Resilient Satellite- and Phased-Array-Radio-Based UAV Navigation

Håkon Vågsether

December 22, 2020

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

PROJECT THESIS DESCRIPTION SHEET

Name:	Håkon Solevåg Vågsether
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Robust satellitt- og faseradio-basert UAV navigasjon
Thesis title (English):	Resilient satellite and phased-array-radio-based UAV navigation

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 is based on a local flat Earth formulation. Future designs should improve upon this in order to support UAV navigation over larger geographical areas.

The following tasks should be considered

1. Perform a short literature review on
    a. PARS-aided INS,
    b. the vulnerability of GNSS to inference and
    c. Kalman filter residual monitoring techniques.
2. (Derive and)Implement an INS and error-state Kalman filter solution for applying GNSS-position-based INS corrections. The INS and filter should be suitable for long range UAV flights.
3. Implement PARS-based aiding as an alternative to GNSS in case of GNSS outage.
4. If time permits implement a residual monitoring technique capable of detecting GNSS drift due to spoofing.
5. Present and discuss your results data collected in the field.
6. Conclude your results and suggest further work.

Start date:	2020-08-17
Due date:	2020-12-17
Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Associate professor Torleiv H. Bryne, Dept. of Eng. Cybernetics, NTNU

Abstract

GNSS-aided INS Kalman Filters are often vital for Unmanned Aerial Vehicle operations. Even though the technology has been around for decades, GNSS receivers are still vulnerable to electromagnetic interference, largely due to the minuscule power of the received signal. An alternative using Phased Array Radio System measurements is presented and implemented. The implementation uses ECEF coordinates and an error-state Multiplicative Extended Kalman Filter. Test data from a fixed-wing flight on the Trøndelag coast is used to validate the implementation, and artificial spoofing is added to the GNSS signal in order to test the spoofing detection.

Samandrag

GNSS-hjulpne INS-baserte Kalman Filter er ofte essensielle for ubemanna fartøy. Teknologien har vore i bruk i fleire tiår, men GNSS-mottakarar er framleis sårbare for elektromagnetisk interferens. Dette skuldast i stor grad krafta til det mottekne signalet, som er særs svakt. I denne rapporten blir ei løysing med fasestyrte radioar utvikla og implementert. Implementasjonen nyttar kartesiske ECEF-koordinatar og eit Multiplicative Extended Kalman Filter (MEKF). Data frå ei testflyging på trøndelagskysten blir nytta til å validere filteret, og kunstig drift er lagt til GNSS-signalet for å teste ei deteksjonsalgoritme for spoofing.

Contents

1 Introduction **8**
- 1.1 Main contributions . . . 9
- 1.2 Outline of the report . . . 9

2 Background and Preliminaries **10**
- 2.1 Attitude and quaternions . . . 10
- 2.2 Notation . . . 13
- 2.3 Reference frames . . . 14
- 2.4 IMU . . . 18
- 2.5 GNSS . . . 19
- 2.6 GNSS vulnerabilities . . . 20
- 2.7 Phased Array Radio Systems . . . 22
- 2.8 Change detection . . . 25

3 Implementation **27**
- 3.1 Error-state filtering . . . 27
- 3.2 INS update . . . 31
- 3.3 GNSS/PARS update . . . 33
- 3.4 Spoofing detection . . . 34

4 Results **36**
- 4.1 Spoofing disabled . . . 38
- 4.2 Spoofing enabled . . . 49

5 Conclusion and Further Work **52**

A Kalman Filter Derivation **54**

B Code **57**
- B.1 mekf_split.m . . . 57

B.2 gravity.m 72
B.3 van_Loan.m 72

List of Tables

2.1 Reserved symbols and operators. 13

3.1 MEKF steps and equations. 28

4.1 $\boldsymbol{Q}_c$, $\boldsymbol{R}_{gnss}$, $\boldsymbol{R}^s_{pars}$ elements 37
4.2 Initial covariance matrix ($\boldsymbol{P}_0$) elements 37
4.3 PARS-aided position error statistics. 48
4.4 PARS-aided attitude error statistics. 48

A.1 Standard Kalman filter steps and equations. 56

List of Figures

2.1 Body-centered reference frames. 14
2.2 The ECEF frame. 15
2.3 A phased array antenna. 22

3.1 The entire filter, with equation references. 30

4.1 Raw GNSS measurements with horizontal position estimates. . . 39
4.2 Raw PARS measurements with horizontal position estimates. . . 39
4.3 Estimated orientation, relative to NED. 40
4.4 Bias estimates. 41
4.5 GNSS-aided position error states with 3σ bounds. 41
4.6 PARS-aided position error states with 3σ bounds. 42
4.7 Velocity estimates. 43
4.8 Height estimates with raw PARS data. 44
4.9 Position estimates in three dimensions. 44
4.10 Roll, pitch and yaw estimates with incorrect initial attitude. . . . 46
4.11 Zoomed yaw angle estimates with incorrect initial attitude. . . . 46
4.12 The GNSS-aided bias estimates with incorrect initial attitude. . . 47
4.13 Improved roll, pitch and yaw estimates. 47
4.14 No spoofing, but false positive. 49
4.15 No spoofing, no false positive. 50
4.16 Spoofing activated and detected. Same parameters as Figure 4.15. 50
4.17 Position estimates with added 1 m/s GNSS drift. 51

Listings

code/mekf_split.m 57
code/gravity.m 72
code/van_Loan.m 72

Acronyms

AGC Automatic Gain Control. 20

ARS Angular Rate Sensor. 11, 13, 18, 41

CUSUM CUmulative SUM. 9, 25, 34, 51, 52

ECEF Earth-Centered Earth-Fixed. 1, 4, 9, 13, 15, 16, 20, 23, 31, 36, 41, 42, 48

ECI Earth-Centered Inertial. 14, 16

EKF Extended Kalman Filter. 27

ENU East-North-Up. 15

ESA European Space Agency. 19

FOC Full Operating Capability. 19

FPV First Person View. 36

GNSS Global Navigation Satellite System. 1, 4, 8, 9, 18–21, 25, 34, 36, 38, 41, 42, 44, 47, 49, 51–53

GPS Global Positioning System. 8, 19, 20

IMU Inertial Measurement Unit. 14, 15, 18, 31, 34, 36, 41, 47, 52

INS Inertial Navigation System. 1, 9, 25

MAE Mean Average Error. 48

ME Mean Error. 48

MEKF Multiplicative Extended Kalman Filter. 1, 3, 8, 9, 27, 34, 52

MRP Modified Rodrigues Parameters. 11

MSS Marine Systems Simulator. 17

NED North-East-Down. 9, 10, 14–16, 23

PARS Phased Array Radio System. 1, 3, 4, 8, 9, 13, 14, 22, 23, 25, 34, 36, 38, 41–44, 48, 49, 51–53

RMSE Root Mean Squared Error. 48

RTK Real-Time Kinematic. 48

RTT Round-trip time. 23

UAV Unmanned Aerial Vehicle. 1, 8, 9, 14, 23, 36, 38, 45, 49, 52, 53

WGS84 World Geodetic System 1984. 16

Chapter 1

Introduction

As its name suggests, Global Navigation Satellite System (GNSS) is an Earth-wide tool for positioning and navigation, and it is used in countless applications across the globe. The last few decades have seen this kind of technology become ubiquitous in our modern society, with good reason. Today, it is used for everything from mobile games to landing systems for aircraft. The technology makes it possible for a boundless number of users to use the service simultaneously, and the accuracy is more than good enough for most everyday use cases. However, its ubiquity does not guarantee accuracy, and several of its properties make it susceptible to both intentional and unintentional interference. (Arienzo 2010) Jamming is a simple way of disrupting the service, and it can be done in ways that target specific users. Other activities, such as spoofing or meaconing, are more complicated but also more versatile in terms of what can be achieved. With the rise of Unmanned Aerial Vehicles and other autonomous devices, robust navigation becomes increasingly important, and launching an expensive drone without a backup navigation system is associated with high risk.

An auxiliary navigation system can be based on many different technologies, such as elecro-optical, acoustic and electromagnetic sensors. Phased Array Radio System (PARS) is proving to be a viable alternative. Although the precision of such systems is worse than GNSS, it is precise enough to be in contention for navigation systems where satellite-based positioning does not work. This includes indoor environments as well as areas where GNSS is being jammed or questions are being raised about its integrity. This project aims to investigate PARS' utility in such environments, and evaluate whether the difference between a GPS-aided and a PARS-aided Multiplicative Extended Kalman Filter can be used to discover active spoofing.

Some reports have been published on this subject recently, most notably Kristof-

fer Gryte's doctoral thesis (Gryte 2020), which shares a data set with this report. The main difference between Gryte's MEKF implementation and the implementation used in this report is the reference frame in which the estimates are stored. This report uses ECEF, while Gryte uses NED. A similar representation is used in Sollie et al. (2019), but with the purpose of pose estimation using multiple receivers. Other related works include Albrektsen et al. (2018*a*), where nonlinear observers are used for motion and attitude estimation.

1.1 Main contributions

This report has two main focuses, PARS-aided navigation and spoofing detection. The following contributions are made:

- A GNSS-aided INS for UAV navigation is developed and implemented.
- The INS is adapted for PARS aiding.
- A variant of the CUSUM algorithm is implemented as a residual monitoring technique.

1.2 Outline of the report

Chapter 2 contains an overview over some theoretical concepts, such as reference frames, Global Navigation Satellite System (GNSS) and Phased Array Radio System (PARS). Section 2.2 lists the symbols and operators used in this document. Chapter 3 discusses the implementation of an error-state Multiplicative Extended Kalman Filter (MEKF) and a spoofing detection algorithm. The implementation's performance on a pre-recorded data set is presented in Chapter 4, and Chapter 5 recapitulates details from the results and states some ideas on further work to be done on the subject.

Chapter 2

Background and Preliminaries

2.1 Attitude and quaternions

Every time a door knob is turned, a rotation is performed. The world is full of things that spin and roll, but the theory behind rotations can be quite elusive. Some rotations can be described in a single dimension (around a fixed axis), and these are simple to work with. However, we are often required to describe the rotation of a physical object in global context, and this requires us to use all 3 dimensions. These rotations are more difficult to represent. 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 prominent 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 2.1b), and are converted to a rotation matrix using the following formula:

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

with ϕ, θ, ψ being the roll, pitch and yaw angles, 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 (or shortest "distance") between two rotations, and this is where quaternions come in. Quaternions have taken the world of spatial rotations by storm, despite their reputation of being difficult to understand. A quaternion is a complex number with 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 $\mathbf{q}_v = (b, c, d)$. The remaining coefficient, a, is typically denoted q_w. These two components are combined to form a 4-element vector $\mathbf{q}$:

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

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

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

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

$$|\boldsymbol{q}| = q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1 \tag{2.6}$$

This is called a unit quaternion, and all quaternions discussed in this report fall under this category. Differences between unit quaternions can be calculated in several ways, some listed in Markley (2003). In this report the 4xModified Rodrigues Parameters (MRP) method, derived in Gryte (2020) Appendix B, will be used exclusively. Differences can be injected into quaternions, not through the use of addition, but multiplication:

$$\boldsymbol{q} = \boldsymbol{r} \otimes \delta \boldsymbol{q} \tag{2.7}$$

where $\delta \boldsymbol{q}$ is the error between the quaternions $\boldsymbol{q}$ and $\boldsymbol{r}$. The error quaternion is calculated using the 4xMRP method given in Markley (2003):

$$\delta \boldsymbol{q}(\delta \boldsymbol{\theta}) = \frac{1}{16 + \delta \boldsymbol{\theta}^T \delta \boldsymbol{\theta}} \begin{bmatrix} 16 - \delta \boldsymbol{\theta}^T \delta \boldsymbol{\theta} \\ 8 \delta \boldsymbol{\theta} \end{bmatrix} \tag{2.8}$$

The following formula from Solà (2017) is used to calculate the incremental rotation quaternion from the Angular Rate Sensor (ARS):

$$\boldsymbol{q}(\boldsymbol{\omega}, T_f) = \begin{bmatrix} \cos\left(\frac{|T_f\boldsymbol{\omega}|}{2}\right) \\ \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|}\sin\left(\frac{|T_f\boldsymbol{\omega}|}{2}\right) \end{bmatrix} \tag{2.9}$$

Finally, a rotation matrix can be computed from a quaternion through the following formula (Fossen 2011):

$$\boldsymbol{R}(\boldsymbol{q}) = \boldsymbol{I}_3 + 2q_w S(\boldsymbol{q}_v) + 2S(\boldsymbol{q}_v)^2 \tag{2.10}$$

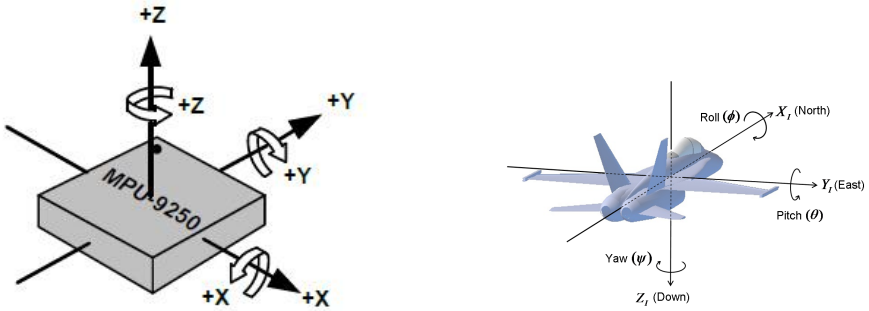
with $S(\cdot)$ denoting the skew symmetric matrix operator:

$$S\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}\right) = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{2.11}$$

2.2 Notation

Table 2.1: Reserved symbols and operators.

Symbol	Explanation
f_f, T_f	Filter rate and period, $f_f = \frac{1}{T_f}$
$\boldsymbol{p}_{bc}^a$	The vector from b to c, decomposed in the $\{a\}$ frame. Used to express positions.
$\boldsymbol{v}_{bc}^a$	The velocity of c relative b, decomposed in the $\{a\}$ frame.
$\boldsymbol{q}_{bc}^a$	A quaternion representation of the rotation of c relative b, decomposed in the $\{a\}$ frame. $q_{bc,w}^a$ is the real/scalar part, $\boldsymbol{q}_{bc,v}^a$ is the imaginary/vector part.
$\boldsymbol{f}_{bc}^a$	The acceleration of c relative b, decomposed in the $\{a\}$ frame.
$\boldsymbol{g}^e$	Gravity vector, decomposed in ECEF.
$\boldsymbol{\omega}_{bc}^a$	The angular rate of c relative b, decomposed in the $\{a\}$ frame.
$\boldsymbol{b}_{acc}^a$, $\boldsymbol{b}_{ars}^a$	Accelerometer and Angular Rate Sensor biases, decomposed in the $\{a\}$ frame.
$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$	Kalman filter system, input and output matrix.
$\boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{P}$	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.
$\bar{x}$	The average of x.
$\boldsymbol{R}_{en}$	A rotation matrix, such that $\boldsymbol{p}_{eb}^e = \boldsymbol{R}_{en}\boldsymbol{p}_{eb}^n$.
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.



(a) An IMU with the m frame drawn in. (Leo Rover Docs 2020)

(b) A fighter jet model with its body axes drawn in. (CH Robotics 2020)

Figure 2.1: Body-centered reference frames.

