universiteit hasselt

HASSELT UNIVERSITY

MASTER THESIS SUBMITTED TO ACHIEVE THE MASTER'S DEGREE IN CS, OPTION HUMAN-COMPUTER INTERACTION AT HASSELT UNIVERSITY

SubwayAPPS: Localisation on underground public transportation systems by using mobile air pressure sensors

---

Author:

Kris VAN ERUM

Promotor:

Prof. dr. Johannes SCHÖNING

2015-2016

Abstract

The smartphone, with its capabilities, is the de facto standard localization tool. It relies on technologies like cellullar based localization, WiFi-positioning and GPS. These technologies provide decent positioning information in daily life. However, they still have certain limitations, namely in urban canyons, inside buildings and underground. Various researchers have recently addressed the problem of current positioning technologies to provide sufficient position information on underground public transportation networks. Either solutions make use of additional and often very expensive infrastructure that is installed into the stations and tunnels, or they rely on the build-in sensors of current smartphones to determine the location of their users underground. Last year, Stockx et al. [28] developed *SubwayPS*, an underground locating system that makes use of the accelerometer in smartphones, has been developed.

In this thesis we present a novel smartphone-based approach named SubwayAPPS that uses the build-in barometer to precisely determine the position of a user in an underground public transportation system by detecting relative air pressure changes. We first compare the depth characteristic of five major underground networks across the globe and show that our novel approach is feasible across those cities. Second, we show with two user tests in Brussels as well as in London, that our approach can outperform other techniques, e.g. techniques that rely on the build-in accelerometers or gyroscopes, under realistic conditions, as we are able to detect about 88% of the stations.

Dutch summary

In grote steden is er een constante stroom van personen die niet vertrouwd zijn met de stad. Londen, bijvoorbeeld, ontving in 2014 28.8 miljoen toeristen. Dit is meer dan drie keer het bevolkingsaantal van Londen. Daarnaast waren er in Londen, in 2013, 50 180 migranten. Gelijkaardig aan Londen zijn er steeds meer en meer megasteden die met eenzelfde probleem kampen.

In de meeste gevallen vertrouwen deze personen op hun smartphone om te kunnen navigeren binnen deze, voor hun, onbekende stad. De meeste moderne smartphones beschikken over een GPS ontvanger. Dankzij deze ontvanger kunnen gebruikers hun locatie opvragen overal ter wereld en onder iedere weersomstandigheid. Deze technologie kan verder uitgebreid worden door, naast de gegevens verkregen van de satellieten, gebruik te maken van externe gegevensbronnen. Dit noemt men A-GPS.

Daarnaast bevatten deze smartphones ondersteuning voor de WiFi technologie. In stedelijke omgevingen zijn er zeer veel WiFi toegangspunten beschikbaar. Er zijn databases beschikbaar die een locatie hebben opgeslagen voor deze WiFi toegangspunten. Door een dergelijke database te combineren met de beschikbare toegangspunten in de omgeving, kan een smartphone zijn locatie bepalen dankzij de WiFi technologie.

Tot slot kan een smartphone gebruik maken van het standaard mobiele netwerk om zijn locatie te bepalen. Dit netwerk heeft nagenoeg wereldwijde dekking en is opgedeeld in verschillende cellen. Een mobiele telefoon verbindt met een specifieke cel om toegang te krijgen tot het netwerk. Ook hier bestaat er een database met cellen en de locatie van hun basisstation. Op deze manier kan de locatie van een telefoon bepaald worden aan de hand van het mobiele netwerk.

Deze drie technologieën volstaan echter niet voor ondergrondse locatiebepaling. De signalen uitgezonden door GPS satellieten kunnen niet doorheen de grond verplaatsen. Hetzelfde geldt voor de WiFi signalen. Er zijn onvoldoende WiFi toegangspunten beschikbaar in de metro. Ook de signalen van het mobiele netwerk zijn niet beschikbaar ondergronds.

Bijgevolg kunnen personen niet vertrouwen op hun smartphone om hun locatie te bepalen terwijl ze gebruik maken van de metro. Dit is problematisch in grote steden. In Londen bijvoorbeeld, verliep 34% van al het openbaar vervoer in 2013 ondergronds. Onze verwachting is dat dit percentage in de toekomst verder gaat stijgen, onder meer door de toenemende verkeersdrukte en om ruimte te besparen.

Om dit probleem op te lossen is er reeds veel onderzoek uitgevoerd. Zo werd geprobeerd de accelerometer, magnetometer en gyroscoop sensors te gebruiken om locatiebepaling in de metro uit te voeren. In deze thesis gebruiken we hiervoor een luchtdruksensor in een smartphone. SubwayAPPS (Subway Air Pressure Positioning System) is een nieuwe methode die enkel gebruik maakt van een luchtdruksensor om de aankomst van een metrotrein in een station te bepalen.

Het algoritme maakt enkel gebruik van luchtdruk metingen. Druk wordt gedefiniëerd als een hoeveelheid kracht per oppervlak ($p = \frac{F}{A}$), met als eenheid pascal (Pa). De luchtdruk kan gemeten worden met behulp van een barometer en is afhankelijk van onder andere het weer. Een (lokale) verandering van luchtdruk verspreid zich altijd gelijkmatig over de hele omgeving.

Voor de werking van het algoritme wordt gebruik gemaakt van twee eigenschappen van luchtdruk: de veranderende luchtdruk bij verschillende hoogtes en het *piston effect*.
Omdat lucht niet gewichtloos is, drukken de bovenste luchtlagen op de onderste luchtlagen. De gemeten luchtdruk is bijgevolg afhankelijk van de hoogte waarop gemeten wordt. In bovenste luchtlagen zal deze lager zijn dan in de onderste lagen. Dit verschil in luchtdruk tussen 2 hoogtes kan omgezet worden naar een verschil in meter door de *hypsometrische formule*: $h = ((\frac{P_0}{P})^{-0.19} - 1) \cdot -44330)$.

Wanneer een voertuig zich voortbeweegt, duwt het de lucht die zich voor het voertuig bevindt opzij. Deze weggeduwde lucht wordt gelijkmatig verdeeld rond het voertuig. Wanneer het voertuig zich echter in een smalle tunnel voortbeweegt, kan de lucht niet gelijkmatig

verdeeld worden. De lucht zal vooral voor het voertuig uit geduwd worden. Dit creëert een hogere druk voor het voertuig en een lagere druk in en achter het voertuig. Dit noemt men het *piston effect*. De impact van dit effect is afhankelijk van de snelheid van het voertuig, de breedte van de tunnel en de mogelijkheid voor de gevangen lucht om te ontsnappen (bijvoorbeeld via ventilatieschachten).

De SubwayAPPS methode steunt op de hoogteverschillen tussen aangrenzende metrostations. Bijgevolg is het belangrijk dat er voldoende hoogteverschillen zijn tussen de opeenvolgende metrostations. Hiervoor werd een studie uitgevoerd die de dieptestructuur van metrolijnen analyseert in Londen, Moskou, Tokyo, Wenen en Brussel.
Uit deze studie blijkt dat het gemiddelde hoogteverschil tussen twee aangrenzende stations meer dan 2 meter is. Dit terwijl de fabrikanten van de luchtdruksensors een precisie beloven van 1 meter. De kans dat het hoogteverschil tussen twee stations groter is dan 1 meter, bedraagt 82.11%. Indien we kijken naar het hoogteverschil bij een trip van twee stations, heeft 88% van deze trips een hoogteverschil groter dan 1 meter. Voor een trip van vier stations verhoogt deze kans verder naar 92%.

Zoals reeds vermeld, maakt het SubwayAPPS algoritme gebruik van de hoogteverschillen tussen aangrenzende metrostations en het piston effect.
Het piston effect zorgt ervoor dat de gemeten luchtdruk terwijl het voertuig beweegt onstabiel is. De veranderende snelheid van de trein, breedte van de tunnel en de aanwezigheid van ventilatieschachten beïnvloeden de werking van het piston effect constant. Wanneer de trein zich in stilstand in het station bevindt, is de gemeten luchtdruk stabiel.

De stabiliteit van de luchtdruk wordt bepaald door de variantie van de luchtdruk te berekenen binnen een bepaald tijdsinterval. Indien deze variatie groter is dan een drempelwaarde, beslist het algoritme dat de luchtdruk onstabiel is. Indien de variantie beneden deze drempelwaarde is, wordt de luchtdruk als stabiel ervaren.

Wanneer de luchtdruk van een onstabiele naar een stabiele toestand gaat, wordt gecontroleerd indien het gemeten hoogteverschil ten opzichte van het vorige station overeenkomt met het werkelijke hoogteverschil tussen het vorige station en het huidige station. Indien deze controle slaagt, beslist het algoritme dat de trein is aangekomen in het station. Wanneer de luchtdruk hierna onstabiel wordt, weet het algoritme dat het voertuig onderweg is naar het volgende station.

Het SubwayAPPS algoritme werd geïmplementeerd in de *MetroNavigator+* applicatie op het Android platform. De originele *MetroNavigator* applicatie werd ontwikkeld door Thomas Stockx. Deze applicatie laat gebruikers toe de huidige status van hun metro reis te volgen. De gebruiker kan het aantal resterende stations bekijken, volgende station bekijken, alsook de reistijd tot het volgende station en de reistijd tot zijn bestemming.

De SubwayAPPS methode werd getest in de metronetwerken van Brussel en Londen. Voor de test in Brussel, werden de 4 lijnen van de metro volledig doorlopen. In totaal zijn er 20 testen gebeurd in Brussel. 88% van de stations werd correct gedetecteerd door het algoritme. In Londen werden 20 ritten getest door het algoritme. Deze ritten hadden een gemiddelde duur van 9 minuten en werden willekeurig gekozen binnen zones 1 en 2 van het Londense metronetwerk. Het SubwayAPPS algoritme kon 62.50% van de bezochte stations correct detecteren. Dit is een beduidend slechter resultaat in vergelijking met de test in Brussel. Dit heeft twee mogelijke oorzaken. De diepte informatie van de stations in Londen is afkomstig uit een document vrijgegeven door TfL (de uitbater van het Londense metronetwerk) en dateert van 2011. De diepte informatie voor de stations in Brussel is afkomstig van zelf uitgevoerde metingen met een luchtdruksensor in een smartphone. Bijgevolg is de diepte informatie over de stations in Brussel accurater en stemt deze meer overeen met de hoogteverschillen die gemeten worden tijdens de uitvoering van het algoritme.
Een tweede mogelijke oorzaak is de grootte en drukte van de stations. De stations in Londen zijn over het algemeen groter en drukker dan deze in Brussel. Vaak verbinden ze meerdere lijnen met elkaar. Bijgevolg is de kans groter dat er, terwijl de gebruiker stil staat in een station, er op een ander platform een trein vertrekt of aankomt. Dit zorgt ervoor dat de gemeten luchtdruk niet stabiel genoeg kan worden voor het SubwayAPPS algoritme.

Een dergelijk probleem kan ook voor komen in een groot station in Brussel. Daarnaast kunnen abrupte weersveranderingen resulteren in een luchtdrukverandering die overeenkomt met een hoogteverschil van 2 meter. Om dit te beperken, meten we enkel de hoogteverschillen tussen twee opeenvolgende stations. Verder kan dit opgelost worden door de actuele weersinformatie te raadplegen.
Tijdens spitsuren kan de metrotrein overbevolkt zijn. Een gebruiker kan dan niet tijdens de volledige rit zitten. Wanneer de gebruiker wisselt tussen zitten en recht staan, verandert de hoogte van de smartphone. Om dit probleem te verhelpen kan een externe luchtdruksensor gemonteerd worden die zich altijd op dezelfde hoogte bevindt, zoals op de schoen van de gebruiker.

Tot slot kan de dienstverlening van het metronetwerk wijzigen. Een voorbeeld hiervan is de sluiting van een station door werken. Indien de trein dit station passeert, kan het algoritme geen stop detecteren. Dit is nefast voor de verdere werking van het algoritme, aangezien het verwachtte hoogteverschil niet wordt aangepast voor het volgende station. Dit kan verholpen worden door service informatie te integreren in het algoritme. Dankzij deze informatie kan het gesloten station overgeslagen worden.

In deze thesis werd het SubwayAPPS algoritme geïntroduceerd. Dit algoritme laat gebruikers toe hun locatie te bepalen tijdens een metrorit door enkel gebruik te maken van een luchtdruksensor in een smartphone. Er is geen externe infrastructuur vereist. Uit onze testen blijkt dat de accuraatheid van het SubwayAPPS algoritme 10% hoger ligt dan reeds bestaande methodes. Daarnaast werd er een analyse uitgevoerd van de dieptestructuur van metronetwerken en de accuraatheid van luchtdruksensoren in smartphones.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction and Motivation

Many cities face a constant stream of people who are unfamiliar with the city. This could be tourists or people who migrate to a city. For example, 28.8 million tourists, both domestic and overseas, visited London in 2014 [20]. This is more than three times the people that live in London. Furthermore, in 2013, London had a net migration of 50 180 people [30]. Similar to London, more and more megacities will face similar streams of people not familiar with the city environment.

In general, these people do not know their way around the city. To navigate, they rely on a smartphone. Modern smartphones have a GPS receiver. Thanks to improvements in the technology of these receivers and the opening of the GPS signal to the public in 2000, the GPS technology allows smartphone users to accurately find their location anywhere in the world and under every weather condition. To further improve the accuracy and time-to-first-fix, a smartphone can make use of other data sources. This can provide satellite position information, an initial time and position estimate, satellite selection etc. This combination of a GPS receiver and the external data source is called Assisted-GPS (A-GPS) [35].

In addition to GPS receivers, modern smartphones are equipped with WiFi receivers. In urban areas, many WiFi access points repeatedly transmit messages that their access point exists. The WiFi receivers detect the available access points in the area, without needing to connect to these access points. Databases with a mapping between the access points and their physical location are available. The access points are identified by their MAC address. The smartphone derives its location by matching the available access points with the access points in the database. This technique is called fingerprinting. This method works best in urban areas because many access points are needed. Because WiFi signals can travel through

walls and buildings, it is well-suited for use indoors and outdoors.

Another technique used by smartphones to localize themselves is the standard cellular network. These networks have almost worldwide coverage. They are divided in cells. Each cell has a base station that takes care of the communication between the network and the mobile phone. The base stations have a known location and a cell ID for identification. From this, with similar techniques to WiFi positioning, the location of the mobile phone can be derived. The accuracy here depends greatly on the size of the cells and the number of cells available. A study by [17] showed that the average accuracy of this method is 506m, with a worst accuracy of 3062m.

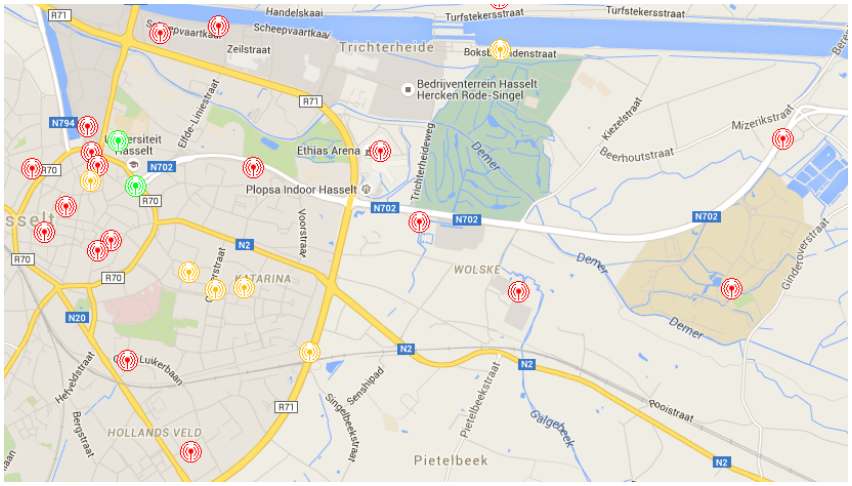

Figure 1.1: Locations of cellular base stations in Hasselt. Red base stations are operational, yellow are approved building applications and green are submitted building applications [3]

Because modern smartphones have these techniques built-in, together with popular map applications (e.g. Google Maps, Apple Maps and Bing Maps) they have become the de facto standard for daily life navigation. Tomtom, one of the biggest sellors of navigation devices, had a big decrease in their standalone navigation devices sales, mainly to navigation applications on smartphones [12].
The ability of smartphones to find its location also has created new possibilities for social media applications. For example, geocaching is an increasingly popular activity where people search for hidden objects (caches) by solving puzzles and navigating with their smartphone. Users can also share their location when posting messages on social media like Twitter and Facebook.

1.1 PROBLEM

In 2015, 34% of all public transport journeys were made underground (see figure 1.2). Still 59% of public transport journeys are made by bus. With the increasing population growth, which causes an increasing traffic congestion, we believe that the share of underground journeys will increment to save time on the road. To save space, it is also better to build tunnels for transportation than to construct roads overground.

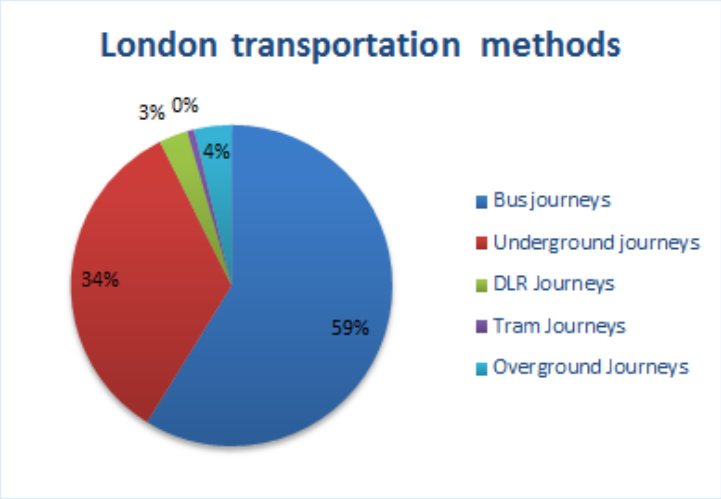

Figure 1.2: Distribution of public transport journeys in London in 2015 [9]

The GPS technique was designed for usage outdoors and uses radio signals to transmit messages from the satellites to the GPS receiver. This has some negative consequences. The GPS signals have difficulties traveling through solid objects. Not all signals can reach the receiver when indoors or underground. As the GPS technique needs at least 4 satellite signals, this makes it unusable in these situations. This problem can not be solved with the enhancements of A-GPS.

The same problem exists for the WiFi-positioning technique. The signals broadcasted by the access points can not travel through concrete. Because of this, the access points above the ground are invisible in the underground. A solution for this is to install many access points in the underground. This is very expensive. For example, the London underground network has 270 stations and spans 408 kilometer. To completely cover this network, many access points are needed.

Cellular positioning suffers from the same problem as well. There is bad cellular service while travelling underground. This decreases the accuracy of the cellular positioning technique. Even if there is a good service underground, the accuracy of this method is not good enough. In London, the average distance between adjacent stations is 1.31 kilometers. From [17], we know that the accuracy of cellular positioning can vary a lot and can get as worse as 3 kilometers.

From this, we can conclude that smartphones, with its three positioning methods, is unable to locate itself accurately while using underground transportation networks. This is problematic, because people depend on their smartphone for navigational aid in unfamiliar environments. This causes people to exit the train at the wrong stop. Therefore researchers and practitioners came up with several techniques to also provide positioning in underground public transportation systems [24,28,29,32] as we outline in detail in the related work section.

Knowing your location while traveling underground also has some other benefits. For example, tourists could be notified about landmarks that are overground while the train is riding. Advertisers could provide targeted advertisements, based on the location of the user.

