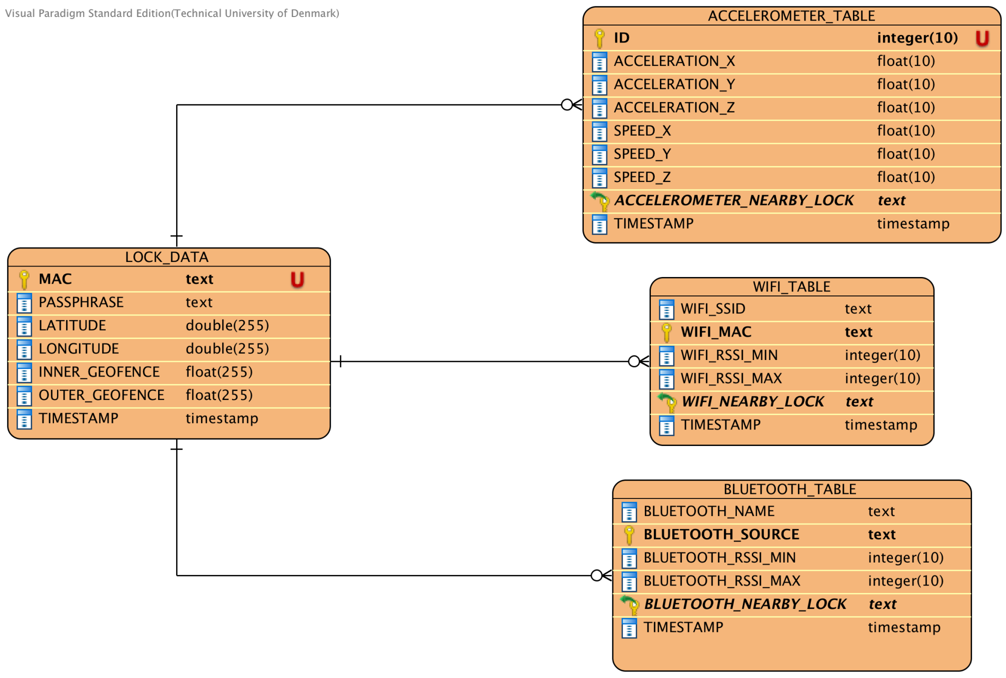


Figure 5.17: Database design for stored decision data.

table, this data includes the MAC address, which is unique, that is used to tie the data to a specific physical lock. Along with this is the passphrase needed to unlock the lock. The lock table also includes the location of the lock in form of longitude and latitude, as well as the inner and outer geofence size in meters. The location is the location of the lock as captured by the phone on the first unlock. Finally, there is a timestamp which includes the time of the most recent update to the row.

The `BLUETOOTH_TABLE` and `WIFI_TABLE` includes Bluetooth and Wifi devices. Both have a one-to-many relation with the `LOCK_DATA`, meaning that to each lock is tied multiple Wifi and Bluetooth devices. A Wifi device includes a SSID, MAC address, the minimum and maximum recorded RSSI where unlocking was approved, and a timestamp of the most recent change. A Bluetooth device consists of the same, but the SSID is called name, and the MAC is called source to adhere to the naming conventions of Bluetooth. Finally there is a foreign key that ties each entry to a lock MAC address.

The primary key for both Bluetooth and Wifi is the combination of the MAC/Source address of the device along with the MAC address of the nearby lock. This allows a single Bluetooth or Wifi device to be tied to multiple different locks and still have independent data for each. The idea is that multiple locks can be near the same Bluetooth or Wifi device by being close to each other physically. The device can also follow the user around, this is true for fitness bands or smart watches that will often be near the user and continuously be announcing their presence with Bluetooth.

The `ACCELEROMETER_TABLE` is different from the Bluetooth and Wifi tables because the accelerometer data is a series of recorded events. The accelerometer data is still tied to a lock with a foreign key to the MAC address, however, the accelerometer data does not have any unique identifier that can be used. Therefore the ID is a randomly generated integer that identifies a series of captured accelerometer events. The `ACCELEROMETER-_TABLE` also includes data in the x,y, and z axis that have been rotated to geographic coordinates along with the calculated speed at each event.

Data Buffer

The data buffer is a circular array that contains a list of four lists at each index. These four lists are the recorded accelerometer, Bluetooth, Wifi, and location data. The recorded data will be saved to the circular buffer every two seconds, and the lists of recorded data are cleared. In order to not put empty data into the data buffer, old data will be used if no new data has been found since the previous flush to the buffer.

Chapter 6

Implementation

6.1 Main Activity, UI, and Manifest

The basic building blocks of any Android application are the main activity, the user interface, and the Android manifest. The main activity is the class that is launched when the application is first opened. The user interface is defined in a number of XML files each corresponding to an activity, and the information about the application must be declared in the `AndroidManifest.xml` file.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.simonjensen.autounlock">
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
  <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
      />
  <uses-permission android:name="android.permission.BLUETOOTH"/>
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
  <uses-permission android:name="android.permission.WAKE_LOCK"/>

  <application android:allowBackup="true" android:icon="@mipmap/ic_launcher
      " android:label="@string/app_name" android:supportsRtl="true"
      android:theme="@style/AppTheme">
    <service android:name=".CoreService" android:enabled="true"/>
    <service android:name=".AccelerometerService" android:enabled="true"/>
    <service android:name=".LocationService" android:enabled="true"/>
    <service android:name=".WifiService" android:enabled="true"/>
    <service android:name=".BluetoothService" android:enabled="true"/>
    <service android:name=".GeofenceService" android:enabled="true"/>
    <service android:name=".DataProcessorService" android:enabled="true"/>
    <service android:name=".ScannerService" android:enabled="true"/>
    <service android:name=".NotificationUtility$NotificationActionService"
        android:enabled="true">
    </service>
  </application>
</manifest>
```

Listing 6.1: AndroidManifest.xml

Listing 6.1 shows the most important aspects of the `AndroidManifest.xml` file. Firstly, the permissions used in the application are defined. Following that, the application details are defined, these include the name, icon, and theme. Lastly, the services that the application provides are defined, these have to be defined here for Android to allow them to run.

The User Interface

Most of the applications work is meant to happen seamlessly in the background, without the user seeing it. The user interface is thus quite minimal, most of the buttons in the main activity are used to start and stop individual services and would not be present in an end product.

Launching the Core Service

Besides the user interface, the main activity has the responsibility of launching the core service. The core service is different from the other services managed by the application as it is meant to be running at all times, rather than continuously starting and stopping.

```java
protected void onStart() {
  super.onStart();

  if (isMyServiceRunning(CoreService.class)) {
      bindService(new Intent(this, CoreService.class), serviceConnection,
          Context.BIND_AUTO_CREATE);
  } else {
      ComponentName coreService = startService(new Intent(this, CoreService
          .class));
      bindService(new Intent(this, CoreService.class), serviceConnection,
          Context.BIND_AUTO_CREATE);
  }

  // Check for location permission on startup if not granted.
  if (ContextCompat.checkSelfPermission(this, Manifest.permission.
      ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
      ActivityCompat.requestPermissions(this, new String[]{Manifest.
          permission.ACCESS_FINE_LOCATION, Manifest.permission.
          WRITE_EXTERNAL_STORAGE}, 1);
  } else

  // Check for permission to write to external storage.
  if (ContextCompat.checkSelfPermission(this, Manifest.permission.
      WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
      ActivityCompat.requestPermissions(this, new String[]{Manifest.
          permission.WRITE_EXTERNAL_STORAGE, Manifest.permission.
          ACCESS_FINE_LOCATION}, 1);
  }
}
```

Listing 6.2: onStart() method in the MainActivity class

Listing 6.2 shows the `onStart()` method of the `MainActivity` class. Here the core service is started if it is not already running, if the service is already running, the activity is simply bound to the service. The `onStart()` method also checks for the needed application permissions, if the necessary permissions have not been granted, the user will be notified and the application will not function properly.

6.2 Core Service

When the core service is started, new instances of the `DataStore`, `Geofence`, as well as intents for the data collection classes are created. The service is started with a foreground notification to prevent being killed and filters for intents are created. Additionally the core service will connect to the `GoogleApiClient` such that it can be used to register geofences. This connection is done asynchronously by the `buildGoogleApiClient()` method in Listing 6.3, whenever the connection to the Google Play Service is established, the core service will start the registering geofences in the `Geofence` class.

```java
private synchronized void buildGoogleApiClient() {
    mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();
}
```

Listing 6.3: buildGoogleApiClient() in the CoreService class

As per the design, the core service handles the starting and stopping of the data collection and processing services. The decision for starting and stopping the data collection is based on the geofences that are attached to the locks that the application knows and can unlock automatically. Once inside the inner geofence, the phone will continue to record data until an unlocking decision has been made, at which point it will turn off the data collection services.

```java
private BroadcastReceiver geofencesReceiver = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
    Bundle extras = intent.getExtras();
    List<String> triggeringGeofencesList = extras.getStringArrayList("
        Geofences");

    if ("GEOFENCES_ENTERED".equals(action)) {
      for (String geofence : triggeringGeofencesList) {
        if (geofence.contains("inner")) {
          for (String outerGeofence : activeOuterGeofences) {
              if (outerGeofence.equals(geofence.substring(5))) {
                  activeOuterGeofences.remove(outerGeofence);
              }
          }
          activeInnerGeofences.add(geofence.substring(5));
          if (!isDetailedDataCollectionStarted) {
              Log.i(TAG, "onReceive: starting detailed data collection");
              isDetailedDataCollectionStarted = true;
              isScanningForLocks = true;
              startAccelerometerService();
              startBluetoothService();
              startWifiService();
              scanForLocks();
          }
        } else if (geofence.contains("outer")) {
          for (String innerGeofence : activeInnerGeofences) {
              if (innerGeofence.equals(geofence.substring(5))) {
                  activeInnerGeofences.remove(innerGeofence);
              }
          }
          activeOuterGeofences.add(geofence.substring(5));
          if (!isLocationDataCollectionStarted) {
              isLocationDataCollectionStarted = true;
              startLocationService();
          }
        }
      }
    } else if ("GEOFENCES_EXITED".equals(action)) {
      for (String geofence : triggeringGeofencesList) {
        if (geofence.contains("inner")) {
          if (activeInnerGeofences.contains(geofence.substring(5))) {
              activeInnerGeofences.remove(geofence.substring(5));
          }
          if (isDetailedDataCollectionStarted && activeInnerGeofences.
            isEmpty()) {
              isDetailedDataCollectionStarted = false;
              isScanningForLocks = false;
              stopAccelerometerService();
              stopBluetoothService();
              stopWifiService();
          }
        } else if (geofence.contains("outer")) {
          if (activeOuterGeofences.contains(geofence.substring(5))) {
              activeOuterGeofences.remove(geofence.substring(5));
          }
          if (isLocationDataCollectionStarted && activeOuterGeofences.
            isEmpty()) {
              isLocationDataCollectionStarted = false;
              stopLocationService();
          }
        }
      }
    }
  }
};
```

Listing 6.4: The BroadcastReceiver geofencesReceiver in the CoreService class

The core service registers a number of broadcast receivers that are called based on intents sent from other classes. The broadcast receiver `geofenceReceiver` in Listing 6.4 is the receiver for the intent sent by the `GeofenceService` when a geofence is entered or exited. Other registered broadcast receivers in the core service class are for when a decision is started, and when tuning of the heuristics are required.

Whenever a new lock is added to the application to be automatically unlocked, the application will collect data about the lock and the surrounding areas. This information is what will be used by the heuristics to start making decisions.

