FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	GPS assisted implementation of way-back function on ultra low power MCU
Student:	Bc. Richard Stanko
Supervisor:	Ing. Jiří Hušák
Study Programme:	Informatics
Study Branch:	Design and Programming of Embedded Systems
Department:	Department of Digital Design
Validity:	Until the end of summer semester 2020/21

Instructions

For an existing embedded application, implement the following functionality. Automatic data transfer between a connected smartphone and a GPS module, via the main target processor(ARM-based low power MCU). These are assistance data and are sent via BLE. Further, develop and tune an algorithm to ensure an "emergency way back" function. The functionality is as follows:
The phone periodically logs the GPS position and transfers it to the application via a BLE link. The application stores these positions in its internal memory (internal or external flash).
In the case of smartphone failure, the application should be able to guide the user back to the starting position, using the stored data.
In addition to the code, provide documentation, results of testing and discuss the limits of the proposed solution and possible future improvements.

References

Will be provided by the supervisor.

doc. Ing. Hana Kubátová, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague January 27, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

GPS assisted implementation of way-back function on ultra low power MCU

Bc. Richard Stanko

Department of Digital Design
Supervisor: Ing. Jiří Hušák

August 6, 2020

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. I further declare that I have concluded an agreement with the Czech Technical University in Prague, on the basis of which the Czech Technical University in Prague has waived its right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act. This fact shall not affect the provisions of Article 47b of the Act No. 111/1998 Coll., the Higher Education Act, as amended.

In Prague on August 6, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Stanko, Richard. *GPS assisted implementation of way-back function on ultra low power MCU*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

V tejto práci sa pojednáva o navigácii na globálnej úrovni, Bluetooth low energy a ich využitie v reálnej aplikácii, hodiniek, ktoré využívajú tieto technológie. V prvej kapitole sa zoznámime s technológiami využívaných v navigácii, Bluetooth-e, a komponentami hodiniek, ktoré pri práci využijeme ako je už spomínaný navigačný modul, Bluetooth, externá pamäť a zbernice určené na komunikáciu s týmito modulmi. Ďalej s asistenčnými datami pre navigáciu, ako ich získať a použiť pre urýchlenie získania presnej polohy z navigačného modulu a ako ich budeme ukladať do externej flash pamäte hodiniek. S navigáciou súvisí aj funkcia návratu domov, ktorá má za úlohu naviesť užívateľa späť po predom získanej trase, nahranej z pripojenej mobilnej aplikácie alebo zaznamenanú hodinkami. Aby bolo možne s hodinkami komunikovať a posielať do nich asistenčné data a trasu pre návrat domov, je nutná mobilná aplikácia, ktorá bude tieto služby poskytovať. Kedže je táto práca zameraná hlavne na embedded aplikáciu a software, je mobilná aplikácia popísaná len v stručnosti. V záverečnej časti práce sú ukázané reálne testy už spomínanej funkcie návratu domov a meranie spotreby, ako na hodinkách tak na vývojovej doske, aby sa ukázalo ako tieto asistenčné data môžu pomôcť redukovať spotrebu.

Klíčová slova GNSS, Navigácia, Asistovaná navigácia, Asistenčné data, Connected aplikácia, Take me home

Abstract

In this thesis we discuss the navigation on a global scale, Bluetooth low energy and the usage of these technologies in a real application, a wrist-worn watch. In the first chapter we will give an introduction to the techologies used in navigation, Bluetooth, the components of the watch which we will use througout this thesis such as the GNSS module, Bluetooth, external memory and the communication buses. We mention the history of the technologies, some basic concepts and analyse the main parts of the thesis which are the assistance data and the Take me home function. In the next chapter we will talk about the assistance data, how they will be acquired and used for a faster position acquisition and how they will be stored into the external flash memory of the watch. The Take me home function which is related to the navigation, will be used to navigate the user back home on a previously logged track either by a connected phone application or by the watch itself. To assure the communication and the sending of the data to the watch a phone application that will provide these services is needed. This thesis is aimed mainly at the embedded application and software so the mobile application will be discussed only briefly. In the final parts of the thesis we will show and discuss some real life tests and power consumption measurements on both the watch and a development board, to show how these assistance data can reduce the consumption.

Keywords GNSS, Navigation, Assisted navigation, Assistance data, Connected application, Take me home

Contents

List of Figures

Introduction

People nowadays depend heavily on their technology residing in their pockets without questioning it. They take its functionality for granted and need it working on demand. As an example we can look at the self-driving cars (autonomous vehicles), which needs throughout verification, testing and in case of failure, it must fail safely in order to ensure the well-being of its crew. These vehicles as well as other so called smart products are *connected*, meaning they are a part of some large network in which they share data. To provide such functionality the engineering team needs to design, implement and test the product to ensure it behaves as planned under all circumstances.

This thesis focuses on navigation and functions connected to it. Devices providing navigation services can be stand-alone or integrated. The first kind can be found in vehicles, usually attached to the windshield or car board. On the other hand the second kind, the integrated devices, can be found for example in the car board as a part of the infotainment system, in mobile or wearable devices etc. Only mobile and wearable devices will be further discussed as the final application is one of them. In these devices, communication between said integrated components is vital. Data needs to be sent and received to ensure correct and fast functionality. The other important, yet often overlooked component until it is depleted, is the battery. The one part which makes sure everything else gets to talk by providing power to it. To ensure the power is distributed only to the components that really need it some power management techniques are implemented. This ensures that when we are not using or do not need the navigation module, no power is distributed to it. Other ways of draining the battery in a "more friendly" manner is making the components *low energy* which decreases the power consumption. Probably the most well known is *Bluetooth Low Energy* or just *BLE*. It is important that the technology providing the means of communication is low energy, especially when the product is meant to be connected and depends on this communication. We want the communication to be low energy because if the application is often transmitting and receiving data, it would be unacceptable to have the

battery drained after a couple of exchanged messages. In fact every device or component which is engaging in some type of communication should be low energy or at least have a power saving mode, this includes the navigation module as well which is exchanging data with individual satellites in orbit.

The target application is a connected product - a watch, in the thesis referred to as "the watch", developed by the company ASICentrum, referred to as "the company" from here on. This thesis is was made in collaboration with this company. The hardware on which all of the software, that was developed for the purposes of this thesis is run, was developed and lent to us by ASICentrum. The base software, such as the operating system and the interfaces for other modules, some of which we will be using (e.g compass), was also developed and provided by the engineers from the company. All the other software used for the purposes of GNSS navigation, assistance data and the phone application was developed by us.

One of the assets of the function that will be discussed throughout this thesis, the *Take me home* feature, is that it provides another means of navigation at the convenience of your hands. While still remaining portable, mobile phones are powerhouses if we look at them from the perspective of computational power but their battery can be quickly drained by the plethora of features they offer. A nearly depleted battery can cause a degradation in function or in some cases even the disabling of some functions, which the user may need. This is were our wrist-worn watch comes in handy, having stored the locations of a previously walked track, it can navigate the user back to his/her parking spot, home or any other starting point. By delegating this task to the watch, the user conserves some battery on the mobile phone, so it can be used later.

To summarize, we will discuss the technology behind satellite navigation also known as GNSS and Bluetooth Low Energy. In the second part of the thesis, we will use these technologies and a couple of others to ensure a fast and low energy navigation and tracking process. We will show how these technologies can cooperate together and with a connected mobile phone, which will provide some assistance data downloaded from the internet. In the final part we will design and develop a *Take me Home* algorithm, which when the connection to a phone is lost or the user triggers it, navigates the user back to a desired location.

Chapter 1

Goals of the thesis

The thesis has several goals as was mentioned before, these goals will be discussed further in this chapter.

We can split these goals into a main one and a couple of side ones. The main goal is to design and develop a so called *Take me Home* function, which will use the GNSS module along with a couple of other modules of the watch to provide a functionality as follows: Once this function is enabled on the watch, the application will periodically store the position of the user to the flash memory of the watch, received from the mobile phone. Upon activation or loss of connection between the phone and the watch, this function will look at the stored positions and navigate the user back to the starting point. The goal here is to develop a reliable function that will provide the user with a way back to his/her starting position under any operational circumstances, meaning there is enough battery to supply the GNSS module and the other parts of the watch, the GNSS module is operational, etc.

One of the other goals is to implement a function to store and use the assistance data provided for the GNSS module. This assistance data are a helpful complement to the GNSS module and they need to be downloaded from the internet. The data provide almanac and ephemeris information about the satellites, this topic is further explained in the next chapter. The goal here is to provide a phone side application which downloads and processes the data and its counterpart in the watch which receives and stores this data as well as supplies it to the module. The key part is the watch side since it passes the data to the navigation module itself and is of embedded nature. The amount of data supplied and its frequency need to be analyzed and the best solution has to be found as a trade-off between power consumption and memory usage.

Finally we need to provide thorough and clear documentation, tests which cover standard use cases of the product and their results and since the power consumption is mentioned throughout this thesis, provide power consumption measurements and the change in the consumption with and without the use of the assistance data.

Chapter 2

Analysis and design

2.1 GNSS

This section will provide information about the GNSS, which is the main object of this thesis. We will talk about what it is, how does it operate and a little bit of history and future. Assisted navigation will also be mentioned, since it is what the assistance data part is all about.

GNSS stands for Global Navigation Satellite System, which is referring to a satellite navigation with global coverage. Satellite constellations providing global coverage include the USA's NAVSTAR Global Positioning System (GPS), China's BeiDou, Russia's Global'naya Navigatsionnaya Sputnikovaya Sistema (GLONASS) and Europe's Galileo.

2.1.1 Brief history

In this part we will very briefly describe the history of the four global satellite constellations, GPS, GLONASS, BeiDou and Galileo.

2.1.1.1 GPS

The NAVSTAR Global Positioning System was developed by the Department of Defense of the government of the USA. It was originally launched in 1973, to improve on previous navigation systems, integrating some ideas from predecessors. At the start it used 24 satellites and was intended for military use. It became fully operational in 1994, with civilian use being allowed since the 1980s. Since then it has received many improvements and in 2019 there were 33 operational and healthy satellites in orbit [7].

2.1.1.2 GLONASS

Development of the GLONASS constellation began in 1976 and similarly to the GPS it was fully operational in 1995. The constellation achieved full coverage of the Russian territory in 2010 and global coverage in 2011. The system was officially completed in 2015 and as of March 2020 it consists of 24 fully operational satellites in orbit[7]. It is worth mentioning that GLONASS lacked behind GPS in promoting commercial use. The first commercial device was released in 2007 being bigger and more expensive than its GPS counterpart. Since then it is being actively promoted for commercial use.

2.1.1.3 BeiDou

BeiDou is the only system consisting of several independent constellations. BeiDou-1 was first launched in 2000 and decommissioned in 2012. Since then Beidou-2 is offering services in the Asia-Pacific region and Beidou-3 is offering world-wide coverage since 2018. As of 2020 the system has a total of 35

operational satellites in orbit. Just like the previous two systems it is offering military and commercial services.

2.1.1.4 Galileo

The beginning stage of the Galileo project was agreed upon in 2003 by the European Union and the European Space Agency. Contrary to the other systems described above it is intended primarily for civilian use. It reached global coverage in 2016 and there are currently 26 satellites in orbit, from which 22 are usable, 2 are spares and 2 are retired.

2.1.2 Navigation

In this section we will define some common terminology widely used when talking about navigation and navigation systems.

According to [8], navigation is "any of the several available methods of determining or planning a ship's or aircraft's position and course by geometry, astronomy, radio signals, etc." This definition consists of two key concepts. Sometime referred to as the *science of navigation* and the *art of navigation*. The first concept deals with determining the position and velocity of a moving body with a respect to a previously known reference. While the second concept is the planning and maintenance of a course from one location to another, avoiding collisions and obstacles.

A *navigation technique* is the determination of a position and velocity by either manual or automatic means. A *navigation system* is a system that does the navigation technique but does so only automatically. These systems are often contained on the navigated vehicle itself, such as automobiles, ships or aircraft, but they can be stand-alone and may require additional components and infrastructure.

The output of such systems or techniques is called a *navigation solution*. This solution usually contains, in addition to the position, a UTC time, altitude, ground speed, heading of motion, accuracies of the previous data and much more, depending on the specific system or technique. These solutions represent the coordinate frame with a respect to a reference frame, the common reference frame being the Earth.

