Bachelor's project

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

Anders Thallaug Fagerli
Odin Aleksander Severinsen

The Error-State Kalman Filter for Singularity-Free State Estimation in Inertial Navigation Systems

Bachelor's project in Mathematical Sciences
Supervisor: Håkon Tjelmeland, Torleiv Håland Bryne
May 2020

NTNU
Kunnskap for en bedre verden

ascend

Anders Thallaug Fagerli
Odin Aleksander Severinsen

The Error-State Kalman Filter for Singularity-Free State Estimation in Inertial Navigation Systems

Bachelor's project in Mathematical Sciences
Supervisor: Håkon Tjelmeland, Torleiv Håland Bryne
May 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

Advances in modern robotics have in the recent years given rise to numerous applications in the field of estimation for autonomous systems. This thesis provides a probabilistic framework for doing estimation of position, velocity and orientation with the error-state Kalman filter, and its application in inertial navigation systems.

The Kalman filter is first presented as a Bayes filter in a hidden Markov model under a linear Gaussian assumption. The extended Kalman filter is further derived for handling nonlinear models, and from this the error-state Kalman filter is presented in its most general form as a way of improving linearization accuracy and allowing for different state paramaterizations to avoid singular state dynamics. Navigation theory is presented together with a discussion on representation of orientation, and together with derived kinematic equations, the error-state Kalman filter is fitted into state estimation of an inertial navigation system. Results from simulated and experimental data show that the error-state Kalman filter adequately estimates the position, velocity and orientation of an unmanned aerial vehicle.

Sammendrag

Fremskritt innen moderne robotikk har de siste årene gitt opphav til mange bruksområder innen estimering for autonome systemer. Denne avhandlingen gir et statistisk rammeverk for estimering av posisjon, hastighet og orientering med et feiltilstands Kalmanfilter, og dens anvendelse innen treghetsnavigasjon.

Kalmanfilteret blir først presentert som et Bayes filter i en skjult Markov modell under en lineær Gaussisk antakelse. Det utvidede Kalmanfilteret blir videre utledet for håndtering av ulineære modeller, og fra dette presenteres feiltilstands Kalmanfilteret i sin mest generelle form som en måte å forbedre lineariseringsnøyaktighet og gi mulighet for forskjellige tilstandsparameteriseringer for å unngå singulær dynamikk. Navigasjonsteori presenteres sammen med en diskusjon om representasjon av orientering, og sammen med utledede kinematiske ligninger blir feiltilstands Kalmanfilteret tilpasset tilstandsestimering av et treghetsnavigasjonssystem. Resultater fra simulerte og eksperimentelle data viser at filteret anslår posisjonen, hastigheten og orienteringen til et ubemannet luftfartøy tilstrekkelig.

Preface

This thesis is written as a finalization of the three-year Bachelor's Programme in Mathematical Sciences at the Norwegian University of Science and Technology, with a specialization in Statistics, as part of the course MA2002 Bachelor's Project in Mathematical Studies.

The thesis is written in the context of competing in a competition for autonomous unmanned aerial vehicles as part of the student-organization Ascend NTNU, with the task of autonomously flying a distance of three kilometers, replacing a physical part on the mast of a ship, and flying back to the point of take-off. Estimating the state of the vehicle at all times is one of the many subproblems in this task, and will set the limits on the scope of this thesis.

We would like to thank our supervisor Håkon Tjelmeland and co-supervisor Torleiv H. Bryne for their continuous support throughout the semester; your feedback and guidance has played an invaluable part in shaping this thesis. A big thanks also goes to Martin L. Sollie, who has relentlessly guided us through many of the theoretical and practical obstacles in the field of inertial navigation, in addition to providing us with necessary data. We would also like to thank Ascend NTNU and all its team members for the incredible learning environment we have been exposed to in the past year, and for challenging students to become better engineers. A special thanks to our group leader Torjus Bakkene, for all his support, encouragement and enthusiasm, and for allowing us to focus on ESKF this spring after our endeavours with VINS during the fall semester. You are truly "one shaft of a man". We would like to thank Edmund F. Brekke and Lars-Christian N. Tokle that provided us with the data used in this thesis during the course TTK4250 Sensor fusion. Lastly, we would like to thank senior executive officer Anniken Skotvoll for her commitment to the students at NTNU, ultimately making this thesis possible.

Anders Thallaug Fagerli,
Odin Aleksander Severinsen

Trondheim, May 2020

Figure 1: *The drone of Ascend NTNU Team 2020.*

Figure 2: *The logo of Ascend NTNU.*

Table of Contents

III Inertial navigation and kinematics

IV Filter validation and results

List of Tables

List of Figures

List of Abbreviations

ANEES Average normalized estimation error squared
ANIS Average normalized innovation squared

ECEF Earth-centered, Earth-fixed
EKF Extended Kalman filter
ESKF Error-state Kalman filter

GNSS Global Navigation Satellite System

HMM Hidden Markov Model

IKF Indirect Kalman filter
IMU Inertial Measurement Unit
INS Inertial Navigation System

KF Kalman filter

LLH Longitude, latitude, height
LMMSE Linear minimum mean square error
LTV Linear time-varying

MAP Maximum a posteriori
MEKF Multiplicative extended Kalman filter
MMSE Minimum mean square error
MSE Mean squared error

NED North-east-down
NEES Normalized estimation error squared

NIS Normalized innovation squared

ODE Ordinary differential equation

PDF Probability density function

RMSE Root mean square error
ROS Robot Operating System

UAV Unmanned aerial vehicle
UKF Unscented Kalman filter

I

Introduction and preliminaries

1 | Introduction

An integral part of making a robot autonomous involves making the robot able of determining, for example, its position and velocity, if any. Such quantities can, in theory, be calculated from exact mathematical models describing the kinematics of the robot. This approach, however, is inherently not feasible in practice. To know the position and velocity of the robot sufficiently accurate at every relevant point in time would require knowledge about the environment that is impossible to acquire, neither before-hand nor under operation. In addition, it would require an incredibly advanced and intricate mathematical model that considers all possible effects that affect the robot's position and velocity, given that such a model even exists. Considering that this algorithm also has to run on a computer with limited computational power, a different approach must be taken.

The solution in modern robotics [1] is to aggregate uncertainties and inaccuracies in the mathematical models as stochastic variables, affecting both the physics of the robot and the sensors that allow the robot to observe the outside world. This has several advantages. Firstly, it allows for much simpler and more tangible models for how the robot moves in its environment. It also accounts for imperfections in a real sensor, letting the robot itself estimate the deviations in the measurements, resulting in more robust and accurate estimates.

Building on this fundamental approach of probabilistic modelling, powerful, mathematical tools from statistics become available for use in state estimation. One such tool is Bayesian inference, building on Bayes' rule for conditional probability. As will become evident, such a method allows for combining information on believed movement of the robot with measurements, yielding a better estimate than each estimate independently. The method also facilitates a recursive algorithm, laying the foundations for the Kalman filter.

The Kalman filter is today used in numerous applications within a variety of fields. This thesis will, in addition to the general statistical theory of the Kalman

filter, study its application in *navigation* for unmanned aerial vehicles (UAVs). Navigation will here be coined as the process of observing variables describing characteristics as the position, velocity and orientation of the UAV, hereby denoted as *states*. Due to the uncertain nature of the movements of the UAV and the imperfect measurements from on-board sensors, the states do not evolve in any deterministic manner, and are therefore reasonably modeled as stochastic processes. The task is then to retrieve best possible *estimates* of the states, given some constraints on e.g. time and processing required by the estimation method.

The aim of this thesis is to present the *error-state Kalman filter* (ESKF) and how it is suited for state estimation of inertial navigation systems, and does it the following the way. Chapter 2 introduces preliminary theory from statistics and the concept of Bayesian inference, with the prior and posterior distributions used by the Kalman filter together with underlying assumptions. Additionally, the mulitvariate Gaussian is discussed. Chapter 3 introduces the linear Kalman filter, an application of Bayesian inference. The underlying models of the Kalman filter are introduced, and how they relate to the prior and posterior distribution in Bayesian inference. The *extended Kalman filter* (EKF) is introduced in Chapter 4, an extension to the Kalman filter for nonlinear systems. Chapter 5 will introduce the ESKF, first in its most general form to further build on top of the EKF presented in the previous chapter as an improved filter for nonlinear estimation. Chapter 6 introduces preliminary theory for understanding *inertial navigation systems* (INS), the challenges it imposes on the filtering algorithm chosen and the sensors used. Chapter 7 will introduce the kinematic model used to predict the movement of the UAV, together with the required states. Chapter 8 then reintroduces the ESKF, now in an INS setting where the kinematic model and sensor models are inserted to complete the algorithm for inertial navigation. With the ESKF for INS finally presented, Chapter 9 will introduce metrics and methods for quantifying the performance of a filter, and Chapter 10 will use these metrics and methods to evaluate the performance of an ESKF implementation written in C++. Chapter 11 will conclude the thesis.

2 | Preliminaries

This chapter provides the necessary background theory leading up to the Kalman filter. There exists numerous textbooks on this theory, but this chapter mostly draws inspiration from [2] and [3], which put the probabilistic theory in a Bayesian context. Chapter 2.1 introduces the concept of Bayesian inference and its relevance to estimation problems, and gives the underlying model assumptions of the Kalman filter in the form of a Markov model. Chapter 2.2 gives a brief overview of the most important estimators related to the Kalman filter, with special emphasis on the *Bayes filter*, which will be later referenced in Chapter 3. Chapter 2.3 rounds off the preliminaries, giving some key properties of the multivariate Gaussian while outlining a fundamental identity that will be used to derive the Kalman filter.

2.1 Bayesian inference and Markov models

Probability gives some measure of the uncertainty connected to an event. The classical interpretation explains probabilities as relative frequencies from repeated experiments, such as a fraction of specific outcomes over all possible outcomes in the sample space. This way of interpreting probability assigns uncertainty to the experiment itself, which is representative if sufficient information about the experiment is given. This methodology will however fail when there is no way of gathering enough information to describe it in terms of relative frequencies. Questions such as “What is the probability of rain tomorrow?" or “Which of two athletes is most likely to win a 100-meter dash, given that one of the athletes sprained an ankle two months ago?" are difficult to answer with a frequentist approach. *Bayesian statistics* is a more suitable interpretation of probability in these cases, assigning beliefs to events. The main critique of this methodology

is its reliance on subjective measures of probability, such as the probability of an event prior to any observations. Its strength is however the ability to update belief based on observed data, shifting the prior belief in either direction. This methodology is extensively used in estimation problems where new observations continuously arrive, and the following chapters will lay the foundation for the mathematical tools utilized in this Bayesian way of doing statistical inference.

2.1.1 Total probability and Bayes' rule

In order to derive the Bayesian inference framework at the core of the Kalman filter, two fundamental laws in probability need to be stated for later reference. The first is the *law of total probability*, which provides a convenient way of deriving a probability distribution when the conditional distribution for the variable is available, including the distribution of the conditioning variable itself. For two random variables, X and Z, the marginal distribution of X can be given by total probability as

$$p(x) = \int p(x|z)p(z)\,\mathrm{d}z, \tag{2.1}$$

where $p(x)$ and $p(z)$ are the marginal distributions of X and Z, respectively, and $p(x|z)$ is the conditional distribution of X given Z. This applies also when conditioning on several other random variables, such that

$$p(x|w) = \int p(x|z,w)p(z|w)\,\mathrm{d}z, \tag{2.2}$$

in the case of some other random variable W.

The second fundamental law is *Bayes' rule*, laying the foundation for Bayesian inference, which gives a relation for how uncertainty changes in light of new observations,

$$p(x|z) = \frac{p(z|x)p(x)}{p(z)}. \tag{2.3}$$

Bayes' rule can also be utilized when conditioning on several variables, such that

$$p(x|z,w) = \frac{p(z|x,w)p(x|w)}{p(z|w)}, \tag{2.4}$$

in the case of some other random variable W. Equation (2.3) can be restated by noting that $p(z)$ in the denominator acts as a normalization constant, making

$p(x|z)$ integrate to 1. Thus, an alternative way of writing Bayes' rule is

$$p(x|z) \propto p(z|x)p(x). \tag{2.5}$$

The statistical properties of $p(x|z)$ are in other words encapsulated by the numerator in (2.3).

Lastly, both Bayes' rule and the law of total probability can be used as above when the probability distributions are multivariate, where $p(\boldsymbol{x})$ denotes the multivariate distribution for the random vector $\boldsymbol{x}$. This will mostly be the case in the remaining parts of the text.

2.1.2 The prior and posterior probability densities

In Bayesian inference, Bayes' rule is utilized by first assigning a distribution $p(x)$ to some random variable X, prior to any observations giving information about X. This is called the *a priori probability density function*, or the *prior* for short.

Observations giving information about X may manifest in some random variable Z. This relationship is given by the likelihood $p(z|x)$, which together with the prior gives the *a posteriori probability density function* $p(x|z)$ in (2.5), or the *posterior* for short. In practical terms, this is updating the belief about X by observing Z.

2.1.3 Markov models

Let $\boldsymbol{x} = [x_1, \cdots, x_n]^\mathsf{T}$ denote a state vector with n states that may evolve in time. The dependence between states over a span of time is in the general case not clear, where the state in one point in time may be dependent on one or several states at other points in time. The evolution of states in time is often modeled as a *Markov model*, or *Markov chain*, which is for many stochastic processes a reasonable representation for how the state evolves.

The essence of a Markov model lies in the *Markov assumption*. Given a system that is described by some state $\boldsymbol{x}$ at some point in time k, denoted $\boldsymbol{x}_k$, the Markov assumption is that the most recent state contains all information about previous states, such that information about these do not affect the distribution of how the state evolves in the future. This can be formulated as

$$p(\boldsymbol{x}_k|\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{k-1}) = p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}). \tag{2.6}$$

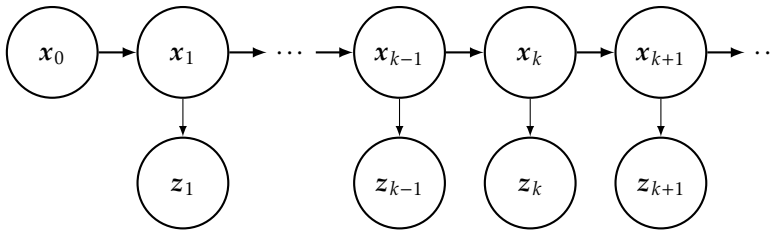


Figure 2.1: *Hidden Markov model with states x hidden in observations z.*

It is notationally convenient to define a joint collection of consecutive states in time from time step 0 up to k as $\boldsymbol{x}_{0:k} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$, such that the Markov assumption may be written $p(\boldsymbol{x}_k|\boldsymbol{x}_{0:k-1}) = p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$.

In the case of partially observable states, observed through some observation vector $\boldsymbol{z} = [z_1, \cdots, z_m]^\mathsf{T}$ with m observations, where observations are conditionally independent in time, the model takes the form of a *Hidden Markov Model (HMM)*. The true states are now hidden, and must be estimated by a chosen estimator. The assumed dependency structure in a HMM is shown as a Bayesian network in Figure 2.1, depicting the dependency between observations and states, and states themselves, in the form of a directed, acyclic graph. The edges connect dependent variables, and the direction tells what variable is dependent on which. The Markov assumption in (2.6) is thus illustrated by a single edge between states. The assumption of conditional independence between observations is given as

$$p(\boldsymbol{z}_k|\boldsymbol{x}_{0:k}, \boldsymbol{z}_{1:k-1}) = p(\boldsymbol{z}_k|\boldsymbol{x}_k), \tag{2.7}$$

and is also illustrated in Figure 2.1, where observations are conditionally independent given the corresponding state. The Markov assumption can be formulated in the same manner, where a future state is conditionally independent on past states, given the current state. This comes from the fact that variables connected together along a path with intermediate variables are independent of each other given information about any of the intermediate variables. Intuitively, this comes as a consequence of how information propagates between variables, as, for example, information about $\boldsymbol{x}_{k-1}$ propagates to $\boldsymbol{x}_k$, meaning $\boldsymbol{x}_k$ contains information about $\boldsymbol{x}_{k-1}$. The full posterior over all states is then given by

$$p(\boldsymbol{x}_{0:k}|\boldsymbol{z}_{1:k}) \propto p(\boldsymbol{x}_0)\prod_{i=1}^{k} p(\boldsymbol{z}_i|\boldsymbol{x}_i)p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1}). \tag{2.8}$$

Note that the state starts at time step 0, with the prior $p(\boldsymbol{x}_0)$. This is the initial distribution of $\boldsymbol{x}$, often given as some subjective measure of initial uncertainty, reflecting the Bayesian interpretation of probability. The power of conditional independence is clear from (2.8), as the number of required distributions for the full conditional distribution is greatly reduced. The transition model $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ and observation model $p(\boldsymbol{z}_k|\boldsymbol{x}_k)$ may, depending on the application, be reasonably modeled as time-invariant. If this is the case, the number of required distributions for full Bayesian estimation is reduced to these two PDFs, in addition to the prior $p(\boldsymbol{x}_0)$.

2.1.4 Sequential Bayesian estimation

The full posterior in (2.8) is rarely of interest due to the computational costs of calculating a full joint distribution for each time step, and is in many applications not of interest in itself. Bayesian inference is more commonly concerned in the marginal distributions acquired by *filtering*, *smoothing* and *prediction*.

Filtering denotes the task of retrieving the marginal distribution of the *current* state, $\boldsymbol{x}_k$, given current and previous observations, $\boldsymbol{z}_{1:k}$, thus giving $p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k})$. This method is most applicable when only the most recent state is to be further used, and the previous states are disregarded as they are currently outdated. This is often the case for real-time systems with high demand on computational speed.

Smoothing denotes the task of retrieving the marginal distribution of a *past* state, $\boldsymbol{x}_{k-T}$ for some previous time step $k-T$, given current and previous observations, thus giving $p(\boldsymbol{x}_{k-T}|\boldsymbol{z}_{1:k})$. This differs from filtering in the sense that past states are now of interest. The previous estimates are updated by new observations, in effect giving better estimates, which may be important in applications where the aim is an estimate of the whole trajectory of states. This has the drawback of an additional computational cost in updating previous estimates, which may for many real-time applications be unfeasible.

Prediction denotes the task of retrieving the marginal distribution of a *future* state, $\boldsymbol{x}_{k+T}$ for some future time step $k+T$, given current and previous observations, thus giving $p(\boldsymbol{x}_{k+T}|\boldsymbol{z}_{1:k})$. The main reason for applying a prediction is to propagate the system when observations currently are unavailable, such that the predicted estimates represent the state until new observations become available

again. The transition model, $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$, is thus solely used to predict new states.

The use of these methods is dependent on the application at hand and on the marginal distribution of interest. Smoothing will in general give better estimates, but with the requirement of previous states being of interest. This will not be the case in the upcoming chapters, so the remaining discussion on sequential Bayesian estimation will be limited to filtering and prediction.

2.2 Estimators

The prior and posterior distributions, discussed in previous chapters, may not necessarily be of main interest in themselves. In most practical applications, retrieving best possible *estimates* of the states is the end goal. An estimate is said to be the output of an *estimator*. In general, an estimator attempts to estimate a parameter of a distribution when given samples from the distribution, giving the estimate $\hat{\boldsymbol{x}}$. In the framework of a HMM, best possible estimates of the state is desired. The following chapter briefly outlines some important estimators which serve as background information in later parts of the thesis.

2.2.1 Maximum a posteriori estimation

The maximum a posterori (MAP) estimator is closely related to the Bayesian methodology presented in Chapter 2.1, aiming to maximize the posterior

$$\hat{\boldsymbol{x}}_{\text{MAP}} = \underset{x}{\operatorname{argmax}}\, p(\boldsymbol{x}|\boldsymbol{z}) = \underset{x}{\operatorname{argmax}}\, p(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x}), \tag{2.9}$$

where the second equality follows from Bayes' rule, with the denominator disappearing in the maximization with respect to $\boldsymbol{x}$. The parameter is here a random variable in itself, with its own prior distribution $p(\boldsymbol{x})$. In cases with few observations the MAP estimator will rely more on the prior, and in cases with many observations it will rely more on the likelihood. It will later be seen that this type of weighting between the prior and likelihood mirrors that of the *Bayes filter*, and later the Kalman filter.

2.2.2 Minimum mean square error estimation

The minimum mean square error (MMSE) estimator minimizes the mean squared error (MSE), defined as

$$\hat{\boldsymbol{x}}_{\text{MMSE}} = \underset{\hat{\boldsymbol{x}}}{\operatorname{argmin}} \, \mathrm{E}\left[\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|_2^2\right] = \mathrm{E}[\boldsymbol{x}|\boldsymbol{z}], \tag{2.10}$$

with a proof of the last equality in [2]. By the *law of total expectation*, $\mathrm{E}[X] = \mathrm{E}[\mathrm{E}[X|Z]]$, the MMSE estimator is seen to be unbiased, a highly desired property of an estimator. The conditional expectation in (2.10) may in general not have a closed-form solution, so a linear estimator of the form $\hat{\boldsymbol{x}} = \mathbf{A}\boldsymbol{z} + \boldsymbol{b}$ may be a necessary approximation in the nonlinear case. The expectation and covariance of this linear mean square error (LMMSE) is given, with a proof in [2], by

$$\hat{\boldsymbol{x}}_{\text{LMMSE}} = \boldsymbol{\mu}_{\boldsymbol{x}} + \Sigma_{\boldsymbol{x},\boldsymbol{z}}\Sigma_{\boldsymbol{z}}^{-1}(\boldsymbol{z} - \boldsymbol{\mu}_{\boldsymbol{z}}), \tag{2.11a}$$

$$\Sigma_{\text{LMMSE}} = \Sigma_{\boldsymbol{x}} - \Sigma_{\boldsymbol{x},\boldsymbol{z}}\Sigma_{\boldsymbol{z}}^{-1}\Sigma_{\boldsymbol{x},\boldsymbol{z}}^{\mathrm{T}}, \tag{2.11b}$$

where $\boldsymbol{\mu}$ and Σ denote the expectation and covariance of the variable(s) in subscript, respectively. The LMMSE estimator is here highlighted due to its strong resemblance to the later derived Kalman filter equations, and can indeed be used to derive these equations by means of MSE minimization in a linear model with Gaussian random variables. Both the MAP and MMSE estimators will later be referenced when discussing properties of the Kalman filter, but the filter itself will here be introduced in a more Bayesian context.

2.2.3 The Bayes filter

The states in a HMM, such as the one depicted in Figure 2.1, are not directly observable and must be estimated. The methods previously discussed may then be deployed to retrieve estimates of $\boldsymbol{x}$. As the states propagate in time, new estimates must continuously be generated. A filter, as discussed in Chapter 2.1.4, may then be devised to iteratively update the estimate as time passes and new observations are available. This is particularly useful in cases where the Markov assumption applies, as all information about previous states are encapsulated in the current state. This is effectively done in the *Bayes filter*, which gives the general equations for filtering in the Bayesian framework.

The desired PDF is the posterior $p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k})$. By applying Bayes' rule and conditional independence on observations, the posterior can be solved as

$$p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}) = \frac{p(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{z}_{1:k-1})p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1})}{p(\boldsymbol{z}_k|\boldsymbol{z}_{1:k-1})}$$
$$= \frac{p(\boldsymbol{z}_k|\boldsymbol{x}_k)p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1})}{p(\boldsymbol{z}_k|\boldsymbol{z}_{1:k-1})}. \tag{2.12}$$

The observation model $p(\boldsymbol{z}_k|\boldsymbol{x}_k)$ is recognized in (2.12), and is normally assumed known. The factor $p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1})$ is a prediction of the current state, given all previous observations, while $p(\boldsymbol{z}_k|\boldsymbol{z}_{1:k-1})$ is a prediction of the current observation. Again, the PDF $p(\boldsymbol{z}_k|\boldsymbol{z}_{1:k-1})$ is not a function of $\boldsymbol{x}$, meaning it only has the effect of normalizing the posterior such that (2.12) can be written

$$p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}) \propto p(\boldsymbol{z}_k|\boldsymbol{x}_k)p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}). \tag{2.13}$$

In order to acquire $p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1})$, which here corresponds to the prior introduced in Chapter 2.1.2, the Markov assumption and (2.2) can be used as

$$p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}) = \int p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{z}_{1:k-1})p(\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1})\,\mathrm{d}\boldsymbol{x}_{k-1}$$
$$= \int p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})p(\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1})\,\mathrm{d}\boldsymbol{x}_{k-1}. \tag{2.14}$$

The transition model $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ is recognized in (2.14), and is, as with the observation model, normally assumed known. With the presence of $p(\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1})$, the posterior of the previous time step, the recursive nature of the Bayes filter is apparent. This recursion terminates at the initial prior, $p(\boldsymbol{x}_0)$, which is given as a subjective measure of initial uncertainty. Together, (2.12) and (2.14) make up the *update* and *prediction* steps of the Bayes filter, respectively. These equations are thus used in an iterative manner from the initial prior, with a prediction and update for each time step.

