universiteit hasselt

Master thesis submitted to achieve the master's degree in CS, option Multimedia at Hasselt University

Going Deeper Underground: Using accelerometers on mobile devices to enable positioning on underground public transportation systems

Author:
Thomas Stockx

Promotor:
Prof. Dr. Johannes Schöning

2013-2014

Abstract

Thanks to rapid advances in technologies like GPS and WiFi positioning, smartphone users are now able to determine their location almost anywhere they go. This is not true, however, of people who are travelling in underground public transportation networks, one of the few types of high-traffic areas where smartphones do not have access to accurate position information. In this thesis, we introduce the problem of underground transport positioning on smartphones and present *SubwayPS*, an accelerometer-based positioning technique that allows smartphones to determine their location substantially better than baseline approaches, even deep beneath city streets. We highlight several immediate applications of positioning in subway networks in domains ranging from mobile advertising to mobile maps and present *MetroNavigator*, a proof-of-concept smartphone and smartwatch application that notifies users of upcoming points-of-interest and alerts them when it is time to get ready to exit the train.

Publications

A major part of the research mentioned in this thesis was submitted as a paper to the *2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. Ubicomp is the premier outlet for novel research contributions that advance the state of the art in the design, development, deployment, evaluation and understanding of ubiquitous computing systems. The paper was in the top 23% of all submitted papers and was invited to the "revise and resubmit" track. At the time of publication of this thesis, the final decision was still pending.

Dutch summary

The following Dutch summary was submitted to the Vlaamse Scriptieprijs and is aimed towards an audience aged 16. Technical details are omitted to make the summary more understandable for this age group.

Nooit meer verdwalen in de metro

Introductie

Dankzij de razendsnelle evolutie van technologieën als GPS en WiFi kunnen smartphonegebruikers bijna overal hun locatie bepalen. Ondergronds werken deze technieken echter niet en slaan bestaande *apps* zoals Google Maps de bal helemaal mis.

Nochtans zijn er verschillende toepassingen mogelijk als we de locatie van een gebruiker kunnen bepalen. Om te beginnen zou een gebruiker zijn locatie kunnen bekijken in bestaande apps zoals Google Maps. Stel je dan voor dat dit uitgebreid wordt met een GPS-achtige applicatie voor de metro. Zo kun je perfect je weg vinden in buitenlandse metronetwerken, ook al ben je er nog nooit geweest. En geloof ons, we hebben dit in 4 steden wereldwijd getest!

Ondergrondse localisatie

SubwayPS (of Subway Positioning System voluit) is een nieuwe methode om de locatie van een smartphone te bepalen zonder dat hier extra hulpmiddelen voor nodig zijn. Hiervoor wordt dezelfde sensor gebruikt als bij verschillende games, namelijk de accelerometer. Deze sensor "voelt" trillingen en zwaartekracht en zorgt er ook voor dat je bij race-games bochten kan maken door je smartphone schuin te houden.

Door een paar wiskundige berekeningen uit te voeren op de metingen van deze sensor, kan bepaald worden of het metrotoestel waar je op zit momenteel aan het rijden is of niet. Omdat we met de hulp van uurroosters weten hoe lang het ongeveer duurt alvorens de gebruiker in het volgende station aankomt, kunnen we een relatieve positie berekenen tussen het vorige en het volgende station. Doordat we de uurroosters gebruiken, kunnen we zelfs weten of de metro stopt in de tunnel in plaats van

aan een station. Zo kunnen we de gebruiker altijd een nauwkeurige locatie geven van waar hij of zij zich bevindt.

Android app

MetroNavigator is een voorbeeld app voor Android toestellen en toont aan hoe *SubwayPS* gebruikt kan worden. De app werkt als een soort ondergrondse GPS: de gebruiker kiest een start- en eind-station en kan verder volledig volgen waar hij zich bevindt. *MetroNavigator* brengt de gebruiker ook op de hoogte van interessante bezienswaardigheden in de buurt zoals musea of andere toeristische attracties. Als je de smartphone toch maar liever in je zak laat steken, kan je ook een smartwatch versie gebruiken. Als de smartwatch verbonden is met de smartphone, toont deze dezelfde meldingen.

Testen, testen, testen...

SubwayPS is getest in 3 verschillende studies. In de eerste studie werd gekeken naar hoe nauwkeurig de detectie van stations is. Hierbij werd aangetoond dat 85.8% van alle stations goed gedetecteerd werden. Vergelijkbare technieken haalden tot nu slechts een percentage van ca. 55%. Deze studie werd uitgevoerd in maar liefst 4 verschillende steden: Brussel, Keulen, Londen en Minneapolis.

In de tweede studie maakten verschillende testpersonen gebruik van de *MetroNavigator* app en gaven hierbij hun feedback. Als we in deze studie eerst weer kijken naar accuraatheid komen we uit bij 85.7%, wat heel vergelijkbaar is met de resultaten uit de eerste studie. We kregen feedback als "*Aangezien GPS hier niet werkt, lijkt dit op een GPS voor ondergrondse treinen Ik wil dit in Google Maps!*" en "*Het is echt cool om te zien waar de trein beweegt en stopt Ik voel mij veiliger als ik kan zien dat we dicht bij het volgende station gestopt zijn wanneer we stil staan in een tunnel*". Het grootste voordeel van *MetroNavigator* was de functie om mensen naar hun eindbestemming te leiden (14 van de 16 participanten), en het weten waar men zich bevindt (12 van de 16 participanten). Een enkele participant zei "*Wow, dit is net magie*" toen de visualisatie van de trein stopte wanneer de echte trein dat ook

deed. Ze zei verder: "*... Eerst dacht ik dat deze app een beetje saai was aangezien het gewoon een metro-icoontje bewoog volgens de uurrooster, maar nu zie ik dat het GPS heeft*". In de derde studie werd bestudeerd of er een verschil is tussen het gebruik van *SubwayPS* en het gebruik van de uurroosters. Hierbij werden 50 ritten gemaakt met de metro, waarbij de resultaten van *SubwayPS* vergeleken werden met de uurroosters. *SubwayPS* detecteerde 85.0% van de stations correct, of 78% volledig juiste ritten. Echter, slechts 42% van de ritten was juist volgens de uurroosters. Om dat dit niet eerlijk is ten opzichte van treinen die al met vertraging vertrekken, hebben we ook gekeken naar de relatieve tijd tussen de stations. Het uur van aankomst maakt in dit geval niets uit, zolang de reistijd tussen de stations maar overeenkomt met de uurroosters. Ook hier haalde de uurroosters slechts een accuraatheid van 50%. Deze studie toonde aan dat *SubwayPS* maar liefst 28% nauwkeuriger is dan zich te baseren op gemiddelde reistijd of uurroosters.

Conclusie

SubwayPS is een nieuwe techniek die het mogelijk maakt om de locatie van een smartphone te bepalen op ondergrondse metronetwerken. Enkel sensoren die al aanwezig zijn op een smartphone worden gebruikt, dus er zijn geen extra kosten voor de vervoersmaatschappijen. Er werd aangetoond dat *SubwayPS* werkt in vier verschillende metronetwerken uit twee verschillende continenten, en accurater is dan de officile uurroosters voorzien door de vervoersmaatschappijen.

Acknowledgments

First and foremost, I would like to express my great appreciation to my promotor, Prof. dr. Johannes Schöning, for his valuable and constructive suggestions, especially during the evaluation phase of this thesis.

I would also like to thank Assistant Prof. Brent Hecht from the University of Minnesota, for his help with crunching the Ubicomp '14 paper which has been written about research done in the context of this thesis.

Assistance provided by Prof. dr. Peter Quax for hosting the collected data was greatly appreciated.

Last but not least I'd like to thank my friends and family for their support and encouragement throughout my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and motivation

Localization techniques have been the subject of research for decades. The most popular technique for outdoor localization is the Global Positioning System (GPS). It was developed in 1973 and was created by the U.S. Department of Defense. Originally it ran with 24 satellites and became fully operational in 1995. After a Boeing 747 carrying 269 cilivians was shot down in 1983 for entering the USSR's prohibited airspace, U.S. President Ronald Reagan issued a directive to make GPS freely available for civilian use once it was sufficiently developed [19]. At first the highest quality of signal was reserved for military use while the signal available to the public was intentionally degraded, called Selective Availability (SA). This changed in 2000 when U.S. President Bill Clinton ordered SA to be turned off, which improved the precision of civilian GPS from about 100 meters to approximately 20 meters, the same accuracy that was already available to the military.

The result was a large increase in the production of GPS devices for private use. Multiple companies started to develop GPS receivers that included a navigation component which made GPS available to a much larger audience, like devices created for in-car use.

In November 2004, Qualcomm[1] announced that they successfully tested assisted GPS for mobile phones, in which the startup performance or Time-To-First-Fix (TTFF) of GPS receivers was improved greatly by using data available from the net-

[1] http://www.qualcomm.com/

work. The inclusion of GPS receivers in mobile phones was another great leap in the availability of GPS-enabled devices to a large audience. The arrival of smartphones has seen a huge choice in mobile applications each with the possibility to use the GPS receiver included in the mobile device. Applications which make use of GPS include Google Maps for Mobile[2] and iOS Maps[3] (both map and navigation software by big technology companies like Google and Apple), but also applications like Foursquare[4], Facebook[5] and Twitter[6] make use of the GPS receiver to allow context-aware content based on the location of the user. There are also open source mapping applications available, e.g. OpenStreetMap[7].

The availability of the GPS to such a wide audience has made navigation systems everyday tools. Applications like mentioned above added more possibilities to GPS-equipped mobile devices than just localization. A GPS navigation system was no longer useless outside of a car, but could be used on foot, while biking and even on public transport. The possibilities of devices equipped with a GPS receiver are still increasing.

1.1 Problem

GPS satellites transmit two radio signals named L1 and L2. The L1 signal frequency of 1575.42 MHz in the UHF band is used by civilian GPS devices. Newer satellites and military-grade GPS devices make use of the L2 signal frequency of 1227.60 MHz. However, the system was designed for outdoor locations and therefore has four main problems when trying to use it for indoor localization which are also shown in figure 1-1:

- Line of sight: These signals travel by line of sight which means that they cannot travel through solid objects like buildings and mountains, but can pass through clouds, glass and plastic. This causes GPS to be unreliable in indoor or underground locations.

[2]http://www.google.com/mobile/maps/
[3]http://www.apple.com/ios/maps/
[4]https://foursquare.com/
[5]https://www.facebook.com/
[6]https://twitter.com/
[7]http://www.openstreetmap.org/

Figure 1-1: GPS signals cannot travel through solid objects due do their frequency.

- Multi path signal: Occurs when the GPS signal has been reflected off objects like buildings or other surfaces before reaching the receiver which increases travel time.

- Number of visible satellites: A GPS receiver needs at least four visible satellites to be accurate.

- Interference: Electronic devices like TV antennas which work with UHF signals can conflict with the GPS signal.

Multiple enhancements to GPS have already been developed to allow solutions for signal problems.

Assisted GPS (AGPS) has been developed specifically for mobile phones which can use their cell signal to improve or quicken GPS localization. The AGPS architecture increases the capability of a stand-alone receiver to conserve battery power, acquire and track more satellites thereby improving observation geometry, and increase sensitivity over a conventional GPS architecture. These enhancements come from knowledge of satellite position and velocity, the initial receiver position, and timestamps supplied by an assistance server.

Another GPS enhancement is the use of a custom build GPS antenna which tries to solve the multipath signal problem. A new antenna developed by the Air Force Institute of Technology is able to recognize and ignore multipath GPS signals. However, this design requires a large receiver, making it practical only in military applications[8].

To allow localization and navigation in indoor and underground locations, other hardware than GPS receivers should be used, which should be independent of reception quality and preferably infrastructure independent due to cost.

A smartphone's ability to detect its location is the cornerstone of mobile applications in domains ranging from location-based services to mobile crowdsourcing. While technologies like GPS and WiFi triangulation have made location detection possible in most places smartphone users go, this is not true everywhere. Underground public transportation systems, which are largely inaccessible to GPS signals and often out of range of WiFi networks, represent a particularly important type of space in which smartphones cannot reliably determine their location. As a result, the millions of locals and tourists who ride subways around the world each day [30] cannot take full advantage of popular location-aware smartphone applications - including mobile map apps - while travelling to and from their destinations.

1.2 Goals

In this thesis we want to locate a user in an underground public transport system with a mobile device by using the accelerometer. For doing so, we developed an Android application for mobile devices. In an initial phase, we collected data on how the underground trains travel on a certain track. We then defined methods to differentiate movement and stations and finally used a machine relative approach based on interpolation to track the user's location. The resulting application is shown in figure 1-3.

[8] http://www.technologyreview.com/news/519811/a-cure-for-urban-gps-a-3-d-antenna/

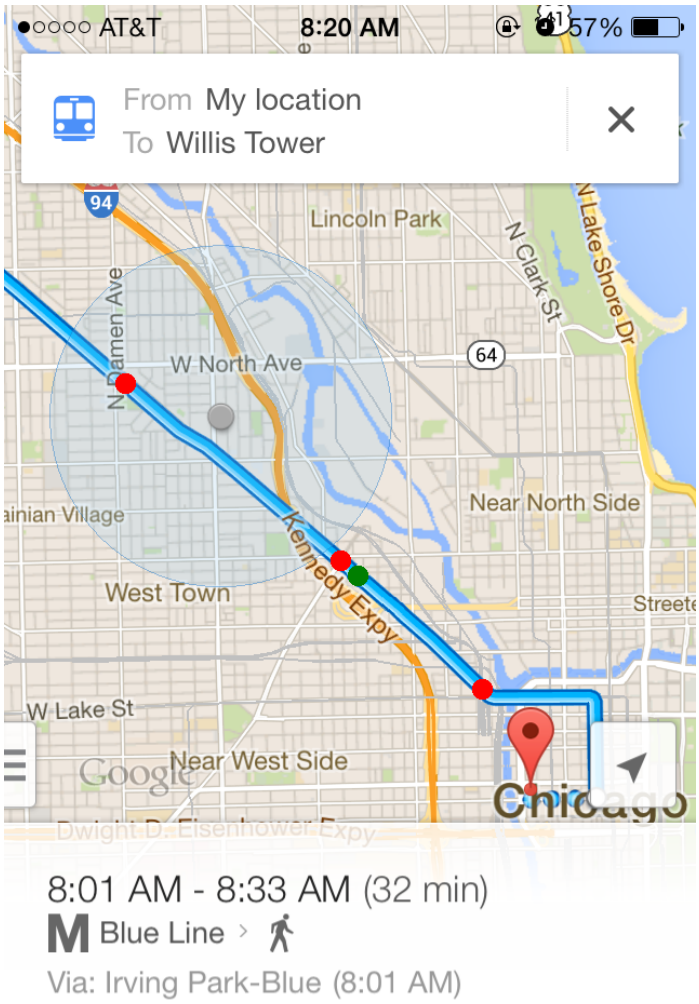

Figure 1-2: A screenshot of the Google Maps iPhone application used on the Blue Line subway in Chicago. The gray dot shows the smartphone's estimated position, while the green dot shows the actual position of the user's train. Red dots are underground stations. One could easily miss their stop if they relied solely on current positioning technology. *SubwayPS* works towards addressing this issue by providing improved positioning while travelling in underground public transport networks (Base map © Google Maps 2014).

