Master's thesis

Håkon Solevåg Vågsether

Integrated Inertial Navigation of UAVs Aided by Phased-Array Radio Measurements

Master's thesis in Cybernetics and Robotics
Supervisor: Torleiv Håland Bryne
Co-supervisor: Kristoffer Gryte
June 2021

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Håkon Solevåg Vågsether

Integrated Inertial Navigation of UAVs Aided by Phased-Array Radio Measurements

Master's thesis in Cybernetics and Robotics
Supervisor: Torleiv Håland Bryne
Co-supervisor: Kristoffer Gryte
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Integrated Inertial Navigation of UAVs Aided by Phased-Array Radio Measurements

Department of Engineering Cybernetics

Håkon Solevåg Vågsether

June 2021

NTNU
Norwegian University of Science and Technology

Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

MSC THESIS DESCRIPTION SHEET

Name:	Håkon Solevåg Vågsether
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Integrert tregleiksnavigasjon av UAV støtta av målingar frå fasestyrt radio.
Thesis title (English):	Integrated inertial navigation of UAVs aided by phased-array radio measurements.

Thesis Description: Global navigation satellite systems (GNSS) receivers are the primary position sensors of unmanned arial vehicle (UAV) navigation systems. GNSS is accurate and globally distributed with multiple redundant systems (GPS, GLONASS, Galileo, BeiDou) being fully or partially operational. These systems are, however, prone to disturbances and interference from both natural and intentional sources due to their low signal strength. A promising GNSS supplement for UAV navigation is phased-array-radio system (PARS)-based positioning. PARS is less accurate than GNSS but is more resilient to malicious denial-of-service attacks.

Existing results on PARS-aided INS for UAV navigation does not exploit altitude information from barometric pressure measurements optimally and rely on a flat Earth assumption. Usage of multiple PARS radio ground beacons are not yet studied

The following tasks for the MSc thesis should be considered.

1. Perform a literature review on
   a. the vulnerability of GNSS to inference
   b. GNSS jamming and spoofing detection methods
2. Develop an INS and INS aiding strategy that supports
   a. long range UAV flights with respect to altitude compensation of PARS range measurements
   b. multiple PARS ground beacons
   c. integration of raw pressure measurements
3. Implement the navigation system in the DUNE software toolchain.
4. Set up an appropriate sensor payload and carry out a flight test in the field.
5. Present and discuss your results based on data collected in the field.
6. Conclude your results and suggest further work.

Start date:	2021-01-06
Due date:	2021-06-02
Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Associate professor Torleiv H. Bryne, Dept. of Eng. Cybernetics, NTNU
Co-supervisor	Researcher Kristoffer Gryte Dept. of Eng. Cybernetics, NTNU

Summary

This Master's thesis discusses the challenges of Unmanned Aerial Vehicle (UAV) navigation in areas where jamming and other forms of radio frequency interference may occur. Although the last few decades has seen several new Global Navigation Satellite System flavors enter the realm of satellite navigation, the risk of operation disruption due to interference is still considerable, and this motivates the investigation of alternative solutions. Many UAV operations are performed within a predefined smaller area, and thus a system using relative positioning based on Phased Array Radio System (PARS) devices is considered.

Positioning, however, only solves half the problem, as the vehicle's orientation must also be known. The orientation can be represented in several ways, and the unit quaternion is chosen for this application. The vehicle's pose is represented relative to the Earth-Centered Earth-Fixed (ECEF) frame, and an estimation method using a Multiplicative Extended Kalman Filter (MEKF) is presented and implemented. This method is tested on several existing datasets, and a new dataset is generated during a field test at Breivika in May 2021. Some augmentations are used, such as PARS measurement outlier rejection and cylindrical coordinates. A height correction method for cylindrical coordinates is also developed and tested. The filter is shown to work well with good data, and it is also shown that it can produce good estimates despite short periods of aiding measurement downtime.

Samandrag

Denne masteroppgåva diskuterer utfordringar ved bruk av ubemanna luftfartøy (UAV) i områder med risiko for jamming og andre former for radiofrekvensforstyrringar. Sjølv om fleire nye satellittnavigasjonssystem har entra den operative fasen er det framleis vanskeleg å sikre seg mot slike angrep, og dette motiverer undersøkinga av alternative løysingar. Mange UAV-operasjonar gjerast innanfor eit førehandsdefinert område, og derfor tek denne oppgåva føre seg eit system som bruker relativ posisjonering basert på målingar frå fasestyrt radioutstyr.

Posisjonering løyser berre halvparten av problemet, ettersom retninga til køyretøyet òg må vere kjend. Retninga kan representerast på fleire måtar, og einingsskvaternionen er valgt for denne applikasjonen. Køyretøyet er representert i Earth-Centered Earth-Fixed (ECEF)-ramma, og ein estimeringsmetode ved bruk av et multiplikativt utvida Kalman-filter (MEKF) presenterast og implementerast som del av denne oppgåva. Denne metoden er testa på fleire eksisterande datasett, og eit nytt datassett er generert ved ein felttest ved Breivika i mai 2021. Eit par forbetringar blir brukte, som avviksavvising av radiomålingane og sylindriske koordinatar. Ein korrigeringsmetode for høgdemålingar er òg utvikla og testa. Filteret viser seg å fungere bra med gode datasett, og det er vist at det kan gi gode estimat til tross for korte periodar med nedetid for radiomålingane.

Preface

This thesis is carried out at the Department of Engineering Cybernetics and is submitted in partial fulfillment of the requirements for the degree of Master of Science in Cybernetics and Robotics.

I would like to thank my supervisor, Torleiv Håland Bryne, for his support and invaluable notes on how to write reports. I would like to thank my co-supervisor, Kristoffer Gryte, for his help with debugging the implementation and his notes on how to set up DUNE with sensors on the physical drone. I would also like to thank Pål Kvaløy and Mika Okuhara for their support with the field test at Breivika. Finally, I would like to express my gratitude to Martin Sollie, who kindly helped me with configuring the STIM300 IMU.

Håkon Vågsether
June 2021

Contents

List of Figures

List of Tables

Abbreviations

AGC Automatic Gain Control. 16

ARS Angular Rate Sensor. xi, 2, 7, 9, 14, 26, 31, 36, 51, 57, 59–61, 67, 69, 73

BVLOS Beyond Visual Line of Sight. 48

CRC Cyclic Redundancy Check. 37

DNS Domain Name Service. 43

DUNE DUNE Unified Navigation Environment. vii, 2, 41, 42, 44, 64, 65, 73, 77

ECEF Earth-Centered Earth-Fixed. iii, xi, 2, 9, 11, 12, 16, 20, 22, 23, 27, 57, 73, 74

ECI Earth-Centered Inertial. 10, 11, 14

EKF Extended Kalman Filter. 26

ENU East-North-Up. 12

ESA European Space Agency. 15

FOC Full Operating Capability. 15

GLONASS Global Navigation Satellite System. 1

GNSS Global Navigation Satellite System. iii, 1–3, 14–17, 20, 43, 44, 64, 73

GNU GNU's Not Unix. 36

GPS Global Positioning System. 1, 15, 16

IMC Intermodule Communication API. 41, 42, 65

IMU Inertial Measurement Unit. 11, 14, 26, 36, 43, 53, 57, 64, 70, 73, 74

INS Inertial Navigation System. 26, 31, 43, 73

IP Internet Protocol. 43, 44, 65

ITK Department of Engineering Cybernetics (Institutt for Teknisk Kybernetikk). 77

MAC Media Access Control. 44, 65

MEKF Multiplicative Extended Kalman Filter. iii, 2, 3, 26, 29, 65, 73, 77

MEMS Micro-Electro-Mechanical Systems. 36

MRP Modified Rodrigues Parameters. 7

MSS Marine Systems Simulator. 13

NED North-East-Down. xi, 5, 10–12, 19, 21, 22, 57

NTNU Norwegian University of Science and Technology. 40, 42, 77

OS Operating System. 40

PARS Phased Array Radio System. iii, xii, 1, 2, 9, 10, 18, 20, 22, 23, 26, 28, 29, 31, 43–45, 47, 48, 55, 57–61, 64, 65, 67, 73, 74

PCB Printed Circuit Board. 40

ROS Robot Operating System. 41

RTT Round-trip time. 18

SPI Serial Peripheral Interface. 40

SSH Secure SHell. 43, 44

TCP Transmission Control Protocol. 64

UART Universal Asynchronous Receiver-Transmitter. 40

UAV Unmanned Aerial Vehicle. iii, 1, 10, 12, 19, 20, 22, 47, 48, 50, 56, 57, 59, 64, 65, 67

UDP User Datagram Protocol. 44, 65

USB Universal Serial Bus. 36, 43

UWB Ultra-Wideband. 1

VPN Virtual Private Network. 43

WGS84 World Geodetic System 1984. 12

1 Introduction

Global Navigation Satellite System (GNSS) is a positioning and navigation system with global coverage. Its properties make it possible for virtually unlimited users to receive uninterrupted service, but these properties also leave the users vulnerable to electromagnetic interference, such as jamming, spoofing and meaconing. Newer GNSS receivers are able to filter out some forms of interference, but this is dependent on the make and model of the receiver. The rise of autonomous vehicles leads to an increasing demand for resilient and reliable navigation solutions, and this demand forms the motivation for this thesis. The jamming threat is becoming increasingly problematic, and as an example, the last few years have seen Norwegian medical helicopters being disturbed by jammers on several occasions. (Adresseavisen, 2021), (Aftenposten, 2019)

Systems such as Global Positioning System (GPS) and Global Navigation Satellite System (GLONASS) are expensive, and some properties such as global coverage are difficult, if not impossible, to reproduce without creating yet another Global Navigation Satellite System. Therefore, resources are being spent on investigating ways to achieve reliable navigation and positioning without relying on GNSS. This thesis focuses on the use of Phased Array Radio System devices' range and angle estimates to produce an alternative position measurement, and its viability as an auxiliary sensor in case of GNSS outages.

Position estimation can be achieved with several different methods and technologies. Cameras (Mourikis and Roumeliotis, 2007) are a viable solution, and key features in the surrounding terrain can be tracked in order to estimate the UAV's movement. The initial position must be given to the system, but such a system requires no further information from a base station, which is very desirable. However, there are many factors that can impact the performance of such navigation solutions, like lighting conditions and visual terrain properties. For example, some areas have very few terrain features that stand out, such as flat sand dunes. Other areas, like the ocean, may have features such as waves or sea spray, but these are moving and might distort the navigation system's conception of movement. Electromagnetic waves are popular for wireless communication, and Ultra-Wideband (UWB)-based systems (Mahfouz et al., 2008) can be used for positioning with high accuracy. Systems of this type are typically used for indoor positioning and other short-range applications, and are not well suited for UAV flights due to their short range.

PARS-based positioning has several merits compared to the other technologies mentioned above. It is independent of lighting conditions, and can operate at long ranges. It requires constant connection to a base station, and the use of radio waves make it susceptible to jamming (albeit less so than GNSS). It could also be argued that PARS devices' primary function, which is communication, make up for this to some extent by eliminating the need for a separate communication system. PARS-based positioning is thus quite convenient. That said, there are still a few challenges related to PARS-based positioning, such as the susceptibility to reflections, which is prominent when flying over water or near large buildings. It also requires radio line-of-sight, which means that the base stations must be appropriately placed. (Gryte et al., 2019)

This subject has been a focus area recently, with Kristoffer Gryte's doctoral thesis (Gryte, 2020) being the most notable example. It also includes the implementation and testing of an Multiplicative Extended Kalman Filter (MEKF), and some of the data sets used in this project have originally been recorded by Gryte. Sollie et al. (2019) includes an Earth-Centered Earth-Fixed (ECEF)-based MEKF implementation, just like this report. However, the focus of that thesis is more towards pose estimation using multiple receivers. ECEF is chosen because it coincides with the output of GNSS receivers and is not inherently affected by the roundness of the Earth. Albrektsen et al. (2018) should also be mentioned, where position and attitude estimation is achieved using separate nonlinear observers for translational and angular motion. This thesis uses an MEKF, because it couples the position and orientation such that the heading can be estimated without a dedicated heading sensor. Note that this comes at a cost, namely increased computational footprint and the lack of proven stability conditions. (Gryte, 2020)

1.1 Main contributions

This thesis builds on my specialization project (Vågsether, 2020). The following contributions are made:

- The navigation system from Vågsether (2020) is implemented in DUNE and modified to suit live flight operations. Various improvements such as Angular Rate Sensor (ARS) calibration and PARS measurement outlier rejection is added.
- The navigation system is tested with prerecorded data from flights at Raudstein, Orkland municipality and Udduvoll, Melhus/Trondheim municipality.
- A flight test is performed at Breivika, Orkland municipality, and the navigation system is tested with data from that flight.
- A height error compensation method for barometer-aided cylindrical PARS positioning over long distances is developed and evaluated using data from the Raudstein flight.

1.2 Outline of this thesis

Chapter 2 consists of the theoretical background for this thesis, from the properties of quaternions to the vulnerabilities of GNSS to electromagnetic interference. Section 2.2 gives an overview of important symbols and characters. Chapter 3 sets the theory in perspective, explaining how it is used to form the basis of a Multiplicative Extended Kalman Filter (MEKF). The implementation is run and evaluated in Chapter 4, and Chapter 5 recounts the most important results and gives suggestions on further steps. Finally, Appendix A contains a brief derivation of a standard Kalman filter.

Parts of this thesis are based on my specialization project (Vågsether, 2020), and the beginnings and ends of these parts will be marked with the symbols [†] and [††], respectively.

[†]The following text is based on (Vågsether, 2020)

[††]The above text is based on (Vågsether, 2020)

2 Background and Preliminaries

2.1 Attitude and quaternions

The term "Navigation" can be defined as "The process or activity of accurately ascertaining one's position and planning and following a route". (lexico.com, 2021) This is a good definition, but it fails to highlight that many applications require determining one's orientation too. Airborne vehicles consume a great deal of energy just to stay in the air, and they are fully responsible for controlling their own orientation. Cars or ships on the other hand are to some extent restricted by the surrounding terrain or water. Orientation is described as the rotation relative to some reference point, and it can be in one or several dimensions. The rotation of a physical object in a global context requires the use of 3 dimensions, which can be complicated. There are several ways to represent a three-dimensional rotation, each with different properties. Some of these representations have glaring disadvantages, which complicate their use further. [†] The group of all 3D rotations is called SO(3). Members of SO(3) can be described using 3x3 rotation matrices. These matrices are not particularly intuitive or compact as a form of representation, but they have a few properties that make them easier to work with than arbitrary matrices. The most important ones are:

$$\boldsymbol{R}^T = \boldsymbol{R}^{-1} \tag{2.1a}$$

$$\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}_3 \tag{2.1b}$$

$$\det \boldsymbol{R} = 1 \tag{2.1c}$$

More compact representations are often used, such as Euler angles or the angle-axis representation. Euler angles are used in this report mainly to describe rotation relative to the North-East-Down (NED) reference frame, following the ZYX convention. This representation involves the roll, pitch and yaw angles (see Figure 2.1), and are converted to a rotation matrix using the following formula (Fossen, 2011):

$$\boldsymbol{R}(\phi, \theta, \psi) = \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\phi) \tag{2.2}$$

with ϕ, θ, ψ being the rotation about the x, y and z axis, respectively. The rotation matrix subscript indicates which axis the rotation is performed around. This is an intuitive representation, but it comes at a cost, such as the added risk of gimbal lock and singularities. It is also not particularly straightforward to determine the difference between two rotations, and this is where quaternions come in. Quaternions have made quite an impact on the world of spatial rotations, despite their reputation of being difficult to understand. A quaternion is a complex

[†]The following text is based on (Vågsether, 2020)

number with one real and three imaginary components:

$$Q = a + bi + cj + dk \tag{2.3}$$

with coefficients $\{a, b, c, d\} \in \mathbb{R}$ and imaginary unit numbers $\{i, j, k\}$. The latter 3 coefficients are often grouped together in a vector $\boldsymbol{q}_v = (b, c, d)$. The remaining coefficient, a, is typically denoted q_w. These two components are combined to form a 4-element vector $\boldsymbol{q}$:

$$\boldsymbol{q} = \begin{bmatrix} q_w \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{2.4}$$

The concept of multiplication exists for quaternions, associated with the $\otimes$ operator (Solà, 2017):

$$\boldsymbol{q} \otimes \boldsymbol{r} = \begin{bmatrix} q_w r_w - \boldsymbol{q}_v^T \boldsymbol{r}_v \\ q_w \boldsymbol{r}_v + r_w \boldsymbol{q}_v + \boldsymbol{q}_v \times \boldsymbol{r}_v \end{bmatrix} \tag{2.5}$$

This can also be written in matrix form:

$$\boldsymbol{q} \otimes \boldsymbol{r} = [\boldsymbol{q}]_L \boldsymbol{r} = (q_w \boldsymbol{I} + \begin{bmatrix} 0 & -\boldsymbol{q}_v^T \\ \boldsymbol{q}_v & \boldsymbol{S}(\boldsymbol{q}_v) \end{bmatrix}) \boldsymbol{r} \tag{2.6}$$

$$\boldsymbol{q} \otimes \boldsymbol{r} = [\boldsymbol{r}]_R \boldsymbol{q} = (r_w \boldsymbol{I} + \begin{bmatrix} 0 & -\boldsymbol{r}_v^T \\ \boldsymbol{r}_v & -\boldsymbol{S}(\boldsymbol{r}_v) \end{bmatrix}) \boldsymbol{q} \tag{2.7}$$

In order to describe 3D rotations with quaternions, we have to constrain the norm:

$$|\boldsymbol{q}| = q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1 \tag{2.8}$$

This is called a unit quaternion, and all quaternions discussed in this report fall under this category. The conjugate of a quaternion is defined (Solà, 2017):

$$(\boldsymbol{q})^* = \begin{bmatrix} q_w \\ -\boldsymbol{q}_v \end{bmatrix} = \begin{bmatrix} q_w \\ -q_x \\ -q_y \\ -q_z \end{bmatrix} \tag{2.9}$$

For unit quaternions, the following relationships hold:

$$(\boldsymbol{q} \otimes \boldsymbol{r})^* = \boldsymbol{r}^* \otimes \boldsymbol{q}^* \tag{2.10}$$

$$(\boldsymbol{q})^* \otimes \boldsymbol{q} = \begin{bmatrix} 1 \\ \mathbf{0}_3 \end{bmatrix} \tag{2.11}$$

$$\begin{bmatrix} 1 \\ \mathbf{0}_3 \end{bmatrix} \otimes \boldsymbol{q} = \boldsymbol{q} \tag{2.12}$$

We also have the following relationship for the time derivative of a quaternion:

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \tag{2.13}$$

where $\boldsymbol{\omega}$ is the the angular rate vector in the local frame defined by $\boldsymbol{q}$. (Solà, 2017) Differences between unit quaternions can be calculated in several ways, some listed in Markley (2003). In this report the four times Modified Rodrigues Parameters (MRP) method will be used exclusively. Differences can be injected into quaternions, not through the use of addition, but multiplication:

$$\boldsymbol{q} = \boldsymbol{r} \otimes \delta\boldsymbol{q} \tag{2.14}$$

where $\delta\boldsymbol{q}$ is the error between the quaternions $\boldsymbol{q}$ and $\boldsymbol{r}$. The error quaternion is calculated using the four times MRP method given in Markley (2003):

$$\delta\boldsymbol{q}(\delta\boldsymbol{\theta}) = \frac{1}{16 + \delta\boldsymbol{\theta}^T\delta\boldsymbol{\theta}} \begin{bmatrix} 16 - \delta\boldsymbol{\theta}^T\delta\boldsymbol{\theta} \\ 8\delta\boldsymbol{\theta} \end{bmatrix} \tag{2.15}$$

The following formula from Solà (2017) is used to calculate the incremental rotation quaternion from the Angular Rate Sensor (ARS) using the angular rate $\boldsymbol{\omega}$ and the sample period T_s:

$$\boldsymbol{q}(\boldsymbol{\omega}, T_s) = \begin{bmatrix} \cos\left(\frac{T_s|\boldsymbol{\omega}|}{2}\right) \\ \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|}\sin\left(\frac{T_s|\boldsymbol{\omega}|}{2}\right) \end{bmatrix} \tag{2.16}$$

A quaternion can be rotated with this angular rate by taking the quaternion product of the original quaternion and $\boldsymbol{q}(\boldsymbol{\omega}, T_s)$. Finally, a rotation matrix can be computed from a quaternion through the following formula (Fossen, 2011):

$$\boldsymbol{R}(\boldsymbol{q}) = \boldsymbol{I}_3 + 2q_w\boldsymbol{S}(\boldsymbol{q}_v) + 2\boldsymbol{S}(\boldsymbol{q}_v)^2 \tag{2.17}$$

with $\boldsymbol{S}(\cdot)$ denoting the skew symmetric matrix operator:

$$S\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{2.18}$$

[††] The skew symmetric matrix operator has a few useful properties, such as $\boldsymbol{S}(v)v = \boldsymbol{0}$. This is shown below:

$$\boldsymbol{S}\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right)\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} yz - yz \\ xz - xz \\ xy - xy \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.19}$$

Also, the product $\boldsymbol{S}(\boldsymbol{a})\boldsymbol{b}$ can be written as $\mathbf{S}(\boldsymbol{b})\boldsymbol{a}$, as shown below:

[††]The above text is based on (Vågsether, 2020)

$$\boldsymbol{S}\left(\begin{bmatrix}a_1\\a_2\\a_3\end{bmatrix}\right)\begin{bmatrix}b_1\\b_2\\b_3\end{bmatrix}=\begin{bmatrix}0&-a_3&a_2\\a_3&0&-a_1\\-a_2&a_1&0\end{bmatrix}\begin{bmatrix}b_1\\b_2\\b_3\end{bmatrix}=\begin{bmatrix}-a_3b_2+a_2b_3\\a_3b_1-a_1b_3\\-a_2b_1+a_1b_2\end{bmatrix} \tag{2.20a}$$

$$-\boldsymbol{S}\left(\begin{bmatrix}b_1\\b_2\\b_3\end{bmatrix}\right)\begin{bmatrix}a_1\\a_2\\a_3\end{bmatrix}=-\begin{bmatrix}0&-b_3&b_2\\b_3&0&-b_1\\-b_2&b_1&0\end{bmatrix}\begin{bmatrix}a_1\\a_2\\a_3\end{bmatrix}=\begin{bmatrix}-a_3b_2+a_2b_3\\a_3b_1-a_1b_3\\-a_2b_1+a_1b_2\end{bmatrix} \tag{2.20b}$$

2.2 Notation

[†]Table 2.1 describes symbols and operator used in this thesis.

Table 2.1: Reserved symbols and operators.

Symbol	Explanation
$\boldsymbol{p}^a_{bc}$	The vector from b to c, decomposed in the $\{a\}$ frame. Used to express positions.
$\boldsymbol{v}^a_{bc}$	The velocity of c relative b, decomposed in the $\{a\}$ frame.
$\boldsymbol{q}^a_{bc}$	A quaternion representation of the rotation of c relative b, decomposed in the $\{a\}$ frame. $q^a_{bc,w}$ is the real/scalar part, $\boldsymbol{q}^a_{bc,v}$ is the imaginary/vector part.
$\boldsymbol{f}^a_{bc}$, $\boldsymbol{a}^a_{bc}$	The specific force and acceleration of c relative b, decomposed in the $\{a\}$ frame.
$\boldsymbol{g}^e$	Gravitational acceleration vector, decomposed in ECEF.
$\boldsymbol{\omega}^a_{bc}$	The angular rate of c relative b, decomposed in the $\{a\}$ frame.
$\boldsymbol{b}^a_{acc}$, $\boldsymbol{b}^a_{ars}$	Accelerometer and Angular Rate Sensor biases, decomposed in the $\{a\}$ frame.
$\boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{P}$	Process noise, measurement noise and estimate covariance matrix.
$\boldsymbol{P}^-, \boldsymbol{P}^+$	A priori and a posteriori (before and after correction) estimate covariance matrix.
Ψ, α	PARS azimuth and elevation.
ϕ, θ, ψ	Roll, pitch, yaw angles.
ϕ, λ, h	Latitude, longitude, height (WGS84).
$\dot{x}$	The first time derivative of x. Two dots indicate the second time derivative, and so on.
$\hat{x}$	An estimate or prediction of x.
$\boldsymbol{x}$	A vector.
$\boldsymbol{X}$	A matrix.
$\boldsymbol{R}_{en}$	A rotation matrix, such that $\boldsymbol{p}^e_{eb} = \boldsymbol{R}_{en}\boldsymbol{p}^n_{eb}$.
p_i	The ith element of the $\boldsymbol{p}$ vector.
P_{ij}	The element corresponding to the ith row, jth column of the matrix $\boldsymbol{P}$.
$\ln(\cdot)$, $e^{\cdot}$	The natural logarithm and exponential operator.
μ, σ, σ^2	Mean, standard deviation and variance.
$\otimes$	Quaternion product
$\boldsymbol{S}(\cdot)$	Skew-symmetric matrix operator
$\mathbb{E}[\cdot]$	Expectation operator
$\delta\cdot$	Error state.
ε	Zero-mean Gaussian noise.

[†]The text below is based on (Vågsether, 2020)

2.3 Reference frames

The use of reference frames is vital when developing navigation systems. The application is of great importance, and while a spacecraft might need to use an advanced coordinate system such as ECI, an agricultural robot can do with a simple base station-centered NED-based frame. Vectors and angles can be decomposed in different frames, expressed by the superscript notation. The following frames are used in this report:

Body frame

The $\{b\}$ (body) frame is centered at the UAV's center of gravity and moves with the origin coincides with the $\{m\}$ frame, but its axes are rotated such that the x axis points towards the front of the vehicle and the z axis points downwards. The $\{b\}$ frame is equal to the NED frame in the event of 0 ° roll, pitch and yaw, shown in Figure 2.1. The terms "body frame" and "$\{b\}$ frame" will be used interchangeably throughout this report, and both refer to the UAV-centered reference frame unless another device is specified. The $\{i\}$ frame will also be used, which is the inertial frame.

Radio (base station) frame

Two different body frames will be used, the $\{b\}$ frame (for the UAV) and the $\{r\}$ frame (for the PARS base station). The $\{r\}$ frame is similarly defined, with the x axis pointing out of the front of the base station antenna, coinciding with the 0 ° elevation and azimuth axis.

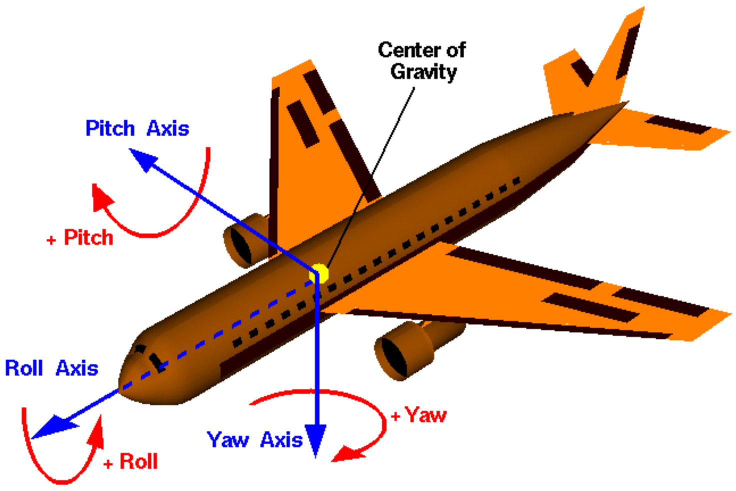

Figure 2.1: An aircraft model with its body axes drawn in. (NASA, 2006)

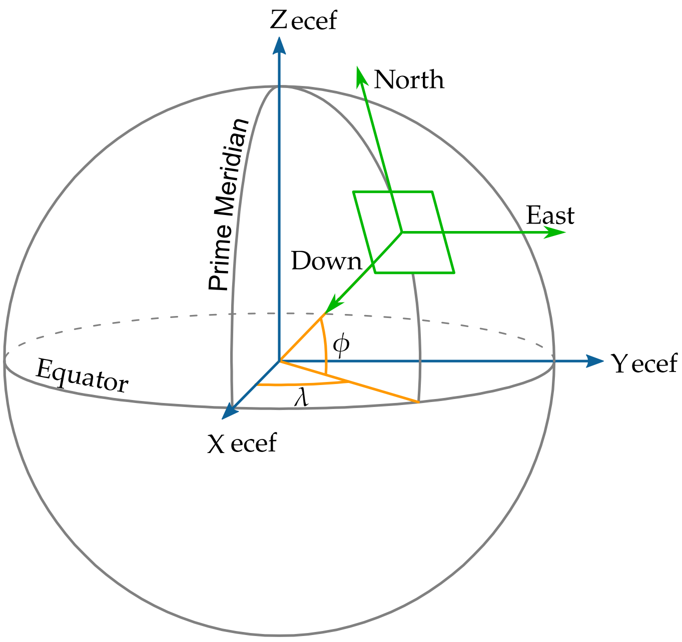

Figure 2.2: The ECEF (blue) and NED (green) frames.