The importance of the Bayes filter should be emphasized. This is filtering for HMMs in the most general case, and constitutes the equations at the core of any applied filtering method in this framework. It will indeed later be shown that this is the underlying structure of the Kalman filter. Two observations should however be highlighted. First, the numerator in (2.12) may generally not result in any known PDF, and may therefore not give any obvious and easily acquirable optimal estimates. Secondly, the product in (2.14) may generally not result in

any analytically integrable function, and solutions such as nonparameteric approximations have to be resorted to. These drawbacks are especially significant for real-time estimation problems, where tedious computations may render the estimation algorithm unfit. Closed-form solutions to these equations are thus of interest, which is the main motivation for Chapter 3.

2.3 The multivariate Gaussian distribution

The multivariate Gaussian distribution proves a fundamental cornerstone to much of estimation theory by virtue to its many unique properties, and has an indispensable role in the Kalman filter framework for this reason. The following chapter intends to summarize these properties for later reference.

2.3.1 Manipulating the Gaussian probability density function

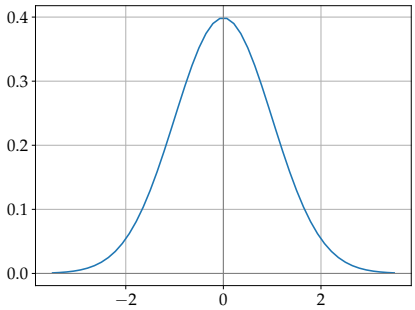
The general, multivariate PDF of the Gaussian distribution is given as the exponential of a quadratic polynomial according to

$$\begin{aligned} p(\boldsymbol{x}) &= \frac{1}{(2\pi)^{n/2}\left|\mathbf{P}\right|^{1/2}} \exp\left(-\frac{1}{2}\left(\boldsymbol{x}-\boldsymbol{\mu}\right)^{\mathsf{T}}\mathbf{P}^{-1}\left(\boldsymbol{x}-\boldsymbol{\mu}\right)\right) \\ &\equiv \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu},\mathbf{P}), \end{aligned} \tag{2.15}$$

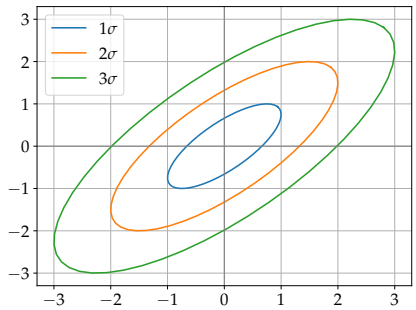
where $\boldsymbol{\mu}$ and $\mathbf{P}$ are the expectation and covariance of the random vector $\boldsymbol{x}$, respectively, and n is the dimension of $\boldsymbol{x}$. See Figure 2.2 for examples of the Gaussian distribution. The quadratic polynomial

$$\left(\boldsymbol{x}-\boldsymbol{\mu}\right)^{\mathsf{T}}\mathbf{P}^{-1}\left(\boldsymbol{x}-\boldsymbol{\mu}\right) = q(\boldsymbol{x}) \tag{2.16}$$

is called the Gaussian's *quadratic form*. As the exponential function is a *monotonic function*, it is also injective, meaning that all mappings between such a quadratic function and its corresponding exponential form is one-to-one. In addition, under the normalization constraint that all PDFs have to integrate to 1, this exponential is proportional to one, and only one, Gaussian. Hence, for all quadratic forms such as (2.16), *there is a unique, corresponding Gaussian*. The implication of this is that when analyzing properties of the Gaussian, one only has to consider the quadratic form.



(a) Plot of the standard Gaussian distribution.



(b) Sigma ellipses of a multivariate Gaussian of two random variables. Each ellipse is a level curve of the PDF when the corresponding quadratic form is equal to either 1σ, 2σ or 3σ. The plot is made with $\boldsymbol{\mu} = [0 \quad 0]^\mathsf{T}$ *and* $\mathbf{P} = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1 \end{bmatrix}$.

Figure 2.2: *Different figures displaying the famous Gaussian distribution.*

Linear transformation of Gaussians

Given a multivariate Gaussian random vector $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{P})$ and another random vector $\boldsymbol{y} = \mathbf{A}\boldsymbol{x} + \boldsymbol{b}$, $\boldsymbol{y}$ is also multivariate Gaussian and distributed according to

$$\boldsymbol{y} \sim \mathcal{N}(\mathbf{A}\boldsymbol{\mu} + \boldsymbol{b}, \mathbf{A}\mathbf{P}\mathbf{A}^\mathsf{T}), \tag{2.17}$$

as shown in [4].

The marginal and conditional distribution of a Gaussian

From the quadratic form of joint Gaussian distribution, its marginal and conditional distributions can be found in closed form. Pay special note to the remarkable fact that *both of these distributions are in fact new Gaussians.*

Given a Gaussian $p(\boldsymbol{x}) = p(\boldsymbol{x}_1, \boldsymbol{x}_2)$ composed of two random vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, where $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{P})$ and with quadratic form given by

$$q(\boldsymbol{x}) = \left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} - \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix} \right)^\mathsf{T} \begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} \\ \mathbf{P}_{21} & \mathbf{P}_{22} \end{bmatrix}^{-1} \left(\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} - \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix} \right), \tag{2.18}$$

the marginal distribution $p(\boldsymbol{x}_1)$ is a new Gaussian with quadratic form given by

$$q(\boldsymbol{x}_1) = \left(\boldsymbol{x}_1 - \boldsymbol{\mu}_1\right)^\mathsf{T} \mathbf{P}_{11}^{-1} \left(\boldsymbol{x}_1 - \boldsymbol{\mu}_1\right). \tag{2.19}$$

This result follows by considering x_1 as a linear transformation of the original random vector according to

$$x_1 = \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{O} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\mathbf{0}}_{b} = \mathbf{A}x + b,$$

where $\mathbf{O}$ is the zero matrix, and applying (2.17).

The conditional distribution $p(x_1|x_2)$ is also a new Gaussian with quadratic form given by

$$q(x_1|x_2) = \left(x_1 - \mu_{1|2}\right)^{\mathsf{T}} \mathbf{P}_{1|2}^{-1} \left(x_1 - \mu_{1|2}\right), \tag{2.20}$$

where

$$\mu_{1|2} = \mu_1 + \mathbf{P}_{12}\mathbf{P}_{22}^{-1}(x_2 - \mu_2), \tag{2.21a}$$

$$\mathbf{P}_{1|2} = \mathbf{P}_{11} - \mathbf{P}_{12}\mathbf{P}_{22}^{-1}\mathbf{P}_{21}. \tag{2.21b}$$

This result, however, is nontrivial to derive, although in principal straight forward, as it contains nontrivial algebraic manipulations. The complete proof can be found in [5]. The result in (2.21) can be seen to be identical with that of the LMMSE estimator in (2.11), showing that the conditional distribution inherits unbiasedness.

2.3.2 The product of multivariate Gaussian distributions

If all PDFs that appear in the Bayes filter presented in Chapter 2.2.3 are assumed multivariate Gaussian, inspecting the product of two such distributions is inevitable, as both the prediction and update in the Bayes filter involves products of PDFs. It should again be emphasized that a closed-form solution to the prediction and update steps of Bayes filter is of main interest. Assuming that the involved PDFs are Gaussian, the Bayes filter can be further investigated to see if the assumption of Gaussianity gives a solution. A key identity, which will be expanded on later,

$$\mathcal{N}(z|x)\mathcal{N}(x) = \mathcal{N}(x|z)\mathcal{N}(z) \tag{2.22}$$

will be assumed to hold, with the parameters of the distributions dropped for notational simplicity. It is not apparent at this point why this must hold, and why

it is of interest to the Bayes filter, but applying the assumption of (2.22) in the Bayes filter will now be seen to give a closed-form solution.

Before delving further, it is paramount to properly understand that (2.22) is not implied from Bayes' rule in (2.3). While it is true in general that Bayes' rule allows for swapping around the involved variables, *there is no assumption about what form the resulting PDFs have after swapping.* To be assured that the product of two Gaussian PDFs is indeed the product of two new Gaussian PDFs would therefore be a significantly simplifying fact.

In order to apply (2.22) to the Bayesian prediction in (2.14), (2.14) must be written in a form compatible with (2.22). The first step is to apply the Markov assumption discussed in Chapter 2.1.3 in reverse by reintroducing $\boldsymbol{z}_{1:k-1}$ and writing

$$\begin{aligned} p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}) &= \int p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) p(\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1}) \,\mathrm{d}\boldsymbol{x}_{k-1} \\ &= \int p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{z}_{1:k-1}) p(\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1}) \,\mathrm{d}\boldsymbol{x}_{k-1}. \end{aligned}$$

The second step is to recall from (2.2) that the law of total probability still holds when all involved PDFs are conditioned on the same variable, in this case $\boldsymbol{z}_{1:k-1}$. Thus, the prediction step, under the assumption that (2.22) holds and with underbraces indicating the corresponding variable in (2.22), becomes

$$\begin{aligned} p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}) &= \int \mathcal{N}(\underbrace{\boldsymbol{x}_k}_{z} | \underbrace{\boldsymbol{x}_{k-1}}_{x}, \boldsymbol{z}_{1:k-1}) \mathcal{N}(\underbrace{\boldsymbol{x}_{k-1}}_{x} | \boldsymbol{z}_{1:k-1}) \,\mathrm{d}\boldsymbol{x}_{k-1} \\ &= \int \mathcal{N}(\underbrace{\boldsymbol{x}_{k-1}}_{x} | \underbrace{\boldsymbol{x}_k}_{z}, \boldsymbol{z}_{1:k-1}) \mathcal{N}(\underbrace{\boldsymbol{x}_k}_{z} | \boldsymbol{z}_{1:k-1}) \,\mathrm{d}\boldsymbol{x}_{k-1} \\ &= \mathcal{N}(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}) \int \mathcal{N}(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k, \boldsymbol{z}_{1:k-1}) \,\mathrm{d}\boldsymbol{x}_{k-1} \\ &= \mathcal{N}(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}) \end{aligned}$$

where the last equality is a result of the fact that $\int \mathcal{N}(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k) \,\mathrm{d}\boldsymbol{x}_{k-1} = 1$ as $\mathcal{N}(\boldsymbol{x}_{k-1}|\boldsymbol{x}_k)$ is a proper PDF.

Similarly for the posterior, by again reintroducing $\boldsymbol{z}_{1:k-1}$ by applying the Markov assumption in reverse and writing $p(\boldsymbol{z}_k|\boldsymbol{x}_k) = p(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{z}_{1:k-1})$, such that

(2.12) becomes

$$\begin{aligned} p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}) &= \frac{p(\boldsymbol{z}_k|\boldsymbol{x}_k)p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1})}{p(\boldsymbol{z}_k|\boldsymbol{z}_{1:k-1})} \\ &= \frac{p(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{z}_{1:k-1})p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1})}{p(\boldsymbol{z}_k|\boldsymbol{z}_{1:k-1})} \end{aligned}$$

and recalling that Bayes' rule in (2.4) still holds when all PDFs are conditioned on the same variable. By now inserting (2.22), this manipulation then yields

$$\begin{aligned} p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}) &= \frac{\mathcal{N}(\overbrace{\boldsymbol{z}_k}^{z}|\overbrace{\boldsymbol{x}_k}^{x}, \boldsymbol{z}_{1:k-1})\mathcal{N}(\overbrace{\boldsymbol{x}_k}^{x}|\boldsymbol{z}_{1:k-1})}{\mathcal{N}(\underbrace{\boldsymbol{z}_k}_{z}|\boldsymbol{z}_{1:k-1})} \\ &= \frac{\mathcal{N}(\overbrace{\boldsymbol{x}_k}^{x}|\overbrace{\boldsymbol{z}_k}^{z}, \boldsymbol{z}_{1:k-1})\mathcal{N}(\overbrace{\boldsymbol{z}_k}^{z}|\boldsymbol{z}_{1:k-1})}{\mathcal{N}(\underbrace{\boldsymbol{z}_k}_{z}|\boldsymbol{z}_{1:k-1})} \\ &= \mathcal{N}(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}). \end{aligned}$$

As seen, the identity in (2.22) proved to give closed-form solutions to the prediction and update of the Bayes filter, in the form of two new Gaussians.

The product identity

It now remains to determine whether, and if so, when, (2.22) actually holds. Indeed, *if, and only if, $\boldsymbol{z}$ and $\boldsymbol{x}$ are linearly related, then* (2.22) *does hold*. This result appears frequently in literature on Bayesian inference and target tracking under different names ([2], [6]–[10]) and will for this thesis be referred to as the *product identity*. The following theorem with proof draws inspiration from [2].

Theorem 1 (The product identity)**.** *Given two multivariate Gaussian random vectors* $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_x, \mathbf{P}_x)$ *and* $\boldsymbol{z}|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{z}; \mathbf{H}\boldsymbol{x}, \mathbf{R})$ *that are linearly related. More explicitly,*

$$\boldsymbol{z} = \mathbf{H}\boldsymbol{x} + \boldsymbol{w}$$

for some $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$*. Then the identity*

$$\mathcal{N}(\boldsymbol{z}; \mathbf{H}\boldsymbol{x}, \mathbf{R})\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_x, \mathbf{P}_x) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\lambda}_z, \mathbf{S})\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{x|z}, \mathbf{P}_{x|z}) \tag{2.23}$$

holds, where

$$\boldsymbol{\lambda}_z = \mathbf{H}\boldsymbol{\mu}_x, \tag{2.24a}$$
$$\mathbf{S} = \mathbf{H}\mathbf{P}_x\mathbf{H}^\mathsf{T} + \mathbf{R}, \tag{2.24b}$$
$$\mathbf{W} = \mathbf{P}_x\mathbf{H}^\mathsf{T}\mathbf{S}^{-1}, \tag{2.24c}$$
$$\boldsymbol{\mu}_{x|z} = \boldsymbol{\mu}_x + \mathbf{W}(\boldsymbol{z} - \boldsymbol{\lambda}_z), \tag{2.24d}$$
$$\mathbf{P}_{x|z} = (\mathbf{I} - \mathbf{W}\mathbf{H})\mathbf{P}_x(\mathbf{I} - \mathbf{W}\mathbf{H})^\mathsf{T} + \mathbf{W}\mathbf{R}\mathbf{W}^\mathsf{T}. \tag{2.24e}$$

Proof. Notice first that the involved PDFs in (2.23) can be written in the more general form

$$p(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x}) = p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z}).$$

The parameters of $p(\boldsymbol{z})$ and $p(\boldsymbol{x}|\boldsymbol{z})$ are found by first combining the quadratic forms of $p(\boldsymbol{x})$ and $p(\boldsymbol{z}|\boldsymbol{x})$ into the quadratic form of the joint PDF $p(\boldsymbol{x}, \boldsymbol{z})$. Summing the two quadratic forms is a result of multiplying the two exponential functions in $p(\boldsymbol{x})$ and $p(\boldsymbol{z}|\boldsymbol{x})$. Combining the quadratic forms is done by

$$\begin{aligned}
&(\boldsymbol{x} - \boldsymbol{\mu}_x)^\mathsf{T}\mathbf{P}_x^{-1}(\boldsymbol{x} - \boldsymbol{\mu}) + (\boldsymbol{z} - \mathbf{H}\boldsymbol{x})^\mathsf{T}\mathbf{R}^{-1}(\boldsymbol{z} - \mathbf{H}\boldsymbol{x}) \\
&= \begin{bmatrix} \boldsymbol{x} - \boldsymbol{\mu}_x \\ \boldsymbol{z} - \mathbf{H}\boldsymbol{x} \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{P}_x^{-1} & \mathbf{O} \\ \mathbf{O} & \mathbf{R}^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} - \boldsymbol{\mu} \\ \boldsymbol{z} - \mathbf{H}\boldsymbol{x} \end{bmatrix} \\
&= \left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\mu}_x \\ \mathbf{H}\boldsymbol{x} \end{bmatrix} \right)^\mathsf{T} \begin{bmatrix} \mathbf{P}_x^{-1} & \mathbf{O} \\ \mathbf{O} & \mathbf{R}^{-1} \end{bmatrix} \left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\mu}_x \\ \mathbf{H}\boldsymbol{x} \end{bmatrix} \right)
\end{aligned} \tag{2.25}$$

The quadratic form in (2.25) is not a proper quadratic form in its current state as $\boldsymbol{x}$ appears as both a variable and expectation in the vector part. However, as the relation between $\boldsymbol{z}$ and $\boldsymbol{x}$ is linear, this can be remedied by the linear

transformation

$$\begin{bmatrix} \mathbf{I} & \mathbf{O} \\ -\mathbf{H} & \mathbf{I} \end{bmatrix} \left(\begin{bmatrix} x \\ z \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mathbf{H}\mu_x \end{bmatrix} \right) = \left(\begin{bmatrix} x \\ z \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mathbf{H}x \end{bmatrix} \right)$$

that is substituted into (2.25). Note that this step would not be possible if z and x were not linearly related. The quadratic form (2.25) can then be further manipulated into the proper quadratic form

$$\begin{aligned}
&\left(\begin{bmatrix} \mathbf{I} & \mathbf{O} \\ -\mathbf{H} & \mathbf{I} \end{bmatrix} \left(\begin{bmatrix} x \\ z \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mathbf{H}\mu_x \end{bmatrix} \right) \right)^{\mathsf{T}} \begin{bmatrix} \mathbf{P}_x^{-1} & \mathbf{O} \\ \mathbf{O} & \mathbf{R}^{-1} \end{bmatrix} \left(\begin{bmatrix} \mathbf{I} & \mathbf{O} \\ -\mathbf{H} & \mathbf{I} \end{bmatrix} \left(\begin{bmatrix} x \\ z \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mathbf{H}\mu_x \end{bmatrix} \right) \right) \\
&= \left(\begin{bmatrix} x \\ z \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mathbf{H}\mu_x \end{bmatrix} \right)^{\mathsf{T}} \begin{bmatrix} \mathbf{I} & -\mathbf{H}^{\mathsf{T}} \\ \mathbf{O} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{P}_x^{-1} & \mathbf{O} \\ \mathbf{O} & \mathbf{R}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{O} \\ -\mathbf{H} & \mathbf{I} \end{bmatrix} \left(\begin{bmatrix} x \\ z \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mathbf{H}\mu_x \end{bmatrix} \right) \\
&= \left(\begin{bmatrix} x \\ z \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mathbf{H}\mu_x \end{bmatrix} \right)^{\mathsf{T}} \left(\begin{bmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{H} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{P}_x & \mathbf{O} \\ \mathbf{O} & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{H}^{\mathsf{T}} \\ \mathbf{O} & \mathbf{I} \end{bmatrix} \right)^{-1} \left(\begin{bmatrix} x \\ z \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mathbf{H}\mu_x \end{bmatrix} \right) \\
&= \left(\begin{bmatrix} x \\ z \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mathbf{H}\mu_x \end{bmatrix} \right)^{\mathsf{T}} \begin{bmatrix} \mathbf{P}_x & \mathbf{P}_x\mathbf{H}^{\mathsf{T}} \\ \mathbf{H}\mathbf{P}_x & \mathbf{H}\mathbf{P}_x\mathbf{H}^{\mathsf{T}} + \mathbf{R} \end{bmatrix}^{-1} \left(\begin{bmatrix} x \\ z \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mathbf{H}\mu_x \end{bmatrix} \right),
\end{aligned} \tag{2.26}$$

where the matrix inversion identities [11]

$$\begin{bmatrix} \mathbf{A} & \mathbf{O} \\ \mathbf{O} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{O} \\ \mathbf{O} & \mathbf{D}^{-1} \end{bmatrix},$$

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{O} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{D}^{-1} \\ \mathbf{O} & \mathbf{D}^{-1} \end{bmatrix}$$

and the transpose of a block matrix

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{\mathsf{T}} = \begin{bmatrix} \mathbf{A}^{\mathsf{T}} & \mathbf{C}^{\mathsf{T}} \\ \mathbf{B}^{\mathsf{T}} & \mathbf{D}^{\mathsf{T}} \end{bmatrix}$$

are used.

From (2.26), the parameters of the marginal distribution $p(z) = \mathcal{N}(z; \lambda_z, \mathbf{S})$ and the conditional distribution $p(x|z) = \mathcal{N}(x; \mu_{x|z}, \mathbf{P}_{x|z})$ are then found by (2.19) and (2.20), respectively. ■

A remark about (2.24e) is in order. The covariance matrix $\mathbf{P}_{x|z}$ is here intro-

duced in its *Joseph form* [12]. Another common form used in literature is

$$\mathbf{P}_{x|z} = (\mathbf{I} - \mathbf{W}\mathbf{H})\,\mathbf{P}_x, \tag{2.27}$$

which is also the form one arrives at when applying (2.20) directly. These two forms are indeed mathematically identical, given that $\mathbf{W}$ is defined as in (2.24c) [13]. However, from a computational standpoint, (2.27) has poor numerical stability, where matrix properties like positive definiteness and symmetry are not guaranteed. The Joseph form in (2.24e), on the other hand, does guarantee these properties [14], and will exclusively be used.

II

The Kalman filter

3 | The linear Kalman filter

Chapter 2 introduced some of the central concepts moving forward, now leading up to the *Kalman filter (KF)*. The Kalman filter is nothing more than a special case of the Bayes filter introduced in Chapter 2.2.3, and aims to solve it analytically by invoking additional assumptions on the distributions in the Bayes filter. More specifically, the Kalman filter assumes that the transition and observation models are *linear* and with *additive, Gaussian* noise.

The linear Gaussian assumption will first be expanded on in Chapter 3.1, describing how the Kalman filter gives a solution to the Bayes filter. Chapter 3.2 will further give explicit forms on the transition and observation model, while Chapter 3.3 solves the Bayes filter in a linear Gaussian model.

3.1 The linear Gaussian assumption

The underlying model for the Kalman filter is, as for the Bayes filter, a HMM, where the posterior $p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k})$ is of interest. Chapter 2.2.3 gave the general equations for solving this posterior, but saw that these equations are not guaranteed closed-form solutions due to the products of the involved PDFs. In Chapter 2.3.2, the product of multivariate Gaussian distributions were investigated, giving closed form solutions to the Bayes filter equations when the model is assumed linear and Gaussian. Equally important, the resulting PDFs proved to be new Gaussians, ensuring that the Bayes filter deals with Gaussians over all time-steps. Finally, the product identity was presented, a key result giving the parameters of the new Gaussians, which will here be used when deriving the Kalman filter equations.

The product identity, as presented in Chapter 2.3.2, made the assumption of two linearly related variables $\boldsymbol{x}$ and $\boldsymbol{z}$, with an additive Gaussian component $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{R})$. These assumptions alone are what turns the Bayes filter into the

Kalman filter, and is here explicitly pointed out to emphasize that the Kalman filter really is a special case of the Bayes filter given the added assumptions. The remaining part is then to add the assumptions to the HMM, giving the transition and observation models, $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ and $p(\boldsymbol{z}_k|\boldsymbol{x}_k)$, and apply the product identity to the Bayes filter equations.

3.2 The state-space model

Due to the Kalman filter's origin in control theory, the previously described HMM is in most literature on the Kalman filter denoted as a *state-space model*. In the context of control theory, the state-space model does not necessarily involve any random variables, but will often do so when taking into account model uncertainties, which will be the case here. The transition model is in the state-space formulation denoted as the *process model*, while the observation model is denoted as the *measurement model*. The sole purpose of this change of naming is to coincide with the literature on Kalman filters, but the model and assumptions remain the same as for the HMM.

3.2.1 The process model

Under the linear Gaussian assumption, the process model $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ takes the form

$$\boldsymbol{x}_k = \mathbf{F}_{k-1}\boldsymbol{x}_{k-1} + \boldsymbol{v}_{k-1}, \tag{3.1}$$

where $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k-1}$ denote the state at time steps k and $k-1$, respectively, $\mathbf{F}_{k-1}$ is the *process matrix* at time step $k-1$ and $\boldsymbol{v}_{k-1}$ is additive process noise at time step $k-1$. From Chapter 3.1, $\boldsymbol{v}_{k-1}$ is assumed to be Gaussian white noise. Thus, it is distributed as,

$$\boldsymbol{v}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1}),$$

and uncorrelated across time, meaning,

$$\mathrm{E}\left[\boldsymbol{v}_k \boldsymbol{v}_l^{\mathsf{T}}\right] = \mathbf{O}, \quad k \neq l,$$

where $\mathbf{O}$ is the zero matrix.