There are several techniques used in navigation and they are [7]:

- **Pilotage** - this technique relies on recognizing ones surroundings. In plain words "you know where you are and know where you want to go". This is one of the oldest techniques.

- **Celestial navigation** - As the name suggest this technique uses angles between celestial objects (the Sun, Moon, planets, stars) and some local objects to determine orientation and location on the surface of the Earth.

As the Earth and the other celestial bodies move some time estimation is required.

- **Dead reckoning** - For this estimation we need to know a starting position and some form of heading of motion, speed and elapsed time. Heading can be obtained by some kind of compass measurement. Then we can provide an estimation for the distance traveled. It is generally implemented by line plotting, which connect successive locations.
- **Radio navigation** - relies on radio frequency sources with known locations, receiving and transmitting technologies, signal structure and availability. GNSS relies on computer technology and highly precise and accurate timing, due to high speed of electromagnetic propagation.
- **Inertial navigation** - Could be described as an automated form of dead reckoning. Where the measurements of heading, speed, etc. are done by other sensors such as gyroscopes and accelerometers.

As we can observe, the first three techniques described above can be dated way back to the times of ancient mathematicians of Greece, the later ones are relatively new, dating back to the 20th century.

So taking into consideration the information above, satellite navigation can be described as a combination of both radio and inertial navigation, with so called Kalman filtering playing a major role in integration.

2.1.3 Coordinate frames

We can look at navigation as a multiple coordinate frame problem. GPS measures the velocity and position in reference to some satellite constellation, but the user would like to have this velocity and position in reference to the Earth. There are two main coordinate frames used in navigation and those are the *Earth-Centered Inertial - ECI* and *Earth-Centered Earth-Fixed - ECEF*. The axes used in both frames are perpendicular. The ECI frame has its origin in the center of mass of the Earth, the x and y axes lie on the equatorial plane, the z-axis is equivalent to the Earth's center of rotation but the axes do not rotate, hence it does not provide and unique navigation frame, therefore it is necessary to specify a time frame at which ECI overlaps with ECEF. The ECEF frame, sometimes referred to as Earth-fixed rotational, has its x-axis intersecting the earth at 0° latitude and 0° longitude. This means that the frame rotates with the Earth [8].

A *Local navigation frame*, has an origin in the point a navigation solution is needed for. Differently to the conventional naming of x,y,z axes it has a Down (D) axis corresponding to the z-axis and pointing towards the mass of the Earth, the North (N) axis is the projection in the plane orthogonal to the z-axis of the line from the user to the North pole[7]. The last axis is the East (E) and corresponds to the y-axis.

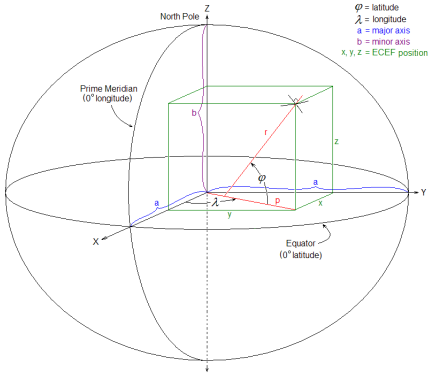

Figure 2.1: ECEF coordinate frame. Source: [1]

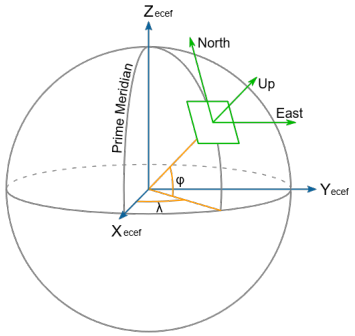

Figure 2.2: Local coordinate frame. Source: [2]

2.1.4 Kalman filter

Named after the Hungarian-American electrical engineer, mathematician Rudolf E. Kalmán who also was the main developer behind the idea. The filter also known as the *linear quadratic estimation*, is an algorithm that uses a series of measurements made over time, which can contain inaccuracies and produces estimates of unknown variables that tend to be more accurate than a single measurement, by estimating a joint probability distribution over the variables for each time frame. It has numerous applications, usually in guidance, navigation and control of vehicles [8, 9].

2.1.5 Satellite navigation

Since satellite navigation is the main force making every topic discussed in the further parts of this thesis possible, we will discuss some key concepts about GNSS navigation principles. The architecture of GNSS navigation systems consists of three parts: control, space and user segments.

The space segment consists of several satellites also known as a *constel-*

lation. In some literature we can come across the term space vehicle (SV) used instead of satellite. These satellite send signals to both the control and user segments. The broadcast signals include both ranging codes and data messages. Ranging codes allow the user equipment to determine the time the received signal was transmitted. Data messages include information about satellite orbits and timing.

The control segment is responsible for correcting any deviation in the space segment. That means some minor orbit or angle corrections. The observation of these attributes is done by several monitoring stations, which have synchronized timing and sending the adjusted data to the satellite by the uplink station.

User segment equipment is usually referred to as receivers. The GNSS segment is often a part of a larger application, with an antenna infrastructure that receives the data messages which are then demodulated and passed on to the navigation processor that computes a position, velocity, time (PVT) solution. To acquire an user's position at least 3 satellites are necessary because we can describe the distance from the user to the satellite by a sphere. The range of the receiver from the satellite can be determined by equation $\rho_j = (t_{sa,j} - t_{st,j}) * c$, where j denotes the number of the satellite or channel and t_{sa} and t_{st} are the times the signal arrives and is transmitted. The receiver can be anywhere on the surface of a sphere with the radius of ρ [8]. Adding a second satellite, therefore achieving an intersection of these spheres gives us a circle, which the user is located on. A third satellite further improves upon this, reducing the total amount of points to two. Sometimes this may be enough because one of the points may lie inside the Earth, in outer space, etc. To resolve this ambiguity a fourth satellite is added. The pictures in 2.3 depicts this.

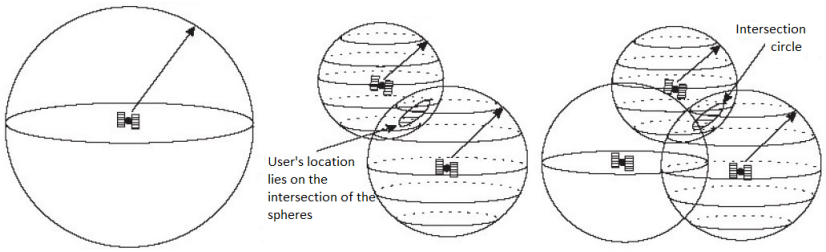

Figure 2.3: The location of the user being more precise as we add more satellites. With 2 satellites the location is on the intersecting circle, with 3 it may be one of two points. Source [3].

2.1.6 Assisted navigation

Before any measurements can be made satellite signals must be acquired. And before getting those signals the GNSS receiver must find the correct frequencies for the satellites and their correct code delay. If a receiver has no prior knowledge of these delays and frequencies it is performing a *cold start*. Since the ephemeris and time-of-week data is transmitted every 30 seconds and it takes 20 seconds to search for frequencies, we can say that a cold start takes approximately 1 minute at the minimum, that is if there were no errors in the transmission. A *warm start* indicates that the receiver has some knowledge of the user's position, time of day, rough idea of the reference frequency, and it will be able to calculate the approximate satellite positions from the almanac. Additionally if the receiver has decoded the ephemeris data for all visible satellites and computed a solution and stored this information in its memory, it would be performing a *hot start*[7].

In poor signal environment, it may be difficult to acquire a GNSS signal and track it and as a result navigation data message could not be demodulated. Therefore the receiver can work with out-of-date ephemeris, satellite clock and calibration parameters, degrading the navigation solution [8].

This can be solved by a so called assisted GNSS system, which provides the missing parameters via a side communication channel. In our case this is the data provided by the phone, which were downloaded beforehand from a dedicated server. This data contains ephemeris information, almanac, and other useful parameters designed to speed up the navigation solution calculation.

We use this process to speed up the time to first fix (TTFF), which reduces the acquisition time of the user's position, therefore reducing the power consumption.

2.2 Bluetooth Low Energy

As the title suggests this section will deal with Bluetooth and the Low energy part of it. We will describe the protocol itself, some history and development and mention how it is used throughout the application.

Bluetooth is a wireless technology used to exchange data between fixed and mobile devices over short distances and building personal area networks. It is managed by the Bluetooth Special Interest Group, which has over 35,000 member companies from telecommunications, networking and computing. The IEEE standardized Bluetooth as *IEEE 802.15.1*, but does not maintain the standard.

Over the years the uses of Bluetooth ranged from exchanging files between PC and mobile devices, listening to music, wireless calls using headsets and car kits. Today the attach rate is almost 100% for mobile phones, tablets and laptops.

Bluetooth Low energy as the title suggests is aimed at low power devices, as it is very useful for applications where it is impossible or inconvenient to recharge often and battery life is of utmost importance. Data communication occurs in short data bursts therefore it is best suited for devices that do not require high data throughput. Nowadays it finds a variety of uses including the popular and growing Internet of Things, healthcare devices, audio devices, smart watches and wearables etc.

Some key features of BLE include:

- Ultra low power
- Small size
- Interoperable
- Low cost
- Fast connection

2.2.1 Brief history

The name comes from the tenth century Danish king named Harald Bluetooth, whose epithet was *Blåtan*. And such as this king united the Danish tribes, Bluetooth was to unite the communication protocols. The logo is a bind rune merging two runes, representing his initials.

The development was initiated in 1989 by Nils Rydbeck, CTO at Ericsson Mobile in Sweden. The purpose was to develop wireless headsets. From 1997 Örjan Johansson took over the project and led the development and standardization. In 1997 IBM approached Ericsson Mobile for collaboration on integrating a mobile phone into a laptop. But since the power consumption of the mobile was too high to allow viable integration, they agreed to integrate Ericsson's wireless technology. They made the it and open industry standard and with Intel, Toshiba and Nokia joining, in 1998 the Bluetooth SIG was launched. The first device with this technology was launched a year later, it was a hands-free mobile headset.

In 2001 Nokia started developing a wireless technology adapted from the Bluetooth standard which would provide low energy consumption and cost. The results were published in 2004 under the name Bluetooth Low End Extension. After further development and support from companies such as STMicroelectronics and Logitech, it was released in 2016 under the name Wibree and in 2007 it was agreed to include it into Bluetooth as an ultra low standard. It was integrated into version 4.0 under the name Bluetooth Low Energy. Version 4.0 of the Core Specification was released in 2010, and then in 2016 version 5.0 was unveiled. ;

2.2.2 Bluetooth architecture

The architecture of the Bluetooth is layered. On the bottom we have the *controller layers* responsible for low-level operations such as discovering devices, making connections, exchanging data, security, etc. This functionality is implemented in a Bluetooth chip also called a Bluetooth Controller. The *upper layers* make use of the functionality provided by the lower layers in order to provide a more complex functionality like music streaming, splitting, sending and reassembling large data chunks, etc. Next we have *profiles* which can be viewed as vertical slices through the layers. They describe how the layers function and communicate together in order to implement a specific use case. Profiles guarantee us that an implementation from one vendor works well with another vendor's implementation. As we can see they form a base of interoperability. One device can of course support one or several profiles. A *Bluetooth application* can also be considered a part of the architecture because it provides and interface, usually and MMI(Man Machine Interface), so that the user can make use of the Bluetooth device. An example of this is selecting a file to transfer or search and connect to a device.