```
void manualUnlock(final String lockMAC) {
  new Thread(new Runnable() {
    public void run() {
      boolean success = true;
      String passphrase = "";

      startAccelerometerService();
      startBluetoothService();
      startWifiService();
      startLocationService();

      try {
        Thread.sleep(20000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }

      stopAccelerometerService();
      stopBluetoothService();
      stopWifiService();
      stopLocationService();

      if (success && recordedLocation.size() != 0) {
        LocationData currentLocation = recordedLocation.get(
            recordedLocation.size() - 1);

        LockData lockData = new LockData(
                lockMAC,
                passphrase,
                currentLocation,
                30,
                100,
                -1f,
                recordedBluetooth,
                recordedWifi
        );
        newLock(lockData);
      } else {
        Log.e(TAG, "No location found, cannot add lock");
      }
    }
  }).start();
}
```

Listing 6.5: manualUnlock() in the CoreService class

Listing 6.5 shows that when a new lock is added, the data collection services will run for 20 seconds. The information collected is added to a new lock entry in the database along with the passphrase for the lock and the default sizes for the two geofences.

6.3 Data Collection

The data collection services are similar in their construction. They all include an `onCreate()` and an `onDestroy()` method. The `onCreate()` method is the first method to be called when the service is started, and the `onDestroy()` method will be called just before the service is stopped.

```
public void onCreate() {
  powerManager = (PowerManager) getApplicationContext().getSystemService(
      Context.POWER_SERVICE);
  wakeLock = powerManager.newWakeLock(PowerManager.FULL_WAKE_LOCK, "
      BluetoothService");
  wakeLock.acquire();

  final BluetoothManager bluetoothManager = (BluetoothManager)
      getSystemService(Context.BLUETOOTH_SERVICE);
  bluetoothAdapter = bluetoothManager.getAdapter();

  // Enabling bluetooth if not enabled.
10    if (!bluetoothAdapter.isEnabled()) {
11      Log.v(TAG, "Bluetooth is off, turning on");
12      bluetoothAdapter.enable();
13
14      // Waiting for bluetooth adapter to turn on.
15      while (true) {
16        if (bluetoothAdapter.isEnabled()) {
17            break;
18        }
19      }
20    }
21
22    setScanSettings();
23    bluetoothAdapter.getBluetoothLeScanner().startScan(null, scanSettings,
          scanCallback);
24  }
```

Listing 6.6: onCreate() from the BluetoothService class

The `onCreate()` method in Listing 6.6 is from the Bluetooth service. On lines 2-4 the Android power manager is called and a wakelock is requested for the service. Without a wakelock, the power manager will stop the data collection once the screen is turned off in order to save power. Here we explicitly request that the Bluetooth service can continue running when the screen is off. Lines 7 and 8 get the Bluetooth adapter and lines 9-20 enables it if Bluetooth is turned off. Line 22 sets the Bluetooth scanning to its most aggressive mode, and line 23 starts the scanning of nearby Bluetooth devices.

```
public void onDestroy() {
    bluetoothAdapter.getBluetoothLeScanner().stopScan(scanCallback);
    wakeLock.release();
}
```

Listing 6.7: onDestroy() from the BluetoothService class

When the service is stopped, the Bluetooth scanning is stopped and the wakelock is released, allowing the smartphone to once again enter its power saving mode when the screen is off. All of the data collection services request a wakelock when they are running and release it when they are stopped, likewise they are starting their scan in their `onCreate()` method and stopping it in their `onDestroy()` method.

BluetoothService

Once the Bluetooth service is started, it calls the `setScanSettings()` method. This method defines the Bluetooth scan settings. As seen in Listing 6.8, the settings are different for devices running Android Marshmallow and devices running Android versions below that. This is because some of the settings used only were introduced in Marshmallow, and thus creates errors on lower versions of Android.

```
private void setScanSettings() {
  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
    scanSettings = new ScanSettings.Builder()
                .setScanMode(ScanSettings.SCAN_MODE_LOW_LATENCY)
                // The following require Marshmallow.
                .setCallbackType(ScanSettings.CALLBACK_TYPE_ALL_MATCHES)
                .setMatchMode(ScanSettings.MATCH_MODE_AGGRESSIVE)
                .setNumOfMatches(ScanSettings.MATCH_NUM_MAX_ADVERTISEMENT)
                .setReportDelay(0)
                .build();
  } else {
    scanSettings = new ScanSettings.Builder()
                .setScanMode(ScanSettings.SCAN_MODE_LOW_LATENCY)
                .setReportDelay(0)
                .build();
  }
}
```

Listing 6.8: setScanSettings() from the BluetoothService class

When the Bluetooth scanning is started, on line 23 in Listing 6.6 the `scanSettings` variable is passed along with the call. The other parameter that is passed along with the method call is the scanCallback object, which is where the results of the scan will be stored. Listing 6.9 shows the `scanCallback` which includes the method `onScanResult()` which is called when a Bluetooth LE advertisement has been found.

```
private ScanCallback scanCallback = new ScanCallback() {
  @Override
  public void onScanResult(int callbackType, ScanResult result) {
    super.onScanResult(callbackType, result);

    String name = result.getDevice().getName();
    String source = result.getDevice().getAddress();
    int RSSI = result.getRssi();
    long time = System.currentTimeMillis();
    ArrayList<BluetoothData> bluetoothDevicesToRemove = new ArrayList<>();

    // Bluetooth data for same lock and old data is deleted
    for (BluetoothData bluetooth : CoreService.recordedBluetooth) {
        if (time - bluetooth.getTime() > 5000) {
            bluetoothDevicesToRemove.add(bluetooth);
        } else if (bluetooth.getSource().equals(source)){
            bluetoothDevicesToRemove.add(bluetooth);
        }
    }
    CoreService.recordedBluetooth.removeAll(bluetoothDevicesToRemove);

    BluetoothData aBluetoothDevice;
    aBluetoothDevice = new BluetoothData(name, source, RSSI, time);
    CoreService.recordedBluetooth.add(aBluetoothDevice);
  }
};
```

Listing 6.9: scanCallback from the BluetoothService class

`onScanResult()` is called for each found Bluetooth device, and contains a `ScanResult` which includes information about the device. When a result is returned, first the relevant data is extracted, then it is stored in a BluetoothData object which is added to the `recordedBluetooth` array in the core service.

WifiService

The `WifiService` is quite similar to the `BluetoothService`, however, the results of a Wifi scan is returned to the `broadcastReceiver` seen in Listing 6.10, and the result is a list of all nearby Wifi access points rather than a single one.

```java
private final BroadcastReceiver broadcastReceiver = new BroadcastReceiver()
    {
  @Override
  public void onReceive(Context context, Intent intent) {
    if (intent.getAction().equals(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION
        )) {
      List<ScanResult> scanResults = wifiManager.getScanResults();
      long time = System.currentTimeMillis();

      CoreService.recordedWifi = new ArrayList<>();
      for (int i = 0; i < scanResults.size(); i++) {
        String SSID = scanResults.get(i).SSID;
        String MAC = scanResults.get(i).BSSID;
        int RSSI = scanResults.get(i).level;

        WifiData aWifiDevice = new WifiData(SSID, MAC, RSSI, time);
        CoreService.recordedWifi.add(aWifiDevice);
      }
    }

    // We force the device to scan for wifi again when we have finished the
        previous scan.
    wifiManager.startScan();
  }
};
```

Listing 6.10: broadcastReceiver from the WifiService class

When the `WifiManager` has completed its scan, the `onReceive()` method in Listing 6.10 is called. Here the first thing we to is to check that the results are available. Afterwards, each Wifi access point is extracted from the results and is stored in the `recordedWifi` array in the core service. Once completed, a new scan is started.

LocationService

The `LocationService` class registers listeners for both network and GPS location updates.

```java
private LocationListener locationListener = new LocationListener() {
  public void onLocationChanged(Location location) {
    if (previousLocation == null) {
      insertLocationData(location);
      previousLocation = location;
    }else if (location.getProvider().equals("network")
           && previousLocation.getProvider().equals("gps")
           && System.currentTimeMillis() - previousLocation.getTime() <
               6000) {
        Log.v(TAG, "Ignoring network location");
    } else {
        Log.v("Timediff", String.valueOf(System.currentTimeMillis() -
             previousLocation.getTime()));
        previousLocation = location;
        insertLocationData(location);
    }
  }

  void insertLocationData(Location location) {
    // Called when adapter new location is found by the network location
        provider.
    long time = System.currentTimeMillis();

    LocationData aLocation;
    aLocation = new LocationData(location.getProvider(),
            location.getLatitude(), location.getLongitude(), location.
                getAccuracy(), time);
    CoreService.recordedLocation.add(aLocation);
  }
};
```

Listing 6.11: locationListener from the LocationService class

Once again the `onLocationChanged()` method is called when Android gets an updated location. Because we have requested location updates from both network and GPS, the method will be called for both. If the GPS provides a consistent location, it is preferred over the network location, network location updates will therefore be ignored if the previous location update was from the GPS and recent. If the previous location was from the GPS and is too old, we consider the network location to be better and allow it to be used. When a location update has been received and not ignored, the `insertLocationData()` method is called, which inserts the location data in the `recordedLocation` array in the core service.

AccelerometerService

The `AccelerometerService` class in Listing 6.12 collects data in form of gravity, linear acceleration, and magnetic field. Each new data point will call the `onSensorChanged()` method, and the data will be copied to the corresponding variable.

```
public void onSensorChanged(SensorEvent event) {
  if (event.sensor == gravitySensor) {
      System.arraycopy(event.values, 0, gravity, 0, event.values.length);
  } else if (event.sensor == magneticFieldSensor) {
      System.arraycopy(event.values, 0, magneticField, 0, event.values.
          length);
      recordCurrentOrientation(magneticField, gravity);
  } else if (event.sensor == linearAccelerationSensor) {
      System.arraycopy(event.values, 0, linearAcceleration, 0, event.values
          .length);
      recordLastSignificantMovement(linearAcceleration);
      accelerometerFilter(linearAcceleration[0], linearAcceleration[1],
          linearAcceleration[2]);
  } else if (event.sensor == rotationVectorSensor && linearAcceleration !=
      null) {
      System.arraycopy(event.values, 0, rotationVector, 0, event.values.
          length);
      rotateAccelerationToWorldCoordinates(linearAcceleration,
          rotationVector, event.timestamp);
  }
}
```

Listing 6.12: onSensorChanged() from the AccelerometerService class

When new data is received from the magnetic field sensor, the current orientation will be updated by the method `recordCurrentOrientation` in Listing 6.13. The orientation of the phone is calculated by the `getOrientation()` method but is returned in radians. The orientation is then converted to degrees before being saved in the core service class.

```java
private void recordCurrentOrientation(float[] magneticField, float[]
    gravity) {
  float R[] = new float[9];
  float I[] = new float[9];
  boolean success = SensorManager.getRotationMatrix(R, I, gravity,
      magneticField);
  if (success) {
    float orientation[] = new float[3];
    SensorManager.getOrientation(R, orientation);
    float azimuth = (float) Math.toDegrees(orientation[0]);
    azimuth = (azimuth + 360) % 360;

    CoreService.currentOrientation = azimuth;
  }
}
```

Listing 6.13: recordCurrentOrientation() from the AccelerometerService class

Updates from the linear accelerometer sensor will record the last significant movement based on a simple high-pass filter that looks for movement less than 0.5 m/s^2 in any direction. The rotation vector sensor provides a vector that can be used to rotate data from device coordinates to world coordinates, this is done in the `rotateAcceleration-ToWorldCoordinates()` method in Listing 6.14. The rotation vector is converted into a rotation matrix which is then inverted. The linear accelerometer data is then multiplied onto the rotation matrix to get the linear acceleration in the world coordinates.

