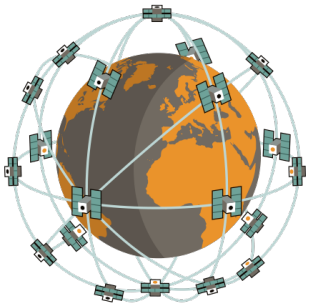

Project Report
Project 1-9
Waypoint navigation
for an autonomous all-terrain vehicle

Jani Arponen
Anand George
Sakke Jussmäki
Tommi Penttilä
Ville Piesala
Markus Sarlin

29.05.2020

Aalto University

ELEC-E8004 - Project work – year 2020

Information page

Students

Jani Arponen
Anand George
Sakke Jussmäki
Tommi Penttilä
Ville Piesala
Markus Sarlin

Project manager

Jani Arponen

Official Instructor

Tabish Badar

Starting Date

16.1.2020

Completion Date

29.5.2020

Approval

The Instructor has accepted the final version of this document Date: 29.5.2020

Abstract

Waypoint navigation in autonomous systems relies heavily in knowing the precise position of the system in question. The Polaris e-ATV used by the Aalto University's Autonomous Systems Research Group is one such system in need of more accurate positioning data. The aim for this project was to implement Differential Global Navigation Satellite System (DGNSS) corrections for the Polaris, using the hardware already installed on the Polaris, with corrections provided by the National Land Survey of Finland.

The project goal was significantly altered as a result of the actions taken by the university in response to the COVID-19 pandemic. The project group lost all access to the hardware. The original goals of the project could not be met, but auxiliary objectives were explored and implemented. Post-processing GNSS data together with the DGNSS corrections was researched, but deemed impossible with the limited tools available during the COVID-19 lockdown. Finally, a simplified satellite based positioning system simulation was implemented in Matlab. An Extended Kalman Filter is used in estimating a simulated rovers state based only on noisy range measurements of said satellites. Future work recommendations for expanding the satellite simulation is also presented.

Contents

Figure 1: The Polaris Ranger e-ATV, the Aalto University autonomous vehicle.

1 Introduction

An autonomous vehicle has been under development in a series of research and student projects at the Aalto University's Autonomous Systems Research Group. The projects have been completed over a period of several years. In these projects the autonomy of a Polaris Ranger electric all-terrain (figure 1) has been improved incrementally with additions of both new hardware and software.

In his master's thesis Tabish Badar [1] – the instructor of this project – underlines the need for more accurate positioning of the autonomous vehicle. Following this suggestion, a project involving the implementation of a Differential Global Navigation Satellite System (GNSS[1]) was included as a topic in the spring 2020 Aalto School of Electrical Engineering project course. According to Badar, the addition of a DGNSS system would expedite the vehicles initialisation process and enhance the precision of the current localisation of the vehicle further improving the performance.

1.1 Necessary theory on Global Navigation Satellite Systems

Due to the nature of our project, it is important to go over some of the theory and maths behind satellite based positioning systems. Global Navigation Satellite System allows for positioning any-

[1]Previously the US-based GPS was the only available system, but currently also the European Galileo, Russian GLONASS, and Chinese BeiDou are openly accessible to all – together they are called GNSS.

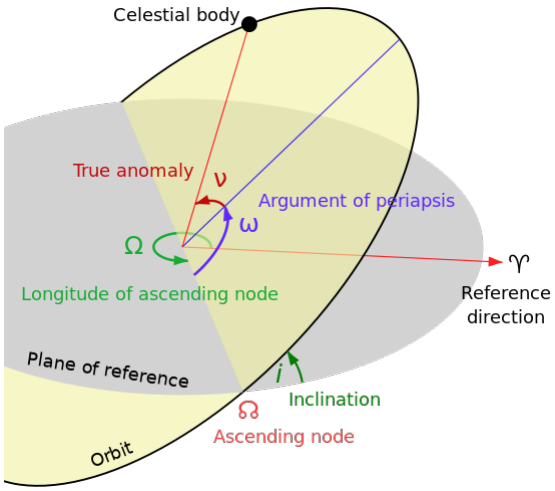

Figure 2: Angles that describe orbits of celestial bodies (satellites). [2]

where on the planet using a constellation of satellites, which continuously broadcast their current position in orbit and the timestamp of their very accurate atomic clocks. The signal is received by a GNSS receiver and the data is used in determining the receivers location.

1.1.1 Satellite orbits

GNSS satellite orbits are described with Keplerian orbital elements [3], with the Earth as the center of mass and with the equator as the plane of reference. A subset of the below parameters are shown in Figure 2. The orbit parameters can be used to convert to the Cartesian coordinates of the satellite at a specific time, which are needed in the satellite ranging equations explained later.

- e Eccentricity – Shape of the elliptical orbit, deviation from a perfect circle.
- a Semimajor axis – Average of the periapsis and apoapsis distances of the ellipse, i.e. closest and furthest away point from the focal point (center of mass).
- i Inclination – Tilt of the orbit compared to the reference plane. Measured at the ascending node, i.e. where the orbit plane intersects the reference plane and passes upward.
- Ω Longitude of the ascending node – Angle of the ascending node when compared to a fixed point on the reference plane.
- ω Argument of periapsis – Angle of the periapsis (closest point to Earth in orbit) with respect to the the ascending node
- θ True anomaly – (v in the figure) Position of the satellite within the orbit at a specific time.

1.1.2 Satellite ranging

There are two equations [4, pp. 506], [5, pp. 482] for the true range r_i between the satellite i^{th} and the receiver

$$\begin{cases} r_i &= \sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} \\ r_i &= c(t_r - T_i) \end{cases} \tag{1}$$

where X_i, Y_i, Z_i are the Cartesian coordinates of satellite i; x, y, z are the Cartesian coordinates of the receiver; c is the speed of light; T_i is the time of signal broadcast; and t_r is the time of signal reception at the receiver.

However, these do not hold in reality due to many sources of error [4, pp. 506], as such the pseudorange p_i to each satellite i can be calculated from

$$p_i = \sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} + cb + w_i \tag{2}$$

where b is the clock bias of the receiver, i.e. the offset from GPS time; and w_i is measurement error, which can be modelled as a random variable.

Since the satellites transmit their position in orbit and their time, it is easy to see from equation 2, that the unknowns of the system are the stationary receivers position x, y, z and clock bias b. Thus, it is necessary to have line of sight to at least 4 satellites in order to solve for the system of equations. When the receiver sees more than 4 satellites, it is possible to reduce the measurement error w through e.g. the least squares estimator or Extended Kalman Filter. These methods are discussed further later in section 8 of this report.

1.1.3 Differential Global Navigation Satellite System

The way GNSS positioning works is by measuring the distances between positioning satellites and the receiver as shown in equation 2. In this type of positioning, however, there are a number of error sources that comprise w_i , the largest of which are ionospheric and topospheric error sources [4, pp. 509]. There are multiple solutions for improving the precision of satellite localisation such as Differential GNSS (DGNSS) and RTK (Real-Time Kinematic) to name a couple. This project concentrates on DGNSS.

Comparing the positioning performance based on GNSS and Differential GNSS, the advantages of DGNSS become apparent. Though the actual accuracy has been improving since the release of the 2008 GPS standard [6, 7], it still does not provide adequate precision for reliable autonomous operation: the GPS standard guarantees a horizontal accuracy of only 7.8 meters for 95 % of the time.

Using DGNSS, most of the aforementioned errors can be subtracted from the measured pseudorange equation 2 directly. The basic concept is to calculate the difference between the GNSS positioning and a geo-reference station, whose position is known to a high degree of accuracy and often employing a more accurate clock than on typical GNSS receivers. As the x, y, z terms are known and b has lower variance (due to a better clock), the error (and its evolution) w_i can be estimated and

thus a correction Δp_i to the pseudorange p_i can be obtained, giving the naive correction equation for the standard receiver

$$\hat{p}_i = \sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} + cb + \Delta p_i + v_i \tag{3}$$

where v_i is the new, smaller measurement error.

This technique works well for the large atmospheric error sources and is applicable within a few tens of kilometers from the reference station, where the corrected position accuracy is in the order of 1 m (68 % of the time). The accuracy decreases as a function of distance by a factor of 1 m per 150 km between the receiver and the reference station. The correction works, since the variations of the atmosphere are effectively the same within that region – more precisely, they are similar for the satellite-reference station and satellite-receiver paths. These corrections are sent to the users' device from the geo-stationary reference station. Different transmission strategies include broadcasting over commercial FM radio and streaming the data over internet with NTRIP (Networked Transport of RTCM via Internet Protocol) [8].

The objective of this project, as described in more detail below, was to utilise the e-ATV's hardware, namely the GNSS receiver and SPAN unit (Synchronous Position, Attitude and Navigation) and implement DGNSS corrections for positioning. The National Land Survey of Finland (NLS) provides these corrections, calculated at 50 FinnRef stations [9] and sent over the internet, free of charge.

2 Objective

2.1 Original objective

A more thorough explanation of the objective is found in Appendix A, which hosts the original project plan. However, here is a short summary.

The original goal of this project was to develop an embedded system that would provide DGNSS corrections for the Polaris e-ATV. The embedded system was supposed to be a Raspberry Pi (RPi) computer running ROS (Robot Operating System), that could have been easily integrated to the current ROS network running on the Polaris. The DGNSS correction data would be obtained from the DGNSS services [11] provided by the National Land Survey of Finland.

The target users for the system were the members of the Autonomous Systems research group, especially those working with the Polaris e-ATV. The system would have been installed and connected to the ROS network of the e-ATVs SPAN (or another system running ROS). The embedded system would post the corrections or corrected coordinates as a ROS topic to the ROS-network. After installation the system would have worked autonomously and further intervention would not have been necessary.

2.2 The Corona virus and its implications

The project was carried out according to the original plan until university premises were closed in March due to COVID-19. Permanent loss of lab access was not expected in our risk analysis (although partial loss was considered pp. 12, Appendix A) and equipment and the e-ATV was no longer accessible. Initially there was hope that the university premises and necessary equipment would be accessible at some later point in time, but after few weeks of uncertainty, it was finally clear that the original objectives would not be achievable. A more thorough and chronological explanation of the effects by COVID-19 are found in Section 7.

2.3 Modified objectives

As the original objectives could no longer be achieved, new objectives for the project were generated. These are also covered in more detail Section 7. The goal was to keep the new objectives as close as possible to the original objective of improving Polaris' position estimate with DGNSS corrections. However, it was not certain what could be done in the changed situation and for this purpose several, different methods had to be tried before feasible objectives were found.

3 Project plan

This chapter presents a summary of the project plan. The full project plan can be found in Appendix A. The effects of COVID-19 on the project plan are explored in detail in Section 7.

3.1 Phases of the project

The project was divided into five different phases. These phases were *planning*, *design*, *business aspects*, *(Software) development and testing*, and *Reporting and documentation*. During the planning phase, the objective was to get acquainted with the given materials, theory, manuals, documents, etc., and figure out what was needed to be done during the project. The goal for the design phase was to design the system based on the original requirements and objectives, based on what was learned during the planning phase. In the business aspects phase a business case for the device under development was planned, documented, and presented. In the (Software) development and testing phase, the planned project tasks were to be implemented and tested. This would, arguably, be the largest and most important phase of the project. Finally, in the reporting and documentation phase, a presentation was to be made for the final gala and this final report was to be written.

The milestones for each of the phases and their deadlines are presented below in Table 1. The deadlines that are **bolded** were hard deadlines imposed by the project course. The deadlines that are in *italics* were subject to change in further iterations and as the project progressed.

Phase	Milestone	Deadline	Description
Planning	P1	2020-02-03	Project plan submission to instructor
	P2	**2020-02-06**	Project plan submission to MyCourses
Design	D1	*2020-03-01*	System infrastructure document
	D2	*2020-03-01*	Software requirements specification
Business	B1	**2020-03-06**	Business aspects seminar slides submission to MyCourses
	B2	**2020-03-10**	Business aspects seminar and presentation
	B3	**2020-03-13**	Business aspects document submission to MyCourses
Development and testing	M1	*2020-02*	A working development platform (RPi with ROS)
	M2	*2020-03*	Client for NLS DGNSS corrections
	M3	*2020-03*	DGNSS corrections published to ROS network
	M4	*2020-04*	Data correctness testing
	M5	*2020-05*	Integration testing with Polaris e-ATV
	M6	*2020-05*	Delivery of the software and associated documents
Reporting	P3	**2020-05-19**	Final gala and presentations
	P4	*2020-05-22*	Final report submission to instructor
	P5	**2020-05-29**	Final report submission to MyCourses

Table 1: Project phases and milestones.

3.2 Division of work

In the original plan, the work was divided among the members of the group. The division of work had taken into account the skills and preferences of the group members. Work was divided into work packages and each work package was to be worked on by a minimum of two people to minimize risk

of not getting work done before deadlines due to i.e. illness or loss of focus. The work packages and tasks are listed in Table 2 – and are further expanded in Table 3. If applicable, the work packages are linked to their associated milestones and the tasks have an associated time estimate in person-hours. The time estimates are exactly that, estimates, and they were very conservative regarding the amount of hours needed.

Work Package	**Description**
WP1 - Project plan:	Writing the project plan and other associated tasks.
WP2 - Project management:	Handle administrative tasks regarding the project, such as communication, scheduling, documenting etc.
WP3 - Business aspects:	The business aspects aim to produce a business proposal, which will be presented in business aspects seminar.
WP4 - Hardware:	Obtaining and handling the separate pieces of hardware needed for the project. Figure out how to integrate them to the system and iron out any possible problems.
WP5 - ROS system:	Set up the ROS environment on RPi to develop on. Later, integrate it to Polaris ROS network.
WP6 - DGNSS client:	Obtain DGNSS corrections from NLS DGNSS API.
WP7 - "our own" client:	Develop our own (simple) algorithm for reducing GPS error e.g. moving average
WP8 - RTK client:	Implement RTK using the RTKLIB.
WP9 - Data collection:	Run the clients on Polaris and collect data on performance. Analyze data and create demonstrations.
WP10 - Final report:	Create final gala presentation and write final report.

Table 2: Work package descriptions.

