TECHNISCHE UNIVERSITÄT WIEN

Institut für Computertechnik
Institute of Computer Technology

Master's Thesis

submitted by

Philipp-Sebastian Vogt, BSc

SIR LoCo - **S**martphone Localization **I**ntegration in **R**OS for **L**ow-**C**ost Precision Agriculture Robots

In partial fulfillment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Vienna, Austria, 2023

Study code:	E066 504
Field of study:	Embedded Systems
Supervisor:	Univ.Prof. Dipl.-Ing. Dr.techn. Axel Jantsch
Co-Supervisor:	Univ.Ass. Dipl.-Ing. Thomas Leopold, BSc

If you find this work useful, please cite it using the following BibTeX entry:

```
@Thesis{Vogt2023,
 type        = {Master's Thesis},
 author      = {Philipp-Sebastian Vogt, BSc},
 title       = {SIR LoCo - Smartphone Localization Integration in ROS for Low-Cost
    Precision Agriculture Robots},
 school      = {Vienna University of Technology (TU Wien)},
 year        = {2023},
 address     = {Gusshausstrasse 27--29 / 384, 1040 Wien},
 month       = {August},
}
```

Contact us:

philippvogt@gmx.at

Acknowledgments

I want to thank my supervisors, Univ.Prof. Dipl.-Ing. Dr.techn. Axel Jantsch and Univ.Ass. Dipl.-Ing. Thomas Leopold, BSc, for supporting me during the whole thesis.

I would also thank my family, girlfriend, and friends, who supported me throughout the university.

Abstract

This thesis aims to evaluate and find a suitable way to localize farming robots and transmit other sensor information from an Android smartphone to it. Multiple agriculture robots utilize expensive RTK receivers and antennas, while old but still working smartphones are thrown away. With the results of this thesis, in most cases, an old smartphone could replace these expensive receivers and even do good to the environment by upcycling otherwise produced electronic waste. The performance was evaluated with and without additional hardware in five different scenarios to showcase different levels of complexity. Another benefit was utilizing multiple other smartphone sensors, even further reducing the need for extra hardware and, therefore, the overall cost and e-waste. One scenario even demonstrated the usage of a Linux subsystem on Android to run even more complex software, while improving the control and accuracy of the active GNSS receiver. It was possible to show that with the help of correction data, a horizontal localization error in the middle cm range could be achieved.

Kurzfassung

Das Ziel dieser Arbeit war es einen Weg zu finden, landwirtschaftlich genutzten Roboter eine kosteneffiziente Möglichkeit zu bieten sich zu lokalisieren. Dabei war der Nachhaltigkeitsgedanke im Vordergrund um nicht noch weitere Hardware zu benötigen. Durch die Nutzung eines alten Android Smartphones, welches zusärtlich eine Vielzahl an Sensoren besitzt, kann dies realisiert werden. Das Smartphone kann GNSS Satelliten Signale empfangen und sich lokalisieren, jedoch nicht in einer ausreichenden Genauigkeit. Durch die Evaluierung und Implementierung mehrer Verbesserungsmöglichkeiten, unter anderem Fehlerkorrekturdaten des APOS, konnte die Fehlergenauigkeit in der Ebene bis in den Zentimeter Bereich verbessert werden. Des weiteren konnte gezeigt werden, dass ein Linux Subsystem auf dem Smartphone installiert werden kann und sogar USB Treiber für ein "Software Defined Radio" und einer aktiven GNSS Antenne im Bereich des Möglichen liegen.

Contents

List of Tables

List of Figures

Acronyms

AMCL	Adaptive Monte Carlo Localization. 19
API	Application Programming Interface. 24, 31
APOS	Austrian Positioning Service. 22, 36, 48, 56
APT	Advanced Packaging Tool. 25, 33
ASIC	Application Specific Integrated Circuit. 59
BEV	Bundesamt für Eich- und Vermessungswesen. 22
BKG	Bundesamt für Kartographie und Geodäsie. 22
CDF	Cumulative distribution function. 41, 42, 44, 47, 49, 52
CEP	Circular error probable. 41, 53, 54, 58
CPU	Central Processing Unit. 33
DC	Direct Current. 34
DGPS	Differential GPS. 17, 19, 21, 22
EKF	Extended Kalman Filter. 26, 27, 30, 37, 52
GDB	GNU Debugger. 56
GNSS	Global Navigation Satellite System. 11, 14–27, 30, 31, 34–36, 38, 39, 46, 54, 59, 60
GNSS-SDR	Global Navigation Satellite System (GNSS) Software Defined Receiver. 17, 18, 23, 59
GPS	Global Positioning System. 14, 19, 20, 34
GUI	Graphical User Interface. 23

HTTP Hypertext Transfer Protocol. 22, 25

I/Q In-Phase & Quadrature. 23, 24, 33, 34

ICT Institute of Computer Technology. 14

IMU Inertia Measurement Unit. 16, 19, 26, 30, 31, 37, 39, 55

LoRaWAN Long Range Wide Area Network (WAN). 17

MSM Multiple Signal Message. 22, 36

MVVM Model View ViewModel. 19

NMEA National Marine Electronics Association. 17, 21, 25, 29, 30, 37, 38

NTRIP Networked Transport of Radio Technical Commission for Maritime Services (RTCM) via Internet Protocol. 18, 19, 22, 35–37, 56

PCB Printed Circuit Board. 17

PPP Precise Point Positioning. 18

QAM Quadrature Amplitude Modulation. 24

REP Robot Operating System (ROS) Enhancement Proposals. 26, 30

RF Radio Frequency. 23, 24, 34

RINEX Receiver Independent Exchange Format. 21, 22

ROS Robot Operating System. 12, 18–20, 25, 26, 30, 31, 34–39, 60

RTCM Radio Technical Commission for Maritime Services. 12, 18, 19, 21, 22, 35, 36, 56, 57

RTK Real Time Kinematic. 16, 17, 19, 21, 36, 47, 59

SC Special Committee. 21, 22

SDR	Software Defined Radio. 15, 17, 20, 23, 33, 34, 52–54, 58–60
SPS	Single-Point Positioning. 47–50
TCP	Transmission Control Protocol. 24, 25, 30, 34, 35
TPMS	Tire Pressure Monitoring System. 60
TXCO	Temperature Compensated Crystal Oscillator. 17, 24
USB	Universal Serial Bus. 33, 34
VRS	Virtual Reference Station. 23
WAN	Wide Area Network. 12, 17

1 Introduction

This chapter will give a brief introduction and motivation for the thesis. This includes a section for motivation and a section for the research questions. The motivation section describes why and how the thesis was done, while the research questions try to reduce the problem into more concrete and answerable questions. Furthermore, the research question section will give a short reason why the smartphone selection was made the way it was.

1.1 Motivation

The main idea of the thesis was the need for localization of a robot platform used in agriculture on a budget. For example, the Acorn Precision Robot from Twisted Fields [1] or a farming robot under construction at Institute of Computer Technology (ICT) called Vermin Collector. The robot should be capable of following planned trajectories based on waypoints to a cm accuracy. This high accuracy is needed to follow the wheel tracks on the field because even 10 cm off, the robot would destroy the crops. Most agricultural machines use a satellite-based navigation system, a Global Navigation Satellite System (GNSS) like Global Positioning System (GPS) or Galileo. Because single signal systems without correction are not precise enough, augmentation methods were developed, which consist of additional hardware and software. This additional hardware adds to the overall cost and would increase the quantity price of a robot. The other idea was to recycle or even up-cycle old smartphones because a smartphone already comes with a GNSS receiver, sensors, and processing power while having a small power consumption.

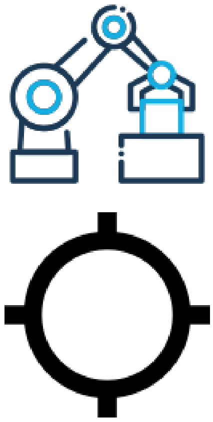


Figure 1.1: Robot Lost without GNSS

This led to the idea of implementing a system on Android that utilizes the GNSS receiver as well as sensors of the smartphone that then get sent to the robot. Multiple research questions were defined to find the best solution and will be described in the following Section 1.2.

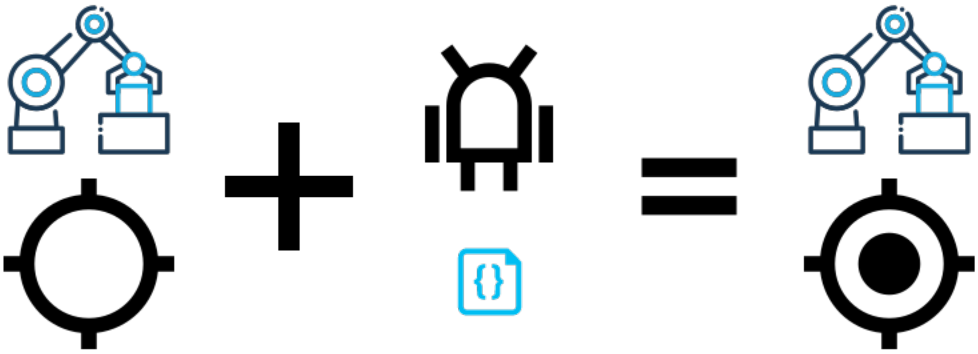


Figure 1.2: Robot with Android can localize itself

1.2 Research Questions

Multiple research questions were defined to find the best solution and set the cornerstone of this thesis.

- How effective is the use of an Android smartphone for the localization of an agricultural robot measured by resource consumption and accuracy?
- How different is the performance between smartphones with and without correction data?
- What is the performance gain (in accuracy) using an external GNSS antenna?

Multiple scenarios on two smartphones were selected to evaluate all questions based on availability and suspected performance gain. For a baseline, two Android smartphones were selected, the Oneplus 6T because it was my daily driver and the Samsung Galaxy S20 because of its Dual Frequency GNSS receiver and relatively low price point. Then an external receiver with an active antenna was tested to establish if a Software Defined Radio (SDR) on Android would work and its performance. Because the GNSS signal alone cannot create precise localization, correction data was applied, and a scenario for Real Time Kinematic (RTK) was created. The smartphone also has multiple sensors for Inertia Measurement Unit (IMU) data, which could also be used for sensor fusion to improve the signal further.

2 State of the Art

In the following chapter, other preceding research is reviewed and recapitulated. This should briefly overview the already established state of the art. Some topics deal with satellite-based localization on a budget, some topics go into more detail while other topics explore the use of Android or lasers for localization of robots.

2.1 Implementation of a multi-band RTK receiver system with Arduino

In [2], Willegger described a GNSS Receiver System consisting of a reference station and a rover. His thesis aimed to implement an affordable system with off-the-shelf electronic components while archiving RTK-level accuracy in the cm range. This was done with two μ-blox ZED-F9P boards, different active GNSS antennas, Long Range WAN (LoRaWAN) antenna and transceivers, and custom-made Printed Circuit Board (PCB)s. The reference station receives and decodes the GNSS signals via the GNSS module, then sends the correction data with National Marine Electronics Association (NMEA) packages via LoRaWAN over the air to the rover, which receives the signals and sends them to the GNSS module on itself. The GNSS modules contain a dual frequency receiver to be able to process L1C/A and L2C signals. The conclusion of his thesis was the successful implementation of a multi-frequency Differential GPS (DGPS) reference/rover system under 1000€ .

GNSS-SDR - SDR integration

In [3], Fernandez et al. showed that a SDR could be combined with their software GNSS Software Defined Receiver (GNSS-SDR) to receive GNSS signals and calculate localization data with it. The only problem they found was the reference clock of the receiver. They concluded that the results would be better with a better reference clock. They even proposed a calibration procedure for better accuracy. In the same year of the publication, in 2013, the RTL-SDR V3 has released with a Temperature Compensated Crystal Oscillator (TXCO), which improved the performance significantly, they later concluded in [4].

Dual Frequency GPS Receiver Implementation in GNSS-SDR

In [5], Danielle Skufca modified the GNSS-SDR suite to enable dual frequency GNSS receiving capabilities. Implementing ionospheric scintillation corrections for the pseudo ranges enables the localization with Precise Point Positioning (PPP) which allows much higher accuracy. The project is uploaded as a fork from the original software to GitHub [6] and can be used for testing. Unfortunately, the software wasn't merged back into its upstream, leading to many features missing compared to the current release.

2.2 Using Raw GNSS data on Android - GSA White Paper

In [7], the European GNSS Supervisory Authority thoroughly explains in the first chapter of the paper, the working principles of GNSS and gives calculation examples for signal decoding and error summation. In the second chapter, the operational structure of the GNSS module of the Android operating system before and after Android Version 7 (Nougat) is depicted. It shows how the GNSS signals traverse the software stack in both scenarios. Also in this chapter are different calculation examples for pseudo-range generation and time estimation. In this chapter's last section, the authors highlight the reader Android apps that already utilize the new Android feature for raw GNSS data. One is the RTCM converter described in Section 2.2.

GeolocPVT

In [8], Grenier and Renaudin showed that data on Android sub cm accuracy could be achieved with the introduction of raw GNSS. They compared different RTCM Streams, their data consumption, and the break-even point of accuracy and data usage for long-time dynamic tracking. The app is using GoGPS (see Section 3.3) to calculate and decode all the GNSS and RTCM signals. While the source code is open-source, the app's current state doesn't function anymore on more modern Android versions, at least on the phones I tried.

RTCM Converter

In [9], Privat et al. demonstrated the possibility of using an Android smartphone to receive raw GNSS measurements, encoding these measurements into the RTCM format, and sending these messages over an Networked Transport of RTCM via Internet Protocol (NTRIP) caster to a decoding software. With PPP WizLite, another app, they also showed that PPP was possible on a smartphone. Unfortunately, this app is no longer maintained and is a closed source, rendering it useless for further research. Fortunately, RTCM Converter still functions on modern Android systems but is also a closed source.

RTKRCV ROS

In [10], Ferreira et al. showed that with modification to RTKLIB (see Section 3.1), the output format of the software suite can be extended by ROS topics and messages. The authors also introduced a new output format for velocity estimates, which further down the processing stream can be used for better localization estimation with, e.g., a Kalman filter. This gives the advantages of the well-established software suite for RTK localization in combination with ROS modules specially designed for robotic use, e.g., robot_localization (see Section 3.4). The authors also intended the use of a modular antenna design and the possibility of a user-friendly reconfiguration allowing users to use the software in no time.

International Standard GNSS Real-Time Data Formats and Protocols

In [11], Heo et al. describe the different positioning techniques like RTK or DGPS and the protocols they use. The main discussed protocols are RTCM SC-104 and NTRIP. They reviewed the different "Addenums" of the RTCM standard and analyzed the message types. They also found that the most used protocol for single-base RTK systems is RTCM 3.0.

2.3 ROS Mobile

In [12], Rottmann et al. described and implemented a ROS Android application, which should display and send data from a mobile phone to a robot. The software was designed with a specific pattern in mind. The Model View ViewModel (MVVM) pattern enables developers to extend the existing code base easily. The app can be configured to implement different modules for interacting (publishing and listening) to ROS topics. They also showed different prototypes and the capability of their software with many examples, e.g., a joystick module to publish cmd_vel messages or a GPS module to track the robot's current position. This thesis used this extensibility to implement a IMU and GNSS streamer module. See Section 4.3.

Outdoor Positioning of Robots with Indoor Methods

In [13] Supper et al. used indoor application-specific sensors to drive their so-called "Mathilda" robot in an urban balcony garden without GNSS reception. They used the ROS package gmapping and Adaptive Monte Carlo Localization (AMCL) to create a map representing the plants as obstacles and the fence of the balcony as boundaries and to localize the robot with laser scanners with a probabilistic algorithm. The results showed, that if no GNSS is available and the position of every obstacle is known beforehand, laser scanners can localize the robot with an absolute distance error within $198.9\,\mathrm{mm}$.

3 Fundamentals

In the following section, important preceding topics and tools are explained. This includes the working principles of GNSS, SDR, Android, and ROS. The GNSS section gives a brief introduction to satellite-based navigation. The SDR section explains how a television receiver can be used to receive satellite signals, while the Android section goes into useful tools for robotic use. In the last section ROS, the software suite for robotics is briefly described.

3.1 GPS

Global Positioning System (GPS) is a satellite-based navigation system founded by the U.S. Department of Defense in 1973. The goal was to achieve a positioning system for military troops worldwide. In the 1980s, civilian use was permitted, which led to the navigation system we use today.