```java
\begin{lstlisting}[language=java, caption={() from the AccelerometerService
    class}, label={lst:accelerometerRotateCoordinates}]
private void rotateAccelerationToWorldCoordinates(float[]
    linearAcceleration, float[] rotationVector, long timestamp) {
  float[] rotationMatrixInverted = new float[16];
  float[] rotationMatrix = new float[16];
  float[] rotatedLinearAcceleration = new float[4];

  // Calculate the rotation matrix from the rotation vector
  SensorManager.getRotationMatrixFromVector(rotationMatrix, rotationVector)
      ;

  // Invert the rotation matrix.
  android.opengl.Matrix.invertM(rotationMatrixInverted, 0, rotationMatrix,
      0);

  // Multiply the linear acceleration onto the inverted rotation matrix to
      get linear acceleration in
  // earth coordinates.
  android.opengl.Matrix.multiplyMV(rotatedLinearAcceleration, 0,
      rotationMatrixInverted, 0, linearAcceleration, 0);

  calculateVelocity(rotatedLinearAcceleration, timestamp);
}
```

Listing 6.14: rotatrotateAccelerationToWorldCoordinateseCoordinates() from the AccelerometerService class

When the linear accelerometer data has been rotated, the current velocity will be calculated. This is done in `calculateVelocity()` method in Listing 6.15.

```java
\begin{lstlisting}[language=java, caption={() from the AccelerometerService
    class}, label={lst:accelerometerRotateCoordinates}]
```

```
private void calculateVelocity(float[] linearAcceleration, long timestamp)
    {
  if (previousTimestamp != 0) {
    dT = (timestamp - previousTimestamp) * NS2S;
    previousTimestamp = timestamp;
    float velocity[] = new float[3];
    velocity[0] = (dT * linearAcceleration[0]) + previousVelocity[0];
    velocity[1] = (dT * linearAcceleration[1]) + previousVelocity[1];
    velocity[2] = (dT * linearAcceleration[2]) + previousVelocity[2];

    processSensorData(linearAcceleration, velocity);

    previousVelocity = velocity;
  } else {
    previousTimestamp = timestamp;
  }
}
```

Listing 6.15: calculateVelocity() from the AccelerometerService class

Once the velocity has been calculated, it is like the other data inserted into the `recordedAccelerometer` array in the core service, as seen in Listing 6.16.

```
private void processSensorData (float[] linearAcceleration, float[]
    velocity) {
  long time = System . currentTimeMillis () ;
  AccelerometerData anAccelerometerEvent = new AccelerometerData (
      linearAcceleration[0], linearAcceleration[1], linearAcceleration[2],
      velocity[0], velocity[1], velocity[2], time) ;
  CoreService.recordedAccelerometer.add(anAccelerometerEvent);
}
```

Listing 6.16: processSensorData() from the AccelerometerService class

6.4 Geofence

The geofences that are used for for starting and stopping data collection are handled by the `Geofence` and the `GeofenceService` classes. These geofences rely on the Google Play Services, therefore the application must be connected to the Google API. This is done by the `CoreService` class when it is started, seen in **??**, once the `mGoogleApiClient` is connected the `CoreService` calls the methods `addGeofence()` and `registerGeofences()` in the `Geofence` class.

```
private void populateGeofenceList(String name, LatLng location, Float
    radius) {
  // Calculate hours to milliseconds
  int expireInHours = 720;
  long expireInMilliseconds = expireInHours * 60 * 60 * 1000;

  geofenceArrayList.add(new com.google.android.gms.location.Geofence.
      Builder()
      .setRequestId(name)
      .setCircularRegion(location.latitude, location.longitude, radius)
      .setExpirationDuration(expireInMilliseconds)
      .setTransitionTypes(com.google.android.gms.location.Geofence.
          GEOFENCE_TRANSITION_ENTER |
              com.google.android.gms.location.Geofence.
                  GEOFENCE_TRANSITION_EXIT)
```

```
        .build());
}
```

Listing 6.17: populateGeofencingList() method in the Geofence class

The `addGeofence()` method calls the `populateGeofenceList`, seen in Listing 6.17, two times for each lock. The two calls add the inner and outer geofence for the locks. The `populateGeofenceList` creates a geofence and adds it to the list of geofences to be registered.

```java
void registerGeofences(Context context, GoogleApiClient googleApiClient) {
  if (!googleApiClient.isConnected()) {
      Log.v(TAG, "GoogleApiClient not connected");
  }

  try {
    LocationServices.GeofencingApi.addGeofences(
            googleApiClient,
            geofencingRequest(),
            getGeofencePendingIntent(context)
    ).setResultCallback((ResultCallback<? super Status>) context);
  } catch (SecurityException securityException) {
    logSecurityException(securityException);
  }
}
```

Listing 6.18: registerGeofences() method in the Geofence class

The `registerGeofences()` method goes through the list of geofences and registers them with the Google Play Services. Registering the geofences also starts the `GeofenceService` which has the purpose of handling the intent sent by the Google Play Services when a geofence has been entered or exited.

```java
protected void onHandleIntent(Intent intent) {
  GeofencingEvent geofencingEvent = GeofencingEvent.fromIntent(intent);
  if (geofencingEvent.hasError()) {
    String errorMessage = getErrorString(this, geofencingEvent.getErrorCode
        ());
    Log.e(TAG, errorMessage);
    return;
  }

  int geofenceTransition = geofencingEvent.getGeofenceTransition();

  List<Geofence> triggeringGeofences = geofencingEvent.
      getTriggeringGeofences();

  ArrayList triggeringLockList = new ArrayList();
  for (Geofence geofence : triggeringGeofences) {
    String lockMAC = geofence.getRequestId();
    Log.i(TAG, "onHandleIntent: " + lockMAC);
    triggeringLockList.add(lockMAC);
  }

  if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER) {
    Intent geofencesEntered = new Intent("GEOFENCES_ENTERED");
    geofencesEntered.putExtra("Geofences", triggeringLockList);
    sendBroadcast(geofencesEntered);
  } else if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT){
    Intent geofencesExited = new Intent("GEOFENCES_EXITED");
```

```
      geofencesExited.putExtra("Geofences", triggeringLockList);
      sendBroadcast(geofencesExited);
   } else {
      Log.e(TAG, getString(R.string.geofence_transition_invalid_type,
          geofenceTransition));
   }
}
```

Listing 6.19: onHandleIntent() method in the GeofenceService class

Whenever the geofence service receives an intent from the Google Play Services, the `onHandleIntent()` method in Listing 6.19 is called. This method sends a new intent to the core service with information about the triggering locks. The new intent differs depending on whether the triggering event was an entering or exiting of a geofence.

6.5 Data Storage

Database

Whenever the application is started, and the database does not exist, it will be created by the `createDatastore()` in Listing 6.20. The creation, deletion, and upgrading is handled by the private class `DatabaseHelper` which is an extension of the Android abstract class `SQLiteOpenHelper` which takes care of opening the database if it exists and creating the database if it does not. `SQLiteOpenHelper` also takes care of upgrading the database and makes sure that transactions are used such that the database always is in a consistent state.

```
private void createDatastore(SQLiteDatabase database) {
   database.execSQL("CREATE TABLE " + LOCK_TABLE + " ("
        + LOCK_MAC + " TEXT PRIMARY KEY, "
        + LOCK_PASSPHRASE + " TEXT, "
        + LOCK_LATITUDE + " DOUBLE, "
        + LOCK_LONGITUDE + " DOUBLE, "
        + LOCK_INNER_GEOFENCE + " FLOAT, "
        + LOCK_OUTER_GEOFENCE + " FLOAT, "
        + LOCK_ORIENTATION + " FLOAT, "
        + TIMESTAMP + " LONG)");

   database.execSQL("CREATE TABLE " + BLUETOOTH_TABLE + " ("
        + BLUETOOTH_NAME + " TEXT, "
        + BLUETOOTH_SOURCE + " TEXT, "
        + BLUETOOTH_RSSI + " INTEGER, "
        + BLUETOOTH_NEARBY_LOCK + " FOREIGNKEY REFERENCES " + LOCK_TABLE + "(
            " + LOCK_MAC + "), "
        + TIMESTAMP + " LONG, "
        + "PRIMARY KEY (" + BLUETOOTH_SOURCE + ", " + BLUETOOTH_NEARBY_LOCK +
            "))");

   database.execSQL("CREATE TABLE " + WIFI_TABLE + " ("
        + WIFI_SSID + " TEXT, "
        + WIFI_MAC + " TEXT, "
        + WIFI_RSSI + " INTEGER, "
        + WIFI_NEARBY_LOCK + " FOREIGNKEY REFERENCES " + LOCK_TABLE + "(" +
            LOCK_MAC + "), "
        + TIMESTAMP + " LONG, "
        + "PRIMARY KEY (" + WIFI_MAC + ", " + WIFI_NEARBY_LOCK + "))");
```

```

  database.execSQL("CREATE TABLE " + ACCELEROMETER_TABLE + " ("
      + ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
      + ACCELERATION_X + " FLOAT, "
      + ACCELERATION_Y + " FLOAT, "
      + ACCELERATION_Z + " FLOAT, "
      + SPEED_X + " FLOAT, "
      + SPEED_Y + " FLOAT, "
      + SPEED_Z + " FLOAT, "
      + TIMESTAMP + " LONG)");
}
```

Listing 6.20: The creation of tables in the database.

Insertion of data into the database is also handled by the `DataStore` class. Listing 6.21 shows the insertion of lock data, insertion of Wifi, Bluetooth, and accelerometer data is done similarly. When inserting new data into a table in the database, the data is first inserted into a `ContentValues` object.

For the actual insertion, a writable database is provided by the `databaseHelper` and the transaction is started by the `beginTransaction()` method. The `beginTransaction()` method begins a transaction in an exclusive mode such that only one transaction can make changes to the database at a time. This prevents other transactions from being carried out at the same time and potentially create inconsistencies in the data. It also allows the transaction to be rolled back if it is not successful. The transaction is marked successful by the `setTransactionSuccessful()` which will commit the changes, and the `endTransaction()` will exit the exclusive mode and allow other transactions to be carried out again.

```
void insertLockDetails(
    String lockMAC,
    String lockPassphrase,
    double lockLatitude,
    double lockLongitude,
    float lockInnerGeofence,
    float lockOuterGeofence,
    float lockOrientation,
    long timestamp
) {
  ContentValues contentValues = new ContentValues();
  contentValues.put(LOCK_MAC, lockMAC);
  contentValues.put(LOCK_PASSPHRASE, lockPassphrase);
  contentValues.put(LOCK_LATITUDE, lockLatitude);
  contentValues.put(LOCK_LONGITUDE, lockLongitude);
  contentValues.put(LOCK_INNER_GEOFENCE, lockInnerGeofence);
  contentValues.put(LOCK_OUTER_GEOFENCE, lockOuterGeofence);
  contentValues.put(LOCK_ORIENTATION, lockOrientation);
  contentValues.put(TIMESTAMP, timestamp);

  try {
    database = databaseHelper.getWritableDatabase();
    database.beginTransaction();
    database.replace(LOCK_TABLE, null, contentValues);
    database.setTransactionSuccessful();
  } finally {
    database.endTransaction();
  }
}
```

Listing 6.21: The insertion of a lock in the database.

Whenever a known lock is encountered, the `getLockDetails()` method in Listing 6.22 will be called to get the saved lock, nearby Wifi access points, and nearby Bluetooth devices such that they can be compared with the newly captured data. The method returns a `LockData` data object that includes all the data that is needed by the heuristics to make a decision.