Thus, we present *SubwayPS*, the first positioning technique that allows modern smartphones to determine their location while travelling in an underground public transportation network. Our technique, which we demonstrate can achieve reasonable - though not perfect - accuracy, does not depend on any instrumentation of the environment. Instead, *SubwayPS* merely requires an accelerometer and gyroscope and an input origin location and end destination. Accelerometers and gyroscopes are

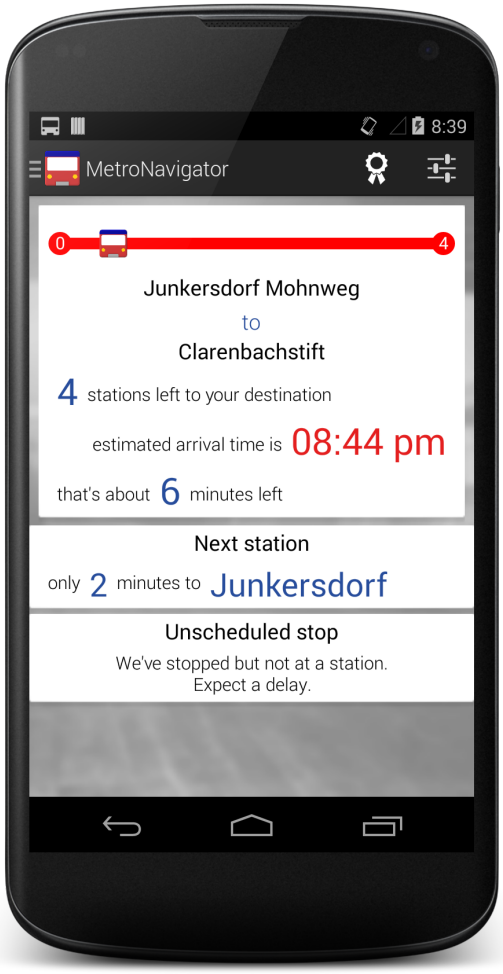

Figure 1-3: Screenshot of the final application.

included in most recent smartphones. Origins and destinations can be easily extracted from routing requests (especially if this type of positioning technology is built into a mobile OS) or inferred from user behavior (as in context-aware technologies like Google Now [10] or similar technologies [16]).

Smartphone positioning in underground public transportation networks allows for wide-ranging benefits. The most basic of those benefits is that *SubwayPS* simply puts subway riders "back on the map". Figure 1-2 shows the result of using Google Maps for iOS while travelling through the underground portion of the Chicago Blue Line.

Because the app cannot obtain position information, it is rendered effectively useless for navigation and orientation purposes.

Our technique can make mobile maps as useful in underground transportation networks as they are above ground. For example, a *SubwayPS*-enabled navigation app could much more effectively help tourists navigate foreign subway systems, especially when they do not speak and/or cannot read the native language. Such an app could, for instance, inform tourists who cannot understand the announcements of when to prepare to exit the train, an especially important concern when the train is crowded. *SubwayPS* can also be used to provide detailed navigation support in complicated subway networks, in which its positioning allows the creation of navigation apps that help users with changing lines.

SubwayPS can also enable location-based applications beyond basic navigation and orientation. For instance, if our technique were built into the location API of a mobile OS, smartphone users would be able to use cached content from apps like Wikipedia or Yelp to easily see the restaurants that are available if they were to get off at the next stop. In addition, mobile advertising becomes possible underground (e.g. "Exit at the next stop to get to restaurant X and receive €5 off").

The core contribution of this thesis is our underground public transportation positioning technique *SubwayPS*, along with three evaluations that show that our technique can achieve reasonable accuracy and substantially outperforms baseline approaches on multiple subway systems.

1.3 Structure of the thesis

This thesis is structured as follows:

In chapter two we put our core contribution in the context of related work. It has been informed by research from three areas: inertial navigation, transportation mode detection, and other approaches to positioning in public transportation networks.

In chapter three we present *SubwayPS* in detail, which is the core contribution of this thesis and allows positioning on underground public transport systems with

reasonable accuracy.

This is followed by chapter four, in which we introduce *MetroNavigator*, a proof-of-concept application based on *SubwayPS*.

In chapter five we demonstrate the accuracy of *SubwayPS* using (1) accelerometer and gyroscope log data collected from four subway systems across two continents, (2) an "in the wild" user study conducted with *MetroNavigator* in the Cologne, Germany subway system, and (3) a third study to compare the application against a time-table only approach.

Finally, we close with chapter 6 and 7, drawing our conclusions and highlighting the many applications of subway positioning in more detail, together with discussion regarding several means by which *SubwayPS*'s accuracy can be improved in future work.

Chapter 2

Related work

The work presented here is informed by research from three areas: inertial navigation, transportation mode detection, and other approaches to positioning in public transportation networks. Below, we will discuss each of these areas in turn, but first we will provide a list of localization methods and technologies so all related work can be easily categorized.

2.1 Categorization of related work

This section consists of two subsections. In the first subsection we describe localization techniques in general. The second subsection gives a brief overview on the most prominent localization technologies for indoor and outdoor use.

2.1.1 Localization techniques

Dead reckoning

Dead reckoning is a localization technique that calculates a user's position based on a previous location and a certain change with respect to the previous location. This change is a certain length over time (speed) in a certain heading (direction). Without any other localization help, dead reckoning will build up a cumulative error in positioning and is therefore not seen as an accurate localization technique. How

longer the technique is used, the larger the cumulative errors will be. If any mistake or error occurs during localization, it will influence all following calculations.

Proximity

Proximity is a technique often used when the technology does not allow to calculate distance between the subject and a known set of points. It uses the concept of *nearness* to this set of points. An easy way to explain this concept is by giving an example. If a method using the WiFi technology does not make use of the signal strength but instead just uses the notion of being connected to a certain access point, the method is based on the proximity technique.

Trilateration

Trilateration is a localization method which uses the measurement of distances using the geometry of circles, spheres, or triangles. It is based on the fact that in two-dimensional geometry, if a point lies on two circles, the circle centers and the two radii provide sufficient information to narrow the possible locations down to two. The same technique can be used in three-dimensional geometry, where it is known that if a point lies on the surfaces of three spheres, the centers of these spheres along with their radii provide sufficient information to narrow the possible locations down to no more than two (unless the centers lie on a straight line). This technique can be used with different types of signals, as long as they provide a measurement of distance (or signal strength). However, it is obvious that this technique is infrastructure dependent.

Triangulation

Triangulation on the other hand involves the measurement of angles to a point from other known points at the end of a fixed baseline rather than using distance directly. The point can be located as the third point of a triangle with one known side and two angles. Triangulation methods were used for accurate large-scale land surveying until the rise of GPS.

Scene analysis

In this technique, a view is examined from a particular point and direction. This can be combined with computer vision to interpret the view and allow localization by calculating where a user is based on his view.

2.1.2 Localization technologies

GPS

In basic GPS operation, trilateration (see subsection 2.1.1) is used to locate the user. GPS receivers calculate their position by timing signals sent by GPS satellites. Each satellite continually transmits messages that include the time the message was transmitted, as well as the satellite position at the time of transmission. These are converted to sphere centers and radii and therefore trilateration is used to calculate the position. More advanced GPS receivers also implement extra techniques like dead reckoning to still provide positioning when fewer than four satellites are visible.

Infrared

One of the first localization implementations was the Active Badge location system, developed at Olivetti Research Laboratory [29], and used diffuse infrared signals. It is based on the proximity method mentioned in subsection 2.1.1. The user can be located by a infrared badge that emits a unique identifier every few seconds. This data is collected by several fixed infrared sensors which are connected to a central server, and the location of the user is defined as near the last infrared sensor that picked up the signal of his badge. Typical to diffuse infrared signals are issues in locations with fluorescent lighting or sunlight because these light sources also emit infrared signals. The effective range of such diffuse infrared signals is several meters, so a lot of sensors are required to cover a large area. It is obvious that this technology requires specific hardware and is infrastructure dependent, so therefore quite costly.

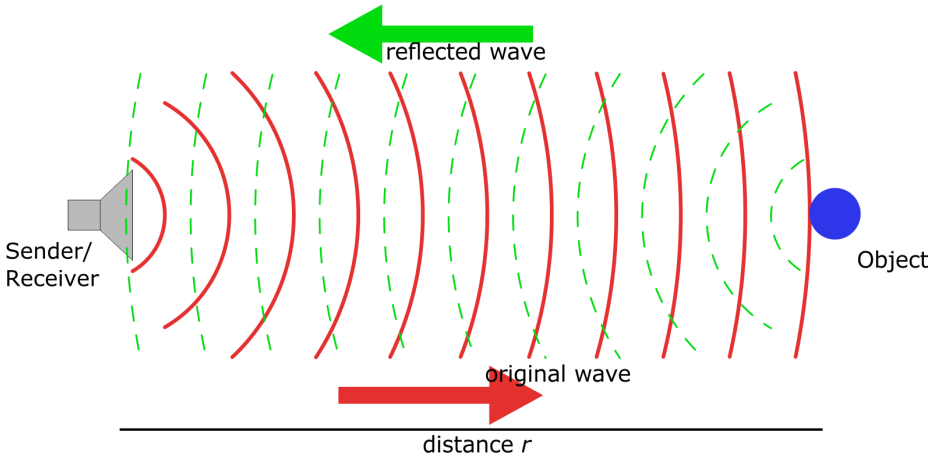

Figure 2-1: Ultrasound waves are bounced off of objects and received after a specific amount of time so distance can be calculated between these objects and the ultrasound sender/receiver.

Ultrasound

An improvement of the Active Badge location system has been developed by AT&T researches, called the Active Bat location system. It uses ultrasound time-of-flight lateration to allow for more accurate positioning by equipping the user with a Bat, consisting of a radio transceiver and an ultrasonic transducer [8]. Ultrasound receiver units are placed at known points in the ceiling, so this system is also infrastructure dependent. Using the times-of-flight of these ultrasound pulses, the distance between a Bat and the receivers can be calculated and its position can be calculated in 3D space using multilateration (see subsection 2.1.1). Disadvantages are the numerous hardware requirements including a base station for sending radio signals, ultrasound receivers and the Bats themselves. Incorrect measurements of distance can also be an issue, caused by reflections of ultrasonic signals in the environment. Finally, no more than one Bat can be located at the same time.

Electromagnetic tracking

Tracking systems which use electromagnetic sensing can sense precise physical positions relative to a magnetic transmitting antenna [21]. These systems generate axial magnetic-field pulses from an antenna that is placed in a fixed location. The user's

position is computed from the position of receiving antennas by measuring response to the field pulse in three orthogonal axes. Of course, the influence of the earth's own magnetic field must be taken into consideration for these computations. Other disadvantages include the implementation costs and the limited range of the technique (up to a few meters). The presence of metallic objects also influences the accuracy greatly. Therefore these systems are not scalable for use in large locations.

Computer Vision

A number of localization approaches are based on the optical signal. Advantages of image matching algorithms are the inclusion of cameras in current generation mobile device hardware and the lack of infrastructure requirements. However, techniques based on this method are not usable in the scope of this thesis. Localization by image matching algorithms is done by interpreting the immediate environment of the user. But since the immediate environment of a user on underground public transport (the inside of a metro car) does not change while movement happens, it is impossible to provide localization. Techniques based on image matching algorithms can be used to determine a users location within a metro car, but not within an entire underground public transport network. Laser ranging techniques on the other hand require specific hardware that is not available for a large audience and are infrastructure dependent. Because of these disadvantages the practical use of methods based on the optical signal is limited. Examples that use these technologies are vSLAM [11] and the orientation part of Kourogi's algorithm[14].

Radio frequency waves including WiFi

Very high frequency radio waves are used in omni-directional ranging techniques by air traffic control. Aircraft can determine their position by receiving radio signals transmitted by a network of fixed ground radio beacons. These ground stations send out a master signal and a highly directional second signal that varies in phase compared to the master. The second signal is timed so that it varies together with the rotation of a secondary antenna. This causes the signal to be out of phase with

the master signal. For example, the signal will be 90 degrees out of phase with the master when the antenna is 90 degrees from north. By comparing the phase of the secondary signal with the master, the bearing of the aircraft can be calculated. By using triangulation and comparing the angle with two different ground stations, the position of the aircraft is calculated. It is rather obvious that the infrastructure for this technique is very expensive, and the range is limited to line of sight. WiFi on the other hand is a very low-strength signal in the radio frequency domain, which causes it to be cheaper but also decreases the range immensely.

Bluetooth

Bluetooth uses short-wavelength microwave transmissions in the ISM band from 2400-2480 MHz[1]. It allows the use of location techniques like trilateration and proximity. One of the latest commercial localization solutions using Bluetooth is the iBeacon by Apple. It uses the proximity technique by showing notifications on a user's mobile device when it is in range of certain iBeacon. Software on the mobile device is able to calculate an estimate distance to the iBeacon based on signal strength. Bluetooth technology is included in most mobile devices but it still requires infrastructure like beacons to function as a localization method.

2.2 Inertial Navigation

SubwayPS uses an approach to positioning motivated by the literature on accelerometer- and gyroscope-based inertial navigation. Although inertial navigation approaches have not been adapted to the context of public transportation positioning as they are here, they have shown to be succesful for other purposes, especially pedestrian indoor navigation (e.g. [1, 6, 22, 23]).

Step detection For instance, Robertson et al. [22] presented a simultaneous localization and mapping (SLAM) approach for pedestrians by using only foot-mounted

[1] http://www.bluetooth.com/Pages/Fast-Facts.aspx

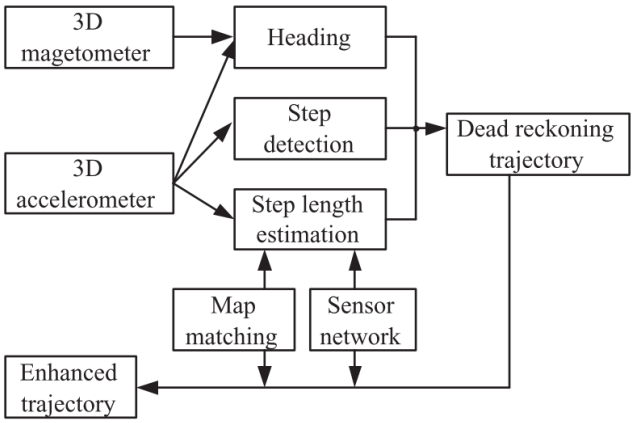

Figure 2-2: The signal flow of the wireless assisted dead reckoning algorithm in the NavMote Experience method. Image courtesy of Fang et al [5].

inertial sensors. Their method, *FootSLAM*, can be used to generate an approximate 2D map of an indoor layout with just 10 minutes of walking. Along the same lines, Grejner-Brzezinka et al. [7] developed a neural network- and fuzzy logic-based inertial positioning system for military and rescue ground personnel. In their implementation the user was equipped with a backpack containing a GPS receiver, an IMU containing a gyroscope and accelerometer, a barometer, and a magnetometer. In addition, the user was equipped with step sensors on the toes and heels to allow for more detailed step detection.