1.2 GOALS

This thesis has three contributions. First we study the structure of underground transportation networks. We do this by analyzing the height differences between stations for five cities. Next, we develop a novel algorithm for detecting underground stations using only a smartphone's air pressure sensor. Our technique is novel because we are the first to use an air pressure sensor together with the known height differences between stations and the effects of the piston effect. We proof that we can achieve reasonable accuracy with our method. Further, we implement this algorithm for the Android platform, so navigation applications can listen for events generated by the algorithm. As an example, we integrated the algorithm with the existing *MetroNavigator+* application, for real-life usage. Finally, we analyze the accuracy of smartphone air pressure sensors for height difference detection. It is essential for our technique that the sensors have good accuracy.

1.3 STRUCTURE OF THE THESIS

This thesis is structured as follows:

- In section 2 we take a look at the existing localization methods. Next, we study and compare specific localization methods for underground transportation systems. After that, we take a look at what air pressure is and how it can be measured with a sensor in a smartphone. To conclude this section, we investigate the behavior of air pressure in closed environments.
- In section 3 we study the depth structure of underground transportation networks. The height differences between stations in five cities are analyzed.
- In section 4 we define the SubwayAPPS algorithm. We do this by plotting and analyzing recorded air pressure data during subway trips.
- In section 5 the implementation of the SubwayAPPS algorithm is explained in detail. The algorithm is implemented on the Android platform. An example of the usage of the algorithm in an application, MetroNavigator+, is given.
- Before we tested the algorithm, we studied the accuracy of the air pressure sensors in smartphones. More specifically, we investigated how sensitive the sensors are for measuring relative height differences. The results of this study are explained in section 6. After this, the SubwayAPPS algorithm was tested on the subway systems of Brussels and London.
- Finally, we draw conclusions in section 7. We discuss how we can further improve localization in underground transportation networks and how air pressure sensors can be used in other applications.

Chapter 2

Related Work

In this section, we will study several localization techniques, compare them and analyze how they can be used on underground transportation networks. Next, we will look at interesting properties of air pressure in general and how air pressure can be measured with a mobile device. To conclude this section, we study the behaviour of air pressure in underground transportation networks.

2.1 Localization techniques and methods

In this section, different general localisation techniques and methods are presented. We structure this by using the taxonomy of Hightower and Borriello [14]. We also present common examples for some techniques.

Location sensing techniques can be divided into the following categories:

- **Triangulation** uses the geometric properties of triangles to determine location. It can be further divided into *lateration* and *triangulation*.
  Lateration uses the distance to multiple reference points to determine the location. In 3 dimensions, at least 4 reference points are required. There are 3 methods to define the distance to the reference point: direct (using a physical action or movement), time-of-flight (using the time it takes for a signal to arrive) and attenuation (using the strength of a signal). A well known example of the lateration technique is GPS. This uses time-of-flight to determine the distance to the GPS satellites.

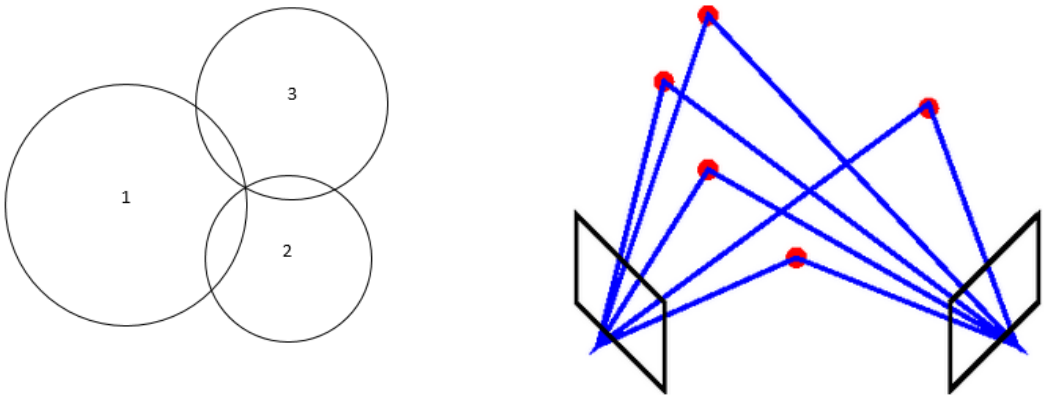

Figure 2.1: a) Example of trilateration in 2 dimensions. The location can be determined by calculating the intersection of the circles that are defined by the distance from reference points (1, 2 and 3), b) Scene analysis: Features (red dots) are observed from different vantage points. The position of the vantage points can be calculated [27]

Angulation uses angles instead of distances to determine the location. To determine a position in a 3D space, one length measurement, one azimuth measurement and two angle measurements are needed.

- **Scene analysis** uses specific distinguishable features of a scene, observed from a particular vantage point, to determine the current position. There are two variations of this technique: *static* and *differential*. In static scene analysis, a predefined dataset of features is used to determine the position. Differential scene analysis looks at the differences of the features in successive scenes. If the features are at a fixed position in the environment, a change of the position of a feature in the consecutive scenes corresponds to a movement of the observer.

- **Proximity** relies on the presence of nearby objects with a known location. There are three approaches for this technique: detecting physical contact, monitoring wireless cellular access points (both cellular and WiFi) and observing automatic ID systems.

- **Inertial navigation** does not rely on external references. Instead it uses sensors (e.g. accelerometer or gyroscope) and information about the initial location to predict the current location incrementally (e.g. [1, 8, 22, 23]).

Furthermore, we can classify localisation techniques using the following properties:

- **Physical position vs. Symbolic location**: a physical position denotes the real physical location (e.g. a coordinate) while a symbolic location gives an abstract idea of where

something is (e.g. at Hasselt University). Note that, if we have the information, we can convert between these representations. For example, if we know the physical positions of subway stations and we know the station we are currently in, we can infer our current physical position.

- **Absolute vs. Relative**: an absolute position system uses the same reference grid for all located objects while in a relative position system each object can have its own reference frame.
- **Localized location computation**: whether the object that is located computes the location itself or the computation is done by the system.
- **Accuracy**: the grain size of the position.
- **Precision**: how often we can get the accuracy.
- **Scale**: at what scale does the system work? This can be in a single room up to worldwide.
- **Recognition**: whether the system can recognize and identify the objects that need to be located.
- **Cost**: the cost (money, space and time) of the system.
- **Limitations**: the limitations of the system. Some systems will not work in specific environments (e.g. GPS does not work well underground).

2.1.1 Example: GPS

The most commonly used localization technique is GPS (Global Positioning System). GPS uses the lateration technique to determine location. Thirty two earth orbiting satellites continuously transmit messages that contain the time of the transmission, according to an atomic clock in the satellite, and the current location of the satellite. A GPS receiver calculates its position by solving the following equations: [4]

$$
\begin{aligned}
(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2 &= [c(t_1 - t_0)]^2 \\
(x_2 - x_0)^2 + (y_2 - y_0)^2 + (z_2 - z_0)^2 &= [c(t_2 - t_0)]^2 \\
(x_3 - x_0)^2 + (y_3 - y_0)^2 + (z_3 - z_0)^2 &= [c(t_3 - t_0)]^2 \\
(x_4 - x_0)^2 + (y_4 - y_0)^2 + (z_4 - z_0)^2 &= [c(t_4 - t_0)]^2
\end{aligned}
$$

where x, y and z are cartesian coordinates, t is the time, c is the speed of light, subscript 0 denotes the receiver and subscripts 1 to 4 denote the 4 satellites. Because there are 4 unknown values (x_0, y_0, z_0 and t_0) there need to be at least 4 equations and thus at least 4 satellites to calculate an accurate position. Besides GPS, there are other systems that use the same technique but are not developed by the United States, e.g. GLONASS (Russia), GALILEO (Europe) and BeiDou-2 (China).

2.1.2 Example: WiFi positioning

Another frequently used localization technique is WiFi positioning. If we take into account all nearby WiFi access points, with a known location, and determine our current location, we use the lateration technique. Another approach is to use the proximity technique and look for the presence of a known WiFi access point. To map a WiFi access point to a specific location, we need a database of access points and their location. The process of gathering this data is called *wardriving*. Examples of existing databases are Skyhook Wireless[1] and Wigle[2].

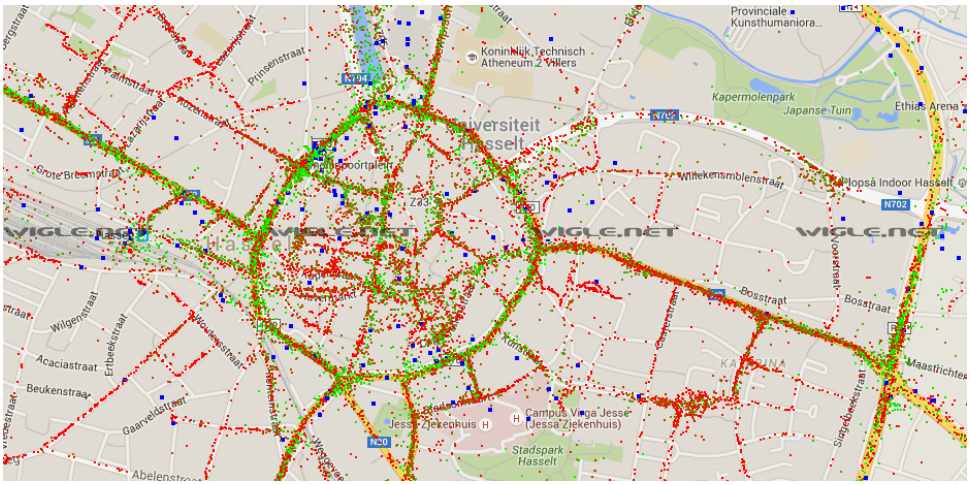

Figure 2.2: Visualization of the Wigle database of known WiFi access points in Hasselt

For example, there is free WiFi available in all stations and tunnels of the Moscow subway system [18]. This makes it possible for the company that operates the subway to track the passengers as they travel[3].
However, this requires the presence of WiFi access points in all the subway stations, which is not always the case. If these need to be installed, this can be a big cost.

[1] `http://www.skyhookwireless.com`
[2] `https://wigle.net/`
[3] `http://map.maximatelecom.ru`

2.1.3 Conclusion

A downside of the GPS technique is that the signal is weak and travels by line of sight. They can not pass through solid objects like buildings and mountains. This is problematic for usage in urban areas where the GPS receiver is surrounded by buildings (urban canyon effect) and locating underground objects, as is the case on subway systems.
WiFi positioning requires the presence of WiFi access points across the whole subway network. This is not the case in all networks and involves a big cost.
Traditional localization methods do not perform well in underground transportation networks. To get an accurate location fix, more specialized techniques are required.

2.2 UNDERGROUND LOCALIZATION TECHNIQUES AND METHODS

Most current smartphones are equipped with various sensors such as accelerometers, magnetometers and barometers to support user input and output. In addition to their main purpose the sensors can also be used to support positing, using inertial navigation techniques (e.g. [1, 8, 22, 23]), when GPS or WiFi positioning are not available.

2.2.1 SubwayPS

SubwayPS [28], presented at Sigspatial 2014, uses the accelerometer to do station detection. This technique relies on the fact that while a subway train is moving, the total acceleration (in all directions) is higher than when the train is stationary. Before the total acceleration is calculated, the gravity component is filtered out by making use of the gyroscope, such that the total acceleration is 0 when the device is at rest. The following formula is used to calculate the total acceleration: $a = \sqrt{a_x^2 + a_y^2 + a_z^2}$

The algorithm uses a hard threshold to determine if the subway train is riding or stationary. If a number of samples is above this threshold, the algorithm decides that the train is moving. Similarly, if a number of samples is below this threshold, the algorithm decides that the train is stationary. These parameters (threshold and number of samples) can be tweaked for specific subway networks. If the parameters were tweaked, around 85% of all stops were classified correctly. With the standard, world-wide parameters, 75% of the stops were classified correctly.

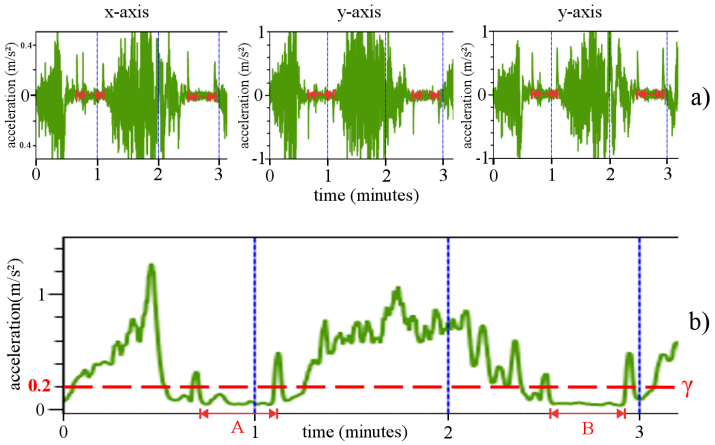

Figure 2.3: Example of the SubwayPS algorithm. The total acceleration during stationary periods (indicated by A and B) is lower than during periods when the train is riding [28]

2.2.2 Co-operative Transit Tracking

Thiagarajan et al. [29] use the accelerometer to make a prediction whether the train is riding or stationary. The accelerometer samples of stationary and riding periods were modeled as Laplace distributions, $f(x|\mu_{mov}, b_{mov})$ and $f(x|\mu_{sta}, b_{sta})$ with μ the median of the training set and $b = \frac{1}{N}\sum_{i=1}^{N}|x_i - \mu|$. Bayes' theorem is then used to calculate the probability that the train is riding:

$$p(mov|x) = \frac{f(x|\mu_{mov},b_{mov})\,p(mov)}{f(x|\mu_{mov},b_{mov})+f(x|\mu_{sta},b_{sta})}.$$

A 30 second time window is used to calculate the mean accelerometer value. During the interval when a subway train is riding and arrives at a station, a probability peak (train is moving) and valley (train is stationary) will occur, which is detected by a peak detector. The average difference between the time that the train arrives at a station and the prediction of the algorithm is 41 seconds.

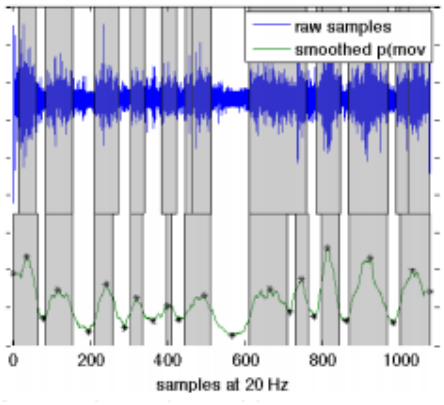

Figure 2.4: Example of the algorithm by Thiagarajan et al. [29] Each interval between two stations corresponds to a peak and trough

2.2.3 StationSense

Another technique that uses the accelerometer is the StationSense passenger tracking system [15]. On top of the accelerometer, it utilises the magnetometer to complement the station detection algorithm. Typically, motors and electrical inverters emit magnetic noise when the train accelerates and decelerates. During stationary periods, the values of the magnetometer have a lower variance. A stop probability is calculated by using Bayes' theorem, similar to [29]. However, the stop probability can sometimes increase due to inertia driving of the trains. To filter out these false positives, readings of the accelerometer are used. With this method, 72% of all detections were indeed a stop and 82% of all stops were classified correctly.

2.2.4 Air Pressure Template Matching

Similar to the work presented in this paper, Watanabe et al. [32] used an air pressure sensor to detect subway stations. Instead of measuring relative height difference, they assume that the height profile of every subway line was unique and these height differences could be measured by an air pressure sensor. They also take into account the structural factors (e.g. ventilation shafts) of each line. They assume that the change of air pressure is unique for each ride between adjacent subway stations. First, a template is generated for each ride between adjacent stations by recording the air pressure while riding the train. The algorithm then tries to match the incoming air pressure values to one of the templates. Because the train speed varies (e.g. because of the crowdedness on the train), the variation of air pressure is elastic in time. To overcome this problem, Dynamic Time Warping (DTW) is used. Furthermore, because the absolute air pressure values change over time (e.g. due to weather differences),

this technique only takes into account the relative change of air pressure. This method was tested in the Tokyo Metro (9 lines with 192 stations) and 85% of all stops were classified correctly.

2.2.5 Comparison

	GPS	**WiFi positioning**	**SubwayPS**	**Co-operative transit tracking**	**StationSense**	**Air pressure template matching**
Sensor	Physical	Symbolic	Accelerometer	Accelerometer	Accelerometer & magnetometer	Barometer
Physical/ symbolic	Physical	Symbolic	Symbolic	Symbolic	Symbolic	Symbolic
Absolute/ relative	Absolute	Absolute	Relative	Relative	Relative	Relative
LLC	Yes	Yes	Yes	Yes	Yes	Yes
Recognition	No	No	No	No	No	No
Accuracy & precision	1-5 meters (95-99%)	WiFi cell size	75-85%	Unknown	72%	85%
Scale	24 satellites worldwide	Worldwide	Subway networks worldwide	Subway networks worldwide	Subway networks worldwide	Analyzed subway networks only
Cost	Expensive infrastructure and receivers	WiFi access points	Accelerometer in smartphone	Accelerometer in smartphone	Accelerometer and magnetometer in smartphone	Barometer in smartphone
Limitations	Not indoors, in urban canyons and underground	Needs to be in range of WiFi access points, wardriving necessary	Tweaking needed for specific subway networks	False positives due to train joilts	Magnetic noise from other train influences results	Template needed

Table 2.1: Comparison of localization techniques using the taxonomy of Hightower and Borriello

In contrast to related work, we are the first to use an air pressure sensor in a mobile phone to provide positioning in underground public transport systems. That goes beyond basic station detection or simple approaches that detect if a train is moving or not. By making use of of the height differences between subway stations and the piston effect, we overcome the problems of template matching approaches using air pressure, that need trips of multiple stations to achieve decent accuracies, as we show in section 6.1.

2.3 SENSING AIR PRESSURE WITH A MOBILE DEVICE

Before we describe the technical implementation of the SubwayAPPS application, in this section we describe how air pressure is measured on mobile devices to make use of these sensors for underground localization. The relationship between air pressure and relative height is explained in detail.

Pressure is defined as the perpendicular force applied per unit area: $p = \frac{F}{A}$ [34]. The SI unit of pressure is the pascal (Pa). Other frequently used units of pressure are shown in table 2.2.

Unit	Symbol	Conversion
pascal	Pa	$1\,Pa = 1\,N/m^2$
bar	bar	$1\,bar = 10^5\,Pa$
millibar	$mbar$	$1\,mbar = 10^2\,Pa$
atmospheric pressure	atm	$1\,atm = 1.013 \cdot 10^5\,Pa$

Table 2.2: Frequently used units of pressure

The air pressure varies due to weather conditions and is related to the altitude. Because air itself is not weightless, the upper air layers exert pressure on the lower air layers, resulting in a higher pressure at sea level. At higher altitudes, the pressure is lower than at sea level.
The relation between pressure and altitude is as follows (*hypsometric formula*) [26]:

$$h = \frac{((\frac{P_0}{P})^{\frac{L \cdot R}{g}} - 1) \cdot (T + 273.15)}{L}$$

where P_0 and P are pressures at different altitudes (in hPa), h is the height different between these points (in meter), L is the lapse rate of the temperature (-0.0065 K/m), R is the universal gas constant (287.053 J/kg.K), g is earth's gravity constant (9.81 m/s^2) and T is the temperature in (°C).
If we assume a constant temperature of 15°C, this formula results in:

$$h = \frac{((\frac{P_0}{P})^{\frac{-0.0065 \cdot 287.053}{9.81}} - 1) \cdot (15 + 273.15)}{-0.0065} = ((\frac{P_0}{P})^{-0.19} - 1) \cdot -44330)$$

Another interesting phenomenon about pressure is Pascal's law: *Pressure applied to an enclosed fluid is transmitted undiminished to every portion of the fluid and the walls of the containing vessel* [34]. This means that when the air pressure changes (for example, due to changing weather conditions), this change will be transmitted to all depths. The differences in pressure at different heights will be the same as before the pressure change.