The PDF $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ can be derived directly from (3.1). When $\boldsymbol{v}_{k-1}$ is assumed Gaussian and the PDF is conditioned on $\boldsymbol{x}_{k-1}$, $\boldsymbol{x}_k$ is just a linear transformation

of $\boldsymbol{v}_{k-1}$, which is a new Gaussian. The expectation is found to be $\mathbf{F}_{k-1}\boldsymbol{x}_{k-1}$ and its covariance $\mathbf{Q}_{k-1}$, giving

$$p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) = \mathcal{N}(\boldsymbol{x}_k; \mathbf{F}_{k-1}\boldsymbol{x}_{k-1}, \mathbf{Q}_{k-1}). \tag{3.2}$$

3.2.2 The measurement model

Under the linear Gaussian assumption, the measurement model $p(\boldsymbol{z}_k|\boldsymbol{x}_k)$ takes the form

$$\boldsymbol{z}_k = \mathbf{H}_k\boldsymbol{x}_k + \boldsymbol{w}_k, \tag{3.3}$$

where $\boldsymbol{z}_k$ is the received measurement at time step k, $\mathbf{H}_k$ is the *measurement matrix* at time step k, $\boldsymbol{x}_k$ the state of the system at time step k and $\boldsymbol{w}_k$ some additive measurement noise at time step k. Similarly to $\boldsymbol{v}_k$, $\boldsymbol{w}_k$ is assumed to be Gaussian white noise. Thus, it is distributed as

$$\boldsymbol{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_k),$$

and uncorrelated across time, meaning

$$\mathrm{E}\left[\boldsymbol{w}_k\boldsymbol{w}_l^{\mathsf{T}}\right] = \mathbf{O}, \quad k \neq l,$$

where $\mathbf{O}$ is the zero matrix.

The PDF $p(\boldsymbol{z}_k|\boldsymbol{x}_k)$ can similarly to (3.2) be derived directly from (3.3), as $\boldsymbol{z}_k$, conditioned on $\boldsymbol{x}_k$, is just a linear transformation of $\boldsymbol{w}_k$, resulting in $\boldsymbol{z}_k$ being Gaussian. Thus, taking the expectation and covariance of (3.3) shows that

$$p(\boldsymbol{z}_k|\boldsymbol{x}_k) = \mathcal{N}(\boldsymbol{z}_k; \mathbf{H}_k\boldsymbol{x}_k, \mathbf{R}_k). \tag{3.4}$$

3.3 The Kalman filter algorithm

With the process and measurement models given specific forms in Chapter 3.2, the Bayes filter equations may be solved such that the posterior $p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k})$ can be given. The Bayes filter was in Chapter 2.2.3 divided into a prediction and an update step, which will also be the case for the Kalman filter, but with an added intermediate step called the innovation step.

3.3.1 Initialization and notation

First, the filter is initialized with some initial, known state $\boldsymbol{x}_0$ and state covariance $\mathbf{P}_0$, giving the Gaussian prior $p(\boldsymbol{x}_0)$, which is only valid until the first measurement arrives. The PDF $p(\boldsymbol{x}_0)$ is used to make the prediction of $\boldsymbol{x}_1$ with expectation $\boldsymbol{\mu}_{1|0}$. Note the used notation. Separating the expectation of the prior at time step k is done with $\boldsymbol{\mu}_{k|k-1}$ denoting predicted $\boldsymbol{x}_k$,

$$\boldsymbol{\mu}_{k|k-1} = \mathrm{E}\left[\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}\right],$$

while $\boldsymbol{\mu}_{k|k}$ denotes the expectation of the posterior,

$$\boldsymbol{\mu}_{k|k} = \mathrm{E}\left[\boldsymbol{x}_k|\boldsymbol{z}_{1:k}\right],$$

with its updated $\boldsymbol{x}_k$ estimate. The same applies for the covariance matrix, where $\mathbf{P}_{k|k-1}$ is the covariance of the prior at time step k,

$$\mathbf{P}_{k|k-1} = \mathrm{Cov}\left[\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}\right],$$

and $\mathbf{P}_{k|k}$ is the covariance of the posterior at time step k,

$$\mathbf{P}_{k|k} = \mathrm{Cov}\left[\boldsymbol{x}_k|\boldsymbol{z}_{1:k}\right].$$

3.3.2 The prediction step

With the assumptions of the state-space model in Chapter 3.2, the involved PDFs in (2.14) are given by the process model (3.2) and the posterior of the previous time step

$$p(\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1}) = \mathcal{N}(\boldsymbol{x}_{k-1}; \boldsymbol{\mu}_{k-1|k-1}, \mathbf{P}_{k-1|k-1}), \tag{3.5}$$

which inductively is assumed Gaussian. As will be seen in the update step, the resulting posterior remains Gaussian. Thus, one only needs to initialize the filter with a Gaussian prior $p(\boldsymbol{x}_0)$ to be assured that the prior and posterior remain Gaussian at all times.

Using the product identity (2.23), the predicted prior may be solved as

$$
\begin{aligned}
p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}) &= \int \mathcal{N}(\boldsymbol{x}_k; \mathbf{F}_{k-1}\boldsymbol{x}_{k-1}, \mathbf{Q}_{k-1})\mathcal{N}(\boldsymbol{x}_{k-1}; \boldsymbol{\mu}_{k-1|k-1}, \mathbf{P}_{k-1|k-1})\,\mathrm{d}\boldsymbol{x}_{k-1} \\
&= \int \mathcal{N}(\boldsymbol{x}_k; \boldsymbol{\mu}_{k|k-1}, \mathbf{P}_{k|k-1})\mathcal{N}(\boldsymbol{x}_{k-1}; \bullet, \bullet)\,\mathrm{d}\boldsymbol{x}_{k-1} \\
&= \mathcal{N}(\boldsymbol{x}_k; \boldsymbol{\mu}_{k|k-1}, \mathbf{P}_{k|k-1}) \int \mathcal{N}(\boldsymbol{x}_{k-1}; \bullet, \bullet)\,\mathrm{d}\boldsymbol{x}_{k-1} \\
&= \mathcal{N}(\boldsymbol{x}_k; \boldsymbol{\mu}_{k|k-1}, \mathbf{P}_{k|k-1}), \qquad (3.6)
\end{aligned}
$$

where the use of $\bullet$ in $\mathcal{N}(\boldsymbol{x}_{k-1}; \bullet, \bullet)$ is to emphasize that the expectation and covariance is of no interest, as $\int \mathcal{N}(\boldsymbol{x}_{k-1}; \bullet, \bullet)\,\mathrm{d}\boldsymbol{x}_{k-1} = 1$, and

$$
\boldsymbol{\mu}_{k|k-1} = \mathbf{F}_{k-1}\boldsymbol{\mu}_{k-1|k-1}, \qquad (3.7)
$$

$$
\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^{\mathsf{T}} + \mathbf{Q}_{k-1}. \qquad (3.8)
$$

Notice that (3.6), more intuitively than direct use of the product identity, can also be derived from the fact that $\boldsymbol{x}_k$ in (3.1) must be Gaussian under the assumptions in Chapter 3.1. This results in just having to find the expectation and covariance of $\boldsymbol{x}_k$ to express its PDF, as the Gaussian distribution is fully parameterized by its expectation and covariance. This gives the expectation

$$
\begin{aligned}
\mathrm{E}[\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}] &= \mathrm{E}[\mathbf{F}_{k-1}\boldsymbol{x}_{k-1} + \boldsymbol{v}_{k-1}|\boldsymbol{z}_{1:k-1}] \\
&= \mathbf{F}_{k-1}\mathrm{E}[\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1}] + \mathrm{E}[\boldsymbol{v}_{k-1}|\boldsymbol{z}_{1:k-1}] \\
&= \mathbf{F}_{k-1}\boldsymbol{\mu}_{k-1|k-1},
\end{aligned}
$$

and covariance

$$
\begin{aligned}
\mathrm{Cov}[\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}] &= \mathrm{Cov}[\mathbf{F}_{k-1}\boldsymbol{x}_{k-1} + \boldsymbol{v}_{k-1}|\boldsymbol{z}_{1:k-1}] \\
&= \mathbf{F}_{k-1}\mathrm{Cov}[\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1}]\mathbf{F}_{k-1}^{\mathsf{T}} + \mathrm{Cov}[\boldsymbol{v}_{k-1}|\boldsymbol{z}_{1:k-1}] \\
&= \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^{\mathsf{T}} + \mathbf{Q}_{k-1},
\end{aligned}
$$

arriving at the same parameters as in (3.6).

3.3.3 The innovation step

As an intermediate step between the prediction and update, the *innovation step* handles the arrival of new measurements z_k. This is done by first calculating the predicted measurement, λ_k, as

$$\begin{aligned}\lambda_k &= \mathrm{E}\left[z_k | z_{1:k-1}\right] \\ &= \mathrm{E}\left[\mathbf{H}_k \boldsymbol{x}_k + \boldsymbol{w}_k | z_{1:k-1}\right] \\ &= \mathbf{H}_k \mathrm{E}\left[\boldsymbol{x}_k | z_{1:k-1}\right] + \mathrm{E}\left[\boldsymbol{w}_k | z_{1:k-1}\right] \\ &= \mathbf{H}_k \mu_{k|k-1}, \end{aligned} \tag{3.9}$$

which is consistent with (2.24a). This is then used to calculate the *innovation* $\boldsymbol{\nu}$, defined as the difference between predicted measurement and true measurement, by

$$\boldsymbol{\nu}_k = z_k - \lambda_k. \tag{3.10}$$

In addition, the *innovation covariance* is calculated as

$$\begin{aligned}\mathbf{S}_k &= \mathrm{Cov}\left[\boldsymbol{\nu}_k | z_{1:k-1}\right] \\ &= \mathrm{Cov}\left[z_k - \lambda_k | z_{1:k-1}\right] \\ &= \mathrm{Cov}\left[z_k | z_{1:k-1}\right] \\ &= \mathrm{Cov}\left[\mathbf{H}_k \boldsymbol{x}_k + \boldsymbol{w}_k | z_{1:k-1}\right] \\ &= \mathbf{H}_k \mathrm{Cov}\left[\boldsymbol{x}_k | z_{1:k-1}\right] \mathbf{H}_k^\top + \mathrm{Cov}\left[\boldsymbol{w}_k | z_{1:k-1}\right] \\ &= \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\top + \mathbf{R}_k. \end{aligned} \tag{3.11}$$

which is consistent with (2.24b).

3.3.4 The update step

Finally, the update step of the Bayes filter gives the desired posterior, given by (2.12). Under the assumptions of the state-space model in Chapter 3.2, the involved PDFs are the measurement model (3.4), the prior (3.6) and

$$p(z_k | z_{1:k-1}) = \mathcal{N}(z_k; \lambda_k, \mathbf{S}_k), \tag{3.12}$$

where λ_k and $\mathbf{S}_k$ are given by (3.9) and (3.11), respectively.

Using the product identity on the posterior then finally gives

$$
\begin{aligned}
p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}) &= \frac{\mathcal{N}(\boldsymbol{z}_k;\mathbf{H}_k\boldsymbol{x}_k,\mathbf{R}_k)\mathcal{N}(\boldsymbol{x}_k;\boldsymbol{\mu}_{k|k-1},\mathbf{P}_{k|k-1})}{\mathcal{N}(\boldsymbol{z}_k;\boldsymbol{\lambda}_k,\mathbf{S}_k)} \\
&= \frac{\mathcal{N}(\boldsymbol{z}_k;\boldsymbol{\lambda}_k,\mathbf{S}_k)\mathcal{N}(\boldsymbol{x}_k;\boldsymbol{\mu}_{k|k},\mathbf{P}_{k|k})}{\mathcal{N}(\boldsymbol{z}_k;\boldsymbol{\lambda}_k,\mathbf{S}_k)} \\
&= \mathcal{N}(\boldsymbol{x}_k;\boldsymbol{\mu}_{k|k},\mathbf{P}_{k|k}),
\end{aligned} \tag{3.13}
$$

where

$$
\boldsymbol{\mu}_{k|k} = \boldsymbol{\mu}_{k|k-1} + \mathbf{W}_k\boldsymbol{\nu}_k, \tag{3.14a}
$$

$$
\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{W}_k\mathbf{H}_k)\,\mathbf{P}_{k|k-1}\,(\mathbf{I} - \mathbf{W}_k\mathbf{H}_k)^\mathsf{T} + \mathbf{W}_k\mathbf{R}_k\mathbf{W}_k^\mathsf{T}, \tag{3.14b}
$$

with $\mathbf{W}_k$ called the *Kalman gain*, defined as

$$
\mathbf{W}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^\mathsf{T}\mathbf{S}_k^{-1}. \tag{3.15}
$$

The algorithm is summarized in Algorithm 1.

Algorithm 1 The Kalman filter algorithm

The prediction step

$\boldsymbol{\mu}_{k|k-1} \leftarrow \mathbf{F}_{k-1}\boldsymbol{\mu}_{k-1|k-1}$

$\mathbf{P}_{k|k-1} \leftarrow \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^\mathsf{T} + \mathbf{Q}_{k-1}$

The innovation step

$\boldsymbol{\lambda}_k \leftarrow \mathbf{H}_k\boldsymbol{\mu}_{k|k-1}$

$\boldsymbol{\nu}_k \leftarrow \boldsymbol{z}_k - \boldsymbol{\lambda}_k$

$\mathbf{S}_k \leftarrow \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^\mathsf{T} + \mathbf{R}_k$

The update step

$\mathbf{W}_k \leftarrow \mathbf{P}_{k|k-1}\mathbf{H}_k^\mathsf{T}\mathbf{S}_k^{-1}$

$\boldsymbol{\mu}_{k|k} \leftarrow \boldsymbol{\mu}_{k|k-1} + \mathbf{W}_k\boldsymbol{\nu}_k$

$\mathbf{P}_{k|k} \leftarrow (\mathbf{I} - \mathbf{W}_k\mathbf{H}_k)\,\mathbf{P}_{k|k-1}\,(\mathbf{I} - \mathbf{W}_k\mathbf{H}_k)^\mathsf{T} + \mathbf{W}_k\mathbf{R}_k\mathbf{W}_k^\mathsf{T}$

3.4 Properties of the Kalman filter

In the previous chapters, the Kalman filter was introduced and derived in a Bayesian setting. What has not yet been properly discussed are the remarkable

properties that the filter possess under the assumption that both the process and measurement model are both linear with additive, Gaussian white noise. By inserting (3.7) and (3.10) into (3.14a), the state estimate of time step k can be distilled into

$$\boldsymbol{\mu}_{k|k} = \mathbf{F}_{k-1}\boldsymbol{\mu}_{k-1|k-1} + \mathbf{W}_k(\boldsymbol{z}_k - \boldsymbol{\lambda}_k), \tag{3.16}$$

which is the general equation for what is called a *linear state observer*. Here, $\mathbf{W}_k$ can in principle be any arbitrary *gain matrix*, and determines how fast the state estimate converges to the actual state [15]. Although (3.16) usually appears when estimating the state of a deterministic system, it is just as applicable for stochastic systems such as (3.1) and (3.3). It can then be shown that, under the assumptions of Chapter 3.1, the Kalman gain given in (3.15) is in fact the optimal gain in the sense of yielding the state estimate with the lowest variance [13].

Additionally, considering (3.14a) and (3.14b) were derived from (2.20) when deriving the product identity in Chapter 2.3.2, the KF estimator is an MMSE estimator, as $\boldsymbol{\mu}_{k|k} = \mathrm{E}[\boldsymbol{x}_k|\boldsymbol{z}_k]$ exactly in the linear, Gaussian case. As a consequence of this, the state estimate is also *unbiased*. Lastly, the KF state estimate maximizes the posterior, as

$$\begin{aligned}
\underset{\boldsymbol{x}_k}{\operatorname{argmax}}\ p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}) &= \underset{\boldsymbol{x}_k}{\operatorname{argmax}}\ \mathcal{N}(\boldsymbol{x}_k;\boldsymbol{\mu}_{k|k},\mathbf{P}_{k|k}) \\
&= \underset{\boldsymbol{x}_k}{\operatorname{argmax}}\ \frac{1}{(2\pi)^{n/2}\left|\mathbf{P}_{k|k}\right|^{1/2}}\exp\left(-\frac{1}{2}q(\boldsymbol{x}_k)\right) \\
&= \underset{\boldsymbol{x}_k}{\operatorname{argmax}}\ \exp\left(-\frac{1}{2}q(\boldsymbol{x}_k)\right) \\
&= \underset{\boldsymbol{x}_k}{\operatorname{argmax}}\ -\frac{1}{2}q(\boldsymbol{x}_k) \\
&= \underset{\boldsymbol{x}_k}{\operatorname{argmin}}\ q(\boldsymbol{x}_k) \\
&= \underset{\boldsymbol{x}_k}{\operatorname{argmin}}\ \left(\boldsymbol{x}_k - \boldsymbol{\mu}_{k|k}\right)^{\mathsf{T}}\mathbf{P}_{k|k}^{-1}\left(\boldsymbol{x}_k - \boldsymbol{\mu}_{k|k}\right) \\
&= \boldsymbol{\mu}_{k|k},
\end{aligned}$$

where the last equality holds as $\mathbf{P}_{k|k}$ is symmetric and thus positive definite, meaning that $\boldsymbol{\mu}_{k|k}$ is the unique global minimum of $q(\boldsymbol{x}_k)$. This shows that the KF estimator is indeed a MAP estimator as well.

4 | The extended Kalman filter

Having introduced the linear Kalman filter in Chapter 3, this chapter intends to first briefly discuss the concept of a nonlinear system together with some methods for solving the estimation problem for such systems. Then, the natural extension of the original linear Kalman filter, the extended Kalman filter, is introduced, together with its inner workings.

4.1 Nonlinear filtering

A substantial weakness of the original, linear formulation of the Kalman filter is the simple fact that few realistic, dynamic systems exhibit such behaviour, but instead have some degree of nonlinearity. Thus, a linearization procedure must be carried out for the process and measurement models to be fit into the Kalman filter framework described in Chapter 3.3. This solution is known as the *extended Kalman filter* (EKF), and is today considered the standard estimation algorithm for navigation systems [16]. The EKF works by linearizing the process and measurement model so the Gaussian assumption can be maintained, although only as an estimated PDF. For many applications, this alone has a satisfying performance.

The EKF is far from the only way of handling nonlinear systems. An exhaustive list over all possible ways is impossible to summarize in any sense of the word, and so a hand-selected few will get an honorable mention. A popular recursive algorithm is the *particle filter*, which estimates the PDF of $\boldsymbol{x}_k$ in a nonparameteric form by *sampling* different possible trajectories of $\boldsymbol{x}_{1:k}$, called *particles*. How the different particles are distributed in the state space can then be interpreted as an approximation of the true PDF of $\boldsymbol{x}_k$. This methodology of state estimation creates the foundation of a whole family of estimation methods called *Sequential Monte Carlo methods* and is a field of its own. A more thorough treatment of

the particle filter in particular can be found in [8]. Another related filter to EKF, called the *unscented Kalman filter* (UKF), approaches the linearization scheme by instead of linearizing the nonlinear model at the expectation and propagating the state estimate and its covariance from that point, a deterministically chosen set of points, called *sigma points*, about the expectation is propagated through the nonlinear model. The transformed points are then used to linearly estimate the expectation and covariance of the new PDF from these transformed points, weighted with predefined weights. More information on UKF can be found in [16]. Lastly, one may discard the stochastic terms of the system model altogether and instead resort to a *nonlinear observer*, where the dynamics of the state estimate $\boldsymbol{\mu}$ follow a model similar to the actual system, but augmented by a feedback injection term that makes the estimate $\boldsymbol{\mu}$ converge to the true state $\boldsymbol{x}$. Equation (3.16) is an example in the linear case. Further discussion on the topic can be found in [17].

4.2 Linearization in EKF

The linearization procedure in the EKF demands some work to derive, and the following chapter will show the required calculations in order to arrive at the desired results.

4.2.1 The process model

The nonlinear process model in the EKF is of the form

$$\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}) + \boldsymbol{v}_{k-1}, \tag{4.1}$$

where $\boldsymbol{v}_{k-1}$ is assumed to have the same properties as in (3.1), that is, Gaussian white noise and $\boldsymbol{f}(\boldsymbol{x}_{k-1})$ is some general, nonlinear vector function. Deriving the process model $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ is trivial as, conditioned on $\boldsymbol{x}_{k-1}$, $\boldsymbol{f}(\boldsymbol{x}_{k-1})$ reduces to a simple constant added to a Gaussian random vector, resulting in

$$p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}) = \mathcal{N}(\boldsymbol{x}_k; \boldsymbol{f}(\boldsymbol{x}_{k-1}), \mathbf{Q}_{k-1}). \tag{4.2}$$

With the process model from (4.2), one can calculate the prior of time step k. Initially, this follows the same procedure as in Chapter 3.3.2 by using a prediction, as

$$
\begin{aligned}
p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}) &= \int p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})p(\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1})\,\mathrm{d}\boldsymbol{x}_{k-1} \\
&= \int \mathcal{N}(\boldsymbol{x}_k; f(\boldsymbol{x}_{k-1}), \mathbf{Q}_{k-1})\mathcal{N}(\boldsymbol{x}_{k-1}; \boldsymbol{\mu}_{k-1}, \mathbf{P}_{k-1})\,\mathrm{d}\boldsymbol{x}_{k-1}. \qquad (4.3)
\end{aligned}
$$

This, however, is where the nonlinear model becomes problematic. The expectation of $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ in the linear case is simply $\mathbf{F}_{k-1}\boldsymbol{x}_{k-1}$, and thus the product identity can be applied directly. This does not hold in the nonlinear case, and as such, the EKF makes a linear approximation by a first-order Taylor approximation of $f(\boldsymbol{x}_{k-1})$ about the expectation at time step $k-1$ by

$$
f(\boldsymbol{x}_{k-1}) \approx f(\boldsymbol{\mu}_{k-1|k-1}) + \mathbf{F}_{k-1}\left(\boldsymbol{x}_{k-1} - \boldsymbol{\mu}_{k-1|k-1}\right), \qquad (4.4)
$$

where

$$
\mathbf{F}_{k-1} \equiv \left.\frac{\partial f}{\partial \boldsymbol{x}_{k-1}}\right|_{\boldsymbol{\mu}_{k-1|k-1}} \qquad (4.5)
$$

is the *Jacobian* of f at $\boldsymbol{x}_{k-1} = \boldsymbol{\mu}_{k-1|k-1}$. Do not be confused by the repeated use of $\mathbf{F}_{k-1}$, as it is also the Jacobian of the linear model in (3.1), since

$$
\frac{\partial}{\partial \boldsymbol{x}_{k-1}}\left(\mathbf{F}_{k-1}\boldsymbol{x}_{k-1}\right) = \mathbf{F}_{k-1}.
$$

Thus, (4.5) is just the general definition of $\mathbf{F}_{k-1}$.