Measurement frame

The $\{m\}$ (measurement) frame is the frame in which the vehicle's movement is measured. The $\{m\}$ frame is centered at the body frame's origin, but rotated such that the vector $\boldsymbol{w}^b = \boldsymbol{R}_{bm}\boldsymbol{w}^m$. The measurement frame in this project will be set to match the frame of the Inertial Measurement Unit (IMU), and thus the rotation matrix $\boldsymbol{R}_{bm}$ depends on how the IMU is mounted on the vehicle.

ECEF frame

The $\{e\}$ (Earth-Centered Earth-Fixed) frame is centered at the Earth's core and rotates with the Earth. The x axis points towards the point where the equator meets the prime meridian, and the z axis points upwards through the geographic North Pole. The y axis is perpendicular to these axes, such that the cross product of the x axis and the y axis is the z axis (following the right hand rule). The ECEF frame is shown in blue in Figure 2.2.

ECI frame

The Earth-Centered Inertial (ECI) frame is nearly identical to the ECEF frame, but it does not rotate with the Earth. Instead, the x axis is fixed in the direction of the vernal equinox. It is useful for satellites and spacecraft, and it will not be used in this report.

NED frame

The $\{n\}$ (North-East-Down) frame is centered at the UAV, but its x, y and z axes are always pointing towards North, East and Down, respectively. Some sources refer to the ENU frame. (Wei et al., 2019); (Giorgi et al., 2010) This is just a flipped version of the NED frame, and they serve the same purpose for all applications. The NED frame is shown in green in Figure 2.2.

Geodetic representation

In contrast to the other frames, the geodetic "frame" is not quite a frame, but a spherical coordinate system given in relation to the Cartesian ECEF frame. The height component is given in relation to the ellipsoid model. The zero latitude, zero longitude axis coincides with ECEF's x axis. The latitude-longitude-height convention is commonly used, due to its readability for humans. However, it is only used as a middle ground between ECEF and NED in this report. The position vector is specified in Cartesian ECEF coordinates. (Misra and Enge, 2011)

Conversions

The use of several reference frames requires the knowledge of how to translate between them. This report uses the WGS84 standard, and a few conversions are recounted below, taken from Fossen (2011). Conversion from NED to ECEF is simple, and only requires a rotation matrix:

$$\boldsymbol{R}_{en}(\phi, \lambda) = \begin{bmatrix} -\sin\phi\cos\lambda & -\sin\lambda & -\cos\phi\cos\lambda \\ -\sin\phi\sin\lambda & \cos\lambda & -\cos\phi\sin\lambda \\ \cos\phi & 0 & -\sin\phi \end{bmatrix} \tag{2.21}$$

Where ϕ is the latitude, λ is the longitude of the NED frame's origin. The inverse operation can be carried out by simply transposing the matrix. Conversion between ECEF and geodetic representation are not as simple. The steps are given in Fossen (2011), and are revisited below. The ECEF coordinates corresponding to a geodetic position can be calculated directly:

$$x \leftarrow (N + h)\cos(\phi)\cos(\lambda) \tag{2.22a}$$

$$y \leftarrow (N + h)\cos(\phi)\sin(\lambda) \tag{2.22b}$$

$$z \leftarrow \left(\frac{r_p^2}{r_e^2}N + h\right)\sin(\phi) \tag{2.22c}$$

with $r_e = 6378137$ m and $r_p = 6356752.314245$ m being the WGS84 ellipsoid's semimajor and semiminor axis, respectively. N is calculated as a function of these values and the latitude:

$$N \leftarrow \frac{r_e^2}{\sqrt{r_e^2 \cos(\phi)^2 + r_p^2 \sin(\phi)^2}} \tag{2.23}$$

The reverse operation is done using an iterative scheme. The first three steps are done only once:

$$\lambda \leftarrow \arctan\left(\frac{y}{x}\right) \tag{2.24a}$$

$$p \leftarrow \sqrt{x^2 + y^2} \tag{2.24b}$$

$$e \leftarrow \sqrt{1 - (r_p/r_e)^2} \tag{2.24c}$$

In addition, a preliminary value for the latitude is found:

$$\phi \leftarrow \arctan\left(\frac{z}{p(1-e^2)}\right) \tag{2.25}$$

The rest is the iterative part, and the following steps are reiterated until η is lower than some error tolerance limit. This project's implementation uses the Marine Systems Simulator (MSS) (cybergalactic/Fossen, 2020) MATLAB library, where this tolerance limit is set to 1e-10.

$$N \leftarrow \frac{r_e^2}{\sqrt{r_e^2 \cos(\phi)^2 + r_p^2 \sin(\phi)^2}} \tag{2.26a}$$

$$h \leftarrow \frac{p}{\cos(\phi)} - N \tag{2.26b}$$

$$\phi_0 \leftarrow \phi \tag{2.26c}$$

$$\phi \leftarrow \arctan\left(\frac{z}{p(1 - e^2 N/(N+h))}\right) \tag{2.26d}$$

$$\eta \leftarrow |\phi - \phi_0| \tag{2.26e}$$

2.4 IMU

An Inertial Measurement Unit (IMU) is often used for navigational purposes, as it supplies information on the sensor's movement relative to the inertial frame (ECI). Mounting the sensor on a vehicle means that the sensor will move and rotate with the vehicle, and as such we can use the sensor to inform us about the vehicle's movement. This is called strapdown navigation. IMUs consist of an accelerometer and an Angular Rate Sensor (ARS), the former measuring the sensor's linear specific force and the latter measuring the sensor's angular velocity. Integrating these measurements enables us to get an estimate of the sensor's position, but it is trivial to see that this position will be local, or relative to the initial position. These IMU-based position estimates are subject to drift due to the bias and noise properties of the sensors, and an additional system, such as GNSS, is often employed to mitigate this. GNSS can not be considered a replacement for IMUs, as IMUs are high rate sensors with a different role.

Accelerometers measure linear specific force, which is a measure of acceleration in relation to free fall. This means that an accelerometer at rest will measure about 9.81 m/s^2 due to the force exerted on the vehicle by the ground. The accelerometer measurements are

$$\boldsymbol{f}^m = \boldsymbol{f}^m_{ib} + \boldsymbol{b}^m_{acc} + \boldsymbol{\varepsilon}^m_{acc} \tag{2.27}$$

such that $\boldsymbol{\varepsilon}^b_{acc}$ is zero mean Gaussian noise and $\boldsymbol{b}^b_{acc}$ is bias term, treated as a first-order Gauss-Markov model:

$$\dot{\boldsymbol{b}}^b_{acc} = -\boldsymbol{T}^{-1}_{acc}\boldsymbol{b}^b_{acc} + \boldsymbol{w} \tag{2.28}$$

where $\boldsymbol{T}_{acc} = T_{acc}\boldsymbol{I}_3$ is a time constant and $\boldsymbol{w}$ is a white noise process. (J. Russell Carpenter, 2018) Angular Rate Sensors measure the sensor's angular motion in rad/s, and the measurements follow a corresponding model

$$\boldsymbol{\omega}^m = \boldsymbol{\omega}^m_{ib} + \boldsymbol{b}^m_{ars} + \boldsymbol{\varepsilon}^m_{ars} \tag{2.29}$$

with corresponding error term and bias

$$\dot{\boldsymbol{b}}^b_{ars} = -T^{-1}_{ars}\boldsymbol{b}^b_{ars} + \boldsymbol{w} \tag{2.30}$$

with bias time constant $\boldsymbol{T}_{ars} = T_{ars}\boldsymbol{I}_3$. (Gryte, 2020)

2.5 Barometers

A barometer is a device that measures the ambient air pressure. The ambient air pressure decreases with increasing distance from Earth's surface, and thus a barometer can be used to estimate one's height. The ambient air pressure is impacted by other atmospheric conditions, and thus a barometer is often calibrated against a reference station on the ground. The following model is used to find the height h_b as a function of the instantaneous air pressure p_b:

$$h_b(p_b) = \frac{T_s}{k_T}\left[\left(\frac{p_b}{p_s}\right)^{-\frac{RK_T}{g_0}} - 1\right] + h_s \tag{2.31}$$

where p_s and T_s represent the surface pressure and temperature at the geodetic height h_s. The other constants, R, g_0 and k_T are the gas constant (287.1 Jkg^{-1}K^{-1}), the average surface acceleration (9.80665 ms^{-2}) and the atmospheric temperature gradient (6.5e-3 Km^{-1}), respectively. (Groves, 2008)

2.6 GNSS

Global Navigation Satellite System (GNSS) is a type of system that utilizes satellites in known orbits around the Earth in order to provide navigation services. Its most famous variant, GPS, has been in service for 25 years (Full Operating Capability (FOC) declared on 17 July 1995 (Subirana et al., 2013)) and is still being developed and improved, with the Block III satellites scheduled to be put into orbit by the end of 2023. While GPS is an American invention, its Russian and Chinese counterparts, GLONASS and BeiDou, have been in development for several decades. In addition, the European Space Agency (ESA) is developing a fourth system called Galileo. Each system also includes a ground segment, a set of ground stations tasked with monitoring and ensuring that the satellites are functioning correctly.

The basic principle of GNSS is the transmission of a carrier wave from each of the satellites. This carrier is modulated with a repeating sequence, like a code or a message, such that the receiver can lock on to the signal and retrieve data from the message. The message contains a timestamp from the satellite's internal clock, and it allows the receiver to tell when the message was sent. The pseudorange is a term reserved for the estimated geometric distance from the satellite to the receiver, based on the estimated time of flight. The pseudorange equation can be found in Misra and Enge (2011) and has several components:

$$\rho = r + c(\delta t_u - \delta t^s) + I_\rho + T_\rho + \varepsilon_\rho \tag{2.32}$$

Where

- ρ is the pseudorange

- r is the true geometric range
- c is the speed of light
- δt_u is the receiver's clock error
- δt^s is the satellite's clock error
- I_ρ is the ionospheric delay
- T_ρ is the troposheric delay
- ε_ρ is reserved for other error sources, such as multipath and receiver noise

A GNSS receiver can use pseudoranges from four satellites to estimate its own position through trilateration. This means that the satellite and the receiver has a one-way type of communication, and the number of users that can be served by a group of satellites at a given point in time is theoretically limitless. This also means that the receiver does not need transmission capabilities, which leads to low complexity, power consumption and cost. The global coverage of the space segment (satellite network) of today's GNSS systems makes the use of receivers convenient, easy and fairly accurate outside of high latitude areas. (Swaszek et al., 2018) GNSS yields the receiver's position in ECEF, (or geodetic after converting from ECEF) and as a result of this, the measurement does not need conversion when used with an ECEF-based navigation system.

2.7 GNSS vulnerabilities

GNSS' convenience and ease of use comes at a price. The low receiver signal at the receiver makes it an easy target for intentional destructive interference such as jamming, spoofing and meaconing. Jamming is the most simple form, and it can be done in several ways. The topic of jamming GPS receivers is nothing new (Pinker and Smith, 1999), while other systems currently lack research on this area. The effectiveness depends on the receiver, and some receivers are reported to be quite resilient against some types of jamming. Jamming leads to loss of lock, and the receiver typically will not be able to get a fix until the jamming ends or the receiver exits the jamming equipment's effective area.

Spoofing is a more sophisticated strategy, but it is very complicated compared to jamming. A successful spoofing attack involves capturing the victim receiver's lock and fooling it to generate false position estimates. Pulling off such an attack requires large computing power, and there are several reasons for an attempt to fail, as listed in Schmidt et al. (2016). A few of the major points are recounted:

- The receiver's Automatic Gain Control (AGC) may notice an abrupt increase in the received signal power. This could cause loss of lock and alert the receiver about the existence of a malicious signal source. The spoofing signal's received power must be estimated and controlled, and this requires knowledge about the victim's position.

- The spoofer might not be able to tell whether the victim receiver has locked on to the spoofing signal or not, and it is possible for the receiver to lock on to the legitimate signal while the spoofing signal is being transmitted.
- Many vehicles have compasses as an auxiliary heading sensor, and a difference between the compass and the satellite-aided navigation computer might alert a human operator to the possibility of spoofing. A human operator might also notice that the movement on the map does not match the movement outside the windows, but this is not as likely for vehicles at high altitude or at sea. In any case, these considerations are less critical in unmanned applications.
- The navigation message relayed by the GNSS signal contains information about the satellites' orbital parameters, and this enables the receiver to calculate each satellite's approximate position several hours ahead in time. Depending on the receiver's complexity, a spoofer might have to simulate the entire constellation in order so as not to send a fake signal from a satellite that is not really in sight from the victim. The spoofer can generate its own virtual constellation and transmit a fake navigation message, but this will also require simulation.

Meaconing is a third alternative, and in a sense, it is a simple form of spoofing. Meaconing is the capture and retransmission of GNSS signals, and this results in the victim believing to be in the same position as the meaconer. This happens because the meaconer retransmits the signals exactly as they were received, and since they are now all coming from the same source, the relative arrival times will be unchanged when they arrive at the victim receiver. This method is thus significantly simpler, but less useful than other forms of spoofing. However, a successful spoofing operation on an unknowing victim remains little short of herculean, and Schmidt's survey describes GNSS spoofing as a future (i.e. not a current) threat.

Radio interference will typically lead to a drop in the receiver's carrier-to-noise ratio (C/N_0), and this can be used to detect an active malicious agent. There are ways to protect a receiver from interference such that it is able to obtain a valid fix while the interference source is active. One way is to couple the receiver with a navigation system, much like what is done in this report. Another approach is spatial filtering. An aircraft can use phased-array antennas to attenuate signals coming from below, making it harder for a jammer to capture the lock of the aircraft's receiver. (Gao et al., 2016)

2.8 Phased Array Radio Systems

A Phased Array Radio System is a type of radio system utilizing multiple antennas in order to achieve directionality. The basic principle is that electromagnetic waves travel at the speed of light, and two receivers will receive the signal at different times given that there is a distance between them. With an appropriate and known signal frequency and distance between the receivers, the signal's originating direction can be determined. The delay will present itself as a phase offset, and the principle can be reversed to electronically steer the signal beam, achieving spatial directionality without using directional antenna elements. This notion of steering is often referred to as beamsteering. This eliminates the need for mechanically steered parabolic antennas, and thus eliminates the need for motor maintenance. A phased array system can stay in contact with several units in different directions without losing directional gain, which is another advantage over traditional mechanically steered directional antennas. The use of PARS has a few drawbacks, like increased cost and complexity. Also, a phased array is designed to work on a specific frequency, since the distance between the elements depend on the wavelength. As such, a PARS system must stay in its intended (and fairly limited) frequency range in order to function correctly. (Herd and Conway, 2015)

The estimated direction can be utilized together with an estimate of the distance, obtained through methods such as the Round-trip time (RTT). These three values form a spherical coordinate system and are enough to determine a remote device's position relative to a base station.

PARS-based positioning - Spherical coordinates

PARS devices can output the distance, elevation and azimuth angle from the base station to the remote device. Hence, for these variables to be used for positioning, the base station's position and orientation needs to be known in order to compute the absolute position. The calculations presented in this and the next subsection is based on Gryte (2020). For this project the base's roll angle is assumed to be zero. Each measurement is subjected to a zero mean Gaussian noise component (gathered in the column vector $\boldsymbol{\varepsilon}_s$):

$$d = d_u + \varepsilon_d \tag{2.33a}$$

$$\Psi = \Psi_u + \varepsilon_\Psi \tag{2.33b}$$

$$\alpha = \alpha_u + \varepsilon_\alpha \tag{2.33c}$$

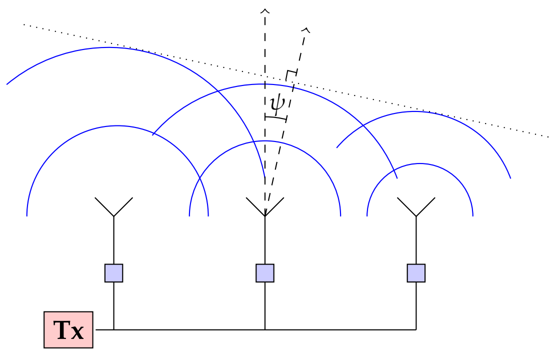

Figure 2.3: A phased array antenna, illustrating its beamsteering capabilities. The blue boxes represent the phase shifting mechanism.

where $\cdot_u$ is the true value and $\varepsilon_\cdot$ is the noise. The measurement-based vector from the base to the UAV (decomposed in the $\{r\}$ frame) is

$$\boldsymbol{p}^r_{rb,s} = d \begin{bmatrix} \cos(\Psi)\cos(\alpha) \\ \sin(\Psi)\cos(\alpha) \\ -\sin(\alpha) \end{bmatrix} \tag{2.34}$$

with the distance d, the azimuth angle Ψ and the elevation angle α. The $\{r\}$ frame will typically be rotated in relation to the base station's NED frame. Assuming an angle of 0 ° for roll, the vector from the base to the UAV (now decomposed in the base station's NED frame) is

$$\boldsymbol{p}^n_{rb,s} = d \begin{bmatrix} \cos(\Psi+\Psi_r)\cos(\alpha+\alpha_r) \\ \sin(\Psi+\Psi_r)\cos(\alpha+\alpha_r) \\ -\sin(\alpha+\alpha_r) \end{bmatrix} \tag{2.35}$$

where α_r, Ψ_r denote the base station's elevation and azimuth, respectively. A non-zero roll angle requires the use of a rotation matrix:

$$\boldsymbol{p}^n_{rb,s} = \boldsymbol{R}_{nr}(\Phi_r, \alpha_r, \Psi_r)\boldsymbol{p}^r_{rb,s} \tag{2.36}$$

Which relates to the UAV's position:

$$\boldsymbol{p}^e_{eb,s} = \boldsymbol{p}^e_{er} + \boldsymbol{R}_{en}(\phi_r, \lambda_r)\boldsymbol{p}^n_{rb,s} \tag{2.37}$$

Note that the base station (not the UAV)'s position is used when computing $\boldsymbol{R}_{en}$. Also, the assumption that the base station is level (zero roll angle) is made. This is a reasonable assumption, since there is nothing to gain from having a non-zero roll angle. However, zero roll angle may not always be obtainable. The assumption is only valid for stationary antennas, and a vehicle-mounted base station will have to take the roll angle into account. In addition, the measurement covariance matrix $\boldsymbol{R}_{pars,s}$ must be mapped to ECEF coordinates from the spherical measurements.

$$\boldsymbol{R}^n_{pars,s} = \boldsymbol{M}_s(d, \Psi, \alpha)\boldsymbol{R}_{pars,s}\boldsymbol{M}_s^T(d, \Psi, \alpha) \tag{2.38}$$

where the $\boldsymbol{M}_s$ matrix is the linearization about $\boldsymbol{\varepsilon}_s = \boldsymbol{0}$:

$$\boldsymbol{M}_s = \left.\frac{\partial \boldsymbol{p}^n_{rb,s}}{\partial \boldsymbol{\varepsilon}_s}\right|_{\boldsymbol{\varepsilon}_s=\boldsymbol{0}} \tag{2.39a}$$

$$\boldsymbol{M}_s = \left.\begin{bmatrix} \frac{\partial d\cos(\Psi)\cos(\alpha)}{\partial \varepsilon_d} & \frac{\partial d\cos(\Psi)\cos(\alpha)}{\partial \varepsilon_\Psi} & \frac{\partial d\cos(\Psi)\cos(\alpha)}{\partial \varepsilon_\alpha} \\ \frac{\partial d\sin(\Psi)\cos(\alpha)}{\partial \varepsilon_d} & \frac{\partial d\sin(\Psi)\cos(\alpha)}{\partial \varepsilon_\Psi} & \frac{\partial d\sin(\Psi)\cos(\alpha)}{\partial \varepsilon_\alpha} \\ \frac{-\partial d\sin(\alpha)}{\partial \varepsilon_d} & \frac{-\partial d\sin(\alpha)}{\partial \varepsilon_\Psi} & \frac{-\partial d\sin(\alpha)}{\partial \varepsilon_\alpha} \end{bmatrix}\right|_{\boldsymbol{\varepsilon}_s=\boldsymbol{0}} \tag{2.39b}$$

$$\boldsymbol{M}_s = \begin{bmatrix} \cos(\Psi)\cos(\alpha) & -d\sin(\Psi)\cos(\alpha) & -d\cos(\Psi)\sin(\alpha) \\ \sin(\Psi)\cos(\alpha) & d\cos(\Psi)\cos(\alpha) & -d\sin(\Psi)\sin(\alpha) \\ -\sin(\alpha) & 0 & -d\cos(\alpha) \end{bmatrix} \tag{2.39c}$$

Finally, it must be rotated to the ECEF frame.

$$\boldsymbol{R}^e_{pars,s} = \boldsymbol{R}_{en}(\phi_r, \lambda_r)\boldsymbol{M}_s(d, \Psi, \alpha)\boldsymbol{R}_{pars,s}\boldsymbol{M}_s^T(d, \Psi, \alpha)\boldsymbol{R}_{ne}(\phi_r, \lambda_r) \tag{2.40}$$

††

PARS-based positioning - Cylindrical coordinates

Due to ground reflections, the elevation angle is the least reliable of the PARS devices' three measurement variables, and the use of a barometer (using the theory from Section 2.5), GNSS receiver or similar height sensor can improve the quality of the output significantly. Similar to the other variables, the height measurement is subjected to noise:

$$h = -h_u + \varepsilon_h \tag{2.41}$$

The range measurement, in combination with the height over the base station, can be used to calculate the horizontal (in the North-East plane) distance d_g between the base station and the UAV:

$$d_g = \sqrt{d^2 - h^2} = \sqrt{(d_u + \varepsilon_d)^2 - (-h_u + \varepsilon_h)^2} \tag{2.42}$$

We start by looking at the position of the UAV relative to the radio coordinate system as with spherical coordindates:

$$\boldsymbol{p}^r_{rb,c} = \begin{bmatrix} d_g\cos(\Psi) \\ d_g\sin(\Psi) \\ h \end{bmatrix} \tag{2.43}$$

††The text above is based on (Vågsether, 2020)

Note the negative third element, since height is positive upwards. This method of calculation makes an additional assumption, namely that the base station is level (zero pitch angle). The base station's yaw angle Ψ_r is simply added to find the vector decomposed in the NED frame:

$$\boldsymbol{p}^n_{rb,c} = \begin{bmatrix} d_g \cos(\Psi + \Psi_r) \\ d_g \sin(\Psi + \Psi_r) \\ h \end{bmatrix} \tag{2.44}$$

The cylindrical measurement covariance matrix also needs to undergo a similarity transform. The cylindrical covariance mapping matrix $\boldsymbol{M}_c$ is calculated in a similar manner.

$$\boldsymbol{\varepsilon}_c = \begin{bmatrix} \varepsilon_d & \varepsilon_\Psi & \varepsilon_h \end{bmatrix}^T \tag{2.45}$$

$$\boldsymbol{M}_c = \left. \frac{\partial \boldsymbol{p}^n_{rb,c}}{\partial \boldsymbol{\varepsilon}_c} \right|_{\boldsymbol{\varepsilon}_c = \boldsymbol{0}} \tag{2.46a}$$

$$\boldsymbol{M}_c = \left. \begin{bmatrix} \frac{\partial d_g \cos(\Psi)}{\partial \varepsilon_d} & \frac{\partial d_g \cos(\Psi)}{\partial \varepsilon_\Psi} & \frac{\partial d_g \cos(\Psi)}{\partial \varepsilon_h} \\ \frac{\partial d_g \sin(\Psi)}{\partial \varepsilon_d} & \frac{\partial d_g \sin(\Psi)}{\partial \varepsilon_\Psi} & \frac{\partial d_g \sin(\Psi)}{\partial \varepsilon_h} \\ \frac{\partial h}{\partial \varepsilon_d} & \frac{\partial h}{\partial \varepsilon_\Psi} & \frac{\partial h}{\partial \varepsilon_h} \end{bmatrix} \right|_{\boldsymbol{\varepsilon}_c = \boldsymbol{0}} \tag{2.46b}$$

$$\boldsymbol{M}_c = \begin{bmatrix} \frac{d \cos(\Psi)}{d_g} & -d_g \sin(\Psi) & \frac{-h_u \cos(\Psi)}{d_g} \\ \frac{d \sin(\Psi)}{d_g} & d_g \cos(\Psi) & -\frac{h_u \sin(\Psi)}{d_g} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.46c}$$

using

$$\frac{d}{dx}\sqrt{(x + c_1)^2 + (c_2 + c_3)^2} = \frac{(x + c_1)}{\sqrt{(x + c_1)^2 + (c_2 + c_3)^2}} \tag{2.47}$$

This leaves:

$$\boldsymbol{R}^e_{pars,c} = \boldsymbol{R}_{en}(\phi_r, \lambda_r)\boldsymbol{M}_c(d, d_g, \Psi, h)\boldsymbol{R}_{pars,c}\boldsymbol{M}_c^T(d, d_g, \Psi, h)\boldsymbol{R}_{ne}(\phi_r, \lambda_r) \tag{2.48}$$

Height correction

The use of cylindrical coordinates comes with a major drawback, which manifests itself at long distances. The assumption that the Earth is flat is made, which means that the base station's 0 ° elevation plane always is the same height above the ellipsoid. This is not true due to the curvature of the Earth. The error introduced by this assumption is negligible for short distance flights, but as shown in Figure 2.4, the height offset will increase with the distance from the base station. It will propagate into the horizontal range, since the height and range is used for this calculation. However, the effect is typically much smaller on this variable,

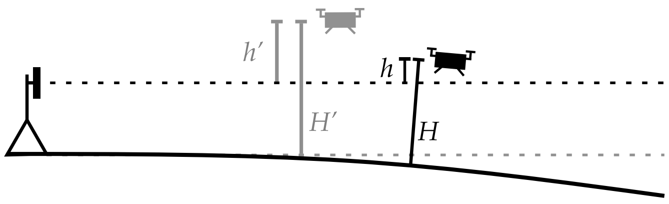

Figure 2.4: The effects of the flat Earth assumption when using cylindrical coordinates. The actual drone is drawn in black, the measured drone is drawn in grey. The base station believes that the drone is higher than it actually is. The shapes and sizes are distorted/exaggerated for clarity.

given that the height usually is significantly smaller than the horizontal range. A range-dependent height error like this is very undesirable if one is to use several PARS ground stations, since this will in practice lead to different height offsets for each ground station. It is therefore desirable to cancel the height error. It can be calculated using the base and UAV's Cartesian coordinates. Let p_i^e and p_j^e be the Cartesian position of the base and UAV, respectively. The angle γ between the two vector can be found using the following formula:

$$\gamma = \arccos \frac{(\boldsymbol{p}_{er}^e)^T \cdot \boldsymbol{p}_{eb}^e}{||\boldsymbol{p}_{er}^e|| \cdot ||\boldsymbol{p}_{eb}^e||} \tag{2.49}$$

Knowing the angle and the distance from the ECEF system's origin (the center of the Earth), the error E can be calculated. The angle γ, while describing the angular distance between the two vectors, also describes the angular distance between the base's NED frame and the drone's NED frame. The error E is defined as the vertical (in the base's NED frame) component of the displacement required to move an object from the base's 0 deg elevation plane to the same height above the geoid (or ellipsoid, depending on the height measurement and model) as the base station. This displacement must happen strictly downwards, i.e. along the NED frame's Down axis. Note that this translation is only applied to the height component, the range and azimuth measurables are unchanged. A right angled triangle is formed between the ECEF origin, the intersection between the base's height sphere and the UAV's position vector p_j^e, and the projection of p_j^e onto p_i^e. The length of the aforementioned projection is $r \cos \gamma$, with $r = ||\boldsymbol{p}_{er}^e||$. The height error E is found as a difference:

$$E = r - r \cos \gamma = r(1 - \cos \gamma) \tag{2.50}$$

which is inserted into the height equation, leading to a new height and horizontal range:

$$h' = -h_u + E + \varepsilon_h \tag{2.51}$$

$$d_g' = \sqrt{d^2 - (h')^2} = \sqrt{(d_u + \varepsilon_d)^2 - (-h_u + E + \varepsilon_h)^2} \tag{2.52}$$

A geometric illustration of (2.50) is shown in Figure 2.5. In conclusion, the PARS measurement can be corrected as such:

$$\boldsymbol{p}^n_{rb,c,\mathrm{corr}} = \begin{bmatrix} d'_g \cos(\Psi + \Psi_r) \\ d'_g \sin(\Psi + \Psi_r) \\ h' \end{bmatrix} \tag{2.53}$$

with the following relationship to ECEF coordinates:

$$\boldsymbol{p}^e_{eb,c,\mathrm{corr}} = \boldsymbol{p}^e_{er} + \boldsymbol{R}_{en}(\phi_r, \lambda_r)\boldsymbol{p}^n_{rb,c,\mathrm{corr}} \tag{2.54}$$

Just like in the uncorrected cylindrical case the (corrected) height must be incorporated in the covariance matrix:

$$\boldsymbol{M}_{c,\mathrm{corr}} = \begin{bmatrix} \frac{d\cos(\Psi)}{d_g} & -d_g \sin(\Psi) & \frac{(-h_u+E)\cos(\Psi)}{d_g} \\ \frac{d\sin(\Psi)}{d_g} & d_g \cos(\Psi) & \frac{(-h_u+E)\sin(\Psi)}{d_g} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.55}$$

$$\boldsymbol{R}^e_{pars,c,\mathrm{corr}} = \boldsymbol{R}_{en}(\phi_r, \lambda_r)\boldsymbol{M}_{c,\mathrm{corr}}(d, d_g, \Psi, h)\boldsymbol{R}_{pars,c}\boldsymbol{M}^T_{c,\mathrm{corr}}(d, d_g, \Psi, h)\boldsymbol{R}_{ne}(\phi_r, \lambda_r) \tag{2.56}$$

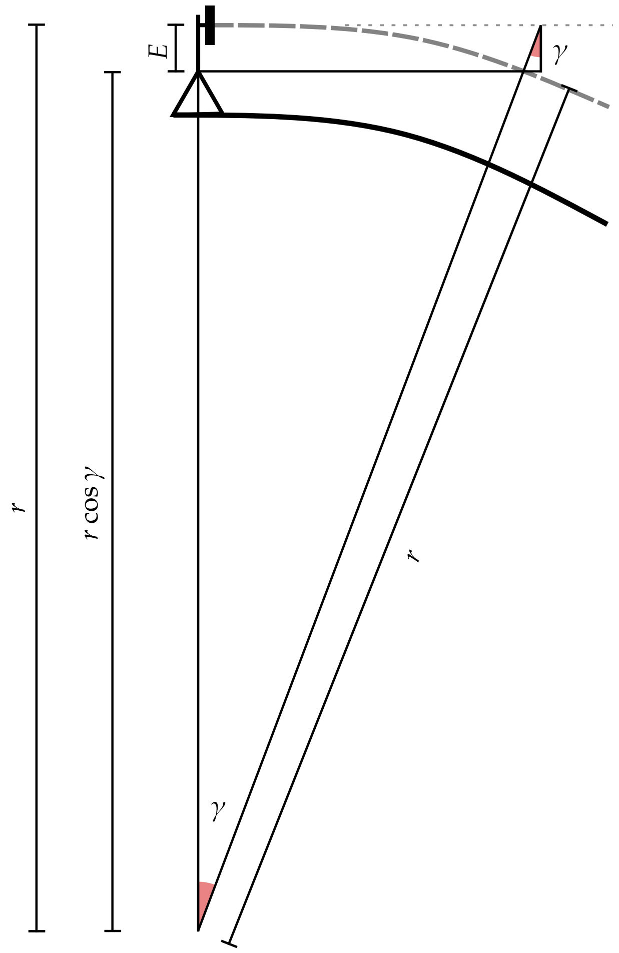

Figure 2.5: The effects of the flat Earth assumption when using cylindrical coordinates. The base station believes that the drone is higher than it actually is. The shapes and sizes are distorted/exaggerated for clarity.