WP#	MS#	Assigned	Task#	ph	Description
WP1	-	All	Task 1.1	20	Reading Tabish's thesis, GNSS tech, etc.
	P1	All	Task 1.2	30	Write first draft of project plan
	-	All	Task 1.3	10	Iterate project plan based on feedback
	P2	Jani	Task 1.4	1	Submission of project plan to MyCourses
WP2	-	All	Task 2.1	180	Weekly project meetings
	-	Jani	Task 2.2	100	Project management administrative work
WP3	-	All	Task 3.1	40	Business aspects documents
	-	Markus, Jani, Ville	Task 3.2	25	Rugged enclosure design for DGNSS module
	B2	All	Task 3.3	35	Business aspects seminar
	B1, B3	Jani	Task 3.4	1	Business aspects documentation submission
WP4	D1	Sakke, Markus	Task 4.1	5	Raspberry Pi and SD-card
	D1	Sakke, Markus	Task 4.2	5	3g/4g USB modem and SIM-card
	D1	Sakke, Markus	Task 4.3	5	GPS antenna/module
	D1	Sakke, Markus	Task 4.4	5	USB to RS232 connector
	D1	Ville, Tommi	Task 4.5	20	Enclosure for the module
	D1	Ville, Tommi	Task 4.6	10	Testing hardware in different environments, within the campus area
	D1	Sakke, Markus	Task 4.7	15	Documenting system infrastructure
	-	Sakke, Markus, Jani	Task 4.8	5	Power supply
WP5	M1	Sakke, Markus	Task 5.1	5	Raspbian config
	M1	Sakke, Markus	Task 5.2	10	ROS running on Raspbian
	M5	Sakke, Markus	Task 5.3	100	Integrating to Polaris ROS network
	M3	Sakke, Markus	Task 5.4	5	ROS publisher for DGNSS corrections
	-	Sakke, Markus	Task 5.5	5	Implement logging for location data
WP6	D2	Anand, Tommi, Ville	Task 6.1	50	Researching NLS DGNSS API
	M3	Anand, Tommi, Ville	Task 6.2	100	Design and coding of the DGNSS client
	M3, M4	Anand, Tommi, Ville	Task 6.3	20	Testing the DGNSS client
WP7	D2	Jani, Anand, Tommi	Task 7.1	80	Design and coding of "our own" GPS averaging client
	M4	Jani, Anand, Tommi	Task 7.2	20	Testing the GPS averaging client
WP8	D2	Markus, Anand, Ville	Task 8.1	100	Researching RTK technology and RTKLIB
	-	Markus, Anand, Ville	Task 8.2	120	Design and coding of the RTK client
	M4	Markus, Anand, Ville	Task 8.3	30	Testing the RTK client
WP9	-	All	Task 9.1	50	Hands-on with Polaris for testing
	-	All	Task 9.2	50	Data collection on different solutions performances
	M4, M5	All	Task 9.3	10	Evaluating the different clients using the Polaris e-ATV, comparing to current solution
WP10	P3	All	Task 10.1	100	Final gala presentation
WP10	P4, M6	All	Task 10.2	100	Final report to Tabish
WP10	P4	All	Task 10.3	20	Possible iteration on final report
WP10	P5	Jani	Task 10.4	1	Final report submission to MyCourses

Table 3: Work packages expanded with assigned members and time estimates.

4 System description and hardware

4.1 The Polaris ATV

The Polaris Ranger electric ATV is a commercially available product. It is a fully electric all-terrain vehicle. This research vehicle has been fitted with various devices to enable autonomous driving. Actuators have been placed on the steering linkages. Both the steering and the drivetrain can be controlled using a PLC (Programmable Logic Controller), which runs the low-level control loops.

The chassis has been modified to include sensors to track odometry while driving autonomously. Other sensors include a rotating LiDAR on the top of the roll cage. Under this sensor is a omnidirectional camera. These two sensors give the ATV a full view of the surroundings both visually and as a 3D pointcloud.

On a higher level, the ATV uses ROS for control. It is used to send data and to control messages between various devices. ROS is also used to run computation nodes that handle for example the waypoint navigation algorithm.

4.2 SPAN

The robot has a hardware unit called SPAN (Simultaneous Position, Altitude and Navigation). It combines data from two different, but complementary positioning and navigation systems: GNSS and Inertial Navigation System (INS). The SPAN unit is a product of NovAtel Inc. The accuracy of GNSS positioning and the stability of the IMU, gyro and accelerometer measurements are tightly coupled to provide an exceptional 3D navigation solution that is stable and continuously available, even through periods when satellite signals are blocked [12].

For reducing GPS measurement errors, differential GPS (DGPS) correction data can be used to correct the pseudoranges. The pseudoranges are the distances from each satellite to the receiver and they are used to calculated the position of the receiver on Earth. The corrections improve the positioning and the error gets reduced to the order of 1 m [8].

4.3 DGNSS Module

A Raspberry Pi 3 (RPi) single board computer was chosen as the platform to implement the DGNSS functionality. Raspbian, a Debian-based operating system for the Raspberry Pi is installed on the board. Since the existing system uses ROS, it too is installed on the RPi. The process for receiving the DGPS correction data and sending it to SPAN is executed as a ROS node.

The National Land Survey of Finland (NLS) provides the DGNSS correction data which can be fetched using a HTTP APIs. In order to fetch the data, the DGNSS module requires internet access. For this purpose a GSM module is used. In order to determine the mount point (the station from which the correction are sent) from which the corrections have to be fetched, a GPS receiver is also connected to the module.

5 Setting up ROS on the Raspberry Pi

Officially ROS is supported only on Ubuntu running on a x86 architecture [13]. The Raspberry Pi is built on an ARM architecture and therefore there are no official precompiled versions that could simply be installed through the *APT* software distribution system. However, it is entirely possible to compile ROS on the RPi from source. Two alternative approaches were attempted: Using a disk image compiled and distributed by Ubiquity Robotics, and compiling ROS from source on Raspbian.

Ubiquity Robotics is a company providing robotic platforms for developers [14]. Meant primarily for their own systems, they provide a Linux disk image with a pre-installed ROS environment through their website [15]. Despite the original purpose, it is well suited for other uses and the license is free for academic use. As explained below, the compiling issues were not platform-related, and this approach was not any more fruitful than compiling ROS on Raspbian.

As a Debian-based Linux system, the Raspbian is capable of running ROS. The ROS tutorial section includes a step-by-step guide on how to compile ROS on Raspbian
(http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi)
[16]. The external sources listed in the tutorial are also essential to know. It should be noted, however, that even if installation is technically possible, it is not guaranteed to work seamlessly with all available packages.

Additionally, the hardware on the RPi Model 3B is not very powerful. The small system-on-a-chip computer has only 1 GB of RAM and compiling ROS is expensive in memory usage. Configuring the system to use disk space as a swap disk is necessary and running the build single-threaded by using the `-j1` command line option is recommended. For swap disk use, it is advisable to use a mass storage device connected to the RPI's USB port instead of using the flash memory card. This is because, flash memories wear out. A flash memory *page* – a subdivision of a memory unit – can be erased 10 000 to 100 000 times, before it no longer stores data reliably [17].

The ROS system is organised in packages. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module [18]. At its core, a ROS package is a directory in the ROS file path containing a file called `package.xml`. These packages are bundled for release as debian packages.

For installation, there are a few default configurations of the different tools and libraries in ROS. These are called metapackages. For example, the recommended *Desktop-Full* metapackage contains the whole suite of graphic user interfaces (GUI), 2D/3D simulators, and packages for navigation and perception. For use with a headless[2] Raspberry Pi, the minimum configuration without any GUI tools is sufficient. This also saves resources and time during compilation. Additional packages can be installed individually if needed.

In the end, compiling ROS was not the primary problem. By carefully following the ROS tutorial (and the external references within) on compiling ROS from source, it is possible to build ROS on a

[2] A headless computer is a computer that is operated without the traditional monitor, mouse and keyboard peripherals. Headless computers and other remotely controlled hardware devices often operate using network control models [19].

Raspberry Pi – a quite time-consuming endeavour. The root problem of the compilation issues was isolated to the novatel-packages provided by the project's instructor. For communicating with the SPAN unit, the novatel-package was required. It has the serial, novatel, and gps_msgs –packages as prerequisites. These need to be installed in addition to the bare-bones installation.

While compiling the additional packages, other issues presented themselves with the compatibility of certain build-tools. Most notably the newer versions of libboost were incompatible and needed to be downgraded.

```
sudo apt-get install libboost1.58-dev
sudo apt-get install libboost1.58-tools-dev
```

The misleading part of the whole build process was, that by building the additional packages individually with the isolated build command

```
catkin_make --only-pkg-with-deps <target_package>
```

the process would always advance – only to fail in the last step.

Due to the nature of the problem and the less-than-informative catkin error messages, a clear, step-by-step guide for retracing the installing process is impossible to present. This is because many of the problems faced, depended on the path taken so far e.g. the individual attributes set on the commands. A concise version is, that the polaris_novatel package was not meant to be run on the RPi and removing that package resolved all issues. Ultimately, this was realised by comparing the code bases of the novatel and the polaris_novatel –packages. The standard novatel package is sufficient.

The Novatel package comes with a readme file that lists the packages dependencies. The readme has a spelling error in the gps-msgs URL. An easier approach is to download the packages using https.

```
wstool set serial --git https://github.com/wjwwood/serial.git
wstool set novatel --git https://github.com/GAVLab/novatel.git
wstool set gps_msgs --git https://github.com/GAVLab/gps-msgs.git
```

The gps_msgs package also needs sensor_msgs as a dependency. This is not bundled in the ros_base installation. It can be installed using rosdep.

The final hurdle was to connect the now working ROS node to the SPAN with USB. The issue was determined to be missing USB drivers. Novatel provides these as a download for Linux systems [10]. These did not work by simply installing and troubleshooting pointed towards missing kernel headers. One solution could have been downgrading the kernel on the RPi, but this was deemed too risky to be tried as a first solution. Unfortunately due to the pandemic, access to the SPAN was lost before this issue was resolved.

6 DGNSS Corrections from NLS

The National Land Survey of Finland (NLS) offers correction data to be applied on GNSS data before using it to calculate an actual position of the receiver. In this project, DGPS corrections in the NLS data are used to improve the accuracy of GPS information.

6.1 Fetch DGNSS Data

DGNSS corrections from NLS are fetched using APIs, which are free of charge. The differential corrections are sent over the internet according to NTRIP protocol (Networked Transport of RTCM via Internet Protocol). Also, the corrections are sent in two versions of RTCM standard, RTCM 2.2 and RTCM 3.2. In this project, the RTCM 2.2 format messages are used. The API is given below in Table 4, which has to be called with proper authentication credential as provided in the NLS's website [11].

Parameter	Value
Caster (IP)	195.156.69.177
Port	2102
Username	avoin
Password	data

Table 4: NLS API details

The API can be called as shown below. In this case, the corrections are fetched from MET_3 station.

```
wget --user 'avoin' --password 'data' "http↩
    ://195.156.69.177:2102/DG_MET3"
```

The response of the API is in ASCII format, which is stored in the file `DG_MET3`, which has to be converted to JSON format to identify the required data and to send to SPAN. This conversion is done using a Linux shell command `gpsdecode`, which converts and outputs the API response in JSON format. A sample code for fetching the API, converting it and storing to a file (`response.json`) is shown below.

```
wget --user 'avoin' --password 'data' "http↩
    ://195.156.69.177:2102/DG_MET3"
# stop it using Ctrl + X after sometime to stop recording
cat DG_MET3 | gpsdecode > response.json
```

The response contains three types of RTCM2 messages as follows. An example JSON is shown below.

1. Type 1 – Differential GPS corrections – 1Hz frequency
2. Type 31 – Differential GLONASS corrections – 1Hz frequency

3. Type 3 – Station parameter – 0.1Hz frequency. This field contains data about the station from which the corrections are fetched.


```
{
    "class": "RTCM2",
    "device": "stdin",
    "type": 31,
    "station_id": 108,
    "zcount": 3172.2,
    "seqnum": 0,
    "length": 19,
    "station_health": 6,
    "satellites": [
        {
            "ident": 23,
            "udre": 0,
            "change": false,
            "tod": 67,
            "prc": -5.220,
            "rrc": 0.020
        },
        ...
        {
            "ident": 12,
            "udre": 0,
            "change": false,
            "tod": 67,
            "prc": -78.360,
            "rrc": -0.160
        }
    ]
}
{
    "class": "RTCM2",
    "device": "stdin",
    "type": 1,
    "station_id": 108,
    "zcount": 3190.2,
    "seqnum": 1,
    "length": 22,
    "station_health": 6,
    "satellites": [
        {
            "ident": 18,
            "udre": 0,
            "iod": 16,
            "prc": -1.960,
            "rrc": -0.002
        },
        ...
        {
            "ident": 30,
            "udre": 0,
            "iod": 8,
            "prc": -28.980,
            "rrc": 0.020
        }
    ]
}
{
    "class": "RTCM2",
    "device": "stdin",
    "type": 3,
    "station_id": 108,
    "zcount": 3190.2,
    "seqnum": 2,
    "length": 4,
    "station_health": 6,
    "x": 2892584.51,
    "y": 1311799.21,
    "z": 5512619.56
}
```


These types can be identified by the key `type` and the value corresponds to the type number as described above. In this project, we are only using the DGPS corrections as SPAN gets the GNSS data from a GPS receiver.

The RTCM fields for type 1 messages contain the DGPS corrections as described in Table 5. [20] For naive DGNSS corrections the `ident` and `prc` can be used in the equation 3.

Field	Description
`ident`	Satellite identifier
`udre`	User differential range error
`iod`	Issue of data
`prc`	Pseudorange correction
`rrc`	Range-rate correction

Table 5: RTCM2 JSON format explained.

6.2 Extracting DGPS Data

In order to send the DGPS to SPAN, we have to filter DGPS data from the API response. Since, the RTCM data is converted to JSON format, the DGPS corrections can easily be filtered from the response in the ROS node running on the module. DGPS data in the response has a key `type` with value `1`, which can be used to identify and filter them from other messages. This filtered data can be sent to SPAN serially to correct the data from GPS receiver. Due to the pandemic and loss of access to SPAN, this could not be tested on hardware.

7 Change of Plans

In this chapter we aim to describe the effects of the COVID-19 pandemic on the project in chronological order and our backup plans going forward.

The global pandemic posed major challenges for the project. The Aalto University campus was closed down on week 12 of 2020 for an undetermined time period [21]. The uncertainty on the duration of the campus closure and light chaos during the switch to remote work, regrettably resulted in lost man-hours. After some time, it became clear that we had lost physical access to Polaris and the lab for the remainder of the project permanently. Thus the original plan could no longer be carried out. The RPi could still be worked on, but without access to Polaris and especially the SPAN, we would not be able to test anything in the original scope of the project.