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:

Measurement frame

The $\{m\}$ (measurement) frame is the frame corresponding to the IMU's raw measurements. Details about the $\{m\}$ frame can be found in the IMU's data sheet, and its relationship to the body frame depends on how and where the unit is mounted. The $\{m\}$ frame is illustrated in Figure 2.1a.

Body frame

The $\{b\}$ (body) frame's 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 deg roll, pitch and yaw, shown in Figure 2.1b. Two different body frames will be used, the $\{b\}$ frame (for the UAV) and the $\{r\}$ frame (for the PARS base station). The term "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.

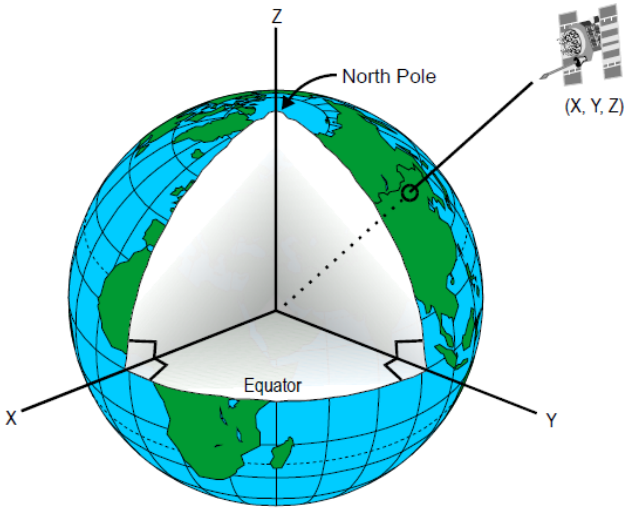


Figure 2.2: The ECEF frame. (Wikipedia 2020)

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 Figure 2.2.

NED frame

The $\{n\}$ (North-East-Down) frame is centered at the IMU, 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.

Geodetic frame

In contrast to the other frames, the geodetic frame is spherical, not Cartesian. Like the ECEF frame, this frame is Earth-Centered and Earth-Fixed, and 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.

ECI frame

The Earth-Centered Inertial (ECI) frame ($\{i\}$ 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.

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:

$$\boldsymbol{R}_{en}(\phi, \lambda) = \begin{bmatrix} -\sin\phi\cos\lambda & -\sin\lambda & -\cos\phi\cos\lambda \\ -\sin\phi\sin\lambda & \cos\lambda & -\cos\phi\sin\lambda \\ \cos\phi & 0 & -\sin\phi \end{bmatrix} \tag{2.12}$$

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 coordinates are not as simple. The steps are found in Fossen (2011), and are revisited below. The ECEF coordinates corresponding to a geodetic position can be calculated directly:

$$x \leftarrow (N + h)\cos(\phi)\cos(\lambda) \tag{2.13a}$$
$$y \leftarrow (N + h)\cos(\phi)\sin(\lambda) \tag{2.13b}$$
$$z \leftarrow \left(\frac{r_p^2}{r_e^2}N + h\right)\sin(\phi) \tag{2.13c}$$

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:

$$N \leftarrow \frac{r_e^2}{\sqrt{r_e^2\cos(\phi)^2 + r_p^2\sin(\phi)^2}} \tag{2.14}$$

The reverse operation is done using an iterative scheme. The first three steps

are done only once:

$$\lambda \leftarrow \arctan\left(\frac{y}{x}\right) \tag{2.15a}$$

$$p \leftarrow \sqrt{x^2 + y^2} \tag{2.15b}$$

$$e \leftarrow \sqrt{1 - (r_p/r_e)^2} \tag{2.15c}$$

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

$$\phi \leftarrow \arctan\left(\frac{z}{p(1-e^2)}\right) \tag{2.16}$$

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.

$$N \leftarrow \frac{r_e^2}{\sqrt{r_e^2 \cos(\phi)^2 + r_p^2 \sin(\phi)^2}} \tag{2.17a}$$

$$h \leftarrow \frac{p}{\cos(\phi)} - N \tag{2.17b}$$

$$\phi_0 \leftarrow \phi \tag{2.17c}$$

$$\phi \leftarrow \arctan\left(\frac{z}{p(1 - e^2 N/(N+h))}\right) \tag{2.17d}$$

$$\eta \leftarrow |\phi - \phi_0| \tag{2.17e}$$

2.4 IMU

An Inertial Measurement Unit (IMU) is often used for navigational purposes, as it supplies information on the sensor's movement in the inertial frame. 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 by the ground. This leads to the addition of the gravity vector in Chapter 3. The accelerometer measurements are

$$\boldsymbol{f}^b = \boldsymbol{f}^b_{nb} + \boldsymbol{b}^b_{acc} + \boldsymbol{\varepsilon}^b_{acc} \tag{2.18}$$

such that $\boldsymbol{\varepsilon}^b_{acc}$ is zero mean Gaussian noise and $\boldsymbol{b}^b_{acc}$ is bias term, treated as a Gauss-Markov model with time constant $\boldsymbol{T}_{acc} = T_{acc}\boldsymbol{I}_3$. Angular Rate Sensors measure the sensor's angular motion in rad/s, and the measurements follow a corresponding model

$$\boldsymbol{\omega}^b = \boldsymbol{\omega}^b_{nb} + \boldsymbol{b}^b_{ars} + \boldsymbol{\varepsilon}^b_{ars} \tag{2.19}$$

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

2.5 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 & Enge (2011) and has several components:

$$\rho = r + c(\delta t_u - \delta t^s) + I_\rho + T_\rho + \varepsilon_\rho \tag{2.20}$$

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 the extreme latitudes. (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.

2.6 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 & 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)

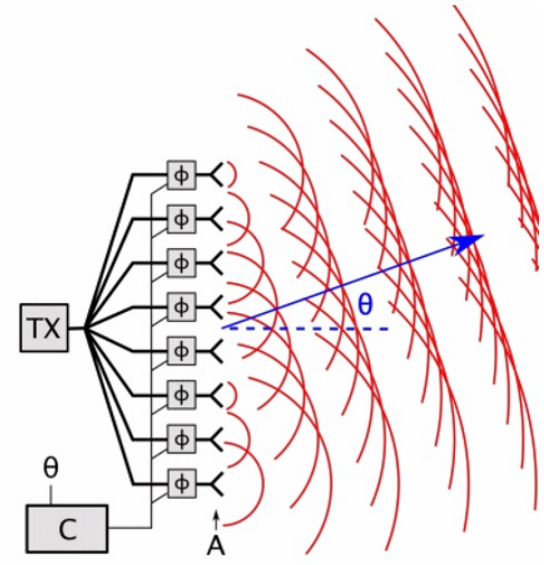


Figure 2.3: A phased array antenna, illustrating its beamsteering capabilities. (Everything RF 2020)

2.7 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 "steer" the signal, 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 & Conway 2015)

The estimated direction can be utilized together with an estimate of the dis-

tance, obtained through methods such as the Round-trip time (RTT). These three values form a spherical coordinate system and are enough to construct an estimate of a remote device's position relative to a base station. Radionor Communications' PARS devices output the distance, elevation and azimuth angle from the base station to the remote device. Hence, for this system to be used for positioning, the base station's position and orientation needs to be known in order to compute the absolute position. For this project the base's roll angle is assumed to be zero, . The vector from the base to the UAV (decomposed in the $\{r\}$ frame) is

$$\mathbf{p}_{rb}^{r} = d \begin{bmatrix} \cos(\Psi)\cos(\alpha) \\ \sin(\Psi)\cos(\alpha) \\ -\sin(\alpha) \end{bmatrix} \tag{2.21}$$

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

$$\boldsymbol{p}_{rb}^{n} = d \begin{bmatrix} \cos(\Psi + \Psi_r)\cos(\alpha + \alpha_r) \\ \sin(\Psi + \Psi_r)\cos(\alpha + \alpha_r) \\ -\sin(\alpha + \alpha_r) \end{bmatrix} \tag{2.22}$$

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:

$$\boldsymbol{p}_{rb}^{n} = \boldsymbol{R}_{nr}(\Phi_r, \alpha_r, \Psi_r)\boldsymbol{p}_{rb}^{r} \tag{2.23}$$

Which relates to the UAV's position:

$$\boldsymbol{p}_{eb}^{e} = \boldsymbol{p}_{er}^{e} + \boldsymbol{R}_{en}(\phi_r, \lambda_r)\boldsymbol{p}_{rb}^{n} \tag{2.24}$$

Note that the base station's position is used when computing $\boldsymbol{R}_{en}$, not the UAV. 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, non-zero roll angle may not always be avoidable. 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 $\mathbf{R}_{pars}$ must be mapped to ECEF coordinates from the spherical measurements. The procedure for this is given in Gryte (2020). Adapted to the ECEF case, the similarity transform is

$$\mathbf{R}_{pars}^{e} = \mathbf{R}_{en}(\phi_r, \lambda_r)\mathbf{M}(\alpha, \Psi, d)\mathbf{R}_{pars}\mathbf{M}^{T}(\alpha, \Psi, d)\mathbf{R}_{ne}(\phi_r, \lambda_r) \tag{2.25}$$

such that

$$\mathbf{M} = \begin{bmatrix} \cos(\Psi)\cos(\alpha) & -d\sin(\Psi)\cos(\alpha) & -d\cos(\Psi)\sin(\alpha) \\ \sin(\Psi)\cos(\alpha) & d\cos(\Psi)\cos(\alpha) & -d\sin(\Psi)\sin(\alpha) \\ -\sin(\alpha) & 0 & -d\cos(\alpha) \end{bmatrix} \tag{2.26}$$

2.8 Change detection

It is often useful to have some kind of metric by which to decide whether an event has occurred or not. This typically falls under the field of Change Detection, and several algorithms (Gustafsson 2001) are available for our application. We would like to decide whether the GNSS receiver is being spoofed or not. Jamming detection would be vastly simpler, since the ultimate goal is loss of lock, but a successful spoofing attack would have the GNSS position and true position drifting apart slowly. Thus, a two-part filter is proposed, with the first part aided by GNSS and the other aided by PARS. A change detection algorithm is applied to the difference between the two filters. In this report, the CUSUM algorithm will be used. A Kalman filter based supervisor module is implemented in Albrektsen et al. (2018*b*) with promising results. The INS in that report is based on separate angular and translational motion observers. Using a Kalman filter instead yields linear motion and attitude estimates without direct attitude sensors from a single module. The spoofing is generated similarly and the same data set is used, so the results from that report will be compared to ours in Chapter 4.

We have a signal (a stochastic process) x, which is generated from a distribution θ_0. We postulate two hypotheses, $\mathcal{H}_0$ (x is generated from θ_0) and $\mathcal{H}_1$ (x is generated from θ_1). For each sample $x[k]$ we calculate

$$s[k] = \ln \frac{p(x[k], \theta_1)}{p(x[k], \theta_0)} \tag{2.27}$$

which, trivially, will be negative if $\mathcal{H}_0$ is true, since the fraction will be less than 1. Conversely, we see that $s[k]$ will be positive if $\mathcal{H}_1$ is true. We keep track of the cumulative sum of s in a separate variable:

$$S[k] = \sum_{i=0}^{k} s[i] \tag{2.28}$$

This process will typically be monotonically decreasing, as long as $\mathcal{H}_0$ is true. We keep track of the current minimum of S, its purpose will be apparent very soon:

$$\hat{n_c} = \arg\min_k S \tag{2.29}$$

Every new sample of S will be the new minimum since S is monotonically decreasing. Hence, $\hat{n_c}$ will only stop changing once we start generating x from the alternative distribution θ_1. This will cause the decision function $G[k]$, which previously has been fairly still around 0, to start rising:

$$G[k] = S[k] - S[\hat{n_c}] \tag{2.30}$$

Once G (now increasing) crosses a certain threshold h, we decide that a change has happened and $\mathcal{H}_1$ is now true. $\hat{n_c}$ gives an estimate for when the change happened, hence the hat. We set $n_d = k$ when G crosses h to indicate the time when the change has been detected.

Chapter 3

Implementation

3.1 Error-state filtering

A top-level view of the algorithm is shown in Figure 3.1, and this chapter will reveal the details of its implementation. The standard Kalman Filter presented in Appendix A is designed for linear systems. In the event of a nonlinear system, an Extended Kalman Filter (EKF) can be used instead. The nominal state propagation equation is altered:

$$\hat{\boldsymbol{x}}_k \leftarrow f(\hat{\boldsymbol{x}}_{k-1}) \tag{3.1}$$

As previously stated, 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). 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 measurement update step also estimates the error states, which are then injected into the nominal states, as shown in Table 3.1. The function $f()$ includes a quaternion product:

$$\hat{\boldsymbol{q}}_k \leftarrow \hat{\boldsymbol{q}}_{k-1} \otimes \boldsymbol{q}'_k \tag{3.2}$$

where $\boldsymbol{q}'_k$ denotes the change in attitude calculated using (2.9).

Table 3.1: The Multiplicative Extended Kalman Filter steps and their equations (autonomous systems).

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

Similarly, the function $g()$ includes:

$$\hat{\boldsymbol{q}}_k \leftarrow \hat{\boldsymbol{q}}_k \otimes \delta \hat{\boldsymbol{q}}_k \tag{3.3}$$

where $\delta \hat{\boldsymbol{q}}_k$ is the error quaternion calculated from the error state vector $\delta \hat{\boldsymbol{x}}_k$, using (2.8). It is found in Gryte (2020) that the error states propagate with the following linearized dynamics:

$$\delta \dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}_c \delta \hat{\boldsymbol{x}} + \boldsymbol{B}_c \boldsymbol{\varepsilon} \tag{3.4}$$

where we have the error state vector and the zero mean Gaussian noise components

$$\delta \hat{\boldsymbol{x}} = \begin{bmatrix} \delta \hat{\boldsymbol{p}}_{eb}^e \\ \delta \hat{\boldsymbol{v}}_{eb}^e \\ \delta \hat{\boldsymbol{\theta}} \\ \delta \hat{\boldsymbol{b}}_{acc}^b \\ \delta \hat{\boldsymbol{b}}_{ars}^b \end{bmatrix} \tag{3.5a}$$

$$\delta \boldsymbol{\varepsilon} = \begin{bmatrix} \boldsymbol{\varepsilon}_{acc}^b \\ \boldsymbol{\varepsilon}_{ars}^b \\ \boldsymbol{\varepsilon}_{b,acc}^b \\ \boldsymbol{\varepsilon}_{b,ars}^b \end{bmatrix} \tag{3.5b}$$

with $\delta\theta$ corresponding to the vehicle's error state attitude, not the error state pitch angle. The equations are given:

$$\delta\dot{\boldsymbol{p}}^e_{eb} = \delta\boldsymbol{v}^e_{eb} \tag{3.6a}$$

$$\delta\dot{\boldsymbol{v}}^e_{eb} = -\boldsymbol{R}_{eb}\boldsymbol{S}(\boldsymbol{f}^b_{nb})\delta\boldsymbol{\theta} - \boldsymbol{R}_{eb}\delta\boldsymbol{b}^b_{acc} - \boldsymbol{R}_{eb}\boldsymbol{\varepsilon}^b_{acc} \tag{3.6b}$$

$$\delta\dot{\boldsymbol{\theta}} = -\boldsymbol{S}(\boldsymbol{\omega}^b_{nb})\delta\boldsymbol{\theta} - \boldsymbol{I}_3\delta\boldsymbol{b}^b_{ars} - \boldsymbol{\varepsilon}^b_{ars} \tag{3.6c}$$

$$\delta\dot{\boldsymbol{b}}^b_{acc} = -\boldsymbol{T}^{-1}_{acc}\delta\boldsymbol{b}^b_{acc} + \boldsymbol{I}_3\boldsymbol{\varepsilon}^b_{b,acc} \tag{3.6d}$$

$$\delta\dot{\boldsymbol{b}}^b_{ars} = -\boldsymbol{T}^{-1}_{ars}\delta\boldsymbol{b}^b_{ars} + \boldsymbol{I}_3\boldsymbol{\varepsilon}^b_{b,ars} \tag{3.6e}$$