3 Implementation

This chapter discusses the implementation of the navigation system. The equations that form the basis for the filter are presented and reiterated, and some of the most central relationships are derived. Familiarity with Kalman filtering is assumed, but a brief derivation of the standard Kalman filter is given in Appendix A. The calibration and outlier rejection methods are presented, before an introduction to some of the more specialized software and hardware components is given. Finally, the practical aspects of the test setup are discussed. The nominal and error state and input noise vectors are given: [†]

$$\hat{\boldsymbol{x}} = \begin{bmatrix} \hat{\boldsymbol{p}}^e_{eb} \\ \hat{\boldsymbol{v}}^e_{eb} \\ \hat{\boldsymbol{q}}^e_{eb} \\ \hat{\boldsymbol{b}}^b_{acc} \\ \hat{\boldsymbol{b}}^b_{ars} \end{bmatrix} \tag{3.1}$$

$$\delta\hat{\boldsymbol{x}} = \begin{bmatrix} \delta\hat{\boldsymbol{p}}^e_{eb} \\ \delta\hat{\boldsymbol{v}}^e_{eb} \\ \delta\hat{\boldsymbol{\theta}} \\ \delta\hat{\boldsymbol{b}}^b_{acc} \\ \delta\hat{\boldsymbol{b}}^b_{ars} \end{bmatrix} \tag{3.2}$$

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \boldsymbol{\varepsilon}^b_{acc} \\ \boldsymbol{\varepsilon}^b_{ars} \\ \boldsymbol{\varepsilon}^b_{b,acc} \\ \boldsymbol{\varepsilon}^b_{b,ars} \end{bmatrix} \tag{3.3}$$

We have the process and system noise covariance matrices $\boldsymbol{Q}_c$ and $\boldsymbol{P}$, corresponding to the error state vector.

$$\boldsymbol{Q}_c = \begin{bmatrix} \boldsymbol{\sigma}^2_{acc} & 0 & 0 & 0 \\ 0 & \boldsymbol{\sigma}^2_{ars} & 0 & 0 \\ 0 & 0 & \boldsymbol{\sigma}^2_{b,acc} & 0 \\ 0 & 0 & 0 & \boldsymbol{\sigma}^2_{b,ars} \end{bmatrix} \tag{3.4}$$

$$\boldsymbol{P} = \begin{bmatrix} P_p\boldsymbol{I}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & P_v\boldsymbol{I}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & P_\theta\boldsymbol{I}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & P_{b,acc}\boldsymbol{I}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & P_{b,ars}\boldsymbol{I}_3 \end{bmatrix} \tag{3.5}$$

[†]The following text is based on (Vågsether, 2020)

We also have the measurement noise covariance matrix $\boldsymbol{R}$, the latter of which depends on the measurement form chosen from Section 2.8. Quaternions do not have the same rules as other attitude representations, and the residual between two quaternions is calculated through multiplication, not subtraction. Note that an alternative additive scheme exists (Markley, 2004), but that is considered out of scope for this project. A Multiplicative Extended Kalman Filter (MEKF), also known as an error-state Kalman Filter, is often better suited than a standard EKF when dealing with quaternions, due to these properties. The steps of an MEKF are similar to an EKF, but the attitude gets special treatment. The PARS update step also estimates the error states, which are then injected into the nominal states. The error states propagate with the following equations:

$$\delta\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}_c \delta\hat{\boldsymbol{x}} + \boldsymbol{B}_c \boldsymbol{\varepsilon} \tag{3.6}$$

Corresponding to the following matrices (derived in Section 3.4):

$$\boldsymbol{A}_c = \begin{bmatrix} \boldsymbol{0}_3 & \boldsymbol{I}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & -\boldsymbol{R}_{eb}\boldsymbol{S}(\boldsymbol{f}_{ib}^b) & -\boldsymbol{R}_{eb} & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & -\boldsymbol{S}(\boldsymbol{\omega}_{ib}^b) & \boldsymbol{0}_3 & -\boldsymbol{I}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & -\boldsymbol{T}_{acc}^{-1} & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & -\boldsymbol{T}_{ars}^{-1} \end{bmatrix} \tag{3.7a}$$

$$\boldsymbol{B}_c = \begin{bmatrix} \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ -\boldsymbol{R}_{eb} & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & -\boldsymbol{I}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{I}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{I}_3 \end{bmatrix} \tag{3.7b}$$

3.1 INS update

The INS update step is run on the arrival of every IMU measurement, and this is where the local movement of the drone is estimated. Every IMU measurement contains two 3D vectors, the accelerometer ($\boldsymbol{f}_{ib}^m$) and Angular Rate Sensor (ARS) ($\boldsymbol{\omega}_{ib}^m$) measurements. The INS update step is run once for every IMU measurement, and thus there is no need for averaging like in Vågsether (2020). The measurements must be rotated from the $\{m\}$ frame to the body frame:

$$\boldsymbol{f}_{ib}^b = \boldsymbol{R}_{bm}\boldsymbol{f}_{ib}^m \tag{3.8a}$$

$$\boldsymbol{\omega}_{ib}^b = \boldsymbol{R}_{bm}\boldsymbol{\omega}_{ib}^m \tag{3.8b}$$

Where $\boldsymbol{R}_{bm}$ is the rotation matrix from the $\{m\}$ frame to the body frame. This matrix must be known in advance, and depends on how the IMU has been mounted on the vehicle. The biases are decomposed in the body frame, and can be subtracted once the rotation has been performed:

$$\hat{\boldsymbol{f}}^b_{ib} \leftarrow \boldsymbol{R}_{bm}\,\hat{\boldsymbol{f}}^m_{ib} - \hat{\boldsymbol{b}}^b_{acc} \tag{3.9a}$$

$$\hat{\boldsymbol{\omega}}^b_{ib} \leftarrow \boldsymbol{R}_{bm}\,\hat{\boldsymbol{\omega}}^m_{ib} - \hat{\boldsymbol{b}}^b_{ars} \tag{3.9b}$$

Next, we calculate the acceleration in the ECEF frame:

$$\hat{\boldsymbol{a}}^e_{eb} \leftarrow \boldsymbol{R}_{eb}\hat{\boldsymbol{f}}^b_{ib} + \boldsymbol{g}^e \tag{3.10}$$

where $\boldsymbol{R}_{eb}$ is calculated using (2.17). The assumption that the $\{e\}$ and $\{i\}$ frames are static in relation to each other is made, this assumption will be made several times in this report. The gravity vector $\boldsymbol{g}^e$ is calculated as a function of the ECEF position. The formula is found in Groves (2008) and reiterated below. Note that the notation has been altered to match this report.

$$\gamma^e_{eb} = -\frac{\mu}{|\boldsymbol{p}^e_{eb}|^3}\left(\boldsymbol{p}^e_{eb} + \frac{3}{2}J_2\frac{R_0^2}{|\boldsymbol{p}^e_{eb}|^2}\begin{bmatrix}(1-5(p^e_{eb,z}/|\boldsymbol{p}^e_{eb}|)^2)p^e_{eb,x}\\(1-5(p^e_{eb,z}/|\boldsymbol{p}^e_{eb}|)^2)p^e_{eb,y}\\(1-5(p^e_{eb,z}/|\boldsymbol{p}^e_{eb}|)^2)p^e_{eb,z}\end{bmatrix}\right) \tag{3.11a}$$

$$\boldsymbol{g}^e = \gamma^e_{eb} + \omega^2_{ie}\begin{bmatrix}1&0&0\\0&1&0\\0&0&0\end{bmatrix}\boldsymbol{p}^e_{eb} \tag{3.11b}$$

with R_0 = 6378137 m, μ = 3.986004418e14 m^{3}s^{-2}, J_2 = 1.082627e-3 m^{3}s^{-2} and ω_{ie} = 7.292115e-5 rad/s denoting the WGS84 equatorial radius, Earth's first and second gravitational constant and rotation rate, respectively. Next, the estimates are updated:

$$\hat{\boldsymbol{p}}^e_{eb} \leftarrow \boldsymbol{p}^e_{eb} + T_f\hat{\boldsymbol{v}}^e_{eb} + \frac{T_f^2}{2}\hat{\boldsymbol{a}}^e_{eb} \tag{3.12a}$$

$$\hat{\boldsymbol{v}}^e_{eb} \leftarrow \hat{\boldsymbol{v}}^e_{eb} + T_f\hat{\boldsymbol{a}}^e_{eb} \tag{3.12b}$$

The estimated angular rate is inserted into (2.16):

$$\hat{\boldsymbol{q}}^e_{eb} \leftarrow \hat{\boldsymbol{q}}^e_{eb} \otimes \begin{bmatrix}\cos\left(\frac{|T_f\hat{\boldsymbol{\omega}}^b_{ib}|}{2}\right)\\ \frac{\hat{\boldsymbol{\omega}}^b_{ib}}{|\hat{\boldsymbol{\omega}}^b_{ib}|}\sin\left(\frac{|T_f\hat{\boldsymbol{\omega}}^b_{ib}|}{2}\right)\end{bmatrix} \tag{3.13}$$

The bias estimates are updated using a first order approximation of the Gauss-Markov model (Section 2.4):

$$\hat{\boldsymbol{b}}^b_{acc} \leftarrow (\boldsymbol{I}_3 - T_f\boldsymbol{T}^{-1}_{acc})\hat{\boldsymbol{b}}^b_{acc} \tag{3.14a}$$

$$\hat{\boldsymbol{b}}^b_{ars} \leftarrow (\boldsymbol{I}_3 - T_f\boldsymbol{T}^{-1}_{gyro})\hat{\boldsymbol{b}}^b_{ars} \tag{3.14b}$$

Now that the nominal states have been updated, $\boldsymbol{P}$ is next:

$$\boldsymbol{P}^- \leftarrow \boldsymbol{A}_d \boldsymbol{P}^+ \boldsymbol{A}_d^T + \boldsymbol{Q}_d \tag{3.15}$$

Note that $\boldsymbol{P}$ corresponds to the error states, not the nominal states. The error states are discussed in the next section. $\boldsymbol{A}_d$ and $\boldsymbol{Q}_d$ can be obtained through discretizing their continuous counterparts. One discretization method is van Loan's method (Loan, 1978), a popular alternative for time-varying systems. The matrices $\boldsymbol{A}_c$ and $\boldsymbol{B}_c$ from (3.7) are inserted into a matrix $\boldsymbol{F}$. The process noise matrix $\boldsymbol{Q}_c$ is also required:

$$\boldsymbol{F} \leftarrow \begin{bmatrix} -\boldsymbol{A}_c & \boldsymbol{B}_c \boldsymbol{Q}_c \boldsymbol{B}_c^T \\ \boldsymbol{0} & \boldsymbol{A}_c^T \end{bmatrix} T_f \tag{3.16}$$

Next, $e^{\boldsymbol{F}}$ is approximated through the use of Taylor expansion:

$$e^{\boldsymbol{F}} \leftarrow \boldsymbol{I} + \boldsymbol{F} + \frac{\boldsymbol{F}^2}{2!} + \frac{\boldsymbol{F}^3}{3!} + \dots \tag{3.17}$$

Third order approximation is deemed precise enough for this application. The result contains the following submatrices:

$$e^{\boldsymbol{F}} = \begin{bmatrix} \cdot & \boldsymbol{A}_d^{-1} \boldsymbol{Q}_d \\ \boldsymbol{0} & \boldsymbol{A}_d^T \end{bmatrix} \tag{3.18}$$

$\boldsymbol{A}_d$ is extracted from the result, and $\boldsymbol{Q}_d$ is recovered as such:

$$\boldsymbol{Q}_d \leftarrow \boldsymbol{A}_d \boldsymbol{A}_d^{-1} \boldsymbol{Q}_d \tag{3.19}$$

3.2 PARS update

The PARS update is run each time a PARS measurement arrives. The measurement $\boldsymbol{y}_{eb}^e$, is equal to $\boldsymbol{p}_{eb}^e$ from Section 2.8, depending on which measurement type is chosen. $\boldsymbol{R}$ is equal to $\boldsymbol{R}_{pars}^e$ from the same section. We start by calculating the Kalman gain, error state vector and new covariance matrix, like in Appendix A:

$$\boldsymbol{K} = \boldsymbol{P}^- \boldsymbol{C}^T (\boldsymbol{C} \boldsymbol{P}^- \boldsymbol{C}^T + \boldsymbol{R})^{-1} \tag{3.20a}$$

$$\delta \hat{\boldsymbol{x}} = \boldsymbol{K}(\boldsymbol{y}_{eb}^e - \hat{\boldsymbol{p}}_{eb}^e) \tag{3.20b}$$

$$\boldsymbol{P}^+ = (\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{C})\boldsymbol{P}^-(\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{C})^T + \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^T \tag{3.20c}$$

Note that $\boldsymbol{R}$ will have to be calculated each time it is used, given by (2.40). Recall that $\delta \hat{\boldsymbol{x}}$ is a 15-vector consisting of the following 3-vectors:

$$\delta \hat{\boldsymbol{x}} = \begin{bmatrix} \delta \hat{\boldsymbol{p}}_{eb}^e \\ \delta \hat{\boldsymbol{v}}_{eb}^e \\ \delta \hat{\boldsymbol{\theta}} \\ \delta \hat{\boldsymbol{b}}_{acc}^b \\ \delta \hat{\boldsymbol{b}}_{ars}^b \end{bmatrix} \tag{3.21}$$

We inject the error state into nominal position and velocity estimate:

$$\hat{\boldsymbol{p}}^e_{eb} \leftarrow \hat{\boldsymbol{p}}^e_{eb} + \delta\hat{\boldsymbol{p}}^e_{eb} \tag{3.22a}$$

$$\hat{\boldsymbol{v}}^e_{eb} \leftarrow \hat{\boldsymbol{v}}^e_{eb} + \delta\hat{\boldsymbol{v}}^e_{eb} \tag{3.22b}$$

The attitude estimate is corrected, using (2.15):

$$\delta\hat{\boldsymbol{q}}^e_{eb}(\delta\hat{\boldsymbol{\theta}}) = \frac{1}{16 + \delta\hat{\boldsymbol{\theta}}^T\delta\hat{\boldsymbol{\theta}}}\begin{bmatrix}16 - \delta\hat{\boldsymbol{\theta}}^T\delta\hat{\boldsymbol{\theta}} \\ 8\delta\hat{\boldsymbol{\theta}}\end{bmatrix} \tag{3.23a}$$

$$\hat{\boldsymbol{q}}^e_{eb} \leftarrow \hat{\boldsymbol{q}}^e_{eb} \otimes \delta\hat{\boldsymbol{q}}^e_{eb}(\delta\hat{\boldsymbol{\theta}}) \tag{3.23b}$$

Note that the error quaternion $\delta\hat{\boldsymbol{q}}^e_{ib}$ will have to be replaced by its shadow set (multiplied by -1) if the attitude error exceeds 180 degrees. This condition corresponds to the norm of $\delta\hat{\boldsymbol{\theta}}$ exceeding 4. Next, we update the bias estimates:

$$\boldsymbol{b}^b_{acc} \leftarrow \boldsymbol{b}^b_{acc} + \delta\boldsymbol{b}^b_{acc} \tag{3.24a}$$

$$\boldsymbol{b}^b_{ars} \leftarrow \boldsymbol{b}^b_{ars} + \delta\boldsymbol{b}^b_{ars} \tag{3.24b}$$

Finally, the error state vector and covariance matrix are reset:

$$\delta\hat{\boldsymbol{x}} \leftarrow \boldsymbol{0} \tag{3.25a}$$

$$\boldsymbol{P}^+ \leftarrow \boldsymbol{G}(\delta\hat{\boldsymbol{q}}^e_{eb})\boldsymbol{P}^+\boldsymbol{G}^T(\delta\hat{\boldsymbol{q}}^e_{eb}) \tag{3.25b}$$

where $\boldsymbol{G}$ is a 15x15 matrix (Solà, 2017):

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{I}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{I}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \delta\hat{q}^e_{eb,w}\boldsymbol{I}_3 - \boldsymbol{S}(\delta\hat{\boldsymbol{q}}^e_{eb,v}) & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{I}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{I}_3 \end{bmatrix} \tag{3.26}$$

This concludes the PARS update step of the (error state) Multiplicative Extended Kalman Filter (MEKF). The full algorithm is shown in Figure 3.1. [††]

[††]The above text is based on (Vågsether, 2020)

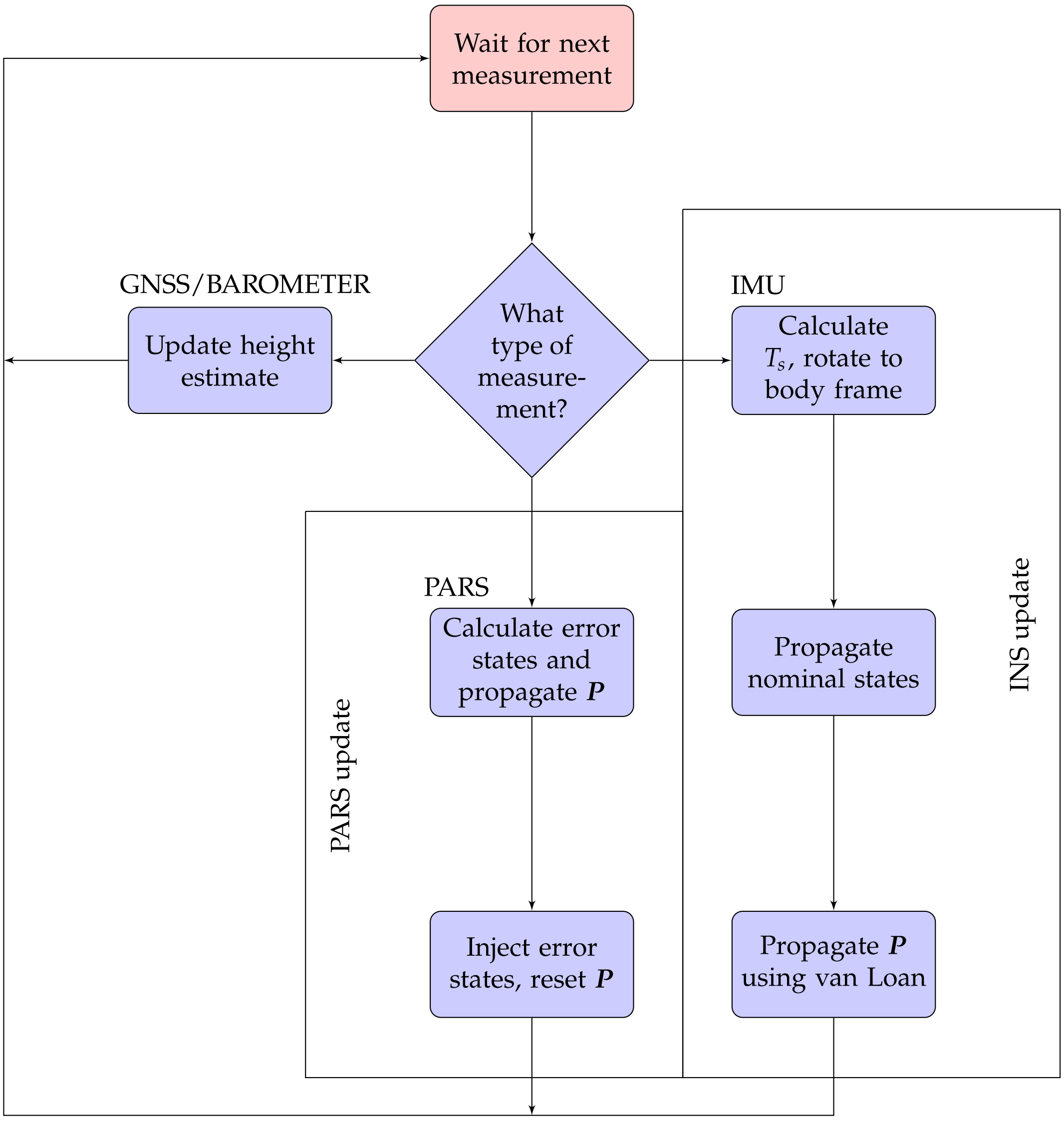
Wait for next measurement
What type of measurement?
GNSS/BAROMETER
Update height estimate
IMU
Calculate T_s, rotate to body frame
Propagate nominal states
Propagate $\boldsymbol{P}$ using van Loan
INS update
PARS
Calculate error states and propagate $\boldsymbol{P}$
Inject error states, reset $\boldsymbol{P}$
PARS update

Figure 3.1: The algorithm, summarized.

3.3 Calibration and outlier rejection

Inspired by Gryte (2020), ARS calibration is added. This is run for 30 seconds at the start of each run, and neither the INS update nor the PARS update is run until the calibration is done. The steps are reiterated:

$$\boldsymbol{R}_{ars} \leftarrow \sigma_{ars}^2 \boldsymbol{I}_3 \approx \boldsymbol{Q}_{ars}/T_f \tag{3.27a}$$

$$\boldsymbol{C}_{ars} \leftarrow \begin{bmatrix} \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{I}_3 \end{bmatrix} \tag{3.27b}$$

Next, the initial measurement update steps (i.e. (3.20)) are run with $\boldsymbol{y}$ being the actual ARS measurement and $\hat{\boldsymbol{p}}$ being $\hat{\boldsymbol{b}}_{ars}^b$, before finally (3.24b) is run. PARS Outlier rejection is also added, using the same χ^2 test as Gryte (2020). The equations are reiterated:

$$T = (\boldsymbol{p}_{eb,pars}^e - \hat{\boldsymbol{p}}_{eb}^e)^T (\boldsymbol{C}\boldsymbol{P}^- \boldsymbol{C}^T + \boldsymbol{R})^{-1} (\boldsymbol{p}_{eb,pars}^e - \hat{\boldsymbol{p}}_{eb}^e) \tag{3.28}$$

If T is greater than some threshold, corresponding to a χ^2 distribution with three degrees of freedom, the measurement is discarded. This is done to keep the navigation system from being misled by stray outliers and burst reflection measurements.

3.4 Derivations

Some of the rows in the system $\boldsymbol{A}_c$ and input noise $\boldsymbol{B}_c$ matrices are trivial, so only the rows pertaining to the velocity and angle error will be presented here. These derivations are given in Gryte (2020), and reiterated in slightly different words here.