7.1 Exploring the options

Without lab access, the focus of the project had to shift to alternative approaches. The core of the project (despite the title) is on correcting the Polaris' GPS position estimate by feeding the SPAN unit with the DGPS corrections from NLS, we tried to focus on this aspect of the project in isolation going forward. The obvious first option was to log GPS data through some device other than SPAN and post process it together with the DGPS corrections in Matlab or Python. As a secondary option we explored simulating the GPS satellites together with the Polaris rover.

Initial research into GPS logging showed that new Android versions have an API for accessing the raw data [22]. It soon proved to be a bigger endeavour, an entire project in its own right. Luckily, as it turns out, this is a point of interest for others as well and the app stores have programs that utilise this API. Section 7.2.1 provides a more detailed account of the logging attempts.

A previous project group [23] had worked on a Gazebo simulation for the Polaris ATV. Gazebo is a 3D simulation environment designed to be used with ROS. We briefly looked into using Gazebo instead of the real ATV. It quickly became obvious that this would not work as Gazebo is not able to provide the necessary advanced GPS simulation we would need. A simple longitude-latitude data could be obtained [24]. However, this would not be suitable for running the DGNSS correction process. Additionally, the benefit of having the simulated ATV would be minimal, as the model did not have the control system of the real ATV implemented. Implementing it by ourselves would not be possible in the available time frame.

Building the GPS simulation directly in Matlab was thus our last real option to follow, which did not have technical blockers. Additionally, it would essentially fit the work package 7 of our original project plan described in Tables 2,3. To simulate GPS, we used simplified satellite trajectories and a simplified rover model with waypoint navigation and tracked states through the GPS measurements. A more thorough explanation of this follows in Section 8.

7.2 Logging Raw GPS data and DGPS Corrections

In order to simulate the whole process of correcting GPS data with DGPS corrections, both GPS and DGPS correction data have to be logged simultaneously. This is because the interference in the satellite signals varies temporally. The first task was to find a way to log GPS data. As mentioned

above, there are a number of applications for this purpose available for Android phones. DGPS data from NLS was directly fetched using the NLS API and converted to JSON format. For the purpose of logging the GPS data, multiple programs were tested in addition to a USB connected GPS receiver, but as to be explained, these efforts did not lead to a successful outcome.

7.2.1 Logging GPS data

Between the numerous applications for GPS logging, only a few file formats are supported. The most obvious one to use was the RINEX-format. RINEX stands for Receiver Independent Exchange Format [25] and it is a ASCII-based data interchange format developed for the very purpose of enabling post-processing the data to produce more accurate positioning. The application used for logging data in the RINEX format was Geo++ RINEX Logger, produced by the same German company that provides the calibration software for NLS [26, 27]. This seemed like a promising approach.

Simultaneous logging of the GPS data and the NLS corrections took some organising as they were done in separate locales and several data-sets were logged. Two straight-line data-sets with minimum obstructions were taken on the Lauttasaari bridge. Another 10 minute, stationary data-set was logged on the pier next to the bridge and a third, longer, and partially obstructed data-set was logged along a path through Ruoholahti to Kamppi. The logging was done on a Nokia 9 Android phone.

Below is a sample of what the logged data looks like. It starts with a header section containing general data about the RINEX file type and version, used device and its GPS antenna, etc. before the actual data log. The logged data is divided into *epochs* marked by the log time and followed by the satellite data. The example includes the RINEX header and two epochs, first one with 17 satellites and the second one with 15 satellites.

```
     3.03            OBSERVATION DATA    M: Mixed            RINEX VERSION / TYPE
Geo++ RINEX Logger   Geo++               20200417 061947 UTC PGM / RUN BY / DATE
************************************************************COMMENT
This file was generated by the Geo++ RINEX Logger App        COMMENT
for Android devices (Version 2.1.6). If you encounter        COMMENT
any issues, please send an email to android@geopp.de         COMMENT
Filtering Mode: BEST                                         COMMENT
************************************************************COMMENT
Geo++                                                        MARKER NAME
ANIMAL                                                       MARKER TYPE
Geo++                Geo++                                   OBSERVER / AGENCY
unknown              HMD Global          Nokia 9             REC # / TYPE / VERS
unknown              Nokia 9                                 ANT # / TYPE
  2885677.5299  1339533.9439  5509529.1038                   APPROX POSITION XYZ
        0.0000        0.0000        0.0000                   ANTENNA: DELTA H/E/N
G    8 C1C L1C D1C S1C C5Q L5Q D5Q S5Q                       SYS / # / OBS TYPES
R    4 C1C L1C D1C S1C                                       SYS / # / OBS TYPES
E   12 C1B L1B D1B S1B C1C L1C D1C S1C C5Q L5Q D5Q S5Q       SYS / # / OBS TYPES
C    4 C2I L2I D2I S2I                                       SYS / # / OBS TYPES
J    8 C1C L1C D1C S1C C5Q L5Q D5Q S5Q                       SYS / # / OBS TYPES
  2020     4    17     6    20     5.7868371     GPS         TIME OF FIRST OBS
 24 R01  1 R02 -4 R03  5 R04  6 R05  1 R06 -4 R07  5 R08  6 GLONASS SLOT / FRQ #
    R09 -2 R10 -5 R11  0 R12 -1 R13 -2 R14 -7 R15  0 R16 -1 GLONASS SLOT / FRQ #
    R17  4 R18 -3 R19  3 R20  2 R21  4 R22 -3 R23  3 R24  2 GLONASS SLOT / FRQ #
G L1C                                                        SYS / PHASE SHIFT
```

```
26 G L5Q -0.25000                                                SYS / PHASE SHIFT
27 R L1C                                                         SYS / PHASE SHIFT
28 E L1B                                                         SYS / PHASE SHIFT
29 E L1C +0.50000                                                SYS / PHASE SHIFT
30 E L5Q -0.25000                                                SYS / PHASE SHIFT
31 C L2I                                                         SYS / PHASE SHIFT
32 J L1C                                                         SYS / PHASE SHIFT
33 J L5Q -0.25000                                                SYS / PHASE SHIFT
34  C1C     0.000 C1P     0.000 C2C     0.000 C2P     0.000      GLONASS COD/PHS/BIS
35                                                               END OF HEADER
36 > 2020  4 17  6 20  5.7868371  3  1
37 Geo++                                                         MARKER NAME
38 > 2020  4 17  6 20  5.7868371  0 17
39 E01  24327403.113                    1735.150          32.100
40 E04  23988988.992                    -335.150          29.200
41 E09  24280601.313                    2367.250          29.300
42 E21  24062111.371                   -1170.650          35.100
43 E27  27575380.385                   -2974.047          28.100
44 E31  28192822.439                    3127.300          34.800
45 G05  24488731.928                   -3685.623          35.500
46 G07  23606282.335                   -3150.500          22.800
47 G08  23735556.440                   -1905.365          25.000
48 G13  20349383.239                      95.095          32.900
49 G15  21321838.023                    1679.826          34.700
50 G17  25201314.917                    3533.706          40.000
51 G20  23824215.862                    1929.151          31.400
52 G21  26213370.785                   -2312.826          31.700
53 G28  20874389.685                     898.027          38.800
54 G30  21237224.299                   -2050.328          36.200
55 R14  19384683.487                   -1320.276          34.300
56 > 2020  4 17  6 20  6.6635446  0 15
57 E01  24326990.161                    1742.900          36.500
58 E04  23988900.116                    -328.650          36.600
59 E09  24280072.342                    2374.150          30.200
60 E21  24062184.382                   -1161.500          32.000
61 E31  28192181.345                    3135.750          35.500
62 G05  24489218.053                   -3677.329          36.000
63 G11  24390054.203                    1947.650          29.200
64 G13  20349234.104                     102.670          37.600
65 G15  21321423.571                    1686.303          30.500
66 G17  25200593.778                    3541.148          42.200
67 G20  23823755.243                    1936.355          36.300
68 G21  26213626.070                   -2303.872          29.400
69 G28  20874115.237                     904.779          39.200
70 G30  21237436.714                   -2045.907          30.500
71 R14  19384797.870                   -1313.919          31.500
```

The problem with the logged data was, that it did not include the satellite orbit data necessary to solve the pseudorange equation 2. Another phone was tested as well, to see if the format would differ between makes, which was not the case. Further detail and explanation of the the data format is presented in section 7.3.

Other possible formats for logging raw GPS data are NMEA, KML, GPX, and CSV. NMEA (National Marine Electronics Association) is an US-based organisation working on interface standards among other things [29]. Their standard includes 76 standard sentences and a number of vendor specific sentences. The available sentences range from the GPS-related ones to meteorological data and water depth [30].

For logging the data we used Ultra GPS Logger Lite which is the light version of a very diverse GPS logging application. It supports logging raw NMEA sentences, which is supposed to be more accurate than other options that log data only once a minute [31]. Similar data-sets were logged for NMEA as for RINEX and the device used was the same Nokia 9. Out all the possible sentences, 4 types were logged. These were GSA, Overall Satellite data; GSV, Detailed Satellite data; RMC, recommended minimum data for GPS; and GGA, Fix information. Two consequtive sets of data in a logged NMEA file is presented below.

```
$GPGSA,A,3,4,5,9,12,,18,,25,26,29,31,,,,,*08
$GPGSV,3,1,20,4,11,327,36.8,5,30,80,30.7,9,10,1,27,12,5,133,24.6*59
$GPGSV,3,2,20,16,10,302,22.9,18,35,191,27.7,23,0,0,34,25,32,146,30.4*5a
$GPGSV,3,3,20,26,40,292,37.3,29,78,108,26.4,31,36,244,0,41,0,0,27.9*6a
$GPRMC,125432,A,6009.6888,N,02453.6275,E,0,,040520,,,*06
$GPGGA,125432,6009.6888,N,02453.6275,E,1,22,2.4790802,25.168,M,,,,*2f
$GPGSA,A,3,2,4,5,9,18,21,,26,31,68,69,75,,,,*35
$GPGSV,3,1,17,2,18,50,30.6,4,11,327,38.6,5,30,80,26.6,9,10,1,28.9*7f
$GPGSV,3,2,17,18,35,191,25.3,21,34,198,0,23,0,0,34,26,40,292,40.9*7a
$GPGSV,3,3,17,31,36,244,26.2,68,21,4,37.3,69,21,59,33.7,75,42,187,26.9*42
$GPRMC,125433,A,6009.6888,N,02453.6275,E,0,,040520,,,*07
$GPGGA,125433,6009.6888,N,02453.6275,E,1,23,2.4790802,25.169,M,,,,*2e
```

Again, the logged data does not include all the necessary information (see section 7.3).

A USB GPS receiver (GlobalSat G-STAR IV) is also used to log GPS data. It gives data in NMEA format and gave a similar data as above. It also does not provide all the necessary information for further processing.

7.2.2 Logging DGPS corrections

DGPS corrections are recorded by calling NLS APIs and storing the response at the same time when GPS data is logged. This data is converted to human readable form – in order to process in MATLAB – using `gpsdecode`, a Linux shell command to convert various types of GNSS data. Since timestamp of each data is needed for the simulation (which is not already available in the fetched data), it is added to each sample based on the time when recording is started. A simple Python script is used to insert timestamps.

7.3 Processing logged GPS data

The thought process for processing the the logged GPS data was to build a model of our rover, a person walking in this case;and form a "true" trajectory for the rover based on the known route walked (or stayed stationary) on the map. The rover could be modeled as a constant speed process with added noise – much like how we ended up simulating the rover in Matlab later on in equations 10,11. The logged GPS data could then be viewed as the measurements for this process, which could be corrected with DGPS. The state estimation would have been done with an Extended Kalman Filter – again much like the simulation approach. Post-processing the data could be done in either Matlab or Python as those were the languages the project group was familiar with.

As described earlier in equation 3, the DGPS data is a correction Δp_i to the pseudorange p_i between

the receiver and the satellite. From the corrected pseudorange, the receivers x, y, z position can be calculated more accurately as the remaining error term has smaller covariance. The correction only affects the pseudorange, thus the satellites exact position must also be known to solve for the rovers corrected position x', y', z', even when the uncorrected position x, y, z is known through the logged GPS data itself. This fact posed hard limits on the required data, which the different logging approaches lacked. Unfortunately neither of the formats could be used. This is described in more detail in the following sections.

7.3.1 RINEX – Receiver Independent Exchange Format

As described and shown in 7.2.1, the RINEX format we managed to produce with the Geo++ app seemed different from ones we explored online [25, 26]. An interactive explanation of the RINEX format is found on the Technical University of Catalonias Research Group of Astronomy and Geomatics' website https://gage.upc.edu/sites/default/files/gLAB/HTML/Observation_Rinex_v3.01.html.

Each epoch line is missing the receiver clock offset value. The measurements lack the carrier phase entry for each satellite. The first data column after the epoch identifies the satellite and which constellation it belongs to (G for GPS, R for GLONASS, S for SBAS and E for Galileo). The second column looks to correlate with the pseudorange value – at least it is of the right magnitude. We believe the third column describes the doppler value for satellite, however we could not verify this. The last column describes the signal strength.

In summary, the RINEX format does not include satellite orbit information and as such it is not possible to solve for the receivers position from the pseudorange equation 2. This was the reason the RINEX format was deemed unreasonable to work with and the exploration focused on NMEA formats next. It may have been possible to fit known satellite orbits from another source to the RINEX data, but even a slight mismatch in timestamps could lead to inaccurate readings.