Other accelerometer-based approaches have combined inertial navigation with GPS and other positioning signals for improved accuracy. The *NavMote Experience* [5], for example, combines accelerometer inertial navigation with other techniques to allow for improved accuracy. It involves a Dead Reckoning Module (DR Module), self-organizing wireless ad hoc networks (sensor Mote network) and an information center with a map database.[5]. The DR module includes an accelerometer and a magnetic compass together with a wireless controller board which integrates radio, processing and power. It combines the samples of the magnetometer with the accelerometer to calculate a heading. Steps can be detected by only using the acceleration signal in the z-axis. This signal has double or triple peaks in every step but since the frequency of normal walking is no more than 3 Hz, a low-pass FIR filter or

moving average can smooth the acceleration samples. When the steps are detected, the step length can be calculated with the same z-axis samples. The amplitude of the signal in the vertical axis is influenced by the walking speed. Based on step length and heading estimation the entire trajectory is reconstructed by displaying it on a map of the region. In the implementation the step detection was done on the NavMote while the other calculations were done on the infrastructure. The sensors in the infrastructure were also used to improve the localization of the mobile device by measuring which sensor was closest. Matching the information of the step length and heading on the map was done manually by a human operator by recognizing certain feature in the maps in the data measured by the NavMote. The whole process is shown in figure 2-2.

Another example is the *Embedded GPS/RFID/Self-contained Sensor System* by Kourogi et al,[14]. As described in the name of their system, Kourogi and colleagues integrated self-contained dead reckoning sensors (accelerometers, gyrosensors, and magnetometers) with GPS and an RFID tag system to adjust for errors in positioning and direction that occur in the dead reckoning process. The self-contained sensors are used to estimate relative displacement by analyzing the human walking locomotion by a dead-reckoning method in the Kalman filter framework [13]. Their system uses the GPS in outdoor locations to adjust errors in position and direction that have been accumulated by the dead-reckoning. While indoors it uses the RFID tag system by having tags integrated in key spot areas. This means that this method is infrastructure dependent for indoor environments while being infrastructure independent for outdoor locations since GPS is available there.

A number of inertial navigation-based positioning techniques for indoor navigation and related purposes have been developed for smartphones. These techniques, however, use the detection of walking steps ("step detection"), together with a method of heading determination to provide indoor navigation. Such techniques have seen huge accuracy improvements in recent research but are unsuitable for localization on underground public transport systems where step detection is not an option.

Prominent examples of step detection positioning techniques are for example Serra

et al. [25], who developed an application that calibrates and downloads a map when the user scans a datamatrix with the integrated camera, on which the user's movement is visualized. Other example of smartphone specific work are *PhotoMap* [24] and *PINwI* [17], which both use the integrated camera to take a picture of a map and allow inertial navigation by step detection on the image of the map.

Over time these inertial navigation systems have seen various improvements in recent research (e.g. [12, 17, 18, 24, 25, 26]), allowing them to provide meter-level positioning accuracy without any infrastructure assistance. Li et al. [15] have built such an end-to-end localization system with a decent mean accuracy of 1.5m or 2m depending on whether or not the device is hand-held or in-pocket, but again by improving step detection and heading detection algorithms. The recently announced Google Project Tango also promises to allow indoor positioning with a high precision. Their custom smartphone hardware provides a dead reckoning approach based on integrated accelerometers and gyroscopes and combines them with a computer vision algorithm that uses the integrated camera to notice movement. However, these techniques are unsuitable for localization in underground public transport systems where step detection is not an option. In contrast, our system will focus solely on accelerometer and gyroscope input and use this data to allow localization on an underground public transport system where step detection nor other existing techniques are valid options.

2.3 Transportation Mode Detection

Besides positioning, Wang et al. [28] showed that accelerometers on mobile devices can be used to detect a user's mode of transport (e.g. walking vs. biking vs. driving). In this research, they used a formula for acceleration synthesization to learn the patterns of different transportation modes. We implement their formula in our *SubwayPS* technique by using it to detect if a train is moving or not. Hemminki et al. [9] have recently achieved improved accuracy on transport mode detection by providing a new algorithm for estimating the gravity component and key characteristic

pattern detection, which involves the detection of mode-specific patterns such as step detection for walking and smooth accelerations for vehicle movements.

Since transport mode detection methods have currently not achieved an accuracy for underground public transport detection of at least 80%, our method is dependent on user input to confirm when the user is on underground transport, since the focus of this paper is on positioning assuming that the user is underground, and not on transport mode detection.

2.4 Positioning in Public Transportation Systems

The final area of work related to *SubwayPS* comes from the literature on positioning in public transportation systems. For instance, the *EasyTracker* system by Biagioni et al. [2] provides a location tracker for public transport that uses GPS traces from above ground transit vehicles (busses), but requires all vehicles to be instrumented with equipment such as GPS receivers and inertial navigation sensors. Zhou et al. [31] developed a positioning technique that combines a variety of sensor measurements from the multiple mobile devices owned by the people on an above-ground transit vehicle to calculate an average location. Thiagarajan et al. [27] describe a crowd-sourced transit tracking system intended to improve real-time arrival and departure estimates. As part of this research, Thiagarajan and colleagues looked at subway systems, but their approach is not designed for positioning on mobile devices and requires a connection to a tracking server, which is often not available while on a subway.

To summarize, in this thesis we present the first smartphone positioning technique (*SubwayPS*) for underground public transportation systems that relies solely on the user's mobile device. As a result, our technique does not require the infrastructure [2], connectivity [27], or critical mass of users [31] needed by existing approaches in this area. Instead, *SubwayPS* merely requires a single accelerometer and gyroscope, both of which are built into most modern smartphones. In addition, *SubwayPS* has important privacy benefits: because all calculations are done locally on the user's mobile

device, *SubwayPS* does not share a user's position with any central server or third parties.

2.5 Overview of related work

Method	Technology	Technique	Infrastructure independent	Required hardware	Other limitations	Applicable to underground public transport	Accuracy
FootSLAM	Inertial sensors	SLAM		Foot-mounted inertial sensors			1-2 meters
Multi-sensor personal navigator with adaptive knowledge based system	GPS, Inertial sensors, step sensors, barometer, magnetometer	Dead reckoning, triangulation, Artificial Neural Networks, Fuzzy Logic	✓	Step sensors on heels and toes, backpack containing IMU, GPS antenna, magnetometer and barometer			< 5 meters
NavMote Experience	Accelerometer, WiFi, Cellular	Dead reckoning, triangulation		Navmote device (integrated sensors)	Human operator required		Distance error: ± 3% Heading error: 1°
Embedded GPS/RFID/Self-contained Sensor System	Accelerometer, gyroscope, magnetometer, GPS, RFID	Dead reckoning, triangulation, proximity		Custom-built system, RFID integrated in environment			1–12 meters
Indoor Pedestrian Navigation System on Smartphone (Serra et al.)	Accelerometer, magnetometer, compass	Dead reckoning	✓	Smartphone	Area instrumented with QR tags, working internet connection required		Mean placement error of 3.8%
PhotoMap	GPS	Triangulation, georeferencing	✓	Smartphone	"You are here"-map and calibration required		Average error rate of 2.1%
PINwI	Smartphone embedded inertial sensors	Dead reckoning	✓	Smartphone	"You are here"-map and calibration required		Undisclosed

Method	Technology	Technique	Infrastructure independent	Required hardware	Other limitations	Applicable to underground public transport	Accuracy
Indoor Localization Method using Phone Inertial Sensors (Li et al.)	Smartphone embedded inertial sensors	Dead reckoning	✓	Smartphone	Digital map required		1.5-2 meters
Project Tango	Smartphone embedded inertial sensors, infrared projector/sensor, computer vision	Dead reckoning, proximity, scene analysis	✓	Custom-built smartphone with integrated custom firmware			Undisclosed
EasyTracker	GPS	Triangulation		Smartphones integrated in busses, external server			3 – 5 meters
How Long to Wait?	Cellular	Triangulation		Smartphone	Multiple devices per vehicle required		Arrival time prediction error of around 20 seconds per stop
Cooperative Transit Tracking using Smartphones (underground method)	Smartphone embedded inertial sensors	Schedule-based hidden Markov model	✓	Smartphone	Only applicable to underground public transport	✓	About 55% of stops detected correctly over 4 tracks
SubwayPS	Smartphone embedded inertial sensors	SubwayPS: movement detection + interpolation	✓	Smartphone	Only applicable to underground public transport	✓	About 85% of stops detected correctly on over 136 tracks

Table 2.1: Overview of the most important positioning or navigation methods mentioned as related work

Chapter 3

The *SubwayPS* Technique

With the *SubwayPS* technique, we show that it is possible to achieve reasonable positioning accuracy on underground public transportation networks without the need for any instrumentation of subway trains or tunnels. Our evidence suggests that *SubwayPS* is usable as-is on most underground public transport systems in the world. We also show how accuracy can be improved by fine-tuning certain parameters to the characteristics of a specific underground public transport system.

3.1 Capturing vehicle movement

To realize this underground public positioning on mobile devices and accelerometers, it is necessary to first find out how noisy and useful the accelerometer data of a mobile device actually is. For this purpose, an Android application was created that could capture and store accelerometer data for further analysis. This data was stored as acceleration values in m/s^2 for each axis (x, y, and z) together with the timestamp of the measured accelerations.

3.1.1 Using acceleration data of each axis separately

Splitting the acceleration in three axes (being x, y and z) allows much more factors to be used in a recognition method. By placing the mobile device in such a way

that an axis is parallel with the movement of the vehicle, each acceleration value gets another meaning than just acceleration at a certain time. For example, a positive acceleration in the x-axis can stand for acceleration of the vehicle while a negative value stands for breaking. Negative values in the y-axis can stand for bends to the left while positive values stand for bends to the right. The third axis can be used to identify bumps or movements up and down. This separation of the axes allows for much more factors to identify certain tracks. The limitation of this method is that the device should be hold in the same manner by all users, since the reason why activity recognition works better with the combined accelerations is the inability to perfectly determine the gravity vector. This also affects calculating the orientation of the device. Therefore in this first phase it was the user's task to hold and keep the device in the correct orientation (z-axis parallel with movement of the vehicle, see figure 3-1).

Initial data collection on bus

Due to the absence of underground public transport in the area of Hasselt University, the first collection of data was done on other public transport, namely the autobus. However, using an autobus to capture accelerometer data added a high number of environmental problems, influencing the measurements. These environmental influences included

- the driver, each driver has his own way of driving, taking turns more sharply or accelerating and breaking faster.
- the existence of gears, changing gears causes the acceleration to not be constant and upon switching gears there is even a moment of deceleration.
- traffic, behaviour of other vehicles will influence the behaviour of your own vehicle such as a car breaking before making a left turn while your bus just needs to go straight ahead.

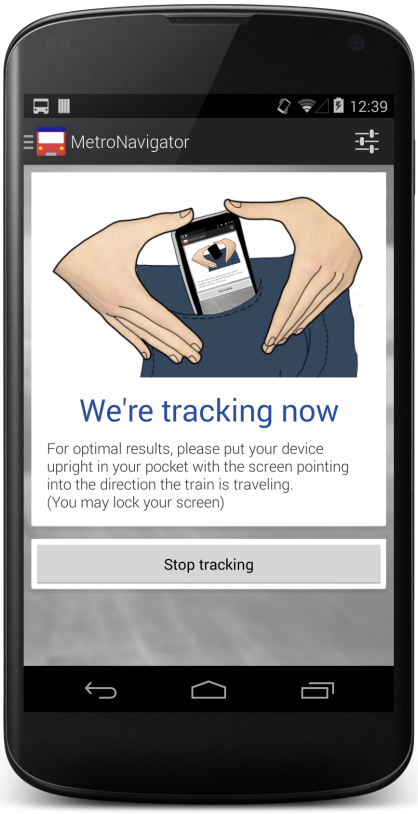

Figure 3-1: Description and image provided by the app to define the required orientation of the device.

- traffic lights which cause the vehicle to come to a halt while there might be no stop nearby. If a traffic light is closely located to a stop it might even be impossible to differentiate between them.

- the bus stops themselves, it is not required for a driver to halt at each stop if there are no people waiting, upon which the data accelerometer will not measure that there is a bus stop.

Due to these problems it is not possible to recognize bus stops by only looking at accelerometer data. We hence updated the mobile application to also store GPS location data so we can match the accelerometer data to certain locations. This made the initial data much easier to process and comprehend.

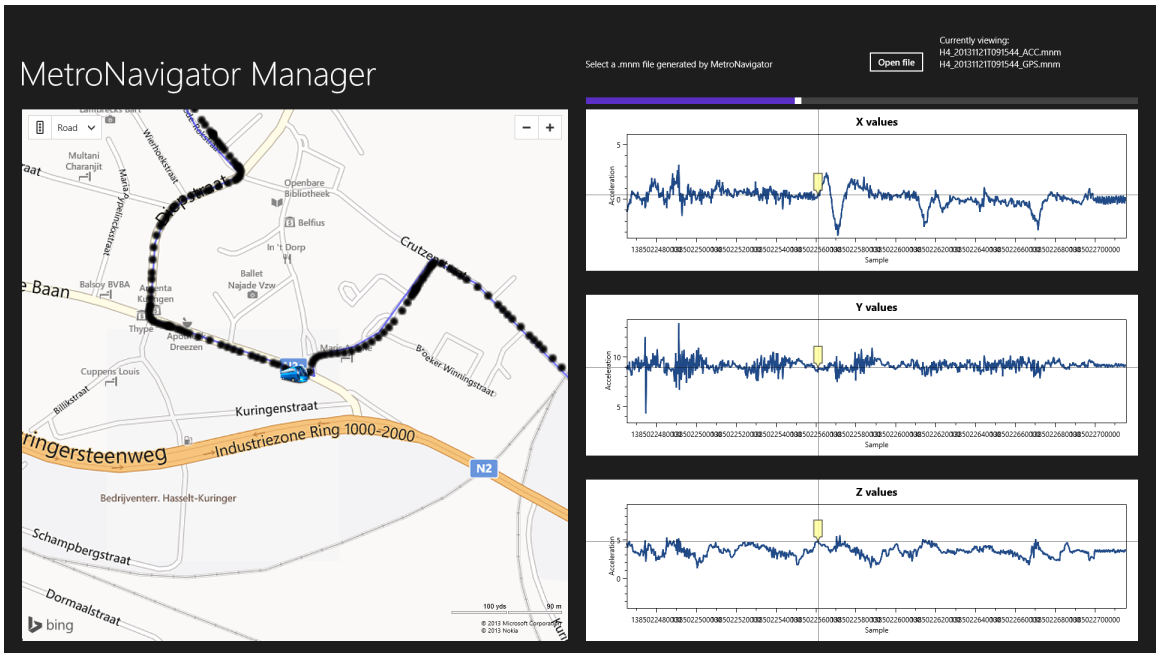

Figure 3-2: Screenshot of the Windows Store application to view and interpret the accelerometer data of public transport by bus.

Interpreting bus data

To interpret the data captured by the first iteration of our application we created a Windows Store application that supported touch to allow for multiple methods of interaction with the data. The application displayed the accelerometer data in three separate graphs, one for each axis. It also displayed a map that shows the tracked route based on the GPS location data. These two views were connected to each other so interacting with one of them also changes the other view. By using this application to visualize the captured data it was possible to see what accelerometer values were measured at an exact location. This connection also worked the other way around so that it was also possible to see where the vehicle was at a specific accelerometer value. A screenshot of this application is shown in figure 3-2.