To continue, $\mathcal{N}\left(\boldsymbol{x}_k; f(\boldsymbol{\mu}_{k-1|k-1}) + \mathbf{F}_{k-1}\left(\boldsymbol{x}_{k-1} - \boldsymbol{\mu}_{k-1|k-1}\right), \mathbf{Q}_{k-1}\right)$ can be rewritten by examining its quadratic form. Writing the quadratic form as $\boldsymbol{y}^\mathsf{T}\mathbf{Q}_{k-1}^{-1}\boldsymbol{y}$, where

$$
\boldsymbol{y} \equiv \boldsymbol{x}_k - \left[f(\boldsymbol{\mu}_{k-1|k-1}) + \mathbf{F}_{k-1}\left(\boldsymbol{x}_{k-1} - \boldsymbol{\mu}_{k-1|k-1}\right)\right],
$$

the quadratic form can be manipulated by rewriting $\boldsymbol{y}$ as

$$
\begin{aligned}
\boldsymbol{y} &= \underbrace{\boldsymbol{x}_k}_{\text{variable}} - \underbrace{\left[f(\boldsymbol{\mu}_{k-1|k-1}) + \mathbf{F}_{k-1}\left(\boldsymbol{x}_{k-1} - \boldsymbol{\mu}_{k-1|k-1}\right)\right]}_{\text{expectation}} \\
&= \boldsymbol{x}_k - f(\boldsymbol{\mu}_{k-1|k-1}) - \mathbf{F}_{k-1}\boldsymbol{x}_{k-1} + \mathbf{F}_{k-1}\boldsymbol{\mu}_{k-1|k-1} \\
&= \underbrace{\left[\boldsymbol{x}_k - f(\boldsymbol{\mu}_{k-1|k-1}) + \mathbf{F}_{k-1}\boldsymbol{\mu}_{k-1|k-1}\right]}_{\text{variable}} - \underbrace{\mathbf{F}_{k-1}\boldsymbol{x}_{k-1}}_{\text{expectation}}
\end{aligned}
$$

so

$$\begin{aligned}&\mathcal{N}\left(x_k; f(\mu_{k-1|k-1}) + \mathbf{F}_{k-1}\left(x_{k-1} - \mu_{k-1|k-1}\right), \mathbf{Q}_{k-1}\right)\\ =\ &\mathcal{N}\left(x_k - f(\mu_{k-1|k-1}) + \mathbf{F}_{k-1}\mu_{k-1|k-1}; \mathbf{F}_{k-1}x_{k-1}, \mathbf{Q}_{k-1}\right).\end{aligned}$$

The product identity can now be applied to (4.3). Simplifying the notation by denoting $s(x_k) \equiv x_k - f(\mu_{k-1|k-1}) + \mathbf{F}_{k-1}\mu_{k-1|k-1}$, the product identity yields

$$\begin{aligned}&\int \mathcal{N}(s(x_k); \mathbf{F}_{k-1}x_{k-1}, \mathbf{Q}_{k-1})\,\mathcal{N}(x_{k-1}; \mu_{k-1|k-1}, \mathbf{P}_{k-1|k-1})\,\mathrm{d}x_{k-1}\\ =\ &\int \mathcal{N}\left(s(x_k); \mathbf{F}_{k-1}\mu_{k-1|k-1}, \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^{\mathsf{T}} + \mathbf{Q}_{k-1}\right)\mathcal{N}(x_{k-1}; \bullet, \bullet)\,\mathrm{d}x_{k-1}\\ =\ &\mathcal{N}\left(s(x_k); \mathbf{F}_{k-1}\mu_{k-1|k-1}, \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^{\mathsf{T}} + \mathbf{Q}_{k-1}\right)\int \mathcal{N}(x_{k-1}; \bullet, \bullet)\,\mathrm{d}x_{k-1}\\ =\ &\mathcal{N}\left(s(x_k); \mu_{k|k-1}, \mathbf{P}_{k|k-1}\right)\end{aligned} \tag{4.6}$$

where $\bullet$ again is used for uninteresting parameters and

$$\mu_{k|k-1} = \mathbf{F}_{k-1}\mu_{k-1|k-1}, \tag{4.7a}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^{\mathsf{T}} + \mathbf{Q}_{k-1} \tag{4.7b}$$

is used in the last equality. The final step to the derivation of the prior is to again manipulate the quadratic form of (4.6). Using the notation $y^{\mathsf{T}}\mathbf{P}_{k|k-1}^{-1}y$, y can be simplified as

$$\begin{aligned}y &= \underbrace{s(x_k)}_{\text{variable}} - \underbrace{\mathbf{F}_{k-1}\mu_{k-1|k-1}}_{\text{expectation}}\\ &= \underbrace{x_k - f(\mu_{k-1|k-1}) + \mathbf{F}_{k-1}\mu_{k-1|k-1}}_{\text{variable}} - \underbrace{\mathbf{F}_{k-1}\mu_{k-1|k-1}}_{\text{expectation}}\\ &= \underbrace{x_k}_{\text{variable}} - \underbrace{f(\mu_{k-1|k-1})}_{\text{expectation}},\end{aligned}$$

showing that the prior can be approximated as

$$p(x_k|z_{1:k-1}) = \mathcal{N}(x_k; f(\mu_{k-1|k-1}), \mathbf{P}_{k|k-1}). \tag{4.8}$$

A simpler and perhaps more intuitive way of arriving at (4.8) is by starting at (4.2), but derive the expectation and covariance of the prior directly from (4.2) by conditioning on $z_{1:k-1}$ instead of x_{k-1}. By linearizing (4.2) with (4.4), it can be approximated as

$$x_k = f(\mu_{k-1|k-1}) + \mathbf{F}_{k-1}\left(x_{k-1} - \mu_{k-1|k-1}\right) + v_{k-1}. \tag{4.9}$$

By taking the expectation of (4.9) conditioned on $z_{1:k-1}$, one gets

$$\begin{aligned}
\mathrm{E}[x_k|z_{1:k-1}] &= \mathrm{E}\left[f(\mu_{k-1|k-1}) + \mathbf{F}_{k-1}\left(x_{k-1} - \mu_{k-1|k-1}\right) + v_{k-1}\middle|z_{1:k-1}\right] \\
&= f(\mu_{k-1|k-1}) + \mathbf{F}_{k-1}\left(\mathrm{E}[x_{k-1}|z_{1:k-1}] - \mu_{k-1|k-1}\right) \\
&= f(\mu_{k-1|k-1}),
\end{aligned} \tag{4.10}$$

where $\mathbf{F}_{k-1}\left(\mathrm{E}[x_{k-1}|z_{1:k-1}] - \mu_{k-1|k-1}\right) = 0$, since $\mathrm{E}[x_{k-1}|z_{1:k-1}] = \mu_{k-1|k-1}$ and $\mathrm{E}[v_{k-1}] = \mathbf{0}$ as $v_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1})$.

Similarly, the covariance is found to be

$$\begin{aligned}
\mathrm{Cov}[x_k|z_{1:k-1}] &= \mathrm{Cov}\left[f(\mu_{k-1|k-1}) + \mathbf{F}_{k-1}\left(x_{k-1} - \mu_{k-1|k-1}\right) + v_{k-1}\middle|z_{1:k-1}\right] \\
&= \mathbf{F}_{k-1}\mathrm{Cov}[x_{k-1}|z_{1:k-1}]\mathbf{F}_{k-1}^{\mathsf{T}} + \mathbf{Q}_{k-1} \\
&= \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^{\mathsf{T}} + \mathbf{Q}_{k-1} \\
&= \mathbf{P}_{k|k-1}.
\end{aligned} \tag{4.11}$$

Under the inductive assumption that the posterior x_{k-1} is Gaussian, the linearized x_k in (4.9) must also be Gaussian. Thus, (4.10) and (4.11) can be directly inserted into a Gaussian distribution, which results in the same PDF as (4.8).

4.2.2 The measurement model

The nonlinear measurement model used in the EKF is

$$z_k = h(x_k) + w_k, \tag{4.12}$$

where w_k is assumed to have the same properties as in (3.3), that is, Gaussian white noise and $h(x_k)$ is, just like $f(x_{k-1})$ in (4.1), some general, nonlinear vector function.

Deriving the linearized likelihood for the EKF follows the same outline as the

derivation given in Chapter 4.2.1, and so the following derivation will be brief. Linearizing $\boldsymbol{h}(\boldsymbol{x}_k)$ about the prediction $\boldsymbol{\mu}_{k|k-1} = \boldsymbol{f}(\boldsymbol{\mu}_{k-1|k-1})$ gives the approximation

$$\boldsymbol{h}(\boldsymbol{x}_k) \approx \boldsymbol{h}(\boldsymbol{\mu}_{k|k-1}) + \mathbf{H}_k \left(\boldsymbol{x}_k - \boldsymbol{\mu}_{k|k-1} \right), \tag{4.13}$$

where

$$\mathbf{H}_k \equiv \left. \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}_k} \right|_{\boldsymbol{\mu}_{k|k-1}} \tag{4.14}$$

is the Jacobian of $\boldsymbol{h}$ at $\boldsymbol{x}_k = \boldsymbol{\mu}_{k|k-1}$. Again, do not be confused by the repeated use of $\mathbf{H}_k$, as it is also the Jacobian of the linear model in (3.3), since

$$\frac{\partial}{\partial \boldsymbol{x}_k} (\mathbf{H}_k \boldsymbol{x}_k) = \mathbf{H}_k.$$

Thus, (4.14) is just the general definition of $\mathbf{H}_k$, similar to how (4.5) is the general definition for $\mathbf{F}_{k-1}$.

The likelihood function has a similar structure to (4.2) as it is given by

$$p(\boldsymbol{z}_k|\boldsymbol{x}_k) = \mathcal{N}(\boldsymbol{z}_k; \boldsymbol{h}(\boldsymbol{x}_k), \mathbf{R}_k). \tag{4.15}$$

From this, the posterior can be approximated as

$$p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}) = \mathcal{N}\left(\boldsymbol{x}_k; \boldsymbol{\mu}_k, \mathbf{P}_k\right), \tag{4.16}$$

with $\boldsymbol{\mu}_k$ and $\mathbf{P}_k$ given by the familiar KF equations in (3.14), where $\mathbf{H}_k$ now is given by (4.14).

4.3 The EKF algorithm

With the prior and posterior approximations used in the EKF derived as (4.8) and (4.16), respectively, the algorithm can be summarized. As it follows the same steps overall as in Chapter 3.3, only the involved formulas are repeated, and is given in Algorithm 2.

Algorithm 2 The extended Kalman filter algorithm

The prediction step

1: $\boldsymbol{\mu}_{k|k-1} \leftarrow f(\boldsymbol{\mu}_{k-1|k-1})$
2: $\mathbf{F}_{k-1} \leftarrow \left.\frac{\partial f}{\partial \boldsymbol{x}_{k-1}}\right|_{\boldsymbol{\mu}_{k-1|k-1}}$
3: $\mathbf{P}_{k|k-1} \leftarrow \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1}^{\mathsf{T}} + \mathbf{Q}_{k-1}$

The innovation step

4: $\boldsymbol{\lambda}_{k|k-1} \leftarrow h(\boldsymbol{\mu}_{k|k-1})$
5: $\boldsymbol{\nu}_k \leftarrow \boldsymbol{z}_k - \boldsymbol{\lambda}_{k|k-1}$
6: $\mathbf{H}_k \leftarrow \left.\frac{\partial h}{\partial \boldsymbol{x}_k}\right|_{\boldsymbol{\mu}_{k|k-1}}$
7: $\mathbf{S}_k \leftarrow \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathsf{T}} + \mathbf{R}_k$

The update step

8: $\mathbf{W}_k \leftarrow \mathbf{P}_{k|k-1}\mathbf{H}_k^{\mathsf{T}}\mathbf{S}_k^{-1}$
9: $\boldsymbol{\mu}_{k|k} \leftarrow \boldsymbol{\mu}_{k|k-1} + \mathbf{W}_k\boldsymbol{\nu}_k$
10: $\mathbf{P}_{k|k} \leftarrow (\mathbf{I} - \mathbf{W}_k\mathbf{H}_k)\,\mathbf{P}_{k|k-1}\,(\mathbf{I} - \mathbf{W}_k\mathbf{H}_k)^{\mathsf{T}} + \mathbf{W}_k\mathbf{R}_k\mathbf{W}_k^{\mathsf{T}}$

5 | The error-state Kalman filter

Chapter 4 introduced local linearization of a nonlinear model as a way to fit the nonlinear models into the linear framework of the KF. The accuracy of this approximation, however, is highly reliant on how well the nonlinearities of the models can be captured by linearization. Some systems may exhibit strong nonlinearities where a local linearization in the fashion of the EKF will not be a sufficiently accurate approximation, which then is a cause for possible divergence. *The error-state Kalman filter* (ESKF) that will be introduced in this chapter approaches the linearization scheme slightly differently than the EKF, which improves this linearization approximation. Though the ESKF has its main motivation within the application of inertial navigation systems, it will here be presented in a more general form. The following chapter draws inspiration from [18] and [2], though they are limited to the context of inertial navigation. Other names used in the literature are the *multiplicative extended Kalman filter* (MEKF) and the *indirect Kalman filter* (IKF).

The following chapter is structured as follows. First, some motivation is built for exactly why the ESKF is used over the EKF, where comparisons are used to make the transition more clear. Then, the general error process and measurement models are derived. Lastly, the steps of the ESKF algorithm are explained.

5.1 Introduction and motivation

Before motivating the use of the ESKF, one should have a general idea of what it is, and so the following will compare it with the EKF. For any state estimation with Bayesian filtering, such as the KF, the goal is to arrive at the *true state*, $\boldsymbol{x}$. As the underlying model is a HMM, the true state is unknown and must be estimated by $\hat{\boldsymbol{x}}$, giving some *nominal, or believed, state*. Since the estimate after

all is an *estimate*, there will be an error between the true state and nominal state, *the error state* $\delta \boldsymbol{x}$. In the EKF, the relation between these three states are assumed to simply be

$$\boldsymbol{x} = \hat{\boldsymbol{x}} + \delta \boldsymbol{x}. \tag{5.1}$$

Despite terminology as true, nominal and error state being common in ESKF literature, but not in EKF literature, these very states do also appear in the EKF, but under different names. Recall the prediction and update equations of the EKF, where $\boldsymbol{x}$ is used instead of μ to distinguish between stochastic states and numeric estimates,

$$\begin{aligned} \boldsymbol{x}_{k|k-1} &= \boldsymbol{f}(\boldsymbol{x}_{k-1}) + \boldsymbol{v}_{k-1}, \\ \boldsymbol{x}_k &= \boldsymbol{x}_{k|k-1} + \mathbf{W}_k \boldsymbol{\nu}_k \end{aligned}$$

restated here for comparison with the same equations, only with $\boldsymbol{x}$, $\hat{\boldsymbol{x}}$ and $\delta \boldsymbol{x}$

$$\begin{aligned} \hat{\boldsymbol{x}}_k &= \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}) \\ \delta \boldsymbol{x}_k &= \mathbf{W}_k \boldsymbol{\nu}_k \\ \boldsymbol{x}_k &= \hat{\boldsymbol{x}}_k + \delta \boldsymbol{x}_k \end{aligned}$$

where the specific details of the actual ESKF algorithm are disregarded. The important take away is that *the ESKF really operates on the same states as the EKF, but where the linearization is at the error state* $\delta \boldsymbol{x}$ *instead of* $\hat{\boldsymbol{x}}$ *to improve linearization accuracy.* See Figure 5.1 for a visualization of how the error state $\delta \boldsymbol{x}$ in general has better linearization properties than the nominal state $\hat{\boldsymbol{x}}$.

5.1.1 Composing the nominal and error state

It is hopefully apparent from Chapter 5.1 that one of the main benefits of using the ESKF over the EKF is the better linearization property. Another huge benefit that will not be elaborated on here is the fact that *the ESKF allows for different state parameterizations in the nominal state* $\hat{\boldsymbol{x}}$ *and the error state* $\delta \boldsymbol{x}$. Under this generalization, (5.1) is rewritten as

$$\boldsymbol{x} = \hat{\boldsymbol{x}} \oplus \delta \boldsymbol{x}, \tag{5.2}$$

where $\oplus$ is the *composition operator* that can be thought of as some function, linear or nonlinear, that combines the nominal and error state into the true state

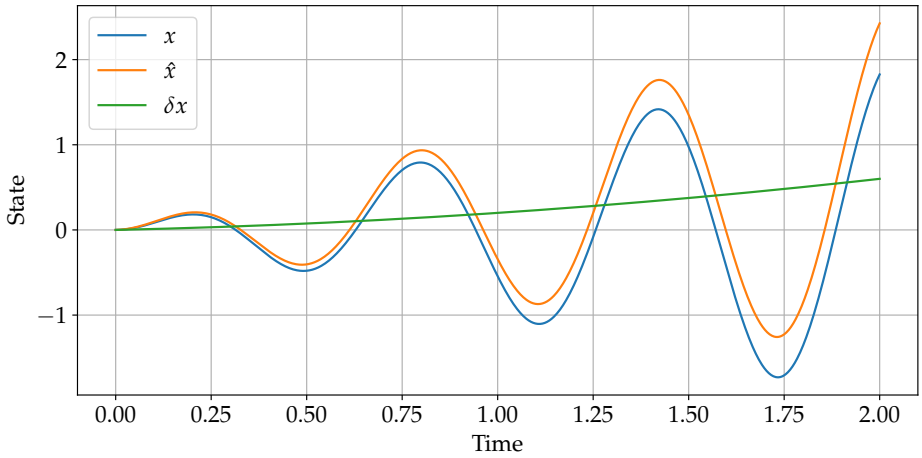


Figure 5.1: *Plot of how the error state might develop compared to the true and nominal state. While EKF linearizes $\hat{\boldsymbol{x}}$, ESKF linearizes $\delta \boldsymbol{x}$, where the linearization is a much better approximation.*

as

$$\boldsymbol{x} = \oplus(\hat{\boldsymbol{x}}, \delta \boldsymbol{x}). \tag{5.3}$$

The notation in (5.2) is used to build intuition of what is really going on and to make it similar to the more familiar composition in (5.1). In the case that the error state is additive on the nominal state, that is, $\oplus = +$, this relationship is just as for the previously discussed KF and EKF. This is in general not the case in the ESKF, and although the previous chapters have assumed an additive relationship on the error, it is here generalized to coincide better with coming chapters.

The main conceptual difference for the ESKF compared to the EKF will be to estimate the error state instead of the true state, while in parallel predicting the nominal state. This indirectly gives an estimate of the true state, which after all is of interest, as the true state is a composition of the nominal and error state, as seen in (5.2). A key point is that once the error state is estimated, it is *injected* into the nominal state to give an estimate of the true state, while subsequently being reset to zero before the next iteration. Thus, the end of each estimation cycle will be

$$\begin{aligned} \hat{\boldsymbol{x}} &\leftarrow \hat{\boldsymbol{x}} \oplus \delta \hat{\boldsymbol{x}}, \\ \delta \hat{\boldsymbol{x}} &\leftarrow \boldsymbol{0}. \end{aligned}$$

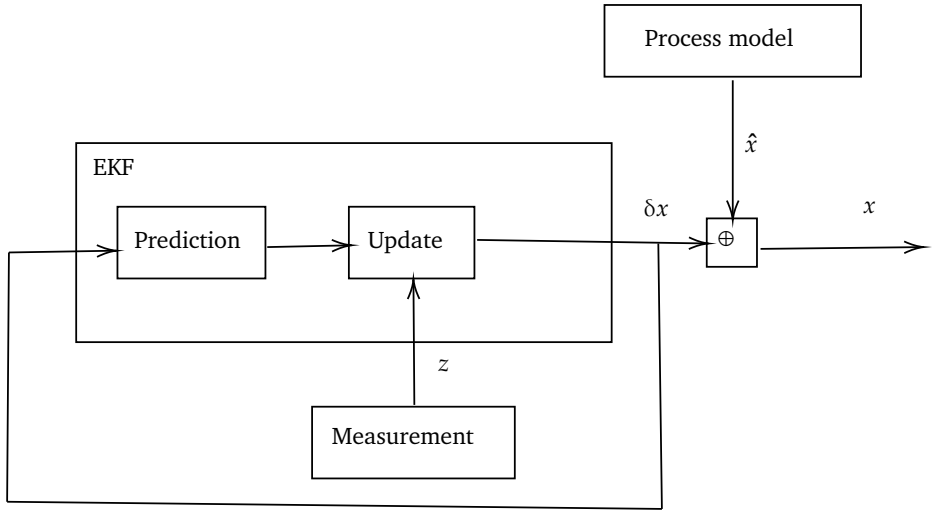


Figure 5.2: *A simplified block diagram showing how ESKF just runs EKF on the error state $\delta \boldsymbol{x}$, with the prediction of the nominal state $\hat{\boldsymbol{x}}$ outside the EKF loop. The two states are composed together after arrival of a new measurement to form the true state x.*

In the next iteration, the error state is again estimated and reset to zero after injection, and the filter continues like this. See Figure 5.2 for an overview over how an ESKF estimation cycle is.

Now, to summarize. The main motivation behind the ESKF is that the error state remains small at all times, due to the reset in each iteration. This has some advantages over the EKF [18], at this point primarily that the linearization remains valid as higher-order products become negligible. At least equally important, and perhaps the main reason for using the ESKF over the EKF, is that *the composition of the nominal and error state does not in general make any assumption of the state-spaces of said states*. This means the nominal and error state may consist of different states, as long as they can be composed to give the true state. This needs further explanation, and will be expanded on in Chapter 8. For now, accept that using different states in the error and nominal state may be beneficial, which will be seen when estimating states that have dynamics that exhibit singularities if operating too far from their origins. As will be seen in Chapter 6, the *orientation* of an object in 3-dimensional space is such a state.

5.2 The error-state system

To estimate the error state in the Kalman filter framework, the process and measurement models for the error state must be provided.

5.2.1 The process model

In order to acquire the process model of the error state, the process model of both the true and nominal state are required. As for the EKF, the process model for the true state takes the nonlinear form of (4.1). The nominal state is then given as the expectation of the true state, where all process noises are removed. The process model for the nominal state is therefore given as simply

$$\hat{\boldsymbol{x}}_k = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}). \tag{5.4}$$

The linearized error process model can then be developed from (4.1), (5.4) and (5.2). Unfortunately, developing a general formula is futile. This follows from the nonlinear relation between the nominal and error state. A naive approach that is something along the lines of Chapter 4.2.1 would be

$$\begin{aligned}
\boldsymbol{x}_k &= \boldsymbol{f}(\boldsymbol{x}_{k-1}) + \boldsymbol{v}_{k-1} \\
&= \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1} \oplus \delta\boldsymbol{x}_{k-1}) + \boldsymbol{v}_{k-1} \\
&\approx \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1} \oplus \boldsymbol{0}) + \left.\frac{\partial \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1} \oplus \delta\boldsymbol{x}_{k-1})}{\partial \delta\boldsymbol{x}_{k-1}}\right|_{\delta\boldsymbol{x}_{k-1}=\boldsymbol{0}} (\delta\boldsymbol{x}_{k-1} - \boldsymbol{0}) + \boldsymbol{v}_{k-1} \\
&= \underbrace{\hat{\boldsymbol{x}}_k}_{\boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}\oplus\boldsymbol{0})} + \underbrace{\tilde{\mathbf{F}}_{k-1}}_{\frac{\partial \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}\oplus\delta\boldsymbol{x}_{k-1})}{\partial\delta\boldsymbol{x}_{k-1}}} \delta\boldsymbol{x}_{k-1} + \boldsymbol{v}_{k-1}.
\end{aligned}$$

However, as $\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k \neq \delta\boldsymbol{x}$ in general, this will not suffice. A bonafide linear process model that is explicit in $\delta\boldsymbol{x}_k$ can therefore not be developed from the same procedure as in Chapter 4.2.1. In other words, *using the Jacobian* $\tilde{\mathbf{F}}_{k-1} = \frac{\partial \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}\oplus\delta\boldsymbol{x}_{k-1})}{\partial\delta\boldsymbol{x}_{k-1}}$ *is in general not correct*. The process model can instead be obtained by first specifying the nonlinear dynamics of the true state, which are in general stochastic. The nominal state dynamics then naturally follow by taking the expectation of the true state dynamics. With these two nonlinear process models specified, the error state dynamics can be constructed by taking a suitable *decomposition* in the form $\delta\boldsymbol{x} = \boldsymbol{x} \ominus \hat{\boldsymbol{x}}$, where $\ominus$ can be considered the inverse function of $\oplus$. The resulting process model can then be linearized. Such an approach will in fact be used

later in Chapter 7 when deriving the kinematics for an INS. Needless to say, the general result of this development is indeed a linear process model on the form

$$\delta \boldsymbol{x}_k = \mathbf{F}_{k-1}\delta \boldsymbol{x}_{k-1} + \boldsymbol{v}_{k-1} \tag{5.5}$$

where $\boldsymbol{v}_{k-1}$ remains Gaussian as $\boldsymbol{v}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1})$. In general, (5.5) is a *linear time-varying (LTV) system* where $\mathbf{F}_{k-1} = \mathbf{F}(\hat{\boldsymbol{x}}_{k-1})$ is a function of $\hat{\boldsymbol{x}}_{k-1}$ and any external input. The fact that $\mathbf{F}_{k-1}$ is a function of $\hat{\boldsymbol{x}}_{k-1}$ should come as no surprise, as the error state dynamics are inherently entangled with the nominal state dynamics. The process model can now be found to be

$$p(\delta \boldsymbol{x}_k | \delta \boldsymbol{x}_{k-1}) = \mathcal{N}(\delta \boldsymbol{x}_k; \mathbf{F}_{k-1}\delta \boldsymbol{x}_{k-1}, \mathbf{Q}_{k-1}). \tag{5.6}$$

5.2.2 The measurement model

Measurements will typically contain information of the true state, and not the error state directly, so the measurement model takes the same nonlinear form as for the EKF in (4.12). Deriving a general expression for the measurement model is, contrary to the process model, possible. Recall that the argumentation at the end of Chapter 5.1.1 for using a nonlinear relation between $\hat{\boldsymbol{x}}_k$ and $\delta \boldsymbol{x}_k$ was because of state dynamics singularities. For the measurement model, however, the underlying continuous-time measurement model contains no differential quantities, and so the measurement vector $\boldsymbol{z}_k$ can safely be constructed such that the innovation $\boldsymbol{\nu}_k$ defined in (3.10), in other words the *measurement error*, is always additive on the nominal measurement $\boldsymbol{h}(\hat{\boldsymbol{x}}_k)$.