Mathematical Background

In [14], Bossert & Bossert describe the working principle of a satellite-based navigation system. In Equation (3.1) and Equation (3.2), they calculate the position of a receiver based on the propagation delay and the position of 3 satellites in a two-dimensional plane. The calculations are drastically simplified and do not account for signal delays, time dilation, or any other disturbance, but they give a good overview.

$$
\begin{aligned}
s_0 &= (x_0, y_0) && \text{Sat 0 Position} \\
s_1 &= (x_1, y_1) && \text{Sat 1 Position} \\
s_2 &= (x_2, y_2) && \text{Sat 2 Position} \\
s_n &= (x_n, y_n) && \text{Receiver Position} \\
&t_0, t_1, t_2 && \text{signal arrival times} \\
&\tau_0 && \text{signal sent time} \\
&c_0 && \text{speed of light}
\end{aligned}
\tag{3.1}
$$

$$\begin{aligned}\sqrt{(x_n - x_0)^2 + (y_n - y_0)^2} &= (t_0 - \tau_0) \cdot c_0 \\ \sqrt{(x_n - x_1)^2 + (y_n - y_1)^2} &= (t_1 - \tau_0) \cdot c_0 \\ \sqrt{(x_n - x_2)^2 + (y_n - y_2)^2} &= (t_2 - \tau_0) \cdot c_0\end{aligned} \tag{3.2}$$

The Equation (3.2) then need to be merged, e.g., with a Newton approximation, to get the final receiver position s_n.

DGPS

In [15] DGPS is described as an enhancement technology focusing on the augmentation of the received signals by a "rover" and "base station" pair. The rover moves freely while the base station remains geographically fixed with a known position. The main improvement this setup gains is the possibility to calculate error terms corresponding to the atmosphere and other disturbances. This is done by calculating the resulting drift of the fixed base station and correcting the rover signal by this drift if the base station is near the rover. This allows for corrections and precision up to the decimeter range.

RTK

While RTK is also a differential GNSS method, it improves the accuracy to be in a centimeter range. This is achieved by sending "code and carrier measurements" [16] from the base station to the rover. This allows the rover to fix the phase ambiguities of the incoming signals. This allows the rover to converge to a position. If a dual-frequency receiver is used, the convergence can be sped up.

NMEA 0183

The National Marine Electronics Association (NMEA) [17] is an organization with the goal of standardizing protocols for nautical and marine electronics. One of these standards is the NMEA 0183 standard, which describes electrical signals and communication data between systems. In [18], Langley describes the data format, which consists of human-readable strings starting with a "$" symbol and a short, two parted identifier of the transmitted package. The first half of the identifier describes the used satellite system, e.g., GP for GPS and GA for Galileo. The second half represents the respective package data that follows, e.g., "$GAGGA..." for a Galileo receiver, which transmits the "GGA"(GPS fixed data) package, which is one of the most used packages. NMEA 0183 can be parsed by multiple software e.g. Matlab [19].

RTCM SC-104

Radio Technical Commission for Maritime Services (RTCM) is an organization for standardizing protocols for marine telecommunication. One of their goals is to unify and standardize GNSS correction data. This standard was called Special Committee (SC)-104. In [11] Heo et al. compared Receiver Independent Exchange Format

(RINEX) with RTCM and the different versions of RTCMSC-104 standard. The first version with DGPS support was version 2.0, which focused on pseudo-range and correction data. Heo et al. also wrote that version 2.3 is still widely used for DGPS or single-base RTK operations. The main drawback, according to Heo et al. is the inflexibility of this format (lack of L1C or L5 support, Galileo support), version 3.0 was released with Galileo support. Heo et al. also analyzed the sent packages of RTCM SC-104 version 3.0 data over NTRIP, a network protocol to distribute GNSS correction data. In RTCM SC-104 version 3.2 Multiple Signal Message (MSM) support was added, to reduce sent packages.

MSM

In Radio Technical Commission for Maritime Services (RTCM) Special Committee (SC)-104 version 3.2 MSM support was added. The main benefit is an easier and more flexible way to send Galileo satellite observation in high precision[20]. The MSM-7 allows for "Transmission of a complete set of RINEX observations with extended resolution" (see [20] and RTCM STANDARD 10403.3), which allows post-processing software to be more aligned with RINEX version 3.0 data, simplifying the data handling.

NTRIP

Networked Transport of RTCM via Internet Protocol (NTRIP) was developed by the German Bundesamt für Kartographie und Geodäsie (BKG)[21] for the transport of RTCM data over a network. The idea was to develop a method to transport GNSS data over Hypertext Transfer Protocol (HTTP) streaming standard while being open source and non-proprietary. The protocol defines three roles of communication partners. The NTRIP caster, the actual HTTP like server, accepts both NTRIP server and client requests and handles the transmission of the data. The NTRIP server, which publishes data to a specified topic, so-called "mountpoints", for clients to receive them and clients, which subscribe to these mountpoints.

YCCaster - NTRIP Caster

In [22] the developers of Hedgehack developed an application that receives, stores, and transmits NTRIP data. This type of server is called an NTRIP Caster. A caster is needed as a middleman between the rover and the calculation unit. The developers allow the use of the caster for personal and academic use.

APOS

The Austrian Positioning Service (APOS) is a service of the Austrian Bundesamt für Eich- und Vermessungswesen (BEV) [23] which offers RTCM correction streams. The service is sold commercially as well as free of charge for research or farming applications. The BEV offers an NTRIP caster with HTTP-basic authentication for user management with different mountpoints in different price ranges, depending on the needed accuracy and time of use. The data for these mountpoints are calculated based on fixed placed GNSS receivers and the coordinates the client needs to initially send at the beginning of the connection. This technique is called Virtual Reference Station (VRS).

RTKLIB

rtklib is open-source software written by Takasu et al. [24] for precise positioning with generic GNSS receivers. The software became the de facto standard for open-source projects with the requirement of high-precision positioning. Multiple other applications use this software as a backbone e.g. see Section 2.2. The software has two sets of software, one with Graphical User Interface (GUI) and one specifically designed for headless, server applications. The software implements different solving algorithms for positioning with different precision (see Listing 3.1). This is archived with model-based correction based on Troposphere and Ionosphere error models.

Listing 3.1: RTKLIB Quality Table (taken from [25])

```
1: Fixed, solution by carrier-based relative positioning and the
integer ambiguity is properly resolved.
2: Float, solution by carrier-based relative positioning but the
integer ambiguity is not resolved.
3: Reserved
4: DGPS, solution by code-based DGPS solutions or single point
positioning with SBAS corrections
5: Single, solution by single-point positioning
```

GNSS-SDR

GNSS Software Defined Receiver (GNSS-SDR) is a software developed by Fernandez et al. [26] and is an open-source software for exploring GNSS signals and all processing stages needed to calculate an accurate position (see Section 2.1). The software is sectioned into different modules that can be controlled and configured individually and swapped with different modules. This allows for custom modules as well as for a variety of different configurations. Multiple different input modules are available, one of which is a module to interface with SDR. The software is written in C++, which allows for cross-compilation.

3.2 SDR

Software Defined Radio (SDR) is a Radio Frequency (RF) receiver that converts RF signals into In-Phase & Quadrature (I/Q) signals. The use cases range from television reception to satellite communication, which mostly lies in the capability to control the receiver with software, hence the name Software Defined Radio (SDR).

I/Q Signals

According to Franks [27, p. 82] and [27, p. 196] every signal can be represented in I/Q form as shown in Equation (3.3). The I component of the signal is the cos term on the left side and the Q component is represented by the right part.

$$A(t) \cdot cos(2\pi ft + \theta(t)) = cos(2\pi ft) \cdot A(t) \cdot \theta(t) - sin(2\pi ft) \cdot A(t) \cdot \theta(t) \tag{3.3}$$

According to different sources([28], [29] and [30]) the main benefit of I/Q signals is the easy post-processing of the signal while still having phase information of the signal. This is needed because almost all RF modulations rely on phase information. Especially for phase-dependent measurements, like GNSS signals or Quadrature Amplitude Modulation (QAM) signals, this information is crucial. With this representation, multiple modulation techniques can be decoded.

rtl-sdr Software

Osmocom developed software called rtl-sdr[31] to interface with DVB-T USB dongles to receive RF signals. These USB devices are based on the RTL2832U[32] microchip. The software became the de facto standard for hobbyists in amateur radio. It allows the user to send the received signals to different applications over Transmission Control Protocol (TCP) or sockets as well as to control the peripheral.

RTL-SDR V3 Dongle

The RTL-SDR V3 USB peripheral[33] is a successor of DVB-T sticks explicitly designed with amateur radio applications in mind. The two biggest improvements compared with other systems are the integrated Bias-T as well as the Temperature Compensated Crystal Oscillator (TXCO). With this addition connecting active antennas became much easier as well as receiving time-critical signals became possible (see Section 2.1).

3.3 Android Tools and Libraries

Android as an operating system for smartphones comes with a variety of libraries from Google and third-party developers. To debug applications also multiple tools for the platform exist. Most smartphones come with multiple sensors which can be utilized by the Android Application Programming Interface (API). One of these sensors is a multiple sources GNSS receiver which got raw data API specifically for precision and research-related applications, as in Section 2.2 ([7]) described.

Google Measurement Tools

In [34] google developed a test framework for GNSS measurement analysis for Android devices. The repository consists of an Android application called GNSSLogger and NMEAUtils which are written in Matlab. The Matlab scripts calculate multiple statistics given an NMEA file, which can be generated in the Android app.

Termux

Termux [35] is a Linux simulator for Android which supports different distributions and allows for cross-compilation of software. It simulates a Linux shell without rooting the phone, and therefore without voiding the warranty of the smartphone. All requirements can be downloaded and are managed by Advanced Packaging Tool (APT), the software managing tool of Debian-based distributions. To specify which distribution is used, PRoot Distro was written to simplify the setup process.

PRoot Distro

The software PRoot Distro [36] allows the installation of custom kernel and Linux distributions over the command line. This is needed because most software packages have dependencies and need a specific distribution to work properly.

GoGPS

GoGPS [37] is an open-source project to analyze GNSS signals based on Matlab and on Java[38]. While the Matlab version is still being developed, the Java version of this software was abandoned by the developers. The Java version was a key component of the Android software GeolocPVT (see Section 2.2) and was integrable into an Android app.

Battery Historian

Battery Historian [39] is an application written by Google to analyze the power consumption of running apps on Android based on the system debug report. The application is started with docker and exposes a HTTP website on a specified TCP port. On the website, all running applications with processor utilization and power consumption can be analyzed. All permissions and peripheral usage are shown, as well as network performance. This allows Android developers to experiment and improve their software and to locate problems with other simultaneously running apps.

3.4 ROS

Robot Operating System (ROS) is a software ecosystem written by Quigley et al. [40] to allow developers to write decentralized software for robots. The communications between modules, the building block for robot operations, is done via a publisher-subscriber pattern managed by so-called "Topics". ROS became the de-facto standard for academic robots because of the support in the community and its open-source nature. The tutorials for newcomers are easy to follow, which lowers the initial hurdle for developers.

REP

On the ROS website [41] the developers declare a format for standardization after the model of Pythons PEP, the ROS Enhancement Proposals (REP). This standardization is needed because ROS is based on a module written by multiple developers and without a clear interface definition and data representation the inter-working of these modules wouldn't be possible. The most important REPs for data handling are REP103 and REP105.

REP 103

REP103[42] with the title "Standard Units of Measure and Coordinate Conventions" defines all units and coordinate conventions as the name implies. This is important because all sensors, e.g. for localization, need to have a coordinate reference system to define the position of the sensor on the robot itself. Also, the way to describe the variance and covariance of a sensor is defined.

REP 105

In REP105[43] with the title "Coordinate Frames for Mobile Platforms" the way how to coordinate frames need to be specified is defined. This is needed, to ensure the correct position transformation of parts of the robot to their position regarding the world map or the start of operation.

Protobuf

Protobuf is used for the serialization and deserialization of data. The library comes with a code generator binary, which generates classes in a selected language (e.g. C++ or Java) given an abstract message format. Multiple other open-source projects use Protobuf for data transfer out of their software to give developers an easy way to extend and use the library.

robot localization

In [44] Moore and Stouch presented a ROS module which utilizes a Extended Kalman Filter (EKF) for sensor fusion to allow mobile robots to estimate a better overall state given multiple different data sources. They showed that sensor fusion of IMU and GNSS data leads to better localization of the robot, hence the name of the ROS package ros_localization.

Kalman Filter

In [45] Kalman described a state estimator that uses the probability distribution of different (sensor) signals to estimate the correct values. According to [46] the state estimator works, because values with a lower difference between them have a smaller standard deviation, indicating a better observation of the actual measured value. The whole filter relies on an iterative process to constantly improve the estimated state of the system. The whole filter utilizes five key algorithms to archive its functionality. For a more detailed explanation, see [46].

Extended Kalman Filter An Extended Kalman Filter (EKF) is a nonlinear version of a Kalman filter [47] according to Ribeiro. The extended version linearizes the inputs to work even without linear inputs. This is needed because GNSS signals typically are of non-linear nature.

4 Proposed System

In this chapter, the measurement setup for accuracy and power, the different scenarios, and the underlying components are discussed and explained. In the scenario overview, the full name of each scenario is explained and then shortened for easier recognition throughout the thesis. Every scenario Is then explained in more detail with a block diagram as overview and code snippets for the essential parts of the system.

4.1 Setup Cost Analysis

For the cost analysis, multiple scenarios were assumed. In Table 4.1 the different scenarios are evaluated regarding cost. The costs are taken from Geizhals, a price comparison website, and Refurbed, a website for refurbished devices. The smartphone selection was mainly driven by my personal daily driver phone (Oneplus) as well as a modern yet affordable Samsung one with a dual frequency GNSS receiver. The RTL-SDR V3 can be bought cheaper from a reseller but might be a replica or fake one so we used a trusted source.

The reference system consists of a ZED-F9P RTK and a u-blox ANN-MB-00 antenna, which is the minimum hardware to receive GNSS signals. The cost of this system doesn't include the extra time and hardware to integrate the module into a system, nor does it include a computer platform to use or relay the data.

	Scenario A	Scenario B	Scenario C	Scenario D	Scenario E	Reference System
Oneplus 6T 176,99€	✓		✓			
Samsung Galaxy S20 Plus Refurbed: 290,99€ (New: 549,99€)		✓		✓	✓	
RTL-SDR V3 59,99€			✓			
ZED-F9P RTK 209,99€						✓
u blox ANN-MB-00 56,40€			✓			✓
APOS Subscription Free (200€ per month, if not for farming or academic use)				✓	✓	
Total Cost	176,99€	290,99€	293,38€	290,99€	290,99€	266.39€

Table 4.1: Cost Matrix

Measurement Setup

For the measurements of all scenarios described in Section 4.2, the smartphones were placed alone on a hard surface, and the measurement was started. The weather conditions were nearly the same on both locations and days. The first measurement location was a tree stump inside my garden; the second was on the roof of the electrical engineering faculty at TU Wien.

Localization Accuracy

For accuracy, the tool "measurement tools" from Google [34] (see Section 3.3) was used. Because of the different origins of the NMEA files, only the Matlab files from the repository were adapted and used. The NMEA 0183 file format was selected as a comparison format because it has a lot of information for post-processing while still being easy to collect from the different platforms. The Matlab scripts were uploaded to Matlab Online[48] for easier file handling between systems.

4.2 Scenario Overview

The following sections describe the multiple scenarios, and important implementation details are discussed. The scenarios were labeled to improve readability. The labels are:

- Scenario A: Oneplus GNSS data via ROS Mobile to ROS (see Figure 4.1)
- Scenario B: Samsung GNSS data via ROS Mobile to ROS (see Figure 4.1)

- Scenario C: Oneplus GNSS-SDR with SDR and active antenna to ROS (see Figure 4.2)
- Scenario D: Samsung RTCM stream via rtkcrv to ROS (see Figure 4.3)
- Scenario E: Samsung RTCM stream via rtkcrv to ROS with robot_localization module with EKF (see Figure 4.4)

4.3 Scenario A and B

In Figure 4.1, both Scenario A and B are depicted. They share the same experimental setup, only with different phones as the main component. On the smartphone, the modified version of ROS Mobile (see Section 4.3) runs and sends GNSS and IMU data via TCP to the ROS master. A NMEA file was saved for comparison with other setups. This was done with the app GNSSLogger described in Section 3.3.

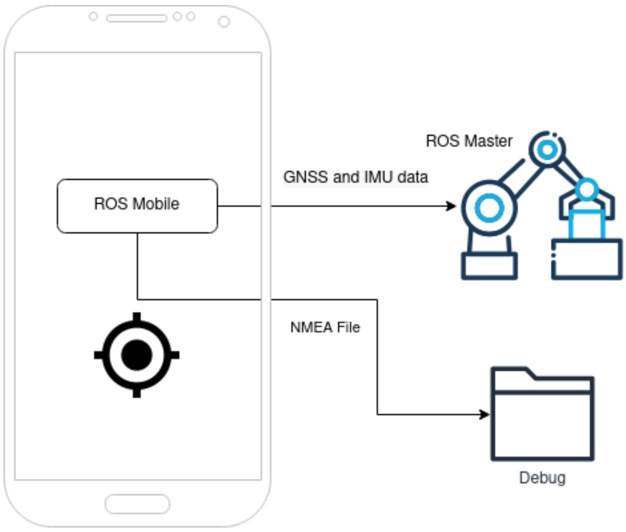


Figure 4.1: Scenario A and B: Overview

ROS Mobile

ROS Mobile, as described in Section 2.3, is an Android app for communication with a ROS robot. It builds upon ROSJava, a ROS interface, and implementation to interact with the ROS framework in the Java programming language. This is needed because ROS is written in C++ and Python, while Android only supports Java. The app was developed with easy expandability in mind. The developers created a tutorial on how to create new so-called "widgets". During the development of the widgets, both REP 103 (see Section 3.4) and REP 105 (see Section 3.4) were consolidated for correct unit handling and data creation. To ensure the correct interoperability with other ROS modules, e.g., the robot_localization module, it was needed to comply with these standards.