7.3.2 NMEA – National Marine Electronics Association

When the RINEX format failed, our secondary approach were the NMEA sentences logged with the Ultra GPS Logger Lite app. The NMEA sentences seemed better than the RINEX format as there was more easily understandable data. The sentences available for logging were GSA, GSV, RMC and GGA. An explanation of the format and contents is as follows [30, 32]:

The GSA sentence holds data for the fix on satellites and the dilution of precision – essentially describing the covariance of the solved rover x, y, z position.

```
1   $GPGSA,A,3,04,05,,09,12,,,24,,,,,2.5,1.3,2.1*39
2
3  Where:
4       GSA      Satellite status
5       A        Auto selection of 2D or 3D fix (M = manual)
6       3        3D fix - values include: 1 = no fix
7                                        2 = 2D fix
8                                        3 = 3D fix
9       04,05... PRNs of satellites used for fix (space for 12)
10      2.5      PDOP (dilution of precision)
11      1.3      Horizontal dilution of precision (HDOP)
```

```
12      2.1       Vertical dilution of precision (VDOP)
13      *39       the checksum data, always begins with *
```

The GSV sentence holds data about the satellite orbits, their current elevation and azimuth which can be used to solve for the X_i, Y_i, Z_i for each satellite i once the trajectory for that satellite is known.

```
1   $GPGSV,2,1,08,01,40,083,46,02,17,308,41,12,07,344,39,14,22,228,45*75
2
3   Where:
4        GSV          Satellites in view
5        2            Number of sentences for full data
6        1            sentence 1 of 2
7        08           Number of satellites in view
8
9        01           Satellite PRN number
10       40           Elevation, degrees
11       083          Azimuth, degrees
12       46           SNR - higher is better
13            for up to 4 satellites per sentence
14       *75          the checksum data, always begins with *
```

The RMC sentence holds data for the receivers current position and speed – the so called Recommended Minimum Navigation Information.

```
1   $GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A
2
3   Where:
4        RMC          Recommended Minimum sentence C
5        123519       Fix taken at 12:35:19 UTC
6        A            Status A=active or V=Void.
7        4807.038,N   Latitude 48 deg 07.038' N
8        01131.000,E  Longitude 11 deg 31.000' E
9        022.4        Speed over the ground in knots
10       084.4        Track angle in degrees True
11       230394       Date - 23rd of March 1994
12       003.1,W      Magnetic Variation
13       *6A          The checksum data, always begins with *
```

The GGA sentence holds data for the GPS receiver and information on whether some corrections are already used.

```
1   $GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47
2
3   Where:
4        GGA          Global Positioning System Fix Data
5        123519       Fix taken at 12:35:19 UTC
6        4807.038,N   Latitude 48 deg 07.038' N
7        01131.000,E  Longitude 11 deg 31.000' E
8        1            Fix quality: 0 = invalid
9                                  1 = GPS fix (SPS)
10                                 2 = DGPS fix
11                                 3 = PPS fix
12                    4 = Real Time Kinematic
```

```
                    5 = Float RTK
                              6 = estimated (dead reckoning) (2.3 feature)
                    7 = Manual input mode
                    8 = Simulation mode
      08            Number of satellites being tracked
      0.9           Horizontal dilution of position
      545.4,M       Altitude, Meters, above mean sea level
      46.9,M        Height of geoid (mean sea level) above WGS84
                        ellipsoid
      (empty field) time in seconds since last DGPS update
      (empty field) DGPS station ID number
      *47           the checksum data, always begins with *
```

Unfortunately, the GSA, GSV, RMC and GGA sentences are not enough to solve the pseudorange equation 3. The GSV holds important information about the satellite orbits, but the elevation and azimuth are only two of the Keplerian orbital elements as described in Section 1.1.1, all of which are time-varying. The remaining orbital elements seem to be included in the GPS Almanac data included in the NMEA ALM message [30, 32].

```
  $GPALM,1,1,15,1159,00,441d,4e,16be,fd5e,a10c9f,4a2da4,686e81,58cbe1,0a4,001,*5B

Where:
      ALM           Orbital data for a specified GPS satellite
      1             Total number of messages
      1             Sentence number
      15            Satellite PRN number
      1159          GPS Week number
      00            SV health, bits 1724 of each almanac page
      441d          Eccentricity
      4e            Almanac reference time
      16be          Inclination angle
      fd5e          Rate of right ascension
      a10c9f        Root of semi-major axis
      4a2da4        Argument of perigee
      686e81        Longitude of ascension node
      58cbe1        Mean anomaly
      0a4           F0 clock parameter
      001           F1 clock parameter
      *5B           the checksum data, always begins with *
```

The most important piece of data is missing for applying the corrections: the pseudorange p_i to each satellite. With the data included in our longs and with the GPS almanac it should be theoretically possible to solve backwards for p_i. Without knowing the algorithms used for this specific GPS logger app, it was not further explored as an option.

8 Simulating GPS

This section describes the main results of our project, the simulation of GPS satellites, the positioning of, and the state estimation of the rover's GPS receiver. The majority of the simulation is based on a book by Yaakov Bar-Shalom et al. [4, pp. 501–533]. Some additional information was gathered from a conference paper by Mark Wickert et al. [33] and a research paper by Ashok Kumar et al. [5].

8.1 Satellite Trajectories

The satellite trajectories were simplified to perfectly circular orbits. The shape of the orbit is not of as great importance to the GPS measurements, as is having access to the satellites coordinates. To coordinate systems were looked for circular orbits, one for the WGS84 (World Geodetic System) coordinate system and the other for ENU (East North Up) coordinates [4, pp. 516]. The two sets of equations are:

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}_{\text{WGS84}} = \begin{bmatrix} R\left[\cos\omega(t)\cos\Omega(t) - \sin\omega(t)\sin\Omega(t)\cos\alpha\right] \\ R\left[\cos\omega(t)\cos\Omega(t) + \sin\omega(t)\sin\Omega(t)\cos\alpha\right] \\ R\sin\omega(t)\sin\alpha \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}_{\text{NAV}} = \begin{bmatrix} R\left[\cos\omega(t)\sin\Omega(t) + \sin\omega(t)\cos\Omega(t)\cos\alpha\right] \\ R\sin\omega(t)\sin\alpha \\ R\left[\cos\omega(t)\cos\Omega(t) - \sin\omega(t)\sin\Omega(t)\cos\alpha\right] - r_{Earth} \end{bmatrix} \tag{5}$$

The terms above are directly related to the Keplerian orbital elements seen in Figure 2 and discussed at the start of this report, ω is the angle the satellite makes relative to the ascending node; Ω is the angle of the ascending node with respect to the reference point in the reference plane; α is the inclination of the orbital plane and R is the radius of the orbit, covering for the elliptic parameters. The ω_0 and Ω_0 terms differentiate the satellites at a given time t_0, in essence these are the initial conditions.

$$\omega(t) = \omega_0 + \frac{360}{43082}(t - t_0)^\circ \tag{6}$$

$$\Omega(t) = \Omega_0 - \frac{360}{86164}(t - t_0)^\circ \tag{7}$$

$$\alpha = 55^\circ \tag{8}$$

$$R = 26560\ \text{km} \tag{9}$$

The simulation is done in Matlab for 6 satellites with the initial values as seen in Table 6 [4, pp. 517]. In code, the initial conditions were abstracted into the function `sat\_idx`.

Calculating the orbits in code is done by calling the `sat\_pos` function at any time instant $t = k$. To guarantee all satellites have line of sight to the rover at ENU $(0, 0, 0)$, $t_0 = -3000$ must be set. Generating the trajectories can be done in a simple loop calling `sat\_pos` with an increasing timestamp. An example plot of satellite trajectories is given in Figure 3.

Index	$\Omega_0(°)$	$\omega_0(°)$
1	325.7	72.1
2	25.7	343.9
3	85.7	214.9
4	145.7	211.9
5	205.7	93.9
6	265.7	27.9

Table 6: Satellite initial positions used in the simulation.

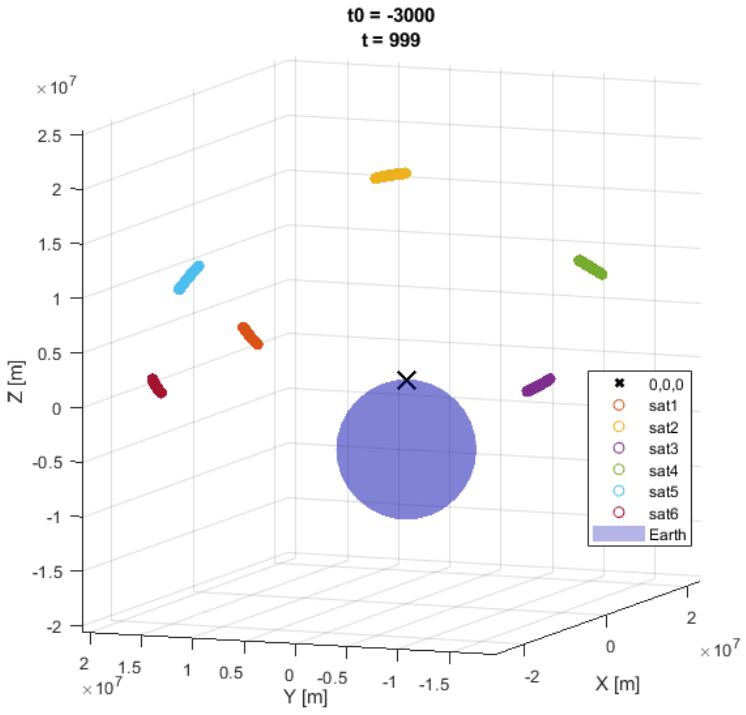

Figure 3: Simulated satellite trajectories around the Earth.

8.2 Rover

It was immediately apparent that we would not be able to simulate the entire Polaris ATV. To keep things simple, we decided to create the minimal rover model such that it would still be relatively simple to expand on if need be. The rovers dynamic model is a simple LTI system that evolves according to the equations 10,11. An example rover trajectory can be seen in Figure 4, it is modelled by and uses the control strategy defined in the below equations and algorithms.

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{10}$$

$$x_k = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{11}$$

For the control law equation 12 we decided to implement a simple LQR (Linear Quadratic Regulator) with reference (waypoint) following. LQ control is an optimal control strategy for linear systems. It produces the control matrix K_{ctrl} that minimizes the cost function 13. The control matrix can be calculated from equation 14 where P is the solution to the discrete time algebraic Riccati equation 15. The matrices C_Q, C_R and N are the cost weight matrices for the state, input and cross cost respectively [34]. The input cost was weighed more harshly to artificially slow down the rover, as shown in equation13.

$$x_{k+1} = Ax_k + B(-K_{ctrl}\hat{x}_k - K_{ref}y_{ref}), \quad \hat{x}_k = Ix_k \tag{12}$$

$$J = \sum_{k=0}^{\infty}(x_k^T C_Q x_k + u_k^T C_R u_K + 2x_k^T C_N u_k) \tag{13}$$

$$K_{ctrl} = (C_R + B^T P B)^{-1}(B^T P A + N^T) \tag{14}$$

$$P = A^T P A + C_Q - (A^T P B + N)(R + B^T P B)^{-1}(B^T P A + N^T) \tag{15}$$

$$C_Q = I^{4\times 4}, \quad C_R = 100 \cdot I^{2\times 2}, \quad N = 0 \tag{16}$$

For the state estimator in equation 12 on the rover simulation, we simply used the true state values. The reasoning was similar to the perfectly spherical satellite trajectories, i.e. the rover itself was not an essential part of the simulation.

Reference following is achieved with the set points y_{ref} and the K_{ref} matrix solved from equation 17. The waypoint following was implemented naively such that if the rover is within a set distance d from the waypoint, a new waypoint will be given to the controller as shown in equation 18.

$$K_{ref} = (C(I - A - BK_{ctrl})^{-1}B)^{-1} \tag{17}$$

$$y_{ref} = \begin{cases} \text{wp}_{i+1} & \text{if } d \leq r \\ \text{wp}_i & \text{otherwise} \end{cases}, \quad d = \sqrt{(x_{rover} - x_{ref})^2 + (y_{rover} - y_{ref})^2} \tag{18}$$

As the waypoints can be arbitrarily far way with this strategy, it was important to clamp the controller input u_k with equation 19. This was done in order to make it behave at least slightly more

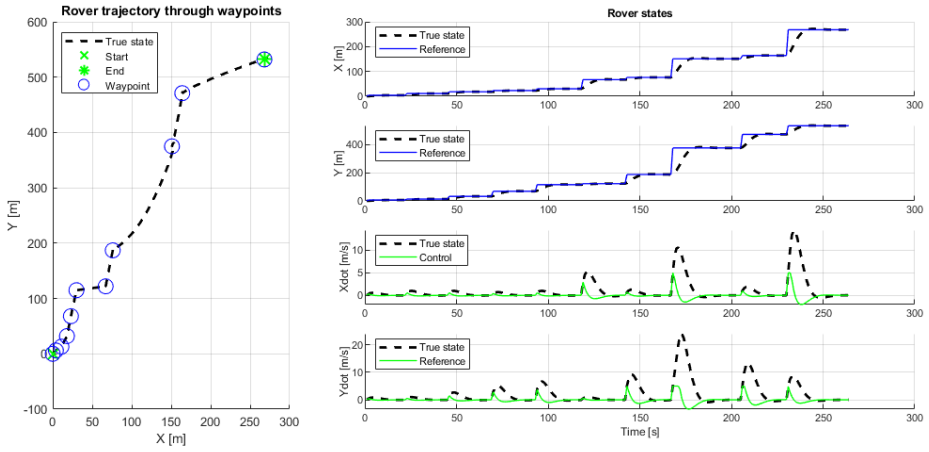

Figure 4: Rover trajectory using the LQR controller.

realistically and reduce the maximum speed of the rover. In hindsight, this could have been achieved better with an MPC (Model Predictive Control) strategy with constraints on the velocity states or by introducing some friction or other dynamics to the model. The velocity problem can be seen in Figure 4 the rover has a very high speed as a response to a large jump in the waypoint distance. Regardless, the rover itself was not the focus of the simulation.

$$u_k = \text{sign}(u_k) \cdot \max(\text{abs}(-K_{ctrl}\hat{x}_k - K_{ref}y_{ref}), u_{MAX}) \tag{19}$$

8.3 Ranging

The pseudorange equation 2 was used as the measurement equation for this system. For the parameters X_i, Y_i, Z_i and x, y, z, the satellites and rovers true coordinates were used in the simulation, additional noise was added with the w_i term. However, simulating the clock bias proved very difficult, as this would imply simulating the underlying quartz crystal or atomic clock, their biases and drifts – and further correcting for the errors between individual satellites and the rover between simulation steps. We decided to omit the clock bias, as we would not have been able to use it for the postprocessing approach anyway – recorded data has clock bias locked at the point of recording, indicated by the timestamp of data – the other option would have been to set it to 0, which in essence does the same.