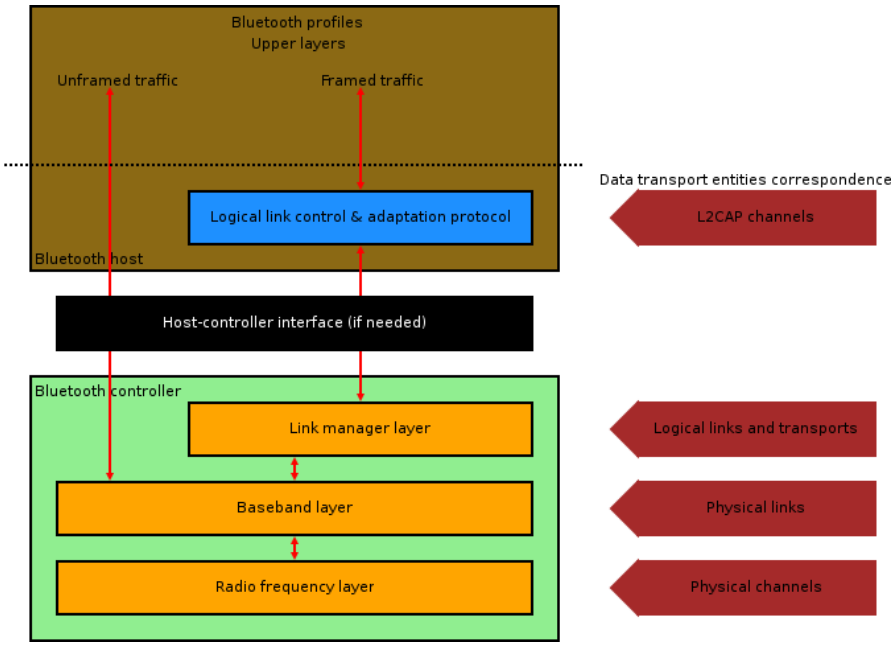

Figure 2.4: Bluetooth protocol stack
Image downloaded from:
https://commons.wikimedia.org/wiki/File:Bluetooth_protocol_stack.png

We can split the protocols used in Bluetooth into to categories. *Core protocols* are defined from scratch and made by the Bluetooth SIG. Some of these protocols include L2CAP, SDP and Link manager. The other group of

protocols are the *adopted protocols*, which are adopted from other standards. In this group we can find protocols such as RFCOMM, OBEX, HID profile, etc. Depending on the device the Bluetooth functionality is implemented on, we can distinguish between Host and Host Controller. The Host is a logical entity that implements the upper layers as seen in 2.4, also implements the application and profiles. On the other hand the Host Controller executes the lower layers of the protocol stack and it is typically embedded in a Bluetooth chip which is attached to the Host. If the two part are not directly interacting with each other a third entity, a Host Controller Interface is added, that ensures communication between the two main entities. Physically this can be a USB, UART, serial link, etc.

As of Bluetooth 3.0 + HS we recognize to types of controllers: Primary and Secondary. A system can have only one Primary controller, which supports either BR/EDR(Basic rate/Enhanced data rate) or LE(Low Energy), or a combination of the prior. The number of Secondary controllers a system can posses is zero or more. It supports one or more alternative MAC/PHY controllers(AMP). These AMP controllers help in increasing the throughput up to 24 Mbps by using the 802.11 transport layer instead of the Bluetooth transport layer.

Each controller is assigned a globally unique 48-bit Bluetooth Device Address, which is used to identify the device, it is similar to a MAC address and it is provided by the same organization, the IEEE. The device can be assigned what is called a device name. It can be changed by the application or the user and provides an easy identification.

Another useful numerical value is the CoD (Class of Device) which is a 24-bit number indicating the capabilities of the device to its vicinity. The first 11 bits determine the service class, the next 5 bits the Major class, 6 bits after that the Minor class and the final 2 bits are the Format type field [10].

2.2.3 Connecting

When establishing connection between two devices, they may follow the following steps. First one of the devices need to be in a state where it can be seen by other device, it has to become *discoverable*. The other device then searches for devices nearby, this is called an *inquiry*. In order to connect to the other device, it has to become connectable, then a connection is created between the devices. After the connection is established one device becomes the *master* and the other the *slave* and they are able to receive and send packets. These devices create what is called a *piconet*, which is the smallest unit of Bluetooth communication. When the connection is not needed anymore, the devices disconnect.

2.3 Hardware

In this section we will briefly describe the hardware used throughout this thesis. Only the hardware that is related to the thesis will be discussed such as the main processor, the GNSS module and the flash memory.

2.3.1 The processor

For the main processing unit we are using the nRF52840 system on chip by the Nordic Semiconductor company. This SoC provides a 64 MHz Cortex-M4 ARM CPU with a floating point unit, 1 MB of flash memory and 256 KB of RAM. Has a 2.4 GHz transceiver and supports Bluetooth 5, Bluetooth Low energy which can run concurrently with Zigbee or Thread. It also has some of the most used buses such as UART, SPI, QSPI and I2C. As was stated before it is multiprotocol capable with full protocol concurrency. It has an exceptionally low power consumption which is achieved using a sophisticated on-chip adaptive power management [11].

2.3.2 The GNSS module

The GNSS module is of the ZOE-M8 series manufactured by the U-Blox company, which is a highly integrated SiP(System in package) based on the high performing M8 concurrent positioning engine. Some of the advantages include its really small size of 4.5 x 4.5 x 1 mm, and up to 3 different GNSS constellations. It can communicate via UART, SPI or DDC(which is I2C compliant). It comes in two voltage supply variants, 1.8V and 3V and has a low power consumption. The module provides an SQI interface for an optional external flash and has several defense mechanisms against message spoofing and jamming [12].

2.3.3 The external flash

While more common than the previously mentioned components, we will describe the external flash since the one provided by the nRF52840 is used in its entirety by the application as well as the RAM. We will use the external flash to store assistance data meant for the GNSS module, which will help in faster fix times and faster position acquisition time means lower power consumption. The MX25R32 is a 32Mb serial flash memory, which is configured as 4,194,304 x 8 internally. It features a serial peripheral interface and a software protocol allowing operation on a 3-wire bus. Programming commands are executed on a byte or page(256 bytes) basis and the erase command is executed on a sector(4 Kilobytes) basis [13].

2.3.4 Component connection

Unfortunately we cannot provide schematics of the wiring or placement of these components, but it can be stated that the main MCU is connected to the external flash by an SPI bus and to the GNSS module by UART. The GNSS module is not connected to any other memory components, and therefore relies entirely on the application to supply data to it via the UART. Block schema depicting the connection of the components is in 2.5.

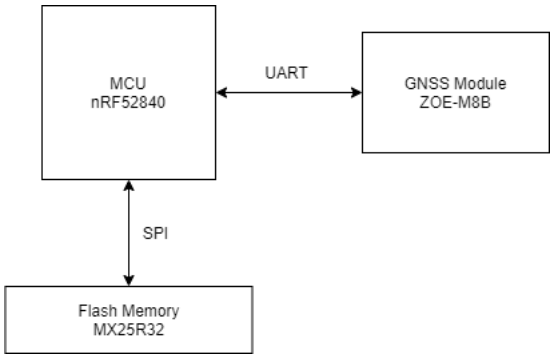

Figure 2.5: Block schema showing the connection of the main components used in the thesis. Source: own work.

2.4 Storing the assistance data

One of the tasks of this thesis is to implement storing and using assistance data for the GNSS module. We can split this tasks into several sub-tasks such as acquiring the assistance data from the internet, processing and formatting the data on the phone side of the application, sending and receiving the data on the watch side of the application and finally storing the data in the external flash memory and using the data. Before we dwell deeper into these topics we will discuss what exactly is this assistance data and how does it help.

2.4.1 The assistance data

This assistance data is a set of precalculated data consisting of ephemeris, almanac, time and satellite status. It helps reducing the required time to calculate a position even under poor signal conditions.

The data comes in several variants, as supported by the manufacturer at the time of writing this thesis. These variants are an online version and an offline version. Since the online variant requires a internet connection, our watch will be provided with offline assistance data.

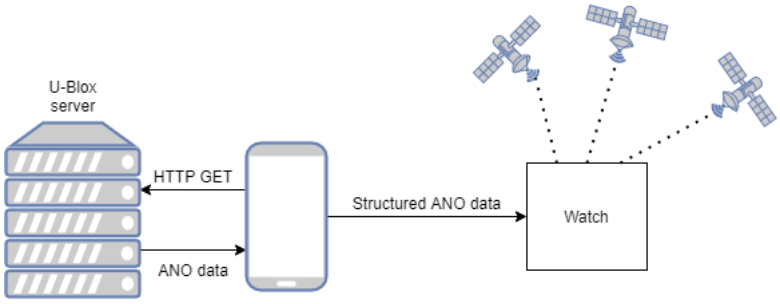

Figure 2.6: Block schema depicting the acquisition of the assistance data. Source: own work.

We will acquire this data from a remote server, specifying which satellite constellations are required, whether to include the almanac or the ephemeris and the period of the data in days.

Depending on the above options we are provided with several kilobytes of structured messages, which can be transferred one by one to the GNSS module.

2.4.2 Downloading the data

Since our watch is not connected to the internet, we will need to download this data by the connected mobile application and then transfer it to the watch by Bluetooth. For this to work the mobile device needs to be connected to the internet and allow the application to use Bluetooth and GPS. The GPS is needed to acquire an initial position, which is supplied alongside the assistance data to provide an even better estimation. The mobile application will download the data, store it for future processing, thus requiring approximately up to additional 200 Kilobytes of free memory space.

2.4.3 Formatting, sending & storing

After the data have been downloaded, we proceed with some processing. The processing consists of counting the almanac messages and the assistance data messages, providing the starting date, number of days the assistance data are valid for and the initial position of the user. This additional information will help us later on with navigating through the data and finding the correct messages for a specific day.

Sending the data involves segmenting it into smaller parts which can be sent via Bluetooth and then stored into the external flash memory of the watch. The whole procedure involves from 250 to 260 chunks, one chunk being 128 bytes long, and takes about 30 to 40 seconds to complete.

2.5 Take me home algorithm design

The main idea behind the *Take me home* algorithm is to navigate the user back using a previously walked track.

Some of the challenges we will be facing include:

- How often do we store a position
- When do we consider a position to be reached
- Using some features of the watch to help navigate
- Possible elimination of loops and track shortening

2.5.1 Getting a track

To be able to navigate the user back to a starting location, we need a track first. We will be able to get a track by two means: the phone application and the watch itself. Using the mobile phone as a logging device, we conserve energy on the watch but we need the mobile phone itself, with the application installed. The application has some software requirements such as the proper version of the Android operating system and hardware requirements such as existence of a Bluetooth and GNSS module (which nowadays is usually not an issue anymore). If we decide to use the watch by itself, we completely eliminate the necessity of another device but the power consumption will increase significantly, since the GNSS module puts a strain on the battery.

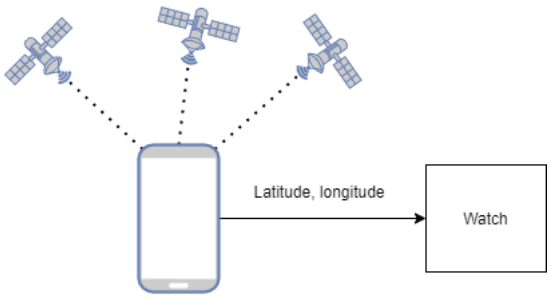

Figure 2.7: Block schema depicting logging of the position by the mobile phone. Source: own work.

The phone application, will periodically acquire a location and send it via Bluetooth and the watch will store it into its external flash memory for later use 2.7.

By using the watch itself, once we acquire a solution from the GNSS module, we extract and store the actual location into RAM 2.8.

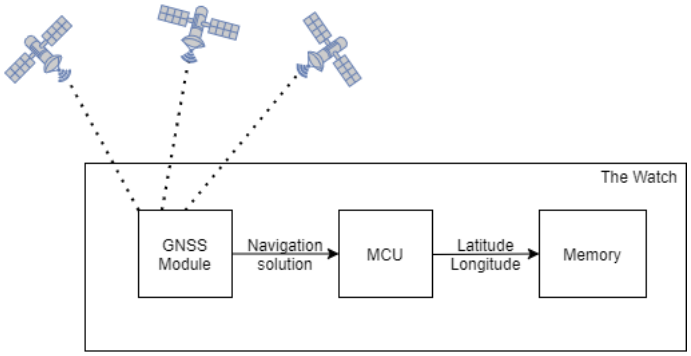

Figure 2.8: Block schema depicting the logging of the position by the watch itself. Source: own work.

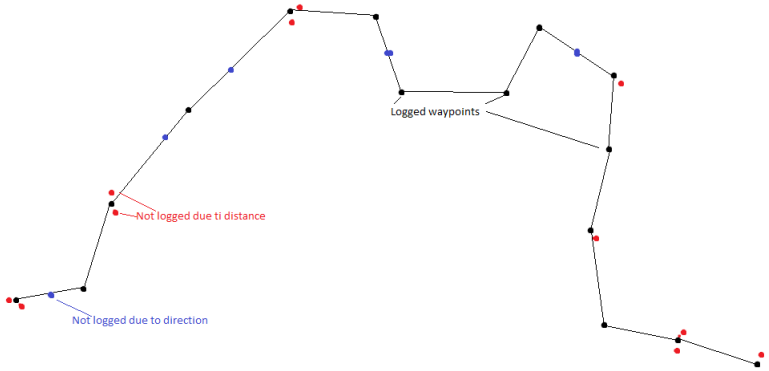

Figure 2.9: Reducing logging of points by filtering out those that are close or in the same direction. Source: own work.