```
LockData getLockDetails(String foundLock) {
  LockData lockData;
  LocationData locationData;
  BluetoothData bluetoothData;
  WifiData wifiData;

  String lockMac;
  String lockPassphrase;
  double lockLatitude;
  double lockLongitude;
  float innerGeofence;
  float outerGeofence;
  float orientation;

  ArrayList<BluetoothData> nearbyBluetoothDevices = new ArrayList<>();
  ArrayList<WifiData> nearbyWifiAccessPoints = new ArrayList<>();

  try {
    database = databaseHelper.getReadableDatabase();
    database.beginTransaction();

    String lockQuery = "SELECT * FROM " + LOCK_TABLE + " WHERE " + LOCK_MAC
        + "='" + foundLock + "';";
    Cursor lockCursor = database.rawQuery(lockQuery, null);

    lockCursor.moveToFirst();
    if (lockCursor.isAfterLast()) {
      // We have not found any locks and return null.
      lockCursor.close();
      return null;
    } else {
      lockMac = lockCursor.getString(lockCursor.getColumnIndex(LOCK_MAC));
      lockPassphrase = lockCursor.getString(lockCursor.getColumnIndex(
          LOCK_PASSPHRASE));
      lockLatitude = lockCursor.getDouble(lockCursor.getColumnIndex(
          LOCK_LATITUDE));
      lockLongitude = lockCursor.getDouble(lockCursor.getColumnIndex(
          LOCK_LONGITUDE));
      innerGeofence = lockCursor.getInt(lockCursor.getColumnIndex(
          LOCK_INNER_GEOFENCE));
      outerGeofence = lockCursor.getInt(lockCursor.getColumnIndex(
          LOCK_OUTER_GEOFENCE));
      orientation = lockCursor.getFloat(lockCursor.getColumnIndex(
          LOCK_ORIENTATION));
      lockCursor.close();
    }

    String bluetoothQuery = "SELECT * FROM " + BLUETOOTH_TABLE + " WHERE "
            + BLUETOOTH_NEARBY_LOCK + "='" + foundLock + "';";
    Cursor bluetoothCursor = database.rawQuery(bluetoothQuery, null);
    if (bluetoothCursor.getColumnCount() != 0) {
      bluetoothCursor.moveToFirst();
      for (int i = 0; i <= bluetoothCursor.getColumnCount(); i++) {
        String bluetoothName = bluetoothCursor.getString(bluetoothCursor.
            getColumnIndex(BLUETOOTH_NAME));
        String bluetoothSource = bluetoothCursor.getString(bluetoothCursor.
            getColumnIndex(BLUETOOTH_SOURCE));
        int bluetoothRSSI = bluetoothCursor.getInt(bluetoothCursor.
            getColumnIndex(BLUETOOTH_RSSI));
        long bluetoothTimestamp = bluetoothCursor.getLong(bluetoothCursor.
            getColumnIndex(TIMESTAMP));

        bluetoothData = new BluetoothData(bluetoothName, bluetoothSource,
            bluetoothRSSI, bluetoothTimestamp);
        nearbyBluetoothDevices.add(bluetoothData);

        if (!(bluetoothCursor.isLast() || bluetoothCursor.isAfterLast())) {
          bluetoothCursor.moveToNext();
        }
      }
    }
    bluetoothCursor.close();

    String wifiQuery = "SELECT * FROM " + WIFI_TABLE + " WHERE "
            + WIFI_NEARBY_LOCK + "='" + foundLock + "';";
    Cursor wifiCursor = database.rawQuery(wifiQuery, null);
    if (wifiCursor.getColumnCount() != 0) {
      wifiCursor.moveToFirst();
      for (int i = 0; i <= wifiCursor.getColumnCount(); i++) {
        String wifiSSID = wifiCursor.getString(wifiCursor.getColumnIndex(
            WIFI_SSID));
        String wifiMAC = wifiCursor.getString(wifiCursor.getColumnIndex(
            WIFI_MAC));
        int wifiRSSI = wifiCursor.getInt(wifiCursor.getColumnIndex(
            WIFI_RSSI));
        long wifiTimestamp = wifiCursor.getLong(wifiCursor.getColumnIndex(
            TIMESTAMP));

        wifiData = new WifiData(wifiSSID, wifiMAC, wifiRSSI, wifiTimestamp)
            ;
        nearbyWifiAccessPoints.add(wifiData);

        if (!(wifiCursor.isLast() || wifiCursor.isAfterLast())) {
            wifiCursor.moveToNext();
        }
      }
    }
    wifiCursor.close();

    locationData = new LocationData(lockLatitude, lockLongitude);
    lockData = new LockData(lockMac, lockPassphrase, locationData,
            innerGeofence, outerGeofence, orientation,
                nearbyBluetoothDevices, nearbyWifiAccessPoints);
    return lockData;
  } finally {
    database.endTransaction();
  }
}
```

Listing 6.22: Getting lock details and associated data from the database.

Data Buffer

The data buffer is implemented as a circular buffer that contains data from a two second scanning interval. The data processor service has the purpose of moving the recorded data lists from the core service to the data buffer and clear the data from the lists for data collection in a new time slice. In order to prevent the lists from being empty, they will not be emptied if no new data has been collected. This means that the tail of the data buffer always contains the most recently collected data.

The decision for unlocking a known lock is started from the data processor service when the Bluetooth signal from a known lock is encountered and the current location is known, seen in Listing 6.23. We allow the list of Wifi access points to be empty because we cannot be sure that any are nearby, the location and lock is needed to make a decision and we will not start the decision making unless they are available.

```
// Do not start decision making before we have at least one nearby
    Bluetooth device (the lock),
// and adapter location. We cannot be sure any Wifi access points are
    nearby.
if (!prevRecordedBluetooth.isEmpty() && !prevRecordedLocation.isEmpty()) {
  for (int i = 0; i < CoreService.recordedBluetooth.size(); i++) {
    if (CoreService.recordedBluetooth.get(i).getSource().equals(
        BluetoothService.SIMON_BEKEY)) {
      foundLocks.add(CoreService.recordedBluetooth.get(i).getSource());
      Intent startDecision = new Intent("START_DECISION");
      startDecision.putStringArrayListExtra("Locks", foundLocks);
      sendBroadcast(startDecision);
    }
  }
  if (!foundLocks.isEmpty()) {
    sendDecisionIntent(foundLocks);
  }
}
```

Listing 6.23: Starting the decision making when a known lock is encountered from the data processor service.

6.6 Heuristics

The `Heuristics` class handles decision, tuning of stored data, and notification to the user that is used to guide the data tuning.

When the inner geofence is entered the service `ScannerService` in Listing 6.24 is started along with the data collection services. The job of the `ScannerService` is to start the heuristics decision once the smartphone is near a lock that it is also inside of the inner geofence of, and there has been no significant movement for two seconds, and the phone is near the recorded orientation. The service will continue to run for 10 minutes or until a decision to unlock has been made.

```
public void run() {
  List<String> foundLocks = new ArrayList<String>();
  ArrayList<String> decisionLocks = new ArrayList<String>();
  long startTime = System.currentTimeMillis();

  while (running) {
    for (BluetoothData bluetoothData : CoreService.recordedBluetooth) {
      if (CoreService.activeInnerGeofences.contains(bluetoothData.getSource
          ())) {
        foundLocks.add(bluetoothData.getSource());
      }
    }
    if (!foundLocks.isEmpty() && System.currentTimeMillis() - CoreService.
        lastSignificantMovement > 2000) {
      for (String foundLock : foundLocks) {
        LockData foundLockWithDetails = CoreService.dataStore.
            getLockDetails(foundLock);
        if (foundLockWithDetails.getOrientation() == -1) {
          NotificationUtility notification = new NotificationUtility();
          notification.displayOrientationNotification(getApplicationContext
              (), foundLockWithDetails.getMAC(), CoreService.
              currentOrientation);
          sendBroadcast(stopScan);
          running = false;
          stopSelf();
        } else if (Math.min(Math.abs(CoreService.currentOrientation -
            foundLockWithDetails.getOrientation()),
                Math.min(Math.abs((CoreService.currentOrientation -
                    foundLockWithDetails.getOrientation()) + 360),
                        Math.abs((CoreService.currentOrientation -
                            foundLockWithDetails.getOrientation()) - 360)))
                < 22.5 ) {
          decisionLocks.add(foundLock);
        }
      }
        if (!decisionLocks.isEmpty()) {
          Intent startDecision = new Intent("START_DECISION");
          startDecision.putStringArrayListExtra("Locks", decisionLocks);
          sendBroadcast(startDecision);

          sendBroadcast(stopScan);
          running = false;
          stopSelf();
        }
```

Listing 6.24: The run() method from the ScannerService

The decision to unlock an encountered lock is done by the `makeDecision` method seen in Listing 6.25. The method compares the most recently collected data with the data stored in the database for each found lock and gives a score. If multiple locks are found, the lock with the highest score is chosen to be unlocked. No door is unlocked if the score for all locks are too low. The decision is made based on the nearby Wifi and Bluetooth devices, how great the distance is in meters between the current location and the stored location, the orientation, and whether the smartphone is inside the geofences or not. Each detected lock is given a point score based and the lock with the most points is chosen. If that lock has enough points, the decision is to unlock.

```
boolean makeDecision(Context context, ArrayList<String> foundLocks) {
  Map<String, Double> lockScores = new HashMap<>();
  Map.Entry<String, Double> maxEntry = null;

  // For each lock nearby, compare the recently recorded data with the
      stored data and give adapter score.
  for (String foundLock : foundLocks) {
    double lockScore = 0;
    int validWifi = 0;
```

```java
    int validBluetooth = 0;
    LockData storedLockData;
    storedLockData = CoreService.dataStore.getLockDetails(foundLock);

    if (!recentWifiList.isEmpty()) {
      for (WifiData storedWifi : storedLockData.getNearbyWifiAccessPoints()
          ) {
        for (WifiData recentWifi : recentWifiList ) {
          if (storedWifi.getMac().equals(recentWifi.getMac())) {
            validWifi++;
          }
        }
      }
      if (validWifi != 0) {
        lockScore += ((double) validWifi / (double)storedLockData.
            getNearbyWifiAccessPoints().size()) * 100;
      }
    }

    if (!recentBluetoothList.isEmpty()) {
      for (BluetoothData storedBluetooth : storedLockData.
          getNearbyBluetoothDevices()) {
        for (BluetoothData recentBluetooth : recentBluetoothList) {
            if (storedBluetooth.getSource().equals(recentBluetooth.
                getSource())) {
              validBluetooth++;
            }
        }
      }
      if (validBluetooth != 0) {
        lockScore += ((double) validBluetooth / (double)storedLockData.
            getNearbyBluetoothDevices().size()) * 100;
      }
    }

    if (!recentLocationList.isEmpty()) {
    float[] results = new float[3];
    Location.distanceBetween(storedLockData.getLocation().getLatitude(),
        storedLockData.getLocation().getLongitude(),
      recentLocationList.get(recentLocationList.size() - 1).getLatitude(),
          recentLocationList.get(recentLocationList.size() - 1).
          getLongitude(),
      results);
    lockScore += 100 - results[0];
}

    if (CoreService.activeInnerGeofences.contains(foundLock)
        && CoreService.activeOuterGeofences.contains(foundLock)) {
      lockScore += 50;
    } else {
      lockScore -= 1000;
    }
    lockScores.put(foundLock, lockScore);
  }