2.3.1 Commonly Used Sensors and Devices

Nowadays, most mid-priced smartphones are equipped with barometers. The primary reason to equip smartphones with a barometer is to improve the time for a GPS fix. The barometer can detect the altitude and thus provide a better z-coordinate for the GPS position calculation [24], as the accuracy of the z-coordinate is the lowest with the GPS technique, compared to x and y. Besides that, smartphones use the barometer sensor to detect on which floor the smartphone is in multistory buildings and to gather weather data [21].

Barometer sensors in smartphones are Microelectromechanical systems (MEMs). MEMs are very small devices, ranging from 0.02 to 1.0 mm, which are manufactured by using photolithography. Most MEMs barometers are of the piezoresistive type. The electrical resistivity of a piezoresistor changes when mechanical strain is applied, for example due to atmospheric pressure.
The barometer exists of a diaphragm over a small vacuum cavity. The air pressure presses the diaphragm into the cavity, which causes a change of the electrical resistivity of the piezoresistors on the diaphragm [24]. This change of resistance can be measured and converted to the atmospheric pressure.
The following factors can influence the MEMs barometer sensor:

- **Temperature:** The piezoresistors are sensitive to the temperature, a different temperature will change the electrical resistivity of the piezoresistor. To adjust for this, current pressure sensors also contain a temperature sensor. The driver takes into account the temperature and compensates the value of the air pressure.

- **Installation bias:** Because we only need the relative change in air pressure to calculate the relative height difference, the installation bias will not have an impact on this project.

- **Aging drift:** The aging drift of a pressure sensor is only significant over the course of months. As we measure the air pressure during short trips, this will not influence this project.

- **Weather:** As mentioned before, the air pressure depends on the current weather conditions. The weather drift can lead to a drift of a few meters in an hour, but intense storms can cause a drift of 3 to 4 meters in only 10 minutes.
  A possible solution is to only measure the relative height difference between adjacent stations as opposed to measuring the height difference compared to the first station of

the trip. This reduces the time window of the measurements and thus the influence of changing weather conditions.

- **Sunlight and wind:** This will not have an impact in underground transportation networks. Furthermore, the MEMs barometer is well protected from sunlight and wind by the case of the smartphone.

The most common barometer sensors used in smartphones are listed in table 2.3. We see that the accuracy is ±0.12 hPa, which corresponds to ±1m. When the pressure sensor has no built-in noise filter, we can smoothen the height values using the following filter [24]:

$$currentHeight = \alpha * sensorHeight + (1 - \alpha) * prevHeight$$

Smartphones that use the Bosch BMP280 barometer sensor do this filtering automatically in a special hardware component.

Barometer chip	**Bosch BMP180**	**Bosch BMP280**	**STM LPS331AP**
Used in	Samsung Galaxy Nexus, Samsung Galaxy S4, Google Nexus 4	Google Nexus 5, Apple iPhone 6	Samsung Galaxy S3
Temperature sensor	Yes	Yes	Yes
Temperature compensation	Quadratic (in driver)	Quadratic (in driver)	Linear (on chip)
Noise filter	No	Yes	No
Relative accuracy	±0.12 hPa	± 0.12 hPa	±0.1 hPa

Table 2.3: Comparison of common smartphone barometer sensors

2.4 AIR FLOW IN UNDERGROUND TRANSPORTATION NETWORKS

Because our suggested method depends on the air pressure, it is important to have an understanding of the air flow in underground transportation networks, as this influences the air pressure.

It is important to understand that when a vehicle is moving, it actually moves through air. Because of this, the air in front of the vehicle gets pushed away to the front and sides, while there is a vacuum behind the moving vehicle that sucks the air into it. When the vehicle

moves in open air, the air can move freely in any direction. But when the vehicle moves in a narrow tunnel (as is the case with underground transportation) the air can not move as freely and is “trapped” between the vehicle and the walls of the tunnel. This is known as the *piston effect* [19]. A sketch of this effect is shown in figure 2.5.

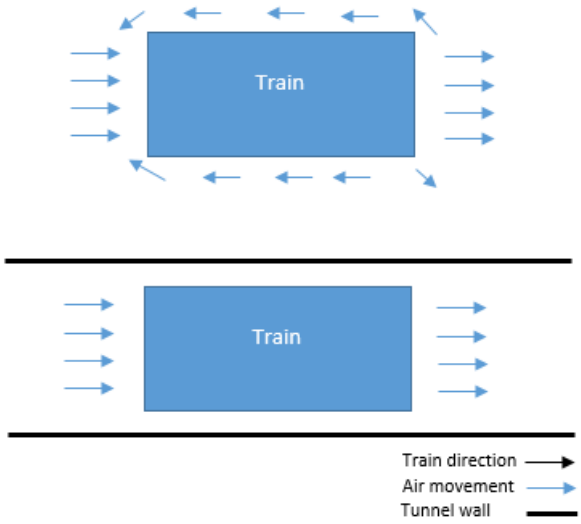

Figure 2.5: Sketch comparing the piston effect with a vehicle that moves in open air (top) and a vehicle that moves in a tunnel (below)

When the subway train enters a tunnel, the air pressure in and behind the train will drop while the air pressure in front of the train increases. When the subway train arrives at a station, the air around train can move freely again. The air pressure normalises: the air pressure in front of the train decreases and the air pressure in and behind the train increases.

The piston effect also depends on the environment. A change in width of the tunnel affects the air flow around the train and thus influences the measured air pressure. Additionally, when the train passes a ventilation shaft in the tunnel, the air in front of the train can escape and air can be sucked in from the ventilation shaft at the back of the train. This reduces the piston effect. Finally, when another train passes on another track, the piston effect of the other train will also affect the air pressure measurements.

We have to take into account this effect when we calculate the current elevation to not confuse it with a change in elevation. But the effect can be useful as well, to detect when a subway train arrives or departs at a station.

Chapter 3

Depth Structure of Underground Transportation Networks

Our method relies on detecting relative height difference between adjacent stations. This can be measured by the built-in smartphone barometer. In the first step we analyse the relative height differences of adjacent subway stations across four different subway networks (London, UK; Moscow, Russia; Tokyo, Japan; Vienna, Austria; Brussels, Belgium) to find out if our approach would work across major underground networks. All cities are among the largest subway networks in the world [33] and the depth information of the stations where available for us. In section 3.3 we also analyse the hypothesis of Watanabe et al. [32] if template matching of air pressure data, could be reliable used across different lines, cities or global networks.

To analyse the subway networks, we calculate the following measures:

- $\boldsymbol{HeightDiff_{station}}$: The height difference between two adjacent stations
- $\boldsymbol{HeightDiffTrip_n}$: The height difference for a trip of length n, where n ranges from 1 to the maximum number of stations in a line.
- $\boldsymbol{HeightDistDiff_{station}}$: The height difference between two adjacent stations, divided by the distance between the stations.
- $\boldsymbol{HeightDistDiffTrip_n}$: The height difference divided by the distance for a trip of length n, where n ranges from 1 to the maximum number of stations in a line.

Notice that $HeightDiff_{station}$ is equal to $HeightDiffTrip_n$ for $n = 1$. The same is true for $HeightDistDiff_{station}$ and $HeightDistDiffTrip_n$.

The distance between the stations was calculated by using the haversine formula, which gives the shortest distance between two points on a sphere. If we want to compare the depths of subway stations, we need to take into account the datum that the depths are given in. A datum is a coordinate system used to locate places on Earth. Both horizontal and vertical datums exist. For this study, we are only interested in vertical datums. The difference between datums is called the *datum shift*. This shift can reach up to hundreds of meters.

In this section, we will analyse the height differences between subway stations in different cities. Furthermore, we take a look at the slope between adjacent stations. At the end of this section, we will make a comparison of the different underground transportation networks.

3.1 RELATIVE HEIGHT OF SUBWAY NETWORKS

3.1.1 London

The height information and the geolocations of the stations were extracted from *Transport for London (TfL)*, the company that operates the London underground transportation network. With 270 subway stations, 10 lines and spanning 402 kilometers, London has the biggest subway network in Europe [10].

The average height difference between stations of the 10 lines of the London underground system is shown in figure 3.16. As can be seen, even for the line with the minimum average height differences between the stations (District line), the average height difference is well above 2m. The average height difference, over all lines, is 5.63m.

When we look at the height differences for trips of different lengths (figure 3.1), we see that in general the height difference increases when the trip length increases. This can make it easier to detect a trip between multiple stations, because the higher elevation difference can be better detected by the barometer.

For the average slope for a trip of length n (figure 3.3), we see that the slope decreases when the trip length increases. This can be explained by the fact that after an ascend, a descend may follow and vice versa. These cancel each other out, resulting in a smaller slope.

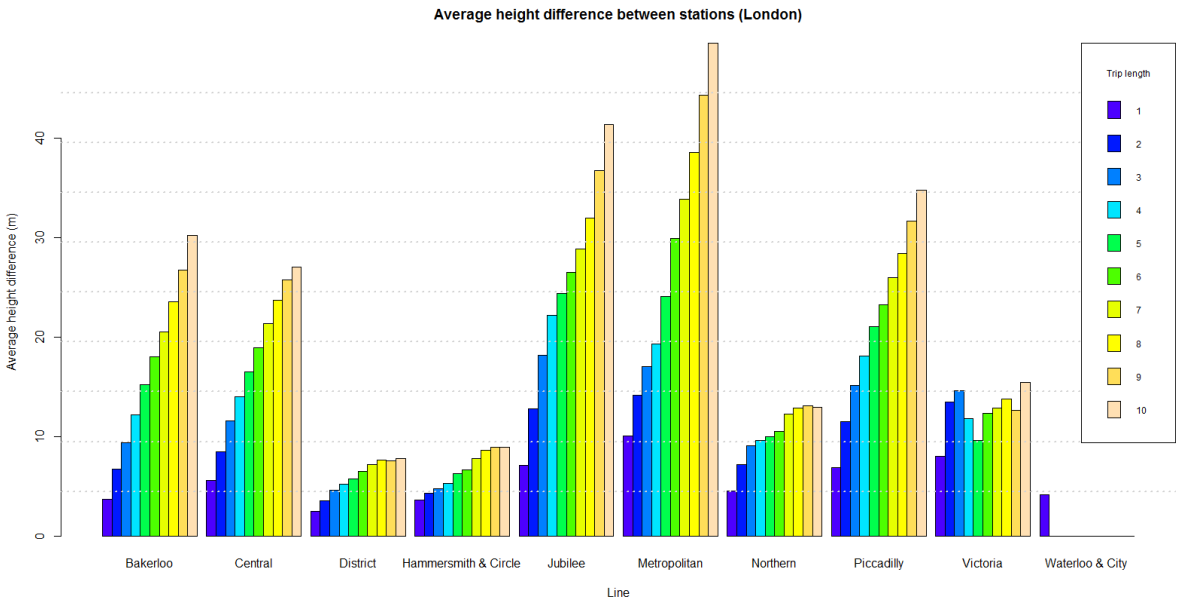

Figure 3.1: Average height difference for a trip of length n in London, per line

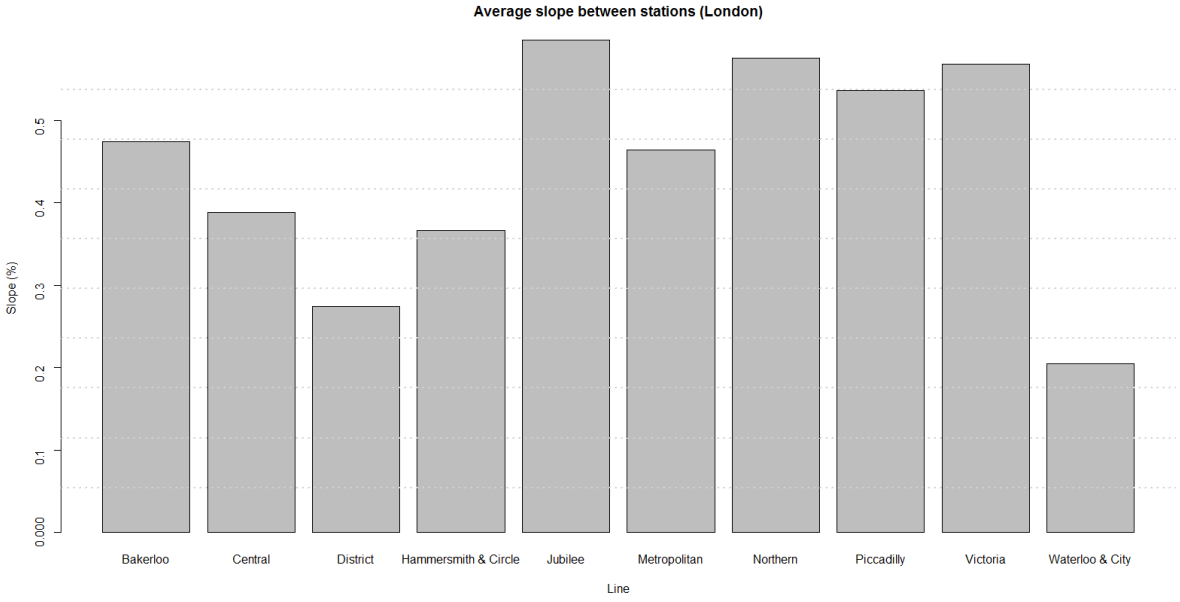

Figure 3.2: Average slope between stations in London, per line

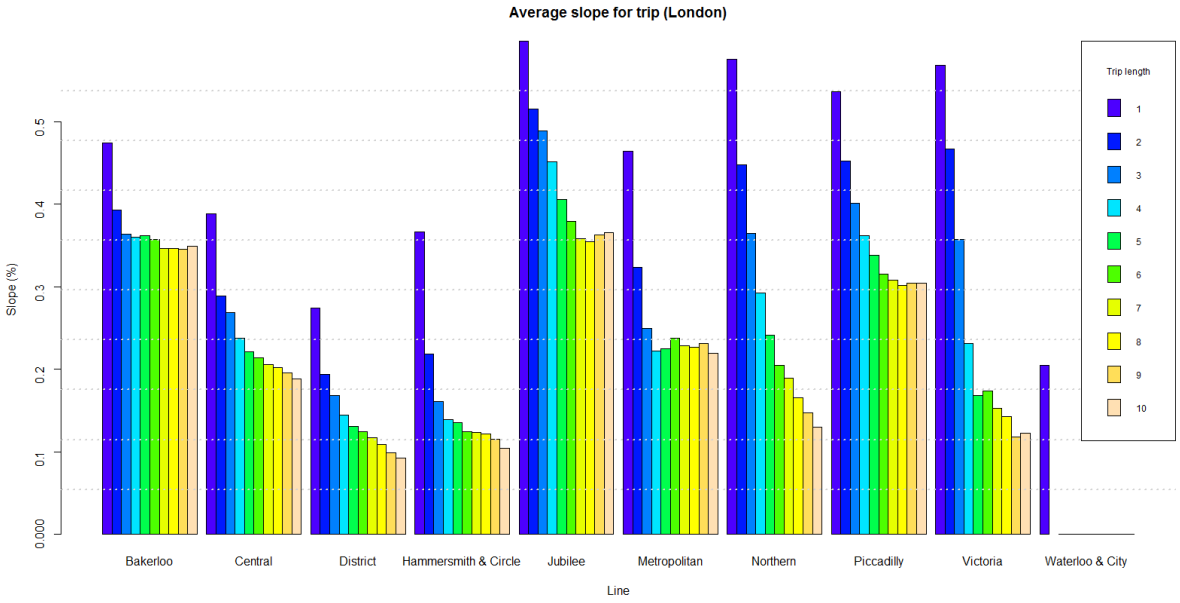

Figure 3.3: Average slope for a trip of length n in London, per line

3.1.2 Moscow

The depth information for the Moscow subway system is based on the 3D visualization of Evgeniy Varfolomeev [31]. For this visualization, the depth information of Alexey Goncharov [11] was used. This consisted of the depth of the stations beneath the ground level. Because the ground of Moscow is not levelled, we could not use this data directly to compare the depth of the stations. To put the depth data in the same datum, the depth data was combined with the elevation data of Google Earth. For example, when we know a station is 10m below ground level and the ground level at the station is 20m above sea level, we can deduce that the subway station is 10m above sea level.