4.3.0.1 Custom Widget - GNSS Sender

To extract the sensor data for GNSS from the Android API, a custom ROS Mobile widget was created. The widget was written in Java and added to the existing source code inside a private git branch. The main challenge, the interfacing, and networking with ROS, was already solved by the creators of ROS Mobile, which is why this framework was chosen. The main task to transfer GNSS data from the smartphone to the robot was to register a "LocationListener", an Android API defined Listener, and to convert the resulting data into a ROS message, both shown in Listing 4.1.

Listing 4.1: Code Snippet GNSS Widget

```
public void onLocationChanged(Location location) {
    GPSSenderData data = new GPSSenderData(location.getLatitude(), location.
       getLongitude(), location.getAltitude(), location.getAccuracy());
    data.setTopic(topic);
    rosDomain.publishData(data);
    }
...
public Message toRosMessage(Publisher<Message> publisher, BaseEntity widget) {
NavSatFix message = (NavSatFix) publisher.newMessage();
message.setLatitude(this.latitude);
message.setAltitude(this.altitude);
message.setLongitude(this.longitude);
message.getStatus().setStatus(NavSatStatus.STATUS_FIX);
message.getStatus().setService(NavSatStatus.SERVICE_GPS);

message.setPositionCovarianceType(NavSatFix.COVARIANCE_TYPE_APPROXIMATED);
double covariance = deviation*deviation;
double[] tmpCov = {covariance,0,0, 0,covariance,0, 0,0,covariance};
message.setPositionCovariance(tmpCov);
...
```

4.3.0.2 Custom Widget - IMU Sender

For sensor fusion and to utilize multiple sensors of the Android system, an IMU sender widget was also created. The framework ROS Mobile was already multithreaded, which made the parallel execution of both widgets easier. The ROS IMU message consists of three different Android sensors, an accelerometer, a rotation sensor, and a gyroscope, all with different refresh rates, which made the data flow handling difficult. This was solved by only sending a ROS IMU message when all data was available, discarding all other data if a sensor has a much higher refresh rate. To mitigate this, the same sensor delay was chosen for all sensors. Fortunately, the Android API already comes with functions to calculate vectors and quaternions, simplifying the data conversion immensely. The code snippet in Listing 4.2 provides a brief overview of the code.

Listing 4.2: Code Snippet IMU Widget

```
...
mSensorManager = (SensorManager) getContext()
                    .getSystemService(Context.SENSOR_SERVICE);
Sensor mSensorAcc = mSensorManager
                    .getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
Sensor mSensorRot = mSensorManager
                    .getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
Sensor mSensorGyr = mSensorManager
                    .getDefaultSensor(Sensor.TYPE_GYROSCOPE);

mSensorManager.registerListener(imuEventListener, mSensorAcc,
                                SensorManager.SENSOR_DELAY_NORMAL);

mSensorManager.registerListener(imuEventListener, mSensorRot,
                                SensorManager.SENSOR_DELAY_NORMAL);

mSensorManager.registerListener(imuEventListener, mSensorGyr,
                                SensorManager.SENSOR_DELAY_NORMAL);

...
public boolean readyToSend(){
    return gyro.readyToSend() && rot.readyToSend()&& acc.readyToSend();
}
...
public Message toRosMessage(Publisher<Message> publisher, BaseEntity widget) {
IMUSenderEntity senderWidget = (IMUSenderEntity) widget;

Imu message = (Imu) publisher.newMessage();
message.setAngularVelocity(gyro.getVector()
                    .toVector3Message(message.getAngularVelocity()));
message.setOrientation(rot.getQuaternion()
                    .toQuaternionMessage(message.getOrientation()));
message.setLinearAcceleration(acc.getVector()
                .toVector3Message(message.getLinearAcceleration()));
...
```

4.4 Scenario C

In Figure 4.2, only the IMU data is sent to the robot via ROS Mobile, while the GNSS information is received with an active antenna, an SDR, and GNSS-SDR. The android smartphone uses a driver called SDR driver from Signalware Ltd [49] to communicate with the SDR and to send the received I/Q (see Section 3.2) data via TCP to GNSS-SDR. The software stack then calculates the estimated position and sends it to ROS over a custom, with a protobuf serialized data format. There is a particular ROS module (see Section 4.4) that decodes and deserializes the message and publishes the location to the correct ROS topic.

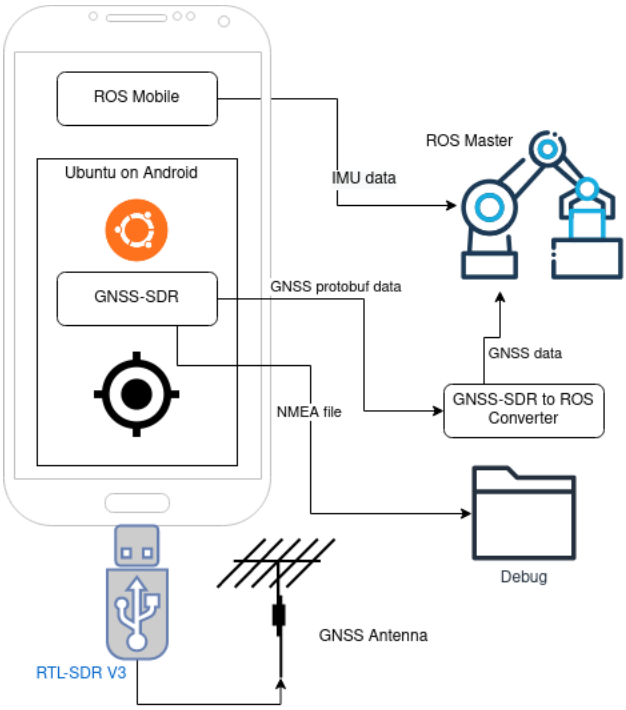


Figure 4.2: Scenario C: Overview

Android Ubuntu

For GNSS-SDR to compile properly, a Ubuntu system was needed. This is caused by the many different dependencies of libraries to compile the software. This led to the usage of termux (see Section 3.3), a Linux distribution command shell simulator. This allowed the use of APT, a package manager for Debian-based distributions, to install the compiler and all dependencies for arm64, the Central Processing Unit (CPU) architecture of most Android smartphones. To ensure the correct distribution was configured, PRoot Distro (also see Section 3.3) was used to create a Ubuntu 20.04 system needed for GNSS-SDR. Unfortunately, drivers for Universal Serial Bus (USB) peripherals are not supported inside the simulated Linux environment, which led to the need for a network-enabled Android rtl-sdr driver.

Android rtl-sdr driver

rtl_tcp_andro- [50] is an Android application to handle the interfacing with USB-based SDR devices and sending the data over TCP to another application. This app was chosen because it uses the rtl_tcp data format, widely supported by GNSS applications, and is compatible with a GNSS-SDR input module. The driver and TCP connection can be started from the termux command line with a command depicted in Listing 4.3. The software registers the "iqsrc://" protocol to start with the driver's "Intent", an Android background task. The port must match the configuration of GNSS-SDR, and the frequency must be set to $1\,575\,420\,000\,\text{Hz} = 1.575\,42\,\text{GHz}$ the frequency of the L1C band of GPS. The sampling rate is set to $2\,\text{MHz}$, which is recommended in the tutorial of GNSS-SDR [4].

Listing 4.3: driver start command

```
am start "iqsrc://-a 127.0.0.1 -p 14423 -f 1575420000 -g 0 -s 2000000"
```

Antenna

In [2, p. 95], Willegger concluded that "The highest accuracy was gained with the Taoglas A.80 antenna and the second best was with the μ-blox ANN-MB antenna." Due to the significant price difference between the two antennas, the cheaper one, the μ-Block ANN-MB was chosen. The antenna is an active one, which led to the need for a bias tee, a power supply, by offsetting the RF signal by an Direct Current (DC) offset. This led to the use of the RTL-SDR V3, which comes with an integrated bias tee.

GNSS SDR

The software GNSS-SDR was downloaded into the termux environment and all dependencies were installed. Then the software was compiled which took approx. $30\,\text{min}$. The software then was configured with the configuration of the example for rtl_tcp input. A small snippet of the configuration can be seen in Listing 4.4 (see full config in Listing A.2). The "SIGNAL_SOURCE" module supports the RtlTcp signals sent by the rtl_tcp_andro- software, which sends the I/Q signals received as complex values via TCP on port 14423 (must be the same as defined in Listing 4.3) over the network.

Listing 4.4: Portion of GNSS-SDR Configuration

```
;######### SIGNAL_SOURCE CONFIG ############
SignalSource.implementation=RtlTcp_Signal_Source
SignalSource.address=127.0.0.1
SignalSource.port=14423
```

After a successful sanity test on a Linux machine with the RTL-SDR V3 and active GNSS antenna directly connected to it, a way to extract the data and send them to ROS was needed. Fortunately, GNSS-SDR defines several interfaces to interact with other software. The output module, called PVT, of the software stack defines a Protobuf interface which allows generating software to receive with the help of a code generator (Protobuf definition file can be seen in Listing 4.5). The last piece was the data transfer from this interface to ROS. This was done in a custom ROS module written explicitly for this task.

Custom GNSS-SDR ROS module - Protobuf deserializer

The custom ROS module was necessary because GNSS-SDR only defines files and GNSS receiver-related streams as output. Only the Protobuf interface can quickly receive the data in another application. The Protobuf definition file snippet can be seen in Listing 4.5.

Listing 4.5: Portion of GNSS-SDR Protobuf(included in [4])

```
double latitude = 17;  // Latitude, in deg. Positive: North
double longitude = 18;  // Longitude, in deg. Positive: East
double height = 19;  // Height, in m
```

With the help of the Protobuf code generator (see Section 3.4), a message class can be generated in C++ code. This code, in combination with the Protobuf library, can then be included in the compilation process of the ROS module to allow to receive the GNSS-SDR data over TCP. The GNSS-SDR data is then available and can be processed and sent as a ROS message via ROS topic. A small code snippet of the reception and converting can be seen in Listing 4.6. The class "gnss_sdr::MonitorPvt gnss_msg;" is the auto-generated Protobuf class and "sensor_msgs::NavSatFix ros_msg;" is the outgoing ROS message.

Listing 4.6: Code Snippet GNSS-SDR to ROS Converter

```
gnss_sdr::MonitorPvt gnss_msg{};
gnss_msg.ParseFromString(message_);
sensor_msgs::NavSatFix ros_msg;

ros_msg.latitude = gnss_msg.latitude();
ros_msg.longitude = gnss_msg.longitude();
ros_msg.altitude = gnss_msg.height();
ros_msg.position_covariance = {
gnss_msg.cov_xx(), gnss_msg.cov_xy(), gnss_msg.cov_zx(),
gnss_msg.cov_xy(), gnss_msg.cov_yy(), gnss_msg.cov_yz(),
gnss_msg.cov_zx(), gnss_msg.cov_yz(), gnss_msg.cov_zz()};
```

4.5 Scenario D

In Figure 4.3, a different approach was taken by streaming raw GNSS observation data via RTCM Converter to a custom ROS module called rtkrcv_ros (see Section 2.2). The Android app connects to a NTRIP caster via

TCP and acts as a NTRIP server. The ROS module connects to this caster as the client and to two different casters for correction data as well as ephemeris data (see Section 5.2.1). The module then calculates a location estimation based on all the data received from the streams with a certain quality/uncertainty (see Listing 3.1). The result is then published as ROS topic for other modules to be used.

Because of some missing RTCM MSM data on the Oneplus smartphone, only the Samsung smartphone was usable for this scenario (see Section 5.2.1).

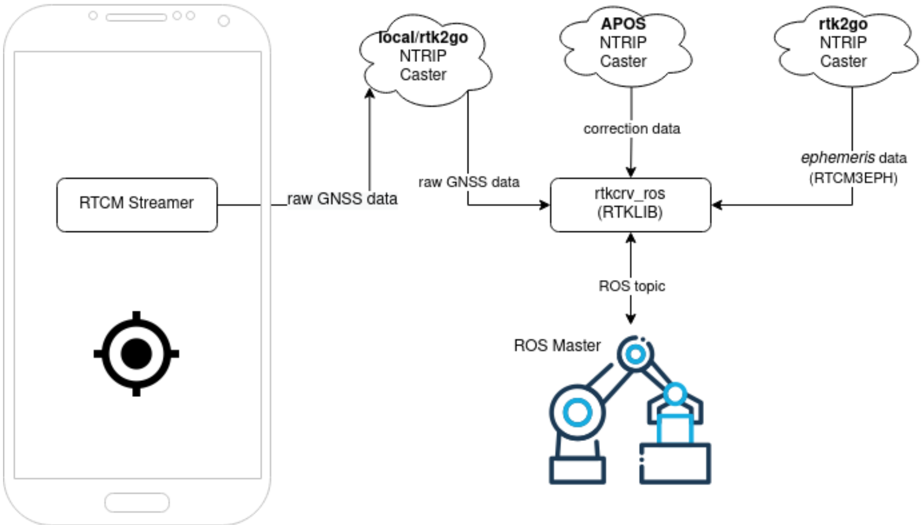


Figure 4.3: Scenario D: Overview

RTCM Streamer

In [9], Privat et al. present an Android app that uses the introduced raw GNSS functionality of Android (see Section 2.2 and Section 2.2) to sent raw GNSS observations over NTRIP to a caster. In Scenario D, this caster was the self-hosted one called YCCaster. The application needs to be configured with the casters IP and port as well as the used mountpoint. The app acts as a server in the NTRIP standard because it publishes data to a mountpoint.

Other NTRIP Mountpoints

Because the rtklib algorithm needs RTK correction data, the Austrian Positioning Service (APOS) mountpoint was added. An estimate must be sent at connection time to get correction data valid for the rover's position to allow APOS to calculate the correct correction data. This is configured in the configuration of the rtkcrv_ros module. Also, a third NTRIP mountpoint with ephemeris data was added. This data describes the positioning of the satellites, which is needed to calculate the position of the rover based on signal run-time.

YCCaster

Most NTRIP casters are only commercially available, which was not feasible for a low-cost robot. Other casters only allow sending production data and no debugging or developing clients. If a client makes too many reconnects, it gets banned from the caster, which is also not ideal. Hosting the caster on a private server with free software is the best possibility. This is doable with YCCaster[22], which can be configured to be password protected if the robot must connect from an external network. The free version only supports limited simultaneous connections, which doesn't impact a single demonstration robot.

rtkcrv ROS module

With rtkcrv_ros Ferreira et al. [10](also see Section 2.2) made the librtk tool suite usable for ROS. This modified version of the binary allows the output to be a ROS topic. The configuration only differs in the output parameters, as seen in Listing 4.7.

Listing 4.7: rtkcrv config snippet

```
inpstr1-type       =ntripcli
inpstr2-type       =ntripcli
inpstr3-type       =ntripcli
inpstr1-path       =user:pass@localhost:2101/MY_MOUNTPOINT
inpstr2-path       =APOS_USER:APOS_PW@217.13.180.215:2201/APOS_Standard
inpstr3-path       =user@pass@ntrip.use-snip.com/RTCM3EPH
inpstr1-format     =rtcm3
inpstr2-format     =rtcm3
inpstr3-format     =rtcm3
inpstr2-nmeareq    =latlon
inpstr2-nmealat    =48.210033
inpstr2-nmealon    =16.363449
outstr1-type       =ros       # CUSTOM OUTPUT TYPE
outstr2-type       =file
outstr1-path       =gps       # ROS TOPIC
outstr2-path       =rtkrcv.nmea
outstr1-format     =vel_enu   # CUSTOM OUTPUT FORMAT
outstr2-format     =nmea
```

4.6 Scenario E

In Figure 4.4, scenario D was copied and expanded by a robot_localization ROS module which performs signal filtering by utilizing an EKF and the IMU data from the phone. For the correct data format, the rtkcrv module was modified to generate NMEA messages as output, and a module, was added to convert these NMEA messages into ROS GPS-Fix messages consumable by the robot_localization module.