3.4.1 Velocity

The true velocity is given as the sum of the estimate and the error:

$$\boldsymbol{v}^e_{eb} = \hat{\boldsymbol{v}}^e_{eb} + \delta\boldsymbol{v}^e_{eb} \tag{3.29}$$

$$\dot{\boldsymbol{v}}^e_{eb} = \dot{\hat{\boldsymbol{v}}}^e_{eb} + \delta\dot{\boldsymbol{v}}^e_{eb} \tag{3.30}$$

$$\dot{\boldsymbol{v}}^e_{eb} = \boldsymbol{a}^e_{eb} = \boldsymbol{f}^e_{eb} + \boldsymbol{g}^e \tag{3.31}$$

$$= \boldsymbol{R}_{eb}\boldsymbol{f}^b_{eb} + \boldsymbol{g}^e \tag{3.32}$$

Using $\boldsymbol{f}^b_{eb,imu} = \boldsymbol{f}^b_{eb} + \boldsymbol{b}^b_{acc} + \delta\boldsymbol{b}^b_{acc} + \varepsilon^b_{acc}$, the equation above can be written as such:

$$= \boldsymbol{R}_{eb}(\boldsymbol{f}^b_{eb,imu} - \boldsymbol{b}^b_{acc,ins} - \delta\boldsymbol{b}^b_{acc} - \varepsilon^b_{acc}) + \boldsymbol{g}^e \tag{3.33}$$

$$\dot{\hat{\boldsymbol{v}}}^e_{eb} = \hat{\boldsymbol{R}}_{eb}(\boldsymbol{f}^b_{eb,imu} - \boldsymbol{b}^b_{acc,ins}) + \hat{\boldsymbol{g}}^e \tag{3.34}$$

$\delta\boldsymbol{b}^b_{acc}$ and ε^b_{acc} are assumed to be zero. $\boldsymbol{R}_{eb}$ is the true attitude-dependent rotation matrix. It can be described as a product:

$$\boldsymbol{R}_{eb} = \hat{\boldsymbol{R}}_{eb}\boldsymbol{R}(\delta\boldsymbol{a}) \tag{3.35}$$

The error rotation matrix can be approximated using the skew operator:

$$\boldsymbol{R}(\delta\boldsymbol{a}) \approx \boldsymbol{I}_3 + \boldsymbol{S}(\delta\boldsymbol{a}) \tag{3.36}$$

$$\delta\dot{\boldsymbol{v}}^e_{eb} = \hat{\boldsymbol{R}}_{eb}\boldsymbol{R}(\delta\boldsymbol{a})(\boldsymbol{f}^b_{eb,imu} - \boldsymbol{b}^b_{acc,ins} - \delta\boldsymbol{b}^b_{acc} - \varepsilon^b_{acc}) + \boldsymbol{g}^e - \hat{\boldsymbol{R}}_{eb}(\boldsymbol{f}^b_{eb,imu} - \boldsymbol{b}^b_{acc,ins}) - \hat{\boldsymbol{g}}^e \tag{3.37}$$

We assume that the gravity vector is perfectly approximated ($\hat{\boldsymbol{g}}^e = \boldsymbol{g}^e$).

$$\delta\dot{\boldsymbol{v}}^e_{eb} = \hat{\boldsymbol{R}}_{eb}\boldsymbol{R}(\delta\boldsymbol{a})(\hat{\boldsymbol{f}}^b_{eb} - \delta\boldsymbol{b}^b_{acc} - \varepsilon^b_{acc}) - \hat{\boldsymbol{R}}_{eb}(\hat{\boldsymbol{f}}^b_{eb}) \tag{3.38}$$

$$\delta\dot{\boldsymbol{v}}^e_{eb} = \hat{\boldsymbol{R}}_{eb}(\hat{\boldsymbol{f}}^b_{eb} - \delta\boldsymbol{b}^b_{acc} - \varepsilon^b_{acc}) + \hat{\boldsymbol{R}}_{eb}\boldsymbol{S}(\delta\boldsymbol{a})(\hat{\boldsymbol{f}}^b_{eb} - \delta\boldsymbol{b}^b_{acc} - \varepsilon^b_{acc}) - \hat{\boldsymbol{R}}_{eb}(\hat{\boldsymbol{f}}^b_{eb}) \tag{3.39}$$

$$\delta\dot{\boldsymbol{v}}^e_{eb} = -\hat{\boldsymbol{R}}_{eb}\delta\boldsymbol{b}^b_{acc} - \hat{\boldsymbol{R}}_{eb}\varepsilon^b_{acc} + \hat{\boldsymbol{R}}_{eb}\boldsymbol{S}(\delta\boldsymbol{a})\hat{\boldsymbol{f}}^b_{eb} - \hat{\boldsymbol{R}}_{eb}\boldsymbol{S}(\delta\boldsymbol{a})\delta\boldsymbol{b}^b_{acc} - \hat{\boldsymbol{R}}_{eb}\boldsymbol{S}(\delta\boldsymbol{a})\varepsilon^b_{acc} \quad (3.40)$$

We apply (2.20):

$$\delta\dot{\boldsymbol{v}}^e_{eb} = -\hat{\boldsymbol{R}}_{eb}(\delta\boldsymbol{b}^b_{acc} + \varepsilon^b_{acc}) - \hat{\boldsymbol{R}}_{eb}\boldsymbol{S}(\hat{\boldsymbol{f}}^b_{eb})\delta\boldsymbol{a} - \hat{\boldsymbol{R}}_{eb}\boldsymbol{S}(\delta\boldsymbol{a})(\delta\boldsymbol{b}^b_{acc} + \varepsilon^b_{acc}) \quad (3.41)$$

$$\delta\dot{\boldsymbol{v}}^e_{eb} = -\hat{\boldsymbol{R}}_{eb}\boldsymbol{S}(\hat{\boldsymbol{f}}^b_{eb})\delta\boldsymbol{a} - \hat{\boldsymbol{R}}_{eb}(\delta\boldsymbol{b}^b_{acc} + \varepsilon^b_{acc}) \quad (3.42)$$

Finally, we linearize with respect to $\delta\boldsymbol{a}$ and $\delta\boldsymbol{b}^b_{acc}$ about the point where both are equal to zero:

$$\delta\dot{\boldsymbol{v}}^e_{eb} \approx \hat{\boldsymbol{R}}_{eb}(-\varepsilon^b_{acc}) + \left.\frac{\partial\delta\dot{\boldsymbol{v}}^e_{eb}}{\partial\delta\boldsymbol{a}}\right|_{\delta\boldsymbol{a}=\delta\boldsymbol{b}^b_{acc}=\boldsymbol{0}_3}\delta\boldsymbol{a} + \left.\frac{\partial\delta\dot{\boldsymbol{v}}^e_{eb}}{\partial\delta\boldsymbol{b}^b_{acc}}\right|_{\delta\boldsymbol{a}=\delta\boldsymbol{b}^b_{acc}=\boldsymbol{0}_3}\delta\boldsymbol{b}^b_{acc} \quad (3.43)$$

$$\delta\dot{\boldsymbol{v}}^e_{eb} \approx -\hat{\boldsymbol{R}}_{eb}\boldsymbol{S}(\hat{\boldsymbol{f}})\delta\boldsymbol{a} - \hat{\boldsymbol{R}}_{eb}(\delta\boldsymbol{b} + \varepsilon^b_{acc}) \quad (3.44)$$

A change of notation yields:

$$\delta\dot{\boldsymbol{v}}^e_{eb} \approx -\hat{\boldsymbol{R}}_{eb}\boldsymbol{S}(\boldsymbol{f}^b_{nb})\delta\boldsymbol{\theta} - \hat{\boldsymbol{R}}_{eb}\delta\boldsymbol{b}^b_{acc} - \boldsymbol{R}_{eb}\varepsilon^b_{acc} \quad (3.45)$$

3.4.2 Attitude

The true quaternion is the quaternion product of the estimated quaternion and the error quaternion. The error quaternion is found by premultiplying both sides by the estimated quaternion's conjugate.

$$\boldsymbol{q}^e_{eb} = \hat{\boldsymbol{q}}^e_{eb} \otimes \delta\boldsymbol{q}^e_{eb} \quad (3.46)$$

$$(\hat{\boldsymbol{q}}^e_{eb})^* \otimes \boldsymbol{q}^e_{eb} = (\hat{\boldsymbol{q}}^e_{eb})^* \otimes (\hat{\boldsymbol{q}}^e_{eb}) \otimes \delta\boldsymbol{q}^e_{eb} \quad (3.47)$$

$$\delta\boldsymbol{q}^e_{eb} = (\hat{\boldsymbol{q}}^e_{eb})^* \otimes \boldsymbol{q}^e_{eb} \quad (3.48)$$

The product rule is applied:

$$\delta\dot{\boldsymbol{q}}^e_{eb} = (\dot{\hat{\boldsymbol{q}}}^e_{eb})^* \otimes \boldsymbol{q}^e_{eb} + (\hat{\boldsymbol{q}}^e_{eb})^* \otimes \dot{\boldsymbol{q}}^e_{eb} \quad (3.49)$$

The true angular rate is a sum of the estimate, bias and error terms.

$$\boldsymbol{\omega}^b_{ib} = \hat{\boldsymbol{\omega}}^b_{ib} - \delta\boldsymbol{b}^b_{ars} - \boldsymbol{\varepsilon}^b_{ars} \quad (3.50)$$

Equation (2.13) is used to find the derivative of both the true and estimated quaternion:

$$\dot{\boldsymbol{q}}^e_{eb} = \frac{1}{2}\boldsymbol{q}^e_{eb} \otimes \begin{bmatrix} 0 \\ \hat{\boldsymbol{\omega}}^b_{ib} - \delta\boldsymbol{b}^b_{ars} - \varepsilon^b_{ars} \end{bmatrix} \quad (3.51)$$

$$\dot{\hat{\boldsymbol{q}}}^e_{eb} = \frac{1}{2}\hat{\boldsymbol{q}}^e_{eb} \otimes \begin{bmatrix} 0 \\ \hat{\boldsymbol{\omega}}^b_{ib} \end{bmatrix} \tag{3.52}$$

Equations (2.10) and (2.9) are applied to (3.52):

$$(\dot{\hat{\boldsymbol{q}}}^e_{eb})^* = \frac{1}{2}\begin{bmatrix} 0 \\ -\hat{\boldsymbol{\omega}}^b_{ib} \end{bmatrix} \otimes (\hat{\boldsymbol{q}}^e_{eb})^* \tag{3.53}$$

Equation (3.49) is rewritten using (3.51)-(3.53):

$$\delta\dot{\boldsymbol{q}}^e_{eb} = \frac{1}{2}\begin{bmatrix} 0 \\ -\hat{\boldsymbol{\omega}}^b_{ib} \end{bmatrix} \otimes (\hat{\boldsymbol{q}}^e_{eb})^* \otimes \boldsymbol{q}^e_{eb} + \frac{1}{2}(\hat{\boldsymbol{q}}^e_{eb})^* \otimes \boldsymbol{q}^e_{eb} \otimes \begin{bmatrix} 0 \\ \hat{\boldsymbol{\omega}}^b_{ib} - \delta\boldsymbol{b}^b_{ars} - \boldsymbol{\varepsilon}^b_{ars} \end{bmatrix} \tag{3.54}$$

Next, we insert (3.48):

$$\delta\dot{\boldsymbol{q}}^e_{eb} = -\frac{1}{2}\begin{bmatrix} 0 \\ \hat{\boldsymbol{\omega}}^b_{ib} \end{bmatrix} \otimes \delta\boldsymbol{q}^e_{eb} + \frac{1}{2}\delta\boldsymbol{q}^e_{eb} \otimes \begin{bmatrix} 0 \\ \hat{\boldsymbol{\omega}}^b_{ib} - \delta\boldsymbol{b}^b_{ars} - \boldsymbol{\varepsilon}^b_{ars} \end{bmatrix} \tag{3.55}$$

We define a new variable, $\delta\boldsymbol{\omega}$, such that

$$\boldsymbol{\omega}^b_{ib} = \hat{\boldsymbol{\omega}}^b_{ib} + \delta\boldsymbol{\omega} = \hat{\boldsymbol{\omega}}^b_{ib} - \delta\boldsymbol{b}^b_{ars} - \boldsymbol{\varepsilon}^b_{ars} \tag{3.56}$$

Equations (2.6) and (2.7) are used to rewrite (3.55) with $\delta\boldsymbol{\omega}$:

$$\delta\dot{\boldsymbol{q}}^e_{eb} = -\frac{1}{2}\begin{bmatrix} 0 \\ \hat{\boldsymbol{\omega}}^b_{ib} \end{bmatrix}_L \delta\boldsymbol{q}^e_{eb} + \frac{1}{2}\begin{bmatrix} 0 \\ \hat{\boldsymbol{\omega}}^b_{ib} + \delta\boldsymbol{\omega} \end{bmatrix}_R \delta\boldsymbol{q}^e_{eb} \tag{3.57}$$

$$\delta\dot{\boldsymbol{q}}^e_{eb} = \frac{1}{2}\begin{bmatrix} 0 & -\delta\boldsymbol{\omega}^T \\ \delta\boldsymbol{\omega} & -\boldsymbol{S}(\delta\boldsymbol{\omega}) - 2\boldsymbol{S}(\hat{\boldsymbol{\omega}}^b_{ib}) \end{bmatrix} \delta\boldsymbol{q}^e_{eb} \tag{3.58}$$

We split the differentiated quaternion into its scalar and vector parts:

$$\delta\dot{q}_w = -\frac{1}{2}(\delta\boldsymbol{\omega})^T\delta\boldsymbol{q}_v \tag{3.59}$$

$$\delta\dot{q}_v = \frac{1}{2}(\delta q_w\delta\boldsymbol{\omega} - \boldsymbol{S}(\delta\boldsymbol{\omega})\delta\boldsymbol{q}_v - 2\boldsymbol{S}(\hat{\boldsymbol{\omega}}^b_{ib})\delta\boldsymbol{q}_v) \tag{3.60}$$

Moving on, we know from Markley (2003) that

$$\delta\boldsymbol{a} = 4\delta\boldsymbol{a}_{mrp} = 4\frac{\delta\boldsymbol{q}_v}{1 + \delta q_w} \tag{3.61}$$

and differentiating $\delta\dot{\boldsymbol{a}}_{mrp}$ gives us

$$\delta\dot{\boldsymbol{a}}_{mrp} = \frac{\delta\dot{\boldsymbol{q}}_v}{1 + \delta q_w} - \frac{\delta\dot{q}_w\delta\boldsymbol{q}_v}{(1 + \delta q_w)^2} \tag{3.62}$$

We insert (3.59) and (3.60) for the derivatives:

$$\delta\dot{\boldsymbol{a}}_{mrp} = \frac{\frac{1}{2}(\delta q_w\delta\boldsymbol{\omega} - \boldsymbol{S}(\delta\boldsymbol{\omega})\delta\boldsymbol{q}_v - 2\boldsymbol{S}(\hat{\boldsymbol{\omega}}^b_{ib})\delta\boldsymbol{q}_v)}{1 + \delta q_w} + \frac{\frac{1}{2}((\delta\boldsymbol{\omega})^T\delta\boldsymbol{q}_v)\delta\boldsymbol{q}_v}{(1 + \delta q_w)^2} \tag{3.63}$$

$$\begin{aligned}\delta\dot{\boldsymbol{a}}_{mrp} =& \frac{1}{2}\frac{\delta q_w \delta\boldsymbol{\omega}}{1+\delta q_w} \\ & - \frac{1}{2}\boldsymbol{S}(\delta\boldsymbol{\omega})\boldsymbol{a}_{mrp} \\ & - \boldsymbol{S}(\delta\hat{\boldsymbol{\omega}}_{ib}^b)\boldsymbol{a}_{mrp} \\ & + \frac{1}{2}(\delta\boldsymbol{\omega}^T\delta\boldsymbol{a}_{mrp})\delta\boldsymbol{a}_{mrp}\end{aligned} \tag{3.64}$$

We use the following relationship

$$\begin{aligned}1-\delta\boldsymbol{a}_{mrp}^T\boldsymbol{a}_{mrp} &= \frac{1+2\delta q_w+(\delta q_w)^2-(\delta\boldsymbol{q}_v)^T\delta\boldsymbol{q}_v}{(1+\delta q_w)^2} \\ &= \frac{2\delta q_w+2(\delta q_w)^2}{(1+\delta q_w)^2} \\ &= \frac{2\delta q_w(1+\delta q_w)}{(1+\delta q_w)^2} \\ &= \frac{2\delta q_w}{1+\delta q_w}\end{aligned} \tag{3.65}$$

and apply it to the first term of (3.64), leaving us with the following expression for $\delta\dot{\boldsymbol{a}}_{mrp}$:

$$\begin{aligned}\delta\dot{\boldsymbol{a}}_{mrp} =& \frac{1}{4}(1-\delta\boldsymbol{a}_{mrp}^T\boldsymbol{a}_{mrp})\delta\boldsymbol{\omega} \\ & - \frac{1}{2}\boldsymbol{S}(\delta\boldsymbol{\omega})\boldsymbol{a}_{mrp} \\ & - \boldsymbol{S}(\delta\hat{\boldsymbol{\omega}}_{ib}^b)\boldsymbol{a}_{mrp} \\ & + \frac{1}{2}(\delta\boldsymbol{\omega}^T\delta\boldsymbol{a}_{mrp})\delta\boldsymbol{a}_{mrp}\end{aligned} \tag{3.66}$$

Finally, we linearize with respect to $\delta\boldsymbol{a}_{mrp}$ and $\delta\boldsymbol{\omega}$:

$$\begin{aligned}\delta\dot{\boldsymbol{a}}_{mrp} &\approx 0 + \left.\frac{\partial\delta\dot{\boldsymbol{a}}_{mrp}}{\partial\delta\boldsymbol{a}_{mrp}}\right|_{\delta\boldsymbol{a}_{mrp}=\delta\boldsymbol{\omega}=\boldsymbol{0}_3}\delta\boldsymbol{a}_{mrp} + \left.\frac{\partial\delta\dot{\boldsymbol{a}}_{mrp}}{\partial\delta\boldsymbol{\omega}}\right|_{\delta\boldsymbol{a}_{mrp}=\delta\boldsymbol{\omega}=\boldsymbol{0}_3}\delta\boldsymbol{\omega} \\ \delta\dot{\boldsymbol{a}}_{mrp} &\approx -\boldsymbol{S}(\delta\hat{\boldsymbol{\omega}}_{ib}^b)\delta\boldsymbol{a}_{mrp} + \frac{1}{4}\delta\boldsymbol{\omega}\end{aligned} \tag{3.67}$$

$$\begin{aligned}\delta\dot{\boldsymbol{a}} &= 4\delta\dot{\boldsymbol{a}}_{mrp} \approx -4\delta\boldsymbol{a}_{mrp}\boldsymbol{S}(\delta\hat{\boldsymbol{\omega}}_{ib}^b) + \delta\boldsymbol{\omega} \\ \delta\dot{\boldsymbol{a}} &\approx -\delta\boldsymbol{a}\boldsymbol{S}(\delta\hat{\boldsymbol{\omega}}_{ib}^b) - \delta\boldsymbol{b}_{ars}^b - \boldsymbol{\varepsilon}_{ars}^b\end{aligned} \tag{3.68}$$

In different notation, this becomes:

$$\delta\dot{\boldsymbol{\theta}} = -\boldsymbol{S}(\boldsymbol{\omega}_{nb}^b)\delta\boldsymbol{\theta} - \boldsymbol{I}_3\delta\boldsymbol{b}_{ars}^b - \boldsymbol{\varepsilon}_{ars}^b \tag{3.69}$$

3.5 STIM300

The STIM300 is a Micro-Electro-Mechanical Systems (MEMS)-based Inertial Measurement Unit (IMU), developed by Sensonor AS of Horten, Norway. It has an Angular Rate Sensor (ARS), accelerometer and inclinometer (the latter not used in this project), all with three orthogonal axes. The STIM300 uses an RS422 port for power and data transfer, and can be connected to a general purpose computer using a RS422 to split USB type A cable, where one USB connector is for power and the other is for data transfer or configuration. When powered and connected to a GNU/Linux system, the device will show up as a standard USB device file, typically `/dev/ttyUSB0`. The device file can be accessed by using a serial terminal such as `minicom` or `picocom`. One thing to note about the use of these terminals is that the device expresses line breaks using the carriage return chatacter (`'\r'`, `<CR>`). This character does not break lines on Linux platforms, but the line feed/newline character(`'\n'`, `<LF>`) does. These two characters infamously cause confusion when transferring files between Windows and Unix/Linux-based systems, since Windows requires both to express a line break. In contrast, Unix/Linux only needs a newline character to express a line break, but this means that `picocom` needs the `-imap` (which maps incoming characters) flag in order to display the response from the sensor properly. (linux.die.net, 2018)

The command used to access the device is `sudo picocom -imap crcrlf -b 1000000 /dev/ttyUSB0`. The `-b` argument accepts the baud rate in symbols per second. The STIM300 uses one device file for both configuration and data transfer, and starts in *Normal Mode* (data transfer mode), which means that the terminal will immediately be flooded with what looks like random characters. This is the measurement data, and the configuration menu can be accessed by writing `SERVICEMODE` and hitting the Enter key. This might take a few tries. If it does not work after 10 attempts, the baud rate might be incorrect. For this device, the baud rate will typically be set to one of 377400, 460800, 921600 or 1843200, but user-defined rates in the 1500 to 5184000 range are also available. The specific unit used for this project is set to 921600, hence the `-b 1000000` flag. Entering *Service Mode* will cause the "random" characters to stop flooding the terminal, and one may issue commands to the unit. Writing `i` and hitting Enter will show the current configuration parameters. The parameters for the specific unit used for the Breivika field test are shown in listing 3.1. Some fields to pay attention are the output units, sample rate (lines 5-8), datagram content (line 50) and bit rate (line 52).

The output units are set using the `u` command, followed by a space, the type of sensor, a comma and the output unit. The output unit which has different options depending on the type of sensor. For example, typing `u g,8` and hitting Enter will set the gyro/ARS to output the angular rate in the `INCREMENTAL ANGLE`

- `DELAYED` format. The `m` command sets the sample rate, and `m 1` sets it to 250 samples per second. The datagram content is configured using the `d` command, which specifies what kind of sensor data to send when the device is in *Normal Mode*. The datagram has a few fixed fields, which cannot be disabled. These are the ID, angle rate, counter, latency and CRC fields. Other enabled fields will be placed between the angle rate and the counter field. The `d` command is also used to set the line termination. As an example, the `d 3,1` command will enable the following datagram contents: ID, rate, acceleration, inclination, counter, latency, CRC. In addition, it will enable <`CR`><`LF`> line termination. Finally, `t` sets the bit rate. `t 921600` sets the bit rate to 921600 bits per second, and `t f,999325` sets it to a user-defined rate of 999325 bits per second. (Sensonor, 2021)

Listing 3.1: STIM300 configuration

```
SERIAL NUMBER = N25581808518268
PRODUCT = STIM300
PART NUMBER = 84507-440000-321 REV G
FW CONFIG = SWD12046 REV 9
GYRO OUTPUT UNIT = [/sample] - INCREMENTAL ANGLE DELAYED
ACCELEROMETER OUTPUT UNIT = [m/s/sample] - INCREMENTAL VELOCITY
INCLINOMETER OUTPUT UNIT = [m/s/sample] - INCREMENTAL VELOCITY
SAMPLE RATE [samples/s] = 250
GYRO CONFIG = XYZ
ACCELEROMETER CONFIG = XYZ
INCLINOMETER CONFIG = XYZ
GYRO RANGE:
  X-AXIS: 400/s
  Y-AXIS: 400/s
  Z-AXIS: 400/s
ACCELEROMETER RANGE:
  X-AXIS: 10g
  Y-AXIS: 10g
  Z-AXIS: 10g
INCLINOMETER RANGE:
  X-AXIS: 1.7g
  Y-AXIS: 1.7g
  Z-AXIS: 1.7g
AUX RANGE: 2.5V
GYRO LP FILTER -3dB FREQUENCY, X-AXIS [Hz] = 262
GYRO LP FILTER -3dB FREQUENCY, Y-AXIS [Hz] = 262
GYRO LP FILTER -3dB FREQUENCY, Z-AXIS [Hz] = 262
ACCELEROMETER LP FILTER -3dB FREQUENCY, X-AXIS [Hz] = 262
ACCELEROMETER LP FILTER -3dB FREQUENCY, Y-AXIS [Hz] = 262
ACCELEROMETER LP FILTER -3dB FREQUENCY, Z-AXIS [Hz] = 262
INCLINOMETER LP FILTER -3dB FREQUENCY, X-AXIS [Hz] = 262
INCLINOMETER LP FILTER -3dB FREQUENCY, Y-AXIS [Hz] = 262
INCLINOMETER LP FILTER -3dB FREQUENCY, Z-AXIS [Hz] = 262
AUX LP FILTER -3dB FREQUENCY [Hz] = 262
AUX COMP COEFF: A = 1.0000000e+00, B = 0.0000000e+00
GYRO G-COMPENSATION:
  BIAS SOURCE, X-AXIS = ACC
  BIAS G-COMP LP-FILTER, X-AXIS = ON
  SCALE SOURCE, X-AXIS = ACC
  SCALE G-COMP LP-FILTER, X-AXIS = ON
  BIAS SOURCE, Y-AXIS = ACC
  BIAS G-COMP LP-FILTER, Y-AXIS = ON
  SCALE SOURCE, Y-AXIS = ACC
  SCALE G-COMP LP-FILTER, Y-AXIS = ON
  BIAS SOURCE, Z-AXIS = ACC
  BIAS G-COMP LP-FILTER, Z-AXIS = ON
  SCALE SOURCE, Z-AXIS = ACC
  SCALE G-COMP LP-FILTER, Z-AXIS = ON
  G-COMPENSATION LP FILTER = 0.010 HZ
DATAGRAM = RATE, ACCELERATION, INCLINATION
DATAGRAM TERMINATION = <CR><LF>
BIT-RATE [bits/s] = 999325
DATA LENGTH = 8
STOP-BITS = 1
```

```
PARITY = NONE
LINE TERMINATION = ON
```

3.6 Sentiboard

The Sentiboard is a sensor interfacing PCB with a timestamping resolution of 10 ns, capable of supporting up to 8 sensors at once. It comes with three Universal Asynchronous Receiver-Transmitter (UART), two Serial Peripheral Interface (SPI), one RS422 and two RS232 ports. The Sentiboard is developed by Senti Systems AS, which is a spinoff from the Department of Engineering Cybernetics at NTNU. The Sentiboard acts as middleman between the sensors and the Odroid, delivering samples with accurate timestamps. The Sentiboard creates two serial ports on the host computer, one for configuration and another for data transfer. The configuration can be accessed using a serial port-intended terminal emulator such as `picocom` or `minicom`. The exact device name may differ depending on the host computer's OS, but the Sentiboard's configuration and data transfer lines can be found using the `/dev/ttyACM0` and `/dev/ttyACM1` device files, respectively. Configuration can be done using the following command: `minicom -con -D /dev/ttyACM0`. This will open a serial terminal, exposing the simple configuration interface. It acts like a regular terminal, accepting Enter-terminated commands. One may type `config r4` to access the configuration for the RS422 port. Other ports can be accessed similarly, with the number denoting which port to configure, i.e. `config u1` opens the configuration menu for the first (out of three) UART ports.

Listing 3.2: Typical Sentiboard sensor configuration menu.

```
Sensor 4
  [e]  enabled                          true
  [p]  powered                          true
  [s]  sync_id                          [ 0xB5 0x62 ]
  [es] end_sync_id
  [l]  length                           <H @ 4 + 8
  [b]  baudrate                         230400
  [m]  min delay between requests [us]  0
  [i]  interrupt                        rising
  [w]  warn rates                       [0  65535]
  [r]  polling                          false
  [d]  poll data:
  [?]  re-print this
  [x]  exit
```

Listing 3.2 shows the configuration parameters for this specific sensor, and these can be changed using the character on the left, i.e. `e false` switches the sensor's `enabled` state from `true` to `false`. The settings can be saved and applied by exiting the sensor configuration menu (`x`), followed by `save` and `reset`. (Senti Systems AS, 2021)

3.7 DUNE and the LSTS toolchain

The LSTS toolchain is a framework for networked vehicle systems developed by the University of Porto's Underwater Systems and Technology Laboratory (LSTS) group. The framework consists of several components that cover various aspects of autonomous vehicle operations, but the use of the framework in this project will be limited to DUNE Unified Navigation Environment, Intermodule Communication API (IMC), and to some extent, Neptus. DUNE is designed to run on each node (i.e. vehicle) and it facilitates inter-node communication using the IMC protocol. DUNE, written in C++, provides infrastructure for various services, or tasks, which can exchange IMC messages and interface with hardware sensors. The tasks employ a simple publisher/subscriber architecture, just like Robot Operating System (ROS). Documentation and build instructions on the LSTS toolchain can be found at the LSTS website or on GitHub.

The most important aspects of DUNE tasks are presented below. Each task has a main source file called Task.cpp, which is automatically generated by the command shown in Listing 3.3:

Listing 3.3: Creating a new DUNE task.

```
python <DUNE_SRC>/programs/scripts/dune-create-task.py \\
../source "Author Name" Navigation/MyNavigationSystem
```

The auto-generated file will be valid, and DUNE will build successfully even though the new task does not do anything yet. The first objective will often be to subscribe to a type of IMC message, shown in Listing 3.4. This call is placed in the task's constructor.

Listing 3.4: Subscribing to the Announce message in DUNE.

```
bind<IMC::Announce>();
```

This subscribes the task to the `Announce` message, and tasks need an overloaded `consume` member function for each message type. A task will not build unless all `bind`ed message types have a corresponding `consume` function, as shown in Listing 3.5.

Listing 3.5: Handling Announce messages in DUNE.

```
void
consume(IMC::Announce *announce)
{
    ...
}
```

Tasks can send messages too, but this does not require any declaration beforehand, such as the `bind()` call in the constructor. A task can simply construct and publish a message at any time, as shown in Listing 3.6.

Listing 3.6: Sending a message in DUNE.

```
IMC::EstimatedState estate;
estate.lat = Angles::radians(59.0);
estate.lon = Angles::radians(10.0);
estate.height = 100.0;
dispatch(estate);
```

DUNE tasks have an `onMain()` member function, shown in Listing 3.7, which is run continuously as long as the task is active. This is the main loop, where much of the task's logic can be placed. Note that a `waitForMessages()` call must be placed in the main loop, or else the task will not be able to receive messages.

Listing 3.7: DUNE tasks' onMain() function.

```
void
onMain(void)
{
    while (!stopping())
    {
        waitForMessages(1.0);
    }
}
```

Another central DUNE concept is the notion of configuration files, which is where one specifies which tasks to run. The configuration files also enable the user to pass parameters to the tasks, and one can specify the debug level on a per-task basis. A typical task entry in a configuration file is shown in Listing 3.8.

Listing 3.8: A DUNE task configuration entry.

```
[Navigation.MyNavigationSystem]
Entity Label        = MyNavigationSystem
Debug Level         = None
Enabled             = Hardware
```

DUNE can be run with the `-p` flag, which enables the user to specify which profile to use. Profiles are useful for differentiating between hardware and simulation runs, and the `Enabled` field in Listing 3.8 specifies for which profiles the task should be enabled (typically `Hardware` or `Simulation`). This means that the same configuration file in practice can be used for several types of runs. The `Enabled` field can also be set to `Always` or `Never`. Listing 3.9 shows the command needed to run DUNE in Hardware mode with a configuration file called `ntnu-x8-002.ini`.