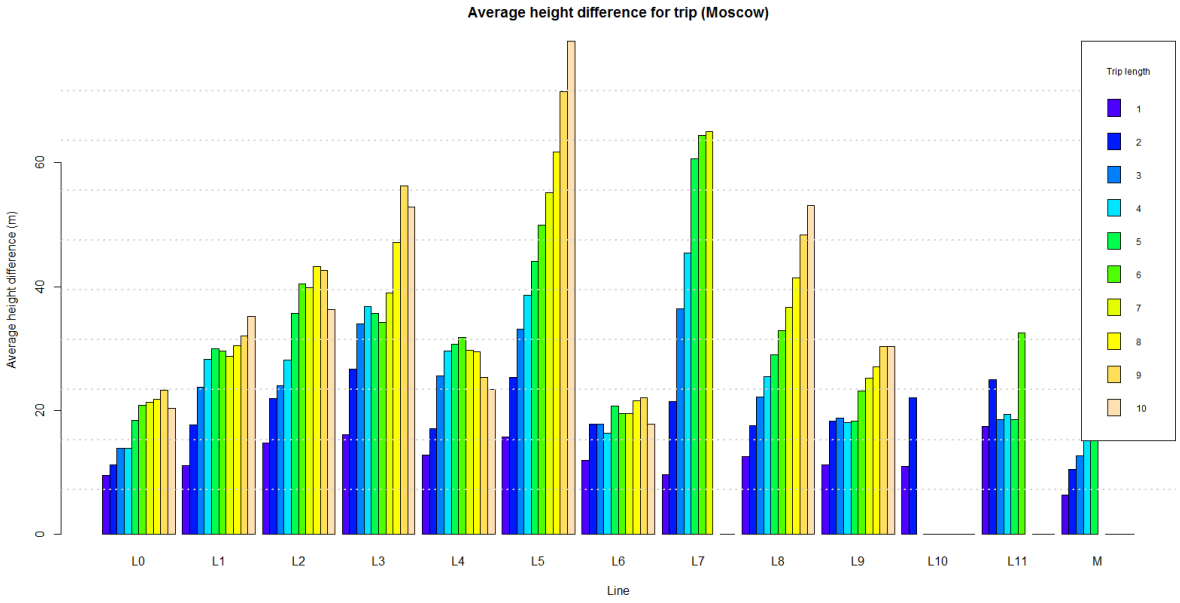

Figure 3.4: Average height difference for a trip of length n in Moscow, per line

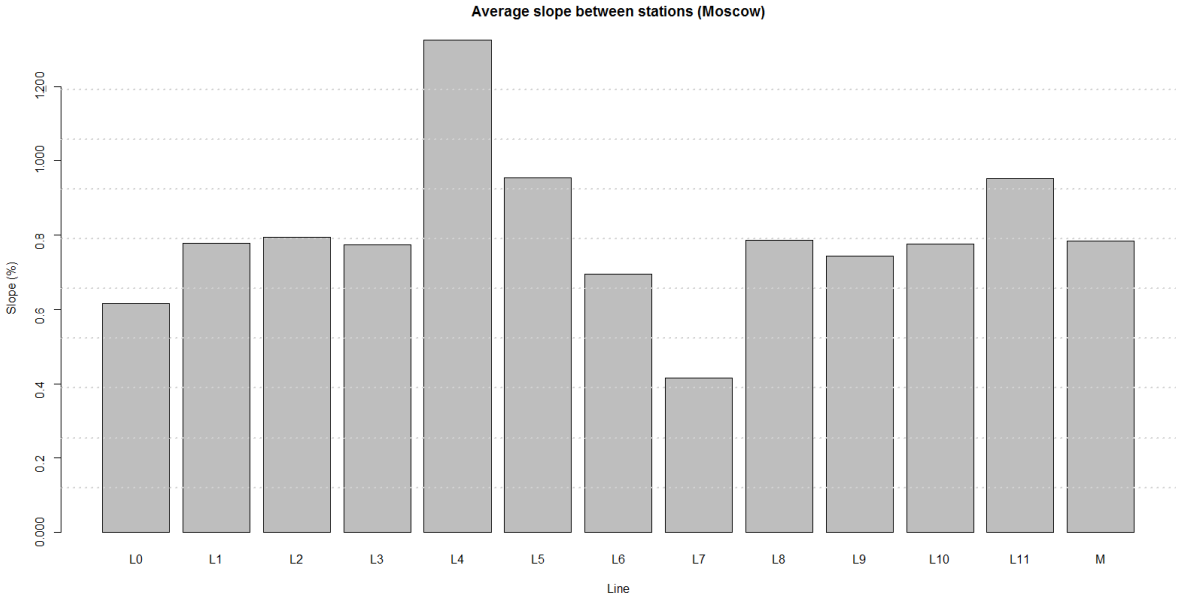

Figure 3.5: Average slope between stations in Moscow, per line

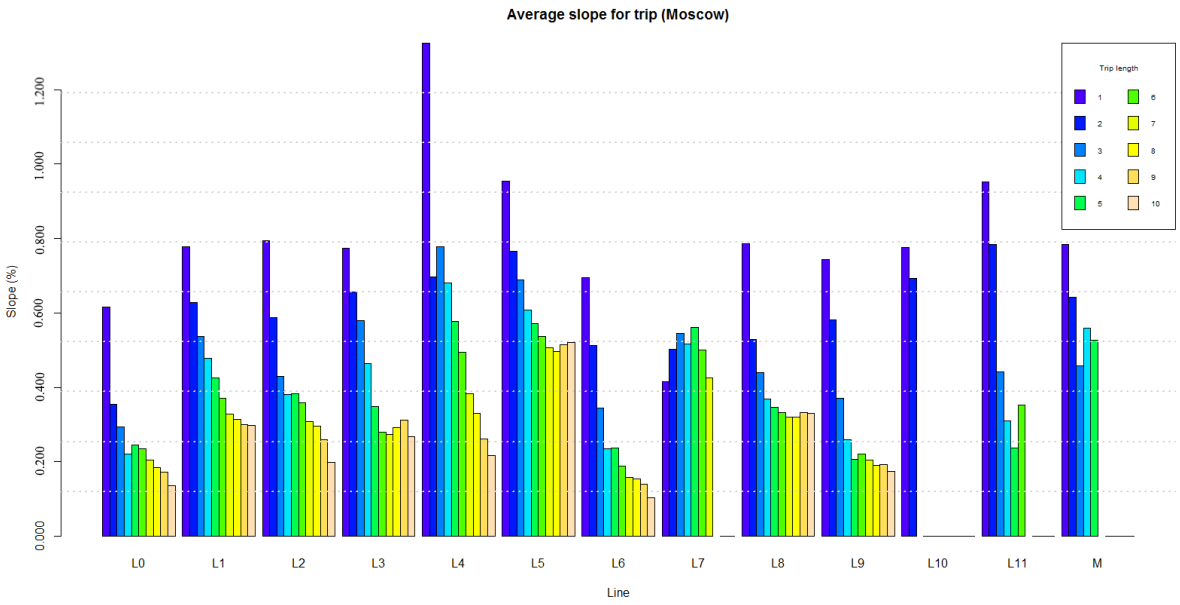

Figure 3.6: Average slope for a trip of length n in Moscow, per line

3.1.3 Tokyo

The depth information of the Tokyo subway system was available on the website of Tokyo Metro[1]. The depths of all stations were given in meters relative to sea level.

[1] `http://www.tokyometro.jp/en/`

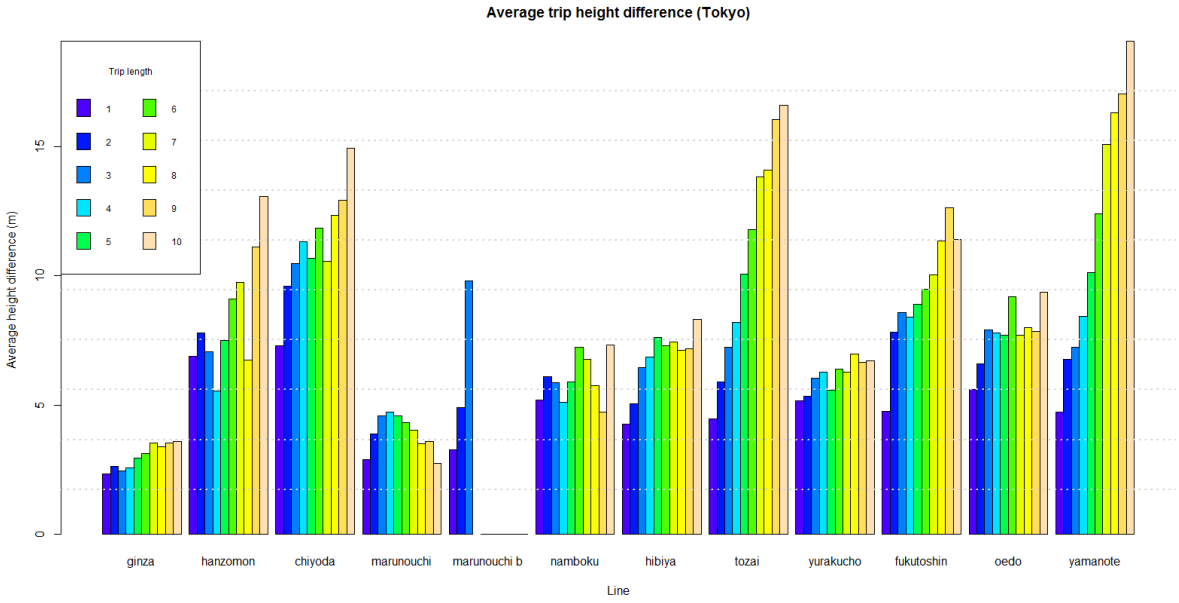

Figure 3.7: Average height difference for a trip of length n in Tokyo, per line

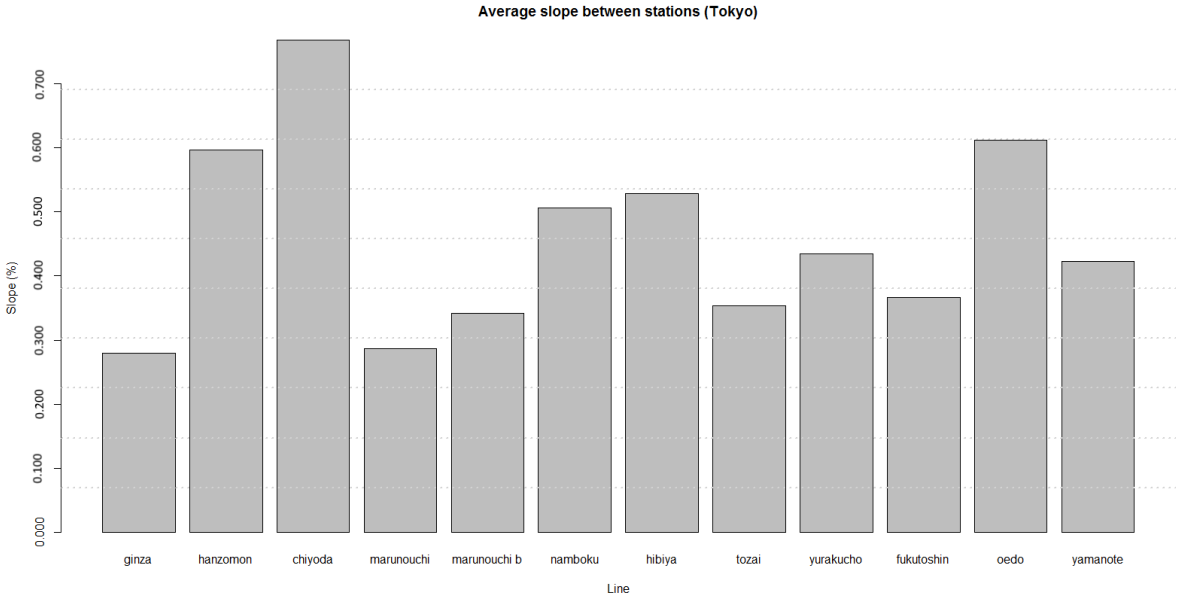

Figure 3.8: Average slope between stations in Tokyo, per line

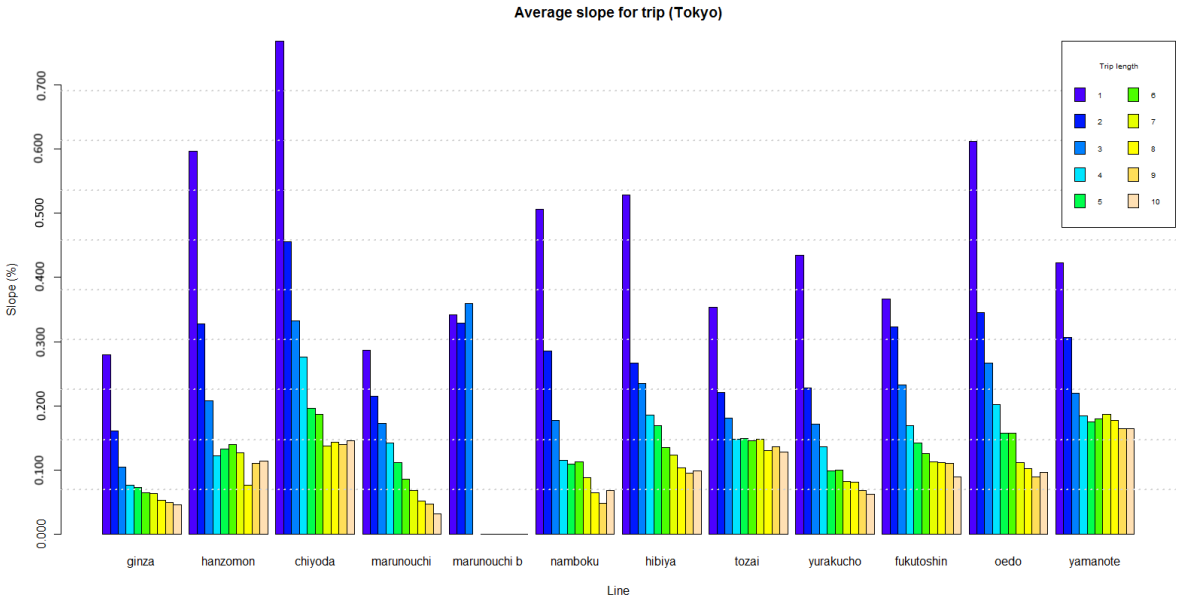

Figure 3.9: Average slope for a trip of length n in Tokyo, per line

3.1.4 Vienna

The data of the Vienna U-Bahn was obtained from Wiener Linien, the company that operates the subway network in Vienna. The locations of the stations were available via the Open Data Portal from the city of Vienna[2].

As we can see in figure 3.16, the height differences between stations are much bigger for line U6. This can be explained by the fact that, except for four short tunnels, the line runs above ground on elevated tracks. Because of this, we will exclude line U6 from the comparison in section 3.2.

[2] `https://open.wien.gv.at/site/open-data/`

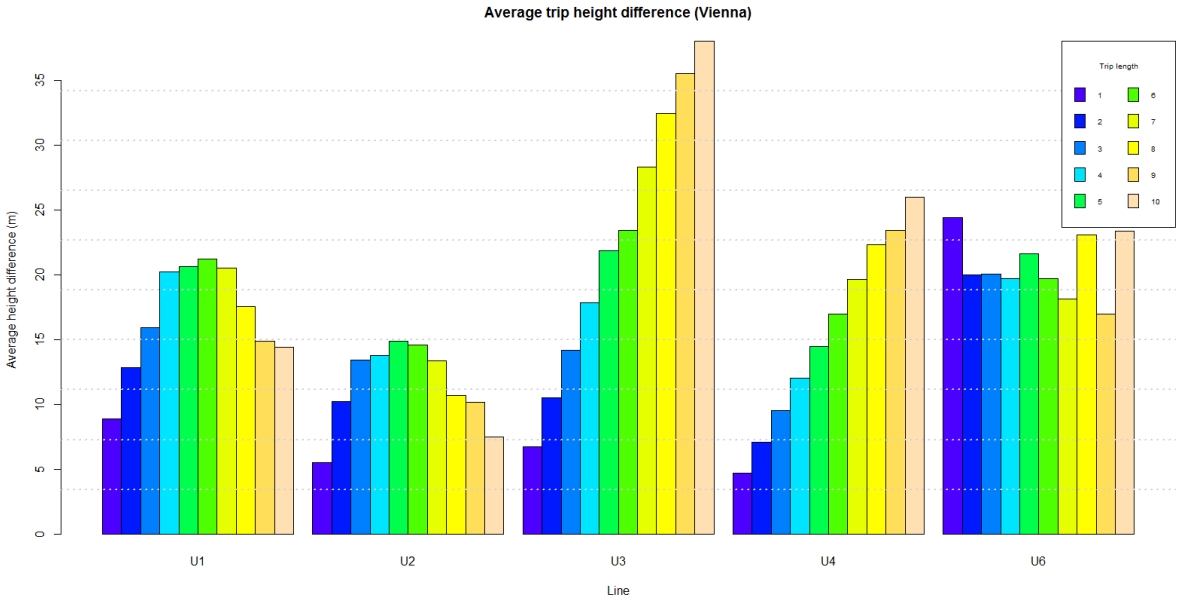

Figure 3.10: Average height difference for a trip of length n in Vienna, per line

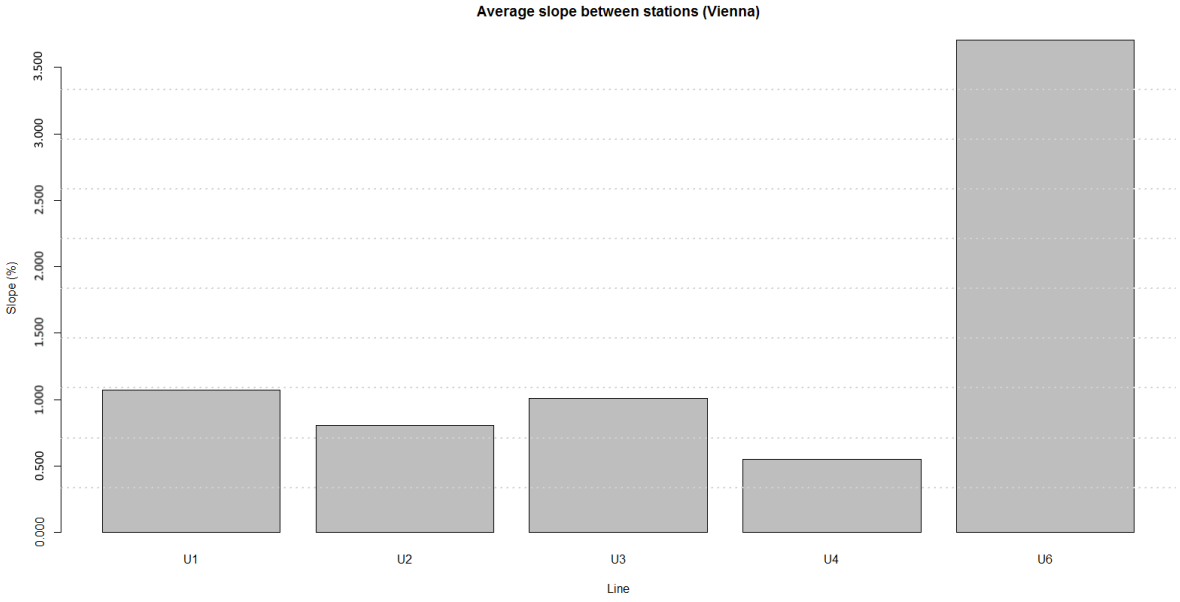

Figure 3.11: Average slope between stations in Vienna, per line

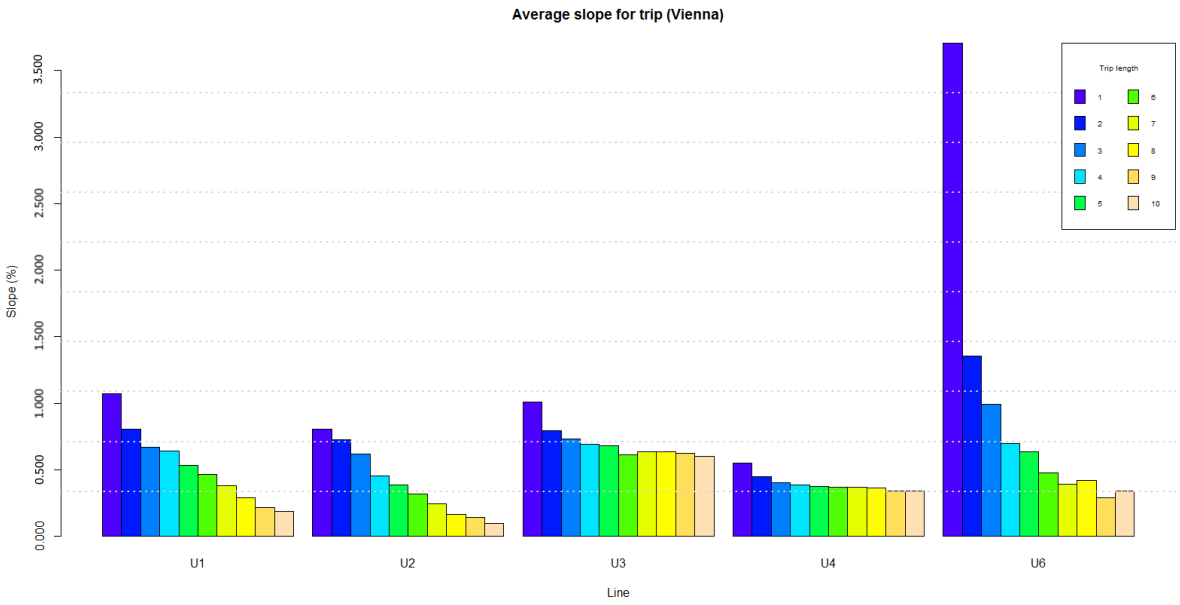

Figure 3.12: Average slope for a trip of length n in Vienna, per line

3.1.5 Brussels

Depth information of the Brussels subway network was unavailable. However for testing purposes, due to its proximity to Hasselt University, it was useful to know the depth of the subway stations in Brussels. We have recorded air pressure data of multiple subway trips in Brussels with the air pressure logger application (see appendix A). This data was recorded on two different days, using a Google nexus 4 (Bosch BMP180) and Apple iPhone 6 (Bosch BMP280). From this data, we calculated the depth information of the Brussels subway stations using the hypsometric formula. If height data between two stations from multiple recorded trips was available, the average value of these recordings was used.