In both cases we only store a location if it is more than 5 meters away from the previous location. This measure is in place to prevent unnecessary logging in case the user is standing still or moving very slowly. We are also keeping count of the total distance of the track and the number of logged locations, which will provide some information to the user.

2.5.2 Navigating

Once we have a track we can navigate backwards, from end to start, location by location. We always use the actual location provided by the GNSS module and use it to calculate a distance and direction to the next location of the

track. To use this calculated direction from one point to another we also need to know the real direction the user is currently heading. For this we can get the heading of motion from the navigation solution provided by the GNSS module or from a compass device. Only then can we correctly display the direction the user needs to go in order to reach the next point. A point is considered reached if we are in a 5 meter radius of that point. Picture 2.10 depicts this behavior.

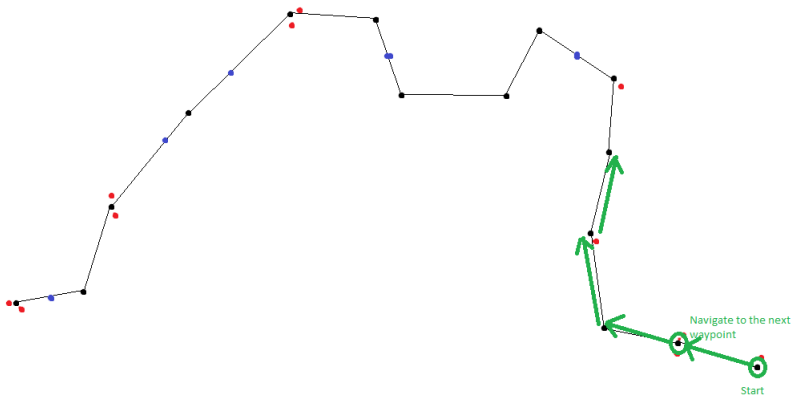

Figure 2.10: Navigating back point by point. We consider a point reached, while being within a 5 meter radius of it. Source: own work.

2.5.3 Shortcuts and loops

There is a high probability that the user can create some loops in the track. We should eliminate these loops or at least mention the possibility of doing so to the user, to shorten his way back. This could be done by finding the nearest location with the latest timestamp and navigating to this location instead. The decision whether to shorten the track or not should be done by the user, because we cannot know if the user created these loops deliberately, by going around some environmental obstacles (such as lakes, ravines, etc), or has done so unintentionally by wandering around.

On the upper track we can see an possible elimination of loops, shown with red color, which can be proposed to the user. While the bottom track has no loops, we can propose some shortcuts to the user, shown with blue color.

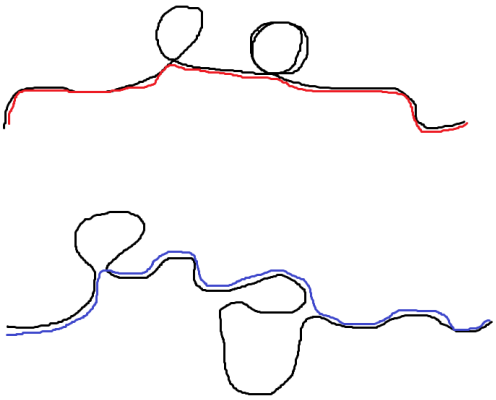

Figure 2.11: Example of track shortening

2.5.4 Weaknesses

The main weakness of the Take me home algorithm is the reception of the signal from the satellites. To navigate the user properly we need a good signal and a valid position fix. This can prove difficult in an environment where the signal can be reflected or blocked. So in an urban or metropolitan area, with tall buildings or other noise inducing elements, the receiver can be prone to multi-path errors. This can lead to either an invalid position fix, or incorrect position altogether, which then results wrong navigation. The assistance data can help with this, but the receiver still needs at least a range measurement from at least 3 satellites to calculate a navigation solution. A possible solution is to use a Kalman filter mention previously, but it requires additional components such as accelerometers, gyroscopes and matrix operations, thus it can be computationally demanding. Multi-path errors can lead to interference, either constructive or destructive, rendering the radio signal too weak to be received.

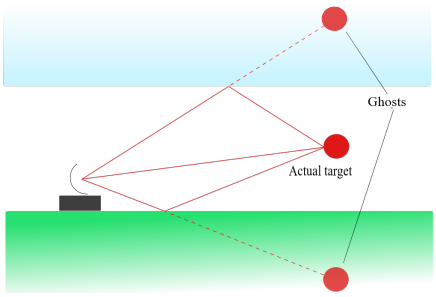

Figure 2.12: Multi-path error propagation. Source [4]

Chapter 3

Implementation

3.1 Assistance data

The assistance data has to be downloaded, given structure and then sent to the watch and processed. We will describe each of this step separately from the perspective of the device these steps happen on. The application that downloads and provides the assistance data was developed from scratch. We chose the Android platform because the author had some previous, albeit little, experience with android development. It was developed and tested using the Android studio IDE and is written in the Java language. The source code of this application is available in the attachment of this thesis.

The software for the watch is written in the C programming language. For development and testing purposes we used a J-Link debugger from Segger, which allows remote debugging and programming. A Real-time operating system (RTOS) is running on the watch, so every time we mention a *task* in the upcoming sections of the thesis, we mean a RTOS task, which can be scheduled and executed.

3.1.1 Download & Structure

The download of the assistance data happens only on the phone side. Here the mobile application connects to the provided website via a HTTP GET request and the data is downloaded as raw. Meaning that, it is already in a form where it only needs to be passed to the GNSS module. The size of this data depends on the parameters we specified in the GET request. We can specify the desired GNSS constellations, number of days, resolution, inclusion of almanac, ephemeris, etc. The most important is the unique token, which grants access to these services.

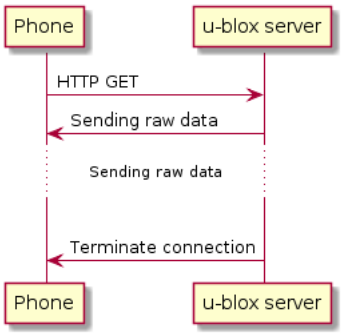

Figure 3.1: Assistance data download via HTTP GET

We receive the data in binary form, as u-blox messages, in our case the almanac followed by the assistance data itself. The data is given structure to help the watch parse and process this data and pass it to the GNSS module via UART. What we attach to this data is the type of the constellation we

are using, number of days the data are valid for, starting date and the size of the almanac and assistance parts.

CON	DATE	DAYS	#ALM	#ANO	LAT	LON	ALM	ANO
0	1	4	5	7	9	13	...	...

Figure 3.2: Structured data

The upper row of 3.2 is the description of information it contains while the bottom row is its starting offset. Description of the columns is as follows:
CON - constellation type - 1 Byte
DATE - starting date of the assistance data - 3 bytes
DAYS - number of days the assistance data are valid for - 1 Byte
#ALM - number of the almanac messages - 2 Bytes
#ANO - number of the assistance data messages per one day - 2 Bytes
LAT - initial latitude - 4 bytes LON - initial longitude - 4 bytes ALM - the almanac itself - size varies
ANO - the assistance data itself - size varies

Listing 3.1: Download the data using androids network fragment

```
String urlString ="...";
networkFragment =
  NetworkFragment
    .getInstance(getSupportFragmentManager(), urlString);
networkFragment.startDownload();
```

This code handles the downloading of the assistance data, the url begin a link to the u-blox server with the proper parameters to download the assistance data and the almanac for GPS and GLONASS. It implemented using a network fragment. In android development a fragment represents a behavior or a portion of the user interface. We use this to implement asynchronous network operations. It also requires a *DownloadCallback* which calls back to the main activity upon finishing the download, progress update of the ongoing download, etc.

3.1.2 Sending & Processing

After the data has been given structure we proceed with sending it via Bluetooth. For this we are using the Nordic UART service, which is a service that makes the Bluetooth connection act as it were a UART interface. The Nordic UART Service is a simple GATT-based service with TX and RX characteristics. Data received from the peer is passed to the application, and the data received from the application of this service is sent to the peer as Handle Value Notifications.

The phone sends out chunks, each 128 bytes assistance data plus 3 bytes framing data long, so 131 bytes in total. After a chunk has been sent, the phone expects an acknowledgment from the watch, after this ACK message is received the phone sends out the next chunk. After all chunks have been sent, the phone sends out a final message, indicating the fact that the transmission has been finished.

All of the operation codes where selected by the author, so that they do not collide with other messages already in use by Bluetooth services. These operation codes may be changed later to comply with the coding standards of the company.

OPCODE	SIZE	DATA
0x8899	0-128	0xB5...

Figure 3.3: Assistance data chunk structure

The OPCODE for sending assistance data is, as stated in 3.3, always 0x8899 in little endian. The SIZE is usually 128 bytes, with the only the last chunk having different size. The DATA is the structured data as presented in 3.2.

OPCODE
0xD0D0

Figure 3.4: Finalizing message

The finalizing message consists only of the OPCODE and its value is always 0xD0D0 in little endian. This message is sent only when all the structured assitance data was transmitted and acknowledged.

OPCODE
0x1122

Figure 3.5: ACK message

Acknowledging message, similarly to the finalizing message consists only of the OPCODE column and its value is always 0x1122 in little endian.

After the reception of each chunk, it is written into the flash memory of the watch at the proper address. The added values of starting date, number of days and sizes of the almanac and assistance data are there to help navigate in the data. We use the current date, compare it to the starting date to determine whether the data are up-to-date, if not, the watch should request the transmission of new data. After that we just find the first message with the current date and send the appropriate amount of messages via UART to the GNSS module. The provided initial latitude and longitude is used to help with the estimation of the location.

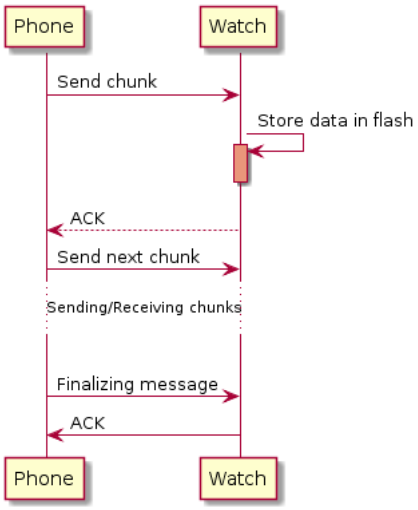

Figure 3.6: Phone - Watch assistance data exchange

Code snippets from both the phone application and the watch will follow, which implement the behavior described above.

Listing 3.2: Start of sending the assistance data

```
private static final int GPS_DATA_MSG1 = 0x99;
private static final int GPS_DATA_MSG2 = 0x88;
private static final int GNSS_CHUNK_SIZE = 128;
private static final int GNSS_DATA_OFFSET = 3;

output.write(GPS_DATA_MSG1);
output.write(GPS_DATA_MSG2);
output.write(GNSS_CHUNK_SIZE);
byte[] toSend = new byte[GNSS_CHUNK_SIZE + GNSS_DATA_OFFSET];
System.arraycopy(structuredData, bytes_sent, toSend, 0,
  ((bytes_length > GNSS_CHUNK_SIZE) ?
  GNSS_CHUNK_SIZE : bytes_length));
output.write(toSend);
toSend = output.toByteArray();
mService.writeRXCharacteristic(toSend);
bytes_length -= GNSS_CHUNK_SIZE;
bytes_sent += GNSS_CHUNK_SIZE;
```

The code above is stared after the user presses the SEND FILE button. It creates the first packet with the adequate header, follow by the length of the data chunk. The data is then written to RXCharacteristic which sends it via Bluetooth to the watch. The following code snippet is responsible for checking the acknowledgment messages and sending the next chunks of data. The `dataSent` variable is a boolean indicating whether all the data have been sent or not.

Listing 3.3: Receiving the ACK message and sending the next chunk of data

```
private void msgReceived(String msg){
  byte[] bts = msg.getBytes();
  if(bts[0] == GPS_DATA_ACK1 &&
    bts[1] == GPS_DATA_ACK2 &&
    !dataSent)
  {
    sendNextChunk();
  }
  if(dataSent)
  {
    mService.disconnect();
  }
}
```

We can see that each time a message is received and the acknowledgment data is correct the function `sendNextChunk()` is called which is identical to the first code snippet shown here. The only difference is in the size of the chunk, which is not the same for the last chunk and has to be adjusted as such.