Due to the limitations mentioned in the previous section, the results were rather noisy and required a lot of manual checks to see if a peak was caused by the route (bends, bus stops) or by traffic causes (red lights). To allow a much faster comprehension of the data, we added a function that changes the color of the data when it

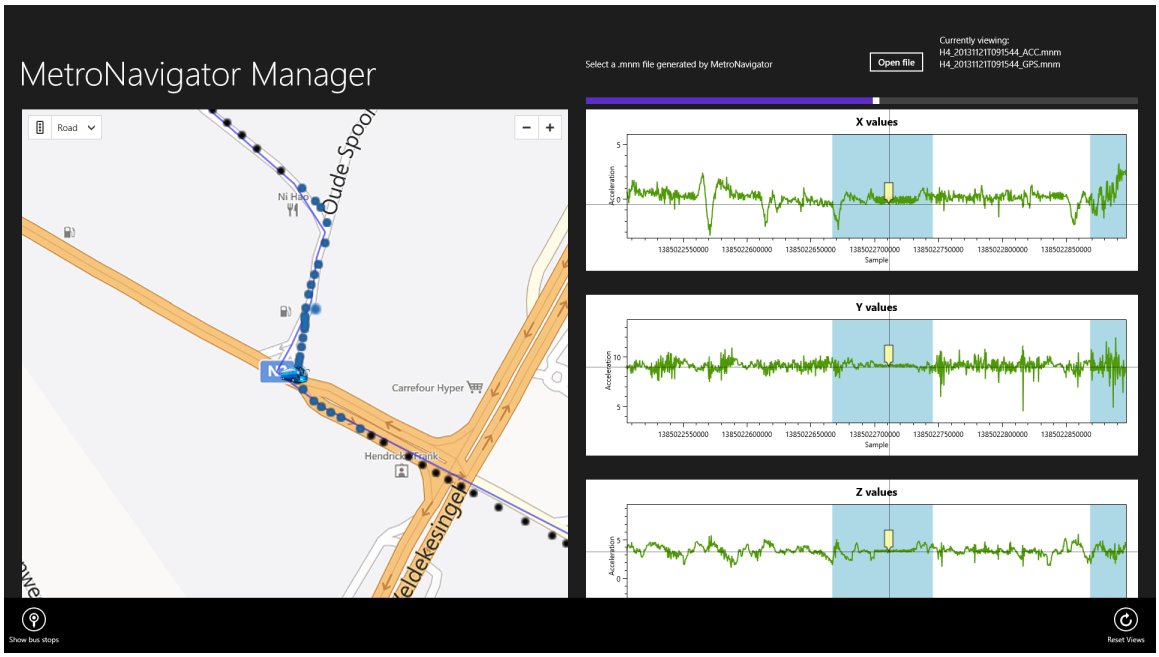

Figure 3-3: Screenshot of the Windows Store application to view and interpret the accelerometer data of public transport by bus, with blue dots depicting bus locations near bus stops.

was within hundred meters of a bus stop. It was now much easier to see if a certain peak was caused by a bus stop or not. A screenshot of this functionality is shown in figure 3-3.

The combination of the accelerometer data and GPS locations provided us with a good reference of positioning of the user. Bends in the route as seen on GPS could be easily mapped on timestamps in the accelerations. The next step was testing the same capture algorithm on underground public transport.

Results on underground public transport

Running the same capture mechanism on underground public transport resulted in several weak points. As mentioned previously, positioning via the accelerometer data was easily matched because there was GPS data available. However, we did not have this data on our underground trackings, making it much more difficult to recognize events and patterns in the accelerometer data. Even worse, the movement of underground vehicles is much smoother. On normal roads, most bends that were usable

for route recognition were bends of 90 degrees, taken at a slow pace after a large brake. However on underground public transport, most bends happen very gradually without any slowdown of the vehicle.

The implementation by using all axes separately was also easily influenced by even the slightest rotations of the device. This caused a change in the gravity vector influencing the measurements. Calculating the orientation on the fly (on a moving vehicle with multiple external forces not including gravity) would be cause of a large amount of overhead. The combination with the slow bends of underground public transport made it even more difficult to differentiate between a small rotational change or a bend, resulting in accelerometer data that was very difficult to read or interpret.

The negative impact of these results proved so extensive that we looked at a different method to use the acceleration data, which we applied directly on public transport. Our previous method proved that accelerometer methods for above-ground localization are not necessarily directly mappable to the underground public transport domain.

3.1.2 Combining the data from all three acceleration axes

According to the research of Wang et al. accelerometer data can be generalized which will result in better activity recognition than interpreting all three axes separately [28]. Capturing vehicle accelerations via this method resulted in more noise and the inability to differentiate between bends, breaking or acceleration made by the vehicle. However, this method is completely orientation independent and should stay usable when the orientation of the mobile device changes. It is also based on the observation that the noise increases on higher speeds instead of taking into account different characteristics of the route.

Underground data collection

To allow for easier testing, we extended our Android application with multiple functionalities that were useful for a testing audience, making it accessible to more people.

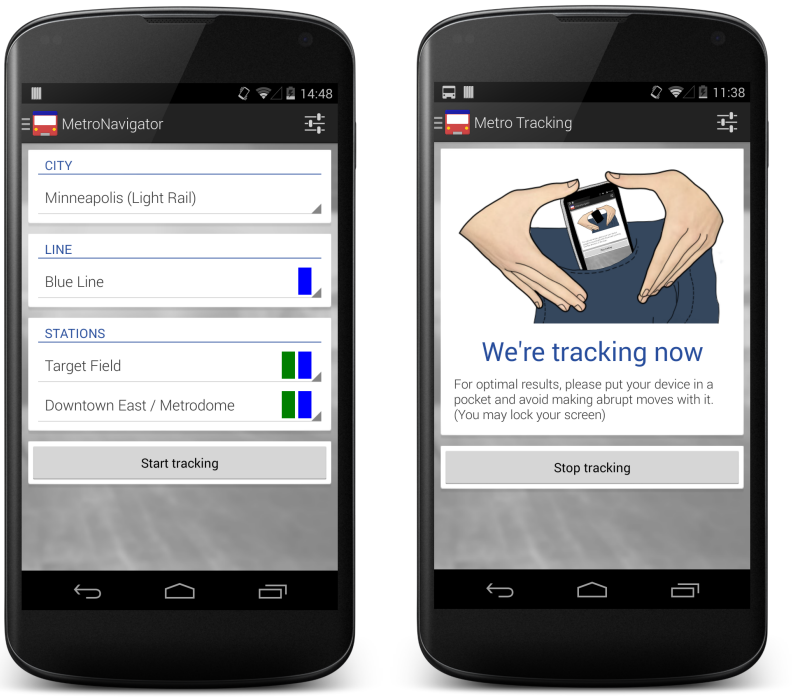

Figure 3-4: Interface of the data capturing application for underground public transport.

The application was updated to include underground transportation information of four different cities namely London, Brussels, Cologne and Minneapolis. This information encompassed all underground lines in each city together with information about all the stations on each line. This allowed the creation of a simple wizard where the user can easily select a city, line, start-, and end-station. A screenshot of the interface can be seen in figure 3-4. The users were still encouraged to avoid making sudden movements so that there would be less noise in the data captured.

3.2 Interpreting measured data

SubwayPS makes use of a smartphone's accelerometer and gyroscope (standard features in recent smartphone models) by combining the accelerometer measurements on all three axes to detect underground train movement. The gyroscope is used to filter

the gravity factor out of the accelerometer measurements, meaning we measure the acceleration applied to the sensor built-in into the device, excluding the force of gravity, in m/s^2. As such, the accelerometer reads ($x = 0m/s^2, y = 0m/s^2, z = 0m/s^2$) when the mobile device is in rest and the coordinate system is defined relative to a default orientation of the accelerometer sensor. The axes are not swapped when the device's screen orientation changes.

In our implementation of *SubwayPS*, we made use of the virtual Linear Accelerometer provided by the Android operating system, which handles the gravity removal internally.

Collecting accelerometer data in a pretest on multiple underground public transport systems showed that station locations overlapped with periods in our data collection where there was generally less acceleration *on all axes of the accelerometer*. This is due to the fact that train movement results in multiple small accelerations in all directions (e.g. think about the small amount of shaking back and forth experienced by passengers when a train is at high speed). As such, by measuring the amount of variance on all axes, it is possible to detect whether the train is moving or not.

Figure 3-5 demonstrates this phenomenon in more detail by showing an example with data collected on the London Underground. Figure 3-5a shows the accelerations measured on each axis of the accelerometer over the same period. Stations are clearly visible in the data in that they have accelerations closer to zero and less variation (around minute 1 and 2.7 in figure 3-5a).

By calculating a general acceleration value based on the three-axis input of the linear accelerometer data it was possible to calculate a variable a that has a large value if there is a large amount of acceleration on any (or all) of the axes. To calculate this value we used the following formula for acceleration synthesization [28]:

$$a = \sqrt{x^2 + y^2 + z^2} \tag{3.1}$$

The resulting a values are smoothed by a rolling average with a window of n samples. We found $n = 100$ samples at 50 Hz to be an appropriate general window. Lower

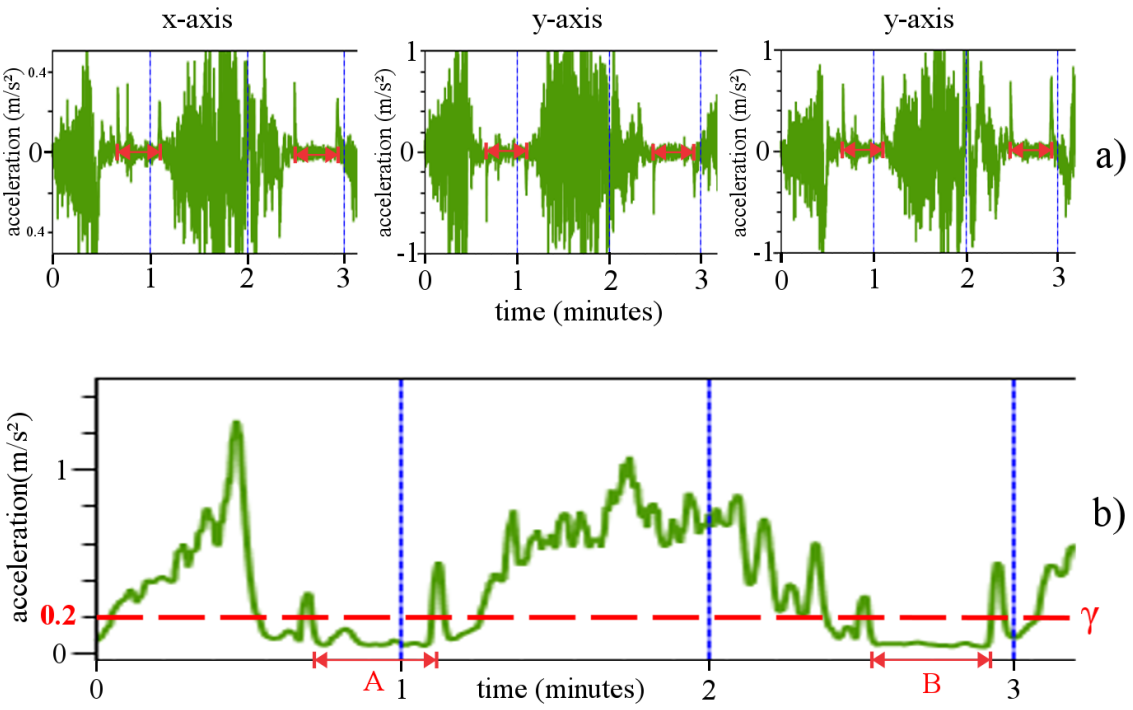

Figure 3-5: In graph a, a visualization of accelerometer values measured on each axis is shown. The blue dotted lines indicate time in minutes while the red intervals indicate stations. The data was collected on the Central Line in London. Graph b shows the result of the acceleration synthesization used in *SubwayPS*, smoothed by a moving average. This result is compared to the threshold to detect train movement.

	World-wide	**London**	**Cologne**
γ	$0.2m/s^2$	$0.2m/s^2$	$0.2m/s^2$
δ_{below}	250 samples at 50 Hz	250 samples at 50 Hz	250 samples at 50 Hz
δ_{above}	350 samples at 50 Hz	250 samples at 50 Hz	500 samples at 50 Hz
n	100 samples at 50 Hz	100 samples at 50 Hz	100 samples at 50 Hz

Table 3.1: General, "world wide" parameters for *SubwayPS*, as well as those specifically tuned for London and Cologne.

values would confuse user movement of the smartphone as false train movement (see Study 3 for more details), while higher values would increase the delay between events (e.g. subway stops) and their detection by *SubwayPS*. Plotting out a results in a graph like that shown in figure 3-5b.

3.2.1 Mapping data on train movement

By comparing the smoothed data with a certain threshold, it is possible to conclude if a train is moving or not. The *SubwayPS* technique uses multiple parameters to parse the data and uses states to define if a train is moving or not, based on a window of samples. The parameters considered are:

$$
\begin{aligned}
n &= \text{window size of moving average} \\
\gamma &= \text{threshold} \\
\delta_{below} &= \text{minimum amount of samples below threshold} \\
\delta_{above} &= \text{minimum amount of samples above threshold}
\end{aligned}
$$

The values for each of the four parameters were determined empirically using data from four different subway systems as described in the evaluation chapter. *SubwayPS* requires at least δ_{below} samples below the threshold before it marks the current accelerometer measurements as a stop, while at least δ_{above} samples above the threshold are required for *SubwayPS* to conclude that the train is moving. These values are used to prevent flagging user movement as train movement (Study 3), but can also improve the accuracy for an individual underground public transport system by slightly adjusting these values based on the characteristics of the trains.

For instance, increased accuracy for the London and Cologne subways can be achieved by using the values in the second and third columns of Table 3.1, respectively. Trains in London accelerated more slowly, resulting in a longer period required to reach δ_{above}, while in Cologne, trains accelerated rather quickly. Because the threshold was reached much faster in Cologne, δ_{above} could be increased to decrease the chance of false positives.

While the "world-wide" parameters were empirically determined to work reasonably well on all subway systems tested (see below), the parameters specifically tuned to the data we collected on each subway system can increase local performance. For example, using the world-wide parameters for *SubwayPS* on data collected in London

results in an accuracy of 74.2%, while using the London parameters increased the accuracy percentage to 85% (Study 1).

3.2.2 Utilizing train schedules

Our stop detection method is complemented by a linear interpolation of the current position based on the time intervals between stations as defined in official timetables. By interpolating over time, we can estimate the absolute location between the previous and the next station. If *SubwayPS* detects a stop that is not scheduled, *SubwayPS* marks it as an "in-between" stop and uses the interpolation of location over estimated time to show a position to the user. Using this method we can provide the users with a position estimate of where exactly the train stopped between two stations. We consider any stop that occurs with less than 70% of the estimated time between stations to be an "in-between" stop, a number that was determined empirically from our Study 1 data. This percentage can be set to a higher value if a timetable is available with seconds-level precision, which was not the case with the four subway systems considered here, all of which list timetables at the precision of minutes.

Chapter 4

Implementation

We implemented a working version of *SubwayPS* on the Android smartphone platform, as well as a proof-of-concept *SubwayPS*-based Android application called *MetroNavigator*, which has both smartphone and smartwatch versions. We first describe our implementation of core *SubwayPS* and then discuss *MetroNavigator*.

4.1 Implementation of *SubwayPS*

To start with the *MetroNavigator* application, an Android implementation of the *SubwayPS* technique was required. This section describes the implementation of *SubwayPS* for Android, together with the difficulties arisen during development.

4.1.1 *SubwayPS* on Android

SubwayPS makes use of the virtual Linear Accelerometer class provided by the Android operating system. It measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z) without the influence of gravity. These accelerations, together with the timestamp of each measurement, were originally stored in a file for each tracking and were measured with an ideal sampling rate of 50 Hz to supply us with data for Study 1.