Corresponding to the following matrices:

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

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

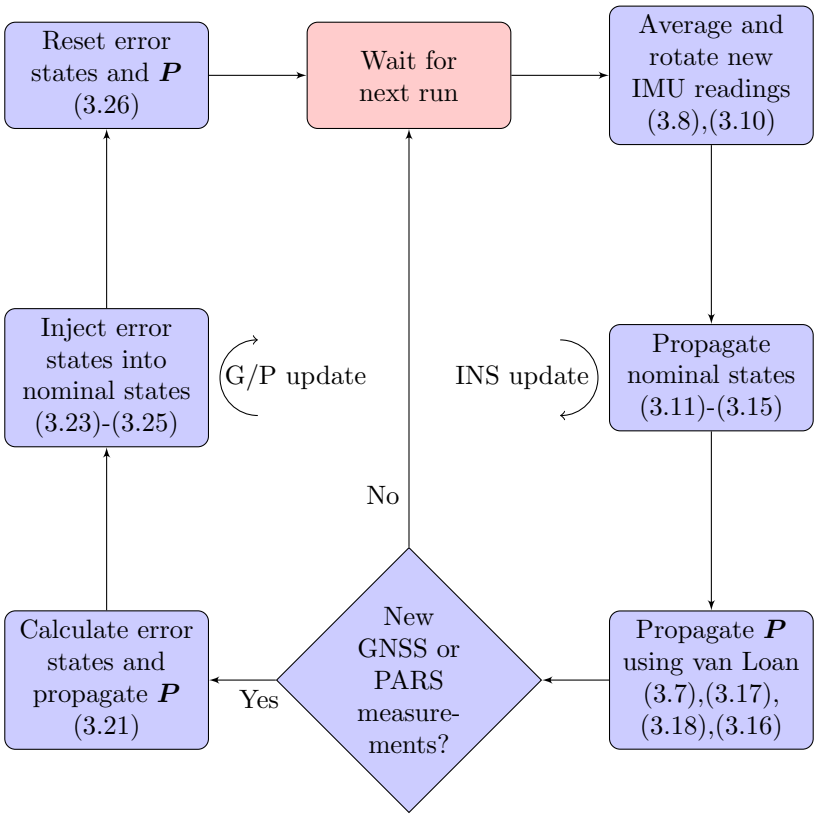


Figure 3.1: The entire filter, with equation references, summarized in a flowchart. "G/P" is short for "GNSS/PARS". The "Wait for next run" block is painted pink because it is a natural starting point for the algorithm.

3.2 INS update

On arrival of a new IMU measurement, we have two inputs, the raw gyroscope and accelerometer measurements. The IMU rate might be higher than the filter rate, in which case the measurements that have arrived since last run are simply averaged:

$$\begin{bmatrix} \bar{\boldsymbol{f}}_{nb}^{m} \\ \bar{\boldsymbol{\omega}}_{nb}^{m} \end{bmatrix} = \frac{1}{n} \sum_{i=1}^{n} \begin{bmatrix} \boldsymbol{f}_{nb}^{m} \\ \boldsymbol{\omega}_{nb}^{m} \end{bmatrix}_i \tag{3.8}$$

Note that this is a simplification, an optimal system would perform compensation for coning, sculling and scrolling motion. (Brouk 2019) Using the theory from Section 2.4, an estimate of the vehicle's motion can be calculated. The measurements are decomposed in the $\{m\}$ frame, and must be converted to the body frame before anything else is done:

$$\boldsymbol{f}_{nb}^{b} = \boldsymbol{R}_{bm} \boldsymbol{f}_{nb}^{m} \tag{3.9a}$$

$$\boldsymbol{\omega}_{nb}^{b} = \boldsymbol{R}_{bm} \boldsymbol{\omega}_{nb}^{m} \tag{3.9b}$$

Where $\boldsymbol{R}_{bm}$ is the rotation matrix from the m-frame to body frame. This matrix depends entirely on the orientation and position of the IMU on the vehicle. In practice, the bias must be subtracted:

$$\hat{\boldsymbol{f}}_{nb}^{b} \leftarrow \boldsymbol{R}_{bm} \ \boldsymbol{f}_{nb}^{m} - \hat{\boldsymbol{b}}_{acc}^{b} \tag{3.10a}$$

$$\hat{\boldsymbol{\omega}}_{nb}^{b} \leftarrow \boldsymbol{R}_{bm} \ \hat{\boldsymbol{\omega}}_{nb}^{m} - \hat{\boldsymbol{b}}_{ars}^{g} \tag{3.10b}$$

Calculate acceleration in ECEF frame:

$$\hat{\boldsymbol{f}}_{nb}^{e} \leftarrow \boldsymbol{R}_{eb}(\hat{\boldsymbol{q}}_{eb}^{e}) \hat{\boldsymbol{f}}_{nb}^{b} + \boldsymbol{g}^{e} \tag{3.11}$$

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.

$$\boldsymbol{\gamma}_{eb}^{e} = -\frac{\mu}{|\boldsymbol{p}_{eb}^{e}|^3} \left(\boldsymbol{p}_{eb}^{e} + \frac{3}{2} J_2 \frac{R_0^2}{|\boldsymbol{p}_{eb}^{e}|^2} \begin{bmatrix} (1 - 5(p_{eb,z}^{e} / |\boldsymbol{p}_{eb}^{e}|)^2) p_{eb,x}^{e} \\ (1 - 5(p_{eb,z}^{e} / |\boldsymbol{p}_{eb}^{e}|)^2) p_{eb,y}^{e} \\ (1 - 5(p_{eb,z}^{e} / |\boldsymbol{p}_{eb}^{e}|)^2) p_{eb,z}^{e} \end{bmatrix} \right) \tag{3.12a}$$

$$\boldsymbol{g}^{e} = \boldsymbol{\gamma}_{eb}^{e} + \omega_{ie}^{2} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{p}_{eb}^{e} \tag{3.12b}$$

with R_0 = 6378137 m, μ = 3.986004418e14 m^3s^{-2}, J_2 = 1.082627e-3 m^3s^{-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:

$$\hat{\boldsymbol{p}}^e_{eb} \leftarrow \boldsymbol{p}^e_{eb} + T_f \hat{\boldsymbol{v}}^e_{eb} + \frac{T_f^2}{2} \hat{\boldsymbol{f}}^e_{nb} \tag{3.13a}$$

$$\hat{\boldsymbol{v}}^e_{eb} \leftarrow \hat{\boldsymbol{v}}^e_{eb} + T_f \hat{\boldsymbol{f}}^e_{nb} \tag{3.13b}$$

We insert the estimated angular rate into (2.9):

$$\hat{\boldsymbol{q}}^e_{eb} \leftarrow \hat{\boldsymbol{q}}^e_{eb} \otimes \begin{bmatrix} \cos\left(\frac{|T_f \hat{\boldsymbol{\omega}}^b_{nb}|}{2}\right) \\ \frac{\hat{\boldsymbol{\omega}}^b_{nb}}{|\hat{\boldsymbol{\omega}}^b_{nb}|} \sin\left(\frac{|T_f \hat{\boldsymbol{\omega}}^b_{nb}|}{2}\right) \end{bmatrix} \tag{3.14}$$

The bias estimates are updated using a first order approximation:

$$\hat{\boldsymbol{b}}^b_{acc} \leftarrow (\boldsymbol{I}_3 - T_f \boldsymbol{T}^{-1}_{acc}) \hat{\boldsymbol{b}}^b_{acc} \tag{3.15a}$$

$$\hat{\boldsymbol{b}}^b_{ars} \leftarrow (\boldsymbol{I}_3 - T_f \boldsymbol{T}^{-1}_{gyro}) \hat{\boldsymbol{b}}^b_{ars} \tag{3.15b}$$

Now that we have propagated the nominal states, we have to update $\boldsymbol{P}$:

$$\boldsymbol{P}^- \leftarrow \boldsymbol{A}_d \boldsymbol{P}^+ \boldsymbol{A}_d^T + \boldsymbol{Q}_d \tag{3.16}$$

Neither $\boldsymbol{A}_d$ nor $\boldsymbol{Q}_d$ are defined yet, they must be obtained by discretizing their continuous counterparts. There are several ways to do this, and van Loan's method (Loan 1978) is a popular alternative for time-varying systems. We begin by inserting the matrices $\boldsymbol{A}_c$ and $\boldsymbol{B}_c$ from (3.7) into a matrix $\boldsymbol{F}$. Additionally, the predefined process noise matrix $\boldsymbol{Q}_c$ is needed:

$$\boldsymbol{F} \leftarrow \begin{bmatrix} -\boldsymbol{A}_c & \boldsymbol{B}_c \boldsymbol{Q}_c \boldsymbol{B}_c^T \\ \boldsymbol{0} & \boldsymbol{A}_c^T \end{bmatrix} T_f \tag{3.17}$$

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

$$e^{\boldsymbol{F}} \leftarrow \boldsymbol{I} + \boldsymbol{F} + \frac{\boldsymbol{F}^2}{2!} + \frac{\boldsymbol{F}^3}{3!} + ... \tag{3.18}$$

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

$$e^{\boldsymbol{F}} = \begin{bmatrix} \cdot & \boldsymbol{A}_d^{-1} \boldsymbol{Q}_d \\ \boldsymbol{0} & \boldsymbol{A}_d^T \end{bmatrix} \tag{3.19}$$

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

$$\boldsymbol{Q}_d \leftarrow \boldsymbol{A}_d \boldsymbol{A}_d^{-1} \boldsymbol{Q}_d \tag{3.20}$$

3.3 GNSS/PARS update

Update Kalman gain, error state and covariance matrix:

$$\boldsymbol{K} = \boldsymbol{P}^- \boldsymbol{C}^T (\boldsymbol{C}\boldsymbol{P}^- \boldsymbol{C}^T + \boldsymbol{R})^{-1} \tag{3.21a}$$

$$\delta\hat{\boldsymbol{x}} = \boldsymbol{K}(\boldsymbol{y}_{eb}^e - \hat{\boldsymbol{p}}_{eb}^e) \tag{3.21b}$$

$$\boldsymbol{P}^+ = (\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{C})\boldsymbol{P}^- (\boldsymbol{I}_n - \boldsymbol{K}\boldsymbol{C})^T + \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^T \tag{3.21c}$$

Where $\boldsymbol{R}$ is equal to $\boldsymbol{R}_{gnss}$ or $\boldsymbol{R}_{pars}$, depending on which sensor the measurement originates from. Note that $\boldsymbol{R}_{pars}$ will have to be calculated each time it is used, given by (2.25). At this point, $\delta\hat{\boldsymbol{x}}$ is a 15-vector consisting of the following 3-vectors:

$$\delta\hat{\boldsymbol{x}} = \begin{bmatrix} \delta\hat{\boldsymbol{p}}_{eb}^e \\ \delta\hat{\boldsymbol{v}}_{eb}^e \\ \delta\hat{\boldsymbol{\theta}} \\ \delta\hat{\boldsymbol{b}}_{acc}^b \\ \delta\hat{\boldsymbol{b}}_{ars}^b \end{bmatrix} \tag{3.22}$$

Update nominal position and velocity estimate:

$$\hat{\boldsymbol{p}}_{eb}^e \leftarrow \hat{\boldsymbol{p}}_{eb}^e + \delta\hat{\boldsymbol{p}}_{eb}^e \tag{3.23a}$$

$$\hat{\boldsymbol{v}}_{eb}^e \leftarrow \hat{\boldsymbol{v}}_{eb}^e + \delta\hat{\boldsymbol{v}}_{eb}^e \tag{3.23b}$$

The attitude estimate is corrected, using (2.8):

$$\delta\hat{\boldsymbol{q}}_{eb}^e(\delta\hat{\boldsymbol{\theta}}) = \frac{1}{16 + \delta\hat{\boldsymbol{\theta}}^T\delta\hat{\boldsymbol{\theta}}} \begin{bmatrix} 16 - \delta\hat{\boldsymbol{\theta}}^T\delta\hat{\boldsymbol{\theta}} \\ 8\delta\hat{\boldsymbol{\theta}} \end{bmatrix} \tag{3.24a}$$

$$\hat{\boldsymbol{q}}_{eb}^e \leftarrow \hat{\boldsymbol{q}}_{eb}^e \otimes \delta\hat{\boldsymbol{q}}_{eb}^e(\delta\hat{\boldsymbol{\theta}}) \tag{3.24b}$$

Note that the error quaternion $\delta\hat{\boldsymbol{q}}_{ib}^e$ 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:

$$\boldsymbol{b}_{acc}^b \leftarrow \boldsymbol{b}_{acc}^b + \delta\boldsymbol{b}_{acc}^b \tag{3.25a}$$

$$\boldsymbol{b}_{ars}^b \leftarrow \boldsymbol{b}_{ars}^b + \delta\boldsymbol{b}_{ars}^b \tag{3.25b}$$

The error state vector and covariance matrix are reset:

$$\delta\hat{\boldsymbol{x}} \leftarrow \mathbf{0} \tag{3.26a}$$

$$\boldsymbol{P}^{+} \leftarrow \boldsymbol{G}(\delta\hat{\boldsymbol{q}}^{e}_{eb})\boldsymbol{P}^{+}\boldsymbol{G}^{T}(\delta\hat{\boldsymbol{q}}^{e}_{eb}) \tag{3.26b}$$

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

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \boldsymbol{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \delta\hat{q}^{e}_{eb,w}\boldsymbol{I}_3 - \boldsymbol{S}(\delta\hat{\boldsymbol{q}}^{e}_{eb,v}) & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \boldsymbol{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \boldsymbol{I}_3 \end{bmatrix} \tag{3.27}$$

This concludes the measurement update step of the (error state) Multiplicative Extended Kalman Filter (MEKF).

3.4 Spoofing detection

In order to detect spoofing, two instances of the filter are run in parallel. One is GNSS-aided, the other PARS-aided. The IMU readings are fed to both filters, and the distance (norm) between the GNSS-aided and the PARS-aided position estimate is calculated. In the event of spoofing, this distance will increase, and this can be detected in a variety of ways. A simple distance threshold is deemed too primitive, and we've opted for a recursive one-sided flavor of the CUSUM algorithm. (Granjon 2013) The CUSUM algorithm requires the following design parameters:

- $\mathcal{H}_0$ - The null hypothesis, i.e. the distribution from which we expect our signal x to be drawn.
- $\mathcal{H}_1$ - The alternative hypothesis, i.e. the distribution from which we expect our signal to be drawn once the change happens.
- h - The decision threshold.

The decision threshold has great significance for the sensitivity of the algorithm. A small h will quickly respond to changes, but is more likely to report false positives. A large h will be less likely to report false positives, but will have a longer delay from the change to the decision and it will be prone to overlook intermittent changes. The implementation used in this report assumes that the change happens in the form of a step in the mean, and thus we are left with the following parameters:

- $\hat{\sigma}$ - The guessed standard deviation of the norm. (Part of the null hypothesis)

- $\hat{\mu}$ - The guessed mean of the norm. (Part of the null hypothesis)
- δ - The new mean after the change has happened. (Part of the alternative hypothesis)
- h - The decision threshold.