Listing 3.9: Running DUNE.

```
./dune -c ntnu-x8-002 -p Hardware
```

Neptus is a command and control software for DUNE clients, also developed by LSTS. It is written in Java and supports IMC out-of-the-box. The concept of consoles is central when using Neptus. A console is an organisation of widgets and plugins, usually with a map and several other displays. Consoles can be tailored to the application, and NTNU UAVLAB's fixed wing console, which was used for the flight test at Breivika, includes an attitude indicator and an air speed display. It is also possible to write custom plugins for Neptus consoles, but that is not done for this project. (Pinto et al., 2013)

3.8 Setup

In this section, the practical aspects of the system are presented. This section also serves as background for the field test results discussed in Chapter 4. The base station is a Radionor Communications CRE2-189 PARS. For the drone, a Skywalker X8 fixed-wing drone is used. The PARS radio is mounted on the lid, while the fuselage contains an array of additional devices. First and foremost, a Pixhawk remote control/autopilot system running ArduPilot (open source autopilot software) is responsible for the controls. The autopilot has a dedicated GNSS receiver. The Pixhawk is connected to a router, which has three Ethernet ports in total. One of them is connected to the PARS device (4 antennas shown on top of the drone). The third and final port is connected to an Odroid XU4, which is where the INS is run. PARS samples are sent via the Ethernet interface, while a Sentiboard is mounted on top of the Odroid in order to read from the IMU via RS422. The Odroid, IMU and SentiBoard is referred to as the payload, and fastened to the fuselage using velcro tape. The payload is shown in Figure 3.2. The XU4 does not have a Wi-Fi adapter, which means that connecting to the Internet in order to download packages can be a challenge. This is solved by connecting the ground station laptop to the internet and routing the Odroid's traffic through the PARS link. This is done using the `sshuttle` tool, which creates an ad-hoc VPN. The following command shown in Listing 3.10 is run on the Odroid.

Listing 3.10: Sshuttle setup to forward traffic through the base station

```
sudo sshuttle -r hkon@10.19.0.133 0/0 -x 10.19.0.133/16
    --python=/usr/bin/python2 --dns
```

Where `hkon@10.19.0.133` is the user and IP address of the ground station laptop. The `-r` flag indicates that the following token specifies the SSH server to route traffic through, and `0/0` means that all traffic should be routed through the VPN. The 10.19 subnet needs to be excluded, hence the `-x` flag and 16-bit subnet mask (255.255.0.0). This causes the existing traffic between the Odroid and the PARS devices to remain untouched by the VPN, which is important, considering that the ground station laptop is not directly reachable by the Odroid. The `-dns` flag makes sure DNS traffic is forwarded as well. Finally, `-python=/usr/bin/python2` tells the laptop to use Python 2 for the server process, which is a workaround for an error likely caused by the laptop and Odroid's Python distributions having a large version gap. Another way of solving this problem would be to use USB tethering, i.e. share a phone's Internet connection via a USB cable. However, this requires that the Odroid has two USB ports if one intends to use the Internet connection and the Sentiboard simultaneously.

The setup procedure is described as follows. First, the base station antenna must be placed on a tripod. It should be directed towards the drone, which should already be in the launcher. In addition, the launcher should be oriented such that the drone will not fly out of sector immediately after takeoff. The position is noted using a GNSS smartphone application, and the heading of the antenna is found using a compass application. The heading should be cross-checked later on a map, since smartphone compasses do not tend to be very accurate. Next, the tripod should be adjusted to make sure the base antenna is level, this can also be done using a smartphone. The next step is to power up the base antenna and connect its Ethernet cable to the dedicated base station computer. At this point, the drone can be powered on. It will take about a minute until the Odroid is reachable via SSH from the base computer, given that the IP and MAC addresses have been entered correctly in the PARS devices' internal routing tables. Once the Odroid is up, the base station's position and orientation is entered into the DUNE configuration file. One might also have to set the initial position and orientation for the filter task at this point, but that depends on how the filter is initialized. A recompilation of DUNE is required if the initial pose is hard-coded. Next, the command in Listing 3.11 should be issued.

Listing 3.11: Command to ping the Odroid at 10 Hz.

```
sudo ping -i 0.1 10.19.60.110
```

This ensures that there is constant 10 Hz traffic between the base computer and the Odroid. At this point, a program that calculates the PARS measurements and forwards them as UDP packets is run, but this software is proprietary and will not be discussed in detail. The final preparation step is to start logging the UDP packets via the command in Listing 3.12 (make sure that the interface name is correct).

Listing 3.12: Command to start logging packets with PARS measurements.

```
sudo tcpdump -n -i eth0 -s 0 -w fireball_‘date +"%Y-%m-%d_%H-%M-%S"‘.cap
    \(not port 80\) and net 10.19.0.0/16
```

This creates a `.pcap` file with a name that includes the date and time. At this point the only remaining step is to launch DUNE, just like in Listing 3.9. Neptus can also be opened in order to watch the estimates in real time. The pilot can now start going through the pre-flight checklist, and once that is done, the drone is cleared for takeoff.

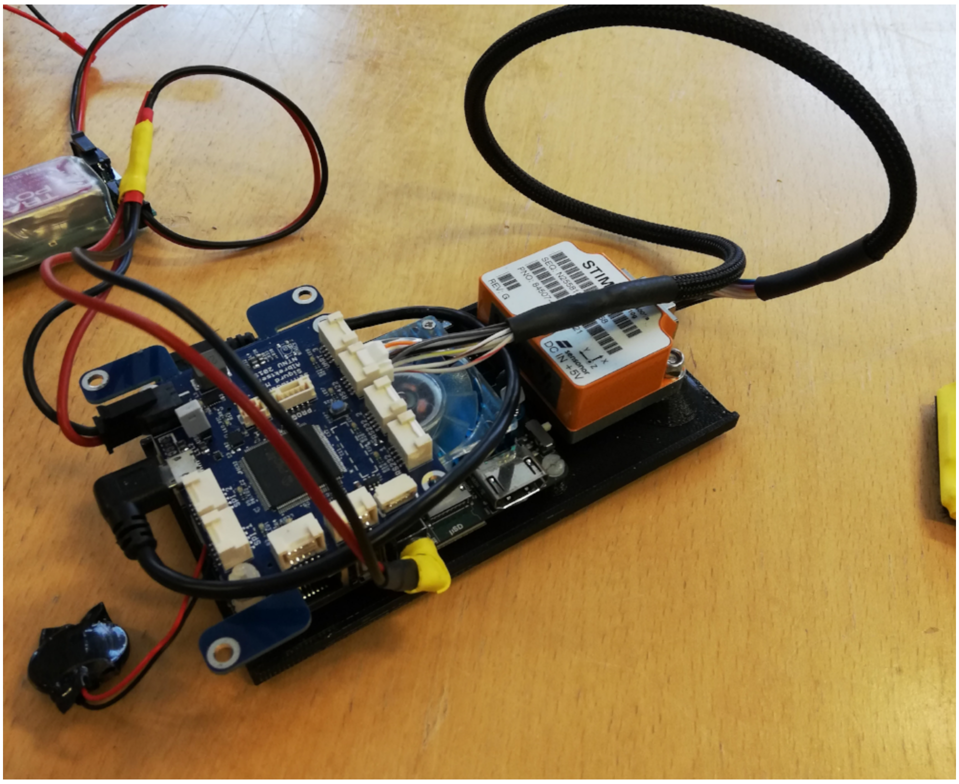

Figure 3.2: The payload, shown outside of the drone. The STIM IMU to the right, and the SentiBoard to the left. The Odroid is barely visible under the SentiBoard. The PARS device is fastened to the lid, and thus not shown in this picture.

4 Results and Discussions

The filter is evaluated using three different datasets, two captured previously and one captured in spring 2021 as part of this project. This chapter includes plots, discussions and other data on the performance of the filter. First, the tuning of the filter will be discussed, before the filter's performance on the pre-existing datasets is presented. A few different aspects of tuning will be shown along the way, with some proposed improvements. Finally, the Breivika field test will be discussed.

4.1 Tuning

As part of the filter evaluation, two datasets are tested. The filter uses the altitude from the barometer-aided autopilot as a height measurement, and calculates the UAV's position using cylindrical coordinates (Section 2.8), thus discarding the PARS elevation measurement. The filter discards measurements that are less than 1 m away. This is because the PARS sensor sometimes outputs invalid measurements with an extremely low distance value. An additional mechanism is added in the χ^2 outlier test (Section 3.3), which only accepts PARS measurements that are inliers by a probability of 0.99. The tuning values from Vågsether (2020) are used as a starting point, and a new $\boldsymbol{R}$ matrix (which must be changed since this implementation uses cylindrical coordinates) has been found through careful tuning for all the datasets. The values shown in Tables 4.1 and 4.2 were found to be the optimal values for the datasets. Some required extra tweaking, but this is discussed later.

Table 4.1: $\boldsymbol{Q}_c$, $\boldsymbol{R}^s_{pars}$ elements

Parameter	Value
σ_{acc}	2.57e-2 m/s/$\sqrt{\text{s}}$
σ_{ars}	9.59e-4 rad/$\sqrt{\text{s}}$
$\sigma_{b,acc}$	2.55e-4 m/s$^{5/2}$
$\sigma_{b,ars}$	6.29e-8 rad/s$^{3/2}$

Parameter	Value
σ_d	50 m
σ_Ψ	3 deg
σ_h	5 m

Table 4.2: Initial covariance matrix ($\boldsymbol{P}_0$) elements

Parameter	Value
P_p	100 m^2
P_v	4 m^2/s^2
P_a	0.03 rad^2
$P_{b,acc}$	1 m^2/s^4
$P_{b,ars}$	3e-6 rad^2/s^2

$$\boldsymbol{Q}_c = \begin{bmatrix} \sigma_{acc}^2 & 0 & 0 & 0 \\ 0 & \sigma_{ars}^2 & 0 & 0 \\ 0 & 0 & \sigma_{b,acc}^2 & 0 \\ 0 & 0 & 0 & \sigma_{b,ars}^2 \end{bmatrix} \tag{4.1a}$$

$$\boldsymbol{R}_{pars}^s = \begin{bmatrix} \sigma_d^2 & 0 & 0 \\ 0 & \sigma_\Psi^2 & 0 \\ 0 & 0 & \sigma_h^2 \end{bmatrix} \tag{4.1b}$$

$$\boldsymbol{P}_0 = \begin{bmatrix} P_p\boldsymbol{I}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & P_v\boldsymbol{I}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & P_a\boldsymbol{I}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & P_{b,acc}\boldsymbol{I}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & \boldsymbol{0}_3 & P_{b,ars}\boldsymbol{I}_3 \end{bmatrix} \tag{4.1c}$$

4.2 The Raudstein dataset

The first dataset is a BVLOS flight from Raudstein, where the UAV flies in a lawnmower pattern away from the takeoff location until it is about 5 km away, after which it returns in a straight line. The dataset does not include a pre-takeoff stationary period, and thus it is unsuitable for calibration. The UAV stays within the ground station's sector for entire flight, but flies at the very edge of the sector in the northern end of the first couple u-turns. The flight is performed almost exclusively over the sea, which makes for some strong reflections off the water. This increases the risk of incorrect elevation angles, and a fair amount of elevation errors are shown in Figure 4.1. The horizontal (geodetic) position is shown in Figure 4.2. The PARS aided estimates follow the PARS positions quite well, despite the fact that the drone does not have time to calibrate before takeoff (the dataset starts right before takeoff). This quick start puts extra strain on the bias estimation, and it takes about 2-300 seconds for the biases to settle properly after an initial spike. This is shown in Figures 4.3 and 4.4.

The decomposed geodetic position is shown in Figure 4.5. The estimates follow the ground truth, but the height is a few meters off, both in the beginning of the flight and in the area where the drone is the furthest away from the base. The initial height offset is likely caused by the bias estimates, which have not settled at this point. The offset subsides at $t = 1400 - 1500$, which is when the bias estimates subside too. One can see that the offset grows larger as the drone is far away from the base station, at most about 2 meters at a distance of 5.2 kilometers. This is likely caused by the curvature of the Earth and the use of cylindrical coordinates, as discussed towards the end of Section 2.8. The barometer-aided height measurement represents the geodetic height, but the use of cylindrical

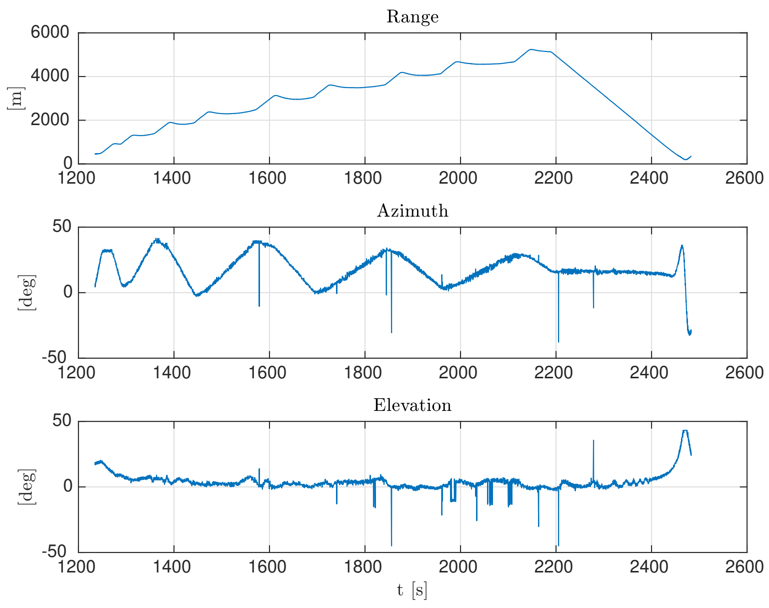

Figure 4.1: Raw PARS measurements from Raudstein. The elevation angle has a significant amount of outliers.

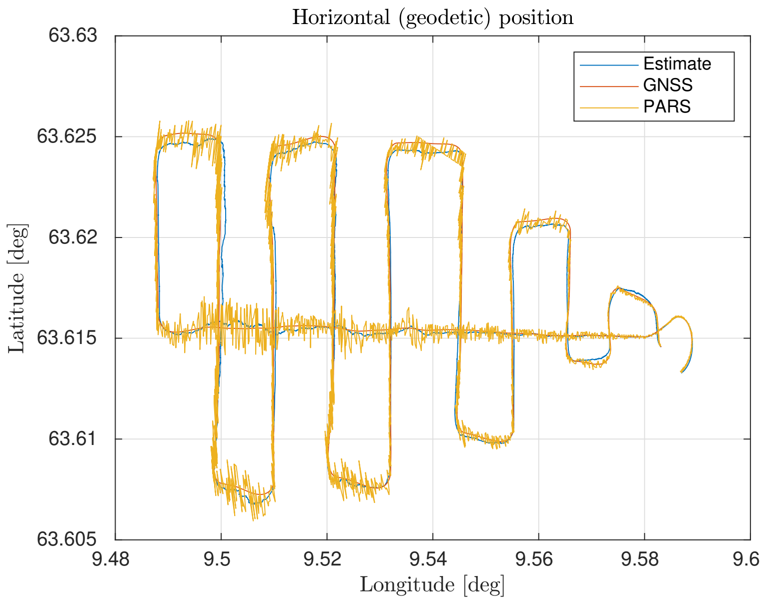

Figure 4.2: Horizontal position from Raudstein.

coordinates means that it is being interpreted as the height in the base station's reference frame. This equality is valid under the assumption of a flat Earth and zero roll and pitch angle for the base station. The fact that the estimate is higher than the ground truth supports this theory, since the curvature of the Earth will cause the UAV to be lower on the horizon the further away it gets. The height correction method proposed in Section 2.8 is tested and evaluated in Section 4.2.1. Figure 4.6 shows the position error norm with 3σ bounds, calculated using the norm of the diagonal of the $\boldsymbol{P}$ matrix, to illustrate the consistency of the filter. The error lies well within the bounds, which indicates that the filter's confidence about the estimates is realistic.

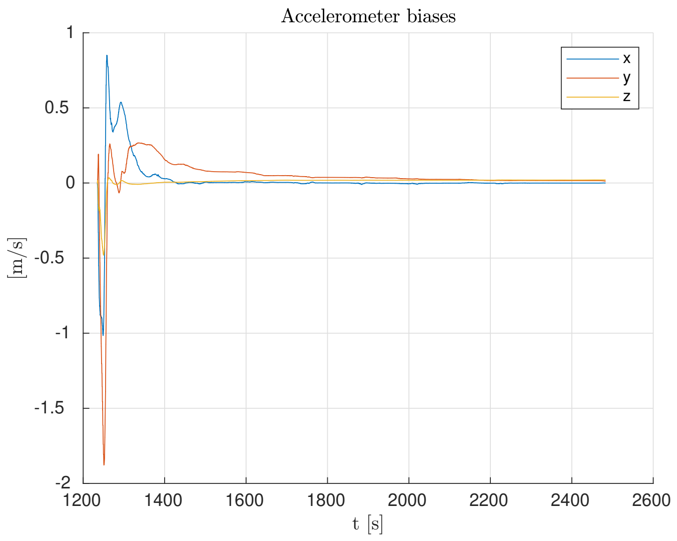

Figure 4.3: Accelerometer bias estimates.

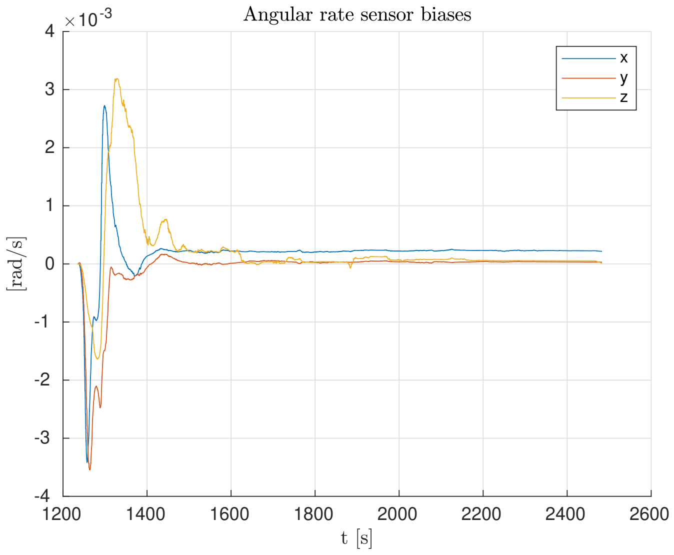

Figure 4.4: ARS bias estimates.

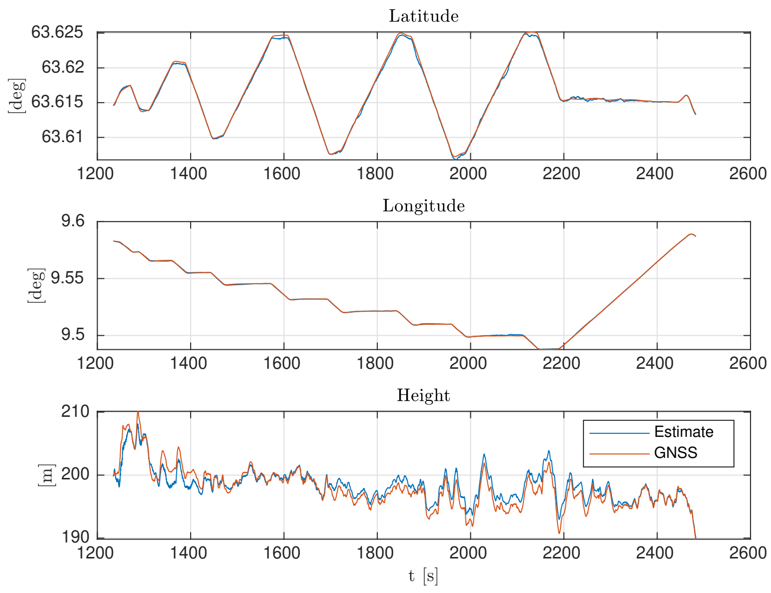

Figure 4.5: Geodetic position (estimate and ground truth) from Raudstein.

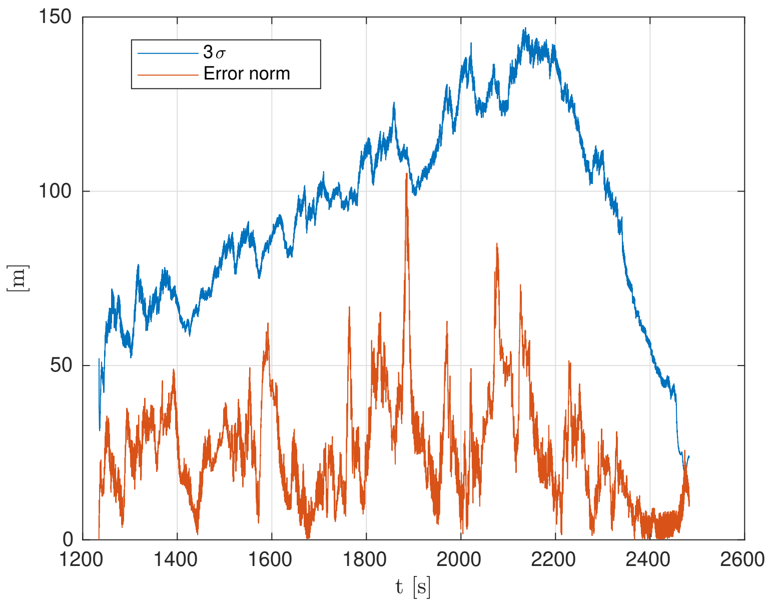

Figure 4.6: 3σ-plot from Raudstein, compared to the true error norm.

The attitude is shown in Figures 4.7. As expected, the u-turns are evident in the roll angle. The yaw angle switches between 0 and 180 degrees, approximately, followed by several minutes of flying due east. The attitude error is shown in Figure 4.8, and after an initial spike in roll and pitch, the angle errors are fairly low. The initial spike is most likely caused by incorrect initial values for the drone's roll and pitch angles, since these are just set to zero. This affects the yaw error, which also has a spike at the start of the flight. The pitch angle is off by a few degrees, as a stationary/slowly fading error. This could be a consequence of the IMU mounting, which is fastened to the fuselage using velcro tape.

Finally, Tables 4.3 and 4.4 show statistics (Mean Error, Mean Average Error and Root Mean Square Error) on the error between the estimates and the ground truth. The errors are comparable, but slightly worse than the errors found in Gryte et al. (2020) for the same dataset. When it comes to the position errors, only the Norm column can be compared directly, since the position is decomposed in different frames. For the attitude, the errors are fairly small and lie in the same range at 1 to 5 degrees. Comparing it to the results from Vågsether (2020), it is found that this thesis' implementation performs significantly better, with the position error norms being about half the size for all error types. We also see that most of the error in Vågsether (2020) is caused by the z axis, which indicates that the outlier rejection has played a big part in rejecting ground reflections. The attitude error is larger than Vågsether (2020) for the pitch and roll axes, this can likely be reduced greatly by rotating $\boldsymbol{R}_{bm}$ such that it matches the autopilot IMU better.

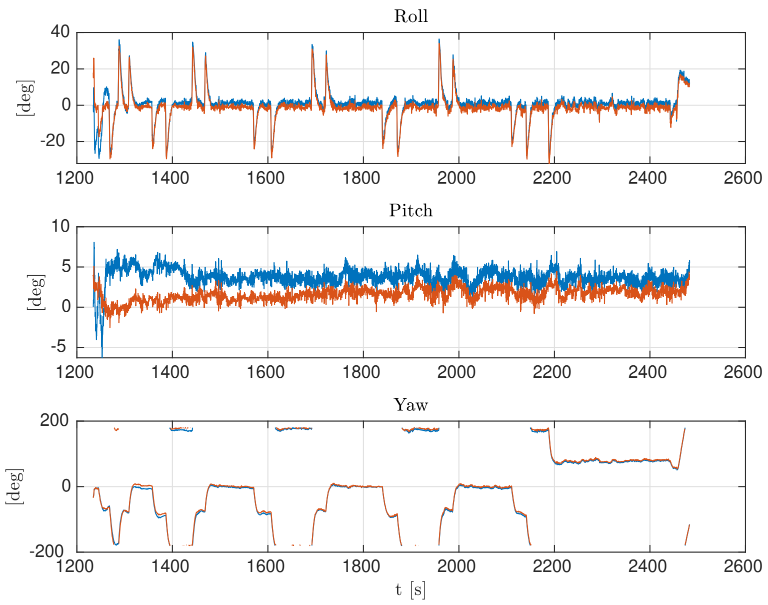

Figure 4.7: Estimated (blue) and true (red) attitude from the Raudstein dataset.

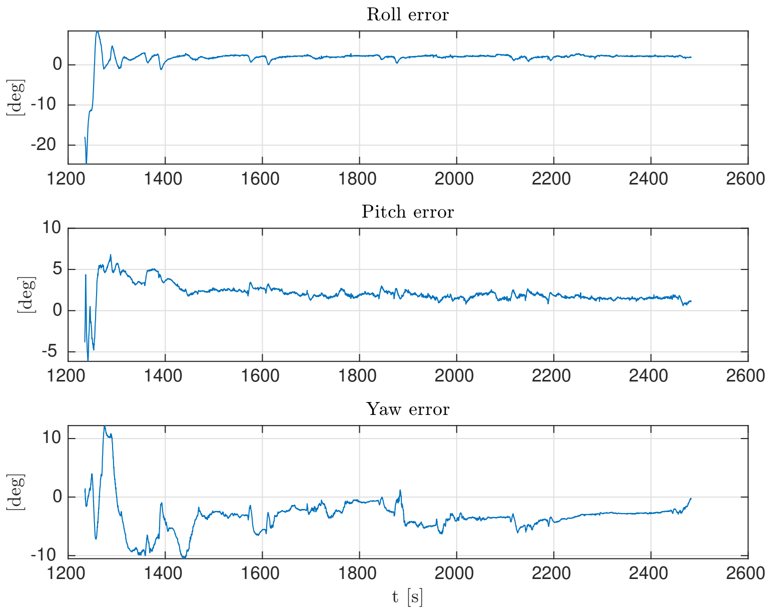

Figure 4.8: Roll, pitch and yaw error from the Raudstein dataset.

Table 4.3: PARS-aided position error statistics for the Raudstein dataset.

Metric	x **[m]**	y **[m]**	z **[m]**	**Norm [m]**
ME	12.89	6.56	-6.33	15.79
MAE	17.34	12.12	8.39	22.76
RMSE	22.56	15.96	11.05	29.76

Table 4.4: PARS-aided attitude error statistics for the Raudstein dataset.

Metric	**Roll [deg]**	**Pitch [deg]**	**Yaw [deg]**	**Norm [deg]**
ME	1.77	2.15	-3.24	4.28
MAE	2.20	2.25	3.70	4.86
RMSE	2.79	2.47	4.31	5.69

4.2.1 Height error compensation

As mentioned in Section 2.8, the use of cylindrical coordinates introduces a height error that becomes larger the further away the drone is. The compensation method introduced in Section 2.8 is implemented, and the results are shown in Figure 4.9. The compensation method appears to cancel most of the error effectively, as the plot indicates. The remaining error could be an offset between the barometer and the ground truth. The height error at the start of the flight fluctuates, but this is likely due to the volatile biases. Both error curves fluctuate, but as the biases settle and the uncorrected curve starts to rise, the corrected curve stays at around 0 m. The uncorrected curve is seen to grow steadily until reaching 2 meters at the highest until it starts declining again as the drone moves back towards the base station from $t = 2200$. Meanwhile, the corrected height error does not seem to be affected by the distance from the base station to the UAV, which indicates that the correction method works as intended.

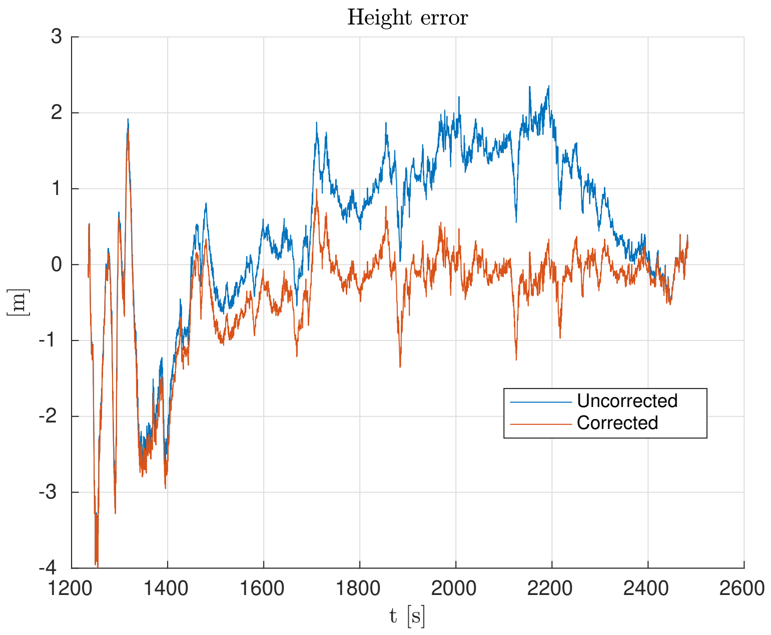

Figure 4.9: Height error plot from Raudstein.