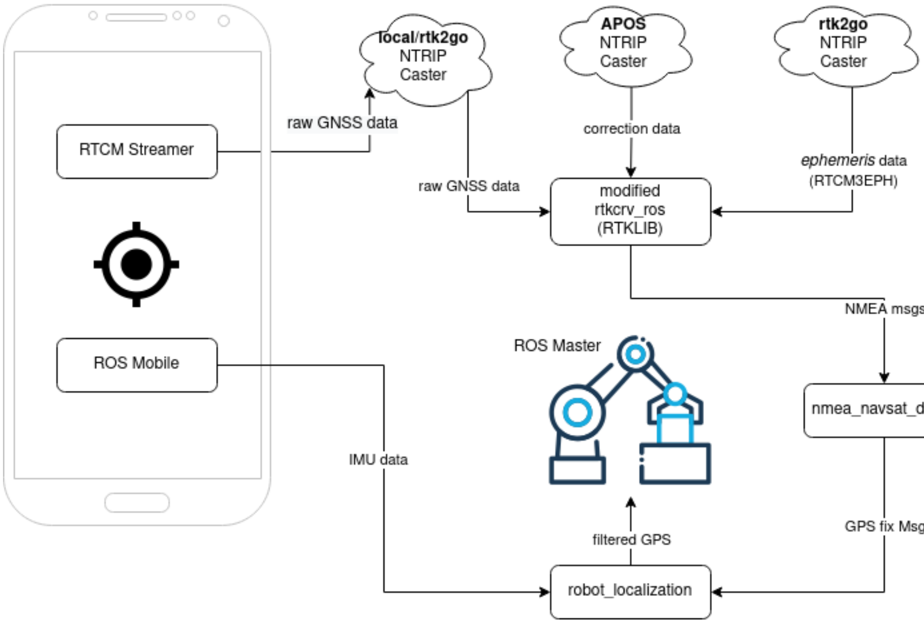


Figure 4.4: Scenario E: Overview

Customized rtkcrv ROS module

Because robot_localization only accepts the NavSat_Fix message format and rtkcrv_ros only supports a custom output message, the code was modified, and a new ROS module was added to accommodate these discrepancies. The new format is from the type "NMEA_Sentence" which is published on the ROS topic "nmea_sentence" which then gets consumed by the added ROS module nmea_navsat_fix. The new config for the modified module can be seen in Listing 4.8 (other fields are the same as Listing 4.7).

Listing 4.8: Custom parameter for rtkcrv

```
outstr1-type   =ros            # ROS OUTPUT TYPE
outstr1-path   =nmea_sentence  # ROS TOPIC
outstr1-format =nmea_psv       # MY CUSTOM ROS FORMAT
```

ROS NMEA driver

After the newly generated output of the rtkcrv module is from type "NMEA_SENTENCE", a NMEA parser was needed to convert the data into a NavSat_Fix, needed by the robot_localization module. This module only requires the input and output topic specified during boot, shown in Listing A.1.

ROS Localization

The last step in the data flow is the filtering step, which is performed by the robot_localization module written by Moore et al. [44] (also see Section 3.4). This ROS package consists of multiple modules. A module for GNSS filtering and for sensor fusion. These modules need three different ROS messages as inputs, IMU data, Odom data, and GNSS data. For the configuration of these modules, the example file "dual_ekf_navsat_example.yml" was modified and used, as well as the Odom data simulated (see Listing A.1). The other two inputs came from the ROS Mobile app and from the nmea_navsat_fix module. The robot_localization then estimates the position and the results are published at a ROS topic for further use.

5 Results and Discussion

In this chapter, the scenarios' results will be discussed, and some problems and difficulties during the development and implementation phase are summarized. In the first part, the results of every scenario are analyzed and then compared. In the discussion section, a summary of the results is given, and obstacles and failures are shortly described.

5.1 Results

This section evaluates and compares the measurement results of the five different scenarios. All scenarios consist of multiple figures representing various errors and error distributions, while other figures show power consumption and the position of the measurement in relation to a satellite map.

5.1.1 Scenario A and B

As in Figure 4.1 depicted, the measurement setup of Scenario A and B is the same, only on two separate phones, for comparison both measurements were taken at the same time to ensure the optimal comparability of both collected data sets. For this, the software GNSS Logger was started simultaneously, and an NMEA file was created. This file was processed with the Google Measurement Tools suite (see Section 3.3) multiple graphics were created. For a better visual representation of the data, a map with both data sets was also generated; see Figure 5.5.

5.1.1.1 Secenario A

Figure 5.1 shows the Longitudinal and Latitudinal error of all measurement points to the actual position. All measurement points are temporally separated by the measurement frequency of the Android phone in connection with other delays in the measurement chain. The curve of the line indicates a drift of the position in both latitude and longitude directions.

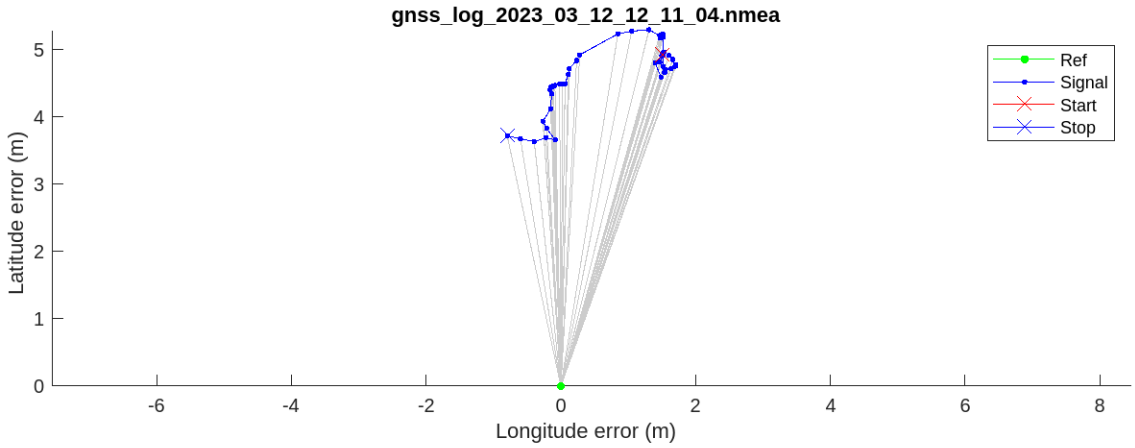


Figure 5.1: Oneplus Static

Figure 5.2 shows the Cumulative distribution function (CDF) of the error, which indicates A) an offset to the true position, visible at the immediate jump on the left-hand side at approx. 3.65 m. The noticeable step at approx. 4.5 m shows a stable phase during the measurements. 50 % of all measurements, also called the Circular error probable (CEP), are within a range of 5.02m. 95 % of all measurements are within a radius of 5.45 m.

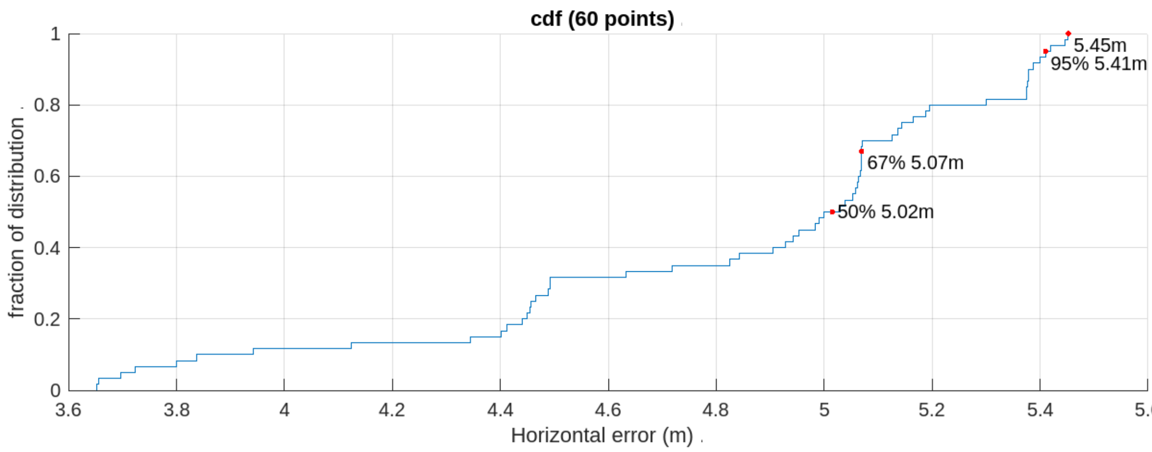


Figure 5.2: Oneplus Static CDF

Table 5.1 shows the power consumption of the Oneplus 6T smartphone during GNSS measurement. The power consumption was extracted from the Android debug report and analyzed with battery historian (see Section 3.3). Because of fluctuation, the average values during the measurement were taken.

avg. Voltage	avg. Discharge	avg. Power
4.07 V	875 mA	3.561 W

Table 5.1: Scenario A Power Consumption

In Table 5.2, the mean and standard deviation of the longitudinal and latitudinal signals in degrees and the mean and standard deviation of the distance error in meters can be seen. The horizontal components of the distance error are shown, as well as the absolute distance error marked as Dist in the table. The mean distance error of $4.82\,\mathrm{m}$ and a standard deviation of $0.513\,\mathrm{m}$ show a subpar accuracy of scenario A.

Lon/Lat/Dist	Mean[deg]	Std[deg]	Mean[m]	Std[m]
Lat	48.251579	3.921e-06	4.673	0.436
Lon	16.349675	1.132e-05	0.898	0.841
Dist	X	X	4.824	0.513

Table 5.2: Statistical measures of Longitudinal and Latitudinal measurements

5.1.1.2 Scenario B

Figure 5.3 shows the Longitudinal and Latitudinal error of all measurement points to the actual position. The measurement starts not far from the correct position but drifts off by multiple meters over time. The path of the measurements follows roughly a line, which indicates bad longtime behavior.

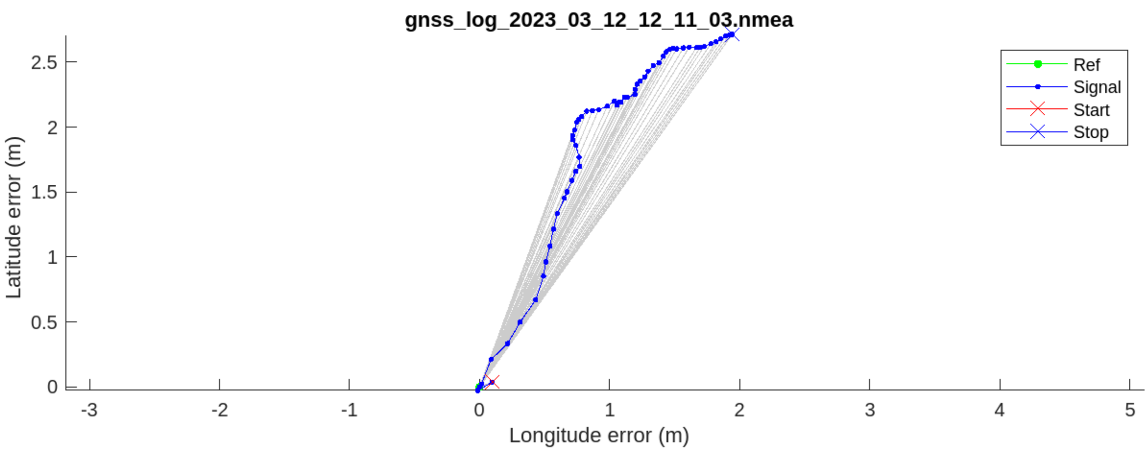


Figure 5.3: Samsung Static

In Figure 5.4, the CDF shows gradually increasing error. This continuously increasing error without steps indicates no stable position during the measurement. $50\,\%$ of all measurement points are within a $2.39\,\mathrm{m}$ radius. $95\,\%$ are in a radius of $3.26\,\mathrm{m}$ and the biggest error recorded was $3.34\,\mathrm{m}$ off the target.

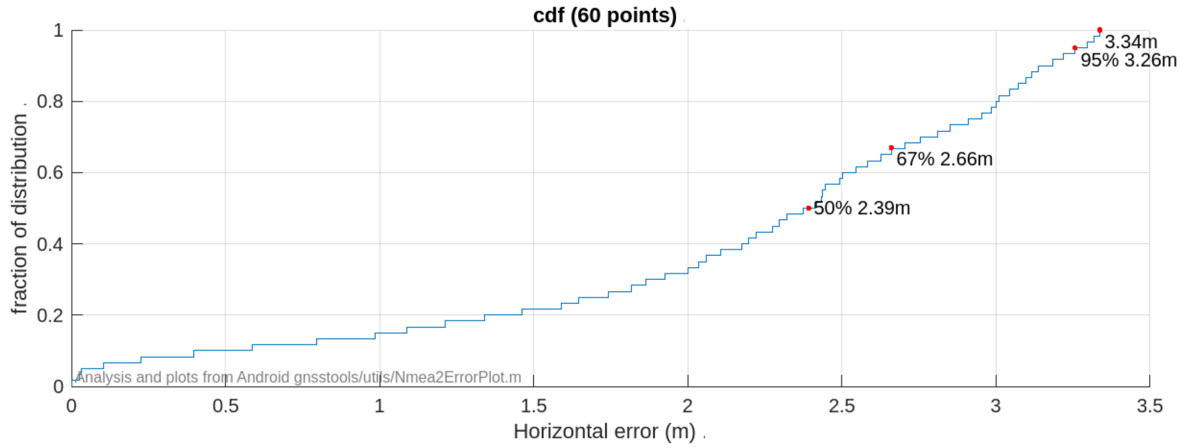


Figure 5.4: Samsung Static CDF

Table 5.3 shows the measured power consumption. The measuring procedure was identical to the measurement of Table 5.1, which occurred simultaneously. The extracted debug report was also analyzed with battery historian (see Section 3.3).

avg. Voltage	avg. Discharge	avg. Power
4.1 V	1074 mA	4.403 W

Table 5.3: Scenario B Power Consumption

In Table 5.4, the mean and standard deviation of the distance error and the longitudinal and latitudinal error can be seen. The distance error means of 2.15 m with a standard deviation of 0.947 m compared with Table 5.2 of scenario A shows a better absolute accuracy, but worse repeatability.

Lon/Lat/Dist	Mean[deg]	Std[deg]	Mean[m]	Std[m]
Lat	48.251553	7.258190e-06	1.893	0.807
Lon	16.349676	7.128428e-06	1.006	0.529
Dist	X	X	2.152	0.947

Table 5.4: Statistical measures of Longitudinal and Latitudinal measurements

In Figure 5.5, both Figure 5.1 and Figure 5.3 are plotted on a satellite photograph of the position. The small green dot at the start of the red line was the actual position of both smartphones. The difference in accuracy between scenario A (blue), the Oneplus phone, and scenario B (red), the Samsung phone, is clearly visible. While red starts at the correct location and drifts away over time, blue doesn't even find the correct position at all.

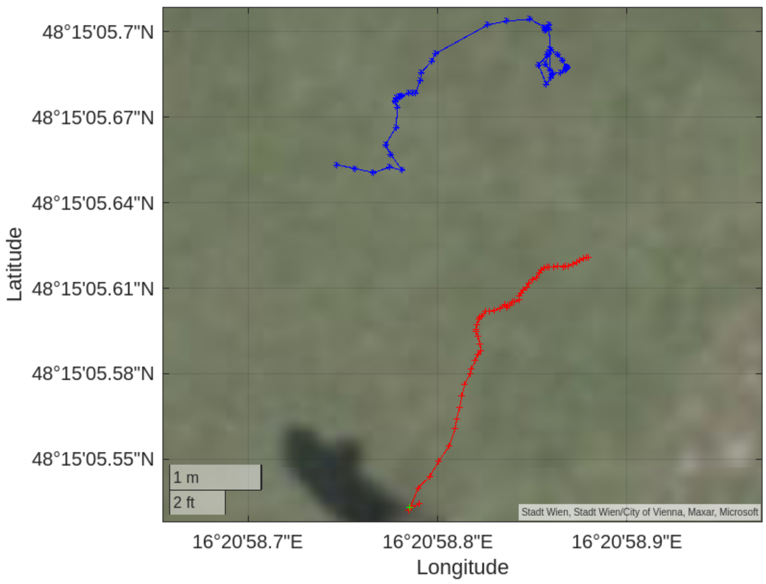


Figure 5.5: Static Map

5.1.2 Scenario C

In Figure 5.6 the localization error of scenario C is plotted. The measurement points are drifting over time in a circular path, which indicates a semi-stable state of error. The distance between the actual position and the start of the detected points indicate a constant offset, which if the signal always behaves this way, could be calculated.