  // Find lock with highest score, if multiple use first one found.
  for (Map.Entry<String, Double> entry : lockScores.entrySet()) {
    if (maxEntry == null || entry.getValue().compareTo(maxEntry.getValue())
        > 0) {
      maxEntry = entry;
    }
  }

  if (maxEntry.getValue() > 250) {
    NotificationUtility notification = new NotificationUtility();
    notification.displayUnlockNotification(context, maxEntry.getKey(),
        recentBluetoothList, recentWifiList, recentLocationList);
    return true;
  } else {
    return false;
  }
}
```

Listing 6.25: The decision making from the heuristics.

Chapter 7

Evaluation and Conclusion

The evaluation of the prototype will be done through a number of tests. First each of the data collection services will be tested independently in order to judge the quality of the data that is being collected. This data is important because the decision for unlocking will be made on the assumption that the data is correct and consistent. If the data is inconsistent or incorrect, the decision will likely not be correct.

Following that, the database will be exported and the data collected about a known will be inspected to see if the stored data is correct. Finally, the application will be tested in a real-world scenario to see if it works like expected.

7.1 Inertial Sensors

Smartphones consist of a large amount of electronics in a very small space, still manufacturers are continuously trying to make the devices thinner and thinner. This means that the sensors in smartphones have to be as small as possible, while still being accurate enough to carry out necessary functionality. The result is that while the sensors are good for detecting the orientation of the phone and movement input for games, they might struggle in scenarios that require more accuracy. In order to understand if phones are accurate enough for the intended usage of recording the movement of a person in order to unlock a door automatically, the two most important sensors, accelerometer and magnetic field sensor, will be tested.

The sensors in phones can be limited in their accuracy both as a result of their size, but also as a result of their location. Other electronic parts in the phone might interfere with the sensors due to lack of, or insufficient electromagnetic shielding.

The tests will be carried out on a Nexus 6 phone from Google. The sensors included in the Nexus 6 for accelerometer, magnetic field sensor, and gyroscope are all made by the company InvenSense.

7.1.1 Accelerometer

From the accelerometer, the interesting measurements are how precisely it can detect acceleration forces applied to the phone and how much noise the sensor generates. The noise will be noticeable in form of inaccuracies when the phone is lying still. Noise can also affect how precise the measurements are over longer periods of time.

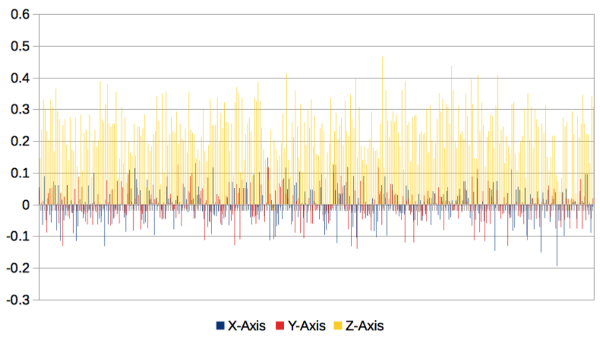


Figure 7.1: Accelerometer sensor measurements on the Nexus 6.

Figure 7.1 shows data outputted from the accelerometer, the data set consists of 313 data points captured over a minute wile the phone was lying completely still with the back of the phone on a table without any metal nearby. The data is from the linear accelerometer sensor in Android which removes gravity acceleration and only reports the acceleration caused by movement of the device.

For the accelerometer to output correct values for this test, all axes should report 0 m/s^2 acceleration for all observations. In the case of Figure 7.1 the x, and y-axes generally output values between -0.1 and 0.1 m/s^2 while the z-axis outputs values between 0 and 0.5 m/s^2.

7.1.2 Velocity

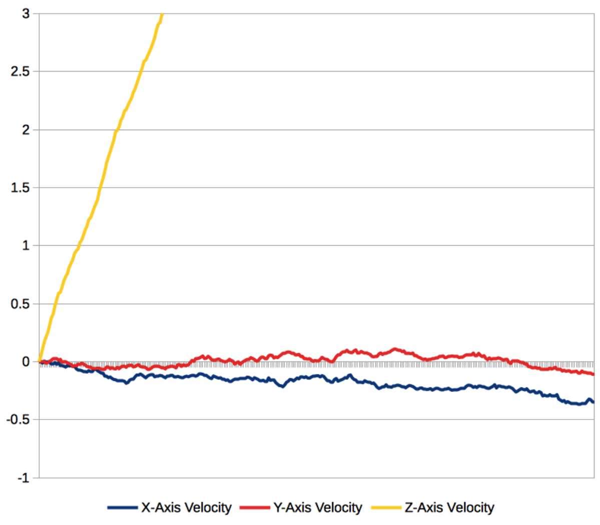


Figure 7.2: Velocity of each axis based on the data set in Figure 7.1.

Figure 7.2 shows that the calculated velocity for the data in Figure 7.1. The calculated velocity for all axes is highly inaccurate, the z-axis is especially bad, but the x and y-axes are not usable either. The velocity of the z-axis continuously increases such that the reported velocity is 16 m/s after one minute. The x-axis reaches -0.36 m/s and the y-axis varies between 0.1 and -0.1 m/s over the course of the minute.

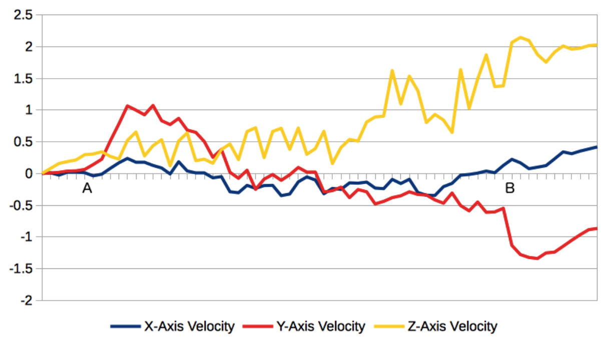


Figure 7.3: Velocity calculation for a 10 meter walk in the y-axis direction with other axes not moving.

Another example can be seen on Figure 7.3. For this test I move forward from a standing position, while holding the phone in the same orientation as it was laying on the table, for about 10 meters and then stop. The forward movement is in the positive y-axis direction with vertical movement on the z-axis. The x-axis is kept as still as possible. The walking starts at A in Figure 7.3 and stopping begins at B, between these two points, the forward velocity of the phone remains the same. This is not what the accelerometer captured. The points the movement starts and stops are quite clearly visible, however, the calculated velocity on the y-axis should remain the same, or close to, between these two points and it does not. Instead the calculated velocity quickly decreases, and starts to indicate that the phone is moving in the opposite direction of what it actually is. When stopping, the calculated velocity of the y-axis speeds up in the opposite direction rather than go to zero. The z-axis is once again very imprecise but the reported x-axis velocity moves between 0.25 and -0.4 m/s.

7.1.3 Magnetic Field Sensor

The interesting measurements from the magnetic field sensor is how precisely the phones orientation compared to the magnetic north pole can be calculated. For this test, the Nexus 6 has been orientated on a table with a compass as the reference.

In order to test the magnetic field sensor, the output has been converted to degrees such that it can be compared directly with a physical compass. The conversion relies on the gravity which was filtered from the linear acceleration used to test the accelerometer.

For the test on Figure 7.4 the Nexus 6 was calibrated by following the instructions from Google Maps (1. Tilt your phone forward and back, 2. Move it side to side, 3. And then tilt left and right)[34] then laid flat on a table with the y-axis pointing 90° east for 10 seconds.

Figure 7.4 shows that the magnetic field sensor, like the accelerometer, has quite a bit of noise. Measurements that should all be close to 90° range from 86.4 to 95.1. The average of all the measurements is however quite accurate with a calculated orientation of 90.6°. Unfortunately this precision does not last long, after 10 minutes in which a part consisted of walking around with the phone in the pocket and another part consisted of

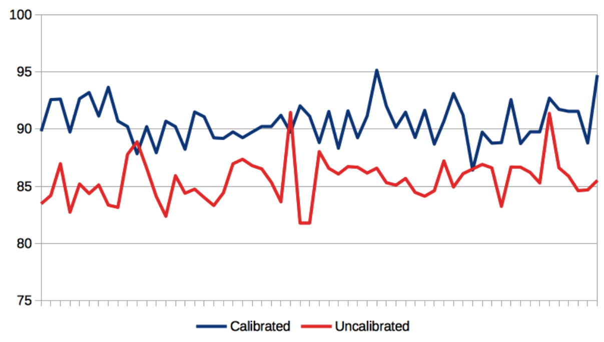


Figure 7.4: Compass precision on the y-axis oriented 90° east.

the phone laying on a table, the measured orientation is between 81.7 and 91.4 with an average of 85.5°.

In addition to the inaccuracies in the sensor, the magnetic field sensor can also easily be influenced by magnetic objects that are placed near the phone. If the phone for example is placed in a pocket or purse that also contains a pair of headphones, the measurements from the magnetic field sensor are unusable.

7.1.4 Gyroscope

The noise from the gyroscope is tested in the same way as the previous two sensors, while lying flat on the table for 10 seconds. Figure 7.5 shows the measurements in all three axes captured during the test. The gyroscope reports that the phone is rotating with between -0.018 and 0.019 rad/s.

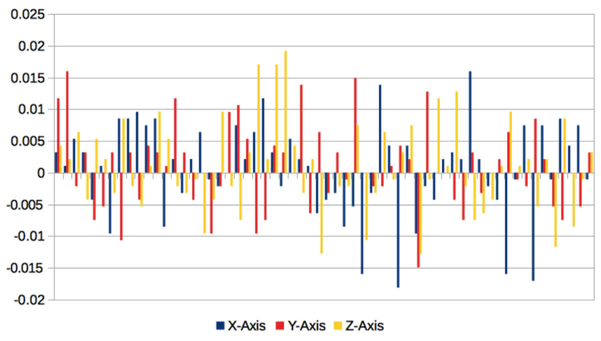


Figure 7.5: Gyroscope noise while lying still.

7.1.5 Rotated Data

Rotating the data collected from the device specific orientation to world coordinates makes use of a combination of sensors. The interesting things to test for this is whether the orientation of the accelerometer data works or not. Following that, the accuracy of the data, and then a test of how similar the data from walking the same path a number of times is.

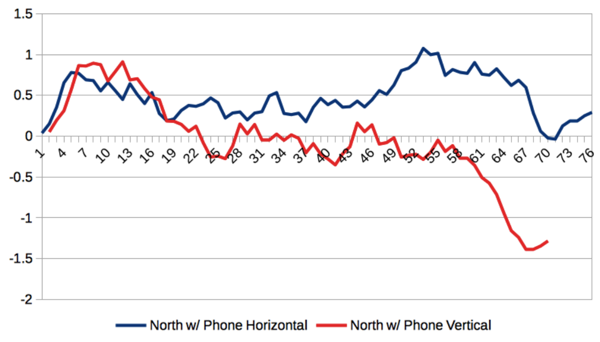


Figure 7.6: Y-axis from walking 10 meters straight north with the phone in two different orientations.

For the test in Figure 7.6 The phone was first held in the horizontal orientation like in Figure 7.3 and walked straight north, then it was held in a vertical orientation with the screen facing backwards. The test shows that the rotation of accelerometer data works as they both show similar movement. The horizontal test in blue resulted in a reasonable approximation of the movement, it starts and ends at a velocity of 0 m/s and is somewhat stable in between. The vertical test has the same initial acceleration but quickly diverges.

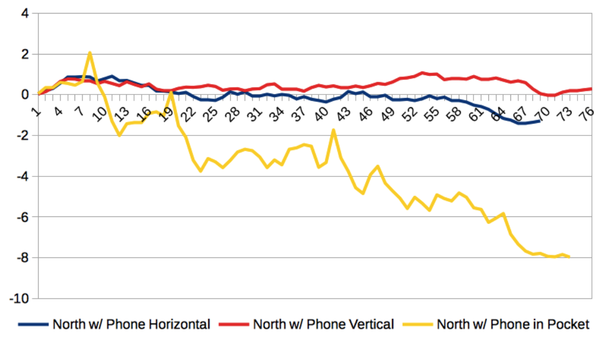


Figure 7.7: Addition of smartphone placed in the pocket.