The measurement model used in the ESKF can be retrieved as

$$\begin{aligned}
\boldsymbol{z}_k &= \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{w}_k \\
&= \boldsymbol{h}(\hat{\boldsymbol{x}}_k \oplus \delta \boldsymbol{x}_k) + \boldsymbol{w}_k \\
&\approx \boldsymbol{h}(\hat{\boldsymbol{x}}_k \oplus \mathbf{0}) + \left.\frac{\partial \boldsymbol{h}(\hat{\boldsymbol{x}}_k \oplus \delta \boldsymbol{x}_k)}{\partial \delta \boldsymbol{x}_k}\right|_{\delta \boldsymbol{x}_k = \mathbf{0}} (\delta \boldsymbol{x}_k - \mathbf{0}) + \boldsymbol{w}_k \\
&= \boldsymbol{h}(\hat{\boldsymbol{x}}_k) + \mathbf{H}_k \delta \boldsymbol{x}_k + \boldsymbol{w}_k \\
\boldsymbol{z}_k - \boldsymbol{h}(\hat{\boldsymbol{x}}_k) &= \mathbf{H}_k \delta \boldsymbol{x}_k + \boldsymbol{w}_k \\
\boldsymbol{\nu}_k &= \mathbf{H}_k \delta \boldsymbol{x}_k + \boldsymbol{w}_k,
\end{aligned} \tag{5.7}$$

where

$$\mathbf{H}_k = \left.\frac{\partial \boldsymbol{h}(\hat{\boldsymbol{x}}_k \oplus \delta\boldsymbol{x}_k)}{\partial \delta\boldsymbol{x}_k}\right|_{\delta\boldsymbol{x}_k = \mathbf{0}}. \tag{5.8}$$

In practice, since $\boldsymbol{h}(\boldsymbol{x}_k)$ only implicitly is a function of $\delta\boldsymbol{x}_k$, (5.8) is evaluated by use of the chain rule,

$$\mathbf{H}_k = \frac{\partial \boldsymbol{h}(\hat{\boldsymbol{x}}_k \oplus \delta\boldsymbol{x}_k)}{\partial \delta\boldsymbol{x}_k} = \frac{\partial \boldsymbol{h}(\boldsymbol{x}_k)}{\partial \boldsymbol{x}_k} \cdot \frac{\partial \boldsymbol{x}_k}{\partial \delta\boldsymbol{x}_k} \equiv \underbrace{\mathbf{H}_{\boldsymbol{x}}}_{\frac{\partial \boldsymbol{h}(\boldsymbol{x}_k)}{\partial \boldsymbol{x}_k}} \cdot \underbrace{\mathbf{X}_{\delta\boldsymbol{x}}}_{\frac{\partial \boldsymbol{x}_k}{\partial \delta\boldsymbol{x}_k}}. \tag{5.9}$$

The actual contents of $\mathbf{H}_{\boldsymbol{x}}$ and $\mathbf{X}_{\delta\boldsymbol{x}}$ can only be evaluated with a specific measurement model and state space, and is not concerned with here. The measurement model can be expressed in $\boldsymbol{z}_k$ as in the KF and the EKF, but because of the form of (5.7), it is here expressed with $\boldsymbol{\nu}_k$ instead as

$$p(\boldsymbol{\nu}_k | \delta\boldsymbol{x}_k) = \mathcal{N}(\boldsymbol{\nu}_k; \mathbf{H}_k \delta\boldsymbol{x}_k, \mathbf{R}_k). \tag{5.10}$$

Note that this only shifts the expectation and leaves the covariance untouched.

5.3 The ESKF algorithm

The ESKF follows a similar procedure to that of the EKF, but with a prediction of the nominal state in parallel with the estimation of the error state. When the error state is estimated by the prediction and update steps, it is subsequently injected into the nominal state to give an estimate of the true state, which is done in the additional injection step.

5.3.1 Notation

For the KF and the EKF, it was useful to separate the prior estimate $\boldsymbol{\mu}_{k|k-1}$ and posterior estimate $\boldsymbol{\mu}_{k|k}$ by time indexes as these are estimates of the same state $\boldsymbol{x}_k$, but based on different information. For the ESKF, however, this is no longer strictly true, as there are now three states, the true state $\boldsymbol{x}_k$, the nominal state $\hat{\boldsymbol{x}}_k$ and the error state $\delta\boldsymbol{x}_k$, that are estimated *simultaneously*, and so a little bookkeeping is necessary. This is done by denoting the estimate of the true state $\boldsymbol{x}_k$ by $\boldsymbol{\mu}_k$ and the estimate of the nominal state $\hat{\boldsymbol{x}}_k$ by $\hat{\boldsymbol{\mu}}_k$. For the error state $\delta\boldsymbol{x}_k$, the estimate of the predicted error state is denoted by $\delta\boldsymbol{\mu}_k$ and the updated error state by $\delta\hat{\boldsymbol{\mu}}_k$. The covariance matrix of the predicted error state is denoted by $\delta\mathbf{P}_k$

while the covariance matrix of the updated error state is denoted by $\delta\hat{\mathbf{P}}_k$.

The main reason for these notations is to more explicitly separate the time at which things happen while the algorithm is running. Although this will be elaborated on later, it is suffice to say that the prediction step in the ESKF runs much more often than the update step, and so the nominal estimate $\hat{\boldsymbol{\mu}}_k$ and error covariance $\delta\mathbf{P}_k$ will almost always be unsynchronized with the true state estimate $\boldsymbol{\mu}_k$ and updated covariance $\delta\hat{\mathbf{P}}_k$.

5.3.2 Initialization

The filter is initialized with some known state $\hat{\boldsymbol{\mu}}_0$, which acts as the initial nominal state. As the filter operates on the error state, an initial error with covariance must be provided. Being at the initial point of the filter, the expected error should be zero, such that

$$p(\delta\boldsymbol{x}_0) \sim \mathcal{N}(\mathbf{0}, \delta\mathbf{P}_0)$$

with some chosen initial covariance $\delta\mathbf{P}_0$.

5.3.3 The prediction step

The nominal state estimate $\hat{\boldsymbol{\mu}}_k$ is propagated by (5.4) as

$$\hat{\boldsymbol{\mu}}_k = f(\hat{\boldsymbol{\mu}}_{k-1}). \tag{5.11}$$

With the process model of the error state given as (5.6), and the previous posterior assumed the form

$$p(\delta\boldsymbol{x}_{k-1}|\boldsymbol{\nu}_{1:k-1}) = \mathcal{N}(\delta\boldsymbol{x}_{k-1}; \delta\boldsymbol{\mu}_{k-1}, \delta\mathbf{P}_{k-1}),$$

the predicted error state is, as always, given by total probability and the product identity

$$p(\delta\boldsymbol{x}_k|\boldsymbol{\nu}_{1:k-1}) = \mathcal{N}(\delta\boldsymbol{x}_k; \delta\boldsymbol{\mu}_k, \delta\mathbf{P}_k), \tag{5.12}$$

with

$$\delta\boldsymbol{\mu}_k = \mathbf{F}_{k-1}\delta\boldsymbol{\mu}_{k-1}, \tag{5.13}$$

$$\delta\mathbf{P}_k = \mathbf{F}_{k-1}\delta\mathbf{P}_{k-1}\mathbf{F}_{k-1}^{\mathsf{T}} + \mathbf{Q}_{k-1}. \tag{5.14}$$

Note that $\delta\boldsymbol{\mu}_{k-1}$ is the expected error from the previous time step. As the filter is initialized with $\delta\boldsymbol{\mu}_0 = \mathbf{0}$ and the error is reset to zero after each injection, $\delta\boldsymbol{\mu}_{k-1} = \mathbf{0}$ for all k. This means (5.12) can be rewritten as

$$p(\delta\boldsymbol{x}_k|\boldsymbol{\nu}_{1:k-1}) = \mathcal{N}(\delta\boldsymbol{x}_k; \mathbf{0}, \delta\mathbf{P}_k). \tag{5.15}$$

5.3.4 The innovation step

When new measurements arrive, the innovation is in the ESKF given by (5.7). The measurement matrix $\mathbf{H}_k$ is calculated with (5.9). Following the same procedure as for the Kalman filter in Chapter 3, the innovation covariance can be shown to be equal to (3.11) where the suitable matrices are used.

5.3.5 The update step

Updating the estimate by an observation is the only way to observe the error state, and is done as before with Bayes' rule and the product identity on the measurement model in (5.10) and prediction in (5.15), giving

$$p(\delta\boldsymbol{x}_k|\boldsymbol{\nu}_{1:k}) = \mathcal{N}(\delta\boldsymbol{x}_k; \delta\hat{\boldsymbol{\mu}}_k, \delta\hat{\mathbf{P}}_k), \tag{5.16}$$

where

$$\delta\hat{\boldsymbol{\mu}}_k = \mathbf{W}_k\boldsymbol{\nu}_k, \tag{5.17a}$$

$$\delta\hat{\mathbf{P}}_k = (\mathbf{I} - \mathbf{W}_k\mathbf{H}_k)\,\delta\mathbf{P}_k\,(\mathbf{I} - \mathbf{W}_k\mathbf{H}_k)^\mathsf{T} + \mathbf{W}_k\mathbf{R}_k\mathbf{W}_k^\mathsf{T}, \tag{5.17b}$$

with Kalman gain $\mathbf{W}_k$. The measurement matrix $\mathbf{H}_k$ is calculated with (5.9).

5.3.6 The injection step

After the error state has been estimated, the composition in (5.2) may be used to give an estimate of the true state. This final step injects the error into the nominal state, giving the estimate of the true state as

$$\boldsymbol{\mu}_k = \hat{\boldsymbol{\mu}}_k \oplus \delta\hat{\boldsymbol{\mu}}_k, \tag{5.18}$$

such that the new nominal state is the estimate of the true state. As the nominal state has its error corrected for by the injection, the estimated error must be reset before the next iteration, and is simply done by

$$\delta\hat{\boldsymbol{\mu}}_k \leftarrow \mathbf{0}. \tag{5.19}$$

To summarize, the ESKF algorithm is given in Algorithm 3. Notice that the prediction of $\delta\boldsymbol{\mu}_k$ is omitted in the algorithm, as it always takes the value of zero.

Algorithm 3 The error-state Kalman filter algorithm

The prediction step

$\hat{\boldsymbol{\mu}}_k \leftarrow f(\hat{\boldsymbol{\mu}}_{k-1})$
$\mathbf{F}_{k-1} \leftarrow \mathbf{F}(\hat{\boldsymbol{\mu}}_{k-1})$
$\delta\mathbf{P}_k \leftarrow \mathbf{F}_{k-1}\delta\mathbf{P}_{k-1}\mathbf{F}_{k-1}^\top + \mathbf{Q}_{k-1}$

The innovation step

$\boldsymbol{\nu}_k \leftarrow \boldsymbol{z}_k - \boldsymbol{h}(\hat{\boldsymbol{\mu}}_k)$
$\mathbf{H}_{\boldsymbol{x}} \leftarrow \left.\frac{\partial \boldsymbol{h}(\boldsymbol{x}_k)}{\partial \boldsymbol{x}_k}\right|_{\boldsymbol{x}_k=\hat{\boldsymbol{\mu}}_k}$
$\mathbf{X}_{\delta\boldsymbol{x}} \leftarrow \left.\frac{\partial \boldsymbol{x}_k}{\partial \delta\boldsymbol{x}_k}\right|_{\boldsymbol{x}_k=\hat{\boldsymbol{\mu}}_k}$
$\mathbf{H}_k \leftarrow \mathbf{H}_{\boldsymbol{x}}\mathbf{X}_{\delta\boldsymbol{x}}$
$\mathbf{S}_k \leftarrow \mathbf{H}_k\delta\mathbf{P}_k\mathbf{H}_k^\top + \mathbf{R}_k$

The update step

$\mathbf{W}_k \leftarrow \delta\mathbf{P}_k\mathbf{H}_k^\top\mathbf{S}_k^{-1}$
$\delta\hat{\boldsymbol{\mu}}_k \leftarrow \mathbf{W}_k\boldsymbol{\nu}_k$
$\delta\hat{\mathbf{P}}_k \leftarrow (\mathbf{I} - \mathbf{W}_k\mathbf{H}_k)\,\delta\mathbf{P}_k\,(\mathbf{I} - \mathbf{W}_k\mathbf{H}_k)^\top + \mathbf{W}_k\mathbf{R}_k\mathbf{W}_k^\top$

The injection step

$\boldsymbol{\mu}_k \leftarrow \hat{\boldsymbol{\mu}}_k \oplus \delta\hat{\boldsymbol{\mu}}_k$
$\delta\hat{\boldsymbol{\mu}}_k \leftarrow \mathbf{0}$

III

Inertial navigation and kinematics

6 | Introduction to inertial navigation systems

The previous chapters have provided much of the general statistical theory for state estimation, without any specifications of what the states are, or in what setting the estimation methods are used. The rest of the thesis will now turn to an application of state estimation in inertial navigation, specifically for a UAV. An *inertial navigation system* (INS) is a system, usually on some vehicle or robot, that fuses measurements from an *inertial measurement unit* (IMU) with measurements from other sensors in order to estimate its own state for navigation. The following chapter intends to introduce important concepts and mathematical tools for working with coordinate frame transformations in addition to the used sensors.

6.1 Coordinate frames

A key concept for INS design is that of coordinate frames, or coordinate systems. The following chapter will introduce the notation that will be used, in addition to the relevant theory of attitude representation and transformations.

6.1.1 Vector notation and intuition

The following will borrow convention from [19] where $\vec{p}_{ab}$ denotes a *coordinateless* vector. That is, this vector is an entity suspended in space with a length and direction, but where only its length can be quantified. Its direction cannot be quantified until it is *decomposed* into a frame. See Figure 6.1 for visualization. In this case, one can see how both $\boldsymbol{p}_{ab}^{a}$ and $\boldsymbol{p}_{ab}^{b}$ are derived from the same vector

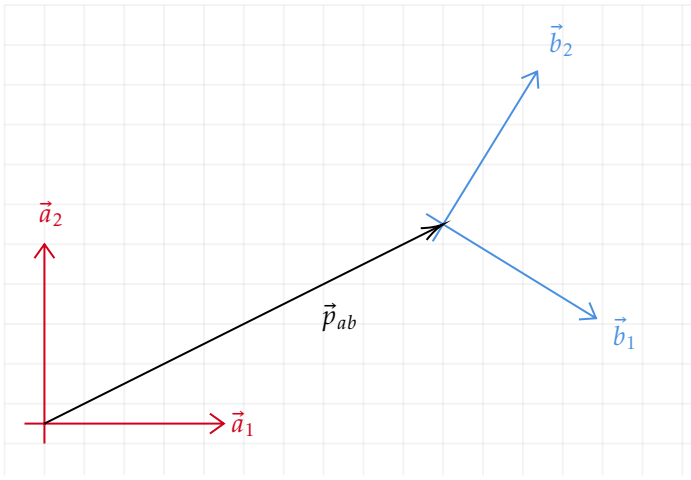
$\vec{p}_{ab}$, but with different coordinates as frame b is tilted relative to frame a. As a consequence, the direction of $\vec{p}_{ab}$ appears more "upwards" in frame b than in frame a because of the different perspectives.

In this example, $\boldsymbol{p}_{ab}^{a}$ has the physical interpretation of being the position of frame b relative frame a. As such, it is useful to define *inertial frame* and *body frame*. The inertial frame is the frame that is defined as stationary and defines a global position for all other bodies living in its world. The body frame is the frame that is attached to some rigid body moving in the world, and thus rotates and translates with this body. In the given example, frame a is the inertial frame and frame b is the body frame. This also applies in general: When some property of frame b is described relative frame a, then frame a is assumed inertial while b can be moving.

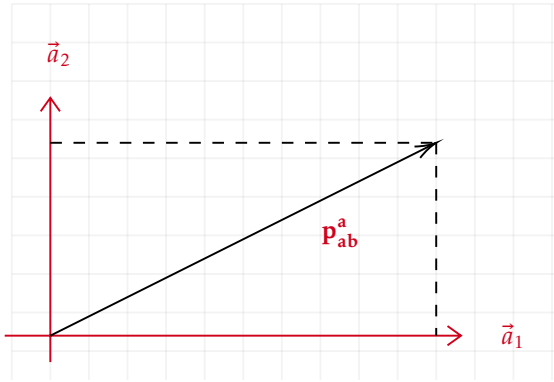
Going back to the example in Figure 6.1, one can see that, following the established notation, $\boldsymbol{p}_{ab}^{b}$ does not have any physical interpretation. The position of frame b relative frame a can only be described in coordinates relative frame a, so decomposing $\vec{p}_{ab}$ in frame b does not give any meaningful information, although it is perfectly doable mathematically. The following example will show that this is usually not the case. Consider a body travelling counterclockwise along the perimeter of the unit circle with constant velocity and inertial frame placed in the center of the circle. See Figure 6.2 for reference. An observer placed in the center of the body frame, with its axes aligned to the body frame, would measure a constant velocity $\boldsymbol{v}_{ab}^{b}$ tangential to the circle perimeter, with zero radial velocity. An observer in the center of the circle, measuring the velocity $\boldsymbol{v}_{ab}^{a}$ of the moving body, however, would measure two sinusoidal components. For an INS, both of these decompositions contain useful information about how the body is moving.

6.2 Attitude representations

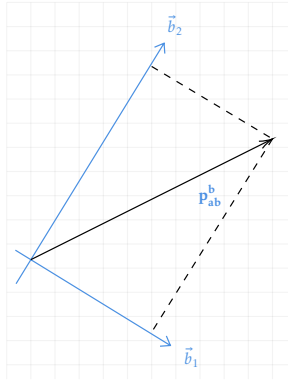
Attitude, or *orientation*, is the property of how one body is oriented relative some reference frame. Many different parameterizations exist for representing attitude [20], and the choice is not arbitrary. As attitude has three degrees of freedom in 3D space, choosing three parameters seems like the obvious choice. A natural choice is then *Tait-Bryan angles*, where the angles represent rotation about the x-, y- and z-axis. These angles are called *roll*, *pitch* and *yaw*, respectively. See Figure 6.3. An alternative representation is *proper Euler angles*, where rotation



(a) *A visualization of how $\vec{p}_{ab}$ is the position of frame b relative frame a.*



(b) *How $\boldsymbol{p}^a_{ab}$ looks like when $\vec{p}_{ab}$ is decomposed in frame a.*

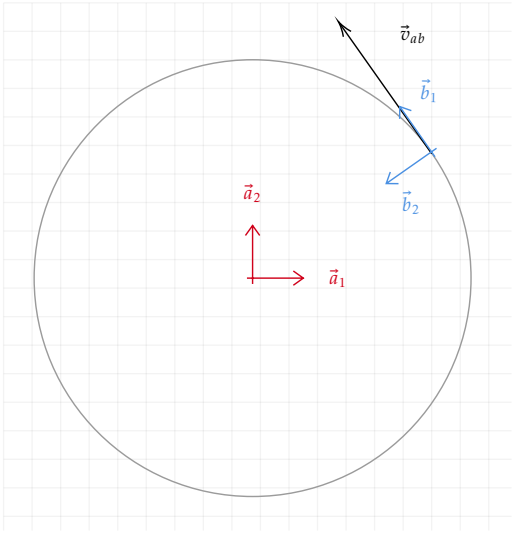


(c) *How $\boldsymbol{p}^b_{ab}$ looks like when $\vec{p}_{ab}$ is decomposed in frame b.*

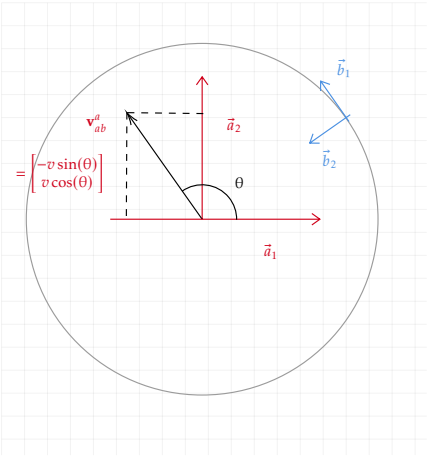
Figure 6.1: *Visualization of how a vector describes properties of one frame relative another, and how it decomposes differently in different frames.*

about an axis is repeated. This is for example by rotating first about the x-axis, then the y-axis, then the x-axis again.

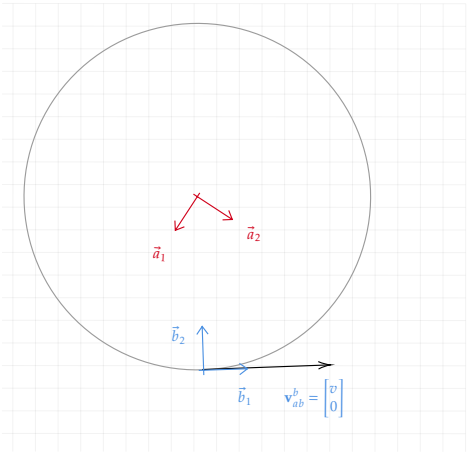
Choosing such a minimal parameterization, that is, where the number of parameters is equal to the degrees of freedom, will however inherently contain singularities. More specifically, an attitude with a pitch of $\pm 90°$ is a transform that is not uniquely defined for a set of Tait-Bryan angles. This phenomena is called *gimbal lock*. Although not shown here, by delving into the mathematics, it can be shown that *it is actually the derivative of the Tait-Bryan angles* at this point that are singular. The system of equations that must be solved becomes under-



(a) *A visualization of how $\vec{v}_{ab}$ is the velocity of frame b relative frame a.*



(b) *How $\boldsymbol{v}_{ab}^{a}$ looks like when $\vec{v}_{ab}$ is decomposed in frame a. Here $\boldsymbol{v}_{ab}^{a} = v \begin{bmatrix} -\sin(\theta) \\ \cos(\theta) \end{bmatrix}$.*



(c) *How $\boldsymbol{v}_{ab}^{b}$ looks like when $\vec{v}_{ab}$ is decomposed in frame b. Note that the entire figure is aligned to frame b. Here $\boldsymbol{v}_{ab}^{b} = v \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.*

Figure 6.2: *Another visualization of how a vector describes properties of one frame relative another, and how it decomposes differently in different frames.*

determined, yielding infinitely many solutions. Thus, even though it is perfectly doable to represent a pitch of $\pm 90°$ for some body, *uniquely determining how it arrived there or exits is not*. See [13], [19] for more examples and explanations. As a consequence, *overparameterization* of the attitude is required, and two such

overparameterizations, the *quaternion* and the *rotation matrix*, will be explored in this chapter.

The following chapter is structured as follows. First, the chapter will define exactly what a *rigid body rotation* really is. Then, the *angle-axis representation* will be introduced for later use and intuition. Lastly, the quaternion and rotation matrix are introduced. Note that much of the related mathematics are found in Appendix B, and are skipped here for brevity.

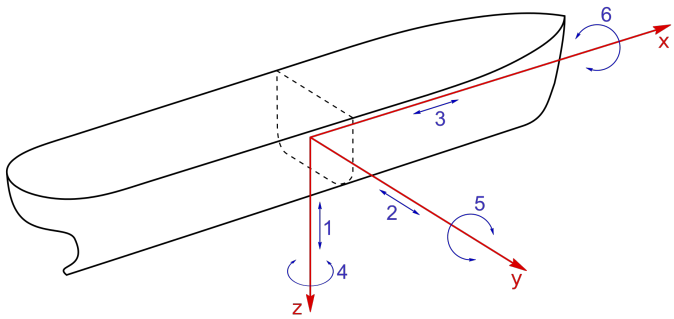


Figure 6.3: *Coordinate frame local to the vehicle, with six degrees of freedom indicated. Picture taken from* `https://en.wikipedia.org/wiki/Degrees_of_freedom_(mechanics)`.

6.2.1 Definition of rigid body rotation

A rigid body rotation is a tranformation that, loosely speaking, preserves volume and structure of a body, hence keeping the body rigid. This definition results in all such transformations to have two key properties. The first property is that the length of a vector is preserved, meaning

$$\boldsymbol{u}^{a\mathsf{T}}\boldsymbol{u}^{a} = \boldsymbol{u}^{b\mathsf{T}}\boldsymbol{u}^{b}, \tag{6.1}$$

irrespectively of the frames a and b. The second property is that the relative position and orientation of vectors is preserved. This means that

$$\boldsymbol{u}^{a\mathsf{T}}\boldsymbol{v}^{a} = \boldsymbol{u}^{b\mathsf{T}}\boldsymbol{v}^{b} \tag{6.2}$$

and

$$\boldsymbol{u}^{a} \times \boldsymbol{v}^{a} = \boldsymbol{w}^{a} \iff \boldsymbol{u}^{b} \times \boldsymbol{v}^{b} = \boldsymbol{w}^{b} \tag{6.3}$$

irrespectively of the frames a and b.

All such transformations form the *rotation group* of $\mathbb{R}^3$, called $SO(3)$, and is what is called a *Lie group*. A Lie group is what is called a *differentiable manifold*, meaning it is a set which members are continuously distributed, or, loosely speaking, "infinitely many and infinitely close to each other". As such, the concept of a Lie group can in some sense be thought of as a generalization of the real line $\mathbb{R}$. More information on Lie groups can be found in [21]. SO stands for *Special Orthogonal* and the 3 comes from the fact that the rotations are in three dimensions. $SO(3)$ is given a more thorough treatment in Appendix B.3.