In our real-time implementation, these accelerometer measurements go through various calculations before being interpreted:

1. A low-pass filter to minimize the basic noise in the accelerometer readings. If a low-pass filter would not be used, even a device in rest would see changes in accelerations due to the inaccuracies of the accelerometer. The code for this low-pass filter can be seen in listing A.3.

2. The readings from the different accelerometer axes are combined using Wang's method for acceleration synthesization [28]. The code for this acceleration synthesization can be seen in listing A.4.

3. A moving average is used to adjust the values. It is important that basic noise is removed from the results, so only noise by train movement is visible in the data. The code for this moving average is can be seen in listing A.5.

These accelerometer measurements are interpreted by the *SubwayPS* technique, which is implemented in a background service and sends *intents* (messages) such as *StationDetected* or *MovingDetected* to the Android OS, which forwards them to all applications that implement a listener for these intents. Any application (e.g. *MetroNavigator*) could implement an intent listener and make use of our detection algorithm. This background service is just a proof of concept. Preferably the *SubwayPS* technique would be implemented on an OS level such as within the location services of Android.

The *SubwayPS* technique is implemented as a state machine in this background service, where the state can be changed depending on accelerometer input. There are four possible states:

1. Maybe no train movement

2. Definitely no train movement

3. Maybe train movement

4. Definitely train movement

The basic states are *Maybe train movement* and *Maybe no train movement*, these are directly influenced by the latest input by the accelerometer. Every incoming sample can change the state to one of these two. If *SubwayPS* stays in a *Maybe* state for a certain period of time (this is a parameter of the *SubwayPS* technique as mentioned in Chapter 3), the state is promoted to its corresponding *Definitely* state. Upon reaching this state the Android OS will be notified of the new state and distribute the notification to every application that wants to use it.

4.1.2 Difficulties during implementation

Each sample measured by the accelerometer is run through a low-pass filter and a few other floating point calculations to synthesize a general sense of acceleration in the combination of all three axes. Afterwards, a moving average algorithm is used to smooth the resulting data, of which the result is used by the state machine of the *SubwayPS* implementation. One of the issues during development was using a sampling rate that was too high for the Android device to keep up with all these calculations in real-time, resulting in a delay of detection and even memory issues when testing during longer periods. However, testing showed that the sampling rate could be easily lowered to 50 Hz instead of the maximum possible sampling rate, without a negative impact on the detection algorithm.

Removing the gravity factor from the accelerometer readings is done by the Android OS itself, since it provides the virtual Linear Accelerometer sensor which handles the calculations. However, this sensor uses readings provided by a gyroscope, which proved unreliable on some devices, especially those of Samsung make. Changing the orientation on devices with such an unreliable gyroscope resulted in large changes in measured accelerations due to the gravity factor not being correctly factored out. This issue is however purely caused by incorrectly calibrated hardware. Utilizing certain quality-labeled devices such as the Nexus-line by Google eliminated these errors.

4.2 Implementation of *MetroNavigator*

MetroNavigator is a proof-of-concept application we built that uses *SubwayPS* as its positioning technology (by implementing an intent listener). In this section, we describe how *MetroNavigator* utilises the *SubwayPS* implementation, how *MetroNavigator* is to be used, and more details about the implementation process.

4.2.1 Utilisation of *SubwayPS*

The *MetroNavigator* application uses these notifications provided by the *SubwayPS* background service and provides a use case for our detection algorithm. It tries to match these intents onto timetable data of the underground public transport system. When stops are detected that are too early to be a next station according to the timetables, the *Definitely train movement* will be interpreted as an "in-between" stop instead of a station.

By differentiating between station and in-between stops it is also possible to visualize the user's position between two stations. Since we know the average time until the next station we show a line to the user, which visualizes the distance between the current and the next station. Upon movement an icon is animated on this line and shows the relative position of the train between the two stations. If an in-between stop is detected, this animation also stops, giving the user a sense of where the train has stopped in the tunnel relative to the previous and next station and thus giving the user a sense of location.

In the current implementation we do not make use of any "real-time" timetables provided by transportation companies, but make use more relative values, namely the average length of trips between two neighbouring stations.

Based on the intents *MetroNavigator* gets from our detection algorithm, it shows event cards to the user such as a "current station" card that shows more information of the current station, a "next station" card if the train starts moving, a "delay" card that shows more information if the train has made an in-between stop, and an "unexpected event" card if something out of the ordinary happens such as a

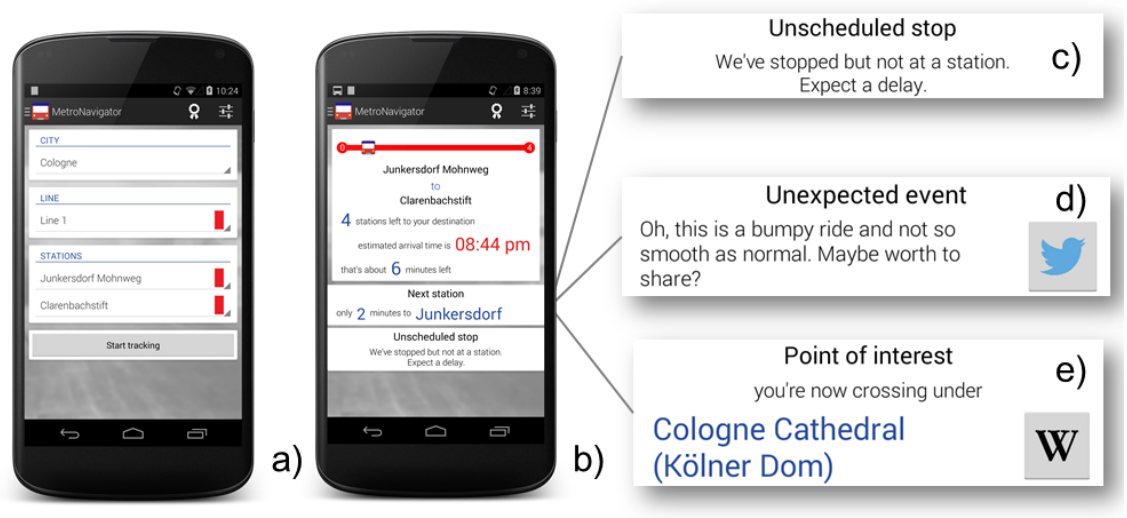

Figure 4-1: *MetroNavigator* application showing different "event cards".

very bumpy ride. These "event cards" are more broadly described in the following subsection.

By providing a location between two stations it is possible to also position a user on an above-ground map. *MetroNavigator* uses this above-ground location to provide POI cards to the user. These can be triggered upon nearing a POI or moving under one.

4.2.2 *MetroNavigator* usage and its interface

The *MetroNavigator* application's UI consists of two main parts. The upper part contains a miniature map with an indicator of the user's current position between the previous and next station, as well as the origin and destination station, the estimated time of arrival, and the number of stops to the end station. This "information body" always stays visible at the top of the UI. Below the information body, in the so-called "information card section", multiple event cards can be shown simultaneously depending on the current position of the user. These cards can be swiped away manually or trigger different third party apps such as Wikipedia or Twitter. Currently our *MetroNavigator* application supports the following different types of event cards:

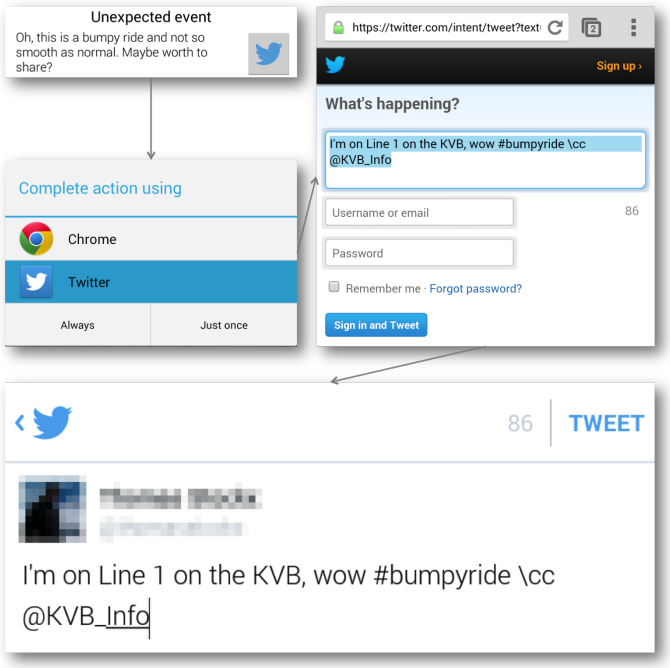

Figure 4-2: Sharing an event card of *MetroNavigator* with a few clicks.

1. The first, and most-often-used type is the notification event card shown in figure 4-1b. When the train approaches the next station, it shows the name of the next station and can indicate possible connections. When the train approaches the final station, this is indicated with a prominent visual design.

2. The application also triggers event cards when the train undergoes unplanned stops in between stations. This is shown in figure 4-1c.

3. Another event card type notifies the user of "rude" driving of the train conductor based on measured accelerations. This is shown in figure 4-2 and one can share this event on Twitter (once connectivity is obtained) with a direct mention of the public transportation company.

4. *MetroNavigator* also contains point-of-interest (POI) event cards that contain information about POIs that are located above the user or at the next station (see figure 4-1e) and provide a link to more information about the POI on Wikipedia, which is locally stored on the device.

4.2.3 Development process

To start with the development of the *MetroNavigator* application, multiple parts of code and UI from the data collection application mentioned in the previous chapter were reused. Small adjustments to that code included:

- Instead of writing accelerometer data to a file they are now saved and used as real-time input for the implementation of the *SubwayPS* method.
- Replacing the static tracking interface as shown in figure 3-1 with an interactive interface showing real-time information.
- Implement methods so the background service of *MetroNavigator* would start and stop automatically instead of depending on user input.

MetroNavigator contains underground transportation information of four different cities, namely Brussels, Cologne, London and Minneapolis. In the proof-of-concept implementation this information is stored in local files, which we will call "information files". However, since this information is stored in seperate files in an assets folder instead of being hard-coded, these information files can also easily be stored on a remote location. Since all this information (both about stations, lines and timetables) is stored in these external information files, *MetroNavigator* can be easily expanded to work in more cities, just by providing more of these information files. One can see a fragment of such an information file in the appendix (see listing A.1).

To allow more detailed detection, timetable info is required. Since it is not an obvious solution to download this information from a remote location while underground due to probably no network being available, this timetable data is also stored in information files. Instead of storing full timetables which would require a large amount of storage and are useless in case of delays or irregular train times, *MetroNavigator* uses another kind of locally stored information files which contain, for each line, all stations and the average relative time between these stations. These information files have been manually created by using official timetables and are in a scale of minutes. A fragment of such an information file can be seen in the appendix, listing A.2. Since

creating these information files with timetable data proved time-consuming, these were created not for entire cities but for certain lines. The *MetroNavigator* application can still function without timetable data, but will detect more false positives since in-between stops will be flagged as stations.

4.3 Limitations of the *MetroNavigator* application

The current implementation of the *MetroNavigator* application uses relative time between two neighbouring stations, based upon official static timetables. This means that they only have a precision in a scale of minutes for most of the public underground transportation systems. This could cause stations being flagged as "in-between" stops instead of as regular stations due to timetable imprecisions. By crowdsourcing we could use the data already captured by our users to improve the precision to a scale in seconds. For example, if official timetables state 2 minutes between stations while the detection algorithm detects a stop at 1 min 40 secs for almost all users, we could systematically lower the relative time used by our application until it averages around the time measured by the detection algorithm.

4.4 Extra functionalities

This section describes extra functionalities for the *MetroNavigator* application to either improve the user experience or allow better marketing in case of a public release of the application.

4.4.1 The *MetroNavigator* Smartwatch Extension

We also implemented a smartwatch version of *MetroNavigator* that can be linked to an Android device running *SubwayPS* via Bluetooth. *SubwayPS* can send messages to the smartwatch application, which are then shown to the user. Due to screen size constraints, the smartwatch version has a much simpler UI than the smartphone

Figure 4-3: Screenshot of the *SubwayPS* smartwatch application that provides basic feedback to the user without needing to take the smartphone out of their pocket.

Figure 4-4: Screenshot of the *SubwayPS* smartwatch application that provides guidance on in which direction to exit the final station to reach a certain POI.

application. It displays only the current or next station to users, as well as the estimated time of arrival. A screenshot is shown in figure 4-3.

The smartwatch version also has a feature not available in the smartphone version: when users reach the final subway station of their trip, the smartwatch version tells users in which direction to go to reach their destination (see figure 4-4). In addition, when selecting a POI in the smartphone application, the smartwatch will show the direction to go to to reach the POI once the user has reached the next station.

4.4.2 Developing an application for marketing on Google Play

If it is planned to distribute an application to a wide audience, there are multiple possibilities to do so. The most popular distribution method is via an application "store". Multiple companies provide application stores for Android devices, and examples of these stores include the Amazon Appstore[1], Google Play[2], and even application stores from different hardware manufacturers such as Samsung Apps[3]. Of these application stores, Google's Google Play is the most used and most popular one, since it comes pre-installed on most Android devices. Google provides many guidelines and utilities for Android application development. If these are utilised in a great application, chances are it could be featured on Google Play, boosting downloads immensely[4]. In this subsection, we will describe certain methods to fully integrate Google guidelines and features which results in a better user experience on an Android device, while also allowing the application to be selected for marketing purposes.

Android UI Guidelines

MetroNavigator is compliant with the Android UI Guidelines and uses multiple design patterns provided by Google. The usage of these patterns guarantees a consistent experience for users.

Such patterns included in the *MetroNavigator* application include:

- the *Action Bar*. An *Action Bar* provides several key functions, it makes important actions prominent and accessible in a predictable way. In *MetroNavigator*, these actions include a shortcut to the Settings menu (which was used a lot during the data collection phase), and a shortcut to the Achievements (more on this later). The *Action Bar* also supports consistent navigation by allowing direct access to the *Navigation Drawer*.

[1] http://www.amazon.com/mobile-apps/b?node=2350149011

[2] https://play.google.com

[3] http://www.samsungapps.com

[4] http://techcrunch.com/2012/01/04/android-markets-featured-apps-seeing-explosive-download-numbers

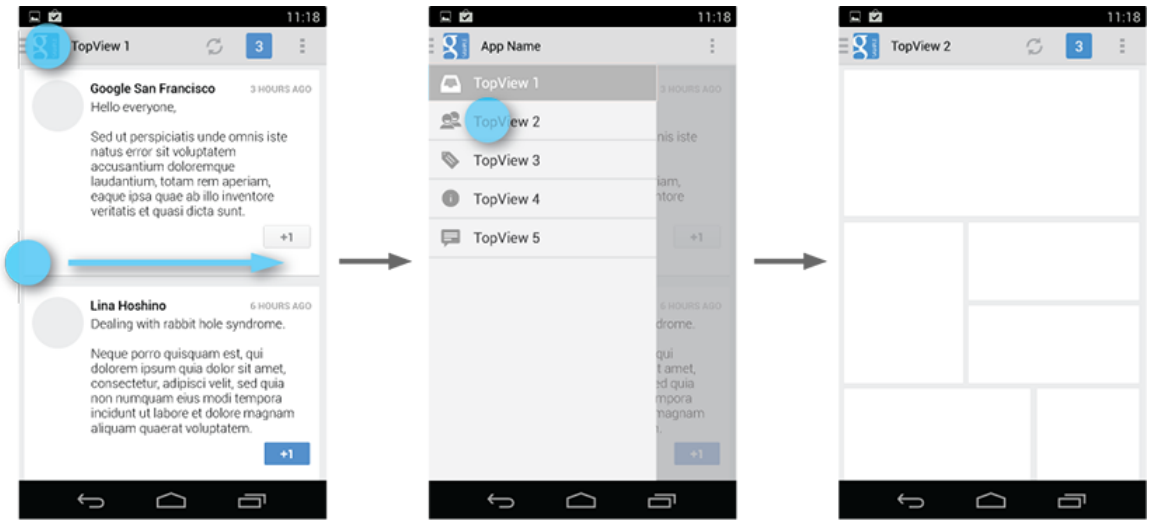

Figure 4-5: Usage of the *Navigation Drawer*. Images courtesy of Google.