The following calculations are made each time step:

$$s = \frac{\delta}{\hat{\sigma}^2}(x - \hat{\mu} - \frac{\delta}{2}) \tag{3.28a}$$

$$S[k] = S[k-1] + s \tag{3.28b}$$

$$G[k] = \max(G[k-1] + s, 0) \tag{3.28c}$$

$$\hat{n}_c = \arg\min_k S \tag{3.28d}$$

where x is the distance between the estimates, assumed to be from a Gaussian distribution with mean $\hat{\mu}$ and variance $\hat{\sigma}^2$. δ represents the mean of the alternative distribution, and thus functions as a kind of threshold parameter. Spoofing is detected once $G[k]$ exceeds h, which is also a threshold parameter. δ decides how far away from zero the measurement can be before it is deemed suspicious, and h controls how long the algorithm will accept suspicious measurements before sounding the alarm.

Chapter 4

Results

The test set was captured during a 40 minute flight in Vassbygda in Trøndelag, Norway. The PARS data was generated using a CRE2-189 ground antenna and a CRE2-144 onboard the UAV, both designed and manufactured by Radionor Communications AS. The CRE2-189 is a directional antenna, and the UAV was outside the antenna's sector for large portions of the first 500 seconds. Consequently, the PARS data from this period has been discarded and replaced by GNSS measurements. The UAV also exits the sector at the end of the flight, so the data is cut off at exactly 35 minutes. The flight was performed using a Skywalker X8 (Skywalker 2020), a fixed wing First Person View (FPV) drone. It was controlled by a Pixhawk 1/ArduPlane (Pixhawk 2020) autopilot with a separate GNSS receiver. The remaining sensors were a u-blox m8t-neo u-blox (2020) GNSS receiver and a Sensonor STIM300 (Sensonor 2020) IMU. Some of the tuning parameters, such as the $\boldsymbol{Q}_c$ matrix, will be equal to the ones found in Gryte (2020), while other parameters, such as the $\boldsymbol{R}$ matrices and $\boldsymbol{P}$ are found through tuning. The GNSS measurements are decomposed in ECEF, as mentioned in Section 2.5. This also means that the GNSS measurement noise matrix is constant, in contrast to the PARS noise matrix, which needs to be recalculated for each measurement. Statistics pertaining to their performance will be presented at the end of Section 4.1. The values of the process and measurement noise matrices are shown in Table 4.1, and the values of the initial covariance matrix $\boldsymbol{P}_0$ are shown in Table 4.2. Note that two separate filters are run in parallel, one receives corrections only from GNSS, the other only from PARS. The matrices are given:

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

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

Parameter	**Value**	**Parameter**	**Value**
σ_{acc}	2.57e-2 m/s/$\sqrt{\text{s}}$	σ_d	10 m
σ_{ars}	9.59e-4 rad/$\sqrt{\text{s}}$	σ_α	0.1 deg
$\sigma_{b,acc}$	2.55e-4 m/s$^{5/2}$	σ_Ψ	0.1 deg
$\sigma_{b,ars}$	6.29e-8 rad/s$^{3/2}$	σ_{gnss}	1 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

$$\boldsymbol{R}_{gnss} = \begin{bmatrix} \sigma^2_{gnss} & 0 & 0 \\ 0 & \sigma^2_{gnss} & 0 \\ 0 & 0 & \sigma^2_{gnss} \end{bmatrix} \tag{4.1b}$$

$$\boldsymbol{R}^s_{pars} = \begin{bmatrix} \sigma^2_d & 0 & 0 \\ 0 & \sigma^2_\alpha & 0 \\ 0 & 0 & \sigma^2_\Psi \end{bmatrix} \tag{4.1c}$$

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

4.1 Spoofing disabled

Figure 4.1 shows the horizontal geodetic position of the GNSS aided estimates and raw GNSS measurements, and we can see that the estimates follow the measurements well. This is expected, since there is little noise in the GNSS measurements. Figure 4.2 is more indicative of the filter's performance, showing the noisy nature of the PARS measurements. The estimates still follow the path, with some small deviations. Some of the measured positions are far away from the path, and would likely be rejected by an outlier rejection mechanism. The filter is reluctant to transition from the loop to the straight line path at the western end of the trajectory, causing it to display a longer turn. The eastern end of the figure also shows the edges of the base station's sensor, forming a 90 degree angle. As mentioned, the PARS measurements from this part of the run is replaced with GNSS data. Moving on to the attitude, Figure 4.3 shows the roll, pitch and yaw estimates (described in Section 2.1) for both the GNSS and PARS aided filters. The reference is sampled from the UAV's Pixhawk autopilot software, and both filters seem to follow the GNSS measurements and Pixhawk attitude estimates well, especially considering the fact that none of the measurements include direct information about the vehicle's attitude.

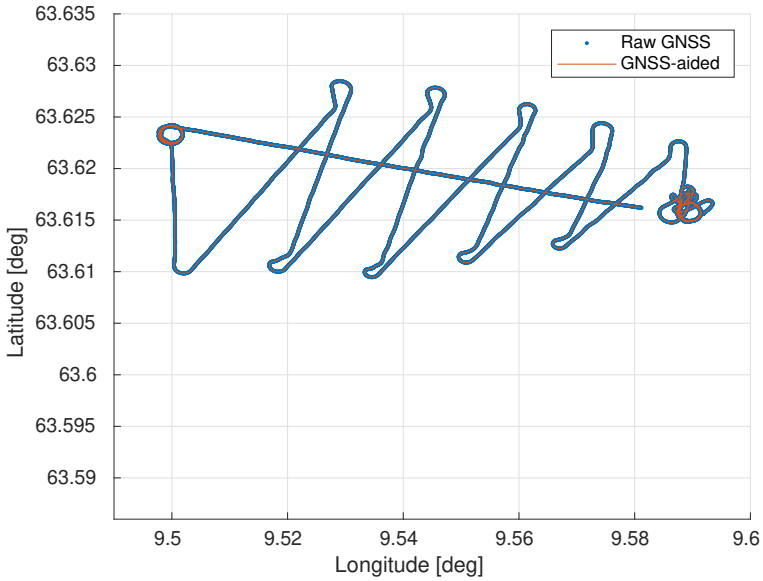


Figure 4.1: Raw GNSS measurements with horizontal position estimates.

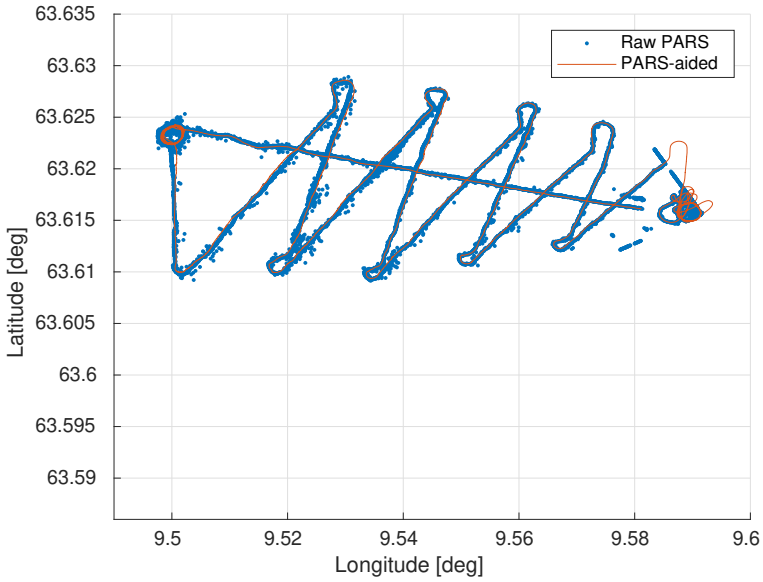


Figure 4.2: Raw PARS measurements with horizontal position estimates.

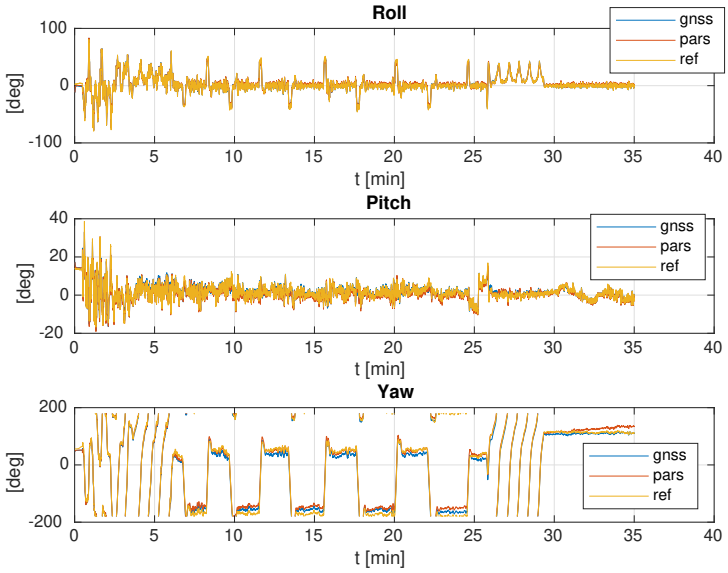
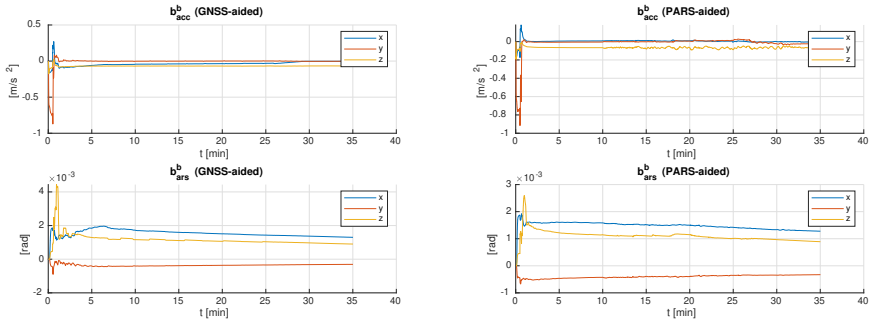


Figure 4.3: Estimated orientation, relative to NED.

Figure 4.4 shows the IMU bias estimates. Their initial values are all zero, and they all move quickly away from the origin before settling down on a slowly moving value. The initial spike is expected, and the fact that they converge to a slowly moving value is a good sign. It means that the filter has found a series of states where each new measurement corresponds well to the prediction. Figure 4.5 shows the error states, i.e. the 9 first elements of the $\delta\hat{\boldsymbol{x}}$ vector, for the GNSS-aided filter. In addition, the standard deviation (computed from the $\boldsymbol{P}$ matrix) multiplied by a factor of three has been drawn in to illustrate the consistency of the filter. The error lies well within the bounds at all times, as is to be expected.



(a) GNSS-aided accelerometer and ARS bias estimates.

(b) PARS-aided accelerometer and ARS bias estimates.

Figure 4.4: Bias estimates.

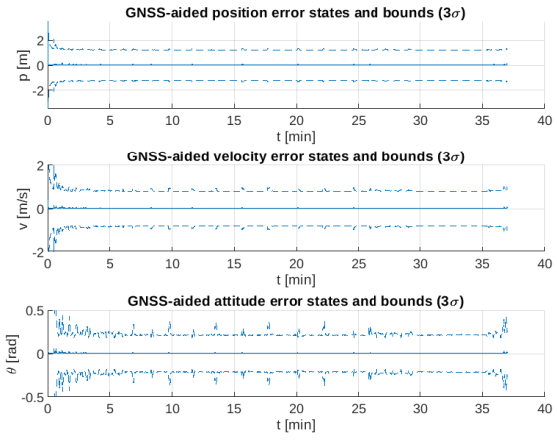


Figure 4.5: GNSS-aided position error states with 3σ bounds. p and v are both decomposed in ECEF, while the attitude is given in the body frame.

Figure 4.6 shows the error states for the PARS-aided filter, and it is simple to see that the estimation errors and uncertainty estimates are higher. These calculations are done at each measurement update, before the error state vector is injected into the nominal estimates. The filter still keeps the errors inside the bounds, save for a few spikes. We can also see the point where the PARS data starts and the GNSS data ends, at 10 minutes. The error bounds widen immediately.

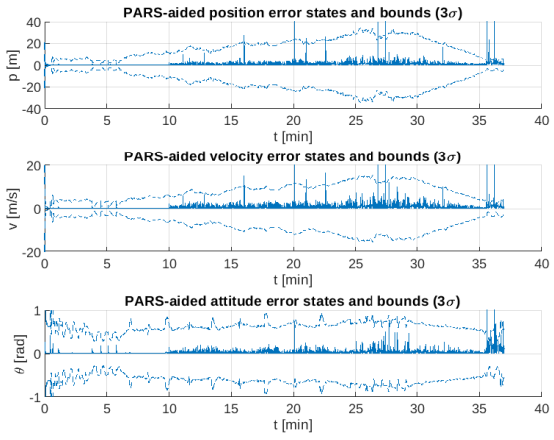


Figure 4.6: PARS-aided position error states with 3σ bounds. p and v are both decomposed in ECEF, while the attitude is given in the body frame.

The results seem reasonable so far, with the filter producing consistent estimates and the two estimated paths looking similar. However, the erraticity and slight positive offset of the PARS-based velocity, shown in Figure 4.7, suggest that there might be a larger discrepancy between the estimates than what has been shown so far. There is one dimension that has not been shown yet: the height. The position estimates so far have shown the 2D positions in latitude and longitude, but the height has been omitted for clarity. Looking at Figure 4.8, large errors in the PARS measurements are presented. These lead to errors in the estimates, and the fact that the PARS-aided velocity estimates are too high now makes sense. Figure 4.9 shows how the full PARS-aided track is affected by erroneous measurements. These erroneous measurements has great effect on the filter, and this is reflected in the position and velocity estimates. Since the problems appear to be isolated to the elevation angle (and partially to the distance measurement), it is possible to discard that value entirely and replace it with a height sensor, such as a barometer. This transforms the PARS measurements from spherical coordinates to cylindrical coordinates, but the two other dimensions (azimuth, distance) can be kept. Gryte (2020) uses an outlier rejection algorithm, and this would likely improve the estimates without adding another sensor. Some of the PARS measurements are several hundred meters off, and these errors would be rejected by lenient rejection mechanisms. A dangerous side effect from introducing outlier rejection is that the mechanism does not inherently know which measurements are outliers and which are not. Therefore care must be taken so that the algorithm does not reject "correct" values.

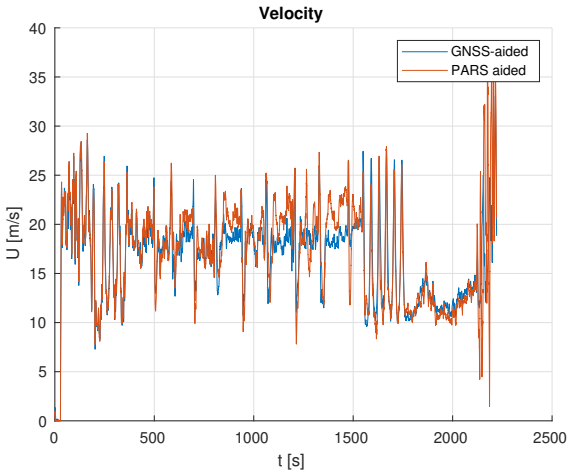


Figure 4.7: Velocity estimates.