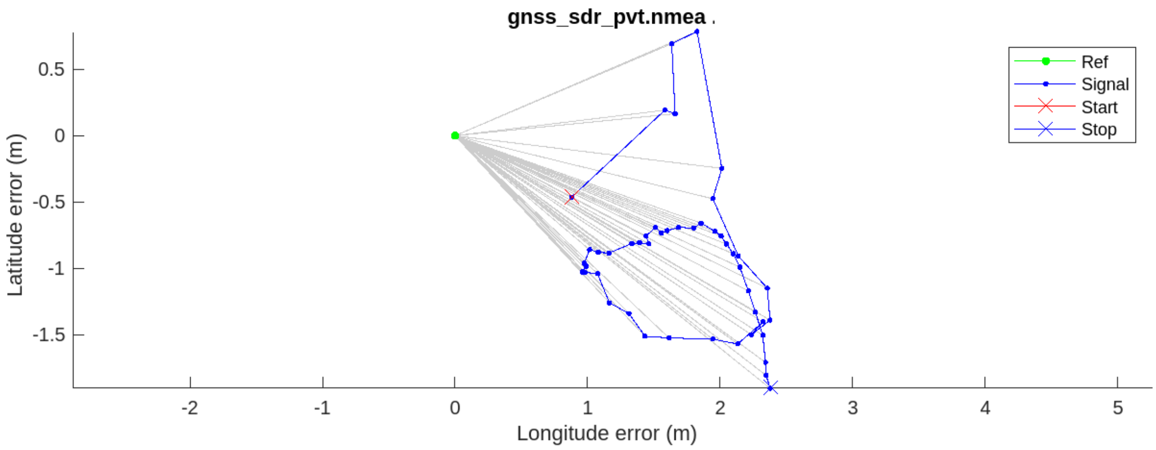


Figure 5.6: SDR Static

Figure 5.7 shows the CDF of the horizontal error of scenario C. The first 20 % of signals are at least 1.5 m away, which indicates a constant offset to the target. 50 % of all signals are inside a radius of 1.97 m. If a constant correction of 1.5 m towards the original target would be added, 50 % of all signals would be inside a radius of 0.47 m.

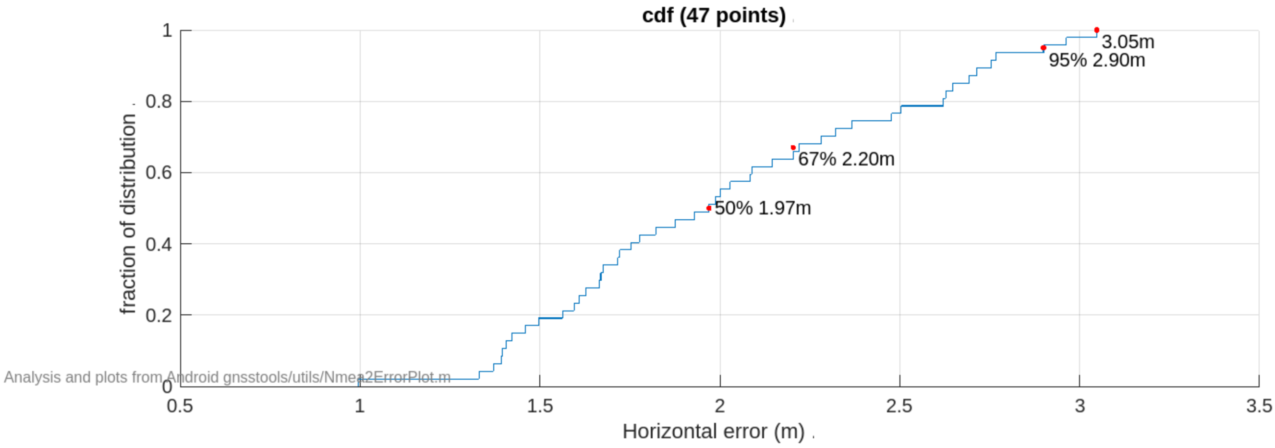


Figure 5.7: SDR Static CDF

Figure 5.8 shows the same image as Figure 5.5, but in addition with a yellow path, the measurements of Scenario C. This allows for better visualization and comparison with the other scenarios. The yellow line represents the new signal while the blue signal is the Oneplus smartphone (Scenario A) and the red signal is the Samsung smartphone (Scenario B). The biggest difference between the previously discussed scenarios and Scenario C is the circular path of Scenario C. This curl could allow for first mitigation with a constant offset to the signal as described prior.

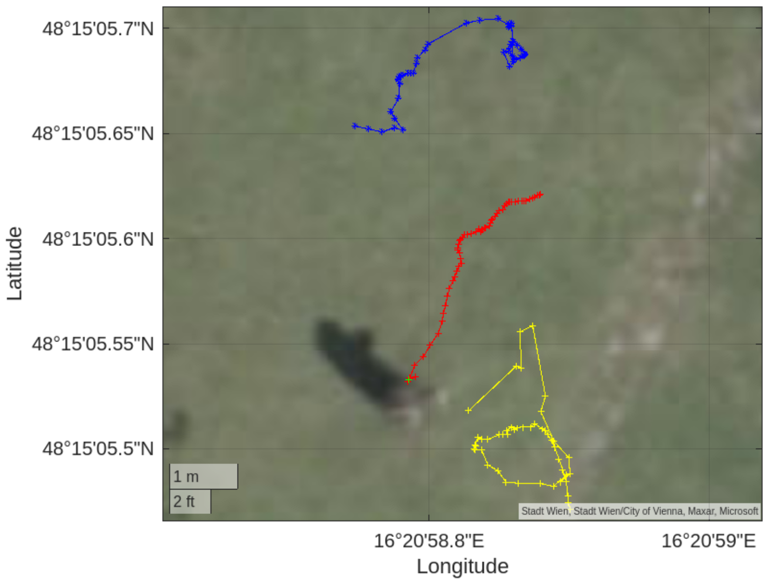


Figure 5.8: Static Map with RTCM data

In Table 5.5 the power consumption of the Android app can be seen. The main difference between Scenario A and C (both measured on the Oneplus smartphone) is the app and the attached USB peripheral. The measuring setup was the same as scenario A, so the debug report was extracted and analyzed with battery historian.

avg. Voltage	avg. Discharge	avg. Power
3.37 V	1239 mA	4.175 W

Table 5.5: Scenario C Android Power Consumption

For the energy consumption of the RTL-SDR V3 a USB power meter was used. The AVHzY CT-2 was selected because it can handle USB communication during the measurement and for its availability(it was already at the institute and ready to use). During the first measurements, it came apparent that the RTL-SDR V3 needs so much power that a simultaneous measurement and function test on the smartphone itself isn't feasible (see Section 5.2.1). So for the power measurement, a PC USB port was used. The result of the measurement can be seen in Figure 5.9.

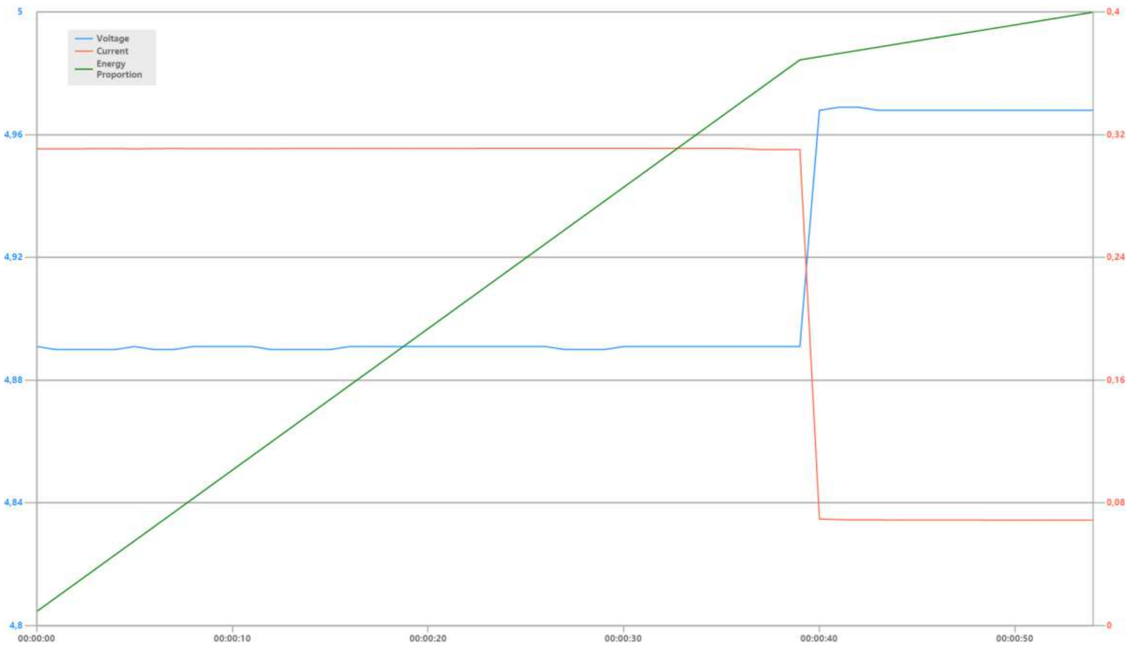


Figure 5.9: energy consumption on vs off

Figure 5.9 clearly shows the different energy consumption during receiving vs the idle state. On the left side, a GNSS signal is measured with an active antenna, which also uses power fed by the internal bias tee of the RTL-SDR. On the right side of the image, the idle power consumption can be seen. The interesting part of the measurement is the significant voltage drop of 0.08 V from 4.97 V to 4.89 V. Also the current draw of 0.31 A during receiving to 0.07 A during idle is significant. This results in a power draw of 1.516 W during receiving and an idle power draw of 0.348 W.

In Table 5.6 the mean and standard deviation in meters and degrees of the positioning error can be seen. A mean of 2.015 m and a standard deviation of 0.515 m improves compared to scenario B the standard deviation with a factor of 2.

Lon/Lat/Dist	Mean[deg]	Std[deg]	Mean[m]	Std[m]
Lat	48.251528	5.118886e-06	-0.917	0.569
Lon	16.349686	6.392644e-06	1.715	0.475
Dist	X	X	2.015	0.515

Table 5.6: Statistical measures of Longitudinal and Latitudinal measurements

5.1.3 Scenario D

For scenario D a completely different approach was selected, see Figure 4.3. The processing of the GNSS signals was transferred to the ROS system, which was necessary because of the complexity of all components. On Android, an RTCM Streamer app was used, which sends all GNSS signals as NMEA messages over NTRIP to a caster. This signal then gets pulled by another application in combination with APOS correction streams and ephemeris streams.

In Figure 5.10 the horizontal error is plotted. The start of the measurement is around 5 m away. The signal converges after some time to the correct position. This behavior was expected, because the librtk library needs some time to switch from only Single-Point Positioning (SPS) to float or fix, all three states of the RTK algorithm (see Section 3.1). This was also verified in Figure 5.13 and led to Section 5.1.3.1. The signals then curl around the correct position, indicating a switch to float.

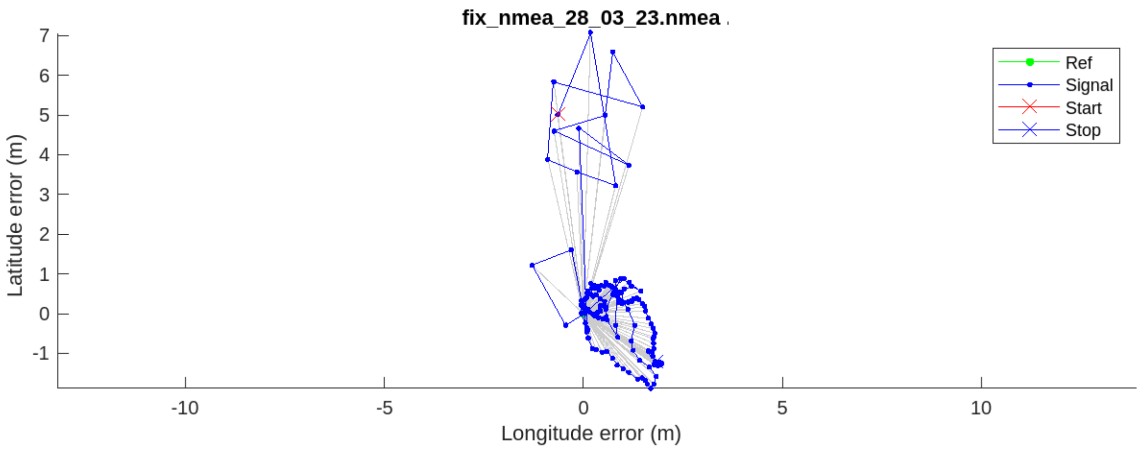


Figure 5.10: RTCM Static

Figure 5.11 shows the CDF of the signals error. The 50 % mark at 1.08 m and the 67 % mark at 1.63 m compared with the worst mark at 7.08 m shows the quality difference between SPS mode and RTK mode.

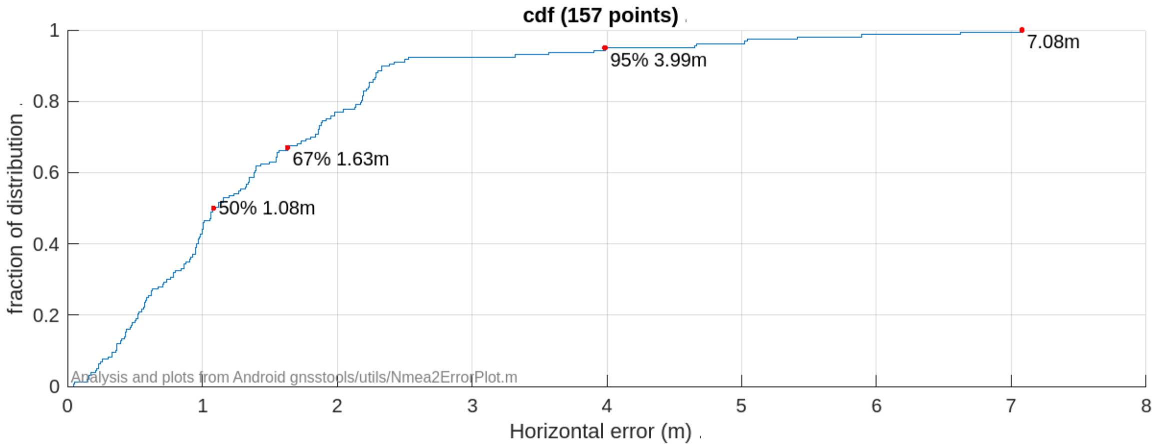


Figure 5.11: RTCM Static CDF

In Figure 5.12 the previously plotted graph (see Figure 5.10) can be seen drawn on a satellite image. The smartphone was placed on a bench at the new electrical engineering building. There the initial SPS position is clearly visible as random and spare placed measurement points. This observation led to the analysis of the signal quality as shown in Figure 5.13.

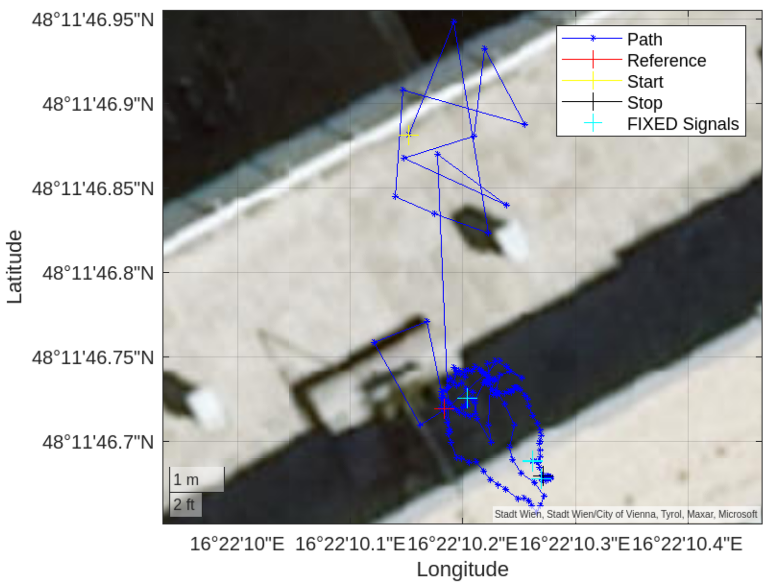


Figure 5.12: RTCM Map

Figure 5.13 shows the quality of the signals the rtklib algorithm calculated for each measurement point (see Section 3.1). The left bar indicates the initial state of the output data with only SPS precision, comparable with the signal quality of scenario B. After 16-17 measurements, the algorithm converges and finds solutions with the APOS correction data, indicated by the other two bars. Most of the solutions only archive float quality with sparse fix quality signals in between, which could indicate a bad antenna quality. Because only the float and fix signals should be analyzed, the SPS signals were removed with a Matlab script, see Section 5.1.3.1.