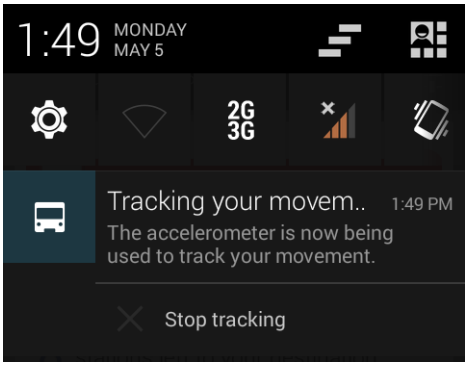

Figure 4-6: Screenshot of the *Ongoing Notification* used by *MetroNavigator*.

- the *Navigation Drawer* is a panel that transitions in from the left edge of the screen and displays the app's main navigation options. As it expands, it overlays the content but not the *Action Bar* and can be brought onto the screen by swiping from the left edge of the screen or by touching the application icon on the *Action Bar*. An example of a *Navigation Drawer* can be seen in figure 4-5.

- An *Ongoing Notification* is used to inform the users about the tracker running in the background. These kind of notifications cannot be removed manually from the notification drawer, but the notification implemented in *MetroNavigator* contains an *Action*, meaning it contains a button to stop the background service from the notification drawer directly. This notification is shown in figure 4-6.

- *Settings* is a place in the app where users can indicate their preferences for how *MetroNavigator* should behave. Using a seperate settings menu is recommended due to two Android design principles:

  - **Decide for me but let me have the final say**: Avoid interrupting the user with the same questions over and over when certain situations arise. The settings predetermine what will always happen in those situations.
  - **Let me make it mine**: The user feels at home and in control.

  The Settings can be seen in figure 4-7, and includes the following:

  - Email address of the user: This was used during the initial data collection phase. Volunteers who collected data with the application were eligable for an Amazon coupon. This setting was implemented before the Google+ login (as mentioned in the following subsection).
  - Default city: The user can select a default city here, which will cause the UI of MetroNavigator to always preselect this city instead of a default value.
  - Enable GPS: *MetroNavigator* and the data collection application contains a feature to track movement in an underground public transport system not implemented in the application. However, since there are several parts of underground public transport systems that run above ground for some periods of time, an option was provided to let the application annotate the accelerometer data with GPS data when available. GPS is only used when tracking transport systems not included in *MetroNavigator*, meaning other cities than London, Cologne, Brussels, and Minneapolis.
  - Clear data: Selecting this option will clear all collected data that has been locally stored on the phone. This data is always saved locally, since there is no guarantee for a working internet connection while travelling underground.

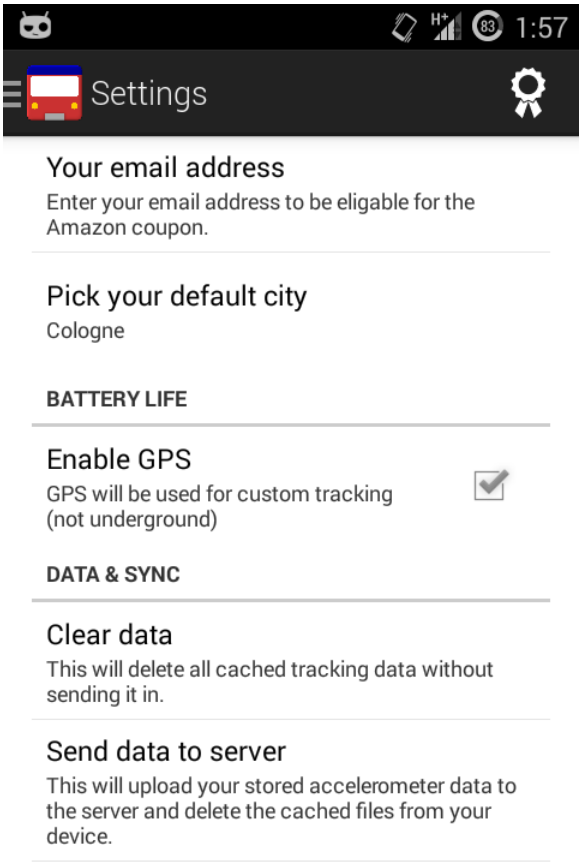

Figure 4-7: Screenshot of the Settings of *MetroNavigator*.

  - Send data to server: This option was used during the data collection phase to send all stored accelerometer data to a server, where it could be interpreted to improve the detection algorithm. It can also still be used to upload data-sets when the detection algorithm made mistakes, so the cause of the mistake can be researched.

- *Pure Android*: *MetroNavigator* is built to fully integrate in an Android device. It is completely themed like a stock Android application, uses the stock platform-specific icons, and most importantly, uses the *Share Action Provider* to integrate with other applications. Links to other applications are not hard-coded but make use of Android's intent API. For example, sharing a bumpy ride through Twitter will launch the user's preferred Twitter application instead of

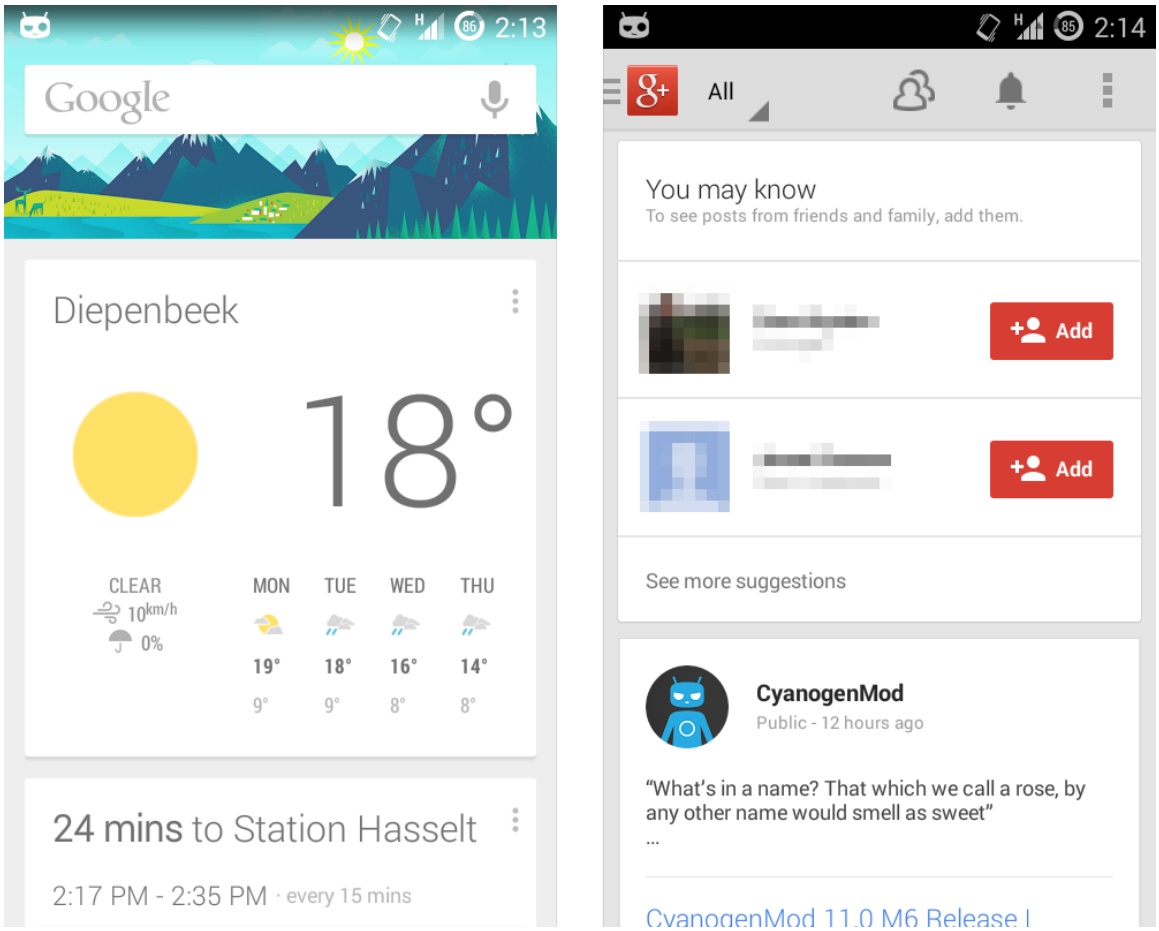

Figure 4-8: Example of the cards UI in Google Now and Google+.

only the stock Twitter client. This can be seen in figure 4-2. If no Twitter client is installed, the user's preferred browser will open to the Twitter website with the tweet already filled in automatically.

- *MetroNavigator* also implements Google's novel *Cards UI*, a custom layout used by many Google applications running on the Android OS. A card contains one piece of specific content, and these cards can be manipulated by swiping them away. The most clear usage of such a card UI is in Google Now and the Google+ applications (figure 4-8). For comparison we refer to figure 4-1 where the *MetroNavigator* layout has already been shown.

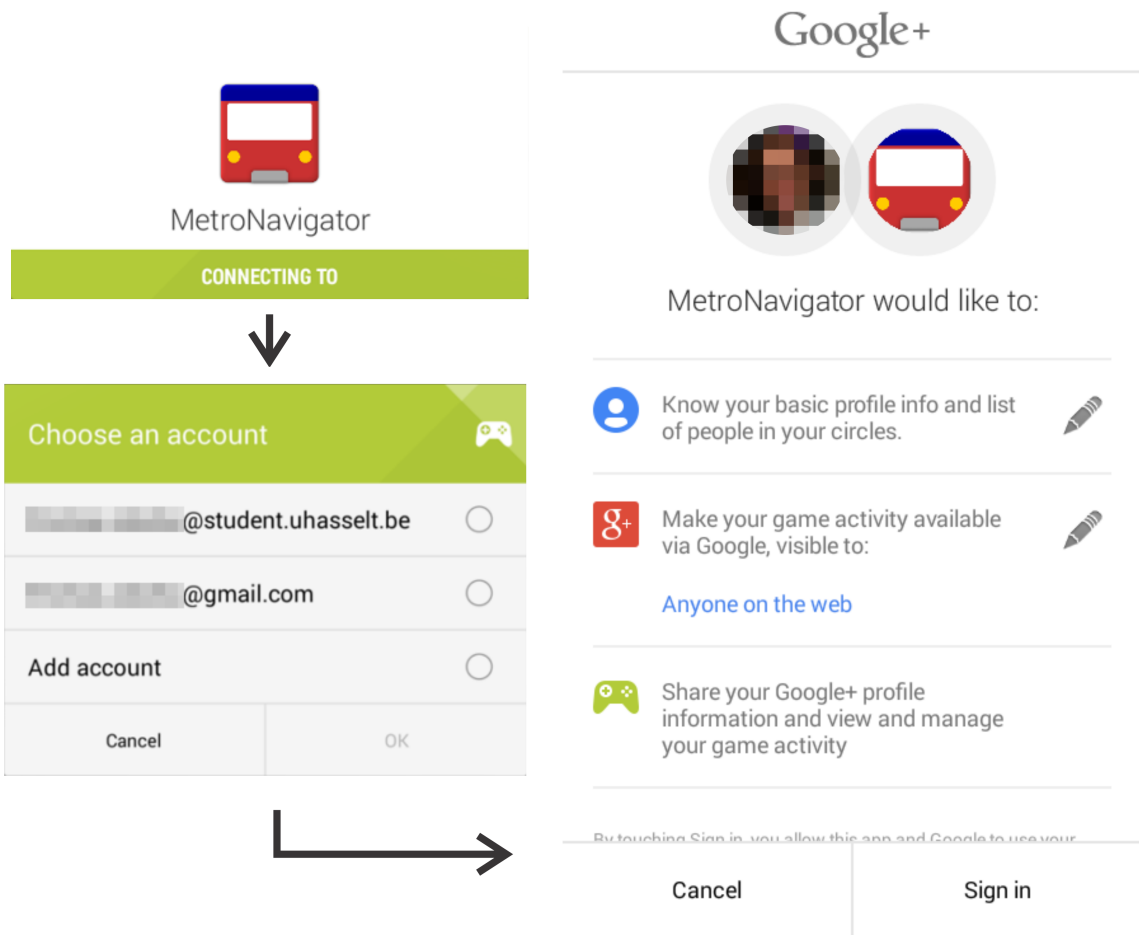

Figure 4-9: Progress of the Google+ Login implemented in *MetroNavigator*.

Google Play Services integration

Google also provides multiple services to allow better integration of an application with the Android ecosystem and Google services. This section describes a few of these services that have been tested and integrated in *MetroNavigator*. It should be stated however, that these services are not required to use *MetroNavigator*, and these have been tested purely for marketing purposes for the eventual public release of *MetroNavigator* to a wider audience.

Google+ Login: Mentioned in the introduction of this paper, it would be great for an application like *MetroNavigator* to "learn" from the user's behaviour. This would for example no longer require the user to manually enter a start and end

location, since these can be inferred from user behaviour, as in technologies like Google Now or similar context-aware recommender systems. To use such existing methods like Google Now, a user should be recognised by the *MetroNavigator* application. On Android, one of the most user-friendly methods to create user accounts for an application is to use an existing Google account, because for being able to use online services of an Android device, one needs a Google account. By integrating this Google account in *MetroNavigator*, it would be easier to access information from a user's Google account, such as routing data stored in Google Maps or Google Now, if Google chooses to make it available for third-party developers. This login system also avoids forcing the user to make "yet another" account on an application/service, which might be cause of a barrier to use *MetroNavigator* as mentioned in the evaluation, study 2. This Google+ login only needs to happen once and is afterwards remembered for the full period while *MetroNavigator* is installed on the device.

Last but not least, utilizing this Google+ login would allow *MetroNavigator* to sync data between multiple devices of the same user. A screenshot of the login process is shown in figure 4-9.

Achievements To allow for more "viral" marketing via social media and word of mouth, achievements were implemented. These complement the raw informational usage of *MetroNavigator* by adding a gamification factor, challenging the user and pushing them to keep using the application, while also allowing sharing these achievements on social media to allow for viral marketing [4]. A screenshot of some implemented achievements can be seen in figure 4-10.

Beta testing Google Play Beta-testing and staged rollouts were used to distribute *MetroNavigator* and the data collection application to our testing audience. It allows creating multiple communities which can each access a different version of the same application. For example, the "Alpha-testers" group received the data-collection application upon installing *MetroNavigator* through Google Play, while members of the "Beta-testers" group received the real working implementation of *MetroNavigator*.