6.2.2 The angle-axis representation

The angle-axis representation follows from *Euler's rotation theorem*, stating that any rigid body rotation can be reduced to a single rotation ϕ about a fixed rotation axis $\boldsymbol{n}$. By enforcing that $\boldsymbol{n}$ has unit length, the resulting angle-axis representation is the vector $\boldsymbol{\phi}$ such that

$$\boldsymbol{\phi} = \phi \boldsymbol{n}. \tag{6.4}$$

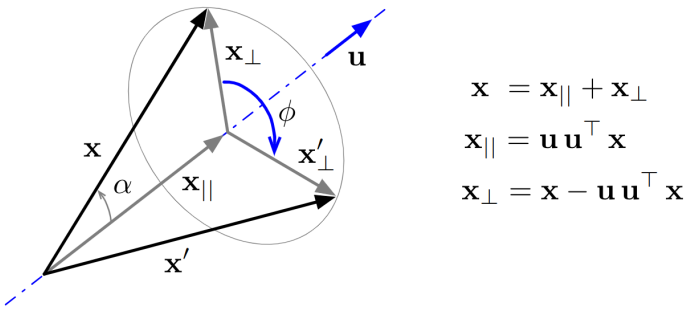


Figure 6.4: *Figure showing how a vector is rotated according to the angle-axis. The figure uses $\boldsymbol{u}$ as unit vector instead of $\boldsymbol{n}$. Figure taken from [18].*

In later derivations, knowing about this representation will prove useful to see how the different parameterizations are connected. It will also be extensively used to describe angular velocities in body frame.

6.2.3 Quaternions

The quaternion as concept was first introduced by Hamilton in 1844 [22]. A quaternion can interpreted in numerous ways, with Hamilton's interpretation being a hypercomplex number formed from a sum of a real number with three orthogonal, imaginary numbers,

$$\boldsymbol{q} = a + bi + cj + dk, \tag{6.5}$$

with $a, b, c, d \in \mathbb{R}$ and i, j, k imaginary. In addition, i, j, k satisify the properties

$$i^2 = j^2 = k^2 = ijk = -1, \tag{6.6}$$

$$ij = -ji = k. \tag{6.7}$$

Notice that the first property is the same as for ordinary complex numbers, $z = a + bi$, where $i^2 = -1$, and the second property is similar to the cross product of the basis vectors in $\mathbb{R}^3$,

$$\boldsymbol{i} \times \boldsymbol{j} = -\boldsymbol{j} \times \boldsymbol{i} = \boldsymbol{k}.$$

Another interpretation that will be used exclusively for the rest of this thesis is to consider quaternions as four-dimensional vectors. That is,

$$\boldsymbol{q} = \begin{bmatrix} \eta \\ \boldsymbol{\epsilon} \end{bmatrix} \in \mathbb{R}^4, \tag{6.8}$$

for some quaternion $\boldsymbol{q}$. Here, the real part is called the scalar part so that $\eta = a \in \mathbb{R}$ and the imaginary parts are combined into the vector part $\boldsymbol{\epsilon} = [\epsilon_1 \quad \epsilon_2 \quad \epsilon_3]^\mathsf{T} = [b \quad c \quad d]^\mathsf{T} \in \mathbb{R}^3$. This interpretation is also the most useful considering $\boldsymbol{q}$ will eventually be inserted into a state vector.

Quaternion as rotation operator

All unit quaternions encode a rotation in three-dimensional space, and such a quaternion can be written as

$$\boldsymbol{q} = \begin{bmatrix} \cos\left(\frac{\phi}{2}\right) \\ \boldsymbol{n} \sin\left(\frac{\phi}{2}\right) \end{bmatrix}, \tag{6.9}$$

where ϕ denotes the angle to rotate with and $\boldsymbol{n}$ is the axis to rotate about, similar to the angle-axis representation.

A rotation of a vector $\boldsymbol{x}^b$ from frame b to frame a is done by first inserting the vector into the vector part of a quaternion with 0 as scalar part, for then to premultiply by $\boldsymbol{q}_b^a$ and postmultiply by its conjugate. Pay special attention to the used notation. A quaternion that transforms from frame b to frame a has b in its subscript and a in its superscript. Thus, the calculation becomes

$$\begin{bmatrix} 0 \\ \boldsymbol{x}^a \end{bmatrix} = \boldsymbol{q}_b^a \otimes \begin{bmatrix} 0 \\ \boldsymbol{x}^b \end{bmatrix} \otimes \boldsymbol{q}_b^{a*} \tag{6.10}$$

where $\boldsymbol{x}^a$ is $\boldsymbol{x}^b$ expressed in frame a and $\otimes$ the *quaternion product* defined in (B.1). Note that (6.10) is often written as

$$\boldsymbol{x}^a = \boldsymbol{q}_b^a \otimes \boldsymbol{x}^b \otimes \boldsymbol{q}_b^{a*} \tag{6.11}$$

for notational simplicity.

The fact that one uses $\frac{\phi}{2}$ and not ϕ in (6.9), in addition to multiplying $\boldsymbol{x}$ by both $\boldsymbol{q}$ and $\boldsymbol{q}^*$ in (6.11), is a consequence of how rotation in four-dimensional space occurs in two orthogonal planes instead of only one as in three-dimensional space, depicted in Figure 6.4. Further details can be found in [18] and is outside the scope of this thesis.

Differential analysis

In order to use quaternions as part of the state space in the ESKF, a differential equation describing their dynamics is needed, and the following chapter intends to derive this.

Over small changes $\Delta\boldsymbol{q}$ in orientation, the corresponding rotation angle $\Delta\phi$ can safely be assumed small. Under the small angle approximations

$$\begin{aligned} \cos(\Delta\phi) &\approx 1, \\ \sin(\Delta\phi) &\approx \Delta\phi, \\ \Delta\phi &\ll 1, \end{aligned}$$

a small rotation $\Delta \boldsymbol{q}$ can be approximated from (6.9) as

$$\Delta \boldsymbol{q} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\Delta \boldsymbol{\phi} \end{bmatrix}, \tag{6.12}$$

where $\Delta \boldsymbol{\phi} = \Delta\phi \boldsymbol{n}$ is the angle-axis representation of the small rotation.

For small rotations, rotation can be treated as an additive quantity, which allows for writing the time derivative of $\boldsymbol{q}(t)$ as

$$\dot{\boldsymbol{q}} = \lim_{\Delta t \to 0} \frac{\boldsymbol{q}(t + \Delta t) - \boldsymbol{q}(t)}{\Delta t}$$

As $\boldsymbol{q}(t + \Delta t)$ is just $\boldsymbol{q}(t)$ perturbed by $\Delta \boldsymbol{q}$, $\dot{\boldsymbol{q}}$ can be derived as

$$\begin{aligned}
\dot{\boldsymbol{q}} &= \lim_{\Delta t \to 0} \frac{\boldsymbol{q}(t + \Delta t) - \boldsymbol{q}(t)}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{\boldsymbol{q}(t) \otimes \Delta \boldsymbol{q} - \boldsymbol{q}(t)}{\Delta t} \\
&= \boldsymbol{q}(t) \otimes \left(\lim_{\Delta t \to 0} \frac{\Delta \boldsymbol{q} - \boldsymbol{q}_{\mathbf{I}}}{\Delta t} \right) \\
&= \boldsymbol{q}(t) \otimes \left(\lim_{\Delta t \to 0} \frac{\begin{bmatrix} 1 \\ \frac{1}{2}\Delta \boldsymbol{\phi} \end{bmatrix} - \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix}}{\Delta t} \right) \\
&= \boldsymbol{q}(t) \otimes \left(\lim_{\Delta t \to 0} \begin{bmatrix} 0 \\ \frac{1}{2}\frac{\Delta \boldsymbol{\phi}}{\Delta t} \end{bmatrix} \right) \\
&= \boldsymbol{q}(t) \otimes \begin{bmatrix} 0 \\ \frac{1}{2}\boldsymbol{\omega} \end{bmatrix} \\
&= \frac{1}{2}\boldsymbol{q}(t) \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}
\end{aligned}$$

where $\boldsymbol{\omega}$ can be interpreted as the angle-axis representation of the instantaneous angular velocity of the body with attitude $\boldsymbol{q}(t)$ decomposed in body frame at time t. For notational simplicity, $\dot{\boldsymbol{q}}$ is usually expressed as

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \boldsymbol{\omega}. \tag{6.13}$$

6.2.4 Rotation matrices

From Chapter 6.2.1 it is clear that rotation is a linear operation, as it is defined from the scalar and cross product, which both are linear operations. This means that $\boldsymbol{u}^b$ can be computed from $\boldsymbol{u}^a$ as

$$\boldsymbol{u}^b = \mathbf{R}_a^b \boldsymbol{u}^a \tag{6.14}$$

where $\mathbf{R}_a^b$ is called the *rotation matrix* from frame a to frame b and is indeed a member of $SO(3)$ that was introduced in Chapter 6.2.1.

Differential analysis

An analysis of the differential behaviour of $\mathbf{R}$ is necessary for later use when linearizing the process model in ESKF, similarly to the quaternion. By time differentiating (B.8) one gets

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{R}^\mathsf{T}\mathbf{R}) &= \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{I} \\ \dot{\mathbf{R}}^\mathsf{T}\mathbf{R} + \mathbf{R}^\mathsf{T}\dot{\mathbf{R}} &= \mathbf{O} \\ \mathbf{R}^\mathsf{T}\dot{\mathbf{R}} &= -\left(\mathbf{R}^\mathsf{T}\dot{\mathbf{R}}\right)^\mathsf{T}, \end{aligned} \tag{6.15}$$

which shows that $\mathbf{R}^\mathsf{T}\dot{\mathbf{R}}$ is a skew-symmetric matrix. This in turn means that there exists a vector $\boldsymbol{\omega} \in \mathbb{R}^3$ such that

$$\mathbf{R}^\mathsf{T}\dot{\mathbf{R}} = S(\boldsymbol{\omega}) \tag{6.16}$$

where $S(\cdot)$ is the *skew operator* defined in (B.10).

The physical interpretation of $\boldsymbol{\omega}$ in $S(\boldsymbol{\omega})$ is the angle-axis representation of the angular velocity of some body with orientation given by $\mathbf{R}$, decomposed in body frame. Note that (6.16) could be written as

$$\dot{\mathbf{R}}\mathbf{R}^\mathsf{T} = S(\boldsymbol{\omega}) \tag{6.17}$$

by time differentiating $\mathbf{R}\mathbf{R}^\mathsf{T} = \mathbf{I}$ instead of $\mathbf{R}^\mathsf{T}\mathbf{R} = \mathbf{I}$. In (6.17) however, contrary to (6.16), $\boldsymbol{\omega}$ *must be decomposed in inertial frame*. For an INS, raw angular velocity measurements are always given in the same frame as the IMU, which is defined as the body frame. Thus, (6.16) follows the classical approach and is the formulation used for the remainder of this thesis.

Rewriting (6.16) as

$$\dot{\mathbf{R}} = \mathbf{R}S(\boldsymbol{\omega}), \tag{6.18}$$

and assuming $\boldsymbol{\omega}$ to be constant with respect to time, (6.18) reduces to a simple ordinary differential equation (ODE), with solution

$$\mathbf{R}(t) = \mathbf{R}(0)\text{expm}(S(\boldsymbol{\omega}t)). \tag{6.19}$$

where $\text{expm}(S(\boldsymbol{\omega}t))$ is the matrix exponential of $S(\boldsymbol{\omega}t)$, defined from the Taylor expansion of the exponential function as

$$\text{expm}(S(\boldsymbol{\omega}t)) = \sum_{n=0}^{\infty} \frac{S(\boldsymbol{\omega}t)^n}{n!}. \tag{6.20}$$

By choosing $\mathbf{R}(0) = \mathbf{I}$ and $\Delta\boldsymbol{\theta} = \boldsymbol{\omega}\Delta t$ for some time period Δt, (6.19) becomes

$$\mathbf{R} = \text{expm}(S(\Delta\boldsymbol{\theta})), \tag{6.21}$$

which is the *exponential mapping* from $\mathfrak{so}(3)$ to $SO(3)$.

Linearizing $\mathbf{R}(t)$ is now possible from (6.19) and (6.20). Rewriting as

$$\begin{aligned}\mathbf{R}(t + \Delta t) &= \mathbf{R}(t)\text{expm}(S(\boldsymbol{\omega}\Delta t)) \\ &= \mathbf{R}(t)\Delta\mathbf{R},\end{aligned}$$

the exponential mapping $\Delta\mathbf{R} = \text{expm}(S(\boldsymbol{\omega}\Delta t))$ can be interpreted as a local perturbation of the body frame over the time period Δt. By truncating $\Delta\mathbf{R}$ to first order and denoting $\Delta\boldsymbol{\theta} = \boldsymbol{\omega}\Delta t$, $\Delta\mathbf{R}$ can be approximated as

$$\Delta\mathbf{R} \approx \mathbf{I} + S(\Delta\boldsymbol{\theta}). \tag{6.22}$$

6.3 Sensors used in an INS

As mentioned briefly in the start of Chapter 6, an INS is defined by fusing sensor data from an IMU with some external measurement. Two common sensors to accompany the IMU are a GNSS module or a camera. The former sensor is the more traditional choice and is common for ships and drones that travel in an area where GNSS is available. Should this not be the case, that is, that the vehicle is travelling in a *GNSS denied area*, then fusing IMU with data from a camera is an

increasingly more popular option. The resulting system executes what is called *visual-inertial odometry*. The topic is not pursued further, and for the remainder of this thesis, fusion between IMU and GNSS will get full attention.

6.3.1 The IMU

The IMU is a sensor that is capable of measuring specific force and angular velocity in its local frame at a high rate, typically 100 Hz and above, by using an *accelerometer* and a *gyroscope*, respectively. Specific force is the acceleration that the IMU experiences in free fall, meaning that an IMU at rest will measure a positive acceleration upwards equal to gravity. Usually, the IMU has three such accelerometers and gyroscopes mounted orthogonally to each other, such that the IMU is capable of delivering measurements with six degrees of freedom, see Figure 6.3.

While the IMU frame may be any arbitrary frame where the raw measurements are decomposed in, for all practical purposes, the IMU can be assumed to be rigidly mounted to the vehicle such that the body frame of the vehicle can be defined equal to the IMU frame. Thus, in the following, IMU frame and body frame are used interchangeably.

Modelling the IMU

The following will use the notation introduced in Chapter 6.1.1 together with notation and models from [23]. The accelerometer of the IMU is modelled as

$$\boldsymbol{f}_{\text{IMU}} = \boldsymbol{a}_{ib}^{b} - \boldsymbol{g}^{b} + \boldsymbol{b}_{a}^{b} + \boldsymbol{w}_{a}, \tag{6.23}$$

where $\boldsymbol{f}_{\text{IMU}}$ is the specific force that the IMU measures, $\boldsymbol{a}_{ib}^{b}$ is the true acceleration of the body frame relative the inertial frame in body frame, $\boldsymbol{g}^{b}$ is gravity in body frame, $\boldsymbol{b}_{a}^{b}$ is a bias affecting $\boldsymbol{a}_{ib}^{b}$ in body frame and $\boldsymbol{w}_{a}$ is white noise affecting $\boldsymbol{a}_{ib}^{b}$. Note that the notation $\boldsymbol{b}_{a}^{b}$ and $\boldsymbol{w}_{a}$ is used instead of the maybe more obvious alternative $\boldsymbol{b}_{\boldsymbol{a}_{ib}^{b}}^{b}$ and $\boldsymbol{w}_{\boldsymbol{a}_{ib}^{b}}$ for notational simplicity. The bias term $\boldsymbol{b}_{a}^{b}$ is modelled as a Wiener process, such that

$$\begin{aligned} \dot{\boldsymbol{b}}_{a}^{b} &= \boldsymbol{w}_{b_a}, \\ \boldsymbol{w}_{b_a} &\sim \mathcal{N}(\boldsymbol{0}, \Sigma_{b_a}). \end{aligned}$$

The white noise $\boldsymbol{w}_a$ is assumed to be Gaussian white noise, such that

$$\boldsymbol{w}_a \sim \mathcal{N}(\boldsymbol{0}, \Sigma_a).$$

Lastly, both $\boldsymbol{w}_a$ and $\boldsymbol{w}_{b_a}$ are assumed isotropic, meaning

$$\boldsymbol{w}_a^u = \boldsymbol{w}_a^v$$

and

$$\boldsymbol{w}_{b_a}^u = \boldsymbol{w}_{b_a}^v$$

irrespectively of frames u and v.

The gyroscope is modelled as

$$\boldsymbol{\omega}_{\text{IMU}} = \boldsymbol{\omega}_{ib}^b + \boldsymbol{b}_g^b + \boldsymbol{w}_g \tag{6.24}$$

where $\boldsymbol{\omega}_{\text{IMU}}$ is the angular velocity that the IMU measures, $\boldsymbol{\omega}_{ib}^b$ is the true angular velocity of the body frame relative inertial frame in body frame, $\boldsymbol{b}_g^b$ is a bias term and $\boldsymbol{w}_g$ white noise. Again, the bias term $\boldsymbol{b}_g^b$ is modelled as a Wiener process, such that

$$\begin{aligned} \dot{\boldsymbol{b}}_g^b &= \boldsymbol{w}_{b_g}, \\ \boldsymbol{w}_{b_g} &\sim \mathcal{N}(\boldsymbol{0}, \Sigma_{b_g}). \end{aligned}$$

The white noise $\boldsymbol{w}_g$ is assumed Gaussian, such that

$$\boldsymbol{w}_g \sim \mathcal{N}(\boldsymbol{0}, \Sigma_g).$$

Lastly, both $\boldsymbol{w}_g$ and $\boldsymbol{w}_{b_g}$ are assumed isotropic, such that

$$\boldsymbol{w}_g^u = \boldsymbol{w}_g^v$$

and

$$\boldsymbol{w}_{b_g}^u = \boldsymbol{w}_{b_g}^v$$

irrespectively of the frames u and v.

Bias modelling

The Wiener process is not the only way of modelling bias for IMU. Another common model used is the *Gauss-Markov process*, which models the bias dynamics as the first-order system

$$\dot{\boldsymbol{b}} = -p\boldsymbol{b} + \boldsymbol{w} \tag{6.25}$$

where $\boldsymbol{b}$ is some bias, p is the inverse time constant of the system, $p = 1/T$, and $\boldsymbol{w}$ is Gaussian white noise that drives the process. Both models have their pros and cons. If external measurements are for some reason lost, the Gauss-Markov process will draw the bias back to zero, as the bias itself is unobservable without external measurements. The Wiener process does not suffer from this. On the other hand, the Wiener process variance is unbounded, meaning it predicts infinite bias in infinite time and is in that case not a good representation of actual bias behavior in an IMU. The variance of the Gauss-Markov process is indeed bounded, and can in that sense be considered a better model for an actual IMU. Note that the Gauss-Markov process becomes a Wiener process when taking the limit

$$T \to \infty \implies p \to 0$$

which again implies that a large time constant T is a simple way of achieving some sort of compromise between the two models. More discussion about bias modelling can be found in [13], [24], [25] and is not discussed further here.

Mounting, scaling and orthogonality errors

Common errors when using an IMU are *mounting, scaling and orthogonality errors*. As these errors are linear in nature, they can be resolved by a *compensation matrix* $\mathbf{C}_a$ for the accelerometer and $\mathbf{C}_g$ for the gyroscope, and can be derived from (6.23) and (6.24), respectively, as follows.

Because of mounting errors, the IMU will in general not be aligned with the "true" body frame of the UAV. A consequence of this is for example that the heading estimate of the UAV will be slightly off. When considering such mounting error, (6.23) can be rewritten to a more general form

$$\boldsymbol{f}_{\text{IMU}} = \mathbf{R}_b^{\text{IMU}}(\boldsymbol{a}_{ib}^b - \boldsymbol{g}^b) + \boldsymbol{w}_a + \boldsymbol{b}_a^b \tag{6.26}$$

where $\mathbf{R}_b^{\text{IMU}}$ is the rotation matrix of the mounting error, transforming the true acceleration and gravity from the true body frame into IMU frame.

Orthogonality errors are manufacturing errors where the axes of the accelerometer are not perfectly orthogonal to each other, which causes correlation between the different sensor readings, as, for example, gravity contaminates the readings of the other axes, without the filter being aware of this being gravity. This can be remedied with a orthogonality matrix $\mathbf{U}_a$ with rows that correspond to the actual direction that each axis of the IMU is measuring. Now (6.26) takes the form

$$\boldsymbol{f}_{\text{IMU}} = \mathbf{U}_a \mathbf{R}_b^{\text{IMU}} (\boldsymbol{a}_{ib}^b - \boldsymbol{g}^b) + \boldsymbol{w}_a + \boldsymbol{b}_a^b. \tag{6.27}$$

The last error to account for is scaling error and is, as the name suggests, that the IMU measures an acceleration that is only proportional to the true acceleration, being off by some scale. As the error is linear, it can once again be remedied by a matrix $\mathbf{D}_a$, where the diagonal elements are the factors that scales the measurement of each axis to be correct. Thus, the most common way of writing (6.23) is then

$$\begin{aligned} \boldsymbol{f}_{\text{IMU}} &= \mathbf{D}_a \mathbf{U}_a \mathbf{R}_b^{\text{IMU}} (\boldsymbol{a}_{ib}^b - \boldsymbol{g}^b) + \boldsymbol{w}_a + \boldsymbol{b}_a^b & (6.28) \\ &= \mathbf{C}_a^{-1} (\boldsymbol{a}_{ib}^b - \boldsymbol{g}^b) + \boldsymbol{w}_a + \boldsymbol{b}_a^b & (6.29) \end{aligned}$$

where

$$\mathbf{C}_a^{-1} \equiv \mathbf{D}_a \mathbf{U}_a \mathbf{R}_b^{\text{IMU}} \tag{6.30}$$

is the inverse of the compensation matrix $\mathbf{C}_a$ mentioned above. The exact same procedure is done to get $\mathbf{C}_g$,

$$\mathbf{C}_g^{-1} \equiv \mathbf{D}_g \mathbf{U}_g \mathbf{R}_b^{\text{IMU}} \tag{6.31}$$

The reason for defining it as the inverse is just a matter of notation, as this allows for writing the true acceleration as

$$\boldsymbol{a}_{ib}^b = \mathbf{C}_a \left(\boldsymbol{f}_{\text{IMU}} - \boldsymbol{w}_a - \boldsymbol{b}_a^b \right) + \boldsymbol{g}^b. \tag{6.32}$$

For the rest of this thesis, however, the matrices $\mathbf{C}_a$ and $\mathbf{C}_g$ will not be included when using the true acceleration $\boldsymbol{a}_{ib}^b$ and angular velocity $\boldsymbol{\omega}_{ib}^b$ in expressions.

6.3.2 The GNSS module

A GNSS module is a sensor that provides position measurements by communicating with satellites that orbit Earth. Satellites broadcast navigation messages modulated over a carrier that contains time of transmission and the *ephemeris* of the satellite, which are parameters that are used to calculate the position and velocity of the satellite at the time of transmission. When the receiver at the surface receives this navigation message, it uses the difference between time of transmission and arrival to estimate the line of sight distance between the satellite and the receiver, called the *pseudorange*. When combined with the position of the satellite at time of transmission, the receiver is able to calculate its own position.

Sensor fusion where IMU is fused with precalculated position measurements from a GNSS module is called *loosely coupled integration*. A more sophisticated method is *tightly coupled integration*. This is where raw measurements such as pseudorange, *Doppler frequency*, which is the shift in frequency of the carrier because of relative motion between the satellite and the receiver, and, in the case of multiple GNSS receivers used in conjunction, the *carrier phase*, are used directly in the filter for better estimation. GNSS modelling is incredibly complex, and further discussion of this topic and how GNSS navigation works in general is outside the scope of this thesis. More information can be found in [13], [26], [27].

GNSS measurement frames

GNSS position measurements can be given in multiple frames, and two will be discussed here. The first frame is *longitude, latitude, height* (LLH). The position along the surface of Earth is given as two angles denoting offset from the *international prime meridian*, which is the meridian that passes through the British Royal Observatory in Greenwich, England, and equator, respectively. The height is given as the height above the Earth's reference ellipsoid. The full specification is given in the *WGS-84 standard* [28]. Although LLH frame is common in navigation, it is not cartesian and will therfore not be used here.

Another common frame is the *earth-centered, earth-fixed* (ECEF) frame. Conversely to LLH, ECEF is a cartesian coordinate frame. It is defined to have its origin in the center of Earth, with its z-axis aligned with the rotational axis of Earth and with the x-axis such that it passes through the point where longitude

and lattiude both equal 0. The direction of the y-axis is then chosen to complete a right-hand system. An important remark about the ECEF frame is that, as a consequence of its definition, the ECEF is not strictly inertial. It rotates with Earth at a rate of [13]

$$\omega_{ie} \approx \frac{1 + 365.25 \text{ cycles}}{365.25 \cdot 24 \text{ hr}} \cdot \frac{2\pi \text{ rad/cycle}}{3600 \text{ s/hr}} = 7.292115 \cdot 10^{-5} \text{ rad/s}.$$

This can be compensated for in (6.24) by adding the term $\mathbf{R}_e^b \boldsymbol{\omega}_{ie}^e$, with $\boldsymbol{\omega}_{ie}^e = [0 \quad 0 \quad \omega_{ie}]^\mathsf{T}$. Although this can be justified for vehicles that travel for a long time with high-precision IMUs capable of measuring the rotation of Earth, such as a maritime vessel that travels over an ocean, it will be neglected for the remainder of this thesis. As the ECEF frame is cartesian and readily available from GNSS module measurements, it is the frame of choice for this thesis.