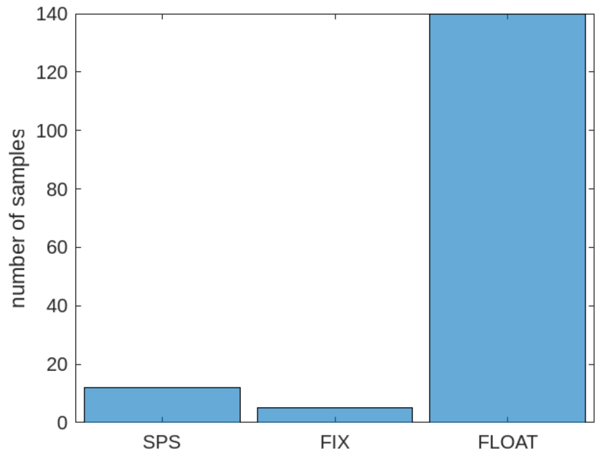


Figure 5.13: RTCM Static GNSS Quality

5.1.3.1 Initial Setup Inaccuracy Mitigation

After identifying the signal quality difference and the effect on the overall quality of the SPS signals, a Matlab script was written to only analyze the remaining signals. This improved the maximal error significantly.

In Figure 5.14 only float and fix quality signals are plotted. In comparison with Figure 5.10 the measured points drift in a circle over time, which indicates a convergence of the rtk algorithm. Because the previous analysis showed very few "fix" signals and more "float" signals, the algorithm tries to converge most of the time.

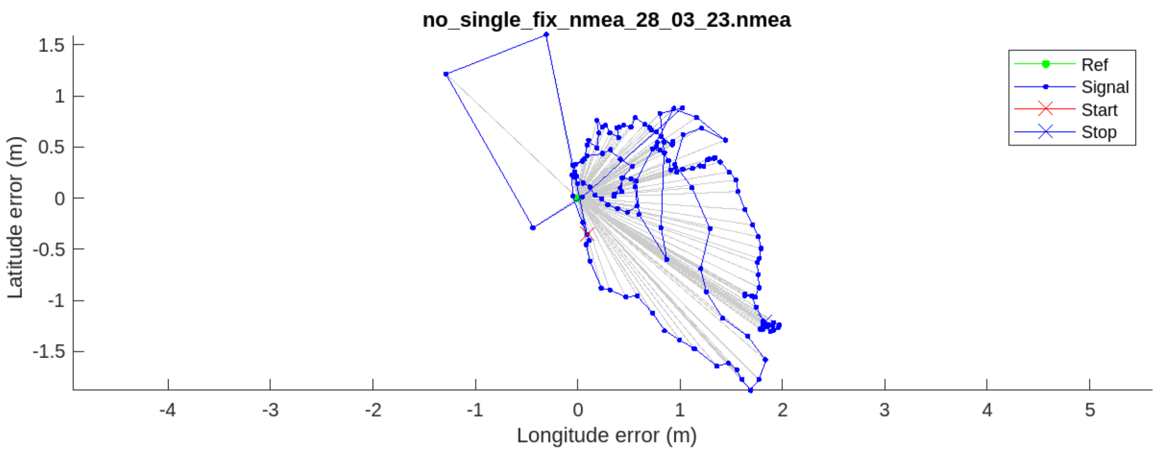


Figure 5.14: RTCM Static no SPS signals

In Figure 5.15 the CDF of the filtered signal can be seen. It clearly shows an improvement of the worst error of 2.53 m compared with Figure 5.11 which was 7.08 m. The 50 % error also improves marginally to 1.01 m.

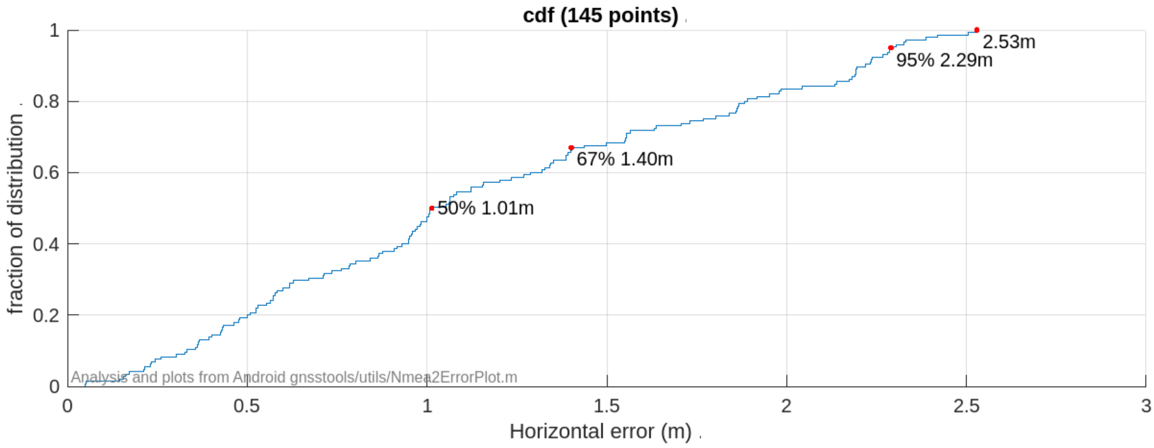


Figure 5.15: RTCM Static No SPS signals CDF

A power consumption measurement was done but because a constant USB connection was necessary, it is not conclusive. So it was not included.

5.1.4 Scenario E

In Scenario E (see Figure 4.4) the available IMU data of the smartphone should be used to improve the localization accuracy even further. For this, the IMU data was also streamed to ROS, and the robot_localization module (see Section 3.4) was used. For comparison reasons, the output files of scenario D which are still available in scenario E as intermediate files were recorded. The following measurements are the intermediate data because the used nmea_navsat_driver has a programming error. For more information see Section 5.2.1.

In Figure 5.16 the same measurement setup of Figure 5.14 was used, only the measurement duration was extended. The SPS signals are already removed, to allow the comparison between the two measurement runs. The signal points clearly converge to the correct position and only deviate from it by a small margin. This indicates a good convergence of the rtk algorithm.

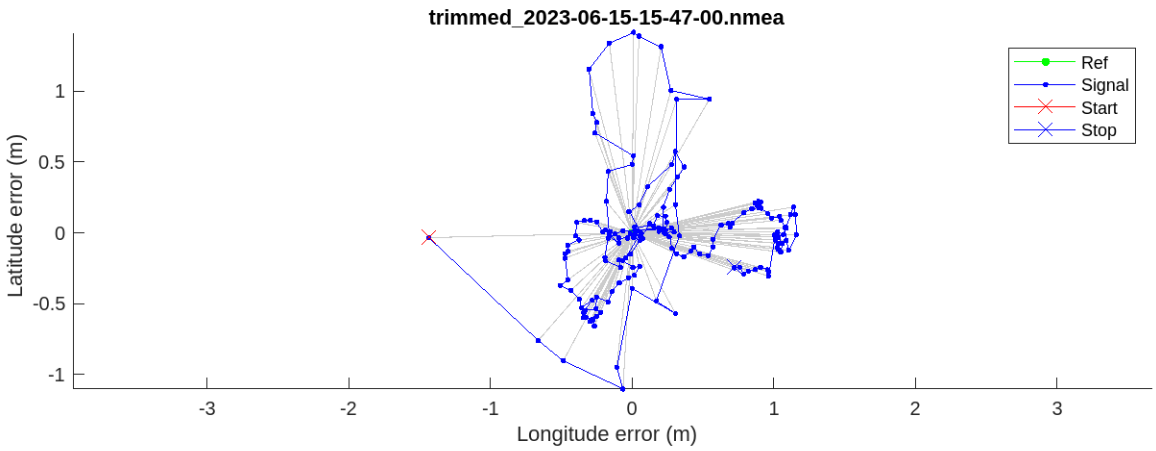


Figure 5.16: RTCM Static

Figure 5.17 shows a accuracy in the sub-meter range with a 50 % error of 0.55 m. This result is even better than scenario D, Figure 5.15. This could have multiple reasons, for example, the measurement duration was longer. The weather could also have an impact on the results. For further discussions of scenario D, the comparison results of scenario E are taken.

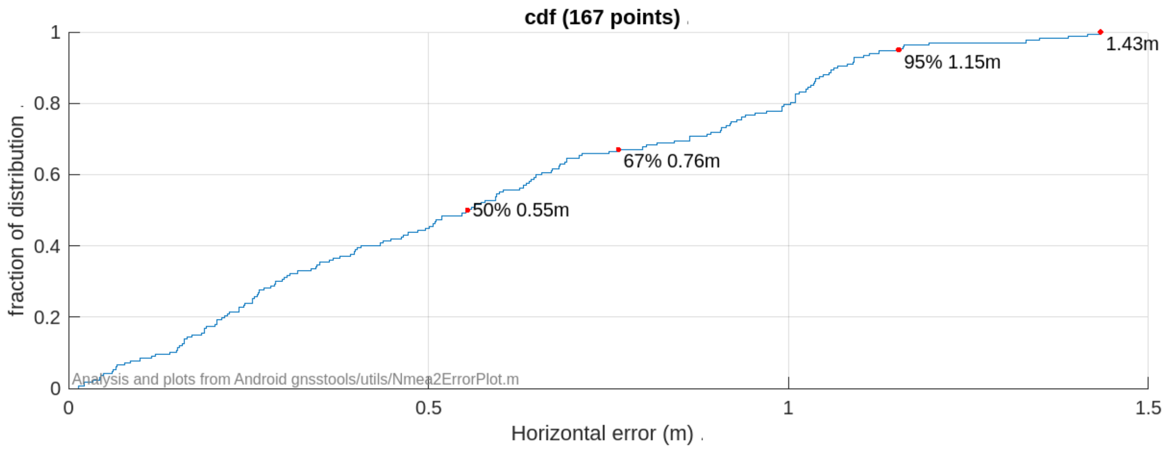


Figure 5.17: RTCM Static CDF

In Figure 5.18 the plot of Figure 5.16 on a satellite photograph can be seen. The lack of contrast indicates good accuracy because if multiple features would be visible, the error would be bigger.

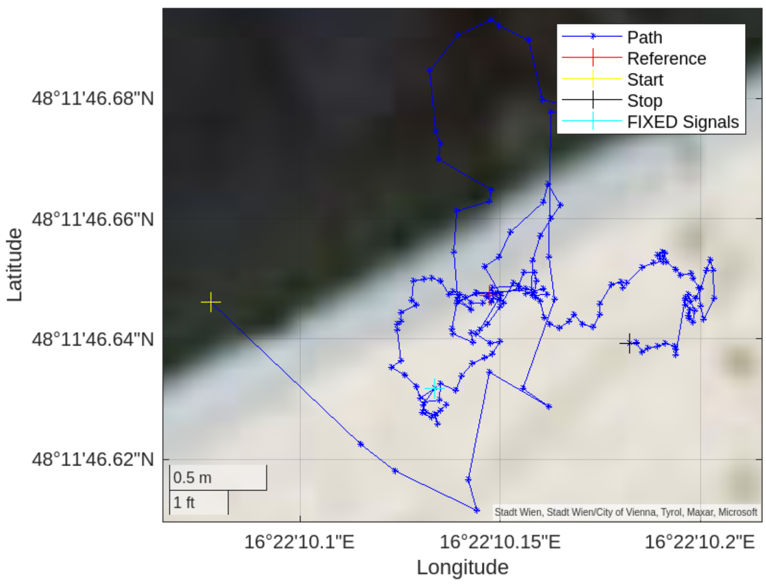


Figure 5.18: RTCM Map

In Table 5.7 the mean and standard deviation of the horizontal distance error can be seen. The mean of 0.585 m with a standard deviation of 0.37 m is a drastic improvement compared to the other scenarios.

Lon/Lat/Dist	Mean[deg]	Std[deg]	Mean[m]	Std[m]
Lat	48.196291	3.651275e-06	-0.018	0.406
Lon	16.369488	6.904178e-06	0.228	0.513
Dist	X	X	0.585	0.369

Table 5.7: Statistical measures of Longitudinal and Latitudinal measurements

5.1.4.1 Variance Comparison before and after the EKF

In Figure 4.4, a nmea_navsat_driver ROS module was added because the modified rtkcrv_ros module generated NMEA messages. Unfortunately, the nmea_navsat_driver ROS module doesn't compute the correct Variance. The module sends 16 as calculated Variance, as shown on the right site in Figure 5.19. The robot_localization module relies on correctly calculated Variance because of the working principle (see Section 3.4) of the underlying EKF. The calculated Variance after the filter stage is around 4.3, which is 4 times smaller or the square root of the initial Variance. This would greatly improve if the input variance was calculated correctly.

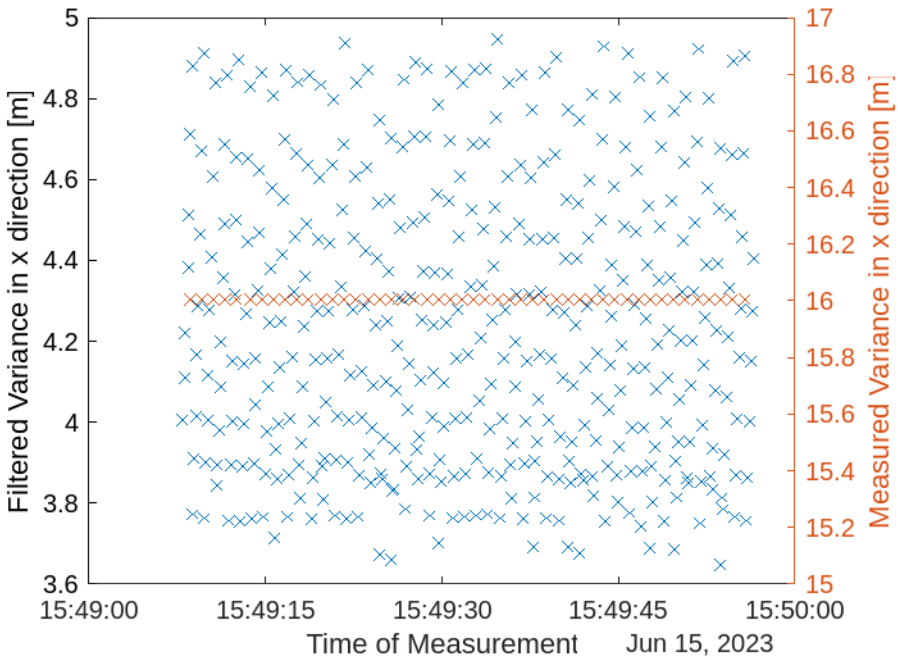


Figure 5.19: Variance of measurement before and after EKF

5.1.5 Scenario Comparison

The results of all CDF error plots are summarized in Table 5.8. For scenario D the intermediate results of scenario E were used. The table shows a big difference between the Oneplus and the Samsung smartphone. This was expected because the Samsung smartphone has a dual-frequency GNSS receiver. This and a possible better antenna result in a noticeable accuracy increase. The active antenna with the SDR and gnss-sdr software performed even better, but only around 40 cm in accuracy. A bigger increase was unexpected but the small increase could be explained because the SDR can only receive on one frequency, while the Samsung smartphone can reduce the error with dual-frequency GNSS. The biggest improvement could be archived with the Samsung smartphone in combination with rtklib and APOS correction data. The improvement between scenarios C and D of 4 times the accuracy (comparing the 50 % values).

cdf error	Scenario A	Scenario B	Scenario C	Scenario D[a]
50 %(CEP)	5.02 m	2.39 m	1.97 m	0.55 m
67 %	5.07 m	2.66 m	2.20 m	0.76 m
95 %	5.41 m	3.26 m	2.90 m	1.15 m

Table 5.8: Scenario Comparison - Accuracy

[a] As described in Section 5.1.4 the data of scenario E with the setup of scenario D was used.

In Table 5.9, the mean and the standard deviation of the distance error of each scenario are shown. Scenario A has the biggest mean, indicating the least accuracy of all scenarios. Scenario B has a much better mean and a higher standard deviation, indicating worse repeatability and longtime accuracy. Scenario C improves the standard deviation by almost 100 % while improving the mean only marginally. The best error performance has scenario D which reduces the mean error by a factor of 4 while still improving the standard deviation of the error.

Scenario	Mean[m]	Std[m]
Scenario A	4.824	0.513
Scenario B	2.152	0.947
Scenario C	2.015	0.515
Scenario D[a]	0.585	0.369

Table 5.9: Scenario Comparison - Statistical Measures

[a] As described in Section 5.1.4, the data of scenario E with the setup of scenario D was used.

For power consumption, scenario A would be the best. The Oneplus only needs 2.561 W to function, while the Samsung smartphone needs 4.403 W for the same measurement. This could be explained by the more complicated dual-frequency receiver and the resulting calculation overhead. Scenario C, which was measured with the Oneplus smartphone, showed a much higher power draw of 4.403 W, of which around 1.516 W were drawn by the USB peripheral. For scenario D, a power measurement was not conclusive, because of the constant USB connection between the smartphone and the PC. It would also be not valid because most of the calculations were "outsourced" from the Android device to the ROS master.