Figure 7.7 is the same test as Figure 7.6 but has an additional measurement with the phone placed in a pocket. The placement in a pocket adds continuous rotation that was not present in the previous test. The drift for this test dwarves the previous two tests. The initial acceleration is once again quite similar and the deceleration at the end is also visible. Everything in between is indicating that the movement is in a southern direction, which is the complete opposite, and at a velocity of up to 6 m/s (21 km/h). So far, the data collected from the inertial sensors is not usable for any kind of pattern recognition.

In an attempt to get better data from the sensors, the sensor delay was decreased from normal to fastest. This increases the number of samples from 6 to 200 per second. The result on Figure 7.8 shows that the initial measurements are still good, but after three seconds all measurements are indicating that the phone is moving south instead of north. One definite improvement is that footsteps are much clearer with the increased fidelity.

For the last test, the phone was placed in a pocket and walked the same path 10 times with as much consistency as possible. The path was 3 meters west, then 20 meters north,

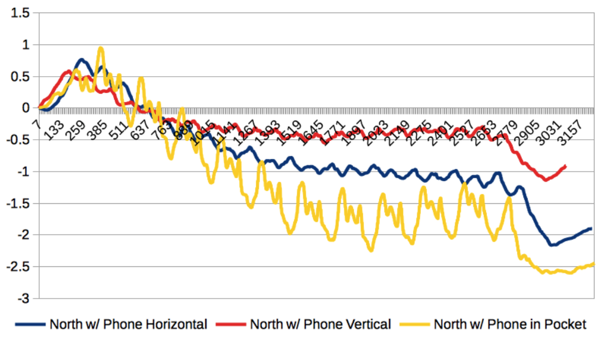


Figure 7.8: Increased number of samples per second.

and then 6 meters west again. The 10 samples should ideally show the same in order to calculate direction of movement and velocity.

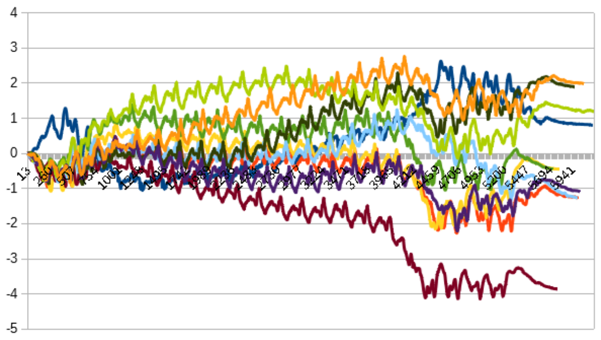


Figure 7.9: The eastern direction for the walked path.

Figure 7.9 shows the velocity in the eastern direction. As the phone is moving west in the beginning the initial velocity is negative. This is correctly reported by all but one of the measurements, and for the first three seconds nine of the graphs are showing about the same, after that they start to diverge.

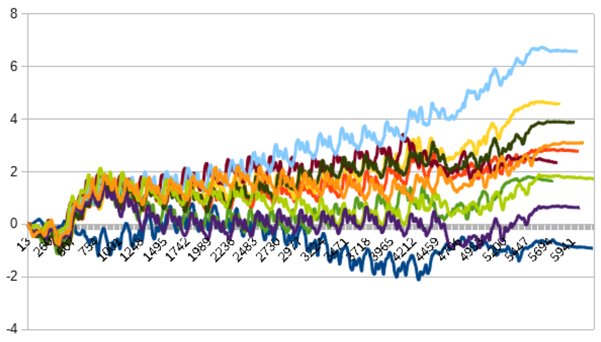


Figure 7.10: The northern direction for the walked path.

Figure 7.10 shows the velocity in the northern direction, and shows much the same results as the eastern direction. The same single measurement is much different from the rest in the beginning. After a couple of seconds the other measurements start to diverge as well.

The total time of these tests is about 25 seconds, at which point they vary with up to 7 m/s, even though the velocity was for the most part constant.

7.2 Bluetooth

Bluetooth is used to communicate with the lock, therefore the test will consist of the frequency with which the phone will receive the Bluetooth announces that the BeKey lock sends out two times a second. Additionally the distance at which the phone can detect the lock, both with a clear line of sight and behind a locked door where it will most often be placed when an unlocking attempt is made.

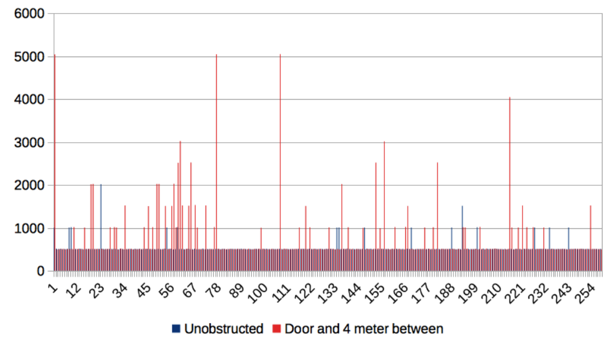


Figure 7.11: Measured time between received Bluetooth signal from BeKey.

The blue test in Figure 7.11 shows the time between Bluetooth signals received from the BeKey lock while the phone is lying next to it. This should be considered the optimal scenario and is used to test how often the phone is capable of receiving Bluetooth signals. As can be seen, almost every announcement is received and the maximal time between two signals is 2 seconds. The red test is the test that emulates the situation where the BeKey is most likely to be used. Here the distance between the phone and the lock is 4 meters and there is a door in between as an obstruction. Most of the signals are still received, but there are more instances where one or more announcements in a row are not received, and the most time between two received announcements is 5 seconds.

At the 4 meter distance the connection is not stable enough to guarantee that the BeKey application can unlock the door, more often than not it will show an error message.

7.3 Wifi

Testing Wifi will be done to see how much the measured RSSI changes and if samples collected over time conform to the expectations of about a 10 dBm difference. If it indeed does, the probability of a correct RSSI from the collected samples should also match. Also interesting is the rate of updates that the phone can provide. This rate will influence how many measurements can realistically be used if Wifi is to be used for indoor positioning.

Figure 7.12 shows the amount of each RSSI value measured over a distance of 3 meters with a direct line of sight to the wireless access point. In blue measurements is a 20 minute long test with 445 measurements which shows an excellent tight grouping of measured RSSI values. Here only 27 of the 445 measurements are different from -53 or

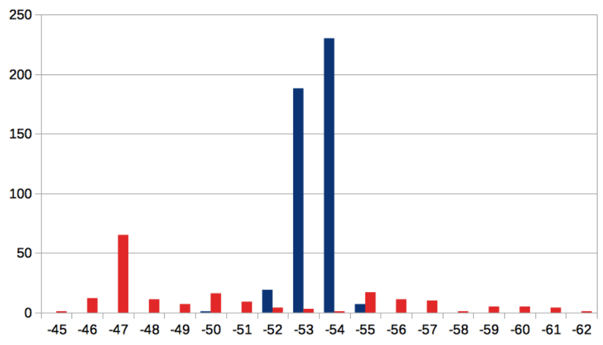


Figure 7.12: Measured Wifi RSSI from 3 meters with direct line of sight.

-54 dBm. The red measurements is the exact same test, in the same spot, measured minutes later for 9 minutes. Here are 183 measurements that vary wildly from each other with -47 being measured most often. The differences between the tests seem to be consistent once the test is started but if it is stopped and restarted the result seems to match either one or the other. The differences in measurements are possibly a result of slight change in location or orientation of the phone. These changes are less than one centimeter of movement with a 5° change in orientation.

The rate of new measurements is quite consistently just under 3 seconds. The least time between two measurements was 2847 ms and the largest time difference was 2956 ms with an average of 2887 ms.

7.4 Location

The location test will be carried out in order to determine how precise both the network location and the GPS location are. In addition the time between updates both when moving and stationary will be recorded and compared. For the accuracy, the estimated accuracy that is provided with a location result will be used.

7.4.1 Network Location

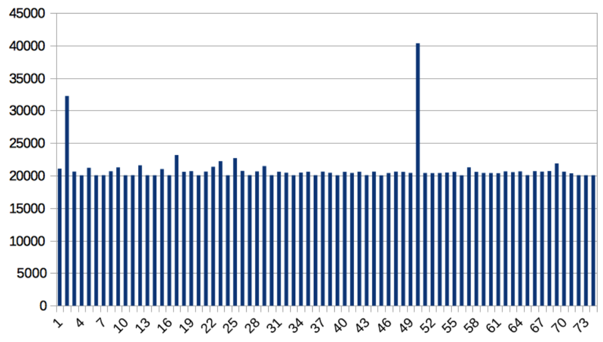


Figure 7.13: Time between location updates from network location.

For the network location, the time between location updates is around 20 seconds. The timing normally varies with a few seconds and once in a while no new location is received for up to 40 seconds. Figure 7.13 shows the update rate from a walk around a suburban neighborhood.

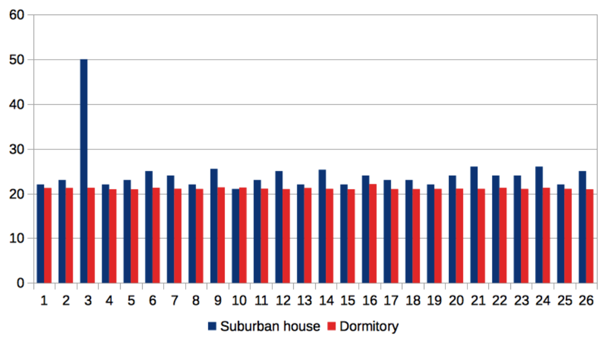


Figure 7.14: Reported accuracy of network location while laying still in two different locations.

The accuracy of the location reported by the network location is influenced by the number of nearby Wifi access points. The results in Figure 7.14 are from two tests where the smartphone was placed on a table indoors in two different locations. The blue test took place in a suburban house with only a few nearby access points, where the red test took place at the Ostenfeld dormitory where the number of access points is much higher. The suburban test has an accuracy of between 21 and 25 meters, with a single reported accuracy of 50 meters. The test at Ostenfeld is much more consistent with between 21 and 22 meters reported. Unexpectedly, the network location at a location with many Wifi access points is not much more precise than at a location with only a few.

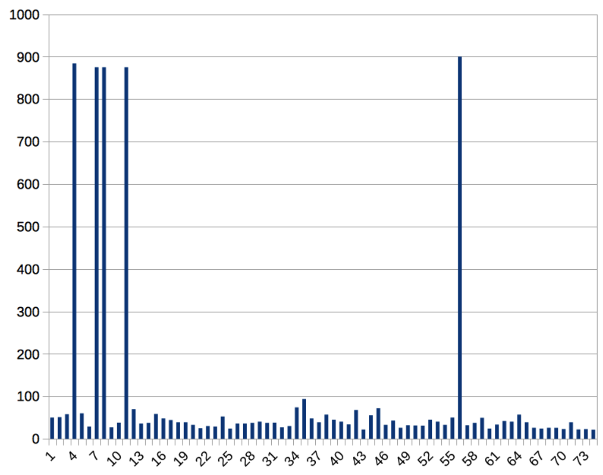


Figure 7.15: Reported accuracy of network location walking around a suburban neighborhood.

The next test in Figure 7.15 is a test of the network location accuracy when walking around a suburban neighborhood. Here the accuracy is significantly worse than inside a building. The most accurate locations were as accurate as the Ostenfeld test, but most of the reported locations have an accuracy of between 30 and 60 meters, a few observations were much worse with a reported accuracy of just under 900 meters. The large dip in accuracy might be because no Wifi access points were nearby and the calculation of the location is done with cellular towers only.

7.4.2 GPS Location

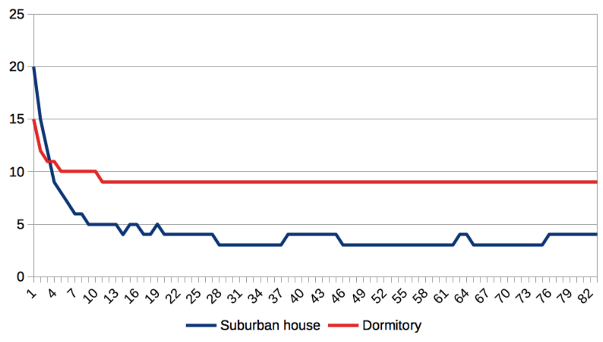


Figure 7.16: Reported accuracy of GPS location while laying still in two different locations.

The accuracy of the GPS location is tested in the same way as the network location. Figure 7.16 shows the stationary test in the same two locations. The blue test in the suburban house starts out with an accuracy of 20 meters and gets more precise over time, after 20 samples the accuracy stops improving and stays at between 3 and 4 meters. This is in line with the best possible accuracy for the GPS system. The red test at the Ostenfeld dormitory starts out a bit more precise than the first test, but does not improve as much. After about 10 samples the accuracy is at 9 meters and stays there. The rate of updates for the GPS is consistently under one second as long as the signal is good enough, if the signal drops then updates also stop.

7.4.3 Geofences

For the geofence, the precision and time it takes Android to notify the application that any geofence has been entered are of interest. If the precision is not good enough, it can be necessary to increase the sizes of the geofences. The same apply if the notifications come long after a geofence has been entered or exited.