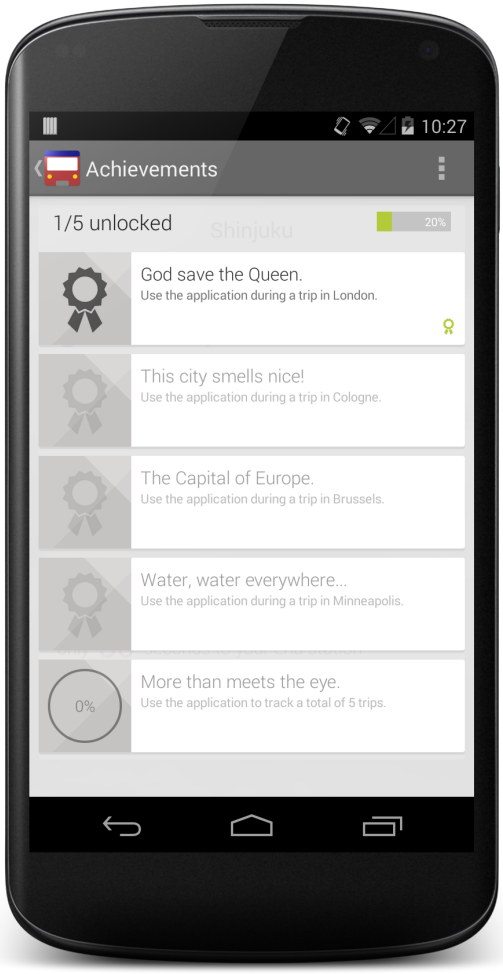

Figure 4-10: Some implemented achievements in *MetroNavigator*.

In the following evaluation section, study 1 users were part of the "Alpha-testers", while the *MetroNavigator* version for study 2 was distributed to the "Beta-testers".

To manage these groups, Google+ communities were used for their synergy with Google Accounts and easy methods of feedback.

Chapter 5

Evaluation

To evaluate *SubwayPS* we conducted three separate studies with the help of the *MetroNavigator* application. The first study, our technical evaluation, was focused on assessing the technical soundness of the positioning technique in a variety of public transportation systems around the globe using log data. The second evaluation, our user study, was focused on understanding the accuracy and user experience of the *MetroNavigator* application in the context of actual subway journeys made by locals and tourists. The third study was designed to compare *SubwayPS* against a timetable-only baseline and to evaluate the effect of arbitrary movements of the smartphone by users.

5.1 Study 1

To goal of the first study was to collect accelerometer data from a variety of diverse subway systems and use this data to inform and evaluate *SubwayPS*.

5.1.1 Participants & Apparatus

For the data collection process, we developed an Android application that captures and stores the data measured by the accelerometer sensor of the mobile device. The application runs on Android devices with Android OS 4.x and higher. The data

collection application supports the public underground transportation networks of four major cities, namely Brussels, London, Cologne, and Minneapolis. Each of these subway systems has unique characteristics: they use different trains, some go above ground, and others merge with vehicular traffic. We wanted to study a diverse set of subway systems in order to support broader generalizability.

To record data, participants first selected a city, a line, and an origin and destination station as seen in figure 4-1a of the *MetroNavigator* application. Please note, that the users did not have access to the features provided in figure 4-1b - 4-1e, as this first phase was done to focus on the robustness of our approach. Participants were advised to select the start and end station of their journey before boarding the train and were told to mainly place their phone in their pockets.

We recruited twelve participants that downloaded the data collection application to their mobile device and asked them to record data "whenever they use an underground public transport system" and also to record very long tracking segments to get tracks that are longer than normal underground rides, so our algorithm could be tested exhaustively (21.51 minutes trip length vs. about 9 minutes in the follow up studies). Most of the participants were Android developers and computer science students who collected the data on a voluntary basis on their daily commutes or on business trips. These users collected a total of about 70 tracks in a period of around two months and we received tracks from every underground transport system supported by the application.

To interpret and analyse the data captured, we created a Windows Store application that displays the accelerometer data in three separate graphs, one for each axis, similar to figure 3-5b. A full screenshot of this application can be seen in figure 5-1. It also displays a map that shows the tracked route based on the entered start and end location of the journey. The graphs and map are connected to each other so interacting with one of them also changes the other view. By using this application to visualize the captured data it was possible to see what accelerometer values were measured over time. The values and our interpretation of them have already been explained in the previous chapter about the *SubwayPS* technique.

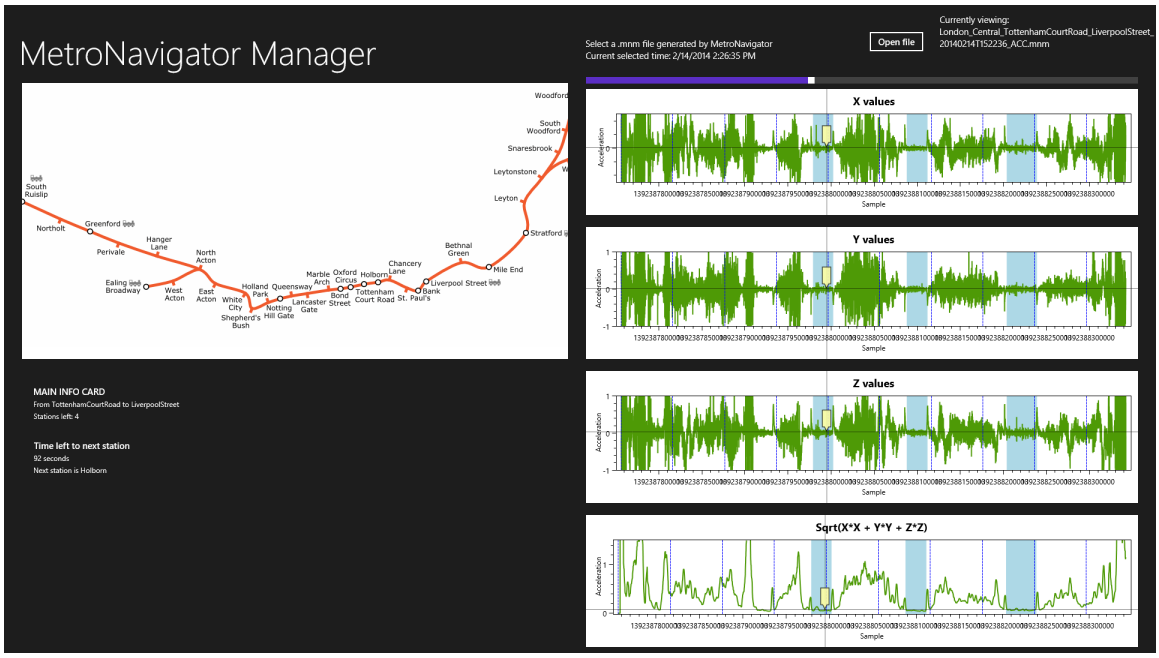

Figure 5-1: Windows store application used for Study 1. In the top left, a map of the subway line is shown. On the right, the accelerometer data is visualised in graphs similar as in figure 3-5. Finally in the bottom left corner, "event cards" are shown as if the data was being handled in real time by the *MetroNavigator* application.

5.1.2 Results

The average number of stations per track was 9.23 and the average tracking length was 21.51 minutes. We analyzed this data to determine the values for each of the *SubwayPS* parameters described in chapter 3 (e.g. δ_{below}, δ_{above}). Using the subway system specific values as shown in Table 3.1, we found that for London, 103 out of 120 stops (105 stations and 15 "in-between" stops) were classified correctly (85.8% excluding the start stations). 10 stations and 7 "in-between" stops were missed. No false positives (detection of a stop where there was none) were recorded.

This can be directly compared to the results of the same tracks tested with the "world-wide" parameters, which caused *SubwayPS* to correctly detect and classify about 74.17% of these stops.

Similar results (around 85% in comparison to around 75% for "world-wide" parameters) were measured in the other three subway systems.

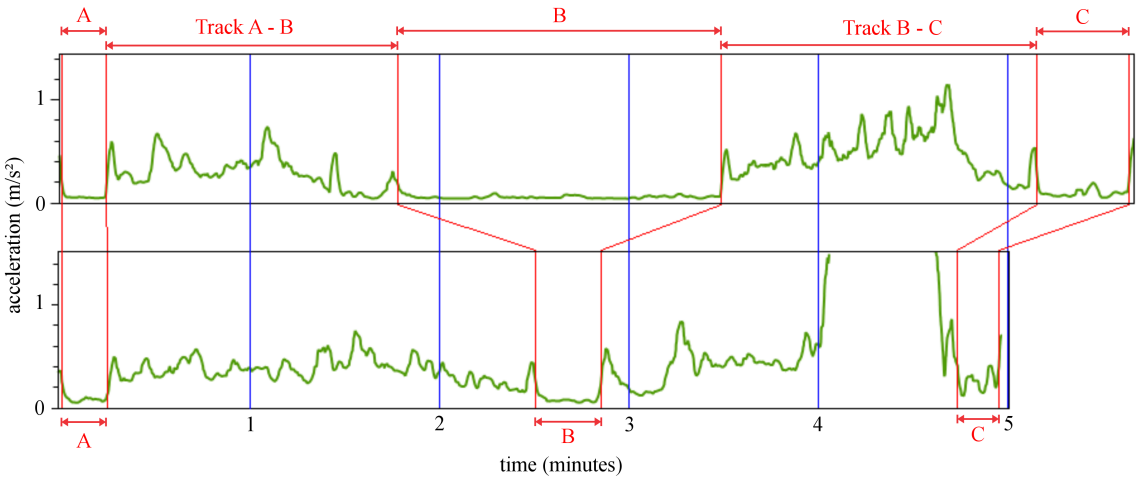

Figure 5-2: Example of day-to-day variability measured on the Piccadilly Line in London. Both graphs picture the data calculated by *SubwayPS*. Stations A, B, and C are clearly visible. These graphs provide a great example of how much variability there is in the measurements. Times between the same two stations can vary greatly (see Track A-B) with a difference of almost one minute. The period during which a train is halted at a station can also vary greatly (see B).

These results are somewhat comparable to those of a study done by Thiagarajan et al. [27]. Using four tracks in the subway system of the Chicago public transportation network (as opposed to our 70 tracks across four subway systems), Thiagarajan et al. were able to achieve around 55% accuracy (compared to our 85.8%), although it is important to note that their focus was on station detection rather than location prediction (which involves detection in-between stops). In Study 3, we show that *SubwayPS* also strongly outperforms a "timetable" baseline.

Examining where *SubwayPS* failed, we noticed that the trains run "smoother" on certain lines (such as parts of the Hammersmith & City line in London) than others in a given subway network. Because *SubwayPS*'s parameters were tuned on a system-wide basis, this variation resulted in errors throughout a given subway system. As noted later, future work could include implementing track-level parameters, which should increase accuracy by a substantial margin.

Another issue arose out of huge variations in travel time for trips between the same two stations. We recorded multiple readings over multiple days for some lines

and noticed a standard deviation of about 7.12 minutes for a trip that normally takes 29 minutes. We looked into this particular case, and found out that this was due to extensive waiting times at stations when trains were overcrowded. These delays could cause misclassification due to the integrated time-tables, as "in-between" stops could be classified as a station if the trip takes too long.

An example of these variations in trip length is visible in figure 5-2. The graphs depict two trips between the same stations and differ greatly in the in-between measurements.

Finally, another source of error was the quality of accelerometer data. For example, badly calibrated gyroscopes in all data captured by Samsung devices resulted in incorrect measurements by the virtual Linear Accelerometer (e.g. accelerations of above 1 m/s^2 on some axes while in rest). It would be straightforward to correct for these errors on a device-by-device basis if these errors are known a-priori.

5.2 Study 2

To test the "in the wild" feasibility and appeal of *SubwayPS* and its proof-of-concept implementation in *MetroNavigator*, a user study was conducted. *MetroNavigator* was pre-installed on a Google Nexus 4 device (as the device has a rather good quality accelerometer and gyroscope built in) and the device was handed out to randomly selected passengers waiting at different station in the Cologne, Germany subway system(*Kölner U-Bahn*, see figure 5-3). Due to novelty effect concerns and the delay induced by training participants to use a novel UI, we focused on the smartphone version of *MetroNavigator* (instead of the smartwatch version) and restricted our participant population to subway riders who were already using an Android device.

Sixteen participants (8 male, 8 female) with an average age of 39 years took part in the study. Eight participants were employees at different local companies in Cologne, seven participants were tourists, and one participant was a student at a local university. The study was conducted across two days between 8 and 10 am (peak travel time) within the period of a week in February 2014.

Figure 5-3: Photograph taken at a station of the Cologne (Germany) underground public transport system where the user study took place.

5.2.1 Procedure

The experimenter approached passengers that were using Android devices at one of the selected subway stations to invite them to participate in the study. If they agreed, the experimenter read a quick script that explained the purpose and functionality of *MetroNavigator* and informed the participant that the experimenter would be riding with him/her to their final destination. Participants were then asked to enter their start and end subway stops and were also requested to "think aloud" when interacting with the application. The script was kept short as we did not want to cause our participants to miss their train. The experimenter boarded the train with the participants and recorded their reactions during the ride.

Due to the relatively short nature of subway trips and our desire to have participants engage with the application, the participants were asked to hold the device in their hands during the trip and not store it in their pockets. The robustness of *SubwayPS* to normal user movement is established in Study 3. We decided to follow this procedure as otherwise the people would just occasionally look at the device in their often short travel and we wanted to collect their feedback on the application and its functionalities. While in the future, we envision that the people would store the device in their pockets and just use it a couple of times during a ride, or use the

smartwatch extension. After leaving the train, participants filled out a background questionnaire about their age, gender, and occupation, and the experimenter conducted a semi-structured interview. The experimenter also explained how the system derives position information by using the accelerometer of the smartphone and the smartwatch extension was explained to the participants to get feedback on its possible uses.

5.2.2 Results - Accuracy

The participants took the train for an average of 6.37 stations (including the start and end stations) and the average time on the train was about 8.75 minutes. This was in contrast to our data collectors of the first study who were instructed to stay on the trains for longer periods to collect more continuous data sets. 78 out of 91 stops (80 stations and 11 "in-between" stops) were classified correctly (85.7% excluding the start stations; 87.9% also including the start stations). Eight stations and 5 "in-between" stops were missed. No false positives were recorded. These results are in line with our previous obtained data of the first study.

5.2.3 Results - Qualitative feedback

During the interviews, 15 participants expressed a positive opinion of *SubwayPS* and were interested in using *MetroNavigator* (or other applications using its *SubwayPS* engine) in the future. We received comments like "*As my GPS does not work here, this is like a GPS for underground trains - I want to have it integrated in Google Maps*" (P3) or "*It is just cool to see the train moving and stopping on the map - I feel safer when I can see that we have stopped close to the next station when we're stopped in the middle of a tunnel*" (P11). The main advantage of *MetroNavigator* for most participants was the ability to be guided to their destination stop (14 out of 16 participants) and the general location awareness (12 out of 16 participants), both of which were mentioned by more than 75% of the participants and both of which are core *SubwayPS* functions. "*Wow - This is like magic*" (P3) was the comment of

one participant when the visualization of her current position stopped between two stations as the train did. She commented "*... at first, I thought this app is a bit boring as it just moves a metro along with the schedule from station to station, but now I see, that it has GPS*".

Participants also enjoyed specific features of the *MetroNavigator* app. In particular, several of the tourists found value in *MetroNavigator*'s POI information cards "*as (they) can easily build local knowledge of a city*" (P2). The smartwatch was generally thought to be a useful extension for such an application. One participant remarked "*as I do not want to look at my smartphone all the time, this (SubwayPS) could be the killer app for smartwatches in the future*" (P13, an IT consultant).