Scenario	avg. Voltage	avg. Discharge	avg. Power
Scenario A	4.07 V	875 mA	3.561 W
Scenario B	4.1 V	1074 mA	4.403 W
Scenario C - overall	3.37 V	1239 mA	4.175 W
Scenario C - SDR	4.89 V	310 mA	1.516 W

Table 5.10: Scenario Comparison - Power Consumption

5.2 Discussion

Section 5.1.5 showed the best result in terms of accuracy with scenario D, using a dual-frequency GNSS receiving capable smartphone (here the Samsung Galaxy S20 Plus) and the librtk tool suite. The best results were archived with the modifications of Ferreira et al. [10] and myself.

In terms of power consumption, the Oneplus smartphone would be the best, but in combination with its localization performance not worth the savings of only 0.842 W compared with the Samsung smartphone.

Description	σ Northing	σ Easting
Scenario D	0.406 m	0.513 m
ANN-MB RTK 0.5m	0.01 m	0.01 m
ANN-MB RTK 1.5m	1.90 m	1.03 m
HX CHX600A DGPS 1.5m	1.08 m	0.58 m
HX CH6601A RTK 1.5m	0.32 m	0.3 m
A.80 RTK 1.5m	0.87 m	1.12 m

Table 5.11: Willeger's Thesis Comparison (see Table 4.18[2, p. 95])

In comparison with Willeggers thesis results shown in Table 4.18[2, p. 95] of his thesis and in Table 5.11, most measurements of Willeger are outperformed by the results of scenario D. Only the RTK 0.5m measurements of Willegger with a standard deviation of 0.01 m in easting and northing direction and some RTK 1.5m measurements, beat Scenario D. Scenario D has a standard deviation in the northing direction of 0.406 m and a standard deviation in the easting direction of 0.513 m, which is comparable with his measurement "HX CH6601A RTK 1.5m" with approx.0.3 m standard deviation in both directions. It outperforms "CH7603A RTK 5m" with a sigma in easting of 0.47 m and a sigma in northing of 0.98 m. Willeger also compared his results with the thesis of Svaton [51] in Table 1.2[2, p. 22], which had a CEP of at best 0.97 m with a standard deviation of 1.01 m. Also, this measurement was beaten by scenario D.

Scenario C showed the possibility to extend and improve the accuracy of an integrated GNSS receiver by adding an inexpensive SDR with an active antenna. When comparing scenario A with a mean of error of

4.824 m and the same smartphone but with the external antenna with a mean of error of 2.015 m, the accuracy could be improved by a factor of 2.

The thesis also showed the possibility of utilizing different sensors of an Android smartphone for usage in robotics. Scenarios A, B, and E showed the usefulness of smartphone-integrated sensors, e.g., IMU measurements, for sensor fusion to improve robot localization.

5.2.1 Obstacles and Failures

The following section will address multiple obstacles that arose during development, implementation, and testing. Most of the failures were caused by a lack of information about the topic or missing documentation of the used software. Some software was even deprecated, so no support could be found.

RTL-SDR Consumption - Power Hiccups

During the power measurement of the RTL-SDR V3 with the VHzY CT-2 connected to the smartphone a hiccup happened. On the start of the driver via Intent, the current consumption exceeded the maximum of the USB port of the smartphone which led to a power hiccup, as seen in Figure 5.20. This further led to the conclusion of the need for a possible active power between the smartphone and the rtl-sdr. Fortunately, the active hub was unnecessary, as shown in Figure 5.9.

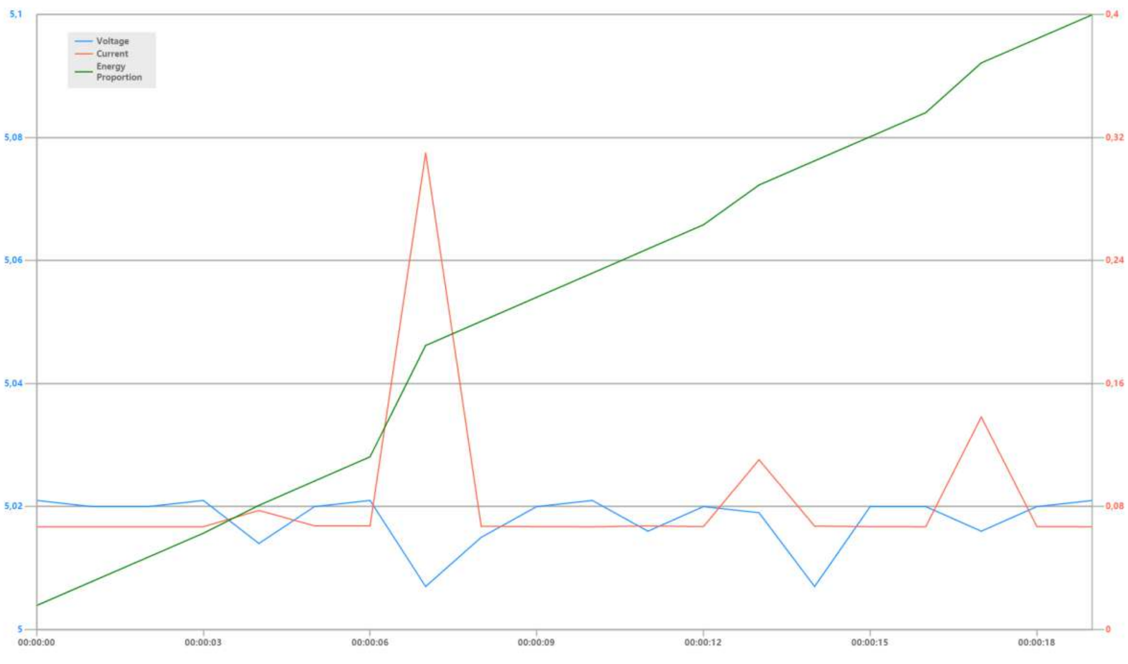
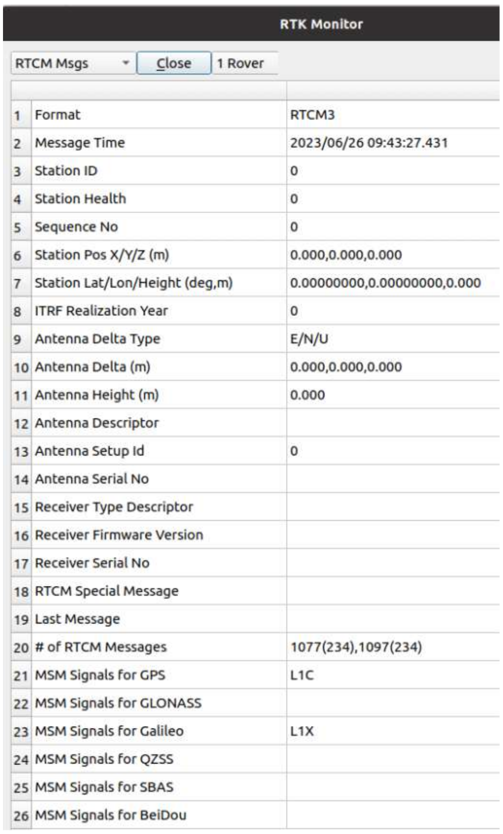


Figure 5.20: hiccups during the start of the rtl driver on Android

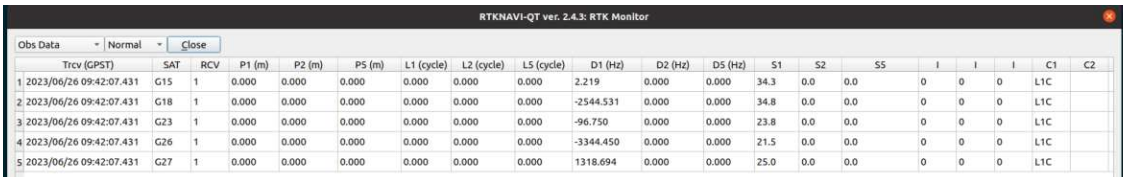
RTCM Converter - Incorrect RTCM MSM data

During the testing of scenario D, a lot of time went into debugging the RTCM 3 message stream of the Oneplus smartphone over the NTRIP caster. rtklib decoded the MSM just fine (see Figure 5.21) but some information was missing (see Figure 5.22), so no fix could be acquired. This led to the test setup only including the Samsung smartphone.



1	Format	RTCM3
2	Message Time	2023/06/26 09:43:27.431
3	Station ID	0
4	Station Health	0
5	Sequence No	0
6	Station Pos X/Y/Z (m)	0.000,0.000,0.000
7	Station Lat/Lon/Height (deg,m)	0.00000000,0.00000000,0.000
8	ITRF Realization Year	0
9	Antenna Delta Type	E/N/U
10	Antenna Delta (m)	0.000,0.000,0.000
11	Antenna Height (m)	0.000
12	Antenna Descriptor	
13	Antenna Setup Id	0
14	Antenna Serial No	
15	Receiver Type Descriptor	
16	Receiver Firmware Version	
17	Receiver Serial No	
18	RTCM Special Message	
19	Last Message	
20	# of RTCM Messages	1077(234),1097(234)
21	MSM Signals for GPS	L1C
22	MSM Signals for GLONASS	
23	MSM Signals for Galileo	L1X
24	MSM Signals for QZSS	
25	MSM Signals for SBAS	
26	MSM Signals for BeiDou	

Figure 5.21: Oneplus RTCM Message Missing Information



	Trcv (GPST)	SAT	RCV	P1 (m)	P2 (m)	P5 (m)	L1 (cycle)	L2 (cycle)	L5 (cycle)	D1 (Hz)	D2 (Hz)	D5 (Hz)	S1	S2	S5	I	I	I	C1	C2
1	2023/06/26 09:42:07.431	G15	1	0.000	0.000	0.000	0.000	0.000	0.000	2.219	0.000	0.000	34.3	0.0	0.0	0	0	0	L1C	
2	2023/06/26 09:42:07.431	G18	1	0.000	0.000	0.000	0.000	0.000	0.000	-2544.531	0.000	0.000	34.8	0.0	0.0	0	0	0	L1C	
3	2023/06/26 09:42:07.431	G23	1	0.000	0.000	0.000	0.000	0.000	0.000	-96.750	0.000	0.000	23.8	0.0	0.0	0	0	0	L1C	
4	2023/06/26 09:42:07.431	G26	1	0.000	0.000	0.000	0.000	0.000	0.000	-3344.450	0.000	0.000	21.5	0.0	0.0	0	0	0	L1C	
5	2023/06/26 09:42:07.431	G27	1	0.000	0.000	0.000	0.000	0.000	0.000	1318.694	0.000	0.000	25.0	0.0	0.0	0	0	0	L1C	

Figure 5.22: Oneplus RTK Observation

RTCM Converter - Missing ephemeris data

While implementing the experiment described in Section 5.1.3, a significant obstacle was identifying the problem of the missing ephemeris data while running rtkcrv_ros. After configuring the rover stream to the android-generated one and the base station stream to the APOS one, the software showed all observations fine but couldn't detect correct satellite positions. The few debugging messages that could be retrieved were not helpful, so I debugged the program with GNU Debugger (GDB). After many hours of debugging, it became clear that some information was still missing. This lead to the inside of including a third RTCM stream with ephemeris data only. This service was also unavailable by APOS, so a third NTRIP caster was needed, preferably a free one. Multiple free casters are available, so I initially used rtk2go. After the first successful test, I discovered the AGBs of the caster do not allow development rovers. This led to the switch to the ntrip.use-snip.com caster, which permitted development use.

GeolocPVT - Tests

The app described in Section 2.2 led to a rabbit hole of tests and, unfortunately, many dead ends. I tried to use the app on both smartphones without success. After an intense debugging session in android studio, I concluded that the current state of the open-source software in its GitLab repository is not working. During the refactoring/rework of the app, the in the paper described RTCM stream was deactivated, and many options were disabled. It also led to the experiments with GoGPS (see Section 3.3), which ultimately also led to a dead end (see Section 5.2.1).

GoGPS - Implementation Nightmare

After discovering this open software (see Section 3.3 and Section 2.2), I decided to implement it also into ros mobile (see Section 2.3). In hindsight, I should have listened to Grenier and Renaudin [8] when they described GoGPS with "The GoGPS Java port does not offer any documentation for implementing the library.". The sheer amount of needed work to implement this into the existing ros mobile app wasn't feasible. Beginning with version compatibility issues encountered after porting the Gradle-enabled library to the ros source code was time-consuming to resolve, yet alone the time needed to understand the partially broken implementation of the library inside the GeolocPVT source code.

robot_localization - nmea_navsat Module variance calculations wrong

Figure 5.19 showed that the nmea_navsat_driver module has a bug when calculating the incoming NMEA data/signals variance. The module constantly outputs 16m as variance, which directly influences the performance of the EKF. The Filter, which weights the input data according to its variance, gives an improved to its input data, but compared with Scenario D, much worse performance. Because of timing constraints, the module was not analyzed or fixed.

6 Conclusion and Outlook

This chapter gives a conclusion of this thesis and its outlook. The research questions are restated and answered in the conclusion section. In the outlook section, multiple improvements are shortly depicted, which weren't possible to implement due to time constraints.

6.1 Conclusion

This thesis aimed to answer multiple research questions regarding the usability of Android smartphones for localization purposes and the accuracy of this localization. The first research question asked: "*How effective is the use of an Android smartphone for the localization of an agricultural robot measured by resource consumption and accuracy?*". This can be answered with the previously done comparison in Section 5.2, which showed that especially scenario E (see Section 5.1.4) is almost as accurate and, therefore, comparable with Willeger's findings. Furthermore is, the resource consumption on a monetary level substantially lower, while the processing capabilities are substantially higher. All Scenarios showed promising power and resource efficiency of Android for robotic applications.

The second research question: "*How different is the performance between smartphones with and without correction data?*" is also answered with the comparison of scenario B (see Section 5.1.1) with scenario D (see Section 4.5) in Table 5.8. The results of the Samsung scenarios showed a significant difference between uncorrected ($2.39\,\mathrm{m}$) and corrected ($0.55\,\mathrm{m}$) signals regarding the CEP. This showed the importance of a good post-processing stack and the need for correction data for high-precision farming robots.

The third research question is concerned with: " *What is the performance gain (in accuracy) using an external GNSS antenna?*". This can be answered with the comparison between scenario A (see Section 5.1.1) and scenario C (see Section 5.1.2) in Table 5.8. Using the external antenna with an attached SDR and another post-processing stack resulted in a significant performance gain. The CEP of the Oneplus smartphone decreased from 5.02 m without an active external antenna to 1.97 m with an active antenna. This showed the importance of an external, active antenna for accurate localization without correction data.

This thesis showed the potential of recycled Android smartphones for agricultural robots regarding localization and as a sensor platform. Especially scenario C (see Section 4.4) showed the possibility of using a Linux subsystem on the smartphone to port or write software for Linux without using the Android libraries or programming language constraints to Java or Kotlin.

The thesis also showed that even with a good post-processing stack and correction data, a smartphone's RTK mode is still not as good as an Application Specific Integrated Circuit (ASIC) implementation regarding localization accuracy.

6.2 Outlook

In this section, multiple improvements are described in their subsection. Some of them describe already feasible improvements but with moderate time and resource requirements, which wasn't possible to do during this thesis. While other improvements should be possible, but weren't tested to be at least feasible. These 'speculative' improvements are the ANT+ integration in Android and the dual frequency approach with internal and external antennae.

Fix of Variance for robot_localization module

To get scenario E to work, either the nmea_navsat_driver module needs to be repaired to calculate the correct variance or another way needs to be found to transfer the localization data from the RTKLIB ROS module into the robot_localization module. This could improve the accuracy of the localization.

Multiple Antenna Setup

With a dual SDR e.g., LimeSDR [52] or Hack-RF[53] ionosphere free/dual frequency GNSS-SDR setup could be achieved. This would increase the cost significantly (approx. € 300 for the HackRF), so it was not considered. In [10], Ferreira et al. describe this setup in combination with pose estimation, the gain pose information, and the robot's direction based on the GNSS signals.

GNSS-SDR controller for ROS

To control the GNSS-SDR software, an API exists for a developer to configure, start and stop the GNSS-SDR pipeline. Multiple examples are already implemented, e.g., [54]. This API could be used to develop a ROS node or adapt the "translation" node described in Section 4.4 to control the pipeline, gain debug and status information, and other data. Because of time constraints, this controlling node was not implemented.

Utilizing the Android ANT+ receiving capabilities

The tested Samsung S20+ includes an ANT+ interface. This allows a developer to integrate all ANT+ Sensors into an Android App (see [55]). ANT+ is a communication standard to exchange health and geobased information between sensors and receivers. Interesting for use in robotics would be tire sensors called Tire Pressure Monitoring System (TPMS) and geo beacons, e.g., the Garmin chirp (the chirp is no longer manufactured, see [56]). The chirp would have allowed the implementation of a way-point system, and the TPMS could be used to track the tire pressure of a robot.