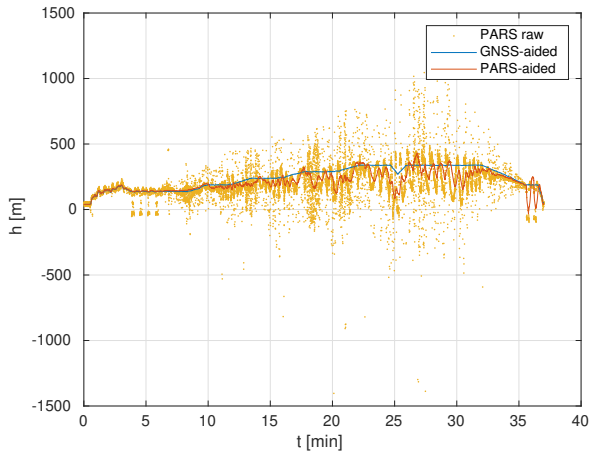


Figure 4.8: Height estimates with raw PARS data.

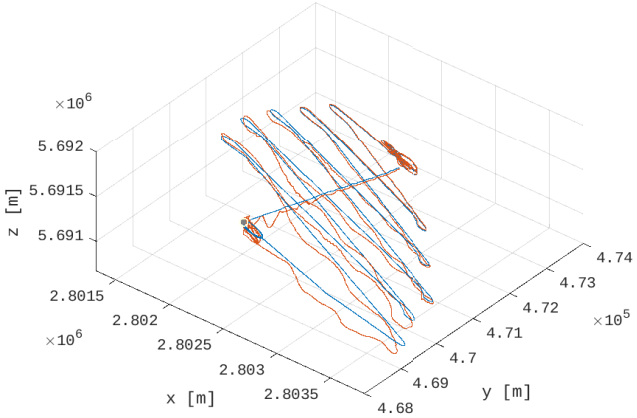


Figure 4.9: Position estimates in three dimensions (ECEF). Seen from the west to illustrate the PARS-aided estimates' (red) deviations from the GNSS-aided estimates (blue).

Before the statistics are presented, the initial states are discussed. So far, an approximately correct unit quaternion ($\boldsymbol{q}^e_{eb,0} = \begin{bmatrix}0.30 & -0.36 & -0.88 & -0.03\end{bmatrix}^T$) has been used as the initial attitude for both filters. Setting this to a wildly incorrect value, such as $\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^T$, results in the filters believing that the UAV is flying upside down ($\pm$ 180 deg roll angle), shown in Figure 4.10. The yaw estimates are also moving in the opposite direction of the Pixhawk reference (Figure 4.11), and this makes sense given that the estimates are upside down. The reason for this is difficult to ascertain, but the error dynamics are linearized around an attitude error of zero, and an upside down initial state is close to the greatest attitude error possible. This has implications for the validity of the modelled dynamics, but one would think that the estimates should be perturbed in the direction of the true states at some point. The fact that the estimate stays upside down during the entire run means that the accelerometer z axis bias estimate has converged to a large value and gotten stuck in some false equilibrium. This value must be large enough to cancel (and in fact, directly contradict) gravity's effects on the accelerometer. Figure 4.12 confirms this theory, with $\boldsymbol{b}^b_{acc,z} \approx -20$ m/s^2. This figure also shows that the bias jumps to -15 m/s^2 almost instantly, hinting that lowering the initial accelerometer bias covariance ($\boldsymbol{P}_{\boldsymbol{b}^b_{acc},0}$) might fix the problem. This means raising the confidence in the initial bias estimates, which are all set to zero initially. Figure 4.13 illustrates the attitude estimates after setting $\boldsymbol{P}_{\boldsymbol{b}^b_{acc},0} = 0.01\boldsymbol{I}_3$, i.e. a reduction by a factor of 100, which resolves the issue.

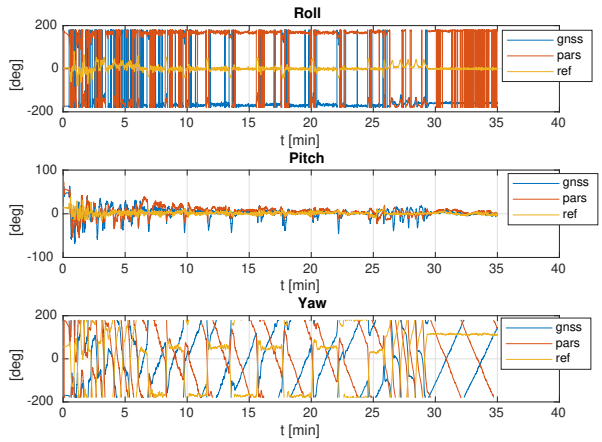


Figure 4.10: Roll, pitch and yaw estimates with incorrect initial attitude. The estimates are upside down and the yaw angle is drifting aimlessly.

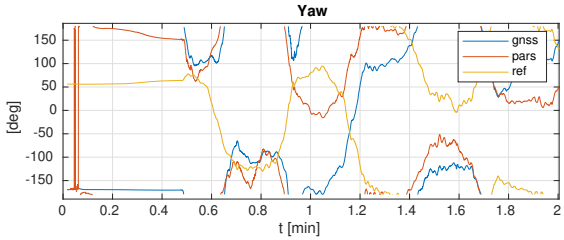


Figure 4.11: Yaw angle estimates with incorrect initial attitude, zoomed in on the first two minutes.

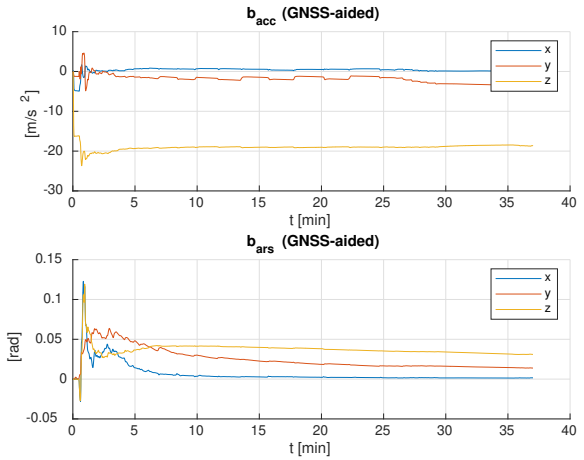


Figure 4.12: The GNSS-aided IMU bias estimates with incorrect initial attitude. The (absolute value of the) z-component is too large (-20), effectively corrupting the IMU readings.

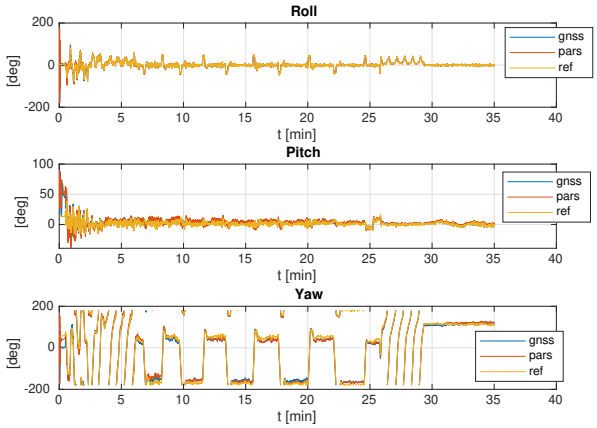


Figure 4.13: Improved roll, pitch and yaw estimates after $\boldsymbol{P}_{b,acc}$ has been reduced from 1 to 0.01 $\mathrm{m}^2/\mathrm{s}^4$.

Finally, statistics are presented. Tables 4.3 and 4.4 show the Mean Error (ME), Mean Average Error (MAE) and Root Mean Squared Error (RMSE) for the PARS-aided position and attitude, respectively. The errors from which the statistics are calculated are gathered at the arrival of each PARS measurement, where the attitude and position estimates *before* the error-state injections are compared to the RTK position and attitude from the Pixhawk reference. Compared to Gryte (2020), the errors are larger, especially the position. This is likely due to how the PARS measurements are treated before being fed to the filter. Gryte's implementation includes an outlier rejection mechanism and PARS' elevation measurements are discarded in favor of a separate altitude reading. The Cartesian error is also dominated by the z-component and (to a lesser extent) the x-component, which is no surprise. Figure 4.8 shows that some of the PARS measurements lie significantly below the true trajectory, and this is reflected in the error statistics since the negative Down vector at these latitudes corresponds to a z-dominated ECEF vector. As mentioned previously, an alternative altitude sensor such as a barometer is believed to greatly improve these estimates. An alternative solution would be to implement some form of outlier rejection, as done in Gryte (2020). However, this requires careful design, as the results could be catastrophic if the algorithm should start rejecting the correct measurements in favor of readings that are based on reflections, for example.

Table 4.3: PARS-aided position error statistics.

Metric	x **[m]**	y **[m]**	z **[m]**	**Norm [m]**
ME	-9.66	-1.54	-28.39	30.04
MAE	15.21	11.44	34.96	39.81
RMSE	23.24	18.06	52.55	60.23

Table 4.4: PARS-aided attitude error statistics.

Metric	**Roll [deg]**	**Pitch [deg]**	**Yaw [deg]**	**Norm [deg]**
ME	0.96	-0.35	6.68	6.76
MAE	1.58	0.95	10.53	10.69
RMSE	2.01	1.45	14.14	14.36

4.2 Spoofing enabled

The spoofing detection mechanism in Section 3.4 is utilized to detect spoofing. The spoofing is simulated by generating a displacement term to the GNSS measurements, increasing by 10 cm in positive x direction each time update. This corresponds to a displacement of 1 m/s. Note that this displacement is not introduced until after the UAV has entered the PARS station's sector, i.e. 800 seconds or 13 minutes and 20 seconds. To begin with, the values $h = 1000000$, $\delta = 200$, $\hat{\sigma} = 10$, $\hat{\mu} = 0$ are chosen. This does not work well, because we get a false positive (shown in Figure 4.14) at about 20 minutes. Increasing $\hat{\sigma}$ to 25 is sufficient to get the algorithm working with this data set, but it is preferable with a method that adapts to the application. In order to correct this, the following values are chosen:

$$\hat{\sigma} = \hat{\mu} = |\text{diag}(P_{p,gnss})| + |\text{diag}(P_{p,pars})| \tag{4.2a}$$

$$\delta = 2\hat{\mu} \tag{4.2b}$$

$$h = 1000000 \qquad \text{(unchanged)} \tag{4.2c}$$

$$\tag{4.2d}$$

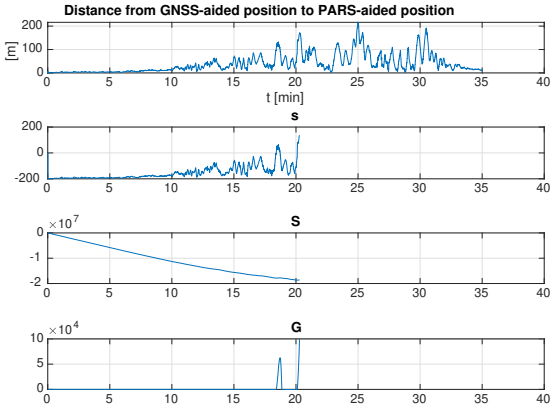


Figure 4.14: No spoofing, but false positive. $\delta = 200$, $\sigma = 10$, $\hat{\mu} = 0$

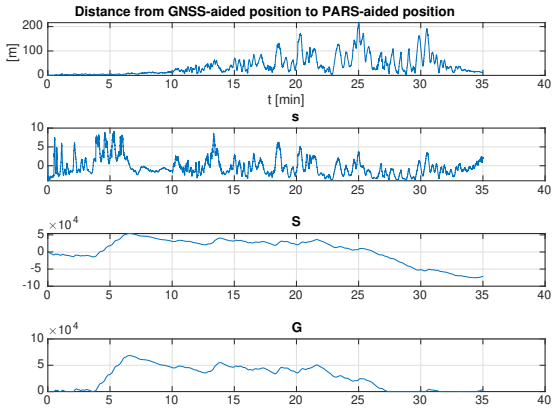


Figure 4.15: No spoofing, no false positive. $\delta = 2\hat{\sigma}$, $\hat{\sigma} = \hat{\mu}$, $\hat{\mu} = |\text{diag}(P_{p,gnss})| + |\text{diag}(P_{p,pars})|$

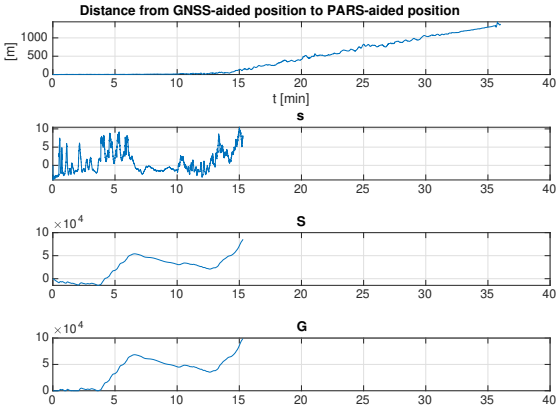


Figure 4.16: Spoofing activated and detected. Same parameters as Figure 4.15.

The results are shown in Figure 4.15. We can see that the G function stays below the limit, and no false positives are reported. Activating the spoofing, we see that the spoofing is discovered fairly quickly, 1 minute and 57 seconds after the spoofing starts. At this point, the GNSS-aided estimate is displaced by 117 meters. This is significantly slower than the Kalman filter implemented in Albrektsen et al. (2018*b*), which detected the spoofing after a displacement of 36 m (at the same displacement rate). That implementation also used a separate altitude sensor in order to discard the elevation angle measurement, yielding smaller PARS errors. The CUSUM implementation should be tested with a similar setup in order to compare the two methods properly. It is likely that a slightly tighter limit (smaller δ, $\hat{\sigma}$, $\hat{\mu}$, h) would be possible for this data set, but it is not advisable given the large elevation angle errors in the PARS measurements and the lack of outlier rejection. There is a trade-off between false positive risk and detection speed, a tight limit is quicker to react and more sensitive, but higher sensitivity also increases the risk of overreaction from the detection mechanism. To give a clearer picture of the results, Figure 4.17 shows the successful spoofing detection in geodetic coordinates. As mentioned in Section 2.6, residual monitoring is only one of the ways to detect spoofing, and several more signal-focused methods exist.

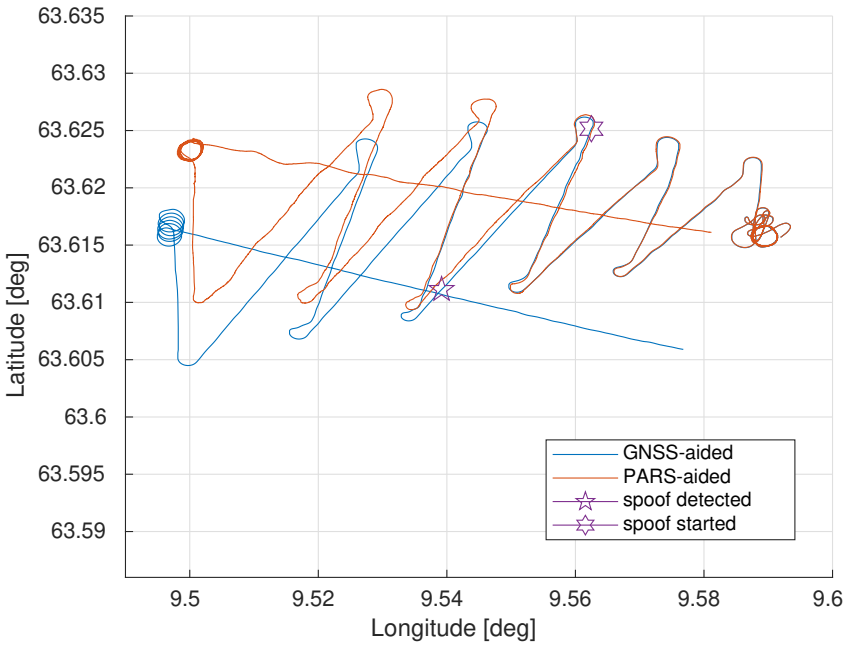


Figure 4.17: Position estimates with added 1 m/s GNSS drift after 800 seconds. The spoofing is detected after 117 seconds.