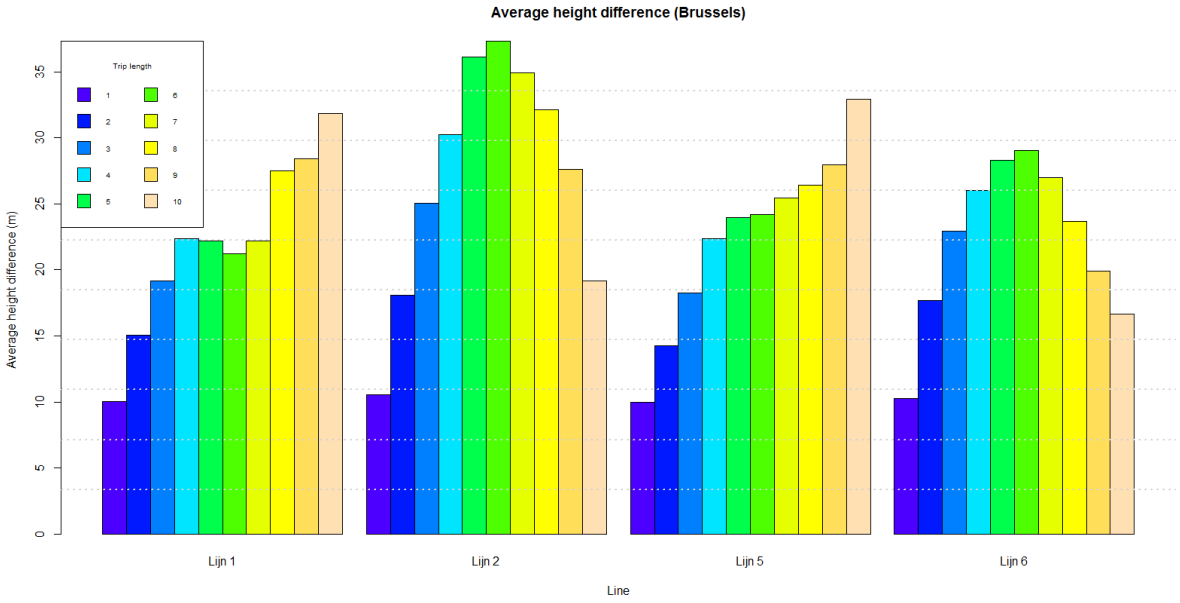

Figure 3.13: Average height difference for a trip of length n in Brussels, per line

We see from figures 3.16 and 3.13 that the height differences between stations for all lines are close to each other. This can be explained by the fact that many of the lines in Brussels share the same tracks. Line 1 and 5 share the same tracks between *Weststation* and *Merode* (12 stations), while line 2 is just a part of the longer line 6.

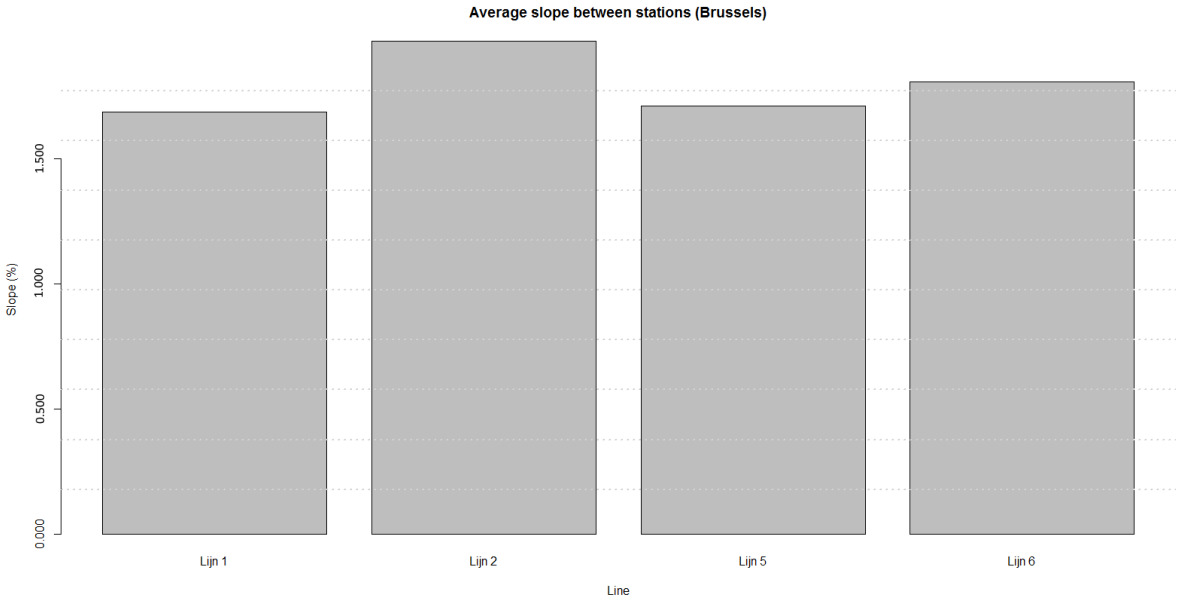

Figure 3.14: Average slope between stations in Brussels, per line

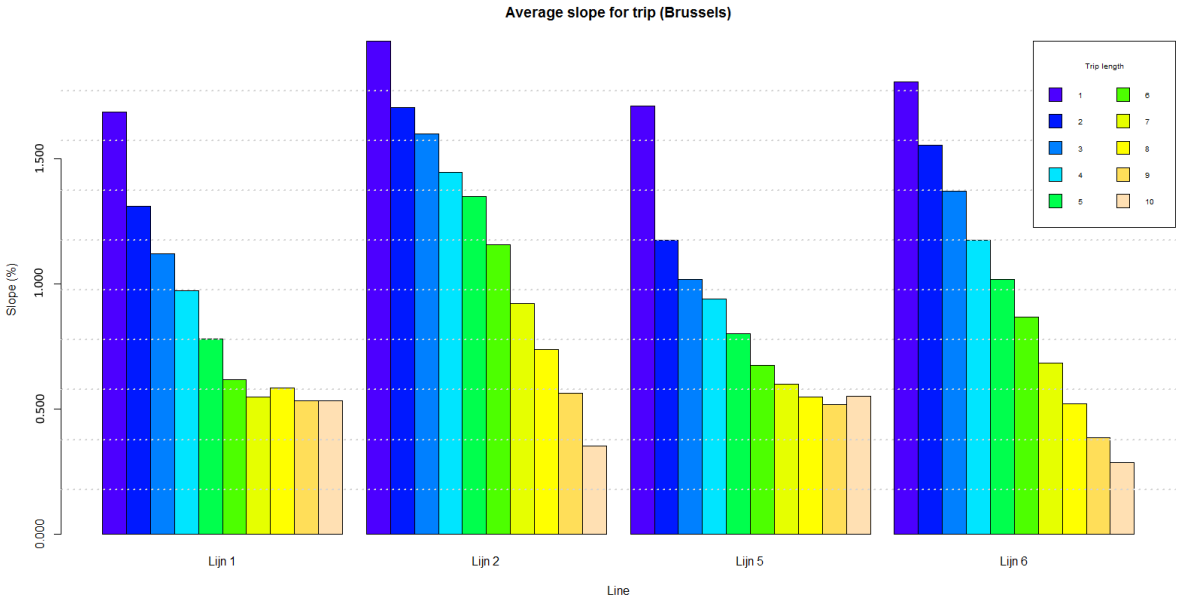

Figure 3.15: Average slope for a trip of length n in Brussels, per line

3.2 COMPARISON

Next, we compare the depth information statistics of the different cities.

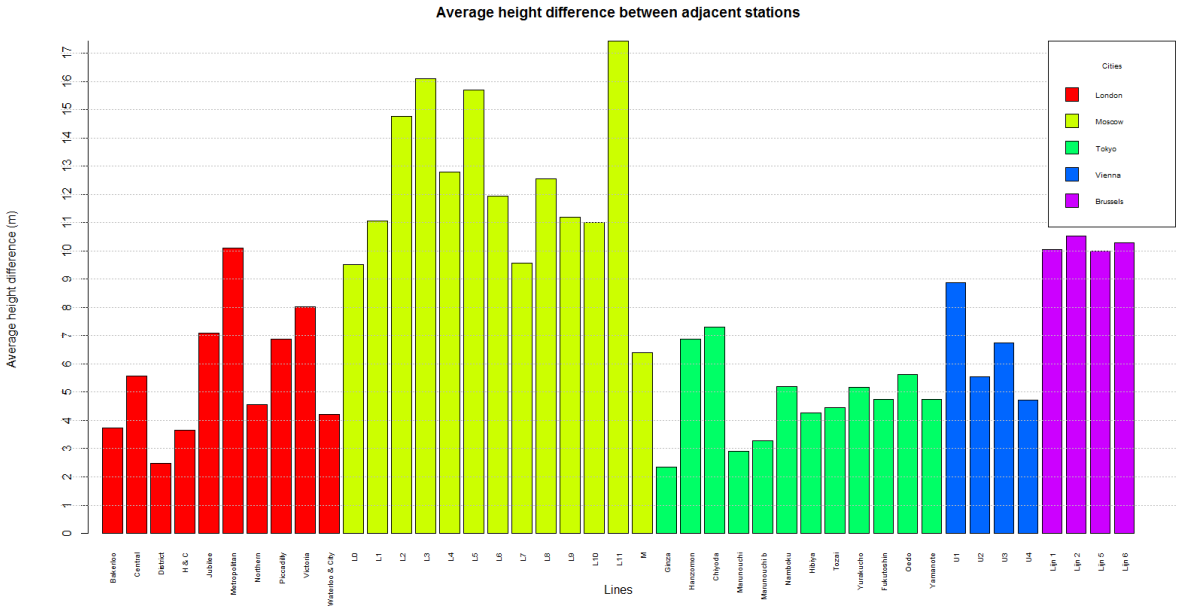

Figure 3.16: Average height difference between subway stations of all lines

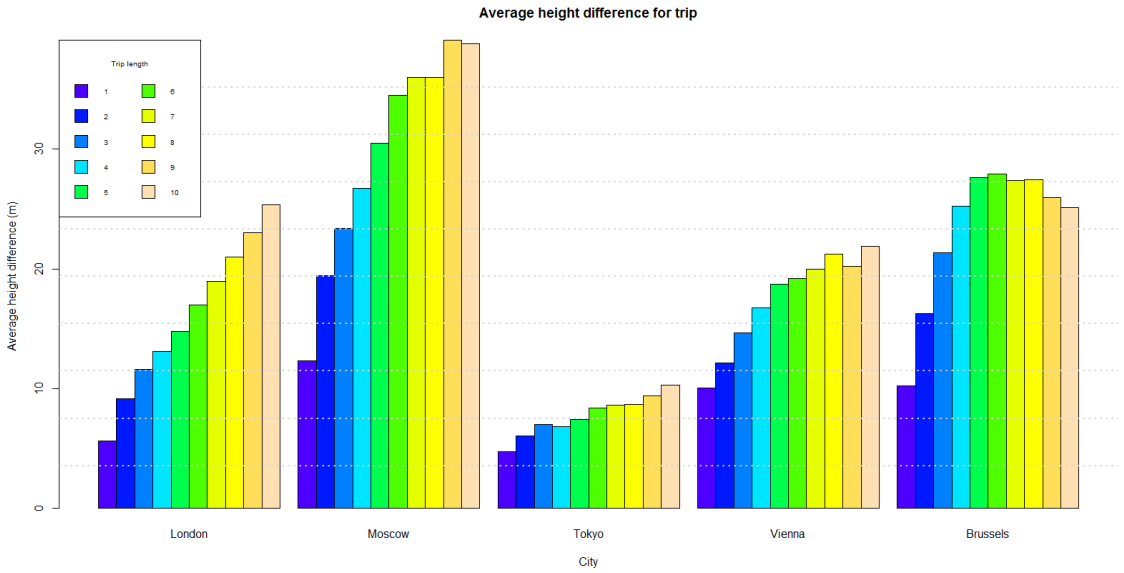

Figure 3.17: Average height difference for a trip of length n, per city

The Moscow subway system has the biggest height differences, so we can expect that our proposed method works best in this city. The Tokyo subway system has the smallest height differences. This can be explained by the fact that there is no river running through Tokyo and no tracks have to be build underneath a river. For all cities, the height difference increases when the trip length increases, making it easier to detect trips covering multiple stations.

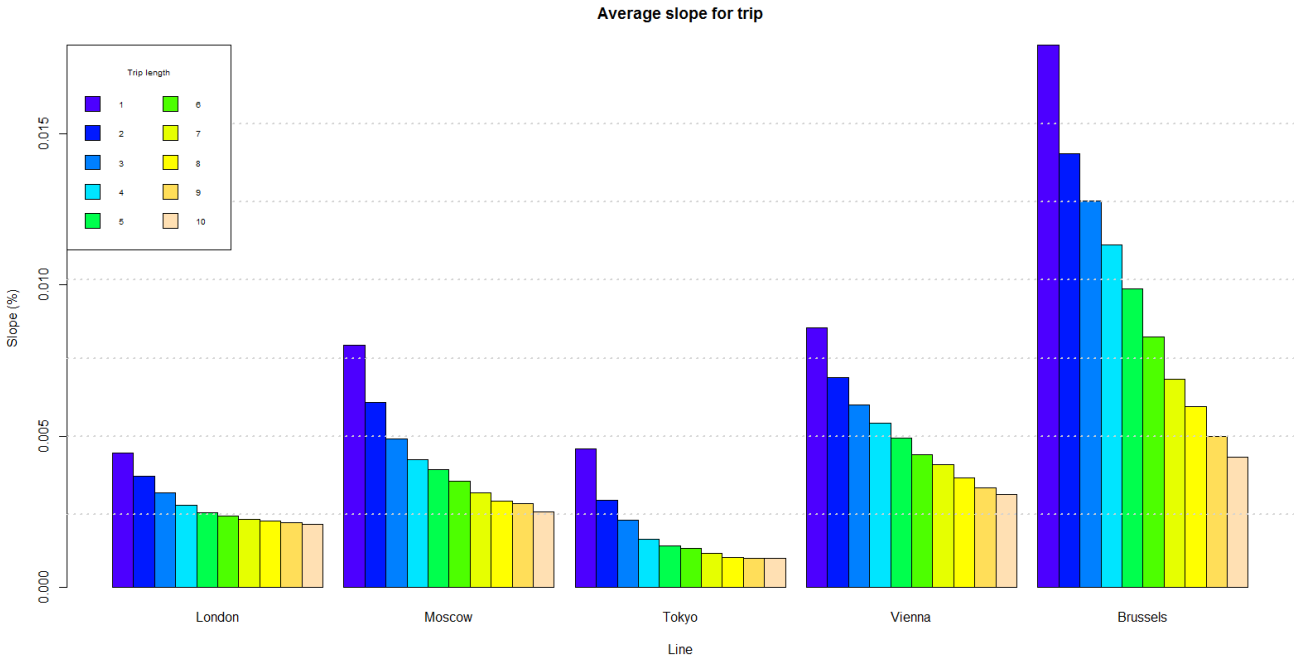

Figure 3.18: Average slope for a trip of length n

We also see a big difference between Brussels and the other cities when we look at the slope between the subway stations. With an average slope of 0.97%, Brussels' average slope is more than twice as big as London's (0.44%).

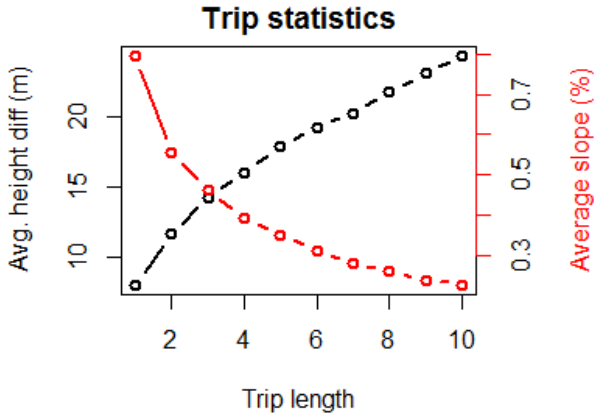

Figure 3.19: Average height difference and slope for trips

As expected, the average height difference between stations increases when the trip length increases (figure 3.19). The average slope decreases when the trip length increases. A negative slope is often followed by a positive slope (e.g. when the track needs to run temporarily deeper underground). Over longer trips, this evens out.

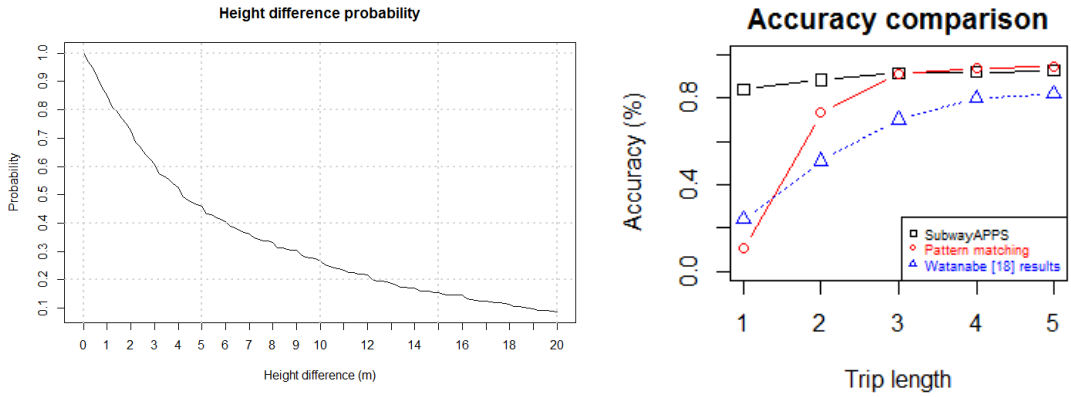

Figure 3.20: a) Chart showing the probability (y-axis) that the height difference between two stations is above a threshold (x-axis), b) Accuracy of pattern matching algorithm (red) [32] compared to our approach SubwayAPPS (black). Please note, that in this case we assume that the pattern matching approach [32] works with a 100% accuracy. The blue dotted line shows the accuracy achieved by [32]

3.3 TECHNICAL EVALUATION

In general, we see that the average height difference between stations for all lines is well above 2 meters (the granularity of the air pressure sensors is around $1m$) for the researched five global subway systems, therefore our approach works in general across all networks. To further analyse this, we calculated at the probability that the height difference between two adjacent stations exceeds the air pressure sensors accuracy. As expected, the probability that the height difference between two adjacent stations is bigger than $0m$ is 100% and the probability naturally decreases when the height difference between two adjacent stations increases (figure 3.20). The probability that two adjacent stations have a height difference of 0.5m is still 91.38%. For a height difference of 1m, this probability is decreased to 82.11%. Therefore, in a worse case scenario, our approach can detect 82.11% of all 1-stations rides across the five networks. Of course, this is a very theoretic value, as the typical subway ride is about 8.75 minutes long and takes 6 stations [28]. For a 2-stations ride, we already can technical detect 88% of all trips and for a 4-station ride nearly 92% (figure 3.20).

Nevertheless, this highlights the difference to the template matching approach by Watanabe et al. [32]. We have also calculated how unique the height difference patterns are across different lines, cities and across the five networks we have investigated. On average, 54% of 1-station trips are unique in an individual line. This increases to 96% for 2-station trips. If we look at the uniqueness of a height difference pattern at the level of a city, we see that only 10% of 1-station patterns are unique. This increases quickly, with 94% uniqueness for a 4-station

trip. This is the same as the uniqueness of 4-station trips at worldwide level. This explains why the template matching approach performs bad on short trips. This improves when the trip length increases.

Chapter 4

Concept

In previous sections we looked at the properties of air pressure, how it behaves in closed environments like underground subway tunnels and how it can be measured with a sensor in a mobile device. Furthermore, we studied the depth structure of underground subway networks. In this section, we use this information to propose an algorithm for subway station detection using only a mobile air pressure sensor.