4.3 The Udduvoll dataset

Tha dataset from Udduvoll is quite different from the Raudstein flight. It starts with a several minute stationary period, before the drone is put in the launcher and takeoff is performed. This means that ARS calibration is appropriate and will be run for 30 seconds at the very beginning. The data is captured from a flight that is closer to the base station than the Raudstein dataset, but it is more challenging to work with, mostly due to missing PARS data. The UAV flies out of the ground station's sector several times, and the PARS data even falls out completely (intentionally) over a 3 minute interval towards the end of the capture. The PARS data is disabled in order to demonstrate the system's robustness to outage, and illustrate that the filter works well on IMU input alone. This creates tuning challenge, and the fact that there are so many tuneable parameters in the system make it difficult to achieve satisfactory performance. Figure 4.10 shows the horizontal (geodetic) positions. The PARS positions are marked in red to show the out-of-sector parts of the flight. The estimate drift towards the Southeast is caused by the period of time where the PARS is disabled. This is not very impressive, considering that the PARS data is only gone for 3 minutes. However, the UAV is turning constantly over this interval, and this type of motion is likely more difficult to estimate than a straight-line trajectory. The drift may to be caused by an ARS bias error; the filter believes that the drone turns more than it actually does. In addition, the 3 minutes of dead reckoning leads to the filter estimating a slowly declining height, as shown in Figure 4.11.

Compared to Gryte's results (Gryte et al., 2020) for the same dataset, this southeastern drift is a major regression. The distance of the drift is very large, larger than the true distance from the base to the drone at any other point in the flight. The large gap between the performance of this implementation and Gryte's might be caused by the choice of reference frames. Gryte uses the NED frame, which means that the barometer's height measurement can be used to correct the isolated z (down) component of the position. Thus, the estimates are being corrected over the 3 minute period even though the PARS data is not present. For this project's ECEF-based implementation, the height is not an isolated component of the position, and a height-only correction cannot be done without touching the rest of the position vector. As a result, the height is only used to correct the PARS measurements, and the 3 minute PARS-less period is estimated using pure dead reckoning only.

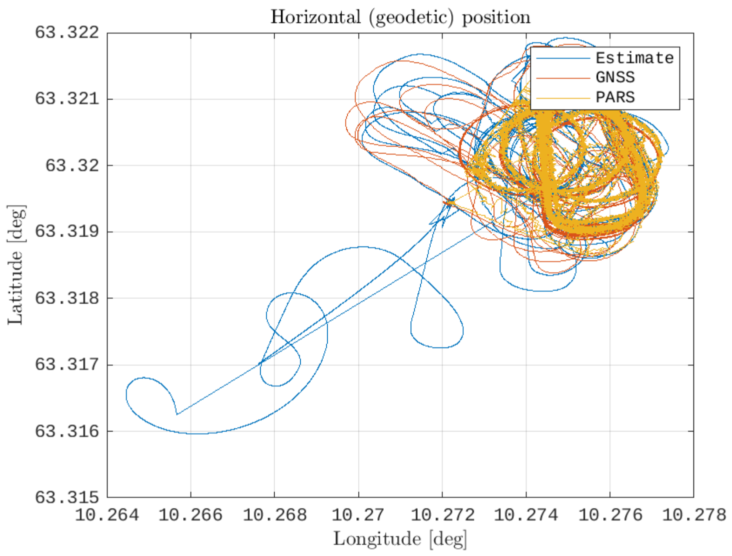

Figure 4.10: Horizontal geodetic position (estimate and PARS) from Udduvoll.

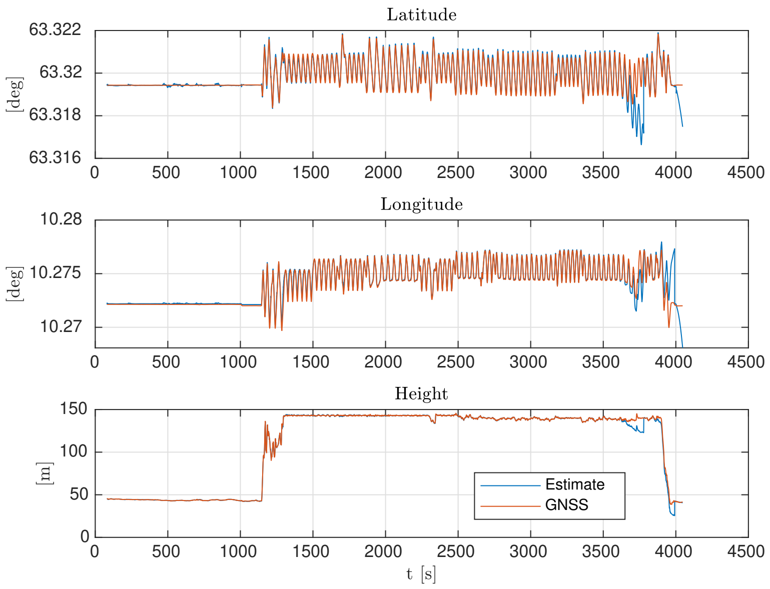

Figure 4.11: Geodetic position (estimate and ground truth) from Udduvoll. The units are in degrees and meters.

Although ARS calibration is enabled for this dataset, the UAV spends a lot of time outside the base's sector, especially shortly after takeoff. This can be seen from the bias estimates (Figures 4.12 and 4.12), where most components don't seem to start settling until after $t = 1500$. This is the point where the UAV starts flying in sector continuously, after flying in and out of sector (especially at the northeastern end of the area) repeatedly. The biases also exhibit a few jumps and spikes before takeoff (takeoff occurs at approximately $t = 1200$), even though the position seems to be very still at these points. This behaviour is caused by the drone being moved around and placed in the launcher.

The position error margins (Figure 4.14) are not as large for the Udduvoll dataset as for the Raudstein flight. The 3σ error bounds are also fairly flat during the course of the flight, which can be attributed to the small variations in distance between the base and the drone. The UAV stays in the same area. In contrast, the large distance for the Raudstein flight manifests as an error bound increase towards the middle of Figure 4.6. The error stays within the bounds for most of the flight, but some spikes are outside the bounds, especially shortly after takeoff. This is expected, since the drone spends so much of this time outside the base's sector. The 3 minute period without PARS is very easy to identify since the 3σ error bound grows to 4.5 km (this has been cropped from the figure in order to

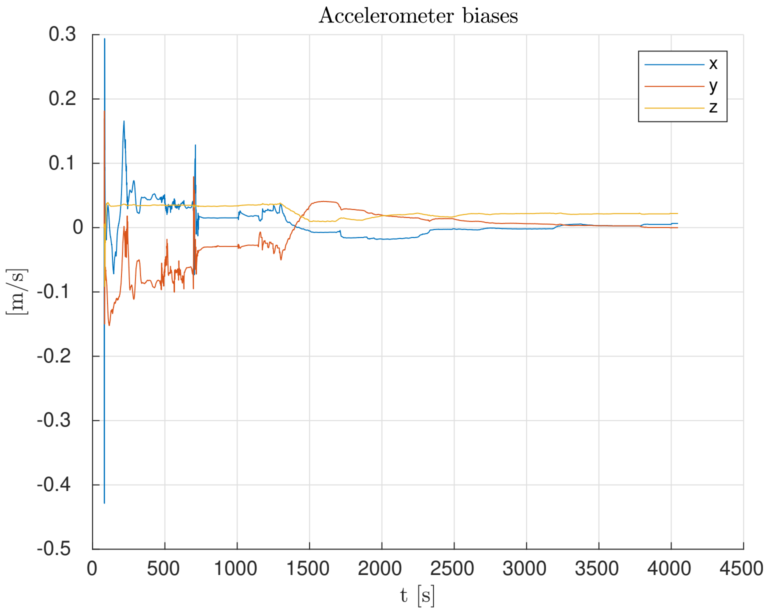

Figure 4.12: Accelerometer bias estimates.

prioritize the rest of the flight). The position estimate error grows to 732 m before PARS is reenabled. The drone also lands out of sector, which is shown as another large spike at the very end of the flight.

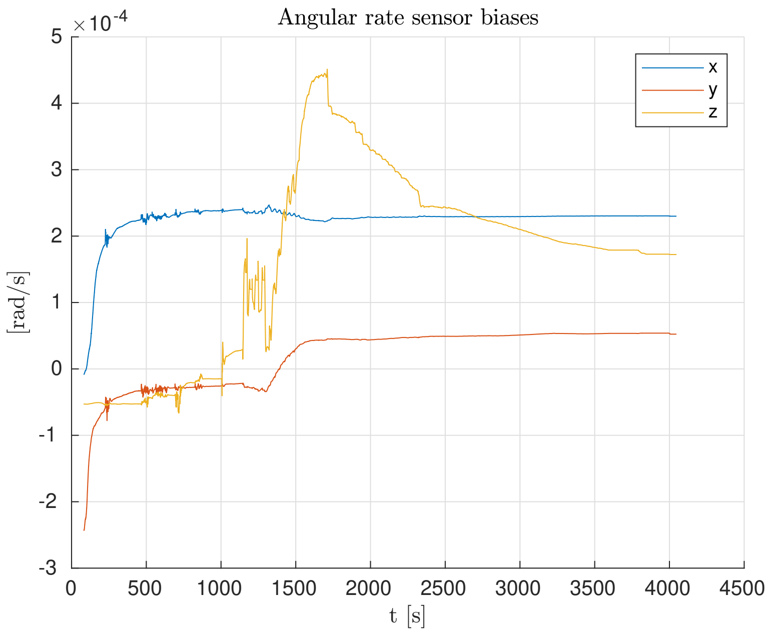

Figure 4.13: ARS bias estimates.

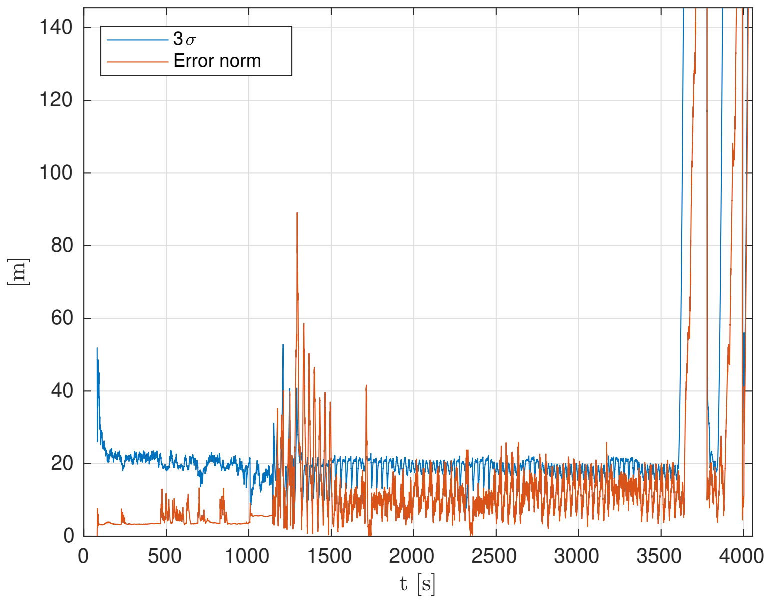

Figure 4.14: 3σ-plot from Udduvoll.

The attitude estimates (Figures 4.15 and 4.16) for the Udduvoll dataset start out much calmer than the Raudstein estimates do, due to the initial stationary period, where the drone is placed flat on the ground. This allows for the use of ARS calibration, and it also means that the initial roll and pitch angle estimates (both zero) are correct. The figure clearly shows that the drone is moved around before takeoff (at $t = 600$) and even shows when the drone is being put in the launcher (at $t = 1000$). After takeoff, the estimates' accuracy looks very similar to that of the Raudstein dataset, with a familiar small positive error in the pitch angle. The estimates seem unaffected by the PARS outage, which is interesting when recalling the large position errors.

The horizontal position error during the PARS cutoff phase can be reduced by skipping the dead reckoning bias updates (equation (3.14)). This will keep the biases from changing unless corrected by PARS. The difference is shown in Figures 4.17 and 4.18. The estimated position is much closer to the true trajectory, and the shapes even overlap. However, this worsens the landing trajectory (coming in from the Northeast) significantly. The landing happens shortly after PARS is reenabled, and thus the filter does not seem to have time to recover before the drone is brought down. The landing trajectory in Figure 4.17 is followed well by the filter, which indicates that a hybrid approach (biases are updated only if PARS data has been received less than a second ago) might be worth investigating.

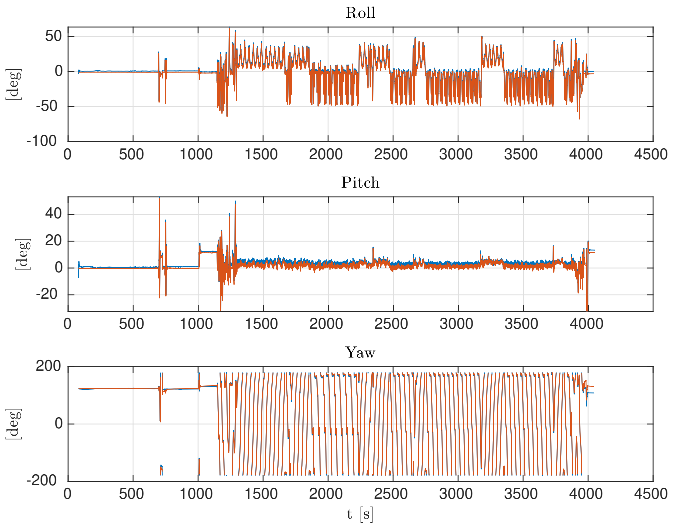

Figure 4.15: Estimated (blue) and true (red) attitude from the Udduvoll dataset.

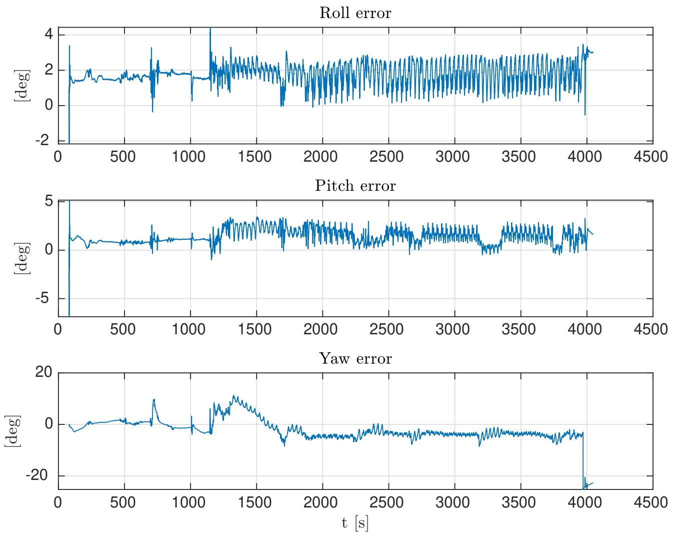

Figure 4.16: Attitude error from the Udduvoll dataset.

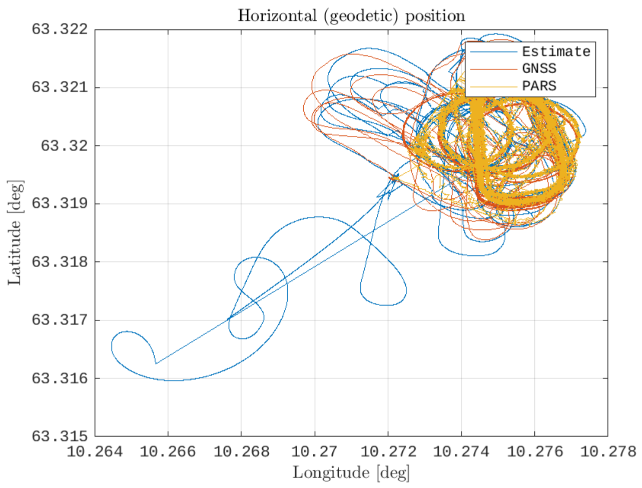

Figure 4.17: Udduvoll position estimates without bias updates.

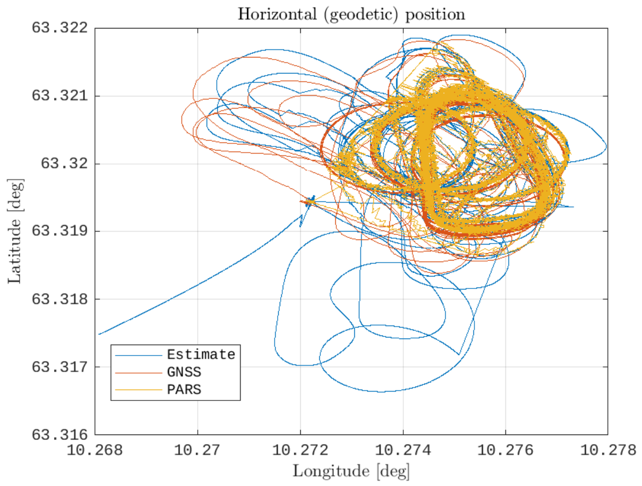

Figure 4.18: Udduvoll position estimates without bias updates.

4.4 The Breivika field test

As part of this project, a trip was taken to Breivika, Orkland in order to test the navigation system online. The UAV was inside the antenna's sector most of the time, with the exception of a few turns towards the end of the flight. Field tests and demonstrations are often associated with technical problems, and this field test was no different. During the preparation and actual testing, several issues arose:

- In order to get IMU data into DUNE, there are three points of configuration that need to be set correctly: The STIM configuration, the SentiBoard configuration, and the SentiBoard DUNE task. Unbeknownst to me, the STIM had been set to a configuration that matched neither the online wiki nor the SentiBoard task. This caused a lot of segmentation faults and errors in the task, and I made the false assumption that the wiki was correct, which caused me to spend several days solving this problem with the wrong approach. I finally fixed it by setting the STIM configuration to match the wiki (except for the baud rate) and SentiBoard task, which solved the problem entirely.

- On arrival at Breivika, a problem was identified with the conversion between radians and degrees, where different parts of code were using varying amounts of decimal places to approximate π. This caused the PARS positions to be several kilometers off when displayed in Neptus, and while it was very simple to fix, it took about an hour to identify.

- Ardupilot's dedicated GNSS receiver and internal position estimates were acting up. The estimates were drifting around erratically, delaying the takeoff further. This problem was solved by righting a bent pin on the Pixhawk autopilot device and moving the launcher and car further away from each other.

- The next problem to occur was sporadic segmentation faults in the SentiBoard DUNE task. This had not happened before this point, and I have not been able to reproduce it since. This caused the entire DUNE process to crash, which led to the discovery of the next problem. The problem was worked around by disabling the `Should log` flag in the SentiBoard task, at which point the segmentation faults subsided.

- It was discovered that the crashing DUNE process also brought the MAVLink router process on the router down with it, which meant that a restart of the process was required to reconnect the Ardupilot on-board software with the ground station. This was baffling, considering that these two processes are not running on the same computer, they are merely connected through a TCP port where DUNE receives updates from Ardupilot. This problem may have been present on shorter tests on the days leading up to the field test at Breivika, but I was never in contact with the Ardupilot software directly, and thus I failed to recognize this problem, much due to the fact that the warning message was buried in messages from other parts of the system. The MAVLink router process died both when DUNE crashed and when DUNE was closed normally (`Ctrl-C`). This problem was solved by moving the payload to the other drone, which required some time to set up, due to the fact that the routers had equal MAC addresses but different IP addresses. This change was difficult to apply in the PARS devices' routing tables.

- Finally, the aforementioned issues were resolved (or at least worked around) and a flight was performed. However, the MEKF estimates only lasted a few seconds after takeoff until they seemed to lose lock on the actual UAV and started to wander around aimlessly. This was very disappointing to see, but the entire flight was logged, and upon later inspection, it was found that the IMC messages containing the PARS measurements were invalid. The azimuth and elevation angles differed greatly from the angles in the raw UDP logs, and this gave the filter very difficult data to work with, and even though the filter's output seemed reasonable while the UAV was in the launcher.The logged angles in Figure 4.19 are clearly unusable when comparing them to the Figure 4.20, which is generated from the raw UDP logs. The plots from this dataset are therefore generated after the field test, using DUNE's replay functionality with the valid PARS measurements saved to a CSV file. The DUNE task reads the CSV file into memory and uses this PARS data instead of the logged IMC messages. The timestamps of the valid PARS data are lined up with the invalid logs using visual inspections and a simple offset. This is not ideal, but it is good enough for the data to be useable.

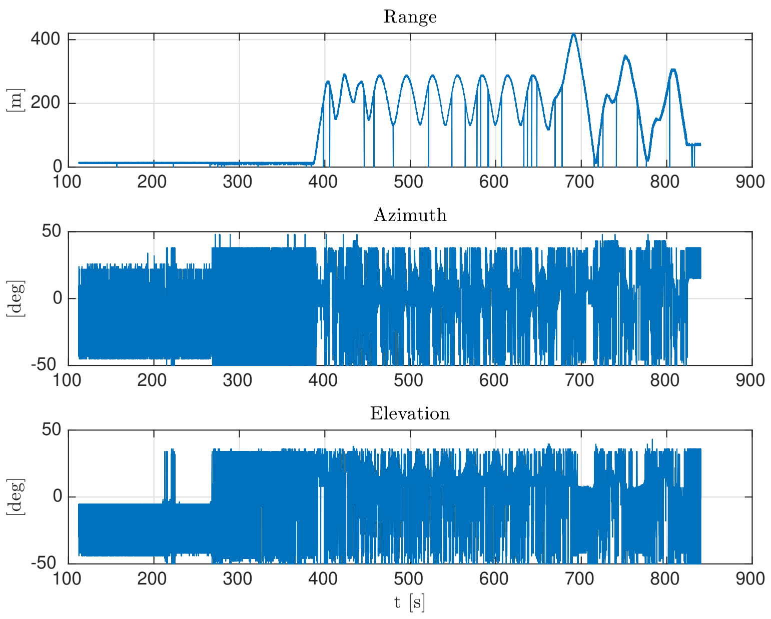

Figure 4.19: Invalid PARS measurements from Breivika (logged IMC messages).

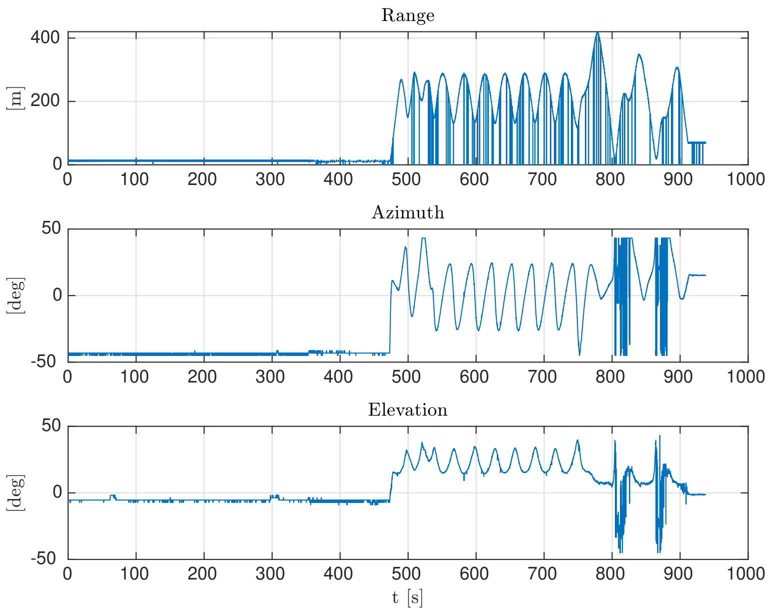

Figure 4.20: Valid PARS measurements from Breivika (logged UDP messages).

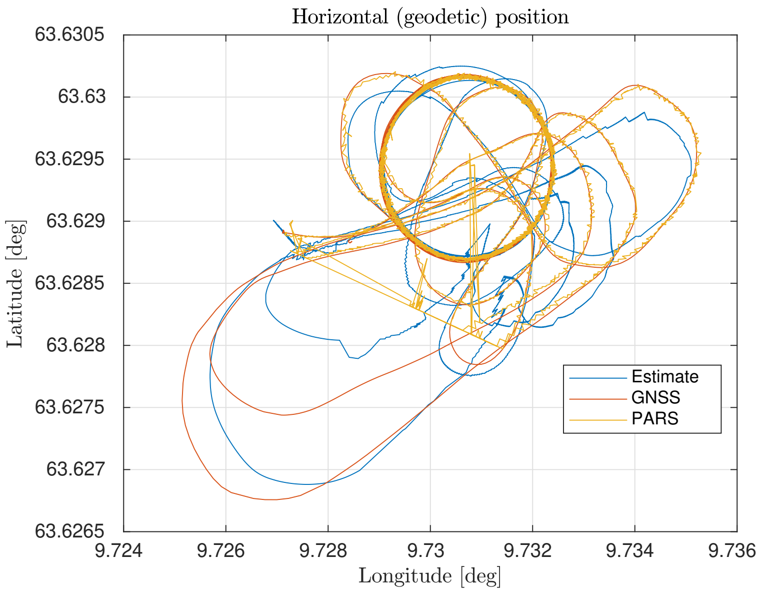

Figure 4.21: Horizontal position from Breivika.

The horizontal position is shown in 4.21. As we can see, the estimates do not move perfectly with the PARS data, nor with the true positions, but this is also a much closer flight than for example the Raudstein dataset. The estimates struggle particularly with the Southeastern portion of the trajectory leading up to the landing, where the drone flies out of sector twice. The full geodetic position is shown in Figure 4.22, where the estimates follow the ground truth well for the most part. One exception is at $t = 725$, where the UAV flies out of sector, which throws the filter slightly off for 15 seconds. The biases are shown in Figures 4.23 and 4.24, and the spikes at the start are from the few seconds when the drone was moved around before being placed in the launcher. Takeoff happens at $t = 400$, and this manifests as a step in the ARS's yaw angle (z component) bias. This is likely caused by an incorrect yaw angle on takeoff, which will be shown later. The effects of the step subsides after half a minute. We can also see that the moments where the drone is out of sector upset the biases further, after a period of relative calmness.

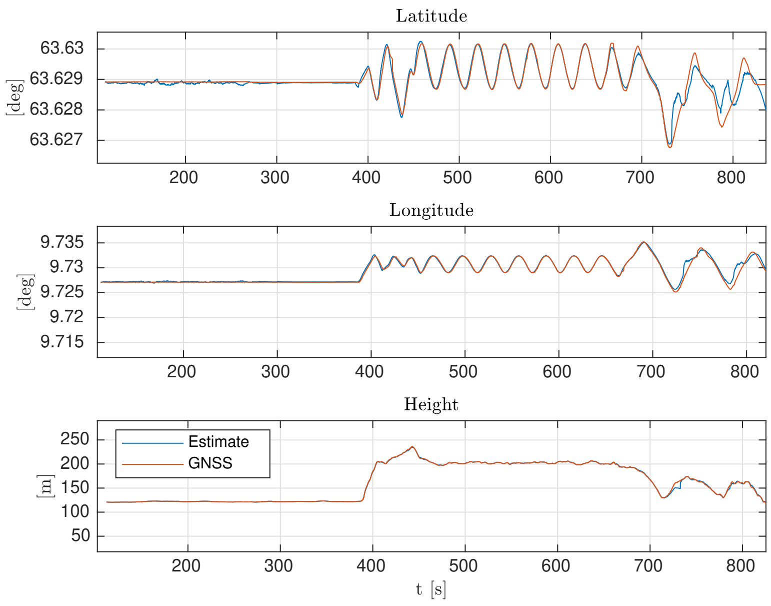

Figure 4.22: Geodetic position from Breivika.

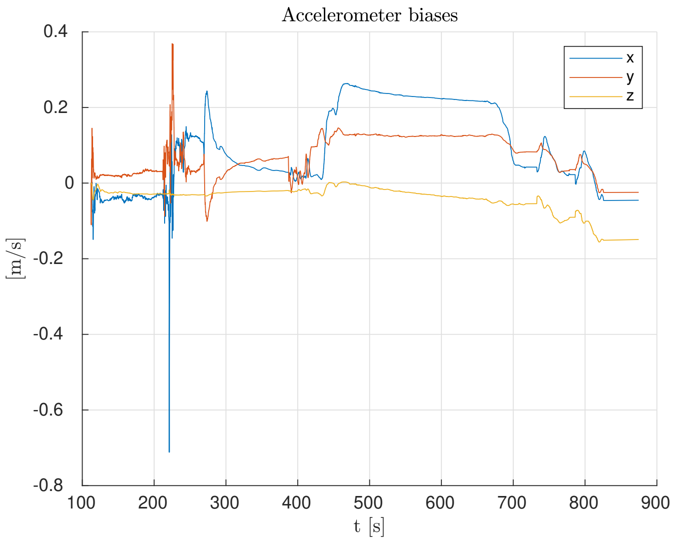

Figure 4.23: Accelerometer bias estimates from Breivika.