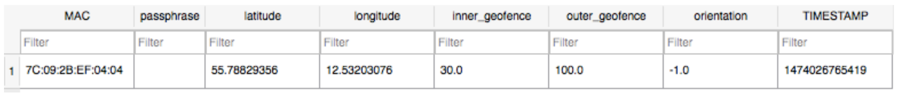
For the geofencing, the network location is used and the precision of the geofence is therefore influenced by the precision of the network location in a given area. In addition to the precision, the time between updates also influences how quickly the application gets notified about a geofence being entered or exited. The combination of the network location being within 20 meters at best and the 20 seconds between updates means that the size of geofences should be larger than first anticipated. For a suburban house, an inner geofence radius of 50 meter should suffice as a starting point, the outer geofence radius should start at between 500 and 1000 meters. This is because of the decreased accuracy of the network location when moving around outside.

Entering and exiting a geofence seems to have a bias for staying in the state it is currently in. Entering a small geofence of 20 meters does not always work, but when the geofence has been entered it will not exit unless the smartphone has been moved more than the 20 meter radius. In order to easier register the entering of small geofences, the GPS can be turned on and the phone will automatically use the improved location data from the

GPS, this means faster location updates and more precision. For this reason, turning on the GPS when the outer geofence is entered will make the inner geofence react quickly.

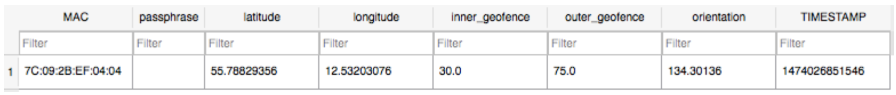
7.5 Database

A database dump will be taken when a lock is first inserted, and then when a number of parameters are updated then compared to see if the changes are as expected.



	MAC	passphrase	latitude	longitude	inner_geofence	outer_geofence	orientation	TIMESTAMP
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	7C:09:2B:EF:04:04		55.78829356	12.53203076	30.0	100.0	-1.0	1474026765419

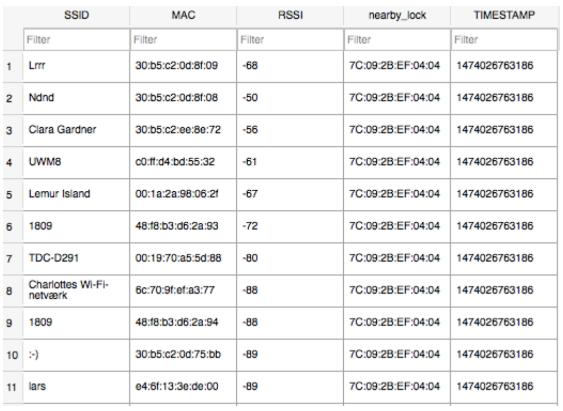
Figure 7.17: Lock table when first added.



	MAC	passphrase	latitude	longitude	inner_geofence	outer_geofence	orientation	TIMESTAMP
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	7C:09:2B:EF:04:04		55.78829356	12.53203076	30.0	75.0	134.30136	1474026851546

Figure 7.18: Lock table with added orientation and smaller outer geofence.

In Figure 7.17 shows the lock table in the database. The table contains the MAC address of an associated lock along with its location. At first the orientation is -1, which means that it has not been recorded and the inner and outer geofences have been set to the initial values. Figure 7.18 shows the same table updated with an orientation of the phone and a smaller outer geofence.



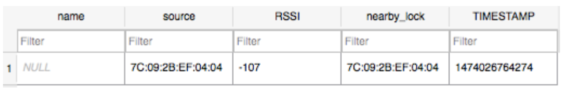
	SSID	MAC	RSSI	nearby_lock	TIMESTAMP
	Filter	Filter	Filter	Filter	Filter
1	Lrrr	30:b5:c2:0d:8f:09	-68	7C:09:2B:EF:04:04	1474026763186
2	Ndnd	30:b5:c2:0d:8f:08	-50	7C:09:2B:EF:04:04	1474026763186
3	Clara Gardner	30:b5:c2:ee:8e:72	-56	7C:09:2B:EF:04:04	1474026763186
4	UWM8	c0:ff:d4:bd:55:32	-61	7C:09:2B:EF:04:04	1474026763186
5	Lemur Island	00:1a:2a:98:06:2f	-67	7C:09:2B:EF:04:04	1474026763186
6	1809	48:f8:b3:d6:2a:93	-72	7C:09:2B:EF:04:04	1474026763186
7	TDC-D291	00:19:70:a5:5d:88	-80	7C:09:2B:EF:04:04	1474026763186
8	Charlottes Wi-Fi-netværk	6c:70:9f:ef:a3:77	-88	7C:09:2B:EF:04:04	1474026763186
9	1809	48:f8:b3:d6:2a:94	-88	7C:09:2B:EF:04:04	1474026763186
10	:-)	30:b5:c2:0d:75:bb	-89	7C:09:2B:EF:04:04	1474026763186
11	lars	e4:6f:13:3e:de:00	-89	7C:09:2B:EF:04:04	1474026763186

Figure 7.19: Wifi table.

Figure 7.19 shows the table of Wifi access points are nearby the associated lock. Each access point has an entry in the database with their SSID, MAC, and RSSI. Additionally, each access point has a lock associated with it. Figure 7.20 shows that only one Bluetooth device was nearby the lock when it was paired, this device is the lock itself.

7.6 Real-world Test

Despite having to make some unwanted change in how the decision is made, because the original idea of using the accelerometer to get a directional movement vector with speed

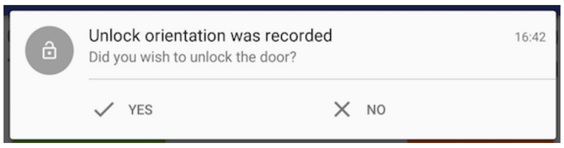


	name	source	RSSI	nearby_lock	TIMESTAMP
	Filter	Filter	Filter	Filter	Filter
1	NULL	7C:09:2B:EF:04:04	-107	7C:09:2B:EF:04:04	1474026764274

Figure 7.20: Bluetooth table.

and direction turned out to be inaccurate, the application is able to make a decision when a lock is nearby. The decision is made when the phone is held still in the prerecorded orientation for two seconds while nearby a known lock. The decision is based of of how many Wifi and Bluetooth signals that are associated with the lock have been recorded recently as well as the distance between the stored location for the lock and the recorded location of the device. If enough of these are available the decision to unlock is made.

The data collection is correctly started by the entering of the two geofences. When a new lock is added, the orientation is not set, this is done the next time the geofence is entered. Figure 7.21 shows the notification presented to the user when the orientation has been recorded by the phone. Pressing yes on this notification will insert the recorded orientation into the lock table in the database, pressing no will let the user try again at a later time.

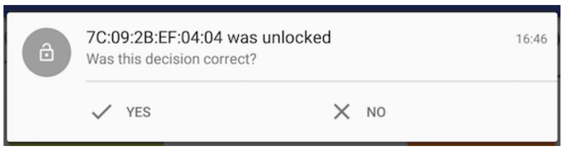


Unlock orientation was recorded
Did you wish to unlock the door?
16:42

✓ YES
✕ NO

Figure 7.21: Notification that the phones orientation has been recorded.

When the lock is encountered later with a recorded orientation, the user is presented with a new notification if the application decides to unlock. This notification in Figure 7.22 gives the user the option to change some of the parameters used for the decision by pressing no, which presents the user with a new activity, Figure 7.23.



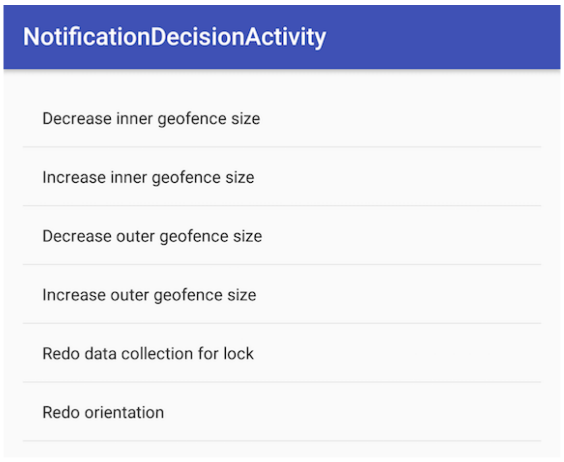
7C:09:2B:EF:04:04 was unlocked
Was this decision correct?
16:46

✓ YES
✕ NO

Figure 7.22: Notification about the decision to unlock.

The application is supposed to remain idle as long as it is inside the inner geofence after making a decision. This relies on the geofences being accurate, but the imprecision of the network location means that even in a small dorm room a radius of 30 meters is too small and results in the phone making decisions when it is not wanted. For a house in the country the inner geofence needs to be quite large if the accuracy of the network location spikes to a 900 meter radius.

Finally, the orientation relies on the magnetic field sensor which is not very accurate, and while it is better than calculating velocity and direction with the accelerometer, it is not accurate enough to be useful in reality. Too many things affect the sensor, and the end user would be annoyed by the inaccuracy.


NotificationDecisionActivity

Decrease inner geofence size

Increase inner geofence size

Decrease outer geofence size

Increase outer geofence size

Redo data collection for lock

Redo orientation

Figure 7.23: Activity to choose which parameter to change.

7.7 Conclusion

Smartphones today have many sensors, some of these work quite well, and some of them are only good enough for what they are currently used for.

The inertial sensors of accelerometer, gyroscope, and magnetic field sensor work well enough for rotating the content on the screen, as input for video games, for creating a visual compass, or for counting steps. Unfortunately, the sensors are quite noisy and the measured acceleration is not accurate enough for calculating velocity or direction of movement. The inertial sensors in ships and airplanes work because they are significantly larger with less noise and drift, they are also fixed in position such that the directions are always the same. Even with the much better accuracy of the inertial navigation systems, they rely on GPS signals to compensate for the drift that they are also affected by. If the inertial sensors in smartphones were accurate enough to get usable data for more than three seconds at a time, there would still need to be a compensation for drift that should be possible without GPS as the application is intended to work indoors where GPS is not always available.

Indoor positioning is possible with the use of Wifi, but because the received signals strength, which is used for determining the position, varies by up to 10 dBm an average of a number of samples is needed for accuracy. The Wifi `BroadcastReceiver` that is registered in the application receives a new measurement once every three seconds which is not quite fast enough to get precision close to the 3.5 meters that GPS can provide if the phone is being moved around at walking speed. The network location provided by Google only updates every 20 seconds and does not seem to get more precise than 20 meters.