GNSS-SDR - utilizing dual-frequency with internal and external receivers

It would be interesting if a fusion of the Android GNSS data of the L1C band and the data of the external SDR receiver on the L5 could be archived. This would allow upgrading even older smartphones to dual-frequency receivers, increasing the accuracy.

ROS Mobile - Utilizing the Smartphone's Touch Display to show Status Information

Another promising application for the smartphone on the robot would be a status screen for multiple important information. This could be realized by adding another custom Widget that subscribes to various topics and publishes actions to control the ROS robot.

Bibliography

[1] Twisted-Fields, *Twisted-Fields/acorn-precision-farming-rover: Source code for Acorn, the precision farming rover by Twisted Fields*. [Online]. Available: https://github.com/Twisted-Fields/acorn-precision-farming-rover (visited on 06/16/2023).

[2] E. Willegger, "Implementation of a multi-band RTK receiver system with Arduino," en, Accepted: 2020-09-22T07:35:26Z Journal Abbreviation: Implementierung eines Multiband-RTK-Empfängersystems mit Arduino, Thesis, Wien, 2020. (visited on 01/10/2023).

[3] C. Fernández-Prades, J. Arribas, and P. Closas, "Turning a Television into a GNSS Receiver," Sep. 2013.

[4] C. Fernández-Prades, *GNSS-SDR operation with a Realtek RTL2832U USB dongle DVB-T receiver*, en, Jun. 2016. [Online]. Available: https://gnss-sdr.org/docs/tutorials/gnss-sdr-operation-realtek-rtl2832u-usb-dongle-dvb-t-receiver/ (visited on 03/31/2023).

[5] D. Skufca, "Dual Frequency GPS Receiver Implementation in GNSS-SDR," en, *ECE Senior Capstone Project*, vol. 2018 Tech Notes,

[6] D. Skufca, *GitHub - dskufca/SrDesign: All code for senior design project, i.e. a copy of GNSS-SDR plus our adaptations of it*. [Online]. Available: https://github.com/dskufca/SrDesign (visited on 02/28/2023).

[7] European GNSS Supervisory Authority., *Using GNSS raw measurements on Android devices: white paper.* en. LU: Publications Office, 2017. (visited on 03/16/2023).

[8] A. Grenier and V. Renaudin, "Efficient Use of SSR RTCM Streams For Real-Time Precise Point Positioning on Smartphones," in *2019 16th Workshop on Positioning, Navigation and Communications (WPNC)*, ISSN: 2164-9758, Oct. 2019, pp. 1–6.

[9] A. Privat, M. Pascaud, and D. Laurichesse, "Innovative smartphone applications for Precise Point Positioning," en, in *2018 SpaceOps Conference*, Marseille, France: American Institute of Aeronautics and Astronautics, May 2018. (visited on 03/16/2023).

[10] A. Ferreira, B. Matias, J. Almeida, and E. Silva, "Real-time GNSS precise positioning: RTKLIB for ROS," *International Journal of Advanced Robotic Systems*, vol. 17, p. 172 988 142 090 452, May 2020.

[11] Y. Heo, T. Yan, S. Lim, and C. Rizos, "International standard GNSS real-time data formats and protocols," Jan. 2009.

[12] N. Rottmann, N. Studt, F. Ernst, and E. Rueckert, *ROS-Mobile: An Android application for the Robot Operating System*, arXiv:2011.02781 [cs], Nov. 2020. [Online]. Available: http://arxiv.org/abs/2011.02781 (visited on 08/30/2022).

[13] G. Supper, N. Barta, A. Gronauer, and V. Motsch, "Localization accuracy of a robot platform using indoor positioning methods in a realistic outdoor setting," *Die Bodenkultur: Journal of Land Management, Food and Environment*, vol. 72, pp. 133–139, Jun. 2022.

[14] "Das Navigationssystem," ger, in *Mathematik der digitalen Medien: präzise - verständlich - einleuchtend*, ser. Lehrbuch Studium, 2., durchgesehene Auflage, Berlin: VDE VERLAG GmbH, 2017, pp. 13–54.

[15] *Differential GPS - Navipedia*. [Online]. Available: https://gssc.esa.int/navipedia/index.php/Differential_GPS (visited on 05/14/2023).

[16] *RTK Fundamentals - Navipedia*. [Online]. Available: https://gssc.esa.int/navipedia/index.php/RTK_Fundamentals (visited on 05/14/2023).

[17] NMEA, *NMEA*, en. [Online]. Available: https://www.nmea.org/ (visited on 06/29/2023).

[18] R. Langley, "NMEA 0183 : A GPS Receiver Interface Standard," 2004. (visited on 06/29/2023).

[19] Matlab, *Parse data from standard and manufacturer-specific NMEA sentences sent from marine electronic devices - MATLAB - MathWorks Deutschland*. [Online]. Available: https://de.mathworks.com/help/nav/ref/nmeaparser-system-object.html (visited on 06/29/2023).

[20] tersus, *New additions in RTCM3 and what is MSM*. [Online]. Available: https://www.tersus-gnss.com/tech_blog/new-additions-in-rtcm3-and-What-is-msm (visited on 07/03/2023).

[21] BKG, *Real-Time*. [Online]. Available: https://igs.bkg.bund.de/ntrip/ (visited on 07/03/2023).

[22] Hedgehack, *YCCaster*. [Online]. Available: https://yccaster.com/ (visited on 03/16/2023).

[23] BEV, *APOS - Austrian Positioning Service*, de. [Online]. Available: https://www.bev.gv.at/Services/Produkte/Grundlagenvermessung/APOS.html (visited on 07/03/2023).

[24] T. Takasu, *RTKLIB: An Open Source Program Package for GNSS Positioning*. [Online]. Available: https://rtklib.com/ (visited on 03/16/2023).

[25] T. Takasu, *RTKLIB Manual*, 2013. [Online]. Available: https://www.rtklib.com/prog/manual_2.4.2.pdf.

[26] C. Fernández–Prades, J. Arribas, P. Closas, C. Avilés, and L. Esteve, "GNSS-SDR: An Open Source Tool For Researchers and Developers," in *Proc. 24th Intl. Tech. Meeting Sat. Div. Inst. Navig.*, Portland, Oregon, Sep. 2011, pp. 780–794.

[27] L. E. Franks, *Signal theory* (Information theory series). Englewood Cliffs, N.J: Prentice-Hall, 1969.

[28] G. Wade, *Signal Coding and Processing*, 2nd ed. Cambridge: Cambridge University Press, 1994. (visited on 07/03/2023).

[29] J. Kirkhorn, "Introduction to IQ-demodulation of RF-data," *Ifbt, Ntnu*, vol. 15, 1999.

[30] D.-I. (C. Wolff, *In-phase & Quadrature- Verfahren - Radar Basics*, de, Publisher: Dipl.-Ing. (FH) Christian Wolff. [Online]. Available: https://www.radartutorial.eu/10.processing/sp06.de.html (visited on 02/15/2023).

[31] osmocom, *Rtl-sdr - rtl-sdr - Open Source Mobile Communications*. [Online]. Available: https://osmocom.org/projects/rtl-sdr/wiki (visited on 02/16/2023).

[32] , *RTL2832 Datasheet*. [Online]. Available: https://z3d9b7u8.stackpathcdn.com/pdf-down/R/T/L/RTL2832-Realtek.pdf (visited on 02/15/2023).

[33] *RTL-SDR Blog V.3. Dongles User Guide*, en-US, Aug. 2016. [Online]. Available: https://www.rtl-sdr.com/rtl-sdr-blog-v-3-dongles-user-guide/ (visited on 02/15/2023).

[34] *GPS Measurement Tools*, original-date: 2016-09-03T01:06:41Z, Jan. 2023. [Online]. Available: https://github.com/google/gps-measurement-tools (visited on 01/26/2023).

[35] termux, *Termux*, en. [Online]. Available: https://termux.dev/en/ (visited on 07/04/2023).

[36] termux, *PRoot Distro*, original-date: 2020-07-20T20:46:15Z, Jul. 2023. [Online]. Available: https://github.com/termux/proot-distro (visited on 07/04/2023).

[37] E. Realini and M. Reguzzoni, "goGPS: Open Source Software for Enhancing the Accuracy of Low-cost Receivers by Single-frequency Relative Kinematic Positioning," *Measurement Science and Technology*, vol. 24, p. 115 010, Oct. 2013.

[38] goGPS, *GitHub - goGPS-Project/goGPS_java: goGPS Java is a GNSS observation processing library.* [Online]. Available: https://github.com/goGPS-Project/goGPS_Java (visited on 02/15/2023).

[39] *Battery Historian*, original-date: 2014-06-20T21:28:26Z, Jan. 2023. [Online]. Available: https://github.com/google/battery-historian (visited on 01/26/2023).

[40] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "ROS: An open-source Robot Operating System," in *ICRA workshop on open source software*, Issue: 3.2, vol. 3, Kobe, Japan, 2009, p. 5.

[41] *REP REP 0 – Index of ROS Enhancement Proposals (REPs) – REP 0 – REP 0 – Index of ROS Enhancement Proposals (REPs) (ROS.org)*. [Online]. Available: https://ros.org/reps/rep-0000.html (visited on 05/23/2023).

[42] *REP 103 – Standard Units of Measure and Coordinate Conventions (ROS.org)*, Dec. 2014. [Online]. Available: https://www.ros.org/reps/rep-0103.html (visited on 03/14/2023).

[43] *REP 105 – Coordinate Frames for Mobile Platforms (ROS.org)*, Oct. 2010. [Online]. Available: https://www.ros.org/reps/rep-0105.html (visited on 03/14/2023).

[44] T. Moore and D. Stouch, “A Generalized Extended Kalman Filter Implementation for the Robot Operating System,” in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, Jul. 2014.

[45] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960. (visited on 07/03/2023).

[46] A. Becker (www.kalmanfilter.net), *Online Kalman Filter Tutorial*, en. [Online]. Available: https://www.kalmanfilter.net/ (visited on 07/03/2023).

[47] M. I. Ribeiro, “Kalman and extended kalman filters: Concept, derivation and properties,” *Institute for Systems and Robotics*, vol. 43, no. 46, pp. 3736–3741, 2004.

[48] *MATLAB Online*, de. [Online]. Available: https://de.mathworks.com/products/matlab-online.html (visited on 04/04/2023).

[49] S. Ltd, *SDR driver – Apps bei Google Play*, de. [Online]. Available: https://play.google.com/store/apps/details?id=marto.rtl_tcp_andro&hl=de (visited on 04/04/2023).

[50] S. Ltd, *Rtl_sdr Adroid Driver*, original-date: 2013-01-09T21:41:19Z, Jun. 2023. [Online]. Available: https://github.com/signalwareltd/rtl_tcp_andro- (visited on 07/04/2023).

[51] M. Svatoň, “Low-cost implementation of Differential GPS using Arduino,” 2016. (visited on 07/06/2023).

[52] *LimeSDR XTRX*, en. [Online]. Available: https://www.crowdsupply.com/lime-micro/limesdr-xtrx (visited on 03/06/2023).

[53] G. S. GADGETS, *HackRF One - Great Scott Gadgets*. [Online]. Available: https://greatscottgadgets.com/hackrf/one/ (visited on 06/16/2023).

[54] Á. C. Juan, *Acebrianjuan/gnss-sdr-monitor*, original-date: 2018-06-21T16:11:08Z, Feb. 2023. [Online]. Available: https://github.com/acebrianjuan/gnss-sdr-monitor (visited on 03/07/2023).

[55] Garmin, *Starting Your Project - THIS IS ANT*. [Online]. Available: https://www.thisisant.com/developer/ant/starting-your-project/#75_tab (visited on 06/16/2023).

[56] Garmin, *Welche Outdoor-Handgeräte sind mit Chirp kompatibel? | Garmin Support-Center*. [Online]. Available: https://support.garmin.com/de-AT/?faq=jxvkyfWfoD3ASnb1inWk19 (visited on 06/16/2023).

A Config Files

A.1 Tmuxinator

Listing A.1: Tmuxinator

```
# ~/.tmuxinator/measurement.yml

name: measurement
root: ~/SETUP

pre_window: source ~/catkin_ws/devel/setup.zsh

windows:
  - core:
      layout: tiled
      panes:
        - roscore
        - ~/Downloads/gnirehtet-rust-linux64/gnirehtet run
        - cd ~/ntrip_caster && ./yccaster
        - sleep 2 && rosrun rtkrcv_ros rtkrcv_ros_node -o Single_antenna.conf
        - sleep 2 && rosrun nmea_navsat_driver nmea_topic_driver fix:=gps/fix
        - sleep 2 && roslaunch vermin_base dual_ekf_navsat_example.launch
        - sleep 3 && rostopic pub /odometry/wheel nav_msgs/Odometry -f odom.yml
            -r 1
```

A.2 GNSS SDR TCP

Listing A.2: GNSS-SDR TCP config

```
[GNSS-SDR]

```

```
;######### GLOBAL OPTIONS ##################
GNSS-SDR.internal_fs_sps=2000000

;######### SIGNAL_SOURCE CONFIG ############
SignalSource.implementation=RtlTcp_Signal_Source
SignalSource.address=127.0.0.1
SignalSource.port=14423
SignalSource.item_type=gr_complex
SignalSource.sampling_frequency=2000000
SignalSource.freq=1575420000
SignalSource.gain=40
SignalSource.rf_gain=40
SignalSource.if_gain=30
SignalSource.AGC_enabled=false
SignalSource.samples=0
SignalSource.repeat=false
SignalSource.dump=false
SignalSource.dump_filename=../data/signal_source.dat
SignalSource.enable_throttle_control=false
SignalSource.swap_iq=false

;######### SIGNAL_CONDITIONER CONFIG ############
SignalConditioner.implementation=Pass_Through

;######### CHANNELS GLOBAL CONFIG ############
Channels_1C.count=8
Channels_1B.count=0
Channels.in_acquisition=1
Channel.signal=1C

;######### ACQUISITION GLOBAL CONFIG ############
Acquisition_1C.implementation=GPS_L1_CA_PCPS_Acquisition
Acquisition_1C.item_type=gr_complex
Acquisition_1C.coherent_integration_time_ms=1
Acquisition_1C.pfa=0.01
Acquisition_1C.doppler_max=5000
Acquisition_1C.doppler_step=250
Acquisition_1C.dump=false
Acquisition_1C.dump_filename=./acq_dump.dat
```

```
;######### TRACKING GPS CONFIG ############
Tracking_1C.implementation=GPS_L1_CA_DLL_PLL_Tracking
Tracking_1C.item_type=gr_complex
Tracking_1C.dump=false
Tracking_1C.dump_filename=./tracking_ch_
Tracking_1C.pll_bw_hz=35.0;
Tracking_1C.dll_bw_hz=1.5;
Tracking_1C.pll_bw_narrow_hz=2.5;
Tracking_1C.dll_bw_narrow_hz=0.5;
Tracking_1C.extend_correlation_symbols=1;
Tracking_1C.dll_filter_order=2;
Tracking_1C.pll_filter_order=3;
Tracking_1C.early_late_space_chips=0.5;
Tracking_1C.early_late_space_narrow_chips=0.25

;######### TELEMETRY DECODER GPS CONFIG ############
TelemetryDecoder_1C.implementation=GPS_L1_CA_Telemetry_Decoder
TelemetryDecoder_1C.dump=false

;######### OBSERVABLES CONFIG ############
Observables.implementation=Hybrid_Observables
Observables.dump=false
Observables.dump_filename=./observables.dat
Observables.enable_carrier_smoothing=false
Observables.smoothing_factor=200

;######### PVT CONFIG ############
PVT.implementation=RTKLIB_PVT
PVT.positioning_mode=PPP_Static  ; options: Single, Static, Kinematic, PPP_Static, PPP_Kinematic
PVT.iono_model=Broadcast ; options: OFF, Broadcast, SBAS, Iono-Free-LC, Estimate_STEC, IONEX
PVT.trop_model=Saastamoinen ; options: OFF, Saastamoinen, SBAS, Estimate_ZTD, Estimate_ZTD_Grad
PVT.enable_rx_clock_correction=false
PVT.output_rate_ms=100
PVT.rinexobs_rate_ms=100
PVT.display_rate_ms=500
PVT.dump_filename=./PVT
PVT.nmea_dump_filename=./gnss_sdr_pvt.nmea;
```

```
PVT.flag_nmea_tty_port=false;
PVT.nmea_dump_devname=/dev/pts/4
PVT.dump=false
PVT.flag_rtcm_server=true
PVT.flag_rtcm_tty_port=false
PVT.rtcm_dump_devname=/dev/pts/1
```

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, am 17.08.2023

[Philipp-Sebastian Vogt]