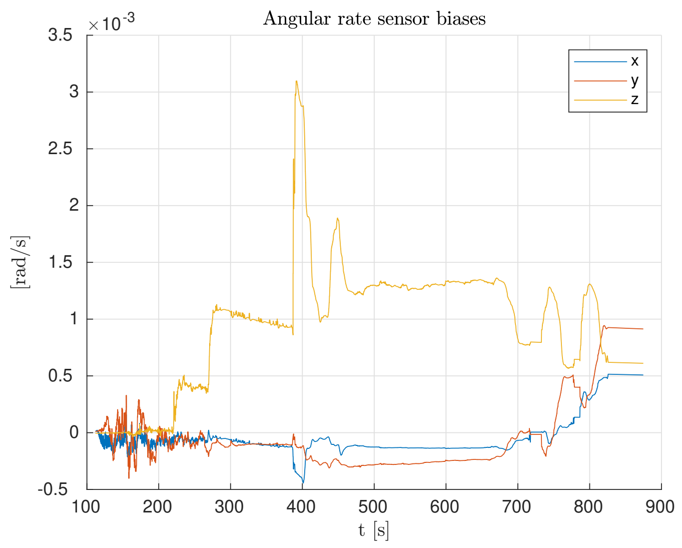

Figure 4.24: ARS bias estimates from Breivika.

The 3σ position error bound is shown in Figure 4.25. The error lies well inside the bound initially, but the margin is a little slim after takeoff and the error frequently exceeds the bounds. The error when the drone flies out of sector is also marked as a sharp peak, and this is expected. The margin between the 3σ error bounds and the error norm can be increased by increasing the $\boldsymbol{Q}_c$ matrix, and this is done in Figure 4.26. A factor of 50 gives a more comfortable margin, but it also decreases the filter's confidence in the IMU measurements, which in this case leads to higher errors when the drone is out of sector. Figures 4.27 and 4.28 show the estimated attitude and error, with the ground truth drawn in red. These plots seem fairly similar to those of the previous datasets, with the familiar small pitch offset. One major difference stands out, namely the slowly moving pre-flight yaw error. The estimates start out well, but after the drone has been moved around, an error starts to accumulate. Looking at Figure 4.27, it is clear that it is the ground truth that is moving, not the estimate. This does not occur in the other datasets, at least not to such a noticeable extent. It does not seem to be the actual behaviour of the drone, since the drone is stationary in the launcher and does not have a reason to move this slowly. This is likely drift in the autopilot's internal estimation algorithm, and it cannot be ruled out that this is related to the problems that we had earlier with the autopilot's internal position estimates.

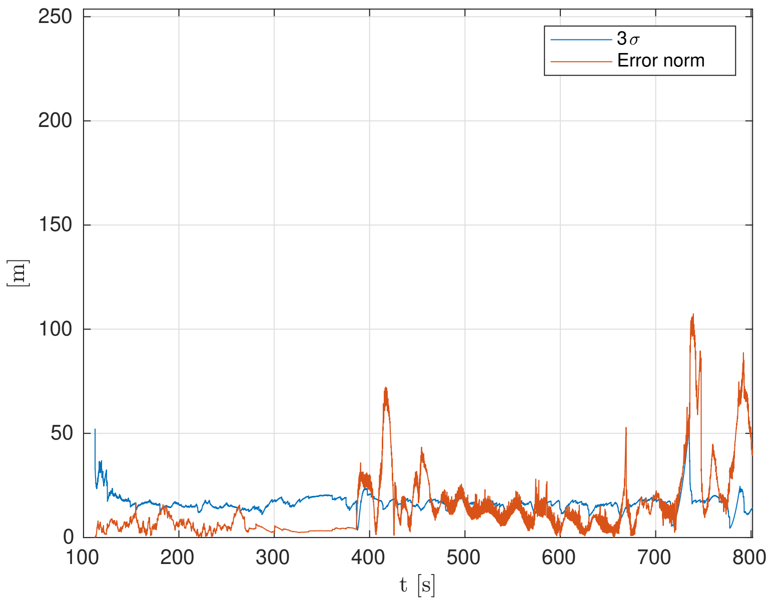

Figure 4.25: 3σ-plot from Breivika with the original $\boldsymbol{Q}_c$ matrix.

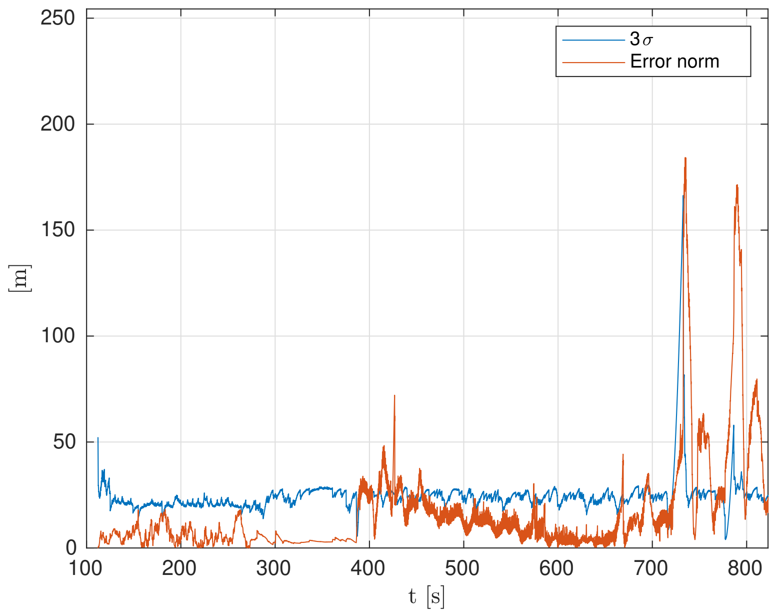

Figure 4.26: 3σ-plot from Breivika. The $\boldsymbol{Q}_c$ matrix has been multiplied by 50, yielding a larger error bound.

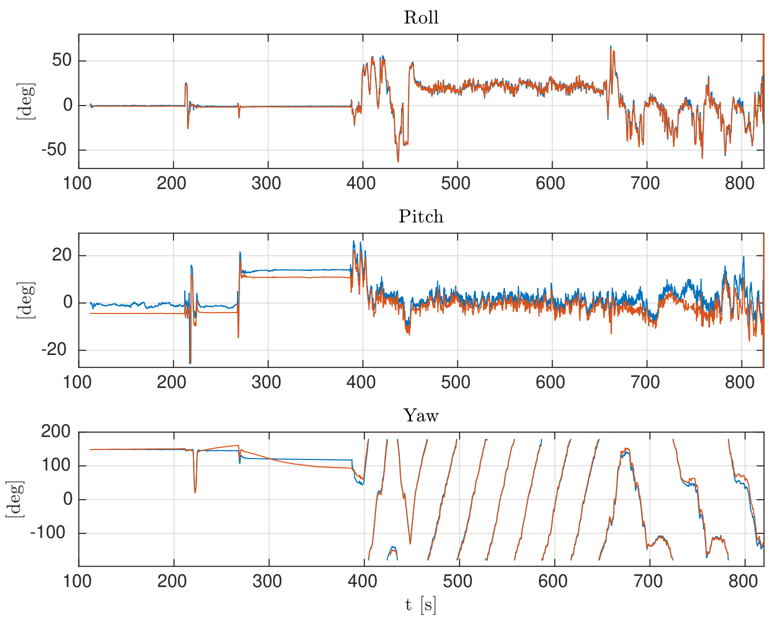

Figure 4.27: Estimated (blue) and true (red) attitude from Breivika.

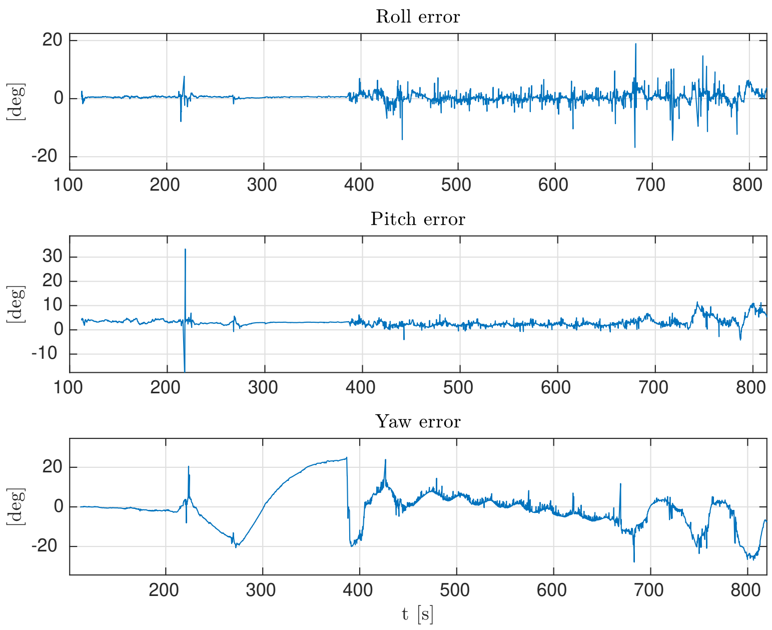

Figure 4.28: Attitude error from Breivika.

5 Conclusion and Further Work

In this report, an ECEF-based Inertial Navigation System for attitude and position estimation is presented. The system is motivated by the rising threat of jamming and other forms of electromagnetic interference. GNSS-based navigation systems can be very accurate, but the low received signal make them easy to disrupt or trick. Using PARS not only increases the resilience against interference, but it also supplies a communication channel that can work over long ranges and with multiple base stations.

The INS that is implemented is based on a Multiplicative Extended Kalman Filter (MEKF), which is one of the ways to solve such a motion estimation problem. It can also be solved with nonlinear observers, but this requires a dedicated heading sensor, even though the computational footprint is smaller. It is desirable to keep the sensors to a minimum, to reduce the need for calibration. The INS is implemented with a few smaller augmentations, such as PARS measurement outlier rejection and ARS calibration. A height correction method is also developed and implemented for cylindrical PARS measurements, and this is tested on a dataset from a long range flight. Height correction enables the support of multiple PARS base stations with different distances, since these would otherwise give very different height estimates. The system is implemented in DUNE Unified Navigation Environment and tested on prerecorded datasets and a live flight. Due to unforeseen circumstances and invalid measurement data, the navigation system failed to estimate the states well online, but running the filter with the corrected data offline showed a great improvement in the system's performance. As shown, the filter performs well with good data. It is able to track the translational and rotational motion of the drone with reasonable accuracy, even with short bursts of ground reflections and PARS outages. Longer periods of PARS outage pose a challenge, which can be mitigated by keeping the IMU biases static while PARS measurements are unavailable. However, there are still some questions to be answered about these types of systems, such as:

- The IMU used in the Skywalker X8 is a tactical grade unit. The system should be tested with cheaper IMUs so as to decide the explicit requirements of the filter.
- The height error compensation method introduced in Section 2.8 is not tested extensively, and should be evaluated for other long-distance flights.
- Although yaw angle estimation has not been a problem in this project, long periods of stationarity can cause it to drift away from the true angle. A mag-

netometer can be used to correct the vehicle's heading estimate. This might justify the use of a cheaper IMU. However, magnetometers need calibration and are susceptible to magnetic field disturbances, so this is not an ideal replacement.

- The use of a barometer and cylindrical coordinates nullifies one of ECEF's greatest advantages, namely the PARS measurements' invulnerability to the curvature of the Earth. Therefore, one may want to look at how the elevation measurement can be processed, especially when flying over bodies of water.
- Tests in steep valleys and fjords should be performed to see how the system acts when subjected to reflections from multiple angles (i.e. not just from the water).
- The introduction of techniques such as multi-hypothesis tracking and tighter coupling between the filter and measurement variable could make outlier rejection easier, by tracking both the true values and the reflections.
- Being a communication (not navigation) system primarily, the PARS device will prefer the wireless channel with the strongest signal, which can often be a reflection. Using the channel with the shortest time-of-flight can decrease the link quality, but increase the quality of the position estimates.
- It should be investigated whether the range measurement can be used as a standalone measurement in the event that the drone is outside the base station's sector.

A Kalman Filter Derivation

[†] Consider the following discrete autonomous system:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}_d \boldsymbol{x}_k + \boldsymbol{v}_k \tag{A.1a}$$

$$\boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{w}_k \tag{A.1b}$$

with $\boldsymbol{x}$ denoting the state vector $\boldsymbol{v}$ and $\boldsymbol{w}$ being zero mean Gaussian noise. We will use $\hat{\boldsymbol{x}}_k^-$ as the a priori state estimate, $\hat{\boldsymbol{x}}_k$ as the a posteriori state estimate and $\boldsymbol{x}_k$ as the true state at time k. We define the error in the estimate:

$$\boldsymbol{e}_k = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k \tag{A.2}$$

Next, we define the process noise and measurement noise covariance matrices:

$$\boldsymbol{Q}_k = \mathbb{E}[\boldsymbol{v}_k \boldsymbol{v}_k^T] \tag{A.3a}$$

$$\boldsymbol{R}_k = \mathbb{E}[\boldsymbol{w}_k \boldsymbol{w}_k^T] \tag{A.3b}$$

Similarly, we define the total error covariance matrix:

$$\boldsymbol{P}_k = \mathbb{E}[\boldsymbol{e}_k \boldsymbol{e}_k^T] \tag{A.4a}$$

$$\boldsymbol{P}_k = \mathbb{E}[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)^T] \tag{A.4b}$$

We define the measurement update equation, which is a scaled error term ($\boldsymbol{K}_k$ being the scaling factor) added to the a priori estimate:

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{y}_k - \boldsymbol{C}\hat{\boldsymbol{x}}_k^-) \tag{A.5}$$

Inserting (A.1b):

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{K}_k \boldsymbol{w}_k - \boldsymbol{K}_k \boldsymbol{C}\hat{\boldsymbol{x}}_k^- \tag{A.6}$$

This is inserted into (A.4b):

$$\boldsymbol{\Gamma} = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^- - \boldsymbol{K}_k \boldsymbol{C}\boldsymbol{x}_k - \boldsymbol{K}_k \boldsymbol{w}_k + \boldsymbol{K}_k \boldsymbol{C}\hat{\boldsymbol{x}}_k^- \tag{A.7a}$$

$$\boldsymbol{P}_k = \mathbb{E}[\boldsymbol{\Gamma}\, \boldsymbol{\Gamma}^T] \tag{A.7b}$$

$$\boldsymbol{P}_k = E[((\boldsymbol{I}_n - \boldsymbol{K}_k \boldsymbol{C})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k) + \boldsymbol{K}_k \boldsymbol{w}_k)((\boldsymbol{I}_n - \boldsymbol{K}_k \boldsymbol{C})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k) + \boldsymbol{K}_k \boldsymbol{w}_k)^T]$$

$$\boldsymbol{P}_k = (\boldsymbol{I}_n - \boldsymbol{K}_k \boldsymbol{C})E[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)^T](\boldsymbol{I}_n - \boldsymbol{K}_k \boldsymbol{C})^T + \boldsymbol{K}_k E[\boldsymbol{w}_k \boldsymbol{w}_k^T]\boldsymbol{K}_k^T$$

We recognize the expression for $\boldsymbol{P}_k$ (A.4b) and $\boldsymbol{R}_k$ (A.3b):

$$\boldsymbol{P}_k = (\boldsymbol{I}_n - \boldsymbol{K}_k \boldsymbol{C})\boldsymbol{P}_k^-(\boldsymbol{I}_n - \boldsymbol{C}^T \boldsymbol{K}_k^T) + \boldsymbol{K}_k \boldsymbol{R}_k \boldsymbol{K}_k^T \tag{A.8}$$

[†]The following text is based on (Vågsether, 2020)

This is one of the equations we will be using. We differentiate the trace of $\boldsymbol{P}_k$ with respect to $\boldsymbol{K}_k$ and set it equal to zero:

$$\boldsymbol{P}_k = \boldsymbol{P}_k^- - \boldsymbol{P}_k^- \boldsymbol{C}^T \boldsymbol{K}_k^T - \boldsymbol{K}_k \boldsymbol{C} \boldsymbol{P}_k^- + \boldsymbol{K}_k (\boldsymbol{C} \boldsymbol{P}_k^- \boldsymbol{C}^T + \boldsymbol{R}_k) \boldsymbol{K}_k^T \tag{A.9a}$$

$$\frac{\partial \operatorname{tr}[\boldsymbol{P}_k]}{\partial \boldsymbol{K}_k} = -\operatorname{tr}[\boldsymbol{P}_k^- \boldsymbol{C}^T] - \operatorname{tr}[\boldsymbol{C} \boldsymbol{P}_k^-] + 2 \operatorname{tr}[\boldsymbol{K}_k (\boldsymbol{C} \boldsymbol{P}_k^- \boldsymbol{C}^T + \boldsymbol{R}_k)] \tag{A.9b}$$

$$0 = -2 \operatorname{tr}[\boldsymbol{P}_k^- \boldsymbol{C}^T] + 2 \operatorname{tr}[\boldsymbol{K}_k (\boldsymbol{C} \boldsymbol{P}_k^- \boldsymbol{C}^T + \boldsymbol{R}_k)] \tag{A.9c}$$

$$2\boldsymbol{P}_k^- \boldsymbol{C}^T = 2\boldsymbol{K}_k (\boldsymbol{C} \boldsymbol{P}_k^- \boldsymbol{C}^T + \boldsymbol{R}_k) \tag{A.9d}$$

We arrive at an expression for $\boldsymbol{K}_k$:

$$\boldsymbol{K}_k = \boldsymbol{P}_k^- \boldsymbol{C}^T (\boldsymbol{C} \boldsymbol{P}_k^- \boldsymbol{C}^T + \boldsymbol{R}_k)^{-1} \tag{A.10}$$

This is the second equation that we will use. The remaining equations are the state estimate propagation steps:

$$\hat{\boldsymbol{x}}_k \leftarrow \boldsymbol{A}_d \hat{\boldsymbol{x}}_{k-1} \tag{A.11a}$$

$$\hat{\boldsymbol{x}}_k \leftarrow \hat{\boldsymbol{x}}_k + \boldsymbol{K}_k (\boldsymbol{y}_k - \boldsymbol{C} \hat{\boldsymbol{x}}_k) \tag{A.11b}$$

and finally, a step that propagates the $\boldsymbol{P}$ matrix and adds the process noise.

$$\boldsymbol{P}_k \leftarrow \boldsymbol{A}_d \boldsymbol{P}_{k-1} \boldsymbol{A}_d^T + \boldsymbol{Q}_d \tag{A.12}$$

The Kalman Filter equations are summarized and sorted in Table A.1.

Table A.1: The standard Kalman Filter steps and their equations (autonomous systems).

Step	**Equations**
Time update	$\hat{\boldsymbol{x}}_k \leftarrow \boldsymbol{A}_d \hat{\boldsymbol{x}}_{k-1}$
	$\boldsymbol{P}_k \leftarrow \boldsymbol{A}_d \boldsymbol{P}_{k-1} \boldsymbol{A}_d^T + \boldsymbol{Q}_d$
Measurement update	$\boldsymbol{K}_k \leftarrow \boldsymbol{P}_k \boldsymbol{C}^T (\boldsymbol{C} \boldsymbol{P}_k \boldsymbol{C}^T + \boldsymbol{R}_k)^{-1}$
	$\boldsymbol{P}_k \leftarrow (\boldsymbol{I}_n - \boldsymbol{K}_k \boldsymbol{C}) \boldsymbol{P}_k (\boldsymbol{I}_n - \boldsymbol{C}^T \boldsymbol{K}_k^T) + \boldsymbol{K}_k \boldsymbol{R}_k \boldsymbol{K}_k^T$
	$\hat{\boldsymbol{x}}_k \leftarrow \hat{\boldsymbol{x}}_k + \boldsymbol{K}_k (\boldsymbol{y}_k - \boldsymbol{C} \hat{\boldsymbol{x}}_k)$

[††]

[††]The above text is based on (Vågsether, 2020)