Chapter 5

Conclusion and Further Work

In this report, the viability of a PARS-aided MEKF is evaluated. The purpose of such a navigation system is as a stand-in for GNSS during long-range UAV flights. PARS requires precise setup and calibration, but the results are promising. The system is able to produce a reasonable estimate of the position, velocity and attitude, together with the IMU biases. This is done even without any prior knowledge about the vehicle's attitude. However, this is sensitive to moderate increases in the initial $\boldsymbol{P}_0$ matrix's accelerometer bias elements. The PARS measurements have a fair bit of noise, mostly because of errors in the reported elevation angle. This has great effect on the estimates, but the azimuth measurements remain acceptable. The error statistics are a little larger than in Gryte (2020), but this is likely due to the lack of outlier rejection. Gryte also uses a separate altitude sensor, effectively discarding the PARS elevation measurement.

A simple implementation of the CUSUM algorithm is implemented, and applied to the residual between the GNSS-aided and PARS-aided position estimates. The tuning parameters are chosen as a function of the $\boldsymbol{P}$ matrices, resulting in an implementation that adapts to the filter's error characteristics. The spoofing detection algorithm is demonstrated, with both appropriate and inappropriate parameters. The algorithm detects a spoofing displacement of 1 m/s within two minutes. A tighter limit could likely be placed on the algorithm, but the number and magnitude of elevation errors in the PARS measurements make it difficult to lower the bounds without introducing false positives. Conclusively, the use of PARS as an alternative to GNSS in GNSS-denied environments is a promising future solution for enabling long-range UAV flights. For areas where GNSS is available, a simple CUSUM-based change detection mechanism can be employed to monitor the integrity of the GNSS fix. The following ideas are

suggested for further work on this topic:

- An altitude sensor such as a barometer can be added to the UAV, giving the opportunity to correct (or discard) the PARS elevation measurements.
- The vehicle's initial attitude is unknown, but it can be estimated using the global position if we assume that the vehicle is flat and at rest. Since we know the global gravity vector $\boldsymbol{g}^e$, we can find an initial attitude that corresponds to zero roll and pitch at this point on the Earth. The yaw angle, however, is still undetermined.
- Other sensors, such as a magnetometer, can be added to provide heading measurements. This can be especially useful in conditions where the filter is started abruptly and in transit, and we do not have time to let the filter settle. It is also possible to utilize multiple GNSS antennas to find the UAV's heading. Other sensors, such as cameras, can also be used to validate the filter's estimates, but this depends on the application, and a camera will be of limited utility in homogeneous environments such as at sea.
- Given that this is meant for online use, it is necessary to test it out on a real UAV. This requires rigorous testing, and it can be very risky, depending on how it is done. Temporarily disabling the autopilot's GNSS receiver and using the filter as input instead is reasonably simple, but one must make every effort to ensure that the filter behaves as expected under the takeover phase, yielding a smooth transition. A rough transition can lead to some dramatic control inputs from the autopilot, possibly resulting in a stall or a crash. The spoofing detection is significantly harder to test, mainly because its illegality makes setting up an experiment challenging, but also because successfully spoofing a GNSS receiver is difficult.

Appendix 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):

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

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

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

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

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

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

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	$\boldsymbol{K}_k \leftarrow \boldsymbol{P}_k \boldsymbol{C}^T (\boldsymbol{C} \boldsymbol{P}_k \boldsymbol{C}^T + \boldsymbol{R}_k)^{-1}$
	$\boldsymbol{P}_k \leftarrow (\boldsymbol{I}_n - \boldsymbol{K}_k \boldsymbol{C}) \boldsymbol{P}_k (\boldsymbol{I}_n - \boldsymbol{C}^T \boldsymbol{K}_k^T) + \boldsymbol{K}_k \boldsymbol{R}_k \boldsymbol{K}_k^T$
	$\hat{\boldsymbol{x}}_k \leftarrow \hat{\boldsymbol{x}}_k + \boldsymbol{K}_k (\boldsymbol{y}_k - \boldsymbol{C} \hat{\boldsymbol{x}}_k)$