As there can be more than four satellites visible to the receiver at any given time, the measurement error can be reduced by e.g. the Least Squares (LS) method. By first linearizing the measurement

equation, the measurements z_i relate to the receivers state x by:

$$\left.\begin{aligned} z_1 &= H_1 x + w_1 \\ z_2 &= H_2 x + w_1 \\ &\vdots \\ z_n &= H_n x + w_n \end{aligned}\right\} \rightarrow z = Hx + w \tag{20}$$

such that estimating the state x can be done through LS when the covariance P_w of the measurements w is known:

$$\hat{x} = [H^T P_w^{-1} H]^{-1} H^T P_w^{-1} z \tag{21}$$

$$P_x = [H^T P_w^{-1} H]^{-1} \tag{22}$$

For our simulation, each satellite has equal covariance and thus the LS estimate of a stationary receiver is:

$$\hat{x} = [H^T H]^{-1} H^T z \tag{23}$$

$$P_x = [H^T H]^{-1} \sigma^2 \tag{24}$$

Line of sight measurements, such as the GPS ranging measurement, are subject to dilution of precision errors which raise from the very small angles between the satellite and the rover, i.e. a satellite directly overhead will have a larger error for the north and east coordinates than up. Modeling the measurement covariance σ^2 is done by the dilution of precision DOP. DOP is modelled as four separate elements of Gaussian white noise: horizontal (east and north), vertical and time. [4, pp. 507-510] These form the geometric dilution of precision GDOP matrix:

$$[H^T H]^{-1} \triangleq \begin{bmatrix} E_{\text{DOP}}^2 & & & \\ & N_{\text{DOP}}^2 & & \\ & & V_{\text{DOP}}^2 & \\ & & & T_{\text{DOP}}^2 \end{bmatrix} \tag{25}$$

The positional measurement error is then given by (and scaled by σ):

$$P_{\text{DOP}} = \sqrt{E_{\text{DOP}}^2 + N_{\text{DOP}}^2 + V_{\text{DOP}}^2} \tag{26}$$

Typical accuracy values for the horizontal, vertical and timing errors as shown on Table 7. In our simulation, we used these as the baseline for the added white noise w_i to GPS measurements, combined in equation 27. In code, the noise is simply added to the true range by generating Gaussian noise and scaling it by the σ in Matlab with `sigma_pseudo*randn(1)`.

$$w_i \sim \mathcal{N}(0, \sigma^2), \quad \sigma = \sqrt{\left(17.8\frac{68\,\%}{98\,\%}\right)^2 + \left(17.8\frac{68\,\%}{98\,\%}\right)^2 + \left(\frac{27.7}{2}\right)^2} \tag{27}$$

Type	Value	Scale
Horizontal	17.8 m	2drms, 98 %
Vertical	27.7 m	2σ, 95 %
Timing	90 ns	Maximum

Table 7: GPS positioning accuracies [4, pp. 590].

8.4 Extended Kalman Filter

The Kalman Filter (KF) is the optimal state estimation algorithm for linear systems with Gaussian noises. The Extended Kalman Filter (EKF), by its very name extends the Kalman Filter to non-linear systems, e.g. the system in equation 28. The extension is done by linearizing the non-linear dynamics or measurement equation by Taylor series expansion, such that the Kalman Filtering algorithm can be used. [4, pp. 208,386]

In our simulation, the EKF constructed to track the rover. This is done through a five state linear system, with a non-linear measurement equation. The state consists of x, y, z coordinates and the $\dot{x}, \dot{y}$ velocities in order $x_k = \begin{bmatrix} x & \dot{x} & y & \dot{y} & z \end{bmatrix}^T$. We decided to not include the clock bias and drift $b, \dot{b}$ as explained earlier, and also the $\dot{z}$ velocity as the change in altitude of the rover was considered unimportant for the simulations sake. The system model was thus as below in equation 28, where $h(\cdot)$ is the true range equation 1 for n satellites.

$$\begin{aligned} x_{k+1} &= Fx_k + q_k, \quad q_k \sim \mathcal{N}(0, Q) \\ y_k &= h(x_k) + w_k, \quad w_k \sim \mathcal{N}(0, W) \\ F &= \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{28}$$

Linearizing the pseudorange equation 2 with respect to each variable results in the measurement Jacobian:

$$h_i(t) = \sqrt{(X_i(t) - x(t))^2 + (Y_i(t) - y(t))^2 + (Z_i(t) - z(t))^2} + cb(t) + w_i(t) \tag{29}$$

$$H_i(t) = \begin{bmatrix} \frac{\partial h_i(t)}{\partial x} & \frac{\partial h_i(t)}{\partial y} & \frac{\partial h_i(t)}{\partial z} & \frac{\partial h_i(t)}{\partial b} \end{bmatrix} = \begin{bmatrix} h_i^x(t) & h_i^y(t) & h_i^z(t) & h_i^b(t) \end{bmatrix} \tag{30}$$

$$h_i^x(t) = \frac{-(X_i(t) - x(t))}{\sqrt{((X_i(t) - x(t))^2 + (Y_i(t) - y(t))^2 + (Z_i(t) - z(t))^2)}} \tag{31}$$

$$h_i^y(t) = \frac{-(Y_i(t) - y(t))}{\sqrt{((X_i(t) - x(t))^2 + (Y_i(t) - y(t))^2 + (Z_i(t) - z(t))^2)}} \tag{32}$$

$$h_i^z(t) = \frac{-(Z_i(t) - z(t))}{\sqrt{((X_i(t) - x(t))^2 + (Y_i(t) - y(t))^2 + (Z_i(t) - z(t))^2)}} \tag{33}$$

$$h_i^b(t) = c \tag{34}$$

where each satellite i gets a row in the measurement matrix for n satellites (for a stationary receiver):

$$H(t) = \begin{bmatrix} H_1(t) \\ H_2(t) \\ \vdots \\ H_n(t) \end{bmatrix} = \begin{bmatrix} h_1^x(t) & h_1^y(t) & h_1^z(t) & h_1^b(t) \\ h_2^x(t) & h_2^y(t) & h_2^z(t) & h_2^b(t) \\ \vdots & \vdots & \vdots & \vdots \\ h_n^x(t) & h_n^y(t) & h_n^z(t) & h_n^b(t) \end{bmatrix} \tag{35}$$

Dropping the clock bias b term from above gives the equivelant linearization for the true range equation 1 and further fitting for the state of our tracked system equation 28 gives:

$$H(t) = \begin{bmatrix} h_1^x(t) & 0 & h_1^y(t) & 0 & h_1^z(t) \\ h_2^x(t) & 0 & h_2^y(t) & 0 & h_2^z(t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ h_n^x(t) & 0 & h_n^y(t) & 0 & h_n^z(t) \end{bmatrix} \tag{36}$$

In our simulation code, the measurement Jacobian and the measurement equations were implemented as lambda-functions that accept satellite and rover structures as argument:

```
% Measurement model(s)
% satellite struct:
%   sat.x, sat.y, sat.z = satellite x,y,z coordinates
%   sat.t = satellite time
% Rover struct:
%   rov.x, rov.y, rov.z = rover x,y,z corodinates
%   rov.b = rover clock bias.
h_range  = @(sat,rov) sqrt((sat.x - rov.x)^2 + (sat.y - rov.y)^2+(sat.z - rov.z)^2);
h_pseudo = @(sat,rov) h_range(sat,rov) + c*rov.b;
```

```
dhdx = @(sat,rov) (sat.x - rov.x)/h_range(sat,rov);
dhdy = @(sat,rov) (sat.y - rov.y)/h_range(sat,rov);
dhdz = @(sat,rov) (sat.z - rov.z)/h_range(sat,rov);
% each row of the full jacobian follows:
H_row    = @(sat,rov) [-dhdx(sat,rov), 0, -dhdy(sat,rov), 0, -dhdz(sat,rov)];
H_row_clk = @(sat,rov) [-dhdx(sat,rov), 0, -dhdy(sat,rov), 0, -dhdz(sat,rov), c, 0];
% position part of H_row only
H_row_p = @(sat,rov) [-dhdx(sat,rov),-dhdy(sat,rov),-dhdz(sat,rov)];
```

EKF expects there to be noise in both the measurements and the dynamical model. There is no noise added to the rover's dynamical model in the simulation,but the changes in to waypoints and control signal u_k can be thought of as unaccounted dynamics and thus represented by the covariance matrix Q. This is the main design parameter for the tracker EKF for following the rover. Bar-Shalom gives an example covariance matrix [4, pp. 533] for an unknown system similar to ours, which we used as the basis for our simulation, but tweaking the design parameter values S_w, S_z.

$$Q = \begin{bmatrix} S_x\frac{\Delta t^3}{3} & S_x\frac{\Delta t^2}{2} & 0 & 0 & 0 \\ S_x\frac{\Delta t^2}{2} & S_x\Delta t & 0 & 0 & 0 \\ 0 & 0 & S_y\frac{\Delta t^3}{3} & S_y\frac{\Delta t^2}{2} & 0 \\ 0 & 0 & S_y\frac{\Delta t^2}{2} & S_y\Delta t & 0 \\ 0 & 0 & 0 & 0 & S_z\Delta t \end{bmatrix} \tag{37}$$

$$S_x = S_y = S_w = 0.01m^2/s^3, \quad S_z = 0.01m^2/s^3 \tag{38}$$

Thus the full Extended Kalman Filter tracking system is given by merging the above to:

True rover:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= h(x_k) + w_i, \quad w_i \sim \mathcal{N}(0, \sigma^2) \end{aligned} \tag{39}$$

EKF predict:

$$\begin{aligned} \hat{x}_k^- &= Fx_{k-1} \\ P_k^- &= FP_{k-1}F^T + Q \end{aligned} \tag{40}$$

EKF update:

$$\begin{aligned} H_x &= H(t)\Big|_{x,y,z=\hat{x}_k^-,\hat{y}_k^-,\hat{z}_k^-} && \textit{(Measurement Jacobian at prediction)} \\ S_k &= H_x P_k^- H_x^T + \sigma^2 I^{n\times n} && \textit{(Innovation covariance)} \\ K_k &= P_k^- H_x^T S_k^{-1} && \textit{(Kalman gain)} \\ \hat{x}_k &= \hat{x}_k^- + K_k(y_k - h(\hat{x}_k^-)) && \textit{(Update state prediction with measurement)} \\ P_k &= P_k^- - K_k S_k K_k^T && \textit{(Update covariance)} \end{aligned} \tag{41}$$

For the simulation, implementing the EKF in code is straight forward and works exactly as the above describes, with $\hat{x}_0$ and P_0 having sensible values. We started with the exact state $\hat{x}_0 = x_0$ and arbitrary $P_0 = 0.001I^{5\times5}$.

8.5 Results

The simulation that we have built is a simplified look into rover state estimation over GPS tracking. An example of the state estimation is shown in Figure 5. The estimation is not perfect to say the least, the velocity states depend entirely on the filter performance, as there is no additional information in the measurements for these. Merging with e.g. Inertial Measurement Unit (IMU) data for the rover would yield better results – and this is what the SPAN module on Polaris is essentially doing. Additionally, information about the input would shift the velocity states, as seen in Figure 6, and in the process reduces the state prediction covariance for the velocity states.

The resulting code is easily modifiable to change the underlying dynamic system of the rover or even the EKF tracking model.

8.6 Future Work

There are some very obvious simplifications in our simulation that should be worked on to improve the simulation accuracy and realism. However, as this was a backup of a backup plan for our project due to the COVID-19 situation, there is little motivation to focus on these over the initial plan of implementing DGNSS on the Polaris ATV. Regardless, in no specific order, the items that would make our simulation better are:

1. Satellite trajectories
   (a) Implement elliptical orbits instead of circular ones.

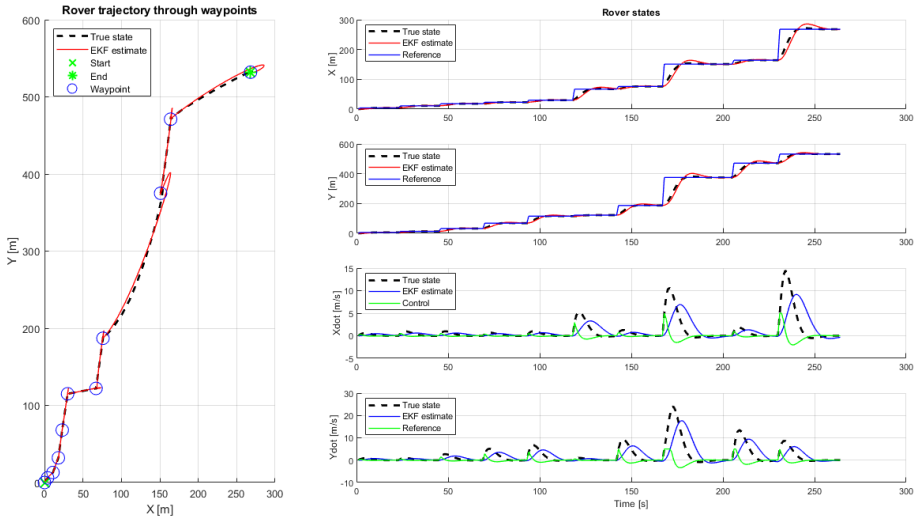

Figure 5: EKF estimate of rover trajectory using only GPS measurements.

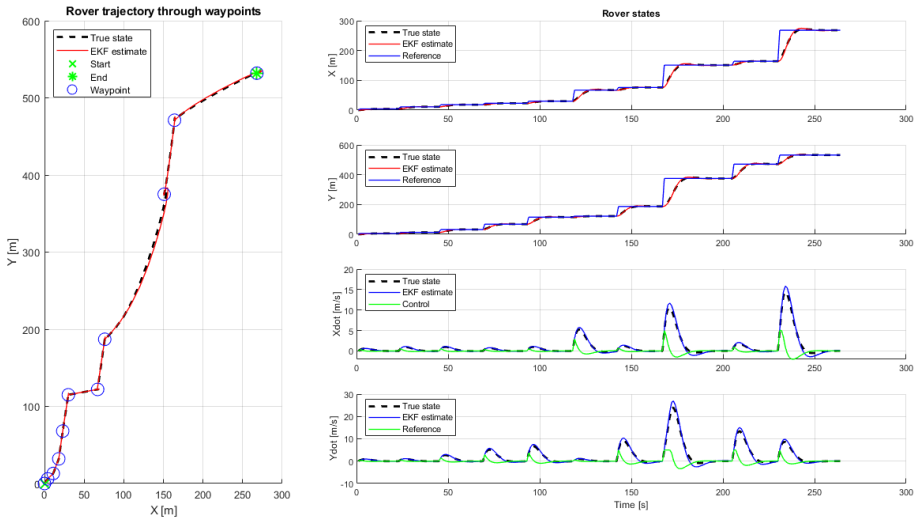

Figure 6: EKF estimate of rover trajectory using GPS measurements and input u_k information.