During the ride of one participant, the *MetroNavigator* application missed three stops in a row. She commented on that by saying "*Oops, it seems that the system missed a stop - that is not good. I would also be happy to help the system to detect the stops, if the system could help me to get out at the right station. That would be totally fine with me*" (P3, a tourist). She continued, that "*even so the system is not perfect, it still can help me a lot on my next trip*". Other critical comments were targeted at the limited set of POI's in the app (e.g. "*Can it also show shops and café places?*", P3). In our prototype, we stored just a few POI's and the dataset can be easily extended.

5.3 Study 3

A third user study was conducted with the *MetroNavigator* application to directly compare the use of *SubwayPS* against an approach that merely interpolated from an official timeline.

5.3.1 Procedure

The study was conducted across two consecutive days between 8 and 6 pm in May 2014 on the Cologne, Germany subway. One experimenter engaged in 50 trips with similar trip length and travel time as measured in the second study. The experimenter

randomly boarded a train, travelled with it for exactly 7 stations and then left the train and boarded the next one arriving at that station. In order to compare against a timetable baseline, the experimenter recorded the exact time at which the train he was riding arrived at each station. No restrictions were placed on the position or orientation of the smartphone in this study. The Study 3 experimenter performed common activities normally executed on a smartphone as reported by Böhmer et al. [3], which involved periodically having the phone in his hands and storing the phone in his pocket during the trips.

5.3.2 Results

The average time on the train was about 9 minutes in total. Across all 50 trips, 282 out of 335 stops (300 stations and 35 "in-between" stops) where classified correctly (82.7% excluding the start stations; 85.0% including the start stations). 35 stations and 18 "in-between" stops were missed. Again, no false positive stops were recorded. These results are in line with our previous results from Study 1 and Study 2.

To evaluate *SubwayPS* against the timetable baseline, we compared the actual arrival time at each stop on all 50 trips to (1) the arrival time indicated by *SubwayPS* and (2) the arrival time indicated by the official subway schedule. Using a 30-second "tolerance" window, we found that *SubwayPS* tracked 39 entire trips with perfect accuracy (78% of trips), while using the official timetable only 21 entire trips (42%) were accurate.

The challenges presented to timetables from the "ripple effects" of a single delay have led some public transport systems to abandon timetables for an "interval"-based approach. In this approach, which was recently examined by Pritchard et al. [20], service is guaranteed every n minutes rather than at specific times.

To understand *SubwayPS*'s performance in this type of subway network, we also evaluated *SubwayPS* against a relative time baseline, which used just the reported travel time between stations. This baseline was able to track 25 out of 50 trips correctly (50%), as opposed to *SubwayPS*'s 39 out of 50.

The trip-level accuracy of *SubwayPS* is directly influenced by trip length and

accumulation errors (if one station is missed, the trip is not tracked correctly), which are known issues for inertial navigation techniques (i.e. "drift" errors). Fortunately, there has been extensive research on solutions for recalibration in inertial navigation. In the Future Work section, some of these solutions applicable to *SubwayPS* are mentioned.

Chapter 6

Conclusion

In this thesis, we presented *SubwayPS*, a smartphone positioning system that puts users "back on the map" when they are travelling in underground public transportation networks. *SubwayPS* does not require any instrumentation of the environment, meaning it can be implemented without any expense of the often-cash-strapped operators of public transportation systems. *SubwayPS* merely requires an accelerometer and a gyroscope - both of which are standard on many modern smartphones - and a start and end destination as input, which can be inferred from user behaviour or extracted from routing requests.

Our evaluation showed that *SubwayPS* works "out of the box" with four subway systems from two different continents. However, we also showed that *SubwayPS*'s accuracy can be increased if its four parameters are tuned specifically for an individual subway network. This tuning is simple, and merely requires a single user to collect accelerometer data as they travel on the subway and use our Windows Store application to identify the appropriate values for these parameters.

In the short term, *SubwayPS* is straightforward enough to be implemented directly into a mobile app as we have done with *MetroNavigator*, our proof-of-concept *SubwayPS* application. We have taken care to ensure that any developer who wishes to implement *SubwayPS* can do so by following the instructions (and using the para-

meters) laid out in the *SubwayPS* chapter. However, if *SubwayPS* (or a technology like it) were built into a mobile OS, it would have the greatest impact, allowing all location-aware mobile apps - including mobile map apps - to function while their users travel in subway networks.

That said, before OS integration can occur, several limitations of *SubwayPS* must be addressed. In its current implementation, accuracy at a trip level is highly dependent on trip length due to accumulation errors. Fortunately, there are various solutions to this issue. For instance, in many subway networks, subway stations have localized WiFi coverage. This means that traditional WiFi positioning techniques can be used to correct any errors before they accumulate.

In our three studies we have shown that *SubwayPS* is about 30% more accurate than previous existing methods or timetable approaches. Also, user feedback proved very positive, showing that there is an audience for an application such as *MetroNavigator*. This also confirms that following the official Android UI guildelines and implementing optional Google services has no negative effects on the user experience, while providing developers with great marketing options for a world-wide public release of such an application.

Chapter 7

Future work

Future work can proceed along four directions.

7.1 Addressing accumulation errors

There are multiple options to address the accumulation (drift) errors that are possible with the current implementation of *SubwayPS*. It is possible to implement WiFi-based positioning correction directly into the *SubwayPS* technique. This should drastically increase per-trip accuracy rates. Implementing *SubwayPS* on an OS-level like part of the Android localization services could automate this process, since a huge number of WiFi hotspots are known to Google's localization services. Traditional WiFi positioning techniques can also be integrated manually since in many subway networks, the subway stations have localized WiFi coverage.

There are a lot of solutions already implemented in the different papers in the Related Work section. As mentioned earlier, the different characteristics of certain subway stations could also provide solutions, as some subway lines have parts running above ground. Since underground positioning is possible with *SubwayPS*, GPS could be automatically enabled on above-ground parts of the track.

Other examples could include a smartphone's integrated microphone, as most subway systems have an automated "announcer" calling out at which station the train arrives. Since these messages are static and do not change over the years, an

audio recognition technique by acoustic fingerprint might also be usable.

7.2 Smartwatch extension

In the future it would be great to develop a smartwatch-only version of *SubwayPS* as we believe such a version would serve as a superior navigational aid on crowded subway networks. Doing so will involve learning to filter out arm movements and facing other smartwatch-specific challenges.

The current version of *MetroNavigator* for the smartwatch has the *SubwayPS* service still running on the connected smartphone and has a limited number of features. Running a full *MetroNavigator* + *SubwayPS* build on the smartwatch would increase the feature set of the smartwatch application.

7.3 External accelerometers

To help protect *SubwayPS* against user movements, a solution could be to use external sensors. For example, an embedded sensor such as the WAX9[1] could be used and placed in locations that don't require much interaction, such as on the chest of the user. Extracting the accelerometers allows the user to keep using his device for "active" applications such as games without influencing the measurements of the accelerometer.

7.4 *SubwayPS* data

Finally, we are considering the means by which crowdsourcing may be used to collect training data. This would provide us with multiple opportunities, such as using different relative times based on time-of-day, and would improve the accuracy of the "in-between" stop detection since the available timetable data would slowly change from a minute-scale precision to a second-scale precision.

[1] https://code.google.com/p/openmovement/wiki/WAX9

Apart from the time scale, data could also be split up from "subway system"-level to "subway track"-level, with different parameters for each track instead of for each subway system. This would increase stop detection accuracy on each line, but a decent amount of tracks from the same line would be necessary, something we did not have during the writing of this thesis. Last but not least, crowdsourcing could easily allow *SubwayPS* to support more subway networks, while we were now limited to the area's of residence of our participants of Study 1.

Appendix A

Code snippets

```
1 Line:Waterloo & City:#95CDBA
2 Waterloo
3 Bank
```

Listing A.1: Example of how an underground transportation line with corresponding stations is stored in an information file. One information file contains this kind of information for every line in a city. This specific fragment details the Waterloo & City line which has only two stations and is part of the information file named "london.txt".

```
1 Weiden West, 1
2 Weiden Schulstr., 2
3 Weiden Zentrum (Stadtbahn), 1
4 Weiden Bahnstr., 2
5 Junkersdorf Mohnweg, 2
6 Junkersdorf, 1
```

Listing A.2: Example of how timetable data is stored in an information file. This specific example is a fragment from Cologne Line 1. The numbers after each station declare the number of minutes between that station and the next one.".

```
protected float[] lowPass( float[] input, float[] output ) {
  if ( output == null ) return input;

  for ( int i=0; i<input.length; i++ ) {
    output[i] = output[i] + ALPHA * (input[i] - output[i]);
  }

  return output;
}
```

Listing A.3: Low-pass filter used to smooth data of the accelerometer.

```
double a = Math.sqrt(
                Math.pow(x, 2) +
                Math.pow(y, 2) +
                Math.pow(z, 2)
            );
```

Listing A.4: Implementation of Wang's method for acceleration synthesization with x y and z being accelerometer data already smoothed by the low-pass filter.

```
if (/* not enough samples yet */) {
  sum += a;
  smoothed = (mSamplecount == periods - 1) ? sum / (double) periods : 0;
  smoothedData.add(smoothed);
} else {
  // remove oldest data, add newest
  sum = sum - data.get((int)mSamplecount - periods) + a;
  smoothed = sum / (double) periods;
  smoothedData.add(smoothed);
}
```

Listing A.5: Moving average calculation of the synthesizated accelerations.

Bibliography

[1] C. Ascher, C. Kessler, M. Wankerl, and G. Trommer. Using orthoslam and aiding techniques for precise pedestrian indoor navigation. In *Proc. of ION GNSS '09*, pages 743–749, 2001.

[2] J. Biagioni, T. Gerlich, T. Merrifield, and J. Eriksson. Easytracker: automatic transit tracking, mapping, and arrival time prediction using smartphones. In *Proc. of Sensys '11*, pages 68–81. ACM, 2011.

[3] M. Böhmer, B. Hecht, J. Schöning, A. Krüger, and G. Bauer. Falling asleep with angry birds, facebook and kindle: a large scale study on mobile application usage. In *Proc. of MobileHCI '11*, pages 47–56. ACM, 2011.

[4] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon. Gamification. using game-design elements in non-gaming contexts. In *Proc. of CHI EA '11*, pages 2425–2428. ACM, 2011.

[5] L. Fang, P. Antsaklis, L. Montestruque, M. McMickell, M. Lemmon, Y. Sun, H. Fang, I. Koutroulis, M. Haenggi, M. Xie, et al. Design of a wireless assisted pedestrian dead reckoning system-the navmote experience. *IEEE Transactions on Instrumentation and Measurement*, 54(6):2342–2358, 2005.

[6] C. Fischer, K. Muthukrishnan, M. Hazas, and H. Gellersen. Ultrasound-aided pedestrian dead reckoning for indoor navigation. In *Proc. of MELT '08*, pages 31–36. ACM, 2008.

[7] D. Grejner-Brzezinska, C. Toth, and S. Moafipoor. Performance assessment of a multi-sensor personal navigator supported by an adaptive knowledge based system. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37:857–866, 2008.

[8] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. *Wireless Networks*, 8(2/3):187–197, 2002.

[9] S. Hemminki, P. Nurmi, and S. Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *Proc. of SenSys '13*, page 13. ACM, 2013.

[10] Google Inc. Control location settings for Google Now - Search Help, *Android Help*. http://support.google.com/websearch/answer/2824786?hl=en, [Online; accessed 29-May-2014], 2014.

[11] N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. Munich. The vslam algorithm for robust localization and mapping. In *Proc. of ICRA '05*, pages 24–29. IEEE, 2005.

[12] M. Keally, G. Zhou, G. Xing, J. Wu, and A. Pyles. Pbn: towards practical activity recognition using smartphone-based body sensor networks. In *Proc. of Sensys '11*, pages 246–259. ACM, 2011.

[13] M. Kourogi and T. Kurata. Personal positioning based on walking locomotion analysis with self-contained sensors and a wearable camera. In *Proc. of ISMAR '03*, page 103. IEEE Computer Society, 2003.

[14] M. Kourogi, N. Sakata, T. Okuma, and T. Kurata. Indoor/outdoor pedestrian navigation with an embedded gps/rfid/self-contained sensor system. In *Advances in Artificial Reality and Tele-Existence*, pages 1310–1321. Springer, 2006.

[15] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In *Proc. of Ubicomp '12*, pages 421–430. ACM, 2012.

[16] B. Lim and A. Dey. Toolkit to support intelligibility in context-aware applications. In *Proc. of Ubicomp '10*, pages 13–22. ACM, 2010.

[17] M. Löchtefeld, S. Gehring, J. Schöning, and A. Krüger. Pinwi: pedestrian indoor navigation without infrastructure. In *Proc. of NordiCHI '10*, pages 731–734. ACM, 2010.

[18] M. Muehlbauer, G. Bahle, and P. Lukowicz. What can an arm holster worn smart phone do for activity recognition? In *Proc. of ISWC '11*, pages 79–82. IEEE, 2011.

[19] S. Pace, G. Frost, I. Lachow, D. Frelinger, D. Fossum, D. Wassem, and M. Pinto. *The Global Positioning System, chapter GPS history, chronology and budgets*. 1995.

[20] G. Pritchard, J. Vines, P. Briggs, L. Thomas, and P. Olivier. Digitally driven: how location based services impact the work practices of london bus drivers. In *Proc. of CHI '14*, pages 3617–3626. ACM, 2014.

[21] F. Raab, E. Blood, T. Steiner, and H. Jones. Magnetic position and orientation tracking system. *IEEE Transactions on Aerospace and Electronic Systems*, (5):709–718, 1979.

[22] P. Robertson, M. Angermann, and B. Krach. Simultaneous localization and mapping for pedestrians using only foot-mounted inertial sensors. In *Proc. of Ubicomp '09*, pages 93–96. ACM, 2009.

[23] A. Ruiz, F. Granja, J. Prieto Honorato, and J. Rosas. Accurate pedestrian indoor navigation by tightly coupling foot-mounted imu and rfid measurements. *IEEE Transactions on Instrumentation and Measurement*, 61(1):178–189, 2012.

[24] J. Schöning, A. Krüger, K. Cheverst, M. Rohs, M. Löchtefeld, and F. Taher. Photomap: using spontaneously taken images of public maps for pedestrian navigation tasks on mobile devices. In *Proc. of MobileHCI '09*, page 14. ACM, 2009.

[25] A. Serra, D. Carboni, and V. Marotto. Indoor pedestrian navigation system using a modern smartphone. In *Proc. of Ubicomp '10*, pages 397–398. ACM, 2010.

[26] M. Susi, V. Renaudin, and G. Lachapelle. Motion mode recognition and step detection algorithms for mobile phone users. *Sensors*, 13(2):1539–1562, 2013.

[27] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson. Cooperative transit tracking using smart-phones. In *Proc. of Sensys '10*, pages 85–98. ACM, 2010.

[28] S. Wang, C. Chen, and J. Ma. Accelerometer based transportation mode recognition on mobile phones. In *Proc. of APWCS '10*, pages 44–46. IEEE, 2010.

[29] R. Want, A. Hopper, V. Falcão, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 1992.

[30] Wikipedia. Metro systems by annual passenger rides, *Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Metro_systems_by_annual_passenger_rides, [Online; accessed 29-May-2014], 2014.

[31] P. Zhou, Y. Zheng, and M. Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proc. of MobiSys '12*, pages 379–392. ACM, 2012.