On the watch side each chunk is received, put into a temporary buffer, stored into flash memory and then an acknowledgment is sent out 3.4.

Listing 3.4: Storing the assistance data

```
uint8_t chunkSize = p_evt->params.rx_data.p_data[2];
uint8_t helperBuff[GNSS_DATA_CHUNK_SIZE];

memcpy(helperBuff,
  p_evt->params.rx_data.p_data + 3, chunkSize);

assist_now_offline_write_data_to_flash(helperBuff,
  gnss_ano_data_offset, chunkSize);

gnss_ano_data_offset += chunkSize;

m_nus_tx_buf               = gnss_resp_buff;
m_nus_total_byte_left_tx = 2;
```

In 3.4 the `gnss_resp_buff` is equal to the message described in 3.5. The `gnss_ano_data_offset` variable is used as an offset to the flash memory to correctly write the data continuously. After that the event manager of the watch is used to trigger a Bluetooth event and send out the acknowledgment. When the finalizing message 3.4 arrives, the watch only responds with an acknowledgment without storing any more data.

3.1.3 Using the assistance data

After the data has been stored in the flash memory it is ready to be used. It is loaded into the GNSS module during the start-up of the GNSS task of the

watch. Since the module has no external non-volatile memory of its own, this data needs to me transmitted on every power-on.

First of all we check whether there is assistance data in the flash memory, and whether it is valid. By valid we mean that it is within a 7 day span of the starting date. We do this by simply comparing the dates, if the current date is greater than the starting date by 7 days, we consider the data invalid, and the user is prompted to provide new assistance data. This is done in 3.5, the `valid` variable is initialized as *false*.

Listing 3.5: Verifying the assistance data

```
if (total_days_current + num_of_days >= total_days_current
   && total_days_current >= total_days_start)
{
  valid = true;
}
// else either invalid data, or invalid date

if (valid)
{
  // data are valid, mark them as ready to read
  assist_now_offline_data_ready();
}
```

Next we load some variables that will help navigating to the correct data of the current day. We get the size of the almanac, number of messages per day, and the initial latitude and longitude. We load the almanac as well because it is necessary every time. Having the size of the messages we can calculate the offset into the flash memory where the data for the current day reside. $Offset = Header + almanac + (current\_date - start\_date) * messages\_in\_one\_day * size\_of\_message$. Where the header represents the data that were added by the phone, that includes the starting date, number of days the data are valid, number of almanac messages, number of messages per day of the assistance data, initial latitude and longitude. With this information we can easily calculate the size of the almanac, since every message is 44 bytes in size. Every assistance data message is 84 bytes in size, this is only valid for GPS and GLONASS constellations[5].

Listing 3.6: Finding the correct assistance data

```
uint8_t  numAlmMsg = assist_now_offline_data[ALM_MSG_OFF];
uint16_t almOffset   = ALM_OFF + numAlmMsg * ALM_MSG_LEN;

uint8_t  numAnoMsg = assist_now_offline_data[MSG_DAY_OFF];
uint32_t elapsedDays = currentDate - startDate;
uint16_t dayOffset = elapsedDays * numAnoMsg * ANO_MSG_LEN;

flash_read(GPS_AREA_ADDR_START + almOffset + dayOffset,
    buffer,
```

```
    sizeof(buffer));
p_data = buffer;
```

Code in 3.6 takes care of finding the correct offset into the flash memory and then reads the data into a buffer, which is a global variable of the assistance data module, and a pointer is returned from the function which is not shown in the snippet. For simplicity the `elapsedDays` variable in the example is calculated as difference of dates, this is different in the real code provided in attached code, where the total number of days are calculated and then subtracted from each other.

All the code listed above is called from the task responsible for handling the GNSS related operations. The assistance data is loaded during the initialization of the task alongside with the current time in UTC format and the provided latitude and longitude 3.7.

Listing 3.7: Loading the assistance data during start-up

```
assist_now_offline_read_data_from_flash();
gps_send_config_position();
uint8_t* p_data = assist_now_offline_get_data();
p_data += GNSS_DATA_ALM_OFF;
p_data =
  gnss_send_almanac(p_data,
    assist_now_offline_get_alm_msg_cnt());
p_data =
  gnss_send_offline_assitance(p_data,
    assist_now_offline_get_ano_msg_cnt());
```

As we can see in 3.7, first the data is read, provided that it is valid, from flash into the buffers of the assistance data module. The sending of the initial latitude and longitude into the GNSS module by UART is handled by the `gps_send_config_position()` function. The `UBX-MGA-INI-POS_LLH` message of the module is used for this purpose, with the latitude, longitude, altitude, and precision of the provided position as its arguments. Finally, we get a pointer to the buffer by calling the `assist_now_offline_get_data()` function, and as the messages are being sent into the GNSS module, we shift this pointer to the correct offset.

3.2 Take me Home algorithm

3.2.1 Position logging

One part of the *Take me Home* algorithm is the logging of the user's location into the watch. For this purpose we will use the connected mobile device and the GPS services it offers. The general idea is as follows: The user will turn on location logging. After 10 seconds or a change in direction greater than 5 degrees the location will be sent over to the watch and stored.

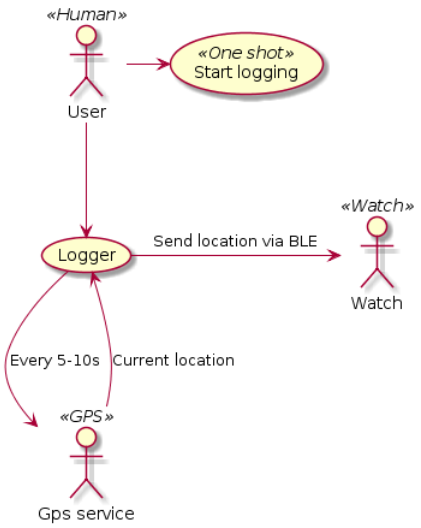

Figure 3.7: Use case of the logger

To further expand on 3.7, after the user turn on logging, the location updates are retrieved in an interval of 1-10 seconds depending on the service. The service which we will be using is the *FusedLocationProviderClient* by Google API. This allows us to get the location either via the GPS circuit of the phone or via network. Having retrieved the current position, we use it and the previously stored location to calculate the distance traveled and the bearing. For this we use the following equations.

$$a = \sin^2(\Delta\varphi/2) + \cos(\varphi_1) * \cos(\varphi_2) * \sin^2(\Delta\lambda/2) \tag{3.1}$$

$$c = 2 * atan2(\sqrt{a}, \sqrt{1-a}) \tag{3.2}$$

$$d = R * c \tag{3.3}$$

Figure 3.8: Haversine formula

To explain the symbols used in 3.8, φ_1 is the latitude of the starting location, φ_2 the latitude of the final location, $\Delta\varphi = \varphi_2 - \varphi_1$ and $\Delta\lambda = \lambda_2 - \lambda_1$ where λ_1, λ_2 are the longitudes of the locations using the same subscripts as their latitude counterparts. The values of latitudes and longitudes are in radians. The *atan2* function is the "two argument arctangent" which return a single value θ such that $-\pi < \theta \leq \pi$ and some $r > 0$ and $x = r\cos(\theta), y = r\sin(\theta)$. R is Earth's radius (mean radius = 6,371 km).

The haversine formula determines the great-circle distance between two point on a spherical surface.

If the programming language we are using does not have the *atan2* function we can calculate as follows: $c = 2\arcsin(min(1, \sqrt{a}))$. a is the square of half the chord length between the points and c is the angular distance in radians.

To calculate the bearing, or to be more precise the initial bearing (or forward azimuth), which if followed in a straight line will take you from the starting point to the ending point. The meaning of the symbols is same as in 3.8.

$$\theta = atan2(\sin(\Delta\lambda)\cos(\varphi_2), \cos(\varphi_1)\sin(\varphi_2) - \sin(\varphi_1)\cos(\varphi_2)\cos(\Delta\lambda)) \quad (3.4)$$

Figure 3.9: Initial bearing formula

By these calculations we determine whether there was a change in the general direction the user is heading. If so, the mobile phone will send a BLE packet to the watch with the current coordinates of the user. The watch will then store this information into its external flash memory.

OPCODE	LAT	LON
0x8899	4B	4B

Figure 3.10: Location packet structure

The BLE packet has a similar structure as the ones used in assistance data transfer, with the difference of the OPCODE and the payload. The latitude and longitude values are first converted into integers by multiplying them by 10^7 and then stored into a byte array in little endian form.

When received by the watch, it checks whether the distance between the new point, and the previous point is greater or equal than 5 meters. If it is so the new point is stored in flash memory, otherwise it is ignored.

OPCODE
0xBAAB

Figure 3.11: Location finalizing packet

After the turning of the logging in the phone application a finalizing packet is sent. This packet tells us that there will be no more location messages and that we can store the total number of locations and total distance into the flash memory.

In 3.12 we can see how the data is stored in the external flash memory. The #POS field is 2 bytes wide and stores the total number of location for the track, the DIST field is also 2 bytes wide and gives us information about the total distance of the track. After that #POS number of locations follow

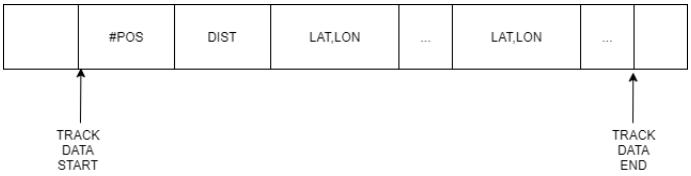

Figure 3.12: Track stored in flash memory

each containing 4 bytes of latitude and 4 bytes of longitude. Therefore we can derive the total number of required bytes for a track as follows: $Total\_bytes = \#POS * 8 + 4$, 8 bytes per location plus the 4 bytes of total location number and distance.

Logging by the watch is done as is described by 3.13. Whenever a new location is available from the GNSS module, the logger calculates the distance from this new location to the previous one, if it greater or equal to 5 meters the location is stored and the variables are updated.

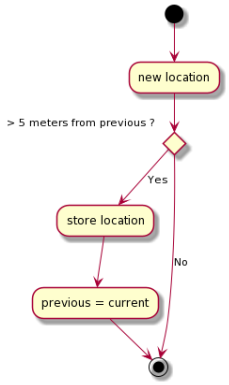

Figure 3.13: Logging by the watch

Whenever we receive an update from the running GPS service, which is timed to be every ten seconds, a broadcast is sent from the service to the main activity. This message is received in the following code sample.

Listing 3.8: Receiving the GPS service broadcast and sending a location to the watch

```
String data = intent.getStringExtra(GPS_MESSAGE);
String[] locationData = data.split(",");
Double lat = new Double(locationData[0]);
Double lon = new Double(locationData[1]);
Double dist = new Double(locationData[2]);
Double bearing = new Double(locationData[3]);
updateUI(lat, lon, dist, bearing);
sendLocation(lat, lon);
```

First we extract the data from the intent message. By extracting we mean parsing the latitude, longitude, bearing and distance. After that we pass these data to two functions. The `updateUI()` function simply displays the received data on the phone screen, while the `sendLocation()` converts the floating point values of latitude and longitude into integers and stores them into a byte array in little-endian form, then this array is sent via Bluetooth into the watch. The conversion is needed because Java stores the values in big-endian form. We only take the first seven digits of the latitude/longitude which corresponds to a precision down to meters. The sending procedure is identical to the one used in the assistance data sending. We use a Bluetooth service and write to the RX Characteristic.

Listing 3.9: Sending a location to the watch

```
        int index = 0;
byte[] msg = new byte[2 + 4 + 4];
msg[index++] = (byte) GPS_DATA_LOC1;
msg[index++] = (byte) GPS_DATA_LOC2;
int iLat = (int) (lat * 10000000);
int iLon = (int) (lon * 10000000);
msg[index++] = (byte) ((iLat) & 0xFF);
msg[index++] = (byte) ((iLat >> 8) & 0xFF);
msg[index++] = (byte) ((iLat >> 16) & 0xFF);
msg[index++] = (byte) ((iLat >> 24) & 0xFF);

msg[index++] = (byte) ((iLon) & 0xFF);
msg[index++] = (byte) ((iLon >> 8) & 0xFF);
msg[index++] = (byte) ((iLon >> 16) & 0xFF);
msg[index++] = (byte) ((iLon >> 24) & 0xFF);
mService.writeRXCharacteristic(msg);
```

The watch counterpart to this handles the reception of the position in the Bluetooth task and calls a function of the GNSS module to handle the storing of the data 3.10.