Appendix B

Code

B.1 mekf_split.m

```
%%

% This file depends on Thor Inge Fossen's MSS library
% (https://github.com/cybergalactic/MSS)

clear all;



colors = [0 0.4470 0.7410;
    0.8500 0.3250 0.0980;
    0.9290 0.6940 0.1250;
    0.4940 0.1840 0.5560;
    0.4660 0.6740 0.1880;
    0.3010 0.7450 0.9330;
    0.6350 0.0780 0.1840];


% Load datasets
load('rtk.mat');
load('stim.mat');
load('cre.mat');

base = [2802320.917 473559.8638 5690836.707]';
base_yaw = deg2rad(-73.5);
base_pitch = deg2rad(0);

Vpars = @(pitch, yaw, dist) dist*[cos(yaw + base_yaw)*cos(pitch +
    base_pitch) sin(yaw + base_yaw)*cos(pitch + base_pitch) -sin(
    pitch + base_pitch)]';

M = @(pitch, yaw, dist) [ cos(yaw)*cos(pitch) -dist*sin(yaw)*cos(
    pitch) -dist*cos(yaw)*sin(pitch);
```

```
                          sin(yaw)*cos(pitch) dist*cos(yaw)*cos(
    pitch) -dist*sin(yaw)*sin(pitch);
                          -sin(pitch) 0 -dist*cos(pitch)];

R_p = diag([10^2, deg2rad(0.1^2), deg2rad(0.1^2)]);

load('pixhawk.mat');


% Set up pixhawk data
timestamp_sec = double(pixhawk.AHR2_TimeUS)*1e-6;
timestamp_sec_gps   = double(pixhawk.GPS_TimeUS)*1e-6;
tow_sec_gps         = double(pixhawk.GPS_GMS)*1e-3;

tow_sec = interp1( timestamp_sec_gps, tow_sec_gps, timestamp_sec, '
    linear' ,'extrap' );
pixhawk = [ tow_sec' pixhawk.AHR2_Roll' pixhawk.AHR2_Pitch' pixhawk
    .AHR2_Yaw' ];


% Set up measurements
gps = [ rtklib.gpst' rtklib.latitude' rtklib.longitude' rtklib.
    height' ];
pars = [ tow' ang_y' ang_x' dist' ];
accl_ = [ stim.tov_gnss' stim.accl_x' stim.accl_y' stim.accl_z' ];
accl_(:,2:4) = accl_(:,2:4)*500;
gyro_ = [ stim.tov_gnss' stim.gyro_x' stim.gyro_y' stim.gyro_z' ];
gyro_(:,2:4) = gyro_(:,2:4)*500*deg2rad(1);
imu = [gyro_ accl_(:,2:4)];

% Measure frame to body rotation matrix, will be used inside the
    loop
R_bm = [-1 0 0; 0 1 0; 0 0 -1];
R_skew = Rzyx( deg2rad(0.76), deg2rad(-4), 0)';
R_bm = R_skew*R_bm;

% NED to ECEF rotation matrix
R_en =  @(lat, lon) [ -sin(lat)*cos(lon), -sin(lon), -cos(lat)*cos(
    lon);
                     -sin(lat)*sin(lon), cos(lon), -cos(lat)*sin(lon)
    ;
                     cos(lat), 0, -sin(lat)];

O3 = zeros(3,3);
I3 = eye(3,3);

% Set filter constants
C = zeros(3,15);
C(1:3,1:3) = eye(3);

R = eye(3);


q_acc = 2.57*10^(-2);
q_gyro = 9.59*10^(-4);
q_bacc = 2.55*10^(-4);
q_bgyro = 6.29*10^(-8);
```

```
81
82 Q = [ q_acc^2*I3 O3 O3 O3;
83      O3 q_gyro^2*I3 O3 O3;
84      O3 O3 q_bacc^2*I3 O3;
85      O3 O3 O3 q_bgyro^2*I3];
86
87 Tacc = 3600;
88 Tgyro = 3600;
89
90 sec = 60*35; % How many secs to run for (60*minutes)
91 filter_rate = 100; % Hz
92 N = sec * filter_rate+100; % How many steps in total
93
94 %% Set up GPS-aided states
95
96 states_gps = struct;
97 states_gps.P = diag([100 100 100 4 4 4 0.030 0.030 0.030 0.01 0.01
       0.01 3*10^-6 3*10^-6 3*10^-6]);
98 states_gps.position = zeros(N,3); % ECEF
99 states_gps.position_geo = zeros(N,3); % geodetic position
100 states_gps.velocity = zeros(N,3); % ECEF
101 states_gps.velocity_norm = zeros(N,1); % ECEF
102 states_gps.attitude = [1, 0, 0, 0]'; % Initial attitude
103 states_gps.error_states = nan(N, 3);
104 states_gps.error_bounds = nan(N, 3);
105 states_gps.errors = nan(N, 8);
106
107 % Approx. correct initial orientation
108 states_gps.attitude = [ 0.30;
109              -0.36;
110              -0.88;
111              -0.03];
112
113 % % Incorrect initial orientation
114 % states_gps.attitude = [-0.0604242,
115 %                -0.971721,
116 %                0.193174,
117 %                -0.121616];
118
119 states_gps.attitude = states_gps.attitude / norm(states_gps.
       attitude);
120
121 states_gps.bias_acc = zeros(N,3);
122 states_gps.bias_gyro = zeros(N,3);
123 states_gps.rpy = zeros(N,3);
124 states_gps.rpy(1,:) = nan(1,3);
125
126 rpy_ref = zeros(N,3);
127 rpy_ref(1,:) = nan(1,3);
128 pixhawk(1,2:end) = nan(1,3);
129
130 % Set initial position
131 [x,y,z] = llh2ecef(deg2rad(9.5919), deg2rad(63.6157), 39.2846);
132
133 states_gps.position(1,:) = [x y z];
134 [lon lat height] = ecef2llh(states_gps.position(1,1), states_gps.
       position(1,2), states_gps.position(1,3));
```

```
states_gps.position_geo(1,1:3) = [rad2deg(lat) rad2deg(lon) height
    ];

states_pars = states_gps;

pars_t = zeros(19148,1);
pars_t_i = 1;

IMU_dt = 1/filter_rate;

GNSS_dt = 1/10;
PARS_dt = 1/10;

first_sec = 383964 + 155;

% Which sample to start with, just to make sure the sensors start
% at the same point in time.
i_imu = 77616;
i_gps = 1493;
i_pars = 2914;
i_pixhawk = 626;


i_pos = 1;

pars_raw_geo = nan(N,3);

pos_diff = nan(N,1);
spoof_displacement = [0 0 0]';

s_ = nan(N,1);
s_(1) = 0;
S_ = nan(N,1);
S_(1) = 0;
G_ = nan(N,1);
G_(1) = 0;

h = 100000;

n_d = nan;
n_c = 0;

detection_pos = nan;
spoof_pos = nan;
nc_pos = nan;

%% Filter (100 Hz)
for t = first_sec:1/filter_rate:first_sec + sec
    % IMU/Time update
    if t > imu(i_imu,1)
        % We have 500 Hz data, but the filter
        % runs at 100 Hz. Take the mean of the last 5 samples
        imu_mean = [0 0 0 0 0 0];
        imu_mean_count = 0;
        while t > imu(i_imu,1)
            i_imu = i_imu + 1;
            imu_mean = imu_mean + imu(i_imu,2:end);
```

```
            imu_mean_count = imu_mean_count + 1;
        end
        imu_mean = (imu_mean ./ imu_mean_count);

        % GNSS estimate
        % Rotate to body frame
        gyro = R_bm * imu_mean(1:3)' - states_gps.bias_gyro(i_pos
    -1,:)';
        gyro = gyro * IMU_dt;
        gyro_norm = norm(gyro);
        accl = R_bm * imu_mean(4:6)' - states_gps.bias_acc(i_pos
    -1,:)';

        % f_e = R_eb f_b + g_e
        accl_ecef = Rquat(states_gps.attitude)*accl + gravity(
    states_gps.position(i_pos-1,:)');

        states_gps.position(i_pos,:) = states_gps.position(i_pos
    -1,:) + IMU_dt * states_gps.velocity(i_pos-1,:) + IMU_dt *
    IMU_dt * accl_ecef' / 2;

        states_gps.velocity(i_pos,:) = states_gps.velocity(i_pos
    -1,:) + IMU_dt*accl_ecef';

         if gyro_norm > 10^(-8)
            d_attitude = states_gps.attitude;
            d_attitude(1) = cos(gyro_norm/2);
            d_attitude(2:4) = (gyro / gyro_norm) *sin(gyro_norm/2);
            states_gps.attitude = quatprod(states_gps.attitude,
    d_attitude);
         end

        states_gps.bias_acc(i_pos, :) = states_gps.bias_acc(i_pos
    -1, :) + IMU_dt * (inv(Tacc*eye(3)) * states_gps.bias_acc(i_pos
    -1, :)')';
        states_gps.bias_gyro(i_pos, :) = states_gps.bias_gyro(i_pos
    -1, :) + IMU_dt * (inv(Tgyro*eye(3)) * states_gps.bias_gyro(
    i_pos-1, :)')';

        A = [ O3 I3  O3              O3
             O3
              O3 O3  -Rquat(states_gps.attitude)*Smtrx(accl)  -
    Rquat(states_gps.attitude) O3
              O3 O3  -Smtrx(gyro/IMU_dt)    O3
               -I3
              O3 O3  O3              -(1/Tacc)*I3
             O3
              O3 O3  O3              O3
             -(1/Tgyro)*I3 ];


        G = [ O3 O3 O3 O3;
            -Rquat(states_gps.attitude) O3 O3 O3;
            O3 -I3 O3 O3;
            O3  O3 I3 O3;
            O3 O3 O3 I3];
```

```
232         [Ad, Qd] = van_Loan(A, G, Q, IMU_dt, 3);
233 
234         states_gps.P = Ad*states_gps.P*Ad' + Qd;
235 
236         % PARS estimate
237         % Rotate to body frame
238         gyro = R_bm * imu_mean(1:3)' - states_pars.bias_gyro(i_pos
    -1,:)';
239         gyro = gyro * IMU_dt;
240         gyro_norm = norm(gyro);
241         accl = R_bm * imu_mean(4:6)' - states_pars.bias_acc(i_pos
    -1,:)';
242 
243         % f_e = R_eb f_b + g_e
244         accl_ecef = Rquat(states_pars.attitude)*accl + gravity(
    states_pars.position(i_pos-1,:)');
245 
246         states_pars.position(i_pos,:) = states_pars.position(i_pos
    -1,:) + IMU_dt * states_pars.velocity(i_pos-1,:) + IMU_dt *
    IMU_dt * accl_ecef' / 2;
247 
248         states_pars.velocity(i_pos,:) = states_pars.velocity(i_pos
    -1,:) + IMU_dt*accl_ecef';
249 
250          if gyro_norm > 10^(-8)
251             d_attitude = states_pars.attitude;
252             d_attitude(1) = cos(gyro_norm/2);
253             d_attitude(2:4) = (gyro / gyro_norm) *sin(gyro_norm/2);
254             states_pars.attitude = quatprod(states_pars.attitude,
    d_attitude);
255          end
256 
257         states_pars.bias_acc(i_pos, :) = states_pars.bias_acc(i_pos
    -1, :) + IMU_dt * (-inv(Tacc*eye(3)) * states_pars.bias_acc(
    i_pos-1, :)')';
258         states_pars.bias_gyro(i_pos, :) = states_pars.bias_gyro(
    i_pos-1, :) + IMU_dt * (-inv(Tgyro*eye(3)) * states_pars.
    bias_gyro(i_pos-1, :)')';
259 
260         A = [ O3 I3  O3              O3
                O3
261               O3 O3  -Rquat(states_pars.attitude)*Smtrx(accl)  -
    Rquat(states_pars.attitude) O3
262               O3 O3  -Smtrx(gyro/IMU_dt)    O3
                   -I3
263               O3 O3  O3              -(1/Tacc)*I3
                  O3
264               O3 O3  O3              O3
                  -(1/Tgyro)*I3 ];
265 
266         G = [ O3 O3 O3 O3;
267              -Rquat(states_pars.attitude) O3 O3 O3;
268              O3 -I3 O3 O3;
269              O3  O3 I3 O3;
270              O3 O3 O3 I3];
271 
272         [Ad, Qd] = calc_Ad_and_Qd_using_van_Loan(A, G, Q, IMU_dt,
```

```
    3);
273
274         states_pars.P = Ad*states_pars.P*Ad' + Qd;
275     else
276         if i_pos ~= 1
277             disp("No IMU data!")
278
279             states_gps.position(i_pos,:) = states_gps.position(
    i_pos-1,:);
280             states_gps.velocity(i_pos,:) = states_gps.velocity(
    i_pos-1,:);
281             states_gps.bias_acc(i_pos, :) = states_gps.bias_acc(
    i_pos-1, :);
282             states_gps.bias_gyro(i_pos, :) = states_gps.bias_gyro(
    i_pos-1, :);
283
284             states_pars.position(i_pos,:) = states_pars.position(
    i_pos-1,:);
285             states_pars.velocity(i_pos,:) = states_pars.velocity(
    i_pos-1,:);
286             states_pars.bias_acc(i_pos, :) = states_pars.bias_acc(
    i_pos-1, :);
287             states_pars.bias_gyro(i_pos, :) = states_pars.bias_gyro
    (i_pos-1, :);
288         end
289     end
290
291     % GNSS update
292     if t > gps(i_gps,1)
293         K = states_gps.P*C'*inv(C*states_gps.P*C' + R);
294         [x_, y_, z_] = llh2ecef(deg2rad(gps(i_gps,3)), deg2rad(gps(
    i_gps,2)), gps(i_gps,4));
295         error = [x_ y_ z_]' + spoof_displacement - states_gps.
    position(i_pos,:)';
296
297         R_nb = R_en(deg2rad(states_gps.position_geo(i_pos-1,1)),
    deg2rad(states_gps.position_geo(i_pos-1,2)))'*Rquat(states_gps.
    attitude);
298         ypr = rotm2eul(R_nb, 'ZYX');
299         oo = [ ypr(3) ypr(2) ypr(1)];
300         oo = rad2deg(oo);
301
302         states_gps.errors(i_pos,1:3) = error';
303         states_gps.errors(i_pos,4:6) = ssa(pixhawk(i_pixhawk,2:end)
     - oo, 'deg')';
304         states_gps.errors(i_pos,7) = norm(error);
305         states_gps.errors(i_pos,8) = norm(states_gps.errors(i_pos
    ,4:6));
306
307
308         dx = K*error;
309         states_gps.error_states(i_gps,1) = sqrt(sum(dx(1:3) .* dx
    (1:3)));
310         states_gps.error_states(i_gps,2) = sqrt(sum(dx(3:6) .* dx
    (3:6)));
311         states_gps.error_states(i_gps,3) = sqrt(sum(dx(6:9) .* dx
    (6:9)));
```

```
312
313         states_gps.P = (eye(15) - K*C) * states_gps.P * (eye(15) -
    K*C)' + K*R*K';
314
315         states_gps.error_bounds(i_gps,1) = 3*sqrt(sum(diag(
    states_gps.P(1:3,1:3))))';
316         states_gps.error_bounds(i_gps,2) = 3*sqrt(sum(diag(
    states_gps.P(3:6,3:6))))';
317         states_gps.error_bounds(i_gps,3) = 3*sqrt(sum(diag(
    states_gps.P(6:9,6:9))))';
318
319         states_gps.position(i_pos,:) = (states_gps.position(i_pos
    ,:) + dx(1:3)');
320         states_gps.velocity(i_pos,:) = (states_gps.velocity(i_pos
    ,:) + dx(4:6)');
321
322         da = dx(7:9);
323
324         delta_q_hat = (1/(16 + da'*da)) * [ 16 - da'*da; 8*da ];
325
326         if norm(da) > 4
327             delta_q_hat = -delta_q_hat;
328         end
329
330         states_gps.attitude = quatprod(states_gps.attitude,
    delta_q_hat);
331         states_gps.attitude = states_gps.attitude / norm(states_gps
    .attitude);
332         states_gps.bias_acc(i_pos,:) = (states_gps.bias_acc(i_pos
    ,:) + dx(10:12)');
333         states_gps.bias_gyro(i_pos,:) = (states_gps.bias_gyro(i_pos
    ,:) + dx(13:15)');
334
335
336         G = [ eye(3) zeros(3) zeros(3) zeros(3) zeros(3);
337               zeros(3) eye(3) zeros(3) zeros(3) zeros(3);
338               zeros(3) zeros(3) delta_q_hat(1)*eye(3)-Smtrx(
    delta_q_hat(2:4)) zeros(3) zeros(3);
339               zeros(3) zeros(3) zeros(3) eye(3) zeros(3);
340               zeros(3) zeros(3) zeros(3) zeros(3) eye(3)];
341
342         states_gps.P = G*states_gps.P*G';
343         i_gps = i_gps + 1;
344     end
345
346     % PARS update
347     if t > pars(i_pars,1)
348         pars_t(pars_t_i) = t;
349         pars_t_i = pars_t_i + 1;
350
351         M_ = M(deg2rad(pars(i_pars,2)), deg2rad(pars(i_pars,3)),
    pars(i_pars,4));
352         R_pars = R_en(deg2rad(63.51559), deg2rad(9.59171)) * M_ *
    R_p * M_' * R_en(deg2rad(63.51559), deg2rad(9.59171))';
353         K = states_pars.P*C'*inv(C*states_pars.P*C' + R_pars);
354
355         pp = (base + R_en(deg2rad(63.51559), deg2rad(9.59171))*
```

```
    Vpars(deg2rad(pars(i_pars,2)), deg2rad(pars(i_pars,3)), pars(
    i_pars,4)))';
356
357         x_ = pp(1);
358         y_ = pp(2);
359         z_ = pp(3);
360         [lon, lat, height] =ecef2llh(x_, y_, z_);
361         pars_raw_geo(i_pos,:) = [rad2deg(lat) rad2deg(lon) height];
362         if t < first_sec + 600
363             [x_, y_, z_] = llh2ecef(deg2rad(gps(i_gps,3)), deg2rad(
    gps(i_gps,2)), gps(i_gps,4));
364         end
365
366         error = [x_ y_ z_]' - states_pars.position(i_pos,:)';
367
368         [x_, y_, z_] = llh2ecef(deg2rad(gps(i_gps,3)), deg2rad(gps(
    i_gps,2)), gps(i_gps,4));
369         error_g = states_pars.position(i_pos,:)' - [x_ y_ z_]';
370
371         R_nb = R_en(deg2rad(states_pars.position_geo(i_pos-1,1)),
    deg2rad(states_pars.position_geo(i_pos-1,2)))'*Rquat(
    states_pars.attitude);
372         ypr = rotm2eul(R_nb, 'ZYX');
373         rpy_ = [ ypr(3) ypr(2) ypr(1)];
374         rpy_ = rad2deg(rpy_);
375
376         states_pars.errors(i_pos,1:3) = error_g';
377         states_pars.errors(i_pos,4:6) = ssa(rpy_ - pixhawk(
    i_pixhawk,2:end), 'deg')';
378         states_pars.errors(i_pos,7) = norm(error_g);
379         states_pars.errors(i_pos,8) = norm(states_pars.errors(i_pos
    ,4:6));
380
381         dx = K*error;
382         states_pars.error_states(i_pars,1) = sqrt(sum(dx(1:3) .* dx
    (1:3)));
383         states_pars.error_states(i_pars,2) = sqrt(sum(dx(3:6) .* dx
    (3:6)));
384         states_pars.error_states(i_pars,3) = sqrt(sum(dx(6:9) .* dx
    (6:9)));
385
386         states_pars.P = (eye(15) - K*C) * states_pars.P * (eye(15)
    - K*C)' + K*R_pars*K';
387
388         states_pars.error_bounds(i_pars,1) = 3*sqrt(sum(diag(
    states_pars.P(1:3,1:3))))';
389         states_pars.error_bounds(i_pars,2) = 3*sqrt(sum(diag(
    states_pars.P(3:6,3:6))))';
390         states_pars.error_bounds(i_pars,3) = 3*sqrt(sum(diag(
    states_pars.P(6:9,6:9))))';
391
392         states_pars.position(i_pos,:) = (states_pars.position(i_pos
    ,:) + dx(1:3)');
393         states_pars.velocity(i_pos,:) = (states_pars.velocity(i_pos
    ,:) + dx(4:6)');
394
395         da = dx(7:9);
```

```
396
397         delta_q_hat = (1/(16 + da'*da)) * [ 16 - da'*da; 8*da ];
398
399         if norm(da) > 4
400             delta_q_hat = -delta_q_hat;
401         end
402
403
404         states_pars.attitude = quatprod(states_pars.attitude,
    delta_q_hat);
405         states_pars.attitude = states_pars.attitude / norm(
    states_pars.attitude);
406         states_pars.bias_acc(i_pos,:) = (states_pars.bias_acc(i_pos
    ,:) + dx(10:12)');
407         states_pars.bias_gyro(i_pos,:) = (states_pars.bias_gyro(
    i_pos,:) + dx(13:15)');
408
409
410         G = [ eye(3) zeros(3) zeros(3) zeros(3) zeros(3);
411               zeros(3) eye(3) zeros(3) zeros(3) zeros(3);
412               zeros(3) zeros(3) delta_q_hat(1)*eye(3)-Smtrx(
    delta_q_hat(2:4)) zeros(3) zeros(3);
413               zeros(3) zeros(3) zeros(3) eye(3) zeros(3);
414               zeros(3) zeros(3) zeros(3) zeros(3) eye(3)];
415
416         states_pars.P = G*states_pars.P*G';
417         i_pars = i_pars + 1;
418     end
419
420
421
422     % Convert ECEF to geodetic
423     % For plotting and for R_en generation
424     [states_gps.position_geo(i_pos,2), states_gps.position_geo(
    i_pos,1), states_gps.position_geo(i_pos,3)] = ecef2llh(
    states_gps.position(i_pos,1), states_gps.position(i_pos,2),
    states_gps.position(i_pos,3));
425     states_gps.position_geo(i_pos,1:2) = rad2deg(states_gps.
    position_geo(i_pos,1:2));
426
427     [states_pars.position_geo(i_pos,2), states_pars.position_geo(
    i_pos,1), states_pars.position_geo(i_pos,3)] = ecef2llh(
    states_pars.position(i_pos,1), states_pars.position(i_pos,2),
    states_pars.position(i_pos,3));
428     states_pars.position_geo(i_pos,1:2) = rad2deg(states_pars.
    position_geo(i_pos,1:2));
429
430
431     % Log position difference
432     pos_diff(i_pos) = norm(states_gps.position(i_pos,:) -
    states_pars.position(i_pos,:));
433
434     if t > first_sec + 800
435         if isnan(spoof_pos)
436             spoof_pos = states_gps.position_geo(i_pos,:);
437         end
438         %spoof_displacement = spoof_displacement + [0.01 0 0]';
```

```
439     end
440
441     % Log roll, pitch & yaw
442     % GPS
443     R_nb = R_en(deg2rad(states_gps.position_geo(i_pos,1)), deg2rad(
        states_gps.position_geo(i_pos,2)))'*Rquat(states_gps.attitude);
444     ypr = rotm2eul(R_nb, 'ZYX');
445     states_gps.rpy(i_pos,:) = [ypr(3) ypr(2) ypr(1)];
446     states_gps.rpy(i_pos,:) = rad2deg(states_gps.rpy(i_pos,:));
447
448     if (states_gps.rpy(i_pos,3) > 170 && states_gps.rpy(i_pos+1,3)
        < -170) || (states_gps.rpy(i_pos,3) < -170 && states_gps.rpy(
        i_pos+1,3) > 170)
449         states_gps.rpy(i_pos,3) = NaN;
450     end
451
452
453     % PARS
454     R_nb = R_en(deg2rad(states_pars.position_geo(i_pos,1)), deg2rad
        (states_pars.position_geo(i_pos,2)))'*Rquat(states_pars.
        attitude);
455     ypr = rotm2eul(R_nb, 'ZYX');
456     states_pars.rpy(i_pos,:) = [ypr(3) ypr(2) ypr(1)];
457     states_pars.rpy(i_pos,:) = rad2deg(states_pars.rpy(i_pos,:));
458
459     if (states_pars.rpy(i_pos,3) > 170 && states_pars.rpy(i_pos
        +1,3) < -170) || (states_pars.rpy(i_pos,3) < -170 &&
        states_pars.rpy(i_pos+1,3) > 170)
460         states_pars.rpy(i_pos,3) = NaN;
461     end
462
463
464     % Log roll, pitch & yaw ground truth
465     if t > pixhawk(i_pixhawk,1)
466         i_pixhawk = i_pixhawk + 1;
467     end
468     rpy_ref(i_pos,:) = wrapTo180(pixhawk(i_pixhawk,2:end));
469
470     sigma_guess = norm(diag(states_gps.P(1:3,1:3))) + norm(diag(
        states_pars.P(1:3,1:3)));
471     delta = sigma_guess*2;
472     mu_guess = sigma_guess;
473
474     % Spoofing detection
475     if isnan(n_d)
476         s_(i_pos+1) = ((delta)/(sigma_guess^2)) * (pos_diff(i_pos)
        - mu_guess - delta/2);
477         S_(i_pos+1) = S_(i_pos) + s_(i_pos+1);
478         G_(i_pos+1) = max(G_(i_pos) + s_(i_pos+1), 0);
479         [o_, nc] = min(S_);
480         nc_pos = states_gps.position_geo(nc,:);
481         if G_(i_pos+1) > h
482             n_d = t;
483             detection_pos = states_gps.position_geo(i_pos,:);
484         end
485     end
486
```

```
487
488
489         i_pos = i_pos + 1;
490     end
491
492     %% PLOTTING
493     close all;
494
495     states_gps.position_geo(states_gps.position_geo == [0 0 0]) = NaN;
496     states_pars.position_geo(states_pars.position_geo == [0 0 0]) = NaN
        ;
497
498     % Get rid of the vertical lines/jumps
499     % GPS
500     for i = 1:size(states_gps.rpy,1)-1
501         if (states_gps.rpy(i,3) > 170 && states_gps.rpy(i+1,3) < -170)
            || (states_gps.rpy(i,3) < -170 && states_gps.rpy(i+1,3) > 170)
502             states_gps.rpy(i,3) = NaN;
503         end
504     end
505
506
507     % PARS
508     for i = 1:size(states_pars.rpy,1)-1
509         if (states_pars.rpy(i,3) > 170 && states_pars.rpy(i+1,3) <
            -170) || (states_pars.rpy(i,3) < -170 && states_pars.rpy(i+1,3)
             > 170)
510             states_pars.rpy(i,3) = NaN;
511         end
512     end
513
514
515     % Pixhawk
516     for i = 1:size(rpy_ref,1)-1
517         if (rpy_ref(i,3) > 170 && rpy_ref(i+1,3) < -170) || (rpy_ref(i
            ,3) < -170 && rpy_ref(i+1,3) > 170)
518             rpy_ref(i,3) = NaN;
519         end
520     end
521
522     states_pars.errors = rmmissing(states_pars.errors);
523     states_gps.errors = rmmissing(states_gps.errors);
524
525     p_m_roll = mean(states_pars.errors(:,4));
526     p_r_roll = sqrt(mean(states_pars.errors(:,4).^2));
527     p_mae_roll = mean(abs(states_pars.errors(:,4)));
528
529     p_m_pitch = mean(states_pars.errors(:,5));
530     p_r_pitch = sqrt(mean(states_pars.errors(:,5).^2));
531     p_mae_pitch = mean(abs(states_pars.errors(:,5)));
532
533     p_m_yaw = mean(states_pars.errors(:,6));
534     p_r_yaw = sqrt(mean(states_pars.errors(:,6).^2));
535     p_mae_yaw = mean(abs(states_pars.errors(:,6)));
536
537     p_m_a = norm([p_m_roll p_m_pitch p_m_yaw]);
538     p_r_a = norm([p_r_roll p_r_pitch p_r_yaw]);
```

```
p_mae_a = norm([p_mae_roll p_mae_pitch p_mae_yaw]);

p_m_x = mean(states_pars.errors(:,1));
p_r_x = sqrt(mean(states_pars.errors(:,1).^2));
p_mae_x = mean(abs(states_pars.errors(:,1)));

p_m_y = mean(states_pars.errors(:,2));
p_r_y = sqrt(mean(states_pars.errors(:,2).^2));
p_mae_y = mean(abs(states_pars.errors(:,2)));

p_m_z = mean(states_pars.errors(:,3));
p_r_z = sqrt(mean(states_pars.errors(:,3).^2));
p_mae_z = mean(abs(states_pars.errors(:,3)));

p_m_p = norm([p_m_x p_m_y p_m_z]);
p_r_p = norm([p_r_x p_r_y p_r_z]);
p_mae_p = norm([p_mae_x p_mae_y p_mae_z]);

% Plot CUSUM
figure()
subplot(4,1,1);
plot((1:1/filter_rate:1.99+sec)/60, pos_diff, 'color', colors(1,:))
    ;
grid on;
hold on;
xlabel('t [min]')
ylabel('[m]')
title('Distance from GPS-aided position to PARS-aided position')
subplot(4,1,2);
subplot(4,1,2)
plot((1:1/filter_rate:1.99+sec)/60, s_, 'color', colors(1,:))
grid on;
title('s')
xlim([0 40])
subplot(4,1,3)
plot((1:1/filter_rate:1.99+sec)/60, S_, 'color', colors(1,:))
grid on;
title('S')
xlim([0 40])
subplot(4,1,4)
plot((1:1/filter_rate:1.99+sec)/60, G_, 'color', colors(1,:))
grid on;
title('G')
hold on;
yline(h);
ylim([0 h+100])
xlim([0 40])

% Plot RPY
figure()
subplot(3, 1, 1);
plot((1:1/filter_rate:1+sec)/60, states_gps.rpy(1:i_pos-1,1), '
    color', colors(1,:));
hold on;
plot((1:1/filter_rate:1+sec)/60, states_pars.rpy(1:i_pos-1,1), '
    color', colors(2,:));
plot((1:1/filter_rate:1+sec)/60, rpy_ref(1:i_pos-1,1), 'color',
```

```
    colors(3,:));
legend('gps', 'pars', 'ref');
title('Roll');
ylabel('[deg]');
xlabel('t [min]');
grid on;
subplot(3, 1, 2);
plot((1:1/filter_rate:1+sec)/60, states_gps.rpy(1:i_pos-1,2), '
    color', colors(1,:));
hold on;
plot((1:1/filter_rate:1+sec)/60, states_pars.rpy(1:i_pos-1,2), '
    color', colors(2,:));
plot((1:1/filter_rate:1+sec)/60, rpy_ref(1:i_pos-1,2), 'color',
    colors(3,:));
legend('gps', 'pars', 'ref');
title('Pitch');
ylabel('[deg]');
xlabel('t [min]');
grid on;
subplot(3, 1, 3);
plot((1:1/filter_rate:1+sec)/60, states_gps.rpy(1:i_pos-1,3), '
    color', colors(1,:));
hold on;
plot((1:1/filter_rate:1+sec)/60, states_pars.rpy(1:i_pos-1,3), '
    color', colors(2,:));
plot((1:1/filter_rate:1+sec)/60, rpy_ref(1:i_pos-1,3), 'color',
    colors(3,:));
legend('gps', 'pars', 'ref');
title('Yaw');
ylabel('[deg]');
xlabel('t [min]');
grid on;

% 3D ECEF position
figure()
plot3(states_gps.position(1:end-100,1), states_gps.position(1:end
    -100,2), states_gps.position(1:end-100,3))
grid on; hold on
plot3(states_pars.position(1:end-100,1), states_pars.position(1:end
    -100,2), states_pars.position(1:end-100,3))
xlabel('x [m]')
ylabel('y [m]')
zlabel('z [m]')

% GNSS Bias estimates
figure()
subplot(2, 1, 1);
hold on;
plot((1:1/filter_rate:1+sec)/60, states_pars.bias_acc(1:i_pos-1,1),
     'color', colors(1,:));
plot((1:1/filter_rate:1+sec)/60, states_pars.bias_acc(1:i_pos-1,2),
     'color', colors(2,:));
plot((1:1/filter_rate:1+sec)/60, states_pars.bias_acc(1:i_pos-1,3),
     'color', colors(3,:));
title('b^b_{acc} (GNSS-aided)');
legend('x', 'y', 'z');
ylabel('[m/s^2]')
```

```
638 xlabel('t [min]')
639 grid on;
640 subplot(2, 1, 2);
641 hold on;
642 plot((1:1/filter_rate:1+sec)/60, states_pars.bias_gyro(1:i_pos-1,1)
        , 'color', colors(1,:));
643 plot((1:1/filter_rate:1+sec)/60, states_pars.bias_gyro(1:i_pos-1,2)
        , 'color', colors(2,:));
644 plot((1:1/filter_rate:1+sec)/60, states_pars.bias_gyro(1:i_pos-1,3)
        , 'color', colors(3,:));
645 title('b^b_{ars} (GNSS-aided)');
646 ylabel('[rad]')
647 xlabel('t [min]')
648 legend('x', 'y', 'z');
649 grid on;
650
651 % PARS Bias estimates
652 figure()
653 subplot(2, 1, 1);
654 hold on;
655 plot((1:1/filter_rate:1+sec)/60, states_pars.bias_acc(1:i_pos-1,1),
         'color', colors(1,:));
656 plot((1:1/filter_rate:1+sec)/60, states_pars.bias_acc(1:i_pos-1,2),
         'color', colors(2,:));
657 plot((1:1/filter_rate:1+sec)/60, states_pars.bias_acc(1:i_pos-1,3),
         'color', colors(3,:));
658 title('b^b_{acc} (PARS-aided)');
659 legend('x', 'y', 'z');
660 ylabel('[m/s^2]')
661 xlabel('t [min]')
662 grid on;
663 subplot(2, 1, 2);
664 hold on;
665 plot((1:1/filter_rate:1+sec)/60, states_pars.bias_gyro(1:i_pos-1,1)
        , 'color', colors(1,:));
666 plot((1:1/filter_rate:1+sec)/60, states_pars.bias_gyro(1:i_pos-1,2)
        , 'color', colors(2,:));
667 plot((1:1/filter_rate:1+sec)/60, states_pars.bias_gyro(1:i_pos-1,3)
        , 'color', colors(3,:));
668 title('b^b_{ars} (PARS-aided)');
669 ylabel('[rad]')
670 xlabel('t [min]')
671 legend('x', 'y', 'z');
672 grid on;
673
674 % Geodetic (horizontal) position
675 figure()
676 grid on;
677 hold on;
678 %plot(gps(1:i_gps,3), gps(1:i_gps,2), 'color', colors(1,:), 'marker
        ', '.', 'linestyle', 'none'); % all GPS points
679 %plot(gps(10:10:i_gps,3), gps(10:10:i_gps,2), 'color', colors(1,:),
         'marker', '.', 'linestyle', 'none'); % only the ones we're
        using
680 %plot(pars_raw_geo(:,2), pars_raw_geo(:,1), 'color', colors(1,:), '
        marker', '.', 'linestyle', 'none');
681 plot(states_gps.position_geo(:,2), states_gps.position_geo(:,1), '
```

```
    color', colors(1,:)); % GPS
plot(states_pars.position_geo(:,2), states_pars.position_geo(:,1),
    'color', colors(2,:)); % PARS
axis([9.49 9.60 63.586 63.635]);
xlabel('Longitude [deg]')
ylabel('Latitude [deg]')
legend('GNSSS-aided', 'PARS-aided')
```