4.1 Overground Transportation

To get an indication if location detection using an air pressure sensor was feasible on transportation networks, some data was recorded while using overground transportation. The data was recorded using the air pressure logger application (see appendix A). The air pressure measurements during a 30 kilometer car trip were recorded, along with location data as provided by the operating system of the smartphone. The location data was used to create a height profile of the trip. Using the Google Maps Elevation API [7], the altitude of each point was retrieved. The relative altitude, as measured by the air pressure sensor, was calculated by using the hypsometric formula (see section 2.3). The result is plotted in figure 4.1.

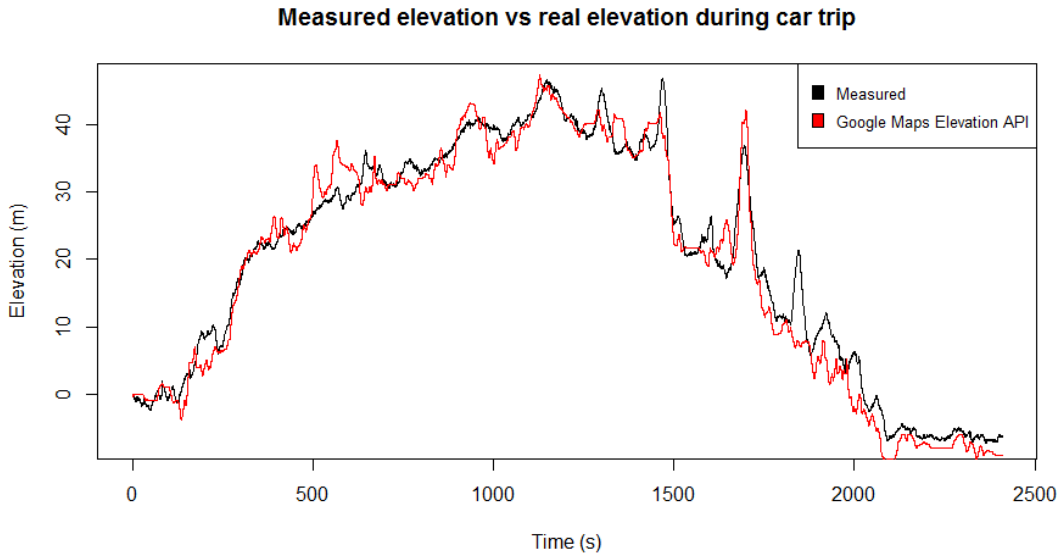

Figure 4.1: Measured elevation with an air pressure sensor (black), compared to real elevation according to the Google Maps Elevation API (red)

From this, we see that there is an apparent relationship between the elevation and the readings of the air pressure sensor in a mobile device while during movement. The absolute average measured height error of the measured elevation, compared to the elevation acquired from the Google Maps Elevation API, for this trip is 2.35m. The big difference between the measured height and the height as retrieved from the Google Maps Elevation API at 1800 seconds, is due to a bridge. The Elevation API returned the elevation below the bridge, while the car drove over the bridge.

However, we should take into account that a car trip is significantly different to a subway trip:

- The car trip was recorded completely in open air. Most subway trips are completely inside tunnels. This can have an effect on the air pressure due to the piston effect.
- A subway train stops at every station. This allows the air pressure to stabilize while the train is stationary. While riding, there is a constant movement of air around the vehicle.
- When driving a car, we only have to stop for traffic lights and holdups. These incidents happen less frequent with subway rides.

- When the subway train has stopped at a station, the doors of the train open. This can create an air flow between the train and the station which influences the air pressure in the train.

4.2 UNDERGROUND TRANSPORTATION

In the next phase, we recorded the air pressure measurements while using an underground transportation network. Five subway trips on three different lines were recorded on a single day in the Brussels underground network. To make sure the recordings were representative, recordings were made in the front, middle and the back of the train.

The recorded data was plotted to analyze the characteristics of air pressure during a subway trip. An example is shown in figure 4.2. Pressure in hPa is plotted on the vertical axis, the time in seconds on the horizontal axis. To indicate whether the subway train was stationary or riding, the background of the chart is altered. A gray background indicates that the train is riding, a white background means that the train is at a station.

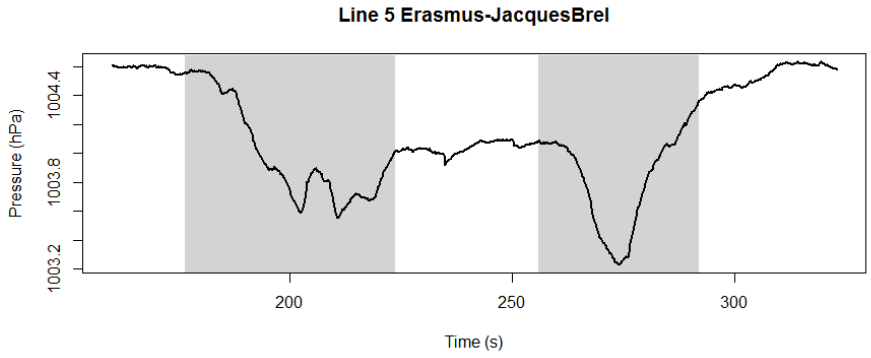

Figure 4.2: Example of air pressure during a subway trip in Brussels

From these measurements, it was clear that the behaviour of air pressure during a subway trip depends on the position on the train, whether the train is in a tunnel or not and the characteristics of the subway tunnel (tunnel width, presence of ventilation shafts etc.). To make our proposed method work universally on all subway lines worldwide, our method needs to take into account these variables. No assumptions about these variables should be made.

4.2.1 Train Arrival Detection

The piston effect is not only measurable when riding a subway train. Changes in air pressure are also present while standing at a station platform when a subway train arrives and departs. Two air pressure recordings were made at the *Stokkel* and *Zuid* stations in Brussels, shown in figure 4.3. A white background indicates a train is at the station, a gray background indicates no train is at the station.
Again, the effect of the piston effect depends on the position of the measurement on the station platform. When standing at the beginning of the platform (where the train arrives), there is an increase in air pressure due to the air stuck in front of the train, followed by a decrease to fill the vacuum behind the train. Finally, the air pressure normalises again to its original state.
When a train departs, there is a sudden decrease of the air pressure to fill the vacuum behind the train (± 150 seconds on the right chart).
This makes it possible to detect the arrival and departure of a subway train by looking at the behaviour of air pressure while standing on a subway platform.

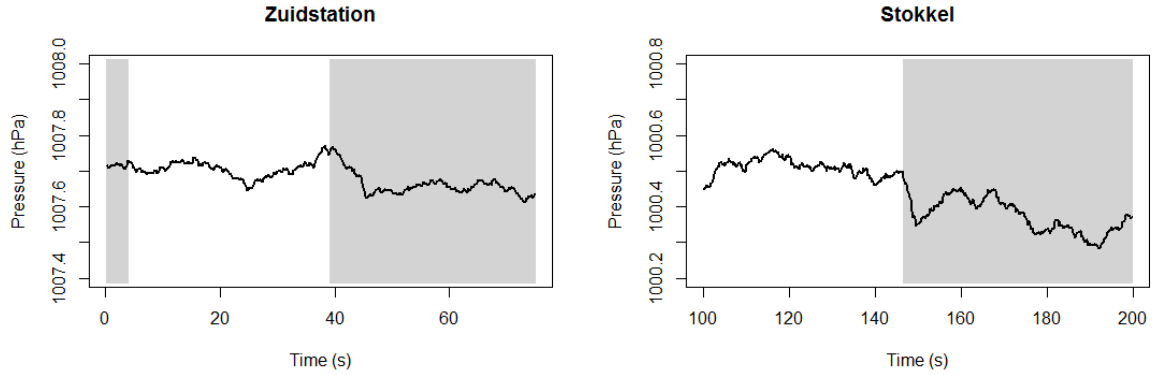

Figure 4.3: Example of air pressure at a subway station

4.3 SUBWAYAPPS

With the SubwayAPPS technique, we show that it is possible to achieve reasonable positioning accuracy on underground transportation networks. SubwayAPPS makes use of the smartphone's barometer by relying on the piston effect and height differences between subway stations. In our current implementation, we also expect the user to provide a start station. This could also be done automatically by selecting the closest subway station near the last known GPS coordinate.

As described earlier, some smartphones already filter the air pressure data on the chip. If the air pressure sensor does not have an internal filter, the received air pressure is filtered using an alpha filter, as used by [24].

We use the piston effect to detect whether the train is currently riding or not. When the train is riding, the piston effect, together with variations in the environment (width of tunnel, ventilation shafts etc.) cause the air pressure to be unstable. When the train has arrived at a station, the air pressure becomes stable again.

To detect if the air pressure is currently stable we look at the variance of the air pressure in a sliding time window. Variance is a statistic measure that indicates the variability and spread of a set of values by looking at the distance between the values and the mean of the set [2]. It is given by the formula $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2$. The square root of the variance is called the standard deviation: $s = \sqrt{s^2}$. When all values in the data set are the same, the variance will be 0. If there is a big spread between the values, the variance will be higher.

There are two parameters necessary to decide whether the air pressure is currently stable or not: the size of the sliding time window and a threshold that determines the boundary between stable and unstable air pressure.We experimentally found the optimal values for the parameters by using different values and count the number of undetected station arrivals and false positives that work across all networks. The optimal values for the sliding time window and threshold are 12 seconds and 0.0015 hPa respectively.

To further improve the SubwayAPPS algorithm, we make use of the height differences between the stations. As mentioned in section 2.3, we can calculate the height difference between two points by their air pressure at these two points by using the *hypsometric formula*:

$$h = ((\frac{P_0}{P})^{-0.19} - 1) \cdot -44330)$$

When we detect that the air pressure is stable, we can not assume directly that the train is stationary. Instead, we first look if the relative height difference between the current position and the previous station, measured by the smartphone's air pressure sensor, matches the real height difference between the stations. Only when the air pressure is stable and the height difference check succeeds, we can conclude that the subway train has arrived at a station.
We experimentally observed that a tolerance of 2 meters between the actual height difference

and the measured height difference is optimal to detect most stations and eliminate false positives. In section 3.2, we concluded that 82.11% of adjacent stations have a height difference above 2 meters. With this height difference check, we can thus eliminate false positives as seen in figure 6.4 if the next station is located more than 2 meters above or below the current station.

Additionally, the height difference check prevents that an unscheduled stop between two stations is classified as an arrival at a station. When the subway train has stopped between two stations, the air pressure can become stable. In this case, the height difference check will fail and the period of stable air pressure will rightfully not be seen as an arrival at a station.

An example of the algorithm with height difference check is shown in figure 4.4. The variance of the air pressure, height difference and result of the algorithm are shown below each other. The blue dotted lines on the height difference chart shows the tolerance for the height difference check. The height difference check succeeds if the measured height difference between the previous stop and the current position is between the two dotted lines. We can see how the false positive between 450 and 500 seconds is filtered out in this version of the algorithm due to the height difference check. After the first stable air pressure period ends, the algorithm only expects a new stop at a relative height difference 9 meter below the current stop. Because of this, the second period of stable air pressure is not seen as a stop, because the relative height difference is $\pm$ 0 meters.

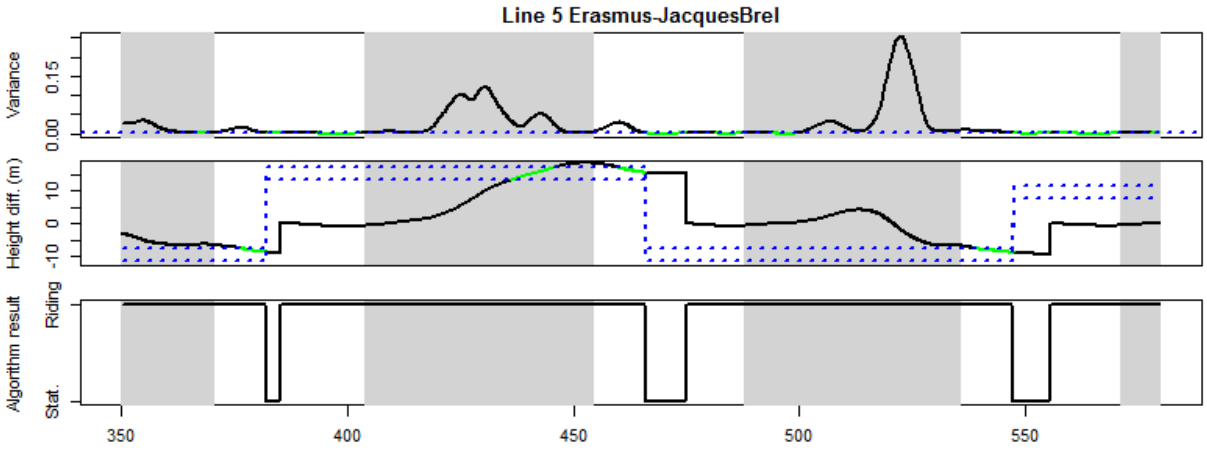

Figure 4.4: Example of the SubwayAPPS algorithm. The top chart shows the variance of the air pressure in the time window. When the variance is below the threshold (blue dotted line), the air pressure is stable. The chart in the middle shows the relative height difference measured between stations. The blue dotted line shows the expected height difference ± tolerance. When the air pressure is stable and the height difference is as expected, the algorithm concludes that the train has arrived at a station (bottom chart).

Chapter 5

Implementation

The SubwayAPPS algorithm introduced in the previous section is implemented in an application on the Android platform, for versions 4.0.4 and up. The application is based on the existing MetroNavigator application, developed by Stockx et al. [28] in their Sigspatial 2015 paper. Furthermore, to facilitate the research and evaluation, the algorithm is implemented in Javascript and R. In this section, we will discuss the details of the implementation of the algorithm in the Android application.

5.1 Implementation on Android

To access the air pressure sensor of the smartphone, we use Android's built-in API to access environment sensors [13]. Once registered, this API returns the air pressure at 5Hz in hPa. Before we can use the returned air pressure for the calculations of the algorithm, smoothing may be necessary. A boolean variable keeps track whether smoothing is recommended. By default, the value is set to *true*. If an air pressure sensor has built-in smoothing (e.g. Bosch BMP280), this can be turned off. The vendor and version number of the sensor can be retrieved using the Android API as shown in listing 5.1. In this example, we check if the version of the Bosch air pressure sensor is better than the Bosch BMP180 (version 1, with no built-in filtering).

```
1 boolean doFiltering = true;
2 if (mPressure.getVendor().contains("BOSCH")
3   && mPressure.getVersion() > 1) {
4         doFiltering = false;
5 }
```

Listing 5.1: Code for retrieving information about the built-in air pressure sensor

We use the same filter as [24], with $\alpha = 0.1$:

$currentPressure = \alpha * sensorPressure + (1 - \alpha) * previousPressure$

This gives results comparable to air pressure sensors with built-in filtering (see section 6.1).

Throughout the duration of the algorithm, we keep a time window to check whether the air pressure is stable or not. We store this values in an *ArrayList*. This allows for quick access to the head (to add the most recent air pressure measurement) and tail (to remove values that fall out of the time window) of the time window. Along with the air pressure, we also store the timestamp of the air pressure measurements in a separate list. This is needed to remove the outliers when updating the time window. The time window is updated each time a new measurement is received by the Android API. The algorithm for updating the time windows is shown in listing 5.2. The size of the time window is 12 seconds, as discussed in the previous section.

```
1  //add new value
2  pressureTimeWindow.add(newPressure);
3  timestampTimeWindow.add(newTimestamp);
4
5  //remove values outside the window
6  while (timestampTimeWindow.size() > 0
7    && timestampTimeWindow.get(0) < newTimestamp - timeWindowSize) {
8          pressureTimeWindow.remove(0);
9          timestampTimeWindow.remove(0);
10 }
```

Listing 5.2: Code for updating the time windows

We keep a boolean, *stopDetected*, to save the current status of the train (riding or stationary). When the algorithm has just started and the time window is not full yet, the variance of the

values in the time window will be low. The algorithm decides that the air pressure is stable. This is acceptable because we assume that the user starts the application when he boards the subway train and it is still waiting to depart. The value of *stopDetected* is thus initialized as true.

When a new value is received from the air pressure sensor, the value is smoothed (if necessary), the time window gets updated and the variance of the time window is calculated. The continuation of the algorithm depends on the stability of the air pressure and the current value of *stopDetected*:

- Stable air pressure:
  - When currently a stop is detected and the measured air pressure is still stable, the status of the train remains stationary. A special window is kept with air pressure measurements when the train is stationary. This is used to calculate the average air pressure during the whole period that the train is stationary. This improves the accuracy of the height difference checks.
  - When the air pressure becomes stable and the current status of the train is riding, there is a possibility that the train has arrived at a station. To assure that this is the case, a height difference check is done. The relative height difference between the current air pressure and the average air pressure at the previous stop is calculated. If this matches the expected height difference, with a tolerance of 2 meters, the algorithm decides that the train has arrived at the station. The values of the time window are copied to the stationary time window and *stopDetected* is set to true.
- Unstable air pressure:
  Unstable air pressure always is an indication that the train is riding. If a stop was detected previously, the train has departed from the station again. The average air pressure from the stationary time window is stored and *stopDetected* is set to false. When no stop was detected, the train remains in its riding state and the algorithm does not take any action.

An overview of the algorithm described above is shown in figure 5.1. The implementation of the algorithm, in pseudocode, is given in appendix B.

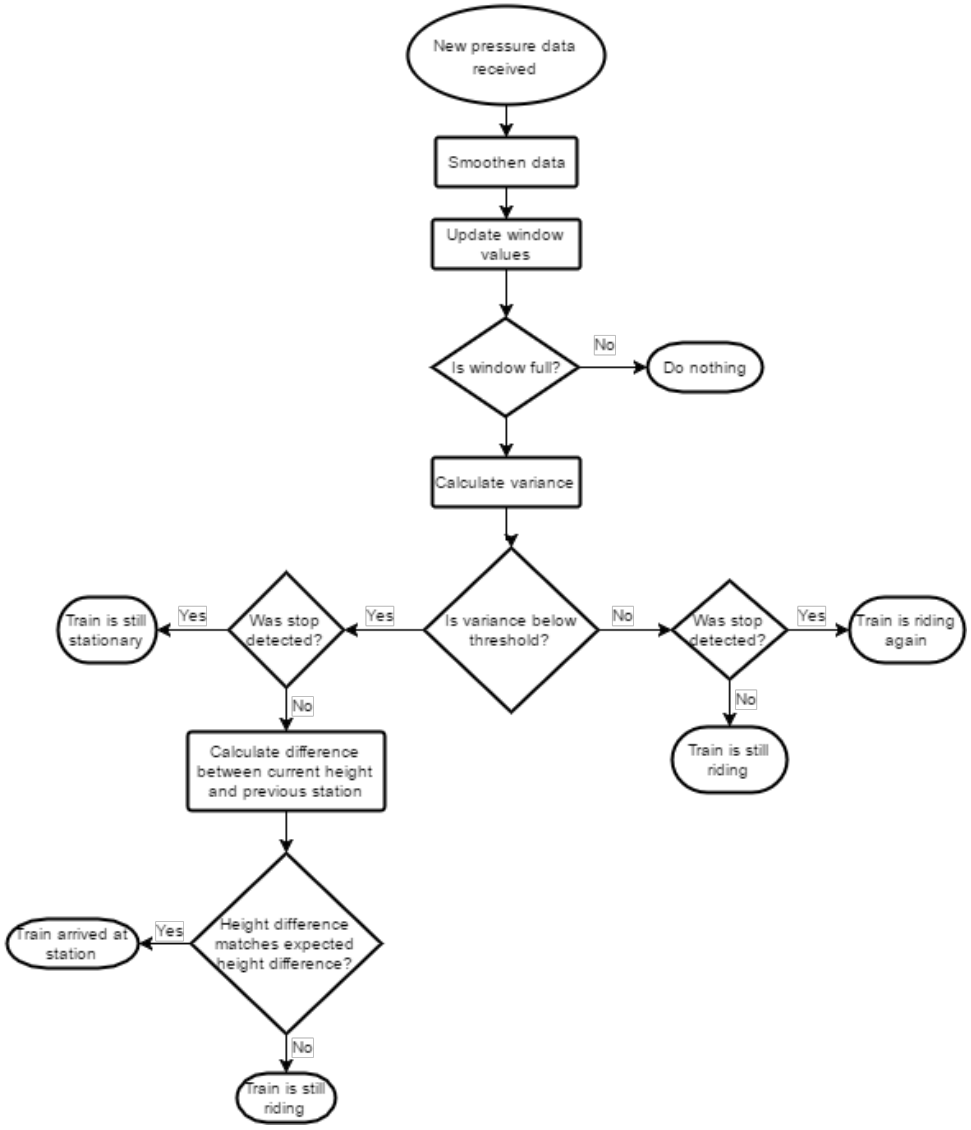

Figure 5.1: Graphical representation of the station detection algorithm using air pressure

5.2 METRONAVIGATOR+

The algorithm described above runs as a background service. When an event, e.g. train arrived at a station, happens, the service sends an *intent* (message) to the Android OS. Every application on the OS can register to receive these intents. The receiving application can then use this information for navigational purposes etc.