Listing 3.10: Storing a location into the watch - BLE

```
gnss_store_phone_location(p_evt->params.rx_data.p_data + 2,
                          false);
```

Then in the GNSS module, we call one of the two functions, determined by the boolean flag `final` of the `gnss_store_phone_location()` function. This flag is set when the final location has been transmitted indicating the fact, that the module should store the total number of locations and the total distance of the the track into the flash memory 3.11.

Listing 3.11: Storing a location into the watch - GNSS module

```
void gnss_store_phone_location(const uint8_t* buffer,
                               bool final)
{
```

```
  if (final)
  {
    take_me_home_store_position_count();
  }
  else
  {
    take_me_home_store_position((const int8_t*)buffer);
  }
}
```

As the names of the functions suggest, the storing of each individual location received is managed by the `take_me_home_store_position()` function, which takes a pointer argument where the received locations are temporarily stored. On the other hand the `take_me_home_store_position_count()` function, handles the storing of the total locations and total distance of the track.

We will take a look at the functions described above in the following code snippet.

Listing 3.12: Storing a location into the watch - Take me home module

```
error = flash_program(GPS_LOG_ADDR_START + LOCATION_OFFSET
    + log_offset * sizeof(int32_t),
    (uint8_t*)buffer,
    2 * sizeof(int32_t),
    false);

ERROR_CHECK(error);

log_offset += LOG_ELEMENT_SIZE;
total_locations++;

previous_location          = current_location;
current_location.latitude  = ((int32_t*)buffer)[0];
current_location.longitude = ((int32_t*)buffer)[1];

current_track.total_locations = total_locations;
if (total_locations > MIN_LOCATIONS)
{
  current_track.total_distance += distanceToNextPoint(
                  previous_location.latitude,
                  previous_location.longitude,
                  current_location.latitude,
                  current_location.longitude);
}
```

Beside the obvious storing of the position into the flash memory we also calculate the total distance of the track in this subroutine in 3.12. The writing to the memory is done by the `flash_program()` function, which takes several arguments: first is the address where we wish to write our data, next is a

pointer to the data, its size and a flag indicating if we want to delete the memory block first. After the writing has been completed without an error, we shift the offsets so the next position is written into the correct address. Finally we compute the distance from the last point to the current point by using the 3.8 equations, which are implemented in the `distanceToNextPoint()` function.

Listing 3.13: Storing the position count and total distance

```
uint32_t data = (total_locations << 16)
      | (current_track.total_distance & 0x0000ffff);

error = flash_program(
      GPS_LOG_ADDR_START + LOG_LOCATION_COUNT_OFFSET,
      (uint8_t*)&data,
      sizeof(data),
      false);

ERROR_CHECK(error);
```

The code in 3.13 is the `take_me_home_store_position_count()` function. Here we store the total number of locations as the upper 16 bits of a 32 bit integer, while the lower 16 bits represent the total distance of the track. Identically as in 3.12, we write into the flash memory by calling `flash_program()` and finally an error check is performed.

Since the watch has a GNSS module of its own, the user can choose to use a track which was logged by the watch itself. Here the position is logged on every GNSS event received from the module but only if its distance from the last point stored is greater than 5 meters.

Listing 3.14: Storing a track by the watch

```
log.latitude  = take_me_home_get_latitude();
log.longitude = take_me_home_get_longitude();

if ((prev_lat != 0) && (prev_lon != 0))
{
  float distance =
      take_me_home_calc_distance_for_watch_track(
      prev_lat, prev_lon,
      log.latitude, log.longitude);
}

if (distance >= TAKE_ME_HOME_REACHED_RADIUS_METERS)
{
  gnss_data_log.total_distance += distance;
  prev_lat = log.latitude;
  prev_lon = log.longitude;
  gnss_data_log_append((uint8_t*)&log,
     sizeof(log));
```

}

The example shown in 3.14 is straightforward: we get the latitude and longitude from the GNSS module, calculate the distance and if it is greater than the threshold (which is 5 meters in this case), we store this location into a buffer. The `log` structure also stores other information which are absent from the code example in 3.14, these include the state of the module, heading of motion and its accuracy and the UTC time. With this we have concluded the part of sending, receiving and storing the individual locations which will be used in the next section to navigate back.

3.2.2 Navigating back

We start the navigation process by retrieving the latest logged point from external flash or from RAM. Next we get the actual location of the user from the GNSS module and calculate the distance and direction from this point to the one from the log. The distance is displayed so the user is aware how far away is the next location. The direction is used in combination with the heading of motion retrieved from the GNSS module or the azimuth from the compass to correctly calculate an angle which is then displayed using the watch hands. This process of acquiring the current location and calculating distances, direction is repeated until we are within a 5 meter radius of the logged point. After that the point is considered reached and we move onto the next point in the log. We repeat everything described above until we reach the oldest(first) logged point. Pseudocode in 1 describes this.

```
Result: Navigating to the start of the track
initialization;
while Location log not empty do
    read next location from log;
    while distance > radius do
        read location from GNSS;
        calculate distance, direction;
        display distance;
        adjust direction;
    end
    mark location as reached;
end
```

Algorithm 1: Take me home main algorithm

Subroutines used in the above main algorithm will be described below. Initialization involves setting the pointers, counters to the last location.

Reading the location depends whether we read it from external flash or from RAM. Reading from RAM is straight forward, we just get the pointer pointing to the buffer containing the locations and move by 8 bytes every time we need a new location. Getting the data from flash is quite similar,

but it involves reading the actual flash memory, for which we need to know the exact address we want to read from. We know where does this data start, how many locations do we have so we can easily calculate the necessary offset. $Offset = start\_address + loc\_offset + \#loc\_cnt * 2 * sizeof(int32\_t)$. Where $\#loc\_cnt$ is the number of stored locations, which we read out prior to this procedure. We need to be careful because this points to the end of the log where the next location would be stored, so to get the actual last logged point, we need to subtract 8 bytes and now we have the address where the last latitude is stored. To further traverse this log we just subtract the number of already processed points. From this address we read out 8 bytes, first 4 are our latitude and the last 4 our longitude.

Getting the location from the GNSS module involves periodically polling a message. The reply to this message contains, among other things, the latitude, longitude, heading of motion, etc. This response is in fact a position, velocity, time (PVT) solution structure.

Byte Offset	Number Format	Scaling	Name	Unit	Description
24	I4	1e-7	lon	deg	Longitude
28	I4	1e-7	lat	deg	Latitude
32	I4	-	height	mm	Height above ellipsoid
36	I4	-	hMSL	mm	Height above mean sea level
40	U4	-	hAcc	mm	Horizontal accuracy estimate
44	U4	-	vAcc	mm	Vertical accuracy estimate
48	I4	-	velN	mm/s	NED north velocity
52	I4	-	velE	mm/s	NED east velocity
56	I4	-	velD	mm/s	NED down velocity
60	I4	-	gSpeed	mm/s	Ground Speed (2-D)
64	I4	1e-5	headMot	deg	Heading of motion (2-D)
68	U4	-	sAcc	mm/s	Speed accuracy estimate
72	U4	1e-5	headAcc	deg	Heading accuracy estimate (both motion and vehicle)

Figure 3.14: Screenshot of the UBX-NAV-PVT message. Taken from the U-Blox receiver description[5].

Referring to 3.14 we read out the latitude and longitude fields, and later in the algorithm the ground speed, heading of motion and heading accuracy estimate fields. After we have the current location we use 3.8 and 3.9 to determine the distance and direction. This message is sent from the GNSS module to the main CPU via UART every second. The period of this message can be configured or turned off altogether and requested manually.

Since the 1 algorithm represents the main routine, the code connected with it is quite large. Because of this we will only show important subroutines of the algorithm.

Let us begin with reading out the latest location from the flash memory. This is done in the following example.

Listing 3.15: Getting the next location from memory

```
uint8_t buffer[LOG_ELEMENT_SIZE * sizeof(int32_t)];

flash_read(GPS_LOG_ADDR_START
```

```
        + log_offset
        - LOG_ELEMENT_SIZE,
        buffer, sizeof(buffer));

log_offset -= LOG_ELEMENT_SIZE;
current_track.current_location_index++;

current_track.current_location.latitude
  = convertToFloat(((int32_t*)buffer)[0]);

current_track.current_location.longitude
  = convertToFloat(((int32_t*)buffer)[1]);

current_track.current_location.status = LOCATION_NOT_REACHED;
```

We start by reading out the correct location from the flash memory, the `log_offset` variable was set either during the reception and storing of the locations, or by reading out the total number of locations prior to calling this function. The latitude and longitude need to be converted to floating point numbers, because they are stored as integers and the algorithm calculating the distance and direction uses trigonometric functions which require floating point arguments. We mark this location as *not reached* and proceed navigating to this point.

Navigating to the next point is done in several steps. First we need to get our actual position, for this we use the GNSS module and the latitude and longitude provided from the 3.14 `UBX-NAV-PVT` message, which is a computed navigation solution. To correctly point the user in the right direction we need to know his heading of motion. We acquire this from the same message as the position, alongside with the accuracy of this heading. If the accuracy is worse than a threshold we use the true north angle from the compass of the watch. This is done because when the user is standing still, we have no heading, or when the accuracy of the heading is unreliable, we can still acquire a measurement from the compass. On the other hand this introduces another module to the function, which will increase the consumption and due to the moving watch hands influencing the magnetic field, the compass measurement and hand movement cannot be done concurrently. When the accuracy of the heading is good, we do not need any other module, because the computed navigation solution provides as with any information we may need for navigation.

Listing 3.16: Getting the heading of motion

```
uint16_t headAcc = gps_get_heading_accuracy();
int32_t  speed   = gps_get_ground_speed();

if ((speed < TAKE_ME_HOME_MINIMUM_SPEED))
{
  first_location_update = true;
```

```
  return TAKE_ME_HOME_NO_HEADING;
}

if (headAcc > TAKE_ME_HOME_HEADING_ACC_THRESHOLD)
{
  heading = TAKE_ME_HOME_NO_HEADING;
}
else
{
  heading = TO_ABS_DEG(gps_get_heading_motion()
          / HEADING_CONVERSION_VALUE);
}
```

First we read out the heading of motion accuracy and the ground speed. We consider the user not moving while the ground speed is below a certain threshold. We indicate a no heading state when either the ground speed is low or the accuracy is above a given threshold. In this case the true north angle is used in another part of the code. It reacts to a compass event from the watch's event manager. If none of the conditions mentioned above are true, than the heading of motion provided by the GNSS module is considered correct and converted to be in an interval from 0 to 359 degrees.

When we have the correct heading of motion, we use it and the direction (bearing) calculated beforehand to correctly point the user in the direction of the next point. This is achieved by subtracting the acquired heading from 360 degrees, by doing that the heading becomes the "north" and by adding our bearing to that we get the direction we need to point the user 3.17. Then we set the steps of the motors of the watch hands to represent this fact. If the compass is used the procedure is the same, but instead of the heading we subtract the true north angle.

Listing 3.17: Adjust the heading correctly

```
uint16_t adjustDirection(float dir, int32_t heading)
{
  uint16_t adjusted = 360 - heading;
  uint16_t adjDir   = TO_ABS_DEG((int16_t)dir);
  return TO_ABS_DEG(adjDir + adjusted);
}
```

When the compass is used, we need to stop it during the moving of the motors, because moving watch hands can influence the magnetic field of the Earth. The direction is shown by the minute hand with the hour hand being opposite to it. The hands move as the heading of the user changes, so does the displayed direction.

3.3 Phone application

The phone application is designed mainly for testing purposes and to demonstrate the assistance data transfer and take me home algorithm. It is a simple android application consisting of one activity and one foreground service.

3.3.1 Handling the assistance data

One of the main purposes of this application is to download the GNSS assistance data and supply it to the watch via Bluetooth. For this action we designed several buttons with actions attached to them.

The GET DATA button handles the download of the data and storing it into a buffer. The download is done by a HTTP GET request to a specified address with some parameters further specifying the constellations, almanac, duration and resolution of the data 3.1. Activity diagram describing this behavior is in 3.15.