(b) Implement proper orbits for satellites, i.e. simulate the actual GPS (and other GNSS) satellites with real-world, real-time orbits.

(c) Implement satellite clock simulation together with clock synchronization between satellites. Take relativistic effects into account.

(d) Implement the pseudorandom signal broadcasting, which carries the satellite position

and time.

2. Rover
   (a) Implement the Polaris dynamic or kinematic model, or attempt to implement this simulation in ROS, where the model already exists.
   (b) Implement the receiver clock simulation together with bias and drift corrections.
   (c) Enhance the waypoint navigation strategy. Current solution very naive.
3. Ranging
   (a) Implement Line of Sight checking for satellite broadcast. Currently simulation accepts signals that would be blocked by the Earth.
   (b) Implement Ionospheric and Tropospheric modeling to separate the error w_i to component parts for DGNSS.
   (c) Implement time of flight based on above atmospheric models and receiver/satellite clock biases.
   (d) Implement multipath errors from signal reflections.
4. Extended Kalman Filter
   (a) Enhance by including the clock bias and drift terms in state estimation.
   (b) Merge with IMU data from rover simulation.
5. DGNSS
   (a) Implement DGNSS reference station and algorithm(s).
   (b) Implement messaging between receiver and DGNSS reference station.

8.7 Exploring Matlab built-in GPS functions

Matlab has a built-in functionality for simulating GPS receivers called `gpsSensor` [35]. We used this together with waypoint generation function `waypointTrajectory` [36] and some coordinate transformation to generate a visualization on how DGNSS corrections work. This was simply a visualization and no actual correction was done outside of directly manipulating the error covariances between two sensors, as seen in Figure 7. The visualization is a standalone Matlab script in the `gps\_sensor\_sim.m` file.

These built-in Matlab functionalities could possibly be used to simulate a larger part of the project. The `gpsSensor` does not produce the ranging and pseudoranging values, nor does it simulate actual satellites. Essentially it only produces GPS measurements around a pre-determined trajectory.

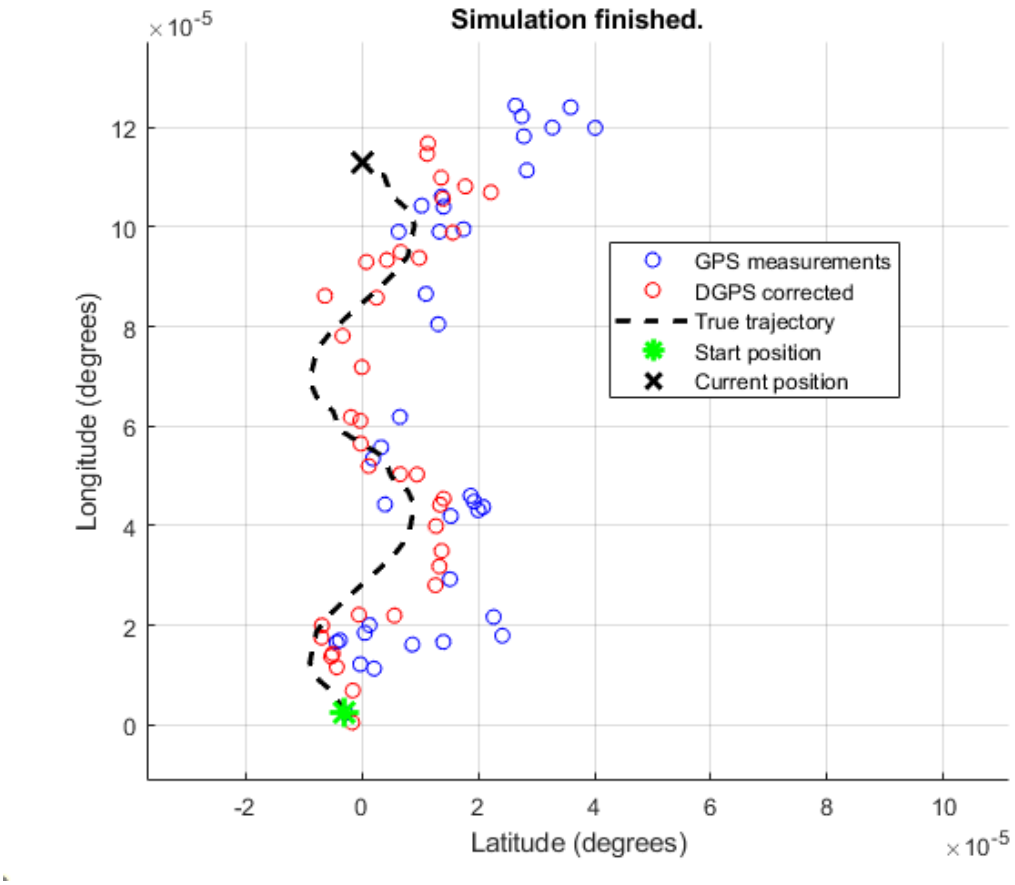

Figure 7: Visualizing GPS and DGPS measurements for a trajectory using Matlab's `gpsSensor` GPS receiver simulator.

9 Reflection of the project

It would be quite a stretch to say that this project was a major success – none of the original goals laid out in the projects description nor the original project plan were able to be completed. The effects of the COVID-19 outbreak and loss of access to the Polaris hardware essentially crippled our project, as nothing could be properly tested without the SPAN or ROS environment. However, it would also be disingenuous to entirely focus on the lockdown and some of the blame – although still a minority – should be shifted to us, the project group. A more focused and timely response to the situation could have resulted in a higher quality version of the final output, even if the contents would have been largely the same. In this section we aim to have a fair retrospective look on the project, the problems and the successes.

9.1 Achieving objectives

When writing the initial project plan for this project, our view on the difficulty or scale of some of the objectives was too naive.

A good example of a mismatched view is underestimating the impact of the business aspects portion on our workflow. The project's main goal of implementing DGNSS corrections to the Polaris was not easily transferable to a business idea and as such, the business aspects document and presentation took over a far too large portion of the focus. However, we felt that the resulting business idea presentation was good practice in simplifying complex technical topics to a short blurb and we believe we were successful in that regard and we count this as a successfully achieved objective of the project.

On the other hand, the effects of COVID-19 were devastating to our project and adapting to the changed and new objectives brought with them some uncertainty. Regardless, we successfully explored our options and formulated the new main objective. First to post-processing logged GPS data and then to simulating a GPS signal. The simulation objective was much more theoretical in nature than any of the original objects, but we felt that we achieved a reasonable goal with the simulation. Noteworthy of the process was the very prolific group meetings, held remotely. Collaboratively achieving consensus on redirecting the project focus is discussed in further detail in section 9.4.

9.2 Timetable

Our initial timeline for the project was just an estimate based on napkin maths. We should have directly budgeted n hours to "project technical issues", covering all possible work packages, instead of baking them in to each task separately, which bloated many of the estimates. Most of the initial time estimates made no sense in the end, as those work packages could not be completed due to the COVID-19 lockdown.

Before the lockdown we experienced a case of underestimating workload, where we encountered the technical issues with ROS and RPi, which were covered in detail in section 5. The initial person-hour estimate for getting ROS running on RPi was 15 hours as shown in Table 3, with an additional 100 hour leeway to get it integrated to the Polaris ROS network – these were woefully bad underestimations in hindsight.

As we did not expect to build ROS multiple times, it made no sense to take into account the extensive build-time on a Raspberry Pi. With a mere 1 GB of memory, even with an additional 2 GB of swap disk space, and a 1.2 GHz processor, it took half an hour to finish a single build sequence – even when they kept failing. With two people waiting for the RPi to build, we used almost 50 hours all together for this work package, leaving our time-budget devastated.

9.3 Risk analysis

Our risk analysis in the project plan (Appendix A, pp. 13) seemed like an overkill at the start of the project. We foresaw the risk of losing access to the Polaris, but only temporarily. We also foresaw the risk of a group member falling ill by making sure each work package was worked on by at least two members. At the time of writing the risk analysis for the project plan in late January and early February, there was news of a novel Corona virus even in Finland [37], but we did not include this in our risk analysis. As such, our risk analysis did not cover the COVID-19 as a possibility, which we doubt any project group did.

Our risk analysis had a few items on technical issues, but we failed to raise the risk of time loss figuring out the ROS/RPi issues outlined in section 5. At the time of writing the risk analysis we felt that these would not pose as large issues as they did in the end, and omitted them from the risk analysis.

The scope creep risk applies in some sense to our exploration of continuation options after lockdown, which resulted in the simulation as the main goal. During the exploration phase the scope creep was high as we looked at what we could do with the tools at hand, if we should simulate and what we should simulate. Locking down the scope for the simulation early made sure we could finish it with the time remaining.

9.4 Project Meetings

Our approach to project meetings was informal as we didn't see the value proposition of formal ones. We started with weekly meetings in person, together with the projects instructor Tabish Badar. When Tabish was not available, we held the meeting for the project group only. Once the COVID-19 lockdown came into effect, we moved into remote meetings with Microsoft Teams. In addition to the weekly meetings, everyday communication was done over the groups Telegram chat channel.

The aim for the project meetings was to keep them informal and short. As such, we had a default agenda for each that included: going over the past weeks work, discussing technical issues and planning/designing necessary parts and finally determining the steps for the following week. The project manager summarized the weekly meetings into a short memo that was sent out by email to the other members and the instructor after each (most) meetings.

The informal approach to the project meetings seemed to work for us. The only real changes that could enhance them would have been using some template for the memos, for faster usage.

It should be mentioned, that after being denied access to the lab the group was slightly flabbergasted. The possibility for completing the project as planned had turned impossible overnight and a new

course of action would have to be agreed upon. When taking into account that teleconferencing takes some getting used, the meetings where the group sought for viable options for completing the project (and the course) were very productive and good. The individuals of the group pulled together towards a common goal.

10 Discussion and Conclusions

In the end we would call the project a tentative success from our end. COVID-19 destroyed any hope of finishing the original project, but we still managed to divert the focus into a related topic, which could be worked on and researched remotely. The GPS simulation taught us more about the GNSS positioning than simply implementing the DGNSS client would have. In this chapter we list our personal learning achievements and some feedback to the project instructor regarding the project from our end.

10.1 Personal notes and learning achievements

Jani Arponen

Initially the project seemed very daunting as I had little to no experience in any of the key elements listed: ROS, C++ and D/GNSS. I didn't learn much about the ROS or C++ sides due to the COVID-19 blocking those aspects of the project. I wanted to look into project management partly to be able to contribute to the project in that sense if not in the technical parts.

After COVID-19 and the project focus shift I was able to contribute much more in technical matters by exploring the post-processing and simulation aspects, researching GNSS and DGNSS algorithms and research and in the end applying my knowledge of Extended Kalman Filtering in the simulation portion. The primary learning achievements for me was all the research I did into D/GNSS – and becoming better at the act of researching and information recovery itself.

Working as the project manager I learned some key aspects that I would do differently, were I to manage another project in the future. These aspects include tools and techniques such as task lists and Pomodoro for managing my own work and keeping up with other group members – these were the secondary learning achievements of this project for me.

Anand George

I was very excited about the project initially as this was a chance to work with and learn more about mobile robots and autonomous navigation. But as the requirements and scope of the project changed into something else, I lost interest in the project. Still it's an opportunity to learn something new. I took up the DGPS corrections part as I had some experience in using various APIs before. It was a challenge initially to understand the data from NLS and to send it to SPAN. Later, I tried to get data from the USB receiver also. Overall, I got some idea about the GNSS and DGNSS corrections and how they can be used in navigation.

Sakke Jussmäki

I enjoyed working with physical hardware so this project was very interesting. The battle with ROS and RPi was very educational in many ways. As I was very hardware focused, moving into the realm of simulation was tough. I did learn some new things about how GNSS works. Not getting to see my work finished in the final result is a bit disappointing.

Tommi Penttilä

I found the topic of the project interesting. It was especially nice to get to see the Polaris e-ATV and its hardware. I feel that I have learned a lot about GPS, especially when I hadn't been that familiar with how the GPS works before this project.

Ville Piesala

The project was OK, but the group was big and other members were more skilled. In hindsight I would have wanted to contribute more, but I still think I was adequate group member. I was really interested in the Polaris e-ATV and I even had a lot of experience on ATV's, but as the project was about GPS and ROS I didn't have useful skills. I learned something about GPS, ROS which was nice, but I am sad that the project was not what was advertised and that my skills were not really needed. I am happy that the project is now done.

Markus Sarlin

According to my initial understanding of the project topic I was looking forward to getting to learn more about the interaction of hardware and software in the setting of an autonomous vehicle. On sort of the hardware part I did get to learn quite a bit, due to the *Build ROS on Raspberry Pi* -adventure not going quite as planned and ending up using the better part of two weeks in the process. The Polaris we did not get to interact with that much that it would constitute a real learning experience.

On the software side I was expecting to be using C++, but it turned out that I got to get familiar with GPS raw data formats. In addition to those, I learned a lot on how GNSS systems work.

10.2 Feedback

In general the project was interesting with clear learning goals and at least one clear project goal, the DGNSS implementation. Some other goals were unfortunately a bit less clear though. It was good to have an introduction to the Polaris in the lab, however a more hands-on approach could have helped to first understand the underlying system in more detail, to better be able to produce additions to the Polaris.

Even if the original plan of the project was realizable, there was no need for a six member group for this project, a more suitable number would have been four or even three. We know the group size was imposed largely from the course side and not much could be done about it. We appreciate no "busy work" was added to the project due to this.

The title and description of the project introduction is slightly off. There wasn't much to do with waypoint navigation itself, just technical implementation of DGNSS with RPi – a nonstandard ROS platform. For some members, this may have influenced the decision on applying for this project over others (in both ways) if it was more clear.

11 List of Appendices

A. **Project Plan** – The project plan document from the start of the project minus the last Gantt chart (found in Appendix B)

B. **Detailed Project Schedule** – Breakdown of the timetable for the original project plan in a Gantt chart format.

References

[1] Tabish Badar, 2019, *Implementation of the autonomous functionalities on an electric vehicle platform for research and education*, Aalto University Master's Thesis.

[2] Lasunncty at the English Wikipedia / CC BY-SA, [Online], Accessed 2020-05-21.