B.2 gravity.m

```
function g_b_e = gravity( p_eb_e )
%p_eb_e2GRAVITY_ECEF - Calculates the gravity vector g_b^e of the
    ECEF frame
%
% Input:    p_eb_e  Cartesian position decomposed in the ECEF frame
% Output:   g_b_e   Gravity vector g_b^e decomposed in the ECEF
    frame
%
% Calculation based on Groves (2008), Principles of GNSS, Inertial,
     and
% Multisensor Integrated Navigation Systems, Artech House

% Parameters
a           = 6378137;        % WGS84 equatorial radius in meters
mu          = 3.986004418e14; % WGS84 Earth gravitational constant
    (m^3 s^-2)
J_2         = 1.082627e-3;    % WGS84 Earth's second gravitational
    constant
omega_ie    = 7.292115e-5;    % Earth's rotation rate (rad/s)

% Distance from the Earth's center
mag_r = norm( p_eb_e );
if mag_r > 0
    z_scale = 5 * (p_eb_e(3) / mag_r)^2;
    gamma = -mu / mag_r^3 *(....
        p_eb_e + 1.5 * J_2 * (a / mag_r)^2 *....
        [.....
        (1 - z_scale) * p_eb_e(1);...
        (1 - z_scale) * p_eb_e(2);...
        (3 - z_scale) * p_eb_e(3);....
        ]);

    g_b_e(1:2,1)    = gamma(1:2) + omega_ie^2 * p_eb_e(1:2);
    g_b_e(3)        = gamma(3);
else
    g_b_e = [0;0;0];
end
```

B.3 van_Loan.m

```
function [ Phid, Qd ] = van_Loan( F, G, Q, Ts, approx_order )
%CALC_PHId_AND_Qd_USING_VAN_LOAN Calulated the transition matirx
    PHId and the discrete
%time process noise covariance Qd such that the covariance can be
%propagated using P[k] = Phid[k]*P[k-1]*Phid'[k] + Qd[k]
```

```
5
6       dim_sys = size(F,1);
7       A = [   -F              G*Q*G';
8               zeros(dim_sys)  F';
9           ]*Ts;
10      if nargin < 5
11          B = expm( A );
12      else
13          B = A^0;
14          for k = 1:approx_order
15              B = B + A^k/factorial(k);
16          end
17      end
18
19      dim_A       = size(A,1);
20      range_Phid  = dim_sys+1:dim_A;
21      range_Qd    = 1:dim_sys;
22
23      Phid    = B(     range_Phid,   range_Phid)';
24      Qd      = Phid*B(  range_Qd,   range_Phid);
25  end
```

Bibliography

Albrektsen, S. M., Bryne, T. H. & Johansen, T. A. (2018*a*), Phased array radio system aided inertial navigation for unmanned aerial vehicles, *in* '2018 IEEE Aerospace Conference', pp. 1–11.

Albrektsen, S. M., Bryne, T. H. & Johansen, T. A. (2018*b*), Robust and secure uav navigation using gnss, phased-array radio system and inertial sensor fusion, *in* '2018 IEEE Conference on Control Technology and Applications (CCTA)', IEEE, pp. 1338–1345.

Arienzo, L. (2010), 'Rf interference vulnerability assessment for gnss receivers', *JRC Scientific and Technical Reports* pp. 5–21.

Brouk, J. D. A. (2019), 'Propagation of uncertainty through coning, sculling, and scrolling corrections for inertial navigation'.

CH Robotics (2020), 'Understanding euler angles'. [Online; accessed November 30, 2020].
URL: *http://www.chrobotics.com/library/understanding-euler-angles*

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

Everything RF (2020), 'What is a phased array antenna?'. [Online; accessed November 30, 2020].
URL: *https://cdn.everythingrf.com/live/1593618423194_637292152227868404.png*

Fossen, T. (2011), *Handbook of Marine Craft Hydrodynamics and Motion Control*.

Gao, G. X., Sgammini, M., Lu, M. & 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. & 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.
URL: *http://www.sciencedirect.com/science/article/pii/S0273117710001419*

Granjon, P. (2013), 'The cusum algorithm - a small review'.

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'.

Gustafsson, F. (2001), *Adaptive Filtering and Change Detection*, Wiley.

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

Leo Rover Docs (2020), 'Accelerometer and gyroscope axes'. [Online; accessed November 30, 2020].
URL: *https://docs.leorover.tech/integrations/imu-module*

Loan, C. (1978), 'Computing integrals involving the matrix exponential', *Automatic Control, IEEE Transactions on* **23**, 395 – 404.

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

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

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

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

Pixhawk (2020), 'Pixhawk — the hardware standard for open-source autopilots'. [Online; accessed December 17, 2020].
URL: *https://pixhawk.org/*

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

Sensonor (2020), 'Stim300'. [Online; accessed December 10, 2020].
URL: *https://www.sensonor.com/products/inertial-measurement-units/stim300/*

Skywalker (2020), 'Skywalker x8'. [Online; accessed December 10, 2020].
URL: *http://skywalkermodel.com/en/76.html*

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. & Johansen, T. (2019), Pose estimation of uavs based on ins aided by two independent low-cost gnss receivers, pp. 1425–1435.

Subirana, J., Zornoza, J., Hernández-Pajares, M., Agency, E. S. & 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. & Swaszek, R. (2018), Limits on gnss performance at high latitudes, pp. 160–176.

u-blox (2020), 'Neo/lea-m8t series'. [Online; accessed December 17, 2020].
URL: *https://www.u-blox.com/en/product/neolea-m8t-series*

Wei, Y., Hong, T., Khelloufi, A., Ning, H. & 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.
URL: *http://www.sciencedirect.com/science/article/pii/S1877050919302339*

Wikipedia (2020), 'Earth-centered inertial'. [Online; accessed December 5, 2020].
URL: *https://upload.wikimedia.org/wikipedia/commons/3/32 /Earth_Centered_Inertial_Coordinate_System.png*