The GPS and Bluetooth both work well with quick updates and reliable data. However, the distance from which the BeKey can actually be unlocked reliably is less than four meters from the lock. This gives the application a couple of seconds to decide whether to unlock or not and unlock the door. The time it takes from pressing the button in the BeKey application and the lock starts moving can be close to one second. Ideally the lock should have a longer communication range if the user is supposed to be able to open the door as soon as they reach it.

Starting data collection based on geofences works surprisingly well, but because of the network locations time between location updates and because these locations are potentially only accurate to within a kilometer, the geofences need to be larger than first anticipated.

Bibliography

[1] Jeff Benjamin for 9to5Mac. *FAQ: How to use Auto Unlock with watchOS 3 and macOS Sierra beta 2.* `http://9to5mac.com/2016/07/07/faq-how-to-use-auto-unlock-macos-sierra-beta-2-watchos-3/`.

[2] Federal Aviation Administration. *Global Positioning System (GPS) Standard Positioning Service (SPS) Performance Analysis Report.* `http://www.nstb.tc.faa.gov/reports/PAN86_0714.pdf#page=22`.

[3] Nirave Gondhia for Android Authority. *Exclusive: leaked Note 7 front panel confirms iris scanner.* `http://www.androidauthority.com/galaxy-note-7-front-iris-scanner-700932/`.

[4] ARM. *TrustZone.* `http://www.arm.com/products/processors/technologies/trustzone/index.php`.

[5] Google ATAP. *Project Jacquard.* `https://atap.google.com/jacquard/`.

[6] Google ATAP. *Project Soli.* `https://atap.google.com/soli/`.

[7] Sean Beaupre. *TRUSTNONE.* `http://theroot.ninja/disclosures/TRUSTNONE_1.0-11282015.pdf`. 2015.

[8] BeKey. *Promotional photo.* `https://bekey.dk/presse/`.

[9] BeKey. *Promotional photo.* `https://kommune.bekey.dk/kommune-hjemmehjaelper-produkter/`.

[10] Bits, Please! *Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption.* `https://bits-please.blogspot.dk/2016/06/extracting-qualcomms-keymaster-keys.html`. 2016.

[11] Stefan Brands and David Chaum. *Distance-bounding protocols (extended abstract).* `https://link.springer.com/chapter/10.1007%2F3-540-48285-7_30`. 1993.

[12] Srdjan Capkun, Karim El Defrawy, and Gene Tsudik. *Group Distance Bounding Protocols.* `https://link.springer.com/chapter/10.1007%2F978-3-642-21599-5_23`. 2011.

[13] Matthew Carlson. *Wifi Door Unlocker.* `https://hackaday.io/project/1992-wifi-door-unlocker`.

[14] CHRobotics. *Using Accelerometers to Estimate Position and Velocity.* `http://www.chrobotics.com/library/accel-position-velocity`.

[15] Mark D. Corner and Brian D. Noble. *Zero-Interaction Authentication.* `https://www.sigmobile.org/awards/mobicom2002-student.pdf`.

[16] CSR. *Keyless Entry.* `http://www.csr.com/products/applications/keyless-entry`.

[17] Google Developers. *Tango.* https://developers.google.com/tango/.

[18] DeviceAtlas. *DEVICEATLAS MOBILE WEB TRAFFIC REPORT Q1 2015.* https://deviceatlas.com/sites/deviceatlas.com/files/pdf/DeviceAtlas%20-%20Mobile%20Traffic%20Report%20Q1%202015.pdf. 2015.

[19] Android API Documentation. *getRotationMatrix().* https://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrix(float[],%20float[],%20float[],%20float[]).

[20] Google Android Documentation. *Android API Device Axes Specification.* http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords.

[21] Google Android Documentation. *Services.* http://developer.android.com/guide/components/services.html.

[22] Full Disk Encryption. *Android.* https://source.android.com/security/encryption/index.html.

[23] Mercedes-Benz S-Class W220 Wiki Encyclopedia. *Keyless-Go.* https://w220.ee/KEYLESS-GO.

[24] Kenneth Gade FFI (Norwegian Defence Research Establishment. *Introduction to Inertial Navigation.* http://www.navlab.net/Publications/Introduction_to_Inertial_Navigation.pdf.

[25] Kenneth Gade FFI (Norwegian Defence Research Establishment. *Introduction to Inertial Navigation and Kalman Filtering.* http://www.navlab.net/Publications/Introduction_to_Inertial_Navigation_and_Kalman_Filtering.pdf.

[26] Chen Feng et al. *Received Signal Strength based Indoor Positioning using Compressive Sensing.* http://www.comm.toronto.edu/~valaee/Publications/Feng-TMC-2011.pdf. 2011.

[27] Davrondzhon Gafurov and Einar Snekkenes. *Towards Understanding the Uniqueness of Gait Biometric.* https://www.researchgate.net/profile/Davrondzhon_Gafurov/publication/224401067_Towards_understanding_the_uniqueness_of_gait_biometric/links/00b49528e834ad8b5f000000.pdf.

[28] Flavio D. Garcia et al. *Lock It and Still Lose It On the (In)Security of Automotive Remote Keyless Entry Systems.* https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_garcia.pdf.

[29] Gartner. *Gartner Says Emerging Markets Drove Worldwide Smartphone Sales to 15.5 Percent Growth in Third Quarter of 2015.* http://www.gartner.com/newsroom/id/3169417. 2015.

[30] David Sachs at Google. *Sensor Fusion on Android Devices: A Revolution in Motion Processing.* https://www.youtube.com/watch?v=C7JQ7Rpwn2k#t=23m20s.

[31] GPS.gov. *GPS Accuracy.* http://www.gps.gov/systems/gps/performance/accuracy/.

[32] Frank Van Graas. *Inertial Navigation Systems.* http://indico.ictp.it/event/a12180/session/23/contribution/14/material/0/0.pdf.

[33] The Bluetooth Special Interest Group. *Bluetooth 5 quadruples range, doubles speed, increases data broadcasting capacity by 800%.* https://www.bluetooth.com/news/pressreleases/2016/06/16/-bluetooth5-quadruples-rangedoubles-speedincreases-data-broadcasting-capacity-by-800.

[34] Google Maps Help. *Calibrate your compass.* https://support.google.com/maps/answer/6145351?hl=en.

[35] Jeffrey Hightower, Gaetano Borriello, and Roy Want. *SpotON: An Indoor 3D Location Sensing Technology Based on RF Signal Strength.* ftp://ftp.cs.washington.edu/tr/2000/02/UW-CSE-00-02-02.pdf. 2000.

[36] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. *Wormhole Attacks in Wireless Networks.* https://www.cs.rice.edu/~dbj/pubs/jsac-wormhole.pdf.

[37] IDC. *Smartphone OS Market Share, 2015 Q2.* http://www.idc.com/prodserv/smartphone-os-market-share.jsp. 2015.

[38] Kia. *Kia K900 2016.* http://www.kia.com/us/en/vehicle/k900/2016/galleries/all?file=gallery_k900_2015_interior_27.

[39] Lindsay Kleeman. *Understanding and Applying Kalman Filtering.* http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated3/kleeman_kalman_basics.pdf.

[40] iOS Developer Library. *Background Execution.* https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BackgroundExecution/BackgroundExecution.html.

[41] Hui Liu et al. *Survey of Wireless Indoor Positioning Techniques and Systems.* http://ahvaz.ist.unomaha.edu/azad/temp/sac/07-liu-localization-techniques-indoor-wireless-sensor-networks.pdf. 2007.

[42] Samantha Murphy for Mashable. *Future Phone Displays Could Take Your Temperature, Analyze DNA.* http://mashable.com/2014/07/18/future-smartphone-displays-could-analyze-your-spit-dna/#lYnnoJbrwqq6.

[43] Ryan Miller. *Wifi-based trilateration on Android.* http://rvmiller.com/2013/05/part-1-wifi-based-trilateration-on-android/.

[44] Patently Mobile. *Future Samsung Devices may use Advanced Laser Speckle Interferometric for Monitoring Health Vitals.* http://www.patentlymobile.com/2016/03/future-samsung-devices-may-use-advanced-laser-speckle-interferometric-for-monitoring-health-vitals.html.

[45] James Newsome et al. *The Sybil Attack in Sensor Networks: Analysis & Defenses.* http://repository.cmu.edu/cgi/viewcontent.cgi?article=1041&context=ece. 2004.

[46] LIONEL M. NI et al. *LANDMARC: Indoor Location Sensing Using Active RFID.* http://www.csd.uoc.gr/~hy539/Assignments/presentations/landmark.pdf. 2009.

[47] Novatel. *GPS Position Accuracy Measures.* http://www.novatel.com/assets/Documents/Bulletins/apn029.pdf.

[48] Michael Fitzgerald Nowlan. *Human Identification via Gait Recognition Using Accelerometer Gyro Forces.* http://www.cs.yale.edu/homes/mfn3/pub/mfn_gait_id.pdf.

[49] Qasim Mahmood Rajpoot. *Secure Localization: A Survey.* 2016.

[50] Aanjhan Ranganathan, Boris Danev, and Srdjan Capkun. *Low-power Distance Bounding.* https://www.researchgate.net/publication/261725689_Low-power_Distance_Bounding. 2014.

[51] STANLEY Security Solutions. *K2 PRODUCT CATALOG.* http://www.lsamichigan.org/Tech/K2_Door-Hardware-Catalog-K2.pdf.

[52] Dieter Bohn at The Verge. *Google built a tiny radar system into a smartwatch for gesture controls.* http://www.theverge.com/2016/5/20/11720876/google-soli-smart-watch-radar-atap-io-2016.

[53] Nils Ole Tippenhauer et al. *On the Requirements for Successful GPS Spoofing Attacks.* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.679.2524&rep=rep1&type=pdf.

[54] Viper. *Viper SmartKey.* https://www.viper.com/smartstart/product/vsk100/viper-smartkey.

[55] Wikipedia. *Smart key.* https://en.wikipedia.org/w/index.php?title=Smart_key&oldid=721035007.

[56] Xamarin. *Mobile Application Development to Build Apps in C#.* https://xamarin.com/platform.

[57] Moustafa Youssef and Ashok Agrawala. *The Horus WLAN Location Determination System.* https://www.cs.umd.edu/~moustafa/papers/horus_usenix.pdf.