Code B

B.1 Task.cpp

This is the MEKF DUNE task that is evaluated in this report. The code below is for illustration, and the full source code can be found at NTNU ITK's UAVLAB Gitlab server. For access to code and datasets, please contact my supervisors Torleiv Håland Bryne or Kristoffer Gryte. The code can be found on the following branches: `haakov/raudstein`, `haakov/udduvoll` and `haakov/breivika` under the `dune-ntnu` repository. These branches also include a folder called `matlab`, where some scripts to display and evaluate the filter estimates are found.

```
// DUNE headers.
#include <DUNE/DUNE.hpp>
#include <DUNE/Math/Matrix.hpp>
#include <DUNE/Math/Quaternion.hpp>
#include <DUNE/Math/EulerAnglesZyx.hpp>
#include <DUNE/Coordinates.hpp>
#include <tuple>
#include <list>
#include <numeric>
#include <mutex>
#include <USER/Coordinates/WGS84.hpp>

#include <chrono>
#include <sys/time.h>
#include <atomic>
#include <iostream>
#include <fstream>


namespace Navigation
{
  namespace haakov_MEKF
  {
    using DUNE_NAMESPACES;
    using Vector = DUNE::Math::Matrix;

    //! %Task arguments.
    struct Arguments
    {
      std::string acc_src;
      bool use_cre;
    };

    struct Task: public DUNE::Tasks::Task
    {
      //! Task arguments.
      Arguments m_args;


    std::mutex lock_;

    std::atomic<bool> m_init;
    std::atomic<bool> m_just_corrected;
    std::atomic<bool> m_calibrating;
    std::atomic<bool> m_correction_waiting;
    std::atomic<bool> m_cre_waiting;
    std::atomic<bool> m_csv;

    std::atomic<int> c_imu;
    std::atomic<int> c_pars;
    std::atomic<int> c_ext;

    Vector m_p_e_eb;
    Vector m_v_e_eb;
    Vector m_q_e_eb;
    Vector m_b_acc_b;
    Vector m_b_ars_b;

    Vector last_gnss_pos, last_gnss_rpy;

    Matrix m_C;
    Matrix m_P;
    Matrix m_Q;

    double t_last;
    double last_height = 0.0;
    double base_height = 0.0;

    double m_imu_msg_prev_time = 0.0;

    Vector m_incoming_acc;
    Vector m_incoming_ars;
    Vector m_incoming_gnss;
    Vector m_incoming_pars;
    int m_acc_points = 0;
    int m_ars_points = 0;

    double m_R_yaw;

    double m_initialize_time = 0.0;

    Matrix Tacc_inv, Tars_inv;
    Matrix last_cre_pos, last_cre_R;

    Matrix R_bm;

    Matrix I3;

    std::ofstream f_imu, f_gnss, f_pars, f_rawpars, f_est, f_rpy, f_P,
  f_error;

    //! Constructor.
```

```cpp
    //! @param[in] name task name.
    //! @param[in] ctx context.
    Task(const std::string& name, Tasks::Context& ctx):
      DUNE::Tasks::Task(name, ctx)
    {
      param("CRE", m_args.use_cre)
      .defaultValue("true")
      .size(1)
      .description("Show CRE position");

      m_args.use_cre = true;

      m_csv.store(true);
      if (m_csv.load())
      {
      f_imu.open ("imu.csv", std::ios::out);
      f_imu << "t,acc_x,acc_y,acc_z,ars_x,ars_y,ars_z" << std::endl;

      f_gnss.open ("gnss.csv", std::ios::out);
      f_gnss << "t,gnss_x,gnss_y,gnss_z,gnss_lat,gnss_lon,gnss_h" << std::endl;

      f_pars.open ("pars.csv", std::ios::out);
      f_pars << "t,pars_x,pars_y,pars_z,pars_lat,pars_lon,pars_h" << std::endl;

      f_rawpars.open ("rawpars.csv", std::ios::out);
      f_rawpars << "t,rawpars_x,rawpars_y,rawpars_z,rawpars_lat,rawpars_lon,rawpars_h,rawpars_d,rawpars_psi,rawpars_alpha" << std::endl;

      f_est.open ("est.csv", std::ios::out);
      f_est << "t,est_x,est_y,est_z,est_vx,est_vy,est_vz,est_lat,est_lon,est_h,est_r,est_p,est_y,baccx,baccy,baccz,barsx,barsy,barsz" << std::endl;

      f_rpy.open ("rpy.csv", std::ios::out);
      f_rpy << "t,roll,pitch,yaw" << std::endl;

      f_P.open ("P.csv", std::ios::out);
      f_P << "t,perr,p3sig,roll_err,roll_3sig,pitch_err,pitch_3sig,yaw_err,yaw_3sig" << std::endl;

      f_error.open ("err.csv", std::ios::out);
      f_error << "t,x_err,y_err,z_err,roll_err,pitch_err,yaw_err,lat_err,lon_err,h_err" << std::endl;
      }

      lock_.lock();
      //bind<IMC::Acceleration>(this);
      //bind<IMC::AngularVelocity>(this);
      bind<IMC::Imu>(this);
      bind<IMC::ExternalNavData>(this);
```

```
        bind<IMC::SphericalPositionMeasurement>(this);

        m_init.store(false);
        m_just_corrected.store(false);
        m_calibrating.store(true);
        m_correction_waiting.store(false);
        m_cre_waiting.store(false);


        m_p_e_eb = Vector(3, 1, 0.0);
        m_p_e_eb(0) = 0.0;
        m_p_e_eb(1) = 0.0;
        m_p_e_eb(2) = 0.0;

        last_gnss_pos = Vector(3, 1, 0.0);
        last_gnss_pos(0) = 0.0;
        last_gnss_pos(1) = 0.0;
        last_gnss_pos(2) = 0.0;

        last_gnss_rpy = Vector(3, 1, 0.0);
        last_gnss_rpy(0) = 0.0;
        last_gnss_rpy(1) = 0.0;
        last_gnss_rpy(2) = 0.0;

        m_v_e_eb = Vector(3, 1, 0.0);

        m_q_e_eb = Vector(4, 1, 0.0);
        m_q_e_eb(0) =  0.300767938616783;
        m_q_e_eb(1) = -0.36092152634014;
        m_q_e_eb(2) = -0.882252619942563;
        m_q_e_eb(3) = -0.0300767938616783;
        m_q_e_eb = m_q_e_eb / m_q_e_eb.norm_2();

        m_b_acc_b = Vector(3, 1, 0.0);
        m_b_ars_b = Vector(3, 1, 0.0);

        m_incoming_acc = Vector(3, 1, 0.0);
        m_incoming_ars = Vector(3, 1, 0.0);
        m_incoming_gnss = Vector(3, 1, 0.0);
        m_incoming_pars = Vector(3, 1, 0.0);

        m_R_yaw = ((M_PI/180.0)*2.0)*((M_PI/180.0)*2.0);

        Tacc_inv = Matrix(3, 3, 0.0);
        Tars_inv = Matrix(3, 3, 0.0);

        Tacc_inv(0, 0) = 0.0002778;
        Tacc_inv(1, 1) = 0.0002778;
        Tacc_inv(2, 2) = 0.0002778;
        //Tacc_inv = 0.2 * Tacc_inv;

        Tars_inv(0, 0) = 0.0002778;
        Tars_inv(1, 1) = 0.0002778;
        Tars_inv(2, 2) = 0.0002778;
        //Tars_inv = 0.2 * Tars_inv;

        R_bm = Matrix(3, 3, 0.0);
        R_bm(0,1) = -1.0;
        R_bm(1,0) = -1.0;
        R_bm(2,2) = -1.0;

        I3 = Matrix(3, 3, 0.0);
        I3(0,0) = 1.0;
        I3(1,1) = 1.0;
        I3(2,2) = 1.0;

        m_C = Matrix(3, 15, 0.0);
        m_C(0, 0) = 1.0;
        m_C(1, 1) = 1.0;
        m_C(2, 2) = 1.0;

        m_Q = Matrix(12, 12, 0.0);
        double q_acc = 2.57*std::pow(10, -2);
        double q_ars = 9.59*std::pow(10, -4);
        double q_bacc = 2.55*std::pow(10, -4);
        double q_bars = 6.29*std::pow(10, -8);

        m_Q(0, 0) = q_acc*q_acc;
        m_Q(1, 1) = q_acc*q_acc;
        m_Q(2, 2) = q_acc*q_acc;

        m_Q(3, 3) = q_ars*q_ars;
        m_Q(4, 4) = q_ars*q_ars;
        m_Q(5, 5) = q_ars*q_ars;

        m_Q(6, 6) = q_bacc*q_bacc;
        m_Q(7, 7) = q_bacc*q_bacc;
        m_Q(8, 8) = q_bacc*q_bacc;

        m_Q(9, 9) = q_bars*q_bars;
        m_Q(10, 10) = q_bars*q_bars;
        m_Q(11, 11) = q_bars*q_bars;

        //m_Q = 50 * m_Q;

        m_P = Matrix(15, 15, 0.0);

        m_P(0, 0) = 100.0;
        m_P(1, 1) = 100.0;
        m_P(2, 2) = 100.0;

        m_P(3, 3) = 4.0;
        m_P(4, 4) = 4.0;
        m_P(5, 5) = 4.0;

        m_P(6, 6) = 0.03;
        m_P(7, 7) = 0.03;
        m_P(8, 8) = 0.03;

        m_P(9, 9)   = 1;
        m_P(10, 10) = 1;
        m_P(11, 11) = 1;

        m_P(12, 12) = 0.000003;
        m_P(13, 13) = 0.000003;
        m_P(14, 14) = 0.000003;
        lock_.unlock();
      }

      //! Update internal state with new parameter values.
      void
      onUpdateParameters(void)
      {
      }

      //! Reserve entity identifiers.
      void
      onEntityReservation(void)
      {
      }

      //! Resolve entity names
      void
      onEntityResolution(void)
      {
      }

      //! Acquire resources.
      void
      onResourceAcquisition(void)
      {
      }

      //! Initialize resources.
      void
      onResourceInitialization(void)
      {
        setEntityState(IMC::EntityState::ESTA_NORMAL, Status::CODE_ACTIVE)
    ;
        war("initialized");
      }

      //! Release resources.
      void
      onResourceRelease(void)
      {
      }


      void
      consume(const IMC::Imu* imu)
      {

        double T_s = 0.0;
        if (m_imu_msg_prev_time > 0.0)
        {
          const fp64_t imu_msg_current_time = imu->angular_velocity->time;
          T_s = double( imu_msg_current_time - m_imu_msg_prev_time );
          m_imu_msg_prev_time = imu_msg_current_time;
        }
        else
        {
          m_imu_msg_prev_time = imu->angular_velocity->time;
          return;
        }


        t_last = imu->angular_velocity->time;
        if (m_initialize_time < 0.1)
        {
          m_initialize_time = t_last;
        }
        if ((t_last - m_initialize_time > 30) && (m_calibrating.load()))
        {
          m_calibrating.store(false);
          war("Calibration done");
        }

        Vector incoming_acc(3, 1, 0.0);
        incoming_acc(0) = imu->acceleration->x;
        incoming_acc(1) = imu->acceleration->y;
        incoming_acc(2) = imu->acceleration->z;

        Vector incoming_ars(3, 1, 0.0);
        incoming_ars(0) = imu->angular_velocity->x;
        incoming_ars(1) = imu->angular_velocity->y;
        incoming_ars(2) = imu->angular_velocity->z;


        if (m_calibrating.load())
        {
          calibUpdate(incoming_acc, incoming_ars);
        }
        else
        {
          if (m_init.load())
          {
            timeUpdate(T_s, incoming_acc, incoming_ars);
          }


        }
```

```
    }

    Matrix
    Rzyx(double phi, double theta, double psi)
    {
      Matrix R = Matrix(3,3,0.0);

      R(0,0) = cos(psi)*cos(theta);
      R(0,1) = -sin(psi)*cos(phi) + cos(psi)*sin(theta)*sin(phi);
      R(0,2) = sin(psi)*sin(phi) + cos(psi)*cos(phi)*sin(theta);

      R(1,0) = sin(psi)*cos(theta);
      R(1,1) = cos(psi)*cos(phi) + sin(phi)*sin(theta)*sin(psi);
      R(1,2) = -cos(psi)*sin(phi) + sin(theta)*sin(psi)*cos(phi);


      R(2,0) = -sin(theta);
      R(2,1) = cos(theta)*sin(phi);
      R(2,2) = cos(theta)*cos(phi);
      return R;
    }

    void
    consume(const IMC::ExternalNavData* gnss)
    {
      if (m_calibrating.load())
      {
        return;
      }
      spew("GNSS recv");
      double x, y, z;
      Coordinates::WGS84::toECEF(gnss->state.get()->lat, gnss->state.get
  ()->lon, gnss->state.get()->height, &x, &y, &z);

      last_height = gnss->state.get()->height;
      if ((180.0/M_PI)*gnss->state.get()->lon < 1.0) {
        war("Skip");
        return;
      }

      Vector gnss_ecef = Vector(3, 1, 0.0);

      gnss_ecef(0) = x;
      gnss_ecef(1) = y;
      gnss_ecef(2) = z;

      if (not m_init.load())
      {
        initialize(gnss_ecef);
      }

      last_gnss_pos(0) = x;
      last_gnss_pos(1) = y;
      last_gnss_pos(2) = z;

      last_gnss_rpy(0) = (180.0/M_PI)*gnss->state->phi;
      last_gnss_rpy(1) = (180.0/M_PI)*gnss->state->theta;
      last_gnss_rpy(2) = (180.0/M_PI)*gnss->state->psi;

      double lat, lon, h;
      Coordinates::WGS84::fromECEF(m_p_e_eb(0), m_p_e_eb(1), m_p_e_eb(2)
  , &lat, &lon, &h);

    }

    void
    consume(const IMC::SphericalPositionMeasurement* pars)
    {
      if (t_last > 840.0) {return;}
      double range;
      double azimuth;
      double elevation;



      if (m_calibrating.load() or not m_init.load())
      {
        return;
      }

      spew("PARS recv");

      double x, y, z;
      Coordinates::WGS84::toECEF(pars->base_lat, pars->base_lon, pars->
  base_height, &x, &y, &z);
      base_height = pars->base_height;

      Vector base_ecef = Vector(3, 1, 0.0);

      base_ecef(0) = x;
      base_ecef(1) = y;
      base_ecef(2) = z;

      Matrix R_p = Matrix(3, 3, 0.0);
      R_p(0,0) = (50.0*50.0);
      R_p(1,1) = ((M_PI/180.0)*3)*((M_PI/180.0)*3);
      R_p(2,2) = 5.0*5.0;

      double horizontal_dist;
      if ( (range < 10000000) && (range > 0.01) && (azimuth <= M_PI) &&
  (azimuth >= -M_PI)  )
      {}

      else
      {
```

```
      war("CRE meas not vaild. Range = %f[m], Azimuth = %f[deg]", range, azimuth*180.0/M_PI);
      return;
    }
    double E = heightError(base_ecef, m_p_e_eb);
    E = 0.0; // Disable height correction
    double pars_height = last_height - base_height - E;
    if (range - abs((double)pars_height) < 5 ) {
      horizontal_dist = range;
    } else {
      horizontal_dist = sqrt(range*range - pars_height*pars_height);
    }

    auto M_ = M_cyl(pars_height, azimuth, range, horizontal_dist);
    auto R_pars = Ren(base_ecef) * M_ * R_p * transpose(M_) * transpose(Ren(base_ecef));

    Vector pars_ecef = Vector(3, 1, 0.0);
    double yaw_offset = (M_PI/180.0)*-10;
    pars_ecef = base_ecef + Ren(base_ecef) * Vpars_cyl(pars_height, azimuth, horizontal_dist, pars->base_yaw + yaw_offset);
    std::cout << "base yaw " << (180.0/M_PI)*(pars->base_yaw + yaw_offset) << std::endl;



    double lat, lon, h;
    Coordinates::WGS84::fromECEF(pars_ecef(0), pars_ecef(1), pars_ecef(2), &lat, &lon, &h);

    Matrix T = transpose(pars_ecef - m_p_e_eb) * inverse(m_C * m_P * transpose(m_C) + R_pars) * (pars_ecef - m_p_e_eb);
    if (T(0,0) < 11.345)
    {

      measurementUpdate(pars_ecef, R_pars);
    } else {war("REJECT");}
  }

  void
  timeUpdate(double stepSize, Vector acc, Vector ars)
  {
    using namespace std::chrono;
    lock_.lock();

    // Rotate imu vectors
    acc = R_bm * acc - m_b_acc_b;
    ars = R_bm * ars - m_b_ars_b;

    Vector g = g_e(m_p_e_eb);

    Vector acc_e = Rquat(m_q_e_eb)*acc + g;
    // Propagate nominal states
```

```
   m_p_e_eb = m_p_e_eb + stepSize * m_v_e_eb + stepSize*stepSize * acc_e * 0.5;
   m_v_e_eb = m_v_e_eb + stepSize * acc_e;

   if (ars.norm_2() > 0.00000001) {
    Vector d_attitude(4, 1, 0.0);
    d_attitude(0) = cos(stepSize *ars.norm_2()/2);
    d_attitude.put(1, 0, (ars / ars.norm_2()) * sin(stepSize * ars.norm_2()/2));
    m_q_e_eb = quaternionProduct(m_q_e_eb, d_attitude);
    m_q_e_eb = m_q_e_eb / m_q_e_eb.norm_2();
   }

   m_b_acc_b = m_b_acc_b - stepSize * Tacc_inv * m_b_acc_b;
   m_b_ars_b = m_b_ars_b - stepSize * Tars_inv * m_b_ars_b;

   // Compute discrete A and Q
   auto Ac = A(Rquat(m_q_e_eb), acc, ars);

   Matrix O3 = Matrix(3, 3, 0.0);
   Matrix I3 = Identity(3);



   Matrix G(15, 12, 0.0);
   G.put(3, 0, -Rquat(m_q_e_eb));
   G.put(6, 3, -I3);
   G.put(9, 6, I3);
   G.put(12, 9, I3);

   auto AdQd = van_Loan(Ac, G, m_Q, stepSize);

   Matrix Ad = std::get<0>(AdQd);
   Matrix Qd = std::get<1>(AdQd);

   m_P = Ad * m_P * transpose(Ad) + Qd;

   lock_.unlock();

   m_just_corrected.store(false);
   double lat, lon, h;
   WGS84::fromECEF(m_p_e_eb(0), m_p_e_eb(1), m_p_e_eb(2), &lat, &lon, &h);
   Vector rpy = R2euler(transpose(Ren()) * Rquat(m_q_e_eb));
  }

  double
  sinc( double var ) const
  {
   if ( abs(var) < 1e-9 )
   {
    //std::cout << "var: " << var << std::endl;
    return 1.0;
```

```
      }
      else
      {
        return (std::sin(var)/var);
      }

    }

    void
    calibUpdate(Vector acc, Vector ars)
    {
      double stepSize = 0.004;
      lock_.lock();

      Matrix C_ars = Matrix(3, 15, 0.0);
      C_ars(0, 12) = 1.0;
      C_ars(1, 13) = 1.0;
      C_ars(2, 14) = 1.0;
      Matrix R_ars = Matrix(3, 3, 0.0);
      R_ars = (m_Q(3,3) / stepSize) * I3;

      Matrix K = m_P*transpose(C_ars)*(inverse(C_ars*m_P*transpose(C_ars
  ) + R_ars));

      Vector dx = K*(ars - m_b_ars_b);
      m_P = (Identity(15) - K*C_ars) * m_P * (transpose(Identity(15) - K
  *C_ars)) + K*R_ars*transpose(K);

      m_b_ars_b = m_b_ars_b + dx.get(12, 14, 0, 0);

      lock_.unlock();
    }

    double
    heightError(Vector base_ecef, Vector uav_ecef)
    {
      double num = (transpose(base_ecef) * uav_ecef)(0);
      double den = base_ecef.norm_2() * uav_ecef.norm_2();
      double gamma = acos(num / den);
      return base_ecef.norm_2() * (1 - cos(gamma));
    }

    void
    measurementUpdate(Vector position, Vector R) {
      double lat, lon, h;
      WGS84::fromECEF(position(0), position(1), position(2), &lat, &lon,
   &h);
      lock_.lock();

      // Calculate error states and Kalman gain
      Matrix K = m_P*transpose(m_C)*inverse(m_C*m_P*transpose(m_C) + R);
      Vector dx = K*(position - m_p_e_eb);
    m_P = (Identity(15) - K*m_C) * m_P * transpose(Identity(15) - K*m_C) + K*R*transpose(K);
    // Inject error into nominal states
    m_p_e_eb = m_p_e_eb + dx.get(0, 2, 0, 0);

    m_v_e_eb = m_v_e_eb + dx.get(3, 5, 0, 0);

    Vector da = dx.get(6, 8, 0, 0);
    Vector dq = Vector(4, 1, 0.0);

    dq.put(0, 0, Matrix(1, 1, 16.0) - transpose(da)*da);
    dq.put(1, 0, 8*da);
    dq =  dq / (16.0 + (transpose(da)*da)(0));
    dq = dq / dq.norm_2();

    // error > 180 deg, replace dq by its shadow set
    if (da.norm_2() > 4.0) {
        dq = -dq;
        std::cout << "Shadow set\n";
    }

    m_q_e_eb = quaternionProduct(m_q_e_eb, dq);
    m_q_e_eb = m_q_e_eb / m_q_e_eb.norm_2();
    m_b_acc_b = m_b_acc_b + dx.get(9, 11, 0, 0);
    m_b_ars_b = m_b_ars_b + dx.get(12, 14, 0, 0);

    // Reset covariance
    Matrix G = Identity(15);
    G.put(6, 6, dq(0) * I3 - skew(dq.get(1, 3, 0, 0)));
    m_P = G*m_P*transpose(G);
    lock_.unlock();
  }

  Vector
  g_e(Vector position) {
    // WGS84
    double a          = 6378137.0;         // equatorial radius in meters
    double mu         = 3.986004418e14;  // Earth gravitational constant
    double J_2        = 1.082627e-3;     // Earth's second gravitational constant
    double omega_ie   = 7.292115e-5;     // Earth's rotation rate (rad/s)

    double mag_r = position.norm_2();

    double z_scale = 5.0 * std::pow((position(2) / mag_r), 2);

    Vector gamma = Vector(3, 1, 0.0);
    Vector ret = Vector(3, 1, 0.0);
    gamma(0) = (1.0 - z_scale) * position(0);
    gamma(1) = (1.0 - z_scale) * position(1);
```

```
        gamma(2) = (3.0 - z_scale) * position(2);

        gamma = (-mu / std::pow(mag_r, 3)) * (position + 1.5 * J_2 * std::
    pow(a / mag_r, 2) * gamma);
        ret.put(0, 0, gamma.get(0, 2, 0, 0) + std::pow(omega_ie, 2) *
    position.get(0, 2, 0, 0));

        ret(2) = gamma(2);
        return ret;
      }

      Matrix A(Matrix R, Vector accl, Vector gyro) {
        Matrix A = Matrix(15, 15, 0.0);
        // row 0
        A.put(0, 3, Identity(3));
        // row 1
        A.put(3, 6, -R * skew(accl));
        A.put(3, 9, -R);
        // row 2
        A.put(6, 6, -skew(gyro));
        A.put(6, 12, -Identity(3));
        // row 3
        A.put(9, 9, -Tacc_inv);
        // row 4
        A.put(12, 12, -Tars_inv);
        return A;
      }

      Vector
      Rquat(Vector quat) {
        auto q = Quaternion(quat);
        return q.rotationMatrix();
        Vector S = Vector(3, 1, 0.0);
        S(0) = quat(1);
        S(1) = quat(2);
        S(2) = quat(3);
        S = skew(S);
        return I3 + 2*quat(0)*S + 2*S*S;
      }

      Vector quaternionProduct(Vector lhs, Vector rhs) {
        Vector result = Vector(4, 1, 0.0);
        Vector eps_l = Vector(3, 1, 0.0);
        Vector eps_r = Vector(3, 1, 0.0);

        eps_l(0) = lhs(1);
        eps_l(1) = lhs(2);
        eps_l(2) = lhs(3);

        eps_r(0) = rhs(1);
        eps_r(1) = rhs(2);
        eps_r(2) = rhs(3);

      result.put(0, 0, lhs(0) * rhs(0) * Vector(1, 1, 1.0) - transpose(
  eps_l) * eps_r);
      result.put(1, 0, lhs(0)*eps_r + rhs(0)*eps_l + skew(eps_l) * eps_r
  );
      return result;
    }

    std::tuple<Matrix, Matrix>
    van_Loan(Matrix A, Matrix G, Matrix Q, double Ts)
    {
      Matrix A_ = Matrix(30, 30, 0.0);
      A_.put(0, 0, -A);
      A_.put(0, 15, G * Q * transpose(G));
      A_.put(15, 15, transpose(A));
      A_ = Ts * A_;

      Matrix B = Identity(30);

      B = B + A_ + A_*A_/2.0;

      Matrix Ad = transpose(B.get(15, 29, 15, 29));
      Matrix Qd = Ad * B.get(0, 14, 15, 29);

      return std::make_tuple(Ad, Qd);
    }

    Vector
    R2euler(Matrix R)
    {
      auto phi = atan2(R(2, 1), R(2, 2));
      auto theta = -asin(R(2, 0));
      auto psi = atan2(R(1, 0), R(0, 0));

      Vector ret = Vector(3, 1, 0.0);
      ret(0) = phi;
      ret(1) = theta;
      ret(2) = psi;
      return ret;
    }

    Matrix
    Ren() {
      double lat, lon, h;
      WGS84::fromECEF(m_p_e_eb(0), m_p_e_eb(1), m_p_e_eb(2), &lat, &lon,
   &h);

      Matrix ret = Matrix(3, 3, 0.0);

      ret(0, 0) = -sin(lat)*cos(lon);
      ret(0, 1) = -sin(lon);
      ret(0, 2) = -cos(lat)*cos(lon);

      ret(1, 0) = -sin(lat)*sin(lon);
```

```
        ret(1, 1) = cos(lon);
        ret(1, 2) = -cos(lat)*sin(lon);

        ret(2, 0) = cos(lat);
        ret(2, 1) = 0;
        ret(2, 2) = -sin(lat);

        return ret;
      }


      Matrix
      Ren(Vector ecef_pos) {
        double lat, lon, h;
        WGS84::fromECEF(ecef_pos(0), ecef_pos(1), ecef_pos(2), &lat, &lon,
     &h);

        Matrix ret = Matrix(3, 3, 0.0);

        ret(0, 0) = -sin(lat)*cos(lon);
        ret(0, 1) = -sin(lon);
        ret(0, 2) = -cos(lat)*cos(lon);

        ret(1, 0) = -sin(lat)*sin(lon);
        ret(1, 1) = cos(lon);
        ret(1, 2) = -cos(lat)*sin(lon);

        ret(2, 0) = cos(lat);
        ret(2, 1) = 0;
        ret(2, 2) = -sin(lat);

        return ret;
      }

      Vector
      Vpars(double pitch, double yaw, double dist, double base_pitch,
    double base_yaw)
      {
        Vector ret = Matrix(3, 1, 0.0);

        ret(0) = cos(yaw + base_yaw)*cos(pitch + base_pitch);
        ret(1) = sin(yaw + base_yaw)*cos(pitch + base_pitch);
        ret(2) = -sin(pitch + base_pitch);
        return dist * ret;
      }

      Matrix
      M(double pitch, double yaw, double dist)
      {
        Vector ret = Matrix(3, 3, 0.0);

        ret(0,0) = cos(yaw)*cos(pitch);
        ret(0,1) = -dist*sin(yaw)*cos(pitch);
      ret(0,2) = -dist*cos(yaw)*sin(pitch);

      ret(1,0) = sin(yaw)*cos(pitch);
      ret(1,1) = dist*cos(yaw)*cos(pitch);
      ret(1,2) = -dist*sin(yaw)*sin(pitch);

      ret(2,0) = -sin(pitch);
      ret(2,1) = 0.0;
      ret(2,2) = -dist*cos(pitch);

      return ret;
    }

    Vector
    Vpars_cyl(double height, double yaw, double horizontal_dist, double
  base_yaw)
    {
      Vector ret = Matrix(3, 1, 0.0);
      double bias = std::exp(-m_R_yaw/2);

      ret(0) = horizontal_dist * cos(yaw + base_yaw)/bias;
      ret(1) = horizontal_dist * sin(yaw + base_yaw)/bias;
      ret(2) = -height;
      return ret;
    }

    Matrix
    M_cyl(double height, double yaw, double dist, double horizontal_dist
  )
    {
      Vector ret = Matrix(3, 3, 0.0);

      ret(0,0) = cos(yaw)*dist / horizontal_dist;
      ret(0,1) = -sin(yaw)*horizontal_dist;
      ret(0,2) = -cos(yaw)*height / horizontal_dist;

      ret(1,0) = sin(yaw)*dist / horizontal_dist;
      ret(1,1) = cos(yaw)*horizontal_dist;
      ret(1,2) = -sin(yaw)*height  / horizontal_dist;

      ret(2,0) = 0.0;
      ret(2,1) = 0.0;
      ret(2,2) = -1.0;

      return ret;
    }

    Matrix
    Identity(int dim)
    {
      Matrix ret = Matrix(dim, dim, 0.0);
      for (int i = 0; i < dim; i++)
      {
          ret(i, i) = 1.0;
        }
        return ret;
      }
      void
      initialize(Vector position)
      {
        m_p_e_eb = position;

        double init_pitch = 0.0;
        double init_roll = 0.0;
        // Raudstein
        double init_hdg = (M_PI/180) * 327.0;

        // Udduvoll
        init_hdg = (M_PI/180) * 122.9;
        // Breivika
        init_hdg = (M_PI/180) * 148.5;

        Matrix Rx = Matrix(3, 3, 0.0);

        Rx(0,0) = 1.0;

        Rx(1,1) = cos(init_roll);
        Rx(1,2) = -sin(init_roll);

        Rx(2,1) = sin(init_roll);
        Rx(2,2) = cos(init_roll);

        Matrix Ry = Matrix(3, 3, 0.0);

        Ry(0,0) = cos(init_pitch);
        Ry(0,2) = sin(init_pitch);

        Ry(1,1) = 1.0;

        Ry(2,0) = -sin(init_pitch);
        Ry(2,2) = cos(init_pitch);

        Matrix Rz = Matrix(3, 3, 0.0);

        Rz(0,0) = cos(init_hdg);
        Rz(0,1) = -sin(init_hdg);

        Rz(1,0) = sin(init_hdg);
        Rz(1,1) = cos(init_hdg);

        Rz(2,2) = 1.0;

        auto intermediate_q = (Ren() * Rz).toQuaternion();
        m_q_e_eb(0) = -intermediate_q(3);
        m_q_e_eb(1) = intermediate_q(0);
        m_q_e_eb(2) = intermediate_q(1);
        m_q_e_eb(3) = intermediate_q(2);
        war("MEKF init!");
        std::cout << "q " << std::endl << m_q_e_eb << std::endl;
        std::cout << "g_E " << std::endl << g_e(m_p_e_eb) << std::endl;

        m_init.store(true);
        return;
      }

      //! Main loop.
      void
      onMain(void)
      {
        using namespace std::chrono;

        auto start_time = steady_clock::now();
        while (!stopping())
        {
          waitForMessages(1.0);
        }
        if (m_csv.load())
        {
        f_imu.close();
        f_gnss.close();
        f_pars.close();
        f_rawpars.close();
        f_rpy.close();
        f_est.close();
        f_P.close();
        f_error.close();
        }
      }
    };
  }
}

DUNE_TASK
```

References

Adresseavisen (2021). Oppdaget ulovlig jammer i bil ved luftambulansebasen i trondheim. [Online; accessed May 30, 2021].
URL: *https://www.adressa.no/incoming/2021/04/27/Oppdaget-ulovlig-jammer-i-bil-ved-luftambulansebasen-i-Trondheim-23872419.ece*

Aftenposten (2019). Skjermen i legehelikopteret gikk i svart. politiet fant årsaken i to lastebiler ved e6. [Online; accessed May 30, 2021].
URL: *https://www.aftenposten.no/norge/i/a25bWO/lastebilsjaafoerer-blokkerte-gps-signalene-til-ambulansehelikopter-det*

Albrektsen, S. M., Bryne, T. H., and Johansen, T. A. (2018). Phased array radio system aided inertial navigation for unmanned aerial vehicles. In *2018 IEEE Aerospace Conference*, pp. 1–11. doi: 10.1109/AERO.2018.8396433.

cybergalactic/Fossen (2020). cybergalactic/mss: Marine systems simulator (mss) - github. [Online; accessed December 16, 2020].
URL: *https://github.com/cybergalactic/MSS*

Fossen, T. (2011). *Handbook of Marine Craft Hydrodynamics and Motion Control*. doi: 10.1002/9781119994138.

Gao, G. X., Sgammini, M., Lu, M., and Kubo, N. (2016). Protecting gnss receivers from jamming and interference. *Proceedings of the IEEE*, 104(6): 1327–1338.

Giorgi, G., Teunissen, P. J., Verhagen, S., and Buist, P. J. (2010). Testing a new multivariate gnss carrier phase attitude determination method for remote sensing platforms. *Advances in Space Research*, 46(2): 118 – 129. GNSS Remote Sensing-1. doi: https://doi.org/10.1016/j.asr.2010.02.023.
URL: *http://www.sciencedirect.com/science/article/pii/S0273117710001419*

Groves, P. D. (2008). *Principles of GNSS, Inertial, and Multi-sensor Integrated Navigation Systems*. Artech House, Inc.

Gryte, K. (2020). Precision control of fixed-wing uav and robust navigation in gnss-denied environments.

Gryte, K., Bryne, T. H., Albrektsen, S. M., and Johansen, T. A. (2019). Field test results of gnss-denied inertial navigation aided by phased-array radio systems for uavs. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1398–1406. doi: 10.1109/ICUAS.2019.8798057.

Gryte, K., Bryne, T. H., and Johansen, T. A. (2020). Unmanned aircraft flight control aided by phased-array radio navigation. *Journal of Field Robotics*, n/a(n/a). doi: https://doi.org/10.1002/rob.22002.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.22002*

Herd, J. S. and Conway, M. D. (2015). The evolution to modern phased array architectures. *Proceedings of the IEEE*, 104(3): 519–529.

J. Russell Carpenter, C. N. D. (2018). Navigation filter best practices.

lexico.com (2021). Navigation | definition of navigation by oxford dictionary. [Online; accessed May 29, 2021].
URL: *https://www.lexico.com/definition/Navigation*

linux.die.net (2018). Picocom(8) - linux man page. [Online; accessed May 18, 2021].
URL: *https://linux.die.net/man/8/picocom*

Loan, C. (1978). Computing integrals involving the matrix exponential. *Automatic Control, IEEE Transactions on*, 23: 395 – 404. doi: 10.1109/TAC.1978.1101743.

Mahfouz, M. R., Zhang, C., Merkl, B. C., Kuhn, M. J., and Fathy, A. E. (2008). Investigation of high-accuracy indoor 3-d positioning using uwb technology. *IEEE Transactions on Microwave Theory and Techniques*, 56(6): 1316–1330. doi: 10.1109/TMTT.2008.923351.

Markley, L. (2003). Attitude error representations for kalman filtering. *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM*, 26: 311–317. doi: 10.2514/2.5048.

Markley, L. (2004). Multiplicative vs. additive filtering for spacecraft attitude determination.

Misra, P. and Enge, P. (2011). *Global Positioning System: Signals, Measurements, and Performance*. Ganga-Jamuna Press.
URL: *https://books.google.no/books?id=5WJOywAACAAJ*

Mourikis, A. I. and Roumeliotis, S. I. (2007). A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3565–3572. doi: 10.1109/ROBOT.2007.364024.

NASA (2006). Roll, yaw and pitch axis orientation for an airplane. [Online; accessed May 31, 2021].
URL: *https://commons.wikimedia.org/wiki/File:Rollpitchyawplain.png*

Pinker, A. and Smith, C. (1999). Vulnerability of the gps signal to jamming. *GPS Solutions*, 3(2): 19–27.

Pinto, J., Dias, P. S., Martins, R., Fortuna, J., Marques, E., and Sousa, J. (2013). The lsts toolchain for networked vehicle systems. In *2013 MTS/IEEE OCEANS - Bergen*, pp. 1–9. doi: 10.1109/OCEANS-Bergen.2013.6608148.

Schmidt, D., Radke, K., Camtepe, S., Foo, E., and Ren, M. (2016). A survey and analysis of the gnss spoofing threat and countermeasures. *ACM Computing Surveys*, 48: 1–31. doi: 10.1145/2897166.

Sensonor (2021). Stim300 datasheet. [Online; accessed May 18, 2021].
URL: *https://sensonor.azurewebsites.net/media/1445/stim300-datasheet.pdf*

Senti Systems AS (2021). Sentidoc. [Online; accessed May 30, 2021].
URL: *https://gitlab.senti.no/senti/senti-doc*

Solà, J. (2017). Quaternion kinematics for the error-state kalman filter. *CoRR*, abs/1711.02508.
URL: *http://arxiv.org/abs/1711.02508*

Sollie, M., Bryne, T., and Johansen, T. (2019). Pose estimation of uavs based on ins aided by two independent low-cost gnss receivers. pp. 1425–1435. doi: 10.1109/ICUAS.2019.8797746.

Subirana, J., Zornoza, J., Hernández-Pajares, M., Agency, E. S., and Fletcher, K. (2013). *GNSS Data Processing*. Number v. 1 in ESA TM. ESA Communications.
URL: *https://books.google.no/books?id=RO8xngEACAAJ*

Swaszek, P., Hartnett, R., Seals, K., Siciliano, J., and Swaszek, R. (2018). Limits on gnss performance at high latitudes. pp. 160–176. doi: 10.33012/2018.15549.

Vågsether, H. (2020). Resilient satellite- and phased-array-radio-based uav navigation.

Wei, Y., Hong, T., Khelloufi, A., Ning, H., and Xiong, Q. (2019). An application research of kalman filter based algorithms in ecef coordinate system for motion models of sensors. *Procedia Computer Science*, 147: 574 – 580. 2018 International Conference on Identification, Information and Knowledge in the Internet of Things. doi: https://doi.org/10.1016/j.procs.2019.01.214.
URL: *http://www.sciencedirect.com/science/article/pii/S1877050919302339*

Integrated Inertial Navigation of UAVs Aided by Phased-Array Radio Measurements

NTNU
Norwegian University of
Science and Technology