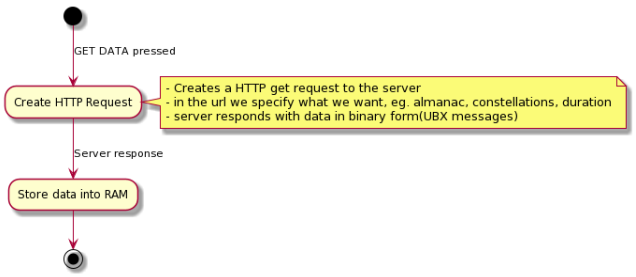

Figure 3.15: GET DATA activity. Source: own work.

Next in the process is the PROCESS DATA button, which saves the data from a buffer into a file to the physical storage of the phone. After that the data are given structure to help us parse them on the watch side and easily navigate them. By giving structure we mean the addition of starting date, number of messages in a day, duration, initial latitude and longitude 3.16.

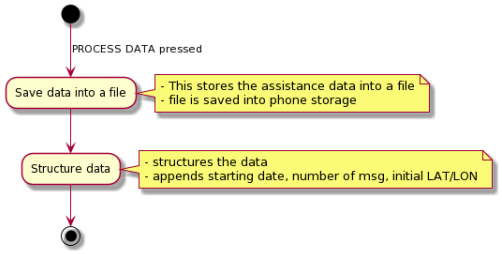

Figure 3.16: PROCESS DATA activity. Source: own work.

When everything is processed and ready the CONNECT button is used to select the desired device we wish to send the data to and then the SEND FILE button initiates the file transfer as shown in 3.6. After everything was sent the application sends a finalizing message indicating this fact 3.17.

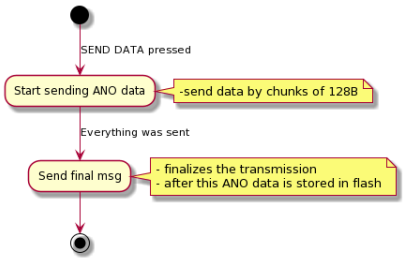

Figure 3.17: SEND DATA activity. Source: own work.

3.3.2 Location service

The foreground service mentioned in the beginning of this section handles all location related actions. That means setting up a periodic location request, the period itself and a callback function that is called whenever a change in location is registered. This function structures the new latitude and longitude alongside the previously acquired location and packs it with the calculated distance and direction into a string. The string is then sent via a broadcast to the main activity of the application where it is stored as well as sent via Bluetooth when a device is connected 3.18.

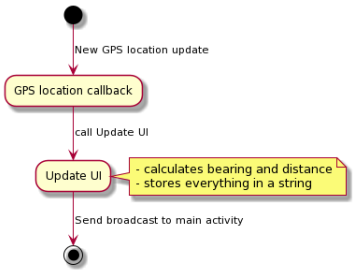

Figure 3.18: Gps foreground service. Source: own work.

3.3.3 Sending the location

The sending of the location is tightly connected to the location service. If the user activated the location sending, each time a new location is reported by the GPS service, it is formatted and sent via Bluetooth to the watch. As mentioned in the Take me home algorithm section, it is converted from floats

to integers and then from big-endian to little-endian and sent. The total size of this packet is ten bytes, two bytes for the operation code and four bytes for latitude and longitude.

Chapter 4

Testing & measurements

In this final chapter of this thesis we will discuss the results of some live tests and measurements done on both a development breadboard and the actual watch.

The live tests consist of walking around with the watch attached to the wrist and using the phone application to send the location to the watch, and the phone itself to log the actual location. Then using the "Take me home" function to navigate to the starting location. All images were created from logs gather from both the watch and the mobile phone and then visualized using the website `https://www.gpsvisualizer.com/`.

The measurements are done by special power measuring equipment both on the development breadboard with an active antenna attached and on the watch. The placement of the antenna and watch were the same for both cases. What we wanted to see was the difference the assistance data will make in reducing the time to position acquisition therefore reducing the power consumption.

4.1 Live tests

4.1.1 First test case

The first test was conducted at the Strahov dormitories in Prague. It was done on a large parking space surrounded by several concrete buildings each of them approximately 20 meters tall.

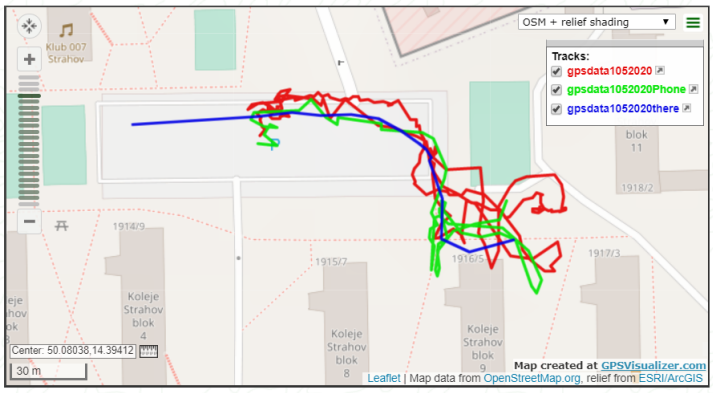

Figure 4.1: First live test using the take me home functionality. Source: own work.

We can see several curves displayed in the picture 4.1. The blue one represents the "way there" and it was logged by the navigation system of a mobile phone. This is the path we would like to navigate back on. The green and red paths are the "way back", with green being the track logged by the phone and the red one being the watch track. We logged them by

multiple methods for comparison. As we can see the red track is inaccurate and we are getting an error of approximately 20 meters. The green track shows, that the algorithm was trying to compensate these errors, that is why there is additional walking going off the actual path. However even after this additional walking we were able to reach the starting point.

The source of these errors could be the so called multipath errors, which are errors in location generated by reflections of the signal (for example by buildings) and therefore resulting in false location. As we will see in another test case, this does not happen in an area without taller buildings.

4.1.2 Second test case

The second test case was done at a large park area with no tall buildings in the vicinity.

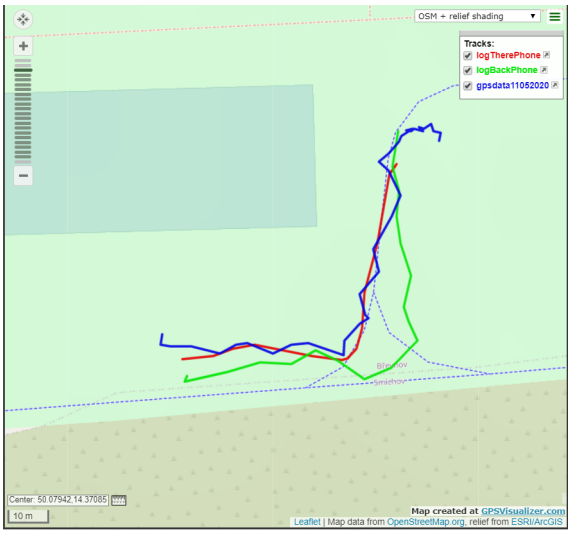

Figure 4.2: Second test case, at Ladronka park in Prague. Source: own work.

Referring to 4.2 the red path is the "way there", and just like in the first test case we have two different tracks for the "way back", one from the mobile phone (green) and one from the watch (blue). We can immediately see that there are almost no inaccuracies and that the track from the watch is more accurate then the one from the phone. This test case consisted of about 3 minutes walking in one direction and the same amount of time in the opposite direction. The logs were extracted at the end of each path to prevent overwriting.

4.1.3 Third test case

The location is the same as the second test case, the difference is the time of day which was approximately 2 to 3 hours after the second test case.

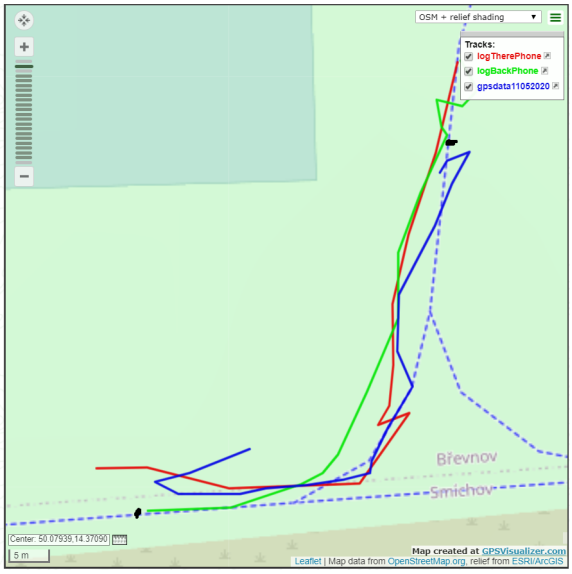

Figure 4.3: Third test case, at Ladronka park in Prague. Source: own work.

Same as the previous test case we can see that the blue path (watch "way back") is more accurate than the green path (phone "way back") and copies the red path (phone "way there") more closely just until the end where it deviates a little, but it is still within the 5 meter acceptance radius.

Figure 4.4: Photo of the watch displaying the actual take me home information - the number of points, the distance and the watch hand point to the direction of this point. Source: own work.

In 4.4 we can see the actual watch displaying the information about the current track. On the photo the track was taken from the watch itself and we can see a total of 77 points, the distance to the next point and the direction shown by the watch hands.

4.2 Power consumption measurements

4.2.1 Equipment

For power consumption measurements we used the N6705C power supply from Keysight technologies 4.5. We were using 3.7 V of supply voltage and 60 mA current limit for both the development board and the watch. The measurements were run for 10 minutes with both the assistance data present and absent.

Figure 4.5: Keysight N6705C power supply used for consumption measurement. Source: own screenshot taken from [6].

4.2.2 Development board measurements

Measurements on the development board were done using an attached active antenna from Taoglas, the Ulysses AA.162 model. The development board is similar to the watch in sense of components used. It is more suitable for development and testing due to its accessibility of individual components, their pins and the ability to easily swap software inside the main CPU.

We can see the measurement results in 4.6. The measurement was run for 10 minutes, which is not visible in the picture due to cropping. In the bottom of the picture we can see the measurement at the markers. Here the markers are set to measure the acquisition portion of the location process. As we can see the consumption during the acquisition phase is around 27 - 28 mA. The difference between the setup with the assistance data and without it, differs only by 0.2 mA in favor of the measurement with the assistance data present.

In 4.7 the measurement of the tracking phase is shown. This phase takes up the majority of the measurement time, about 65% - 70%. The power

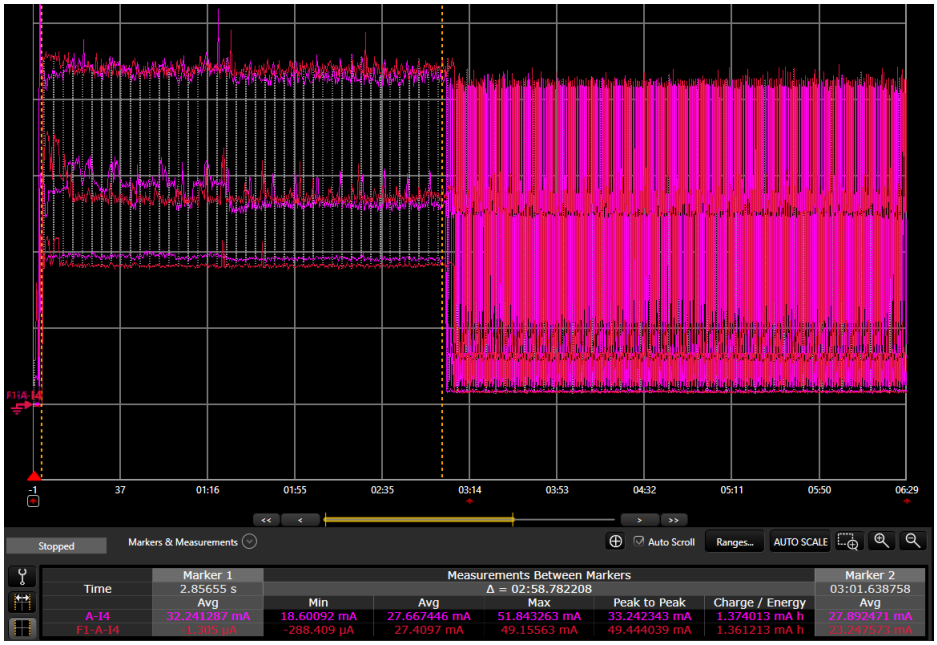

Figure 4.6: Measurement on the development board with an attached active antenna. The red color depicts the measurement with the assistance data, the purple color without the data - Acquisition phase. Source: own screenshot taken from Keysight software[6].

consumption decreases from 28 mA to 5 mA and this is one of the reasons we want to shorten the acquisition phase as much as possible. Just as in the acquisition phase we do not notice any significant drop in consumption when we compare the two setups.