Modelling the GNSS module

The general GNSS model used is

$$\boldsymbol{p}_{\text{GNSS}} = \boldsymbol{p}_{eb}^e + \mathbf{R}_b^e \boldsymbol{r}_{bm}^b + \boldsymbol{w}_p^e \tag{6.33}$$

where $\boldsymbol{p}_{eb}^e$ denotes the position of body frame relative the ECEF frame, $\mathbf{R}_b^e \boldsymbol{r}_{bm}^b$ denotes the *lever arm*, that is, the position of the GNSS antenna relative the origin of the body frame, transformed into ECEF frame and $\boldsymbol{w}_p^e$ is white noise affecting $\boldsymbol{p}_{ie}^e$ decomposed in ECEF frame. The white noise $\boldsymbol{w}_p^e$ is assumed Gaussian, such that

$$\boldsymbol{w}_p^e \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_p).$$

It is, however, *not assumed to be isotropic*. Although having a lever arm is perfectly normal for an INS, it will be taken to be zero, $\boldsymbol{r}_{bm}^b = \mathbf{0}$, for the remainder of this thesis. Equation (6.33) thus reduces to

$$\boldsymbol{p}_{\text{GNSS}} = \boldsymbol{p}_{eb}^e + \boldsymbol{w}_p^e. \tag{6.34}$$

7 | Kinematics

The *kinematics* of an INS describe the motion of said INS. For navigation with error state estimation, kinematics of the nominal, true and error state are required. However, before stating the equations, the state space of each state needs to be populated. Although large state spaces are possible for sophisticated filters, such as for tightly coupled integration mentioned in Chapter 6.3.2 where the number of states can easily surpass 100 states, the following will resort to a more essential state-space approach for reliable estimation when using loosely coupled integration.

The chapter is outlined as follows. First, the true and nominal states are defined, together with the continuous-time kinematics that govern how they evolve with time. Then, the error state is defined, where its continuous-time kinematics are derived. Lastly, the nominal and error state kinematics are discretized for later use in Chapter 8. The true state is not discretized as its purpose is only to be able to derive the continuous-time kinematics of the error state and is never explicitly computed in the ESKF. The derivations will follow the same outline as in [18] together with its simple notation. A remark on the use of continuous-time models is in order before going on. Previous chapters have inspected models in discrete time under the assumption that these models are given. However, for a practical application such as INS, it is common and much more intuitive to derive the process model in continuous-time, for then to discretize it afterwards with mathematical tools that need not be intuitive, although the discretized result is correct.

7.1 The true and nominal state

As mentioned in Chapter 7, the true and nominal states will resort to a more essential state-space approach for reliable estimation when using loosely coupled integration. Therefore, they are defined to be

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}_{eb}^{e} \\ \boldsymbol{v}_{eb}^{e} \\ \boldsymbol{q}_{b}^{e} \\ \boldsymbol{b}_{a}^{b} \\ \boldsymbol{b}_{g}^{b} \end{bmatrix}, \tag{7.1a}$$

$$\hat{\boldsymbol{x}} = \begin{bmatrix} \hat{\boldsymbol{p}}_{eb}^{e} \\ \hat{\boldsymbol{v}}_{eb}^{e} \\ \hat{\boldsymbol{q}}_{b}^{e} \\ \hat{\boldsymbol{b}}_{a}^{b} \\ \hat{\boldsymbol{b}}_{g}^{b} \end{bmatrix}, \tag{7.1b}$$

respectively. The chosen states should come as no surprise. Position and velocity are indisputably coupled and required for navigation. Attitude is equally required to transform IMU measurements to inertial frame so that position and velocity can be propagated. Representing attitude in the form of a quaternion is chosen to have a minimal parameterization that is singularity-free. Lastly, biases in IMU measurements are estimated to have more reliable predictions, mitigating drift between update steps and improving filter performance.

7.1.1 The true state kinematics

The true state continuous-time kinematics are given as

$$\dot{\boldsymbol{p}}_{eb}^{e} = \boldsymbol{v}_{eb}^{e}, \tag{7.2a}$$

$$\dot{\boldsymbol{v}}_{eb}^{e} = \boldsymbol{a}_{eb}^{e}, \tag{7.2b}$$

$$\dot{\boldsymbol{q}}_{b}^{e} = \frac{1}{2}\boldsymbol{q}_{b}^{e} \otimes \boldsymbol{\omega}_{eb}^{b}, \tag{7.2c}$$

$$\dot{\boldsymbol{b}}_{a}^{b} = \boldsymbol{w}_{b_a}^{b}, \tag{7.2d}$$

$$\dot{\boldsymbol{b}}_{g}^{b} = \boldsymbol{w}_{b_g}^{b}. \tag{7.2e}$$

The true acceleration $\boldsymbol{a}_{eb}^{e}$ in (7.2b) can be found by rewriting (6.23) into

$$\boldsymbol{a}_{eb}^{e} = \mathbf{R}_{b}^{e}\left(\boldsymbol{f}_{\text{IMU}} - \boldsymbol{b}_{a}^{b}\right) - \boldsymbol{w}_{a} + \boldsymbol{g}^{e} \tag{7.3}$$

where the isotropic properties of w_a are used, $g^e = \mathbf{R}_b^e g^b$, the intertial frame i is defined as ECEF frame e and $\mathbf{R}_b^e$ is the rotation matrix representation of q_b^e found by using (B.17). Likewise, the true angular velocity ω_{eb}^b in (7.2c) is found by rewriting (6.24) into

$$\omega_{eb}^b = \omega_{\text{IMU}} - b_g^b - w_g. \tag{7.4}$$

By inserting (7.3) and (7.4) into (7.2b) and (7.2c), respectively, the true kinematic system is found to be

$$\dot{p}_{eb}^e = v_{eb}^e, \tag{7.5a}$$
$$\dot{v}_{eb}^e = \mathbf{R}_b^e(f_{\text{IMU}} - b_a^b) - w_a + g^e, \tag{7.5b}$$
$$\dot{q}_b^e = \frac{1}{2} q_b^e \otimes \left(\omega_{\text{IMU}} - b_g^b - w_g\right), \tag{7.5c}$$
$$\dot{b}_a^b = w_{b_a}^b, \tag{7.5d}$$
$$\dot{b}_g^b = w_{b_g}^b. \tag{7.5e}$$

7.1.2 The nominal state kinematics

The nominal state continuous-time kinematics are found by simply taking the expectation of the true state kinematic system in (7.5), which gives

$$\dot{\hat{p}}_{eb}^e = \hat{v}_{eb}^e, \tag{7.6a}$$
$$\dot{\hat{v}}_{eb}^e = \hat{\mathbf{R}}_b^e(f_{\text{IMU}} - \hat{b}_a^b) + g^e, \tag{7.6b}$$
$$\dot{\hat{q}}_b^e = \frac{1}{2} \hat{q}_b^e \otimes \left(\omega_{\text{IMU}} - \hat{b}_g^b\right), \tag{7.6c}$$
$$\dot{\hat{b}}_a^b = 0, \tag{7.6d}$$
$$\dot{\hat{b}}_g^b = 0, \tag{7.6e}$$

where $\hat{\mathbf{R}}_b^e$ is the rotation matrix representation of $\hat{q}_b^e$.

7.2 The error state

Chapter 5 discussed how the composition operator $\oplus$ that injects the error state into the nominal state does not make any assumption of the state spaces of the nominal or error state. This allows the error state to estimate the error of the

same states as the true and nominal state, *but with different parameterizations*. A full justification for the error state will be given in Chapter 8 and will for now simply be defined to be

$$\delta \boldsymbol{x} = \begin{bmatrix} \delta \boldsymbol{p} \\ \delta \boldsymbol{v} \\ \delta \boldsymbol{\theta} \\ \delta \boldsymbol{b}_a \\ \delta \boldsymbol{b}_g \end{bmatrix}, \tag{7.7}$$

where $\delta\boldsymbol{\theta}$ is the attitude error and is represented with the angle-axis representation discussed in Chapter 6.2.2 and all other states are just the arithmetic difference between the true and nominal state.

As a quaternion can be formed from an angle-axis with (6.9), the composition of the nominal state and the error state can now be defined as

$$\boldsymbol{x} = \hat{\boldsymbol{x}} \oplus \delta \boldsymbol{x} = \begin{bmatrix} \hat{\boldsymbol{p}}^e_{eb} + \delta \boldsymbol{p} \\ \hat{\boldsymbol{v}}^e_{eb} + \delta \boldsymbol{v} \\ \hat{\boldsymbol{q}}^e_b \otimes \delta \boldsymbol{q} \\ \hat{\boldsymbol{b}}^b_a + \delta \boldsymbol{b}_a \\ \hat{\boldsymbol{b}}^b_g + \delta \boldsymbol{b}_g \end{bmatrix} \tag{7.8}$$

where

$$\delta \boldsymbol{q} = \begin{bmatrix} \cos\left(\frac{\|\delta\boldsymbol{\theta}\|}{2}\right) \\ \frac{\delta\boldsymbol{\theta}}{\|\delta\boldsymbol{\theta}\|} \sin\left(\frac{\|\delta\boldsymbol{\theta}\|}{2}\right) \end{bmatrix} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix} \tag{7.9}$$

as $\|\delta\boldsymbol{\theta}\|$ is assumed small.

7.2.1 The error state kinematics

Instead of deriving the general, nonlinear kinematics for the error state, as was done for the true state in (7.5) and the nominal state in (7.6), the following will derive the *linearized kinematics*. The reason for this is to later be able to directly insert the derived equations into the appropriate matrices in Chapter 5.2 when deriving the ESKF for INS in Chapter 8. The result is so fundamental that it is stated in Theorem 2 together with proof, based on the proof given in [18].

Theorem 2 (The linearized error state kinematics)**.** *Given the true state* $\boldsymbol{x}$ *and the nominal state* $\hat{\boldsymbol{x}}$ *with kinematics given by* (7.5) *and* (7.6)*, respectively. Given also the error state* $\delta\boldsymbol{x}$ *that is defined by* (7.7) *and related to* $\boldsymbol{x}$ *and* $\hat{\boldsymbol{x}}$ *by* (7.8)*. Then the linearized kinematics of* $\delta\boldsymbol{x}$ *are given as*

$$\delta\dot{\boldsymbol{p}} = \delta\boldsymbol{v}, \tag{7.10a}$$

$$\delta\dot{\boldsymbol{v}} = -\hat{\mathbf{R}}_b^e S\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right)\delta\boldsymbol{\theta} - \hat{\mathbf{R}}_b^e\delta\boldsymbol{b}_a - \boldsymbol{w}_a, \tag{7.10b}$$

$$\delta\dot{\boldsymbol{\theta}} = -S\left(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g\right)\delta\boldsymbol{\theta} - \delta\boldsymbol{b}_g - \boldsymbol{w}_g, \tag{7.10c}$$

$$\delta\dot{\boldsymbol{b}}_a = \boldsymbol{w}_{b_a}, \tag{7.10d}$$

$$\delta\dot{\boldsymbol{b}}_g = \boldsymbol{w}_{b_g}. \tag{7.10e}$$

Proof. Deriving (7.10a), (7.10d) and (7.10e) is trivial, as the corresponding kinematic equations for both the true and nominal state are already linear. Deriving (7.10b) and (7.10c), however, is not trivial. A couple of assumptions and approximations are required to arrive at the desired results. First, the rotation matrix for the true state $\mathbf{R}_b^e$ can be approximated by linearizing about the nominal attitude with (6.22) as

$$\mathbf{R}_b^e \approx \hat{\mathbf{R}}_b^e\left(\mathbf{I} + S\left(\delta\boldsymbol{\theta}\right)\right). \tag{7.11}$$

Second, all error states are assumed to be *small signals*, which simply means they are small in magnitude. This implicates that, loosely speaking, a "product" of two small signals is negligible. Here, this will be used to say that

$$S(\delta\boldsymbol{u})\delta\boldsymbol{v} \approx \mathbf{0}. \tag{7.12}$$

where $S(\cdot)$ is the skew operator defined in (B.9). Additionally, as $\mathrm{E}\left[\boldsymbol{w}\right] = \mathbf{0}$, the similar assumption

$$S(\delta\boldsymbol{u})\boldsymbol{w} \approx \mathbf{0} \tag{7.13}$$

will be made.

The expression for $\delta\dot{\boldsymbol{v}}$ can be derived by first taking the difference between

(7.5b) and (7.6b) to arrive at

$$\begin{aligned}\delta\dot{\boldsymbol{v}} &= \dot{\boldsymbol{v}} - \dot{\hat{\boldsymbol{v}}} \\ &= \underbrace{\mathbf{R}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \boldsymbol{b}_a^b\right) - \boldsymbol{w}_a + \boldsymbol{g}^e}_{\dot{\boldsymbol{v}}} - \underbrace{\left(\hat{\mathbf{R}}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) + \boldsymbol{g}^e\right)}_{\dot{\hat{\boldsymbol{v}}}}. \end{aligned} \tag{7.14}$$

By applying the approximation in (7.11), canceling $\boldsymbol{g}^e$ and substituting $\boldsymbol{b}_a = \hat{\boldsymbol{b}}_a + \delta\boldsymbol{b}_a$, (7.14) can be written as

$$\begin{aligned}&\mathbf{R}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \boldsymbol{b}_a^b\right) - \boldsymbol{w}_a + \boldsymbol{g}^e - \left(\hat{\mathbf{R}}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) + \boldsymbol{g}^e\right) \\ \approx\ &\underbrace{\hat{\mathbf{R}}_b^e\left(\mathbf{I} + S(\delta\boldsymbol{\theta})\right)}_{\mathbf{R}_b^e}\left(\boldsymbol{f}_{\text{IMU}} \underbrace{-\hat{\boldsymbol{b}}_a^b - \delta\boldsymbol{b}_a}_{-\boldsymbol{b}_a^b}\right) - \boldsymbol{w}_a + \boldsymbol{g}^e - \hat{\mathbf{R}}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) - \boldsymbol{g}^e \qquad (7.15) \\ =\ &\hat{\mathbf{R}}_b^e\left(\mathbf{I} + S(\delta\boldsymbol{\theta})\right)\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b - \delta\boldsymbol{b}_a\right) - \boldsymbol{w}_a - \hat{\mathbf{R}}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right). \qquad (7.16)\end{aligned}$$

After multiplying out terms and canceling, (7.16) can be further manipulated into

$$\begin{aligned}&\hat{\mathbf{R}}_b^e\left(\mathbf{I} + S(\delta\boldsymbol{\theta})\right)\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b - \delta\boldsymbol{b}_a\right) - \boldsymbol{w}_a - \hat{\mathbf{R}}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) \\ =\ &\hat{\mathbf{R}}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) - \hat{\mathbf{R}}_b^e\delta\boldsymbol{b}_a + \hat{\mathbf{R}}_b^e S(\delta\boldsymbol{\theta})\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) \qquad (7.17) \\ &- \hat{\mathbf{R}}_b^e \underbrace{S(\delta\boldsymbol{\theta})\delta\boldsymbol{b}_a}_{\approx\mathbf{0}} - \boldsymbol{w}_a - \hat{\mathbf{R}}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) \\ =\ &-\hat{\mathbf{R}}_b^e\delta\boldsymbol{b}_a + \hat{\mathbf{R}}_b^e S(\delta\boldsymbol{\theta})\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) - \boldsymbol{w}_a. \qquad (7.18)\end{aligned}$$

where the assumption in (7.12) is invoked. Lastly, applying the property (B.11) of $S(\cdot)$, (7.18) can finally be written as

$$\begin{aligned}&-\hat{\mathbf{R}}_b^e\delta\boldsymbol{b}_a + \hat{\mathbf{R}}_b^e S(\delta\boldsymbol{\theta})\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) - \boldsymbol{w}_a \\ =\ &-\hat{\mathbf{R}}_b^e\delta\boldsymbol{b}_a - \hat{\mathbf{R}}_b^e S\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right)\delta\boldsymbol{\theta} - \boldsymbol{w}_a \\ =\ &-\hat{\mathbf{R}}_b^e S\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right)\delta\boldsymbol{\theta} - \hat{\mathbf{R}}_b^e\delta\boldsymbol{b}_a - \boldsymbol{w}_a.\end{aligned}$$

which is the result in (7.10b).

The expression for $\delta\dot{\boldsymbol{\theta}}$ is derived using a slightly different strategy because of the quaternion product in the composition between the nominal and error state.

Starting with

$$\boldsymbol{q}_b^e = \hat{\boldsymbol{q}}_b^e \otimes \delta\boldsymbol{q} \tag{7.19}$$

and time differentiating, one gets

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{q}_b^e &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\hat{\boldsymbol{q}}_b^e \otimes \delta\boldsymbol{q}\right) \\
\dot{\boldsymbol{q}}_b^e &= \dot{\hat{\boldsymbol{q}}}_b^e \otimes \delta\boldsymbol{q} + \hat{\boldsymbol{q}}_b^e \otimes \delta\dot{\boldsymbol{q}} \\
\underbrace{\frac{1}{2}\boldsymbol{q}_b^e \otimes \left(\boldsymbol{\omega}_{\mathrm{IMU}} - \boldsymbol{b}_g^b - \boldsymbol{w}_g\right)}_{\dot{\boldsymbol{q}}_b^e} &= \underbrace{\frac{1}{2}\hat{\boldsymbol{q}}_b^e \otimes \left(\boldsymbol{\omega}_{\mathrm{IMU}} - \hat{\boldsymbol{b}}_g^b\right)}_{\dot{\hat{\boldsymbol{q}}}_b^e} \otimes \delta\boldsymbol{q} + \hat{\boldsymbol{q}}_b^e \otimes \delta\dot{\boldsymbol{q}} \\
\frac{1}{2}\underbrace{\hat{\boldsymbol{q}}_b^e \otimes \delta\boldsymbol{q}}_{\boldsymbol{q}_b^e} \otimes \left(\boldsymbol{\omega}_{\mathrm{IMU}} - \boldsymbol{b}_g^b - \boldsymbol{w}_g\right) &= \frac{1}{2}\hat{\boldsymbol{q}}_b^e \otimes \left(\boldsymbol{\omega}_{\mathrm{IMU}} - \hat{\boldsymbol{b}}_g^b\right) \otimes \delta\boldsymbol{q} + \hat{\boldsymbol{q}}_b^e \otimes \delta\dot{\boldsymbol{q}}
\end{aligned}$$

where (7.5c), (7.6c) and (7.19) are substituted in appropriately and the product rule for differentiation of quaternion products (B.3) is used. After eliminating the common $\hat{\boldsymbol{q}}_b^e$ prefactor, multiplying both sides by 2 and isolating $\delta\dot{\boldsymbol{q}}$, one gets

$$2\delta\dot{\boldsymbol{q}} = \delta\boldsymbol{q} \otimes \left(\boldsymbol{\omega}_{\mathrm{IMU}} - \boldsymbol{b}_g^b - \boldsymbol{w}_g\right) - \left(\boldsymbol{\omega}_{\mathrm{IMU}} - \hat{\boldsymbol{b}}_g^b\right) \otimes \delta\boldsymbol{q}. \tag{7.20}$$

To continue, (7.20) must be written into two equations, one for the scalar part and one for the vector part of $\delta\dot{\boldsymbol{q}}$. To do this, the quaternion products will be multiplied out by using (B.2). It can be shown that (B.2a) and (B.2b) can be written as [18]

$$L(\boldsymbol{q}) = \begin{bmatrix} \eta & \mathbf{0}^\mathsf{T} \\ \mathbf{0} & \eta\mathbf{I} \end{bmatrix} + \begin{bmatrix} 0 & -\boldsymbol{\epsilon}^\mathsf{T} \\ \boldsymbol{\epsilon} & S(\boldsymbol{\epsilon}) \end{bmatrix}, \tag{7.21a}$$

$$R(\boldsymbol{q}) = \begin{bmatrix} \eta & \mathbf{0}^\mathsf{T} \\ \mathbf{0} & \eta\mathbf{I} \end{bmatrix} + \begin{bmatrix} 0 & -\boldsymbol{\epsilon}^\mathsf{T} \\ \boldsymbol{\epsilon} & -S(\boldsymbol{\epsilon}) \end{bmatrix}, \tag{7.21b}$$

respectively, where $S(\cdot)$ is the skew operator from (B.9) and $\boldsymbol{q} = [\eta \quad \boldsymbol{\epsilon}^\mathsf{T}]^\mathsf{T}$. Note also that

$$\boldsymbol{q} \otimes \boldsymbol{\omega} = \boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \quad \text{and} \quad \delta\dot{\boldsymbol{q}} = \frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{2}\delta\dot{\boldsymbol{\theta}} \end{bmatrix}.$$

Inserting all this into (7.20) gives

$$
\begin{aligned}
2\delta\dot{\boldsymbol{q}} &= R(\boldsymbol{\omega}_{\text{IMU}} - \boldsymbol{b}_g^b - \boldsymbol{w}_g)\delta\boldsymbol{q} - L(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b)\delta\boldsymbol{q} \\
2\delta\dot{\boldsymbol{q}} &= \left(R\left(\boldsymbol{\omega}_{\text{IMU}} - \boldsymbol{b}_g^b - \boldsymbol{w}_g\right) - L(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b)\right)\delta\boldsymbol{q}
\end{aligned}
$$

$$
2\underbrace{\begin{bmatrix} 0 \\ \frac{1}{2}\delta\dot{\boldsymbol{\theta}} \end{bmatrix}}_{\delta\dot{\boldsymbol{q}}} = \begin{bmatrix} 0 \\ \delta\dot{\boldsymbol{\theta}} \end{bmatrix} = \left(\underbrace{\begin{bmatrix} 0 & -\left(\boldsymbol{\omega}_{\text{IMU}} - \boldsymbol{b}_g^b - \boldsymbol{w}_g\right)^{\mathsf{T}} \\ \left(\boldsymbol{\omega}_{\text{IMU}} - \boldsymbol{b}_g^b - \boldsymbol{w}_g\right) & -S\left(\boldsymbol{\omega}_{\text{IMU}} - \boldsymbol{b}_g^b - \boldsymbol{w}_g\right) \end{bmatrix}}_{R(\cdot)} - \underbrace{\begin{bmatrix} 0 & -\left(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b\right)^{\mathsf{T}} \\ \left(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b\right) & S(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b) \end{bmatrix}}_{L(\cdot)} \right) \underbrace{\begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix}}_{\delta\boldsymbol{q}}.
$$

Substituting $\boldsymbol{b}_g^b = \hat{\boldsymbol{b}}_g^b + \delta\boldsymbol{b}_g$ gives

$$
\begin{bmatrix} 0 \\ \delta\dot{\boldsymbol{\theta}} \end{bmatrix} = \left(\begin{bmatrix} 0 & -(\boldsymbol{\omega}_{\text{IMU}} \overbrace{-\hat{\boldsymbol{b}}_g^b - \delta\boldsymbol{b}_g}^{-\boldsymbol{b}_g^b} - \boldsymbol{w}_g)^{\mathsf{T}} \\ (\boldsymbol{\omega}_{\text{IMU}} \underbrace{-\hat{\boldsymbol{b}}_g^b - \delta\boldsymbol{b}_g}_{-\boldsymbol{b}_g^b} - \boldsymbol{w}_g) & -S(\boldsymbol{\omega}_{\text{IMU}} \underbrace{-\hat{\boldsymbol{b}}_g^b - \delta\boldsymbol{b}_g}_{-\boldsymbol{b}_g^b} - \boldsymbol{w}_g) \end{bmatrix} - \begin{bmatrix} 0 & -(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b)^{\mathsf{T}} \\ (\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b) & S(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b) \end{bmatrix} \right) \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix}.
$$

By using that $S(\cdot)$ is a bilinear operator from (B.12) and canceling of terms, the last expression becomes

$$
\begin{bmatrix} 0 \\ \delta\dot{\boldsymbol{\theta}} \end{bmatrix} = \begin{bmatrix} 0 & \left(\delta\boldsymbol{b}_g + \boldsymbol{w}_g\right)^{\mathsf{T}} \\ -(\delta\boldsymbol{b}_g + \boldsymbol{w}_g) & -2S(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b) + S(\delta\boldsymbol{b}_g) + S(\boldsymbol{w}_g) \end{bmatrix} \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix}. \tag{7.22}
$$

The scalar equation just states that the sum of two small signal products is 0, and is of no interest. By examining the vector part, however, the expression for $\delta\dot{\boldsymbol{\theta}}$

can now be retrieved, as

$$\begin{aligned}\dot{\delta\boldsymbol{\theta}} &= -\delta\boldsymbol{b}_g - \boldsymbol{w}_g - S(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b)\delta\boldsymbol{\theta} + \underbrace{\frac{1}{2}S(\delta\boldsymbol{b}_g)\delta\boldsymbol{\theta}}_{\approx \mathbf{0}} + \underbrace{\frac{1}{2}S(\boldsymbol{w}_g)\delta\boldsymbol{\theta}}_{\approx \mathbf{0}} \\ &= -\delta\boldsymbol{b}_g - \boldsymbol{w}_g - S(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b)\delta\boldsymbol{\theta} \\ &= -S(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b)\delta\boldsymbol{\theta} - \delta\boldsymbol{b}_g - \boldsymbol{w}_g.\end{aligned}$$

where the assumptions in (7.12) and (7.13) are invoked. ■

7.3 Discretizing the kinematics

The final step to implementing the kinematics of an INS into the ESKF framework presented in Chapter 5 is to discretize the nominal and error state kinematics. Note that the following will skip the indexing of the arbitrary time step k to keep the notation simple. The nominal state is propagated using classical mechanics with the approximation that the measured specific force and angular velocity are constant between time steps. This is a good approximation when for example using a high-end IMU capable of averaging many samples over the time length between two prediction steps. Under this assumption, the discretized nominal state kinematics are found to be

$$\hat{\boldsymbol{p}}_{eb}^e \leftarrow \hat{\boldsymbol{p}}_{eb}^e + \hat{\boldsymbol{v}}_{eb}^e \Delta t + \frac{1}{2}\left(\hat{\mathbf{R}}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) + \boldsymbol{g}^e\right)\Delta t^2, \tag{7.23a}$$

$$\hat{\boldsymbol{v}}_{eb}^e \leftarrow \hat{\boldsymbol{v}}_{eb}^e + \left(\hat{\mathbf{R}}_b^e\left(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b\right) + \boldsymbol{g}^e\right)\Delta t, \tag{7.23b}$$

$$\hat{\boldsymbol{q}}_b^e \leftarrow \hat{\boldsymbol{q}}_b^e \otimes \Delta\boldsymbol{q}, \tag{7.23c}$$

$$\hat{\boldsymbol{b}}_a^b \leftarrow \hat{\boldsymbol{b}}_a^b, \tag{7.23d}$$

$$\hat{\boldsymbol{b}}_g^b \leftarrow \hat{\boldsymbol{b}}_g^e. \tag{7.23e}$$

where Δt is the time length between two time steps,

$$\Delta\boldsymbol{q} = \begin{bmatrix} \cos(\frac{\boldsymbol{\omega}_{\text{IMU}}\Delta t}{2}) \\ \frac{\boldsymbol{\omega}_{\text{IMU}}}{\|\boldsymbol{\omega}_{\text{IMU}}\|}\sin(\frac{\boldsymbol{\omega}_{\text{IMU}}\Delta t}{2}) \end{bmatrix} \tag{7.24}$$

and $\Delta t^2 = (\Delta t)^2$ is used for notational simplicity.