One such application that listens for these intents is MetroNavigator+, an application developed by Stockx et al. [28]. The application lets users select a start and destination station.

It then tracks the subway trip and shows number of stations left, time left to next station, the name of the next station, the estimated arrival time and a visualization of the progress between the previous and next station. While riding, the user can share tweets about his ride and see Wikipedia information about landmarks that he passes. Finally, a smartwatch extension allows the user to follow the progress of his trip on his watch.

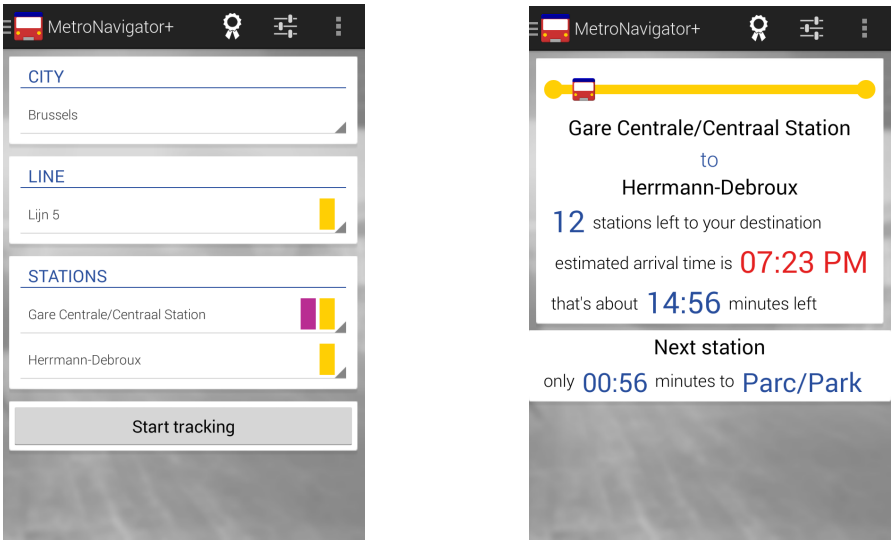

Figure 5.2: Screenshots of the MetroNavigator application

The SubwayAPPS algorithm needs to know the relative height differences between the adjacent stations of the trip. Because of this, the storage of the station data in the MetroNavigator application needed to be changed. The data is saved in JSON format, as shown in listing 5.3. To facilitate the querying of the data, a datastore class was implemented to retrieve the information.

```
[
  {
    "City": "London",
    "Latitude": 51.528308,
    "Longitude": -0.3817765,
    "Lines": [
      {
        "Name": "Bakerloo",
        "Color": "#B36305",
        "Stations": [
          {
            "Name": "Elephant & Castle",
            "Latitude": 51.495849394281,
            "Longitude": -0.10072431173602
          }
        ],
        "Segments": [
          {
            "From": "Elephant & Castle",
            "To": "Lambeth North",
            "HeightDifference": 6, //in meters
            "Time": 2.13 //in minutes
          }
        ]
      }
    ]
  }
]
```

Listing 5.3: JSON format for data in the MetroNavigator application

Chapter 6

Evaluation

In this section, we evaluate our proposed SubwayAPPS method. First, we look at the performance of air pressure sensors to detect height differences. Next, we calculate optimal parameter values for the SubwayAPPS algorithm. To conclude this chapter, we evaluate the algorithm in Brussels and London and compare it to Stockx' method [28].

6.1 Technical Evaluation

The manufacturers of pressure sensors claim that the sensors have an accuracy of ±0.1 hPa. This corresponds to a height difference of ±1 meter. To make our proposed method function properly, it is important that a good accuracy is achieved. Height accuracy tests were executed to test the accuracy of the Bosch BMP180 and Bosch BMP280 pressure sensors.

The tests were executed by a Google nexus 4 (Bosch BMP180) and an Apple iPhone 6 (Bosch BMP280). Both phones were put in a pocket. The pocket was attached to a rope which was marked every 10cm. The pocket was then hung over a balcony. Every 10 seconds, the pocket was manually lowered 10cm and then kept stable at the same height. This was repeated until a distance of 230cm was completed. The air pressure logger application (see appendix A) was used on both phones to record the air pressure. To record the real current height, a simple stopwatch application was developed using JavaScript. This application recorded the current timestamp and height when a button was clicked. The output was exported to a *csv*-file. To make sure the clocks of both the phones and computer were in sync, they were synchronized by using NTP.
The results were analyzed using the R programming language. The height difference, relative

to the start height, is calculated by using the hypsometric formula explained in section 2.3. The results are visualized in figure 6.1.

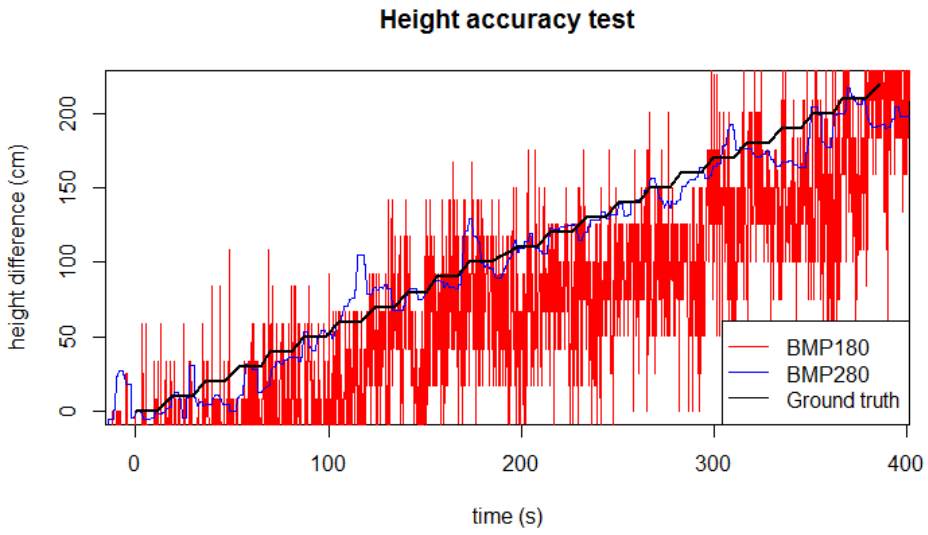

Figure 6.1: Unfiltered results of the height accuracy test

We can see that there is a significant difference between the BMP180 (red) and BMP280 (blue) sensors. The readings of the BMP180 sensor are not smooth and have great variance. The BMP280 has a built-in noise filter that smooths the results internally. The readings of the BMP180 sensor can be smoothened manually by using a simple filter: $currentHeight = \alpha * sensorHeight + (1 - \alpha) * prevHeight$, with $\alpha = 0.1$. The filtered results are shown in figure 6.2. For completeness, the results of the BPM280 were filtered using the same filter as well.

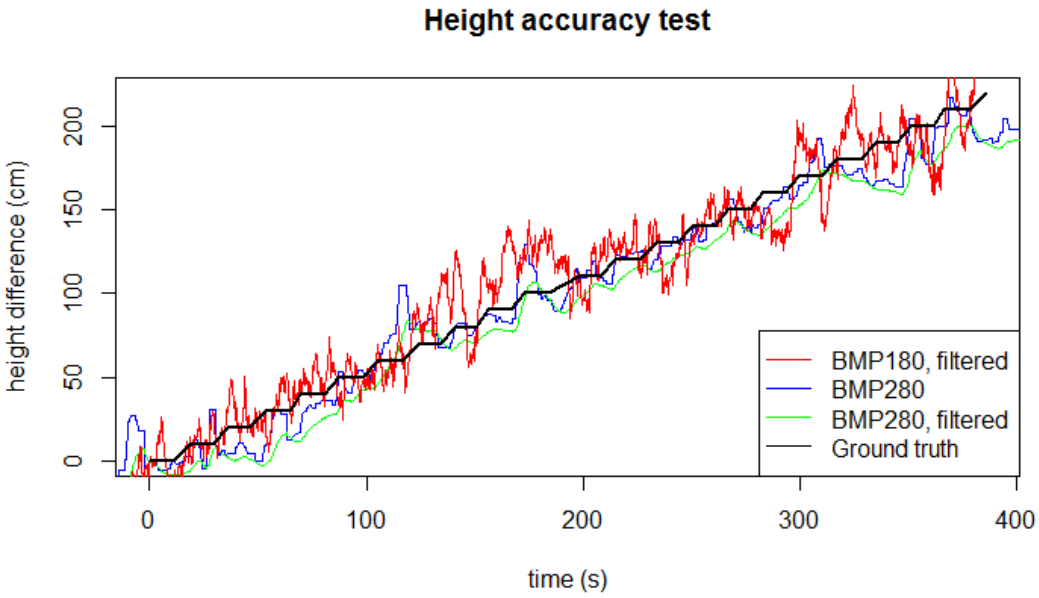

Figure 6.2: Filtered results of the height accuracy test

To compare the air pressure sensors, statistics about the error of the measured height difference were calculated. This was done by looking at the absolute difference between the measured height difference and the real (ground truth) height difference for the periods when the pocket was kept stable at the same height. A boxplot of these statistics is shown in figure 6.3.

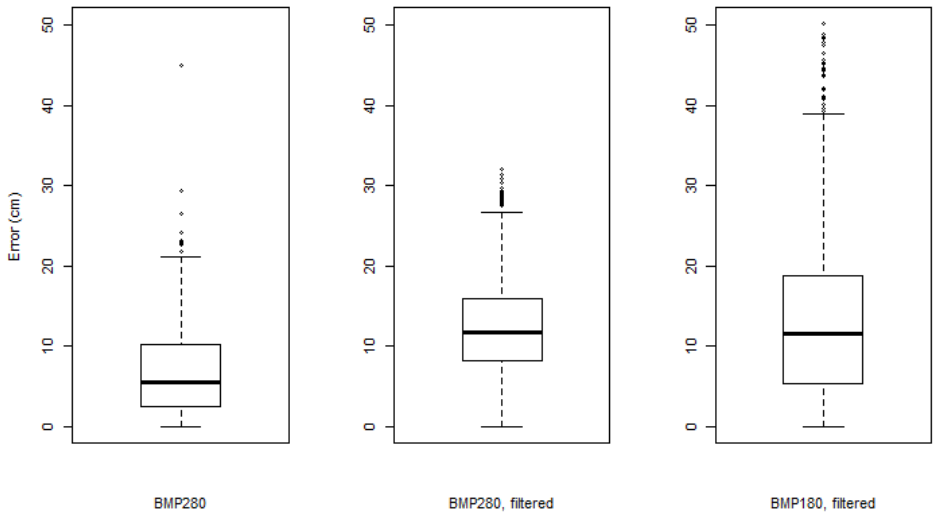

Figure 6.3: Boxplot of the error of the height accuracy tests

It is clear that filtering the results of the BMP280 again does not improve the results. The internal filter of the BMP280 is the optimal filter, further filtering worsens the result. The average height error for the BPM180 is 13.43cm, with a maximum error of 50.09 cm, which is comparable to the results of the BPM 280, 7.88cm and 44.93cm respectively. This is a better accuracy than is indicated by the manufacturer of the pressure sensors. The boxplot also clearly shows the bigger variance of the BMP180 sensor.

In section 3, we investigated the height differences between adjacent subway stations. We concluded that 91.38% of adjacent stations have a height difference of at least 0.5m. If we combine this with the results of the height accuracy tests, we can conclude that we can detect 91.38% of subway stations by only taking into account the height differences between subway stations.

6.2 PARAMETER DETERMINATION

The SubwayAPPS algorithm depends on two variables to execute the station detection: the size of the time window that holds the measured air pressure values and a threshold that decides whether the air pressure is stable or not.

To make a good assumption about accurate values for these parameters, we manually marked the time intervals on our recorded air pressure logs where the air pressure was stable. We found that the average duration and variance for these intervals were 16.2 seconds and 0.0015 hPa respectively. For further optimization, we manually found the optimal values for the parameters by using different values and compare the results with the real truth. The following statistics were used:

- **Undetected stations**: the number of stations that was undetected
- **False positives**: number of times stable air pressure was detected while the train was still riding
- **Sum**: the sum of the above statistics

Undetected stations	0,0005	0,001	0,0015	0,002
5	0	0	0	0
8	2	0	0	0
10	6	2	0	0
12	13	7	0	0
15	20	16	13	8

False positives	0,0005	0,001	0,0015	0,002
5	109	170	204	206
8	38	63	89	104
10	16	24	39	55
12	7	12	20	29
15	4	2	6	12

Sum	0,0005	0,001	0,0015	0,002
5	109	170	204	206
8	40	63	89	104
10	22	26	39	55
12	20	19	20	29
15	24	18	19	20

Table 6.1: Results for different values for time window size (vertical) and threshold (horizontal)

From table 6.1 we can see that the number of undetected stations decreases when the time window size decreases and the variance threshold increases. However, the number of false positives increases in this case. Because we can still filter out the false positives by making use of the height information between adjacent stations, it is more important that the number of undetected stations is zero. We conclude that the optimal values for the time window size and variance threshold are 12 seconds and 0.0015 hPa respectively. With these values, we still have 20 false positives, meaning that the algorithm has detected 20 stops while the subway train was still riding. An example of the station detection algorithm making only use of the variance of the air pressure is shown in figure 6.4. The variance of the air pressure in the time window is plotted over time. The threshold between stable and unstable air pressure is indicated by the blue line. When the variance is below the threshold, it is plotted in green.

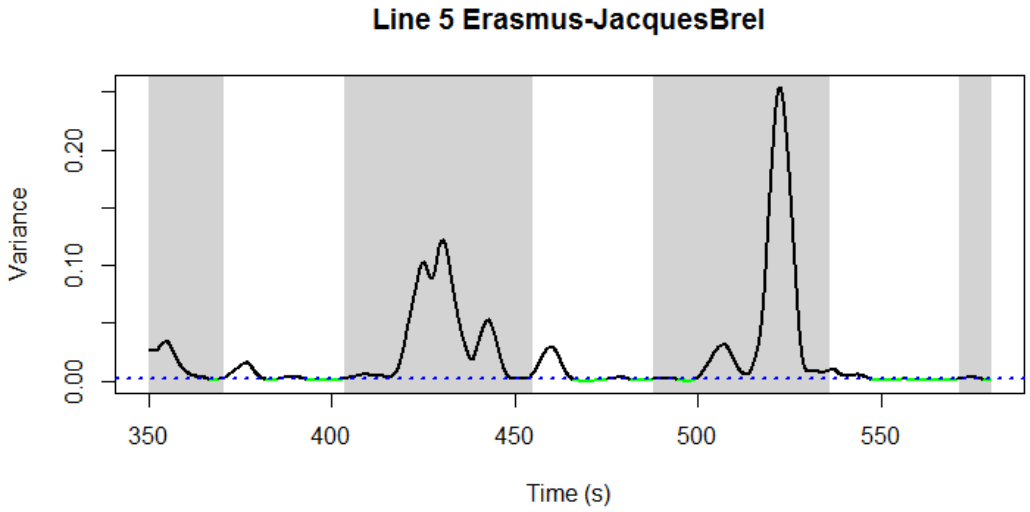

Figure 6.4: Example of the station detection algorithm using only stability of air pressure

All station arrivals are detected. It is clear that when the train is riding, the variance of the air pressure is significantly higher than when the train is stationary. We can also see some false positives, more specifically between 450 seconds and 500 seconds. We see that the air

pressure becomes stable as the train arrives at the station. Then, while still stationary, it becomes unstable for a short while. This happens twice before the train starts riding again at ± 500 seconds.

6.3 EMPIRICAL EVALUATION

To evaluate the SubwayAPPS algorithm, a real-world study was done in the subway networks of Brussels and London. A logger application was developed to record the air pressure, along with information about the current station and the riding status of the train, while riding the train (see appendix A). This application saves the recorded data in the CSV format. The logs were recorded with a Google Nexus 4 (Bosch BMP180 pressure sensor) and iPhone 6 (Bosch BMP280 pressure sensor). The logs were recorded at a random position in the train, to prevent the position on the train from influencing the evaluation results. The phone used to record was kept on the lap or in the pocket of the user.

The algorithm is implemented in R. The recorded logs are imported and the algorithm runs over the data. The result is plotted as four charts:

- The status of the train (riding or stationary) as indicated by the user while recording the log
- The prediction of the algorithm of the status of the train
- The current variance of the air pressure in the sliding time window
- The current height difference compared to the previous detected station

This allows us to easily count the number of detected stations and to analyze why the algorithm fails. When an arrival at a station is not detected, no recalibration is done. This causes the subsequent stations on the trip to be undetected, because the algorithm relies on relative height differences between adjacent stations.

6.3.1 Results

In Brussels, the algorithm was tested on two separate days. In total, 20 recordings were made with an average time of 22 minutes. The average number of stations per trip was 14. The BMP180 sensor was used for 12 recordings, the other were recorded by the BMP280 sensor. All lines were recorded completely at least once.

With the BMP180 pressure sensor, 162 out of 170 stations (95.29%) were detected correctly. 8 out of 12 (66.66%) complete trips were detected correctly.
Using the BMP280 pressure sensor, 92 out of 114 stations (80.70%) were detected correctly. 5 out of 8 (62.5%) trips were detected correctly.

A second test was done in London. 9 logs were recorded with the BMP180 sensor and 10 with the BMP280 sensor. The trips were randomly chosen on the London underground network in zones 1 and 2. The goal of this test was to board the train, ride it for 7 minutes and then exit at the next stop. The average trip time of this test is 9 minutes. The average length of a trip was 4 stations.
Here, the BMP180 sensor classified 20 out of 30 stations (66.66%) correctly. 5 out of 9 complete trips (55.55%) were successfully classified.
The BMP280 classified 21 out of 36 stations (58.33%) correctly. It could only detect 1 complete trip correctly.

	BMP180	**BMP280**	**Total**
Brussels	95,29%	80,70%	88,00%
London	66,66%	58,33%	62,50%
Total	80,98%	69,52%	

Table 6.2: Accuracy results of the evaluation

From this results, we can conclude that there is a noticeable difference between the Bosch BMP180 and BMP280 sensors. From the analysis, we see that some station stops are not detected because the air pressure readings are not stable enough to be under the stable threshold. A possible cause for this, is the built-in filter of the BMP280 sensor. This filter is not as strong as the alpha filter we use to filter the readings of the BMP180 sensor. Because the filter of the BMP280 is not as strong, the variance of the values in the time window does not decrease enough when the train is stationary.