It has to be noted that this measurement is done on the board as a whole. That means that we measure the whole functionality, so the consumption includes the activity of the MCU, the communication buses, display, other modules that are running and of course the GNSS module.

4.2.3 Watch measurements

To be able to measure the current on the watch, we need to attach it to a external component which enables us to access certain pins and modules of the watch. We connect our external power supply onto this external component to power the watch and measure the consumption at the same time.

Unfortunately we were not able to get a proper position fix, using the same location as the active antenna, and due to the character of the measurement setup we were not able to move it outside. Still we provide a measurement for 10 minutes just to demonstrate the power consumption of the acquisition phase. As we see in 4.8 the consumption is on 35 - 37 mA the whole time,

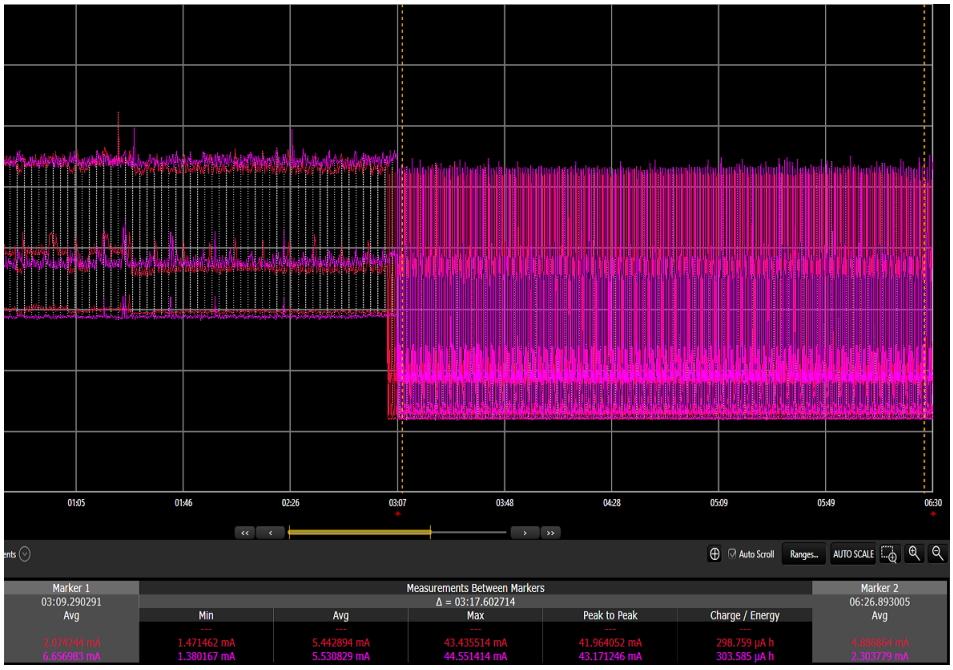

Figure 4.7: Measurement on the development board with an attached active antenna. The red color depicts the measurement with the assistance data, the purple color without the data - Tracking/Power optimized phase. Source: own screenshot taken from Keysight software[6].

which will drain the battery of the watch in no time. The unsuccessful result of this measurement lies in the placing of the watch. While the active antenna is strong enough to pick up the signal even from this place, the passive antenna fails to do so and the fact that the building is blocking half of the field of vision of the watch is not helpful. There is bound to be signal blocking and multi-path interference.

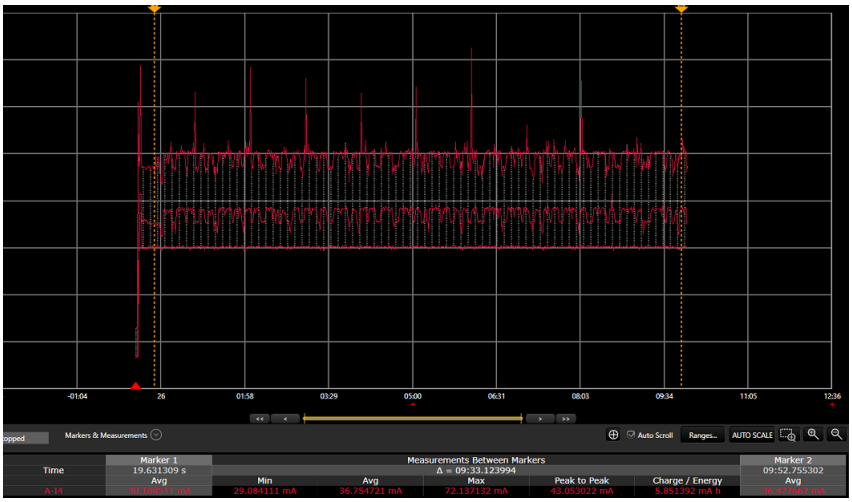

Figure 4.8: Measurement on the watch which has a passive antenna, measurement with assistance data present - acqusition phase only. Source: own screenshot taken from Keysight software[6].

Conclusion

Navigation is a topic which can be talked about excessively and several thick books have been written about this topic. In this thesis we covered the basics of navigation briefly in the first chapters. We mentioned how navigation evolved from plain piloting, through radio waves and into nowadays satellite systems. Brief history of the largest Global Navigation Satellite Systems such as the GPS, GLONASS, Galileo and BeiDou, how they evolved and how the majority of these systems was designed for military purposes at first. Then we moved onto Bluetooth low energy, which is another key technology behind the main goals of this thesis. Brief history, key architectural concepts and connections were discussed.

In the second chapter we moved onto the main goals of the thesis, discussing the analysis and design of the assistance data and the Take me home functionality. The target application for these functions is a wrist-worn watch and the hardware of this watch which was used throughout the thesis is the nRF52480 MCU from Nordic Semiconductor and the ZOE-M8B navigation module by U-Blox. We started by downloading the necessary assistance data from the dedicated U-Blox servers. For this purpose we used a mobile phone application written for Android. This mobile application was developed by the author of this thesis from scratch, and serves as a demonstrating and testing application. It was written in the Java programming language and the Android platform was chosen because of previous experience of the author. After the data has been downloaded the application attaches to them some meta data, such as the starting date, validity of the data, initial location, etc. Then the assistance data are sent via Bluetooth to the watch, which stores them into its flash memory. When everything has been sent and stored, the data are supplied to the GNSS module. The main purpose behind the assistance data is to speed up the acquisition of the user's location and thus lowering the power consumption because the acquisition phase is quite power demanding. It also helps in low signal environments because the receiver needs only a range measurement from the satellites due to the fact that the assistance

data provide everything else.

The Take me home functionality uses the above mentioned results in the acquisition of a track. The main purpose of this function is to navigate a user back on a previously acquired track. This is useful when the other means of navigation runs out of battery, needs to conserve battery, stops functioning or the user simply wants to go back on the walked track. For this purpose we introduce two means of getting a track. The first is via a mobile phone, which uses the same application as the assistance data. The application periodically sends a location via Bluetooth to the watch which stores it. The second type of track is logged by the watch itself and using the assistance data speeds up the process of acquisition so the user can get accurate locations. If both tracks are present in the watch, the user is prompted to chose one for the Take me home function. The function then navigates the user back, point by point, displaying the distance to the next point and the watch hands show the direction to this point. For the watch hands to properly point in the right direction we need a heading of motion, this is acquired from the GNSS module or if it is inaccurate or the user is standing still we get it from the compass. The compass has the advantage that is always provides data but it is another module that needs to be calibrated and powered. While the GNSS heading of motion is provided in the solution we use to get the location, but can get inaccurate.

As stated before, the main weakness of these functions and the GNSS in general is the signal reception and multi-path errors. Weak signal can be partially resolved by the assistance data. The presence of the assistance data eliminates the need to get this information from the satellites, but a range measurement is still needed to determine the location. That means when the path to the satellite is blocked by something it may be impossible to obtain a location. The multi-path errors are a phenomenon where the radio signal reaches the receiver by more than one path. This can be caused, for example, by reflections from buildings, mountains or water bodies. This causes multi-path interference, including constructive or destructive interference, causing the signal to become too weak. Possible solution to this is the usage of a Kalman filter as stated in the GNSS section of the thesis. One more limitation that comes into mind is the memory. One week of assistance data take up about 33 kilobytes, but the storing of the track is what is limiting. We can store a track depending on the usage of the flash memory by other functions of the watch.

For future improvements we can recommend the usage of the previously mentioned Kalman filter. Here the important question to ask is, whether the navigation module and its features are that essential, that it is worth to implement such a filter, which uses matrix operations and data from several other modules. Another improvement may be to change the chip of the GNSS module. The chip discussed in this thesis can offer assistance data for GPS and GLONASS only, although it can use all the other constellations. Having

the ability to acquire assistance data for both Galileo and BeiDou alongside GPS and GLONASS could speed up the acquisition phase even more and retain position even better.

All of the code used and presented in this thesis is provided in the attached portable medium. Both the phone application written in Java and the code for the watch in C are documented. Sometimes only a part of the code which was written by the author and then inserted into a bigger module is provided.

Bibliography

[1] Kr7cmw0l: Diagram of Earth Centered, Earth Fixed coordinates in relation to latitude and longitude. [cit. 2020-06-28]. Dostupné z: https://commons.wikimedia.org/wiki/File:ECEF.png

[2] Mike1024: ECEF ENU Longitude Latitude relationships. [cit. 2020-06-28]. Dostupné z: https://commons.wikimedia.org/wiki/File:ECEF_ENU_Longitude_Latitude_relationships.svg

[3] simeon2, I.: GPS trilateration. [cit. 2020-06-30]. Dostupné z: https://commons.wikimedia.org/wiki/File:GPS_trilateration_fig1.jpg

[4] Lithium57: Multipath propagation. [cit. 2020-07-26]. Dostupné z: https://commons.wikimedia.org/wiki/File:Multipath_propagation_diagram_en.svg

[5] U-Blox: *u-blox 8 / u-blox M8 Receiver description [online]*. [cit. 2020-06-25]. Dostupné z: https://www.u-blox.com

[6] Keysight: Keysight N6705C. [cit. 2020-07-02] - Screenshot. Dostupné z: https://www.keysight.com/en/pd-2747858-pn-N6705C/dc-power-analyzer-modular-600-w-4-slots?cc=CZ&lc=eng

[7] Mohinder S. Grewal, C. G. B., Angus P. Andrews: *Global Navigation Satellite Systems, Inertial Navigation, and Integration*. John Wiley & Sons, Incorporated, 2013.

[8] Groves, P. D.: *Principles of GNSS, Inertial, and Multi-sensor Integrated Navigation Systems*. Artech House, 2007.

[9] Eubank, R. L.: *A Kalman Filter Primer*. CRC Press LLC, 2005.

[10] Gupta, N.: *Inside Bluetooth Low Energy*. Artech House, 2013.

[11] NORDIC Semiconductor: *nRF52840 Product Brief [online]*. [cit. 2020-07-25]. Dostupné z: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840

[12] U-Blox: *ZOE-M8B Product Brief [online]*. [cit. 2020-07-25]. Dostupné z: https://www.u-blox.com/en/product/zoe-m8b-module

[13] Macronix: *MX25R32 Product Brief [online]*. [cit. 2020-07-25]. Dostupné z: https://www.macronix.com/en-us/products/NOR-Flash/Serial-NOR-Flash/Pages/spec.aspx?p=MX25R3235F&m=Serial%20NOR%20Flash&n=PM2159

Appendix A

List of abbreviations

AGNSS Assisted global navigation satellite system

BLE Bluetooth low energy

GLONASS Global'naya Navigatsionnaya Sputnikovaya Sistema

GNSS Global navigation satellite system

GPS Global positioning system

MCU Micro Controller Unit

SPI Serial Peripheral Interface

UART Universal asynchronous receiver/transmitter

Appendix B

Contents of the attached CD

- `readme.txt` ..... brief CD content description
- `src`
  - `impl` ..... source codes of the implementation
  - `thesis` ..... source code of the thesis in LaTeX
- `text` ..... text of the thesis
  - `thesis.pdf` ..... text of the thesis in PDF format
  - `thesis.ps` ..... text of the thesis in PS format