[3] Wikipedia contributors, 2020, *Orbital elements – Wikipedia, The Free Encyclopedia* [Online], Accessed 2020-05-21

[4] Bar-Shalom Yaakov, Li X. Rong, Kirubarajan Thiagalingam, 2001, *Estimation with Applications to Tracking and Navigation*, John Wiley & Sons, Inc.

[5] Kumar.N, Ashok & Chittineni, Suresh & Rao, Gottapu, 2018, *Extended Kalman Filter for GPS ReceiverPosition Estimation*, 10.1007/978-981-10-7566-7.

[6] USA Department of Defence, 2008, *Global Positioning System Standard Positioning Service Performance Standard*, 4th Edition, [Online]

[7] Frank van Diggelen and Per Enge, 2015, *The World's first GPS MOOC and Worldwide Laboratory using Smartphones"*, Proceedings of the 28th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2015), pp. 361–369.

[8] European Space Agency Navipedia, 2011, *Differential GNSS*, [Online], Accessed 2020-05-24.

[9] *FinnRef GNSS stations - National Land Survey of Finland*, [Online], Accessed on 2020-05-18.

[10] *SPAN-IGM-S1 - Novatel*, [Online], Accessed on 2020-05-28.

[11] *National Land Survey of Finland Positioning services - DGNSS services*, [Online], Accessed 2020-02-04.

[12] *SPAN on OEM6 Firmware Reference Manual Rev 8 - NovAtel*, [Online], Accessed 2020-02-27.

[13] *ROS – Target Platforms*, [Online], Accessed on 2020-05-25.

[14] *Ubiquity Robotics – TEAM: Helping those who build robots build them faster*, [Online], Accessed 2020-5-28.

[15] *Ubiquity Robotics Downloads*, [Online], Accessed 2020-3-7.

[16] *Installing ROS Kinetic on the Raspberry Pi*, [Online], Accessed 2020-2-21.

[17] John Ousterhout, *CS 140: Operating Systems, Managing Flash Memory*, Stanford University, 2014, [Online], Accessed 2020-5-28.

[18] *ROS.org – Packages*, 2019, [Online], Accessed 2020-5-28.

[19] *Definition - What does Headless Computer mean?*, 2013, [Online], Accessed 2020-5-28.

[20] RTCM Special Committee No. 104, August 20 2001, *RTCM 10402.3 RECOMMENDED STANDARDS FOR DIFFERENTIAL GNSS (GLOBAL NAVIGATION SATELLITE SYSTEMS) SERVICE VERSION 2.3*, Radio Technical Commission For Maritime Services, Virgina U.S.A.

[21] Aalto University Crisis Management Team (CMT), *Koronavirukseen liittyvää tietoa 13.3. – Information on coronavirus 13 March – Information om coronaviruset 13.3*, 2020-03-13.

[22] *Android Developer – Build location-aware apps*, [Online], Accessed 2020-3-19.

[23] Lukas Wachter, Onur Sari, Valtteri, and Karhu Panu Pellonpää, *Building a Simulation Platform for Polaris e-ATV in ROS using Gazebo*, Aalto University, School of Electrical Engineering Project course Final report, [Online], Accessed 2020-5-4.

[24] *hector_gazebo_plugins*, [Online, Accessed 2020-5-3.

[25] International GNSS Service (IGS), RINEX Working Group and Radio Technical Commission for Maritime Services Special Committee 104 (RTCM-SC104), *RINEX The Receiver Independent Exchange Format*, [Online], July 14th, 2015.

[26] Google Play, *Geo++ RINEX Logger*, [Online], Accessed 2020-3-21.

[27] *Positioning Services – RINEX service*, [Online], Accessed 2020-3-21.

[28] *Observation RINEX 3.01 Format*, [Online], Accessed 2020-3-22.

[29] *National Marine Electronics Association*, [Online], Accessed 2020-5-10.

[30] Eric S. Raymond, *NMEA Revealed*, Version 2.23, 2019 [Online], Accessed 2020-5-4.

[31] Google Play, *Ultra GPS Logger Lite*, [Online], Accessed 2020-5-4.

[32] *NMEA data*, [Online], Accessed 2020-05-10

[33] Wickert Mark, Siddappa Chiranth, 2018, *Exploring the Extended Kalman Filter for GPS Positioning Using Simulated User and Satellite Track Data*, Proceedings of the 17th Python in Science Conference (SciPy 2018), pp. 84–91.

[34] MathWorks, *lqr documentation*, [Online], Accessed on 2020-05-22.

[35] Mathworks, *gpsSensor documentation* [Online], Accessed on 2020-05-10.

[36] Mathworks, *waypointTrajectory documentation* [Online], Accessed on 2020-05-10.

[37] Wikipedia Contributors, *Timeline of the COVID-19 pandemic in January 2020*, Wikipedia, The Free Encyclopedia, 2020, [Online], Accesssed on 2020-05-26

Project Plan
Project 1-9
Waypoint navigation
for an autonomous all-terrain vehicle

Jani Arponen
Anand George
Sakke Jussmäki
Tommi Penttilä
Ville Piesala
Markus Sarlin

04.02.2020

Appendix A: Project Plan

1 Information page

Students

Jani Arponen
Anand George
Sakke Jussmäki
Tommi Penttilä
Ville Piesala
Markus Sarlin

Project manager

Jani Arponen

Official Instructor

Tabish Badar

Approval

The Instructor has accepted the final version of this document Date: xx.2.2020

Changelog

Version	Date	Author	Description
1.0	2020-02-02	All	First draft
1.1	2020-02-03	All	Work estimation and schedule added

Table 1: Document changelog.

Appendix A: Project Plan

Contents

Appendix A: Project Plan

2 Background

Aalto University's Autonomous Systems Research Group has been developing an autonomous vehicle over a number of years. This vehicle is based on a Polaris electric all-terrain vehicle (e-ATV), which has been incrementally improved with additions of hardware and software together enabling autonomous operation. These improvements have been made in a series of study and research projects, such as this one.

Recently, the instructor of this project, Tabish Badar, wrote a master's thesis based on the e-ATV, concluding that, in addition to expediting the vehicles initialisation process, the more precise localisation with a differential Global Navigation Satellite System (DGNSS) would be advisable.

The National Land Survey of Finland provides free DGNSS correction data for code-based positioning. Where conventional satellite navigation systems reach an accuracy of a few meters to tens of meters, differential GNSS can improve accuracy to centimeter range. The positioning corrections are based on error models calculated from positioning signals of stationary reference stations whose positions are precisely know. These deviations are sent to the user's receiver, which first corrects its own observed distance to the satellites before computing its position.

Options for more precise positioning, utilising these aforementioned corrections, include the open DGNSS service and Real-Time Kinematic service (RTK). Open DGNSS is free of charge and can be used in real-time. It enables a precision of about half meter, where as the RTK, only available for research purposes, provides up to centimeter-level accuracy.

Whether the utilised technology will be the open DGNSS or the RTK – which, considering the nature of the project, might be an option – the improved positioning should support more reliable positioning and operation of the autonomous vehicle.

3 Expected output

This project aims to produce an embedded device that shall provide Differential Global Navigation Satelite System (DGNSS) corrections to the Polaris e-ATV. The embedded device shall be a Raspberry Pi (RPi) computer running Robot Operating System (ROS) to easily integrate to the current ROS network running on the Polaris. The DGNSS correction data shall be obtained from the DGNSS service (link) provided by the Finnish National Land Survey (NLS).

The aimed users for the embedded device are the members of the Autonomous Systems research group, especially those working with the Polaris e-ATV. The business development plan expands the target group to anyone in need of a device providing DGNSS corrections on their ROS platform.

The expected user experience is to install the device, connect it to the ROS network and configure the needed parameters for said connection(s), after which the device shall work autonomously and silently without need for any user intervention.

The temporal performance of the embedded device depends on the DGNSS service provided by NLS. Position correction signals are offered at a frequncy of 1 Hz, which shall be the expected temporal performance for providing DGNSS corrections to the ROS network. The performance

Appendix A: Project Plan

and/or the correctness of the data shall be measured on the Polaris e-ATV and the expectation is that the variance of the position shall be lower, when using the DGNSS corrections.

The expected demonstration of successful completion of the project is likely to be a rosbag visualisation on RViz. Rosbags are a kind of log files used within the ROS framework. A bag is set to store ROS messages – communications between ROS nodes – which can then be played back and analysed offline. Hands-on-testing with the e-ATV is expected within the framework of this project, yielding the necessary test data This test data is then be visualised for demonstration purposes.

4 Phases of project

4.1 Planning

This is the current phase of the project and consists of figuring out the requirements and rough scope of the project, by studying the provided documentation on the project topics and technologies. The goal for this phase is to produce this project plan document.

4.2 Design

During the design phase, we will be designing the system based on the requirements and scope outlined in the project plan. Further research on the project topics, learning new technologies and software will also continue during the design phase. The initial project requirements and scope will be iterated on and honed during this phase as solutions and/or problems arise during the systems design and research. The output of this phase will be some form of documentation that will outline the basic system infrastructure, technologies to be used, and further requirements for the software and how it can be divided into work packages.

The design phase will also output some rudimentary software requirements, which are then refined in agile fashion when actually implementing each software package.

4.3 Business aspects

The business aspects portion of the project is an integral part of the design phase and will be running concurrently with it. The milestones for this part are the pitch presentation and related documentation.

4.4 (Software) development and testing

This is the "bulk" of the project and runs concurrently with the end of the design phase. The early work in this phase will influence the design portion. The software can be roughly split into individual packages that each have their own milestones, which are verified with some tests. To start with, the embedded device must be running ROS, to enable future development. The next steps can be ran somewhat concurrently and include writing a client that connects to the NLS DGNSS service, a client that publishes data to the ROS network in the correct format, introduce logging etc. other features that will be specified further as the design process uncovers them. Each feature

Appendix A: Project Plan

shall be tested first as separate packages and later on integration tests with Polaris e-ATV shall be done. Once the integration tests are successful, final delivery for the project can be done.

4.5 Reporting and documentation

Documenting the progress can and should be done during the full length of the project to ease the reporting phase. The deliverables for the reporting phase are the final report; and the documentation the and presentation for the final gala.

4.6 Milestones

The project milestones with deadlines are explained below. The deadlines that are **bolded** are hard deadlines imposed by the project course. The deadlines that are in *italics* are subject to change in further iterations and shall be updated once known.

Phase	Milestone	Deadline	Description
Planning	P1	2020-02-03	Project plan submission to instructor
	P2	**2020-02-06**	Project plan submission to MyCourses
Design	D1	*2020-03-01*	System infrastructure document
	D2	*2020-03-01*	Software requirements specification
Business	B1	**2020-03-06**	Business aspects seminar slides submission to MyCourses
	B2	**2020-03-10**	Business aspects seminar and presentation
	B3	**2020-03-13**	Business aspects document submission to MyCourses
Development and testing	M1	*2020-02*	A working development platform (RPi with ROS)
	M2	*2020-03*	Client for NLS DGNSS corrections
	M3	*2020-03*	DGNSS corrections published to ROS network
	M4	*2020-04*	Data correctness testing
	M5	*2020-05*	Integration testing with Polaris e-ATV
	M6	*2020-05*	Delivery of the software and associated documents
Reporting	P3	**2020-05-19**	Final gala and presentations
	P4	*2020-05-22*	Final report submission to instructor
	P5	**2020-05-29**	Final report submission to MyCourses

Table 2: Project phases and milestones.

5 Work breakdown structure (WBS)

The projects work breakdown structure (WBS) is simple, though some sections are omitted or are included as subtasks of their respective tasks. Estimating the amount of work required for each task has proven to be a bit difficult, as none of the project group members are software developers. The WBS provided in the below figure 1 is split into largeish units rather arbitrarily, based mostly on the type of work required. Additional work not broken down in this WBS, but is included within each applicable task, include things such as writing documentation, code review, test cases etc.

Unit 1.1 includes all project management tasks, which are recurring for the entire duration of the project. Expecting roughly 4-5 person-hours per week of project management, though a

Appendix A: Project Plan

larger chunk of that will be spent at the start of the project (e.g. this project plan document), but project meetings will be held weekly for the duration of the project, which will take considerable time as well.

Unit 1.2 has all things hardware, procuring and testing them. We estimate roughly 20 to 30 person-hours total for this section, but there may be additional, unforeseen tasks regarding hardware, such as compatibility issues with the GPS antenna/module.

Unit 1.3 takes care of the system level tasks, that are not specifically related to the hardware, although closely linked on some items, such as the 3G/4G modem and GPS antenna/module configurations. Setting up the ROS environment also touches on the software tasks, but we feel this is a valid separation to make. The estimated time for the system level tasks are anywhere from 50 to 150 person-hours, depending on possible complexities.

Unit 1.4 contains the "bulk" of the project, as stated in the scheduling section. The primary focus of creating a DGNSS client using the NLS DGNSS service should not prove too difficult, as the NLS DGNSS API has decent documentation. The GPS averaging and RTK clients refers to the possibility of testing how different approaches improve locationing. There may be some data format conversion needed between the DGNSS corrections and the expected format the embedded computer or SPAN module uses in the Polaris ROS network. The ROS publisher should be relatively standard and not pose any issues implementing, but this also depends on the ROS network specifics in the Polaris. We expect this unit to take anywhere from 400 to 700 person-hours of work to complete.

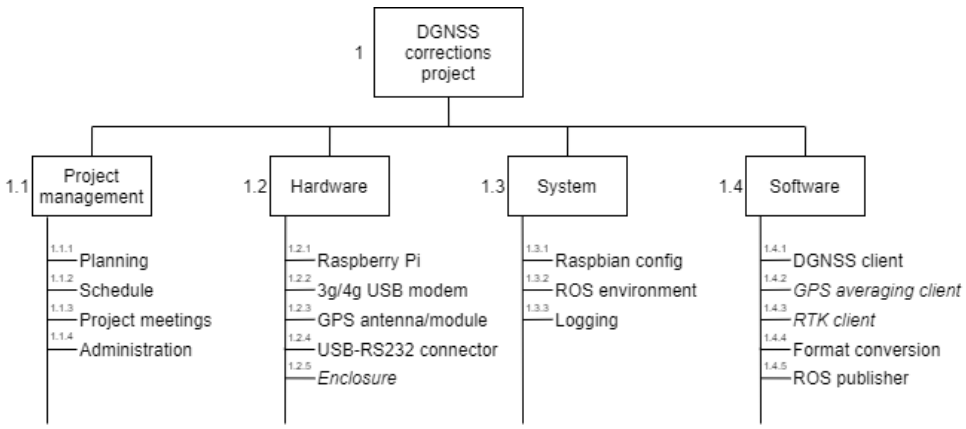

Figure 1: Work Breakdown Structure

Appendix A: Project Plan

6 Work packages and Tasks of the project and Schedule

Each work package mentioned in this section of the document shall be worked on by a minimum of two people to minimize risk. The work packages and tasks are listed in the below table 3 and the work package contents are expanded more in section 6.1. Additional, smaller tasks are explained in section 6.2. If applicabe, the work packages are linked to their associated milestones and the tasks have an associated time estimate in person-hours. The time estimates are exactly that, estimates, and we have been very conservative regarding the amounts. We expect additional tasks to come up as the project lives.