Further, we notice a difference in performance between the tests in Brussels and London. From the analysis, we see two causes for this. First, we notice that the air pressure while stationary does not become stable enough for the algorithm to detect a stop. The subway stations in London are bigger and busier than those in Brussels. They often connect multiple lines. When a train arrives or departs from the same station but at a different platform, this destabilizes the air pressure in the whole station. Another difference between the test in

Brussels and London, is the source of the data of the height differences between the stations. The data for London comes from TfL and dates from 2011. We have no control over the accuracy of the data. Changes in the depth structure of some lines may have happened. The data for Brussels is derived from measurements done by the same pressure sensors that we use to do the test. This is more suitable for the execution of the algorithm.

The results for the Brussels underground network are slightly better than Stockx' [28] method, which uses the accelerometer to do station detection. However, this method performs better for the London underground network. We expect that the performance of our method improves when the depth structure of the network is mapped using the same air pressure sensors used for the SubwayAPPS algorithm.
Our method does not depend on the type of train or the smoothness of the tracks, something that negatively influenced Stockx' algorithm. Another advantage of our method is that we can rely on two properties of air pressure: the piston effect and the changing air pressure at different altitudes. This allows us to filter out most false positives. A downside of SubwayAPPS is that the depth structure of the underground network must be known prior to tracking. This information is not publicly available for all underground networks. Preferably, the depth structure is mapped with an air pressure sensor, for accurate results.

Chapter 7

Discussion and Conclusion

In this thesis, we presented SubwayAPPS, a novel method that uses the air pressure sensor in a smartphone to detect the riding status of a subway train. SubwayAPPS uses the characteristics of air pressure in closed environments (like subway tunnels) and the height differences between adjacent stations to execute the detection. It does not need any additional instrumentation of the environment, sparing the operators of subway networks who are often on a low budget. The only information the algorithm needs, is a start and end station and a depth map of the subway line (gathered manually using an air pressure sensor or otherwise).

In our first study, we showed that the height differences between adjacent subway stations are big enough for air pressure sensors in modern smartphones to detect. For this, the subway networks of five major cities (London, Moscow, Tokyo, Vienna and Brussels) were analyzed. Furthermore, we tested the accuracy of air pressure sensors to detect height differences. From this, we saw that we could theoretically classify 82.11% of subway stations by looking at their height differences alone.

Our evaluation shows that our method, when the depth structure of the underground network is mapped with an air pressure sensor, can get a 10% better accuracy than existing solutions. Despite this, the algorithm is simple to understand and can be implemented in less than 100 lines of code. As a proof of concept, the algorithm is implemented on the Android platform with the MetroNavigator+ application.

7.1 LIMITATIONS

However, there might be various reasons why the algorithm might fail at some times. As mentioned before in our evaluation of the algorithm in London, arriving or departing trains on other platforms in the same station can influence the stability of the air pressure. This can obstruct the station detection algorithm.

When a train brakes before it arrives at a station and then drives slowly into the station, the algorithm may be detecting the stop too early. This can be solved by using additional sensors like the accelerometer or a brightness sensor that detects whether the train is still in the tunnel or not.

Furthermore, drastic changes of the weather change the air pressure universally. During an intense storm, an air pressure change which corresponds to a height difference of 3 to 4 meters can happen in only 10 minutes. The probability that this happens is small, as the average riding time between adjacent stations is 1 to 2 minutes. We can resolve this issue by involving weather information (e.g. base pressure at sea level) for the calculations of the algorithm. This way, we can compensate for the drift in air pressure due to weather conditions.

During rush hours, trains can be very crowded. The user might only be able to sit down during a part of the trip. Alternating between a sitting and standing position changes the height of the smartphone. Even when the user remains in the same position during the whole trip, putting the phone from his hands into his pockets or vice versa changes the height of the phone. This can be resolved by using an external air pressure sensor. This sensor can be attached to a body part that remains at the same height, e.g. the user's shoe.

Finally, the algorithm fails when the services of the subway network change. An example of this, is the temporary closure of a station due to adjustments of the station. The algorithm will expect the train to stop at the closed station, but can not detect this when the train rides past the station. Consequently, all next stations can not be detected by the algorithm, because the expected height difference is not adjusted. To resolve this, the application should frequently receive service updates from the underground network operator. The algorithm can take this info into account and skip the closed station.

The algorithm can not recover when it misses a station stop. As a consequence, all subsequent stations are not detected as well. A recalibration function could put the algorithm back on track. For this, the algorithm could use another localization method (e.g. WiFi positioning), or he could manually indicate the current station during a stop. The algorithm can then resume its tracking.

7.2 FUTURE WORK

7.2.1 Piston effect

The presence of an air pressure sensor in mobile devices opens up new possibilities for applications. The piston effect is an interesting phenomenon that can be detected by mobile devices.
Although the effect is greater when an object is moving inside a closed and narrow environment, it also exists when an object moves in an open environment. in our tests, we saw that our algorithm works for overground segments as well. this suggests that looking at the stability of the air pressure might be useful for transport-mode detection.

Another situation where the piston effect occurs, is when an elevator moves through an elevator shaft. The elevator cabin pushes the air away while moving. The air can escape the elevator shaft through the landing doors (if they are not air-tight). This can possibly be detected by an air pressure sensor in a mobile device.

7.2.2 Pattern Matching with Neural Networks

From our analysis of the behavior of air pressure while on a subway train, we saw some recurring patterns, mainly due to the piston effect. The air pressure was mostly unstable when the train was riding, and stable when the train was stationary. We detected these patterns by looking at the variance of the air pressure in a sliding time window. Another method to use this patterns for station detection is making use of neural networks. A neural network is a set of artificial processing units (neurons) which are interconnected via a set of weights [5]. Signals can then travel from one or multiple input nodes, through these processing units via the connections, to one or multiple output nodes. This network is based on the decision making via neurons that happen in the human brain. Such a network is well-suited for tasks like pattern recognition.

In our case, a neural network can be used to make a decision whether the train is stationary or not by looking at patterns in the measured air pressure. Before we can use the network to make this decision, a training phase is needed. We can do this by feeding the network the recorded air pressure, together with the manual indications of the status of the train.

7.2.3 Train Arrival Detection in Station

From our test data, we noticed a change in air pressure when a train arrive or departs while standing on a station platform. This can be used to notify the user of an arriving subway train. It can also aid the SubwayAPPS algorithm to decide when to start the execution of the algorithm.

7.2.4 Crowdsourcing Data

From our evaluation, we concluded that the accuracy of the algorithm increases when the height structure of the subway line is mapped by an air pressure sensor. Mapping a complete subway network is a time-consuming job. To facilitate this, users who use the algorithm while riding the subway could automatically record the measured air pressure data. This recorded data can then be uploaded for processing to a server. With more information, a better estimate of the real height differences between the stations could be calculated.

7.2.5 Sensor Fusion

Previous research has shown that other sensors can be used for localization when riding a subway network. The accelerometer [15,28,29] and magnetometer [15] have been successfully utilized for location purposes. To develop a better algorithm, these existing algorithms could be fused. This algorithm could use other techniques to overcome the shortcomings of a method. For example, the height difference check from SubwayAPPS could be used to filter out false positives for an accelerometer based method. A weighted average could be used to make a decision about the riding state of the train.

Appendices

Appendix A

Air Pressure Log Application

To gather air pressure data while using public transportation networks, an application that logs the air pressure was developed. The app was developed for the Android and iOS platforms and used the pressure sensor present in the smartphone. By developing our own application we had have more control over the logger. We are sure that we log with the maximum accuracy and can define the output format. Finally, we can store additional information such as the geographical coordinates, whether the vehicle is stationary or riding and the current station. The interface of the application is shown in figure A.1.

The logger application logs the following information:

- The date and time. This is saved in a human-readable format and as a timestamp, which makes it easier to create charts with Microsoft Excel.
- The air pressure. This is done through the built-in pressure sensor of the device [13]. If no pressure sensor is available, an alert dialog will be displayed to notify the user.
- The moving state of the vehicle. This is saved as a boolean value (0 = stationary, 1 = riding). The user has to indicate this manually.
- The name of the last station that the vehicle passed. The user has to indicate this manually in the application. The city, line and station can be selected through spinner widgets. Currently, the subway stations of Brussels, Paris and London are available. More cities can be added by specifying them in a JSON file.
- The geographical coordinates (latitude and longitude). This information is gathered through the built-in Android location API. This data is particularly useful when using

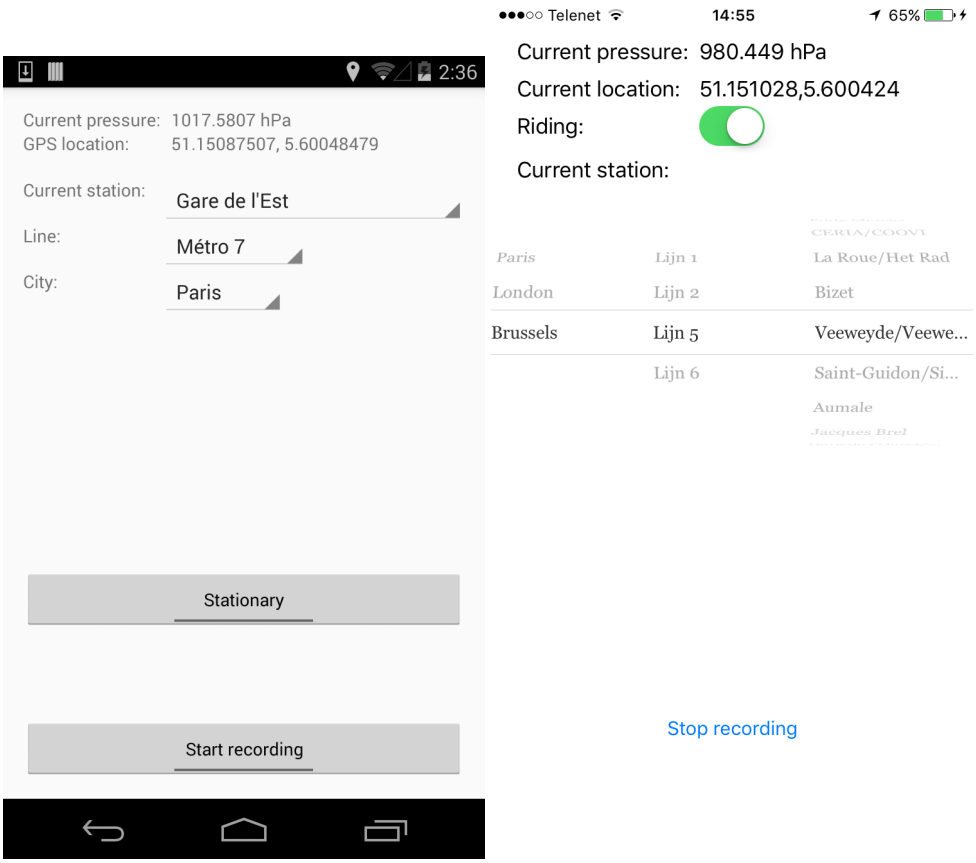

Figure A.1: The pressure logger application on Android (left) and iOS (right)

the logger application on overground public transport networks. Through the location data, the altitude of the location can be retrieved. This allows us to compare the calculated altitude (from the air pressure data) with the real altitude.

Each time one of these is updated, a new entry is added to the log file.

The logging data is saved to a csv text file. This makes it easy to use the data in Microsoft Excel or R to do further analysis. To make the file accessible, it is saved on the external memory of the phone (if this is available, if not it is saved to the internal memory), in the *Documents* directory. On iOS, the recorded log can be exported by using File sharing in iTunes.

Appendix B

Station Detection Algorithm Code

```
1  //the previous pressure value, used for smoothing
2  float previousPressure = null;
3  //array to save the pressure values in the time window
4  ArrayList<float> pressureTimeWindow = new ArrayList<>();
5  //array to save the timestamp values in the time window
6  ArrayList<long> timestampTimeWindow = new ArrayList<>();
7  //size of the time window, in milliseconds
8  private int timeWindowSize = 12 * 1000;
9  //the treshold for the variance of the pressure values
10 double varianceTreshold = 0.0015;
11 //boolean that tracks whether the train is currently stationary
12 boolean stopDetected = true;
13 //air pressure at previous stop, used to calculate the relative height difference
14 float previousStopPressure = null;
15 //array to keep the pressure values while the train is stationary,
16 //used to calculate previousStopPressure
17 ArrayList<float> stationaryWindow = new ArrayList<>();
18 private double heightTolerance = 2; //tolerance in meters
19 private Long startTimestamp = null; //timestamp of the start of the algorithm
20
21 function doStationDetection(long currentTimestamp,float currentPressure) {
22   if(startTimestamp == null) { //save start timestamp
23     startTimestamp = currentTimestamp;
24   }
25
26   //smoothen pressure data if necessary
27   if (doFiltering) {
28     currentPressure = smoothPressure(currentPressure);
29     //save pressure for next step
30     previousPressure = currentPressure;
31   }
32
33   //update window values
34   updateWindows(currentPressure, currentTimestamp);
35
36   //calculate variance
```

```
37    float windowVariance = calculateVariance();
38
39    if ((currentTimestamp - startTimestamp) * 1000 < timeWindowSize) {
40      //add pressure to stationaryWindow to calculate pressure of current stop
41      stationaryWindow.add(currentPressure);
42    } else {
43      if (windowVariance < varianceTreshold) {
44        //variance is below treshold
45        if (stopDetected) {
46          //Train is still stationary
47          //add current pressure to stationary window
48          stationaryWindow.add(currentPressure);
49        } else {
50          //calculate height difference
51          float heightDifference = calculateHeightDifference(previousStopPressure,
52            calculateMean(pressureTimeWindow))
53
54          if (Math.abs(heightDifference - expectedHeightDifference) < heightTolerance) {
55            //Train arrived at station
56            //create stationary window
57            stationaryWindow.clear();
58            stationaryWindow.addAll(pressureTimeWindow);
59            stopDetected = true;
60          } else {
61            //Train is still riding
62          }
63        }
64      } else {
65        //variance is above treshold
66        if (stopDetected) {
67          //Train is riding again
68          stopDetected = false;
69          previousStopPressure = calculateMean(stationaryWindow);
70        } else {
71          //Train is still riding
72        }
73      }
74    }
75  }
```

Listing B.1: Pseudocode of the SubwayAPPS algorithm

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] Michael Baron. *Probability and Statistics for Computer Scientists, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2013.

[3] BIPT. Kadaster antennesites. `http://www.sites.bipt.be/`. Consulted on May 11, 2016.

[4] Geoffrey Blewitt. Basics of the gps technique: observation equations. *Geodetic applications of GPS*, pages 10–54, 1997.

[5] SS Cross, RF Harrison, and RL Kennedy. Introduction to neural networks. *The Lancet*, 8982(346):1075–1079, 1995.

[6] Android Developers. Android api guide: Environment sensors. `http://developer.android.com/guide/topics/sensors/sensors_environment.html`. Consulted on May 2, 2016.

[7] Google Developers. The google maps elevation api. `https://developers.google.com/maps/documentation/elevation/intro`. Consulted on April 16, 2016.

[8] 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.

[9] Transport for London. Public transport journeys by type of transport. `http://data.london.gov.uk/dataset/public-transport-journeys-type-transport`. Consulted on May 11, 2016.

[10] Transport for London. Tfl: Facts & figures. https://tfl.gov.uk/corporate/about-tfl/what-we-do/london-underground/facts-and-figures. Consulted on April 11, 2016.

[11] Alexey Goncharov. Transport schemes. http://www.alexeygoncharov.com/index1-eng.html. Consulted on November 25, 2015.

[12] The Guardian. Navigating decline: what happened to tomtom? https://www.theguardian.com/business/2015/jul/21/navigating-decline-what-happened-to-tomtom-satnav. Consulted on May 11, 2016.

[13] Android Developer Guide. Android: Environment sensors. http://developer.android.com/guide/topics/sensors/sensors_environment.html. Consulted on October 20, 2015.

[14] Jeffrey Hightower and Gaetano Borriello. Location systems for ubiquitous computing. *Computer*, (8):57–66, 2001.

[15] Takamasa Higuchi, Hirozumi Yamaguchi, and Teruo Higashino. Tracking motion context of railway passengers by fusion of low-power sensors in mobile devices. In *Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 163–170. ACM, 2015.

[16] N. Kawaguchi, M. Yano, S. Ishida, T. Sasaki, Y. Iwasaki, K. Sugiki, and S. Matsubara. Underground positioning: Subway information system using wifi location technology. In *Mobile Data Management: Systems, Services and Middleware, 2009. MDM '09. Tenth International Conference on*, pages 371–372, May 2009.

[17] Marian Mohr, Christopher Edwards, and Ben McCarthy. A study of lbs accuracy in the uk and a novel approach to inferring the positioning technology employed. *Computer Communications*, 31(6):1148–1159, 2008.

[18] Germain Moyon. Business insider: Moscow metro's wi-fi revolution as city gets wired. http://www.businessinsider.com/afp-moscow-metros-wi-fi-revolution-as-city-gets-wired-2014-12?IR=T. Consulted on February 10, 2016.

[19] Song Pan, Li Fan, Jiaping Liu, Jingchao Xie, Yuying Sun, Na Cui, Lili Zhang, and Binyang Zheng. A review of the piston effect in subway stations. *Advances in Mechanical Engineering*, 5:950205, 2013.

[20] London & Partners. London tourism report 2014-2015. `http://files.londonandpartners.com/l-and-p/assets/media/tourismannualreview2014-15.pdf`. Consulted on May 6, 2016.

[21] Pressurenet. Pressurenet: The weather's future. `https://www.pressurenet.io/`. Consulted on November 2, 2015.

[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] Kartik Sankaran, Minhui Zhu, Xiang Fa Guo, Akkihebbal L Ananda, Mun Choon Chan, and Li-Shiuan Peh. Using mobile phone barometer for low-power transportation context detection. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 191–205. ACM, 2014.

[25] Bosch Sensortec. Restricted data sheet bmp180 digital pressure sensor. `https://ae-bst.resource.bosch.com/media/products/dokumente/bmp180/BST-BMP180-DS000-12~1.pdf`. Consulted on October 21, 2015.

[26] Portland State Aerospace Society. A quick derivation relating altitude to air pressure. `http://psas.pdx.edu/RocketScience/PressureAltitude_Derived.pdf`. Consulted on October 21, 2015.

[27] Henrik Stewenius, Christopher Engels, and David Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284–294, 2006.

[28] Thomas Stockx, Brent Hecht, and Johannes Schöning. Subwayps: towards smartphone positioning in underground public transportation systems. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 93–102. ACM, 2014.

[29] Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. Cooperative transit tracking using smart-phones. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 85–98. ACM, 2010.

[30] GLA Intelligence Unit. 2014 round population projections. `http://data.london.gov.uk/dataset/2014-round-population-projections`. Consulted on May 6, 2016.

[31] Evgeniy Varfolomeev. 3d-model of moscow metro. `http://varf.ru/metro3d/?en=1&p=-90&t=45&d=41.05255888325765`. Consulted on November 25, 2015.

[32] Takafumi Watanabe, Daisuke Kamisaka, Shigeki Muramatsu, and Hiroyuki Yokoyama. At which station am i?: Identifying subway stations using only a pressure sensor. In *Wearable Computers (ISWC), 2012 16th International Symposium on*, pages 110–111. IEEE, 2012.

[33] Wikipedia. List of metro systems — wikipedia, the free encyclopedia, 2016. [Online; accessed 8-June-2016].

[34] H.D. Young and R.A. Freedman. *Sears and Zemansky's University Physics*. Addison-Wesley series in physics. Addison-Wesley, 2000.

[35] Paul A Zandbergen. Accuracy of iphone locations: A comparison of assisted gps, wifi and cellular positioning. *Transactions in GIS*, 13(s1):5–25, 2009.