Appendix A: Project Plan

WP#	MS#	Assigned	Task#	ph	Description
WP1	-	All	Task 1.1	20	Reading Tabish's thesis, GNSS tech, etc.
	P1	All	Task 1.2	30	Write first draft of project plan
	-	All	Task 1.3	10	Iterate project plan based on feedback
	P2	Jani	Task 1.4	1	Submission of project plan to MyCourses
WP2	-	All	Task 2.1	180	Weekly project meetings
	-	Jani	Task 2.2	100	Project management administrative work
WP3	-	All	Task 3.1	40	Business aspects documents
	-	Markus, Jani, Ville	Task 3.2	25	Rugged enclosure design for DGNSS module
	B2	All	Task 3.3	35	Business aspects seminar
	B1, B3	Jani	Task 3.4	1	Business aspects documentation submission
WP4	D1	Sakke, Markus	Task 4.1	5	Raspberry Pi and SD-card
	D1	Sakke, Markus	Task 4.2	5	3g/4g USB modem and SIM-card
	D1	Sakke, Markus	Task 4.3	5	GPS antenna/module
	D1	Sakke, Markus	Task 4.4	5	USB to RS232 connector
	D1	Ville, Tommi	Task 4.5	20	Enclosure for the module
	D1	Ville, Tommi	Task 4.6	10	Testing hardware in different environments, within the campus area
	D1	Sakke, Markus	Task 4.7	15	Documenting system infrastructure
	-	Sakke, Markus, Jani	Task 4.8	5	Power supply
WP5	M1	Sakke, Markus	Task 5.1	5	Raspbian config
	M1	Sakke, Markus	Task 5.2	10	ROS running on Raspbian
	M5	Sakke, Markus	Task 5.3	100	Integrating to Polaris ROS network
	M3	Sakke, Markus	Task 5.4	5	ROS publisher for DGNSS corrections
	-	Sakke, Markus	Task 5.5	5	Implement logging for location data
WP6	D2	Anand, Tommi, Ville	Task 6.1	50	Researching NLS DGNSS API
	M3	Anand, Tommi, Ville	Task 6.2	100	Design and coding of the DGNSS client
	M3, M4	Anand, Tommi, Ville	Task 6.3	20	Testing the DGNSS client
WP7	D2	Jani, Anand, Tommi	Task 7.1	80	Design and coding of "our own" GPS averaging client
	M4	Jani, Anand, Tommi	Task 7.2	20	Testing the GPS averaging client
WP8	D2	Markus, Anand, Ville	Task 8.1	100	Researching RTK technology and RTKLIB
	-	Markus, Anand, Ville	Task 8.2	120	Design and coding of the RTK client
	M4	Markus, Anand, Ville	Task 8.3	30	Testing the RTK client
WP9	-	All	Task 9.1	50	Hands-on with Polaris for testing
	-	All	Task 9.2	50	Data collection on different solutions performances
	M4, M5	All	Task 9.3	10	Evaluating the different clients using the Polaris e-ATV, comparing to current solution
WP10	P3	All	Task 10.1	100	Final gala presentation
WP10	P4, M6	All	Task 10.2	100	Final report to Tabish
WP10	P4	All	Task 10.3	20	Possible iteration on final report
WP10	P5	Jani	Task 10.4	1	Final report submission to MyCourses

Table 3: Work packages

Appendix A: Project Plan

6.1 Work packages

WP1 - Project plan:	Writing the project plan and other associated tasks.
WP2 - Project management:	Handle administrative tasks regarding the project, such as communication, scheduling, documenting etc.
WP3 - Business aspects:	The business aspects aim to produce a business proposal, which will be presented in business aspects seminar.
WP4 - Hardware:	Obtaining and handling the separate pieces of hardware needed for the project. Figure out how to integrate them to the system and iron out any possible problems.
WP5 - ROS system:	Set up the ROS environment on RPi to develop on. Later, integrate it to Polaris ROS network.
WP6 - DGNSS client:	Obtain DGNSS corrections from NLS DGNSS API.
WP7 - "our own" client:	Develop our own (simple) algorithm for reducing GPS error e.g. moving average.
WP8 - RTK client:	Implement RTK using the RTKLIB.
WP9 - Data collection:	Run the clients on Polaris and collect data on performance. Analyze data and create demonstrations.
WP10 - Final report:	Create final gala presentation and write final report.

6.2 Tasks

In general, the tasks outlined in the above table 3 cover each task for a given work package, but below are examples on what kind of work goes into each task that requires programming:

1. Documentation
2. Research and design
3. Programming
4. Testing
5. Code review
6. Integration

6.3 Detailed schedule

The schedule expressed in the Appendix A is a rough estimate with somewhat placeholder dates on completion of specific work packages. These will, without question, live as the project advances and the design gets more and more concrete. Updating the schedule and this Gantt chart is the main way to keep track of work progress for the project meetings.

Appendix A: Project Plan

7 Work resources

7.1 Personal availability during the project

Table 4 shows the expected availability of team members for the duration of the project – week 5 to week 21.

	Jani Arponen	Anand George	Sakke Jussmäki	Tommi Penttilä	Ville Piesala	Markus Sarlin
Week 5	8	12	15	10	10	7
Week 6	8	12	15	10	10	10
Week 7	8	12	15	10	10	10
Week 8	20	8	15	5	15	21
Week 9	15	10	10	15	10	10
Week 10	15	10	10	15	0	14
Week 11	15	10	10	15	10	10
Week 12	15	10	0	15	10	14
Week 13	10	10	10	15	10	10
Week 14	5	10	10	10	5	10
Week 15	5	5	10	5	8	7
Week 16	0	20	10	15	8	21
Week 17	25	20	10	15	5	21
Week 18	25	20	5	15	0	21
Week 19	25	20	5	15	8	21
Week 20	25	20	5	15	8	21
Week 21	25	20	5	15	8	21
Total	249	229	160	215	135	249

Table 4: Number of hours available for the project (excluding lectures and seminars) per week.

7.2 Personal goals

Listed below are the articulated points of interest for team members, outlining which parts of the project each would like to partake.

Tommi Penttilä: I would like to learn how to use ROS, improve my c++ programming skills and learn more of autonomous vehicles but I'm fine with doing any tasks that this project has.

Jani Arponen: I am interested in autonomous systems, especially self-driving cars, which I am looking to learn more about to figure out, if it is something I want to work with in the future. I also wish to improve my programming skills and learn project management in a software project.

Anand George: Interested in autonomous mobile robots and their applications. I would like to improve my skills in ROS and programming but also ready to work on any task for completing this project.

Markus Sarlin: A holistic interest in autonomous systems. Within the scope of this project this means, I would like to get more practise using ROS and C++ in addition to deepening my

Appendix A: Project Plan

understanding of using a GPS to guide an autonomous vehicle.

Ville Piesala: Programming on embedded systems, especially Rasperry pi is always interesting. I have no particular goals except passing the course, but learning new things is always good.

Sakke Jussmäki: I hope to learn more about ROS in real applications and get a deeper understanding of embedded systems.

8 Cost plan and materials

There is no allocated budget for the project, but some funding is needed. Table 5 below describes the materials that are required to complete the project. There can also be additional costs and materials that are not foreseen. Some of the required materials might be available at the electric workshop (Sähköpaja) for free, but this option should not be relied on. Based on the table below and accounting for unforeseen costs a budget of 200€ is required.

Item Name	Estimated Cost	Description
Raspberry pi 3	44,90 €	The hardware the software is run on. Can be version 4 too.
SIM card/GPS module	90 €	Used to communicate with NLS. These modules could be separate, but combined models exist.
RS232 to usb cable	15 €	Used to connect the system to computer via serial connection.
SIM card	5,90 €	The example price is saunalahti prepaid
Case	5 €	The system is working outside in cold, hot and wet conditons so it needs protection.
Total	**161 €**	

Table 5: Items needed for the project.

9 Other resources

During the final testing phases of the project, access to the Polaris e-ATV is needed.

For each weekly meeting there is need for a room that shall be reserved before the meeting.

10 Project management

This section contains the responsibilities for the roles in this project.

Project manager

The project manager is responsible for all administrative work for the project such as submitting documents to MyCourses and the instructor, calling project meetings and writing the meeting minutes. Additionally the project manager is responsible for the communication between the project group and the instructor. The project manager also handles updating the schedule of the project based on completed milestones.

Appendix A: Project Plan

Instructor

The instructor is responsible for reserving the meeting room for weekly meeting, as the students in the project group lack the required privileges to reserve a suitable meeting room. In case the instructor is unavailable to reserve a room, the project manager will reserve some room available in Aalto Space. Additionally the instructor is responsible for the budget and giving access to the Polaris e-ATV for the project group, when testing the positioning performance.

Work package leader

The work package leader is responsible for the documentation and delivery of the work package. Each member in the work package group works on the given package, but the leader is ultimately responsible for the timely delivery of the work. The work package leader is also responsible for communicating the progress on the work in and outside of the weekly project meetings.

11 Project Meetings

The project group will meet weekly on Mondays at 12:15 to discuss project related topics. The default agenda will be discussing the progress on last weeks actions, possible problems and planning actions for the next week. Other topics are welcome in the meetings. By default, the project manager will write the meeting memo/minutes and emails them to the rest of the group and the instructor.

12 Communication plan

Communication in this project is done by using different methods. The main communication method is weekly face-to-face meetings with group members and the instructor. Outside of the meetings communication with the instructor is done by using email. Emailing with the instructor is mostly done by the project manager (Jani Arponen) who shares information with the rest of the group. Communicating with group members is done by using telegram group outside of the weekly meetings.

13 Risk management

In the below table 6 we have identified possible risks for the success of the project and some notes on how to handle them.

14 Quality

The main goal for the project is to improve the quality of the waypoint navigation system. This can be tested with the actual ATV by comparing the accuracy before and after. The module itself can be evaluated by repeating a GPS measurement in the same location over multiple days. The instructor has the most knowledge of the navigation system and is therefore the best person to define the increase in performance.

Appendix A: Project Plan

Risk	Probability	Severity	Mitigation
NLS DGNSS API uptime and availability	Very low	Critical	It is not possible for the project group to affect the availability of a third party API.
Loss of code	Very low	High	All code shall be committed to the projects git repository, which shall be backed up periodically.
Loss of person(s) responsible for work package or task	Very low	Medium	Every work package and task shall be worked on by minimum two people, meaning, at least two people are aware of what is going on for said package or task. Each package and task shall be documented to a degree that any other group member with sufficient skills on the topic shall be able to continue the work.
Raspberry Pi SD card corruption/breaking	Low	Low	SD card corruption is a known failure mode for Raspberry Pi's. An image shall be created to speed up deployment.
Scope creep	Unknown	High	Due to limited time, any additional scope creep may be hard to deal with, as in any other software development project, e.g. the RTK client may be too complex, and was not included in the initial pitch of the project.
Polaris unavailability	Unknown	Medium	The Polaris e-ATV may be unavailable for the projects needs when it is time to do testing or trying to integrate our embedded device to the ROS network. It may be possible to conduct some of the tests without access to Polaris.

Table 6: Project risks.

Reviewing the quality of code should be done after each Git commit by members of the group. Everyone is responsible for this together. Unsatisfactory parts should be noted and communicated back to the original programmer.

15 Changes to the project plan

At the very least, minor alterations to this plan are to be expected. Possible change include modifications to the timetable, both regarding the work breakdown and individual availability, work allocation, and project work packages.

Possible examples of work allocation changes are unexpected technical difficulties e.g. in compiling necessary packages or wicked software bugs, which would require more hands on deck. Given, that none of the team members are exactly professional software developers, these kind of problems seem likely.

Changes in work packages would entail unforeseen changes in the project requirements or unexpectedly tedious necessitating employing more resources to the task. Not to seem to pessimistic, also unanticipatedly straightforward tasks are plausible. It is doubtful that this would lead to major changes, unless occurring at the last-minute-panic-stage of the project.

Appendix A: Project Plan

Ideally these changes would be negotiated in conjunction with the whole team, preferably at the weekly meetings. It is, however, possible that not all team members will be present at all meetings, posing a problem with regard to plan adjustments.

Minor changes can be made with a majority vote, but any changes that would significantly affect a team members works load over a given time period would have to be decided unanimously. In the event of the affect member not being present at a meeting, these would then have to be agreed upon using email or the designated Telegram-channel.

16 Measures for successful project

Project success depends on many factors. The main goal of producing an embedded device that improves the positioning accuracy is easier to measure than some of the other goals explained below.

Project should advance in the planned schedule, and the schedule should be updated as the project advances and new tasks reveal themselves. In the project plan there are deadlines and milestones for when specific tasks should be done, which, in the best case scenario, are done before their deadline. In a more realistic manner, some tasks will take more time to get done than what was planned before starting project development, which is not necessarily a bad thing, as long as their effect is not catastrophic and other tasks can be scheduled around the delay. The main goal is to have the hard deadlines imposed by the project course met and additional care must be taken to make sure they are, as they can not be rescheduled.

Initially, the project had no budget, but very early on it became evident that some purchases must be made. The goal for the budget is to not spend an excessive amount of money and figuring out how to obtain hardware for low cost or, even better, for free. Free hardware may impose some risks to quality due to unknown condition. Breaking hardware or inflating the budget in some other way lowers success.

Team satisfaction is important part of project success to the team members. Each member evaluates how satisfied they were with the project work and the project final outcome. For a successful project, team members are satisfied with the effort that they put in for the project and the quality of the final outcome.

Lastly, the project client, Tabish, evaluates the project outcome from a technical and project management standpoint.

Appendix B: Detailed Project Schedule