As the linearized error state kinematics are only nonlinear in the nominal

state, they form a LTV, as predicted in Chapter 5.2.1. Hence, the kinematics can be written in state-space form as

$$\delta\dot{\boldsymbol{x}} = \mathbf{A}(\hat{\boldsymbol{x}})\delta\boldsymbol{x} + \mathbf{G}(\hat{\boldsymbol{x}})\boldsymbol{n}, \tag{7.25}$$

where

$$\boldsymbol{n} = \begin{bmatrix} \boldsymbol{w}_a \\ \boldsymbol{w}_g \\ \boldsymbol{w}_{b_a} \\ \boldsymbol{w}_{b_g} \end{bmatrix}, \quad \boldsymbol{n} \sim \mathcal{N}(\mathbf{0}, \Sigma_n), \tag{7.26}$$

$$\Sigma_n = \begin{bmatrix} \Sigma_a & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \Sigma_g & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \Sigma_{b_a} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \Sigma_{b_g} \end{bmatrix}, \tag{7.27}$$

$$\mathbf{A}(\hat{\boldsymbol{x}}) = \begin{bmatrix} \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & -\hat{\mathbf{R}}_b^e S(\boldsymbol{f}_{\text{IMU}} - \hat{\boldsymbol{b}}_a^b) & -\hat{\mathbf{R}}_b^e & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & -S(\boldsymbol{\omega}_{\text{IMU}} - \hat{\boldsymbol{b}}_g^b) & \mathbf{O} & -\mathbf{I} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \end{bmatrix}, \tag{7.28}$$

$$\mathbf{G}(\hat{\boldsymbol{x}}) = \begin{bmatrix} \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ -\hat{\mathbf{R}}_b^e & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & -\mathbf{I} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} \end{bmatrix}. \tag{7.29}$$

$\mathbf{A}(\hat{\boldsymbol{x}})$ can then be discretized by [29]

$$\mathbf{F} = \text{expm}(\mathbf{A}(\hat{\boldsymbol{x}})\Delta t) \tag{7.30}$$

where $\mathbf{F}$ is the discrete-time process matrix that appears in (5.5) and $\text{expm}(\cdot)$ is the matrix exponential function defined in (6.20).

This is a valid approach if the only goal is to discretize $\mathbf{A}(\hat{\boldsymbol{x}})$. However, a more convenient approach in practice is by using *Van Loan's formula* [30], as this has the added benefit of also computing the discrete-time process covariance matrix

$\mathbf{Q}$. The matrices $\mathbf{A}(\hat{\boldsymbol{x}})$, Σ_n and $\mathbf{G}(\hat{\boldsymbol{x}})$ are used to first form the matrix

$$\mathbf{M} = \begin{bmatrix} -\mathbf{A}(\hat{\boldsymbol{x}}) & \mathbf{G}(\hat{\boldsymbol{x}})\Sigma_n\mathbf{G}(\hat{\boldsymbol{x}})^\mathsf{T} \\ \mathbf{O} & \mathbf{A}(\hat{\boldsymbol{x}})^\mathsf{T} \end{bmatrix} \Delta t. \tag{7.31}$$

Then, propagating (7.31) through the matrix exponential function gives a matrix on the form

$$\operatorname{expm}(\mathbf{M}) = \begin{bmatrix} \bullet & \mathbf{F}^{-1}\mathbf{Q} \\ \mathbf{O} & \mathbf{F}^\mathsf{T} \end{bmatrix} \tag{7.32}$$

where $\bullet$ indicates a submatrix of $\operatorname{expm}(\mathbf{M})$ that is of no interest. From (7.32), the discrete-time process matrix $\mathbf{F}$ and discrete-time process covariance matrix $\mathbf{Q}$ can be extracted. Note that $\mathbf{F}$ and $\mathbf{Q}$ are only valid the same time step as $\hat{\boldsymbol{x}}$, meaning they have to be recomputed every time $\hat{\boldsymbol{x}}$ is propagated.

8 | The ESKF applied for inertial navigation systems

The kinematics of an INS were in Chapter 7 given for the true, nominal and error state, so it should come as no surprise that the aim is applying the ESKF for inertial navigation. In fact, the ESKF was engineered specifically for such state estimation, having its majority of applications, if not its only, in inertial navigation. The following chapter motivates its use in an INS, and briefly outlines how it can be implemented.

8.1 Motivating ESKF for INS

The EKF as a nonlinear filter features a very generic, efficient and simple framework for state estimation, and therefore sees extensive use in a wide variety of fields. However, when considering INSes, the generic framework of the EKF does unfortunately not suffice. In Chapter 6.2, the choice of parameterization of attitude for the state vector was discussed. As choosing a minimal parameterization will inherently contain singularities, it was concluded that overparameterization of the attitude is required.

This solution is, however, not compatible with the original EKF framework. As the attitude is determined by three degrees of freedom, any overparameterization will induce linear dependency between the parameters. As such, a state covariance matrix computed with an overparameterized attitude representation will contain linearly dependent rows, making it rank deficient and singular, potentially resulting in the filter to diverge. Additionally, the representation of attitude may not even be represented as vector, for example by using the rotation matrix that was discussed in Chapter 6.2.4 as state.

The ESKF introduced in Chapter 5, however, circumvents this troubling matter. By separating the system into its nominal, true and error state, the two former states can use a different attitude parameterization than the latter, as was done in Chapter 7.2. By only estimating the offset between the predicted and true state of the system in the error state, using three parameters for estimating the attitude is considered safe as the error state will be far from any singularities. Chapter 5.1 mentioned that a benefit of using the error state is that all higher order terms than first order are often safe to neglect, making the computation of Jacobians fast and reliable. In addition to that, the error state dynamics evolve at a slower rate than the nominal and true state dynamics, making it possible to run the ESKF update at a slower rate and instead only run the prediction step in between [18], [24]. This is especially favorable for an INS, as an IMU is in general capable of delivering measurements that drive the kinematics at a much higher rate than any other sensor integrated into the system, for example a GNSS module.

8.2 The estimation procedure

With the general ESKF algorithm described in Chapter 5 and the kinematic model of the nominal and error state in Chapter 7, the ESKF algorithm can now be determined for an INS. Note that, as in Chapter 7.3, time indexing of states, such as $\boldsymbol{p}_{eb}^{b}$, is skipped for brevity.

8.2.1 The intermediate predictions

As mentioned in Chapter 8.1, the prediction step in the ESKF is usually run much more frequently than the update step and is executed every time a new measurement from the IMU arrives. The first substep is to propagate the nominal state according to (7.23). The second substep is to discretize the continuous-time system matrices $\mathbf{A}(\hat{\boldsymbol{x}})$ and Σ_n in (7.28) and (7.27), respectively. This is done with Van Loan's formula in (7.31) and (7.32). This is necessary for the last substep, where the covariance matrix $\delta\mathbf{P}_k$ is propagated according to (5.17b).

8.2.2 Measurement arrival

Only when a GNSS measurement arrives, the full update step is run. Recalling that the GNSS measurement model is given by (6.34) and the true state by (7.1a),

the measurement matrix $\mathbf{H}_k$ can be calculated from (5.9). The matrix $\mathbf{H}_{\boldsymbol{x}}$ is straight forward to obtain, as

$$\begin{aligned}\mathbf{H}_{\boldsymbol{x}} &= \frac{\partial \boldsymbol{h}(\boldsymbol{x}_k)}{\partial \boldsymbol{x}_k} \\ &= \frac{\partial \boldsymbol{p}^e_{eb}}{\partial \boldsymbol{x}_k} \\ &= \begin{bmatrix}\mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O}\end{bmatrix}.\end{aligned}$$

Obtaining $\mathbf{X}_{\delta x}$ is done by

$$\begin{aligned}\mathbf{X}_{\delta x} &= \frac{\partial \boldsymbol{x}_k}{\partial \delta \boldsymbol{x}_k} \\ &= \begin{bmatrix}\mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{\Theta}_{\delta\theta} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I}\end{bmatrix}\end{aligned}$$

where $\mathbf{\Theta}_{\delta\theta}$ is found to be

$$\begin{aligned}\mathbf{\Theta}_{\delta\theta} &= \frac{\partial(\hat{\boldsymbol{q}}^e_b \otimes \delta\boldsymbol{q})}{\partial \delta\boldsymbol{\theta}} \\ &= \frac{\partial(\hat{\boldsymbol{q}}^e_b \otimes \delta\boldsymbol{q})}{\partial \delta\boldsymbol{q}} \cdot \frac{\partial \delta\boldsymbol{q}}{\partial \delta\boldsymbol{\theta}} \\ &= \frac{\partial L(\hat{\boldsymbol{q}}^e_b)\delta\boldsymbol{q}}{\partial \delta\boldsymbol{q}} \cdot \frac{\partial [1 \quad \frac{1}{2}\delta\boldsymbol{\theta}^\mathsf{T}]^\mathsf{T}}{\partial \delta\boldsymbol{\theta}} \\ &= \frac{1}{2}\begin{bmatrix}\eta & -\epsilon_1 & -\epsilon_2 & -\epsilon_3 \\ \epsilon_1 & \eta & -\epsilon_3 & \epsilon_2 \\ \epsilon_2 & \epsilon_3 & \eta & -\epsilon_1 \\ \epsilon_3 & -\epsilon_2 & \epsilon_1 & \eta\end{bmatrix} \cdot \begin{bmatrix}0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1\end{bmatrix} \\ &= \frac{1}{2}\begin{bmatrix}-\epsilon_1 & -\epsilon_2 & -\epsilon_3 \\ \eta & -\epsilon_3 & \epsilon_2 \\ \epsilon_3 & \eta & -\epsilon_1 \\ -\epsilon_2 & \epsilon_1 & \eta\end{bmatrix}.\end{aligned}$$

Following this, the Kalman gain $\mathbf{W}_k$ is calculated with (3.15) to get the updated error state estimate and covariance by (5.17).

The injection and reset step

The injection of nominal state estimate with the updated error estimate is executed according to (5.18). Here, with the presence of attitude as a state, an additional step is done that is not in the general ESKF algorithm. The states $\delta\boldsymbol{b}_a$, $\delta\boldsymbol{b}_g$ and $\delta\boldsymbol{\theta}$ are before injection expressed locally in the uninjected, nominal frame, and so a transformation of their covariances should be done to express them locally in the injected, nominal frame. By taking the Jacobian $\mathbf{J}_k$ of the error state *after* injection $\delta\boldsymbol{x}_k^+$ with respect to the error state *before* injection $\delta\boldsymbol{x}_k$,

$$\mathbf{J}_k = \frac{\partial \delta\boldsymbol{x}_k^+}{\partial \delta\boldsymbol{x}_k},$$

the covariance is reset by

$$\delta\hat{\mathbf{P}}_k \leftarrow \mathbf{J}_k \delta\hat{\mathbf{P}}_k \mathbf{J}_k^\mathsf{T}. \tag{8.1}$$

It is easy to see that $\mathbf{J}_k$ will be a block-diagonal matrix where each submatrix along the diagonal is the Jacobian of the state after injection with respect to itself before injection,

$$\mathbf{J}_k = \begin{bmatrix} \frac{\partial \delta\boldsymbol{p}^+}{\partial \delta\boldsymbol{p}} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \frac{\partial \delta\boldsymbol{v}^+}{\partial \delta\boldsymbol{v}} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \frac{\partial \delta\boldsymbol{\theta}^+}{\partial \delta\boldsymbol{\theta}} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \frac{\partial \delta\boldsymbol{b}_a^+}{\partial \delta\boldsymbol{b}_a} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \frac{\partial \delta\boldsymbol{b}_g^+}{\partial \delta\boldsymbol{b}_g} \end{bmatrix}.$$

The Jacobian can be computed by first noticing that the first two submatrices

$$\frac{\partial \delta\boldsymbol{p}^+}{\partial \delta\boldsymbol{p}} = \frac{\partial \delta\boldsymbol{v}^+}{\partial \delta\boldsymbol{v}} = \mathbf{I}$$

as they are expressed in inertial frame, and the covariance is therefore unaltered by the injection. The last three Jacobians are found as follows. Notice that the injection step leaves the true state untouched, $\boldsymbol{x}_k^+ = \boldsymbol{x}_k$, so

$$\hat{\boldsymbol{x}}_k^+ \oplus \delta\boldsymbol{x}_k^+ = \hat{\boldsymbol{x}}_k \oplus \delta\boldsymbol{x}_k \tag{8.2}$$

must hold. In order to find $\frac{\partial \delta\boldsymbol{\theta}^+}{\partial \delta\boldsymbol{\theta}}$, recall that the inverse of a unit quaternion is its conjugate defined in (B.4) and that all quaternions involved in ESKF are unit

quaternions as they encode a rotation. The quaternion part of (8.2) is

$$\hat{\boldsymbol{q}}_b^{e+} \otimes \delta\boldsymbol{q}^+ = \hat{\boldsymbol{q}}_b^e \otimes \delta\boldsymbol{q}. \tag{8.3}$$

and the quaternion part of (5.18) is

$$\hat{\boldsymbol{q}}_b^{e+} = \hat{\boldsymbol{q}}_b^e \otimes \delta\hat{\boldsymbol{q}} \tag{8.4}$$

where it is emphasized that $\delta\hat{\boldsymbol{q}}$ is a member of $\delta\hat{\boldsymbol{\mu}}_k$ and is therefore a numeric estimate. By premultiplying (8.3) by $(\hat{\boldsymbol{q}}_b^{e+})^*$ and inserting (8.4), the error quaternion *after* injection $\delta\boldsymbol{q}^+$ can be expressed in terms of the error quaternion *before* injection $\delta\boldsymbol{q}$ as

$$\begin{aligned}
\delta\boldsymbol{q}^+ &= (\hat{\boldsymbol{q}}_b^{e+})^* \otimes \hat{\boldsymbol{q}}_k \otimes \delta\boldsymbol{q} \\
&= (\hat{\boldsymbol{q}}_b^e \otimes \delta\hat{\boldsymbol{q}})^* \otimes \hat{\boldsymbol{q}}_k \otimes \delta\boldsymbol{q} \\
&= \delta\hat{\boldsymbol{q}}^* \otimes \hat{\boldsymbol{q}}_b^{e*} \otimes \hat{\boldsymbol{q}}_k \otimes \delta\boldsymbol{q} \\
&= \delta\hat{\boldsymbol{q}}^* \otimes \delta\boldsymbol{q} \\
&= L(\delta\hat{\boldsymbol{q}}^*)\delta\boldsymbol{q} \\
\begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta}^+ \end{bmatrix} &= \begin{bmatrix} 1 & \frac{1}{2}\delta\hat{\boldsymbol{\theta}}^{\mathsf{T}} \\ -\frac{1}{2}\delta\hat{\boldsymbol{\theta}} & \mathbf{I} - S(\frac{1}{2}\delta\hat{\boldsymbol{\theta}}) \end{bmatrix} \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix}
\end{aligned}$$

where, by inspecting the vector part of the last expression, reveals that

$$\begin{aligned}
\frac{1}{2}\delta\boldsymbol{\theta}^+ &= -\frac{1}{2}\delta\hat{\boldsymbol{\theta}} + \frac{1}{2}\left(\mathbf{I} - S\left(\frac{1}{2}\delta\hat{\boldsymbol{\theta}}\right)\right)\delta\boldsymbol{\theta} \\
\delta\boldsymbol{\theta}^+ &= -\delta\hat{\boldsymbol{\theta}} + \left(\mathbf{I} - S\left(\frac{1}{2}\delta\hat{\boldsymbol{\theta}}\right)\right)\delta\boldsymbol{\theta} \\
\implies \frac{\partial\delta\boldsymbol{\theta}^+}{\partial\delta\boldsymbol{\theta}} &= \mathbf{I} - S\left(\frac{1}{2}\delta\hat{\boldsymbol{\theta}}\right).
\end{aligned} \tag{8.5}$$

Perhaps unexpectedly, following the same procedure for $\delta\boldsymbol{b}_a^+$ and $\delta\boldsymbol{b}_g^+$ reveals that

$$\frac{\partial\delta\boldsymbol{b}_a^+}{\partial\delta\boldsymbol{b}_a} = \frac{\partial\delta\boldsymbol{b}_g^+}{\partial\delta\boldsymbol{b}_g} = \mathbf{I}.$$

Thus, the Jacobian $\mathbf{J}_k$ in (8.1) is given by

$$\mathbf{J}_k = \begin{bmatrix} \mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{I} - S\left(\frac{1}{2}\delta\hat{\boldsymbol{\theta}}\right) & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{I} \end{bmatrix}. \tag{8.6}$$

IV

Filter validation and results

9 | Filter validation and tuning

Having derived the necessary equations to deploy the ESKF in an actual INS, the filter must be validated by some criteria in order to verify its performance. For artificial data, where the true states are known, the estimates can easily be verified by comparing to the true states. In most situations, however, the true states are unknown, demanding other means for verification. This chapter intends to introduce some possible ways of validating the filter performance, both for artificial and experimental data, and how the filter can be tuned to give best performance.

9.1 Visual inspection

The most basic form for validating the filter is by visually inspecting the output of the filter. For artificial data this can be done by plotting the estimates against the true states, which gives some qualitative measure of how well the filter performs. This is, however, more troublesome in an experimental setup, as the true states are unavailable. The estimates must here be compared to what can be reasonably expected from the experiment, and for an INS the positional estimates may be most easily comparable to what is expected. In an INS setup with GNSS, the positional measurements from the GNSS may also indicate where the positional estimates should be.

Although these qualitative methods of validating the filter give some rough indication of its performance, more quantitative methods are desirable for further validation and tuning.

9.2 Filter consistency

Filter consistency is a systematic method for computing statistics that give information of the *consistency* of a filter. Following [2] and [31], a filter is deemed consistent when its errors correspond to the modeling assumptions of the filter, that is, the errors are on average zero with covariance as yielded by the filter. In this context, the errors are both the state errors and the innovations. More specifically, both the state errors and innovations should be acceptable as zero-mean and have magnitude commensurate with their covariance, in addition to the innovations being white. It goes without saying that the criteria on state errors can only be tested when the true states are available, so this is limited to the artificial case, but the innovations may also be tested on experimental data.

Consistency is typically first tested by a χ^2-test on the *normalized estimation error squared* (NEES)

$$\varepsilon = (\boldsymbol{x} - \boldsymbol{\mu})^\mathsf{T} \mathbf{P}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \tag{9.1}$$

and *normalized innovation squared* (NIS)

$$\varepsilon^\nu = \boldsymbol{\nu}^\mathsf{T} \mathbf{S}^{-1} \boldsymbol{\nu}, \tag{9.2}$$

both of which are χ^2-distributed under the assumption of a linear and Gaussian model. If $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{z} \in \mathbb{R}^m$, the NEES and NIS are distributed as

$$\begin{aligned} \varepsilon &\sim \chi^2_n, \\ \varepsilon^\nu &\sim \chi^2_m, \end{aligned}$$

respectively, where n and m are the degrees of freedom, or equivalently the expectations. A χ^2-test for the state error can then be formulated as

$$H_0 : \mathrm{E}[\varepsilon] = n \quad \text{against} \quad H_1 : \mathrm{E}[\varepsilon] \neq n, \tag{9.3}$$

while the test for the innovation is

$$H_0 : \mathrm{E}[\varepsilon^\nu] = m \quad \text{against} \quad H_1 : \mathrm{E}[\varepsilon^\nu] \neq m, \tag{9.4}$$

where the NEES and NIS are the respective test statistics. By choosing some level of significance, α, the filter may be tested for consistency. Note that the

filter is consistent under the null-hypothesis, H_0. Equivalently, and perhaps more illustrative, one may construct a two-sided confidence interval for the NEES and NIS to check for filter consistency.

The overall consistency is typically tested by constructing a confidence interval for the *average* NEES and NIS over all time steps, denoted ANEES and ANIS, where

$$\bar{\varepsilon} = \frac{1}{N}\sum_{k=1}^{N}\varepsilon_k \tag{9.5}$$

and

$$\bar{\varepsilon}^v = \frac{1}{N}\sum_{k=1}^{N}\varepsilon_k^v \tag{9.6}$$

are the ANEES and ANIS, respectively, over the total number of time steps, N. The distributions are then found as

$$N\bar{\varepsilon} \sim \chi^2_{Nn},$$
$$N\bar{\varepsilon}^v \sim \chi^2_{Nm},$$

where the respective confidence intervals are given by

$$P\left(\frac{1}{N}\chi^2_{Nn,1-\frac{\alpha}{2}} \leq \bar{\varepsilon} \leq \frac{1}{N}\chi^2_{Nn,\frac{\alpha}{2}}\right) = 1-\alpha, \tag{9.7}$$

$$P\left(\frac{1}{N}\chi^2_{Nm,1-\frac{\alpha}{2}} \leq \bar{\varepsilon}^v \leq \frac{1}{N}\chi^2_{Nm,\frac{\alpha}{2}}\right) = 1-\alpha, \tag{9.8}$$

for some chosen confidence level α. The filter is said to be consistent in terms of NEES and NIS if the ANEES and ANIS lie within their respective confidence bounds. Similarly, the filter is also consistent if a sufficient percentage of the NEES and NIS are within their respective bounds.

If the filter is *inconsistent* in terms of NEES and NIS, the filter typically tested for bias and whiteness to identify the problem. The tests for bias and whiteness follow a similar procedure as for NEES and NIS, but as they are not used in the remainder of this thesis they will not be further discussed here. A treatment of bias and whiteness tests can be found in [31].

9.3 Root mean square error

The *root mean square error* (RMSE) provides an absolute measure of the state error in the filter, and is given by

$$\boldsymbol{x}_{\text{RMSE}} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)^{\mathsf{T}}(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)}. \tag{9.9}$$

A low RMSE is perhaps one of the most desirable properties of a filter, as it gives a direct indication of its performance, but is unfortunately only measurable when the true states are available. The use of RMSE for validation is therefore limited to artificial data.

9.4 Tuning

Tuning denotes the process of adjusting variables to give best possible performance according to some performance criteria. In this context, the filter is said to be tuned when its performance is satisfactory, for example according to the methods of validation previously discussed. These methods can only be used *after* the experiment is conducted, so tuning usually happens as a post-process. For the ESKF, the only possible variables to adjust are the process and measurement model covariances, the bias covariance of the gyroscope and accelerometer, and the initial state. Although the covariances have previously been given as time-variant variables, they will for simplicity be assumed time-invariant in the remainder of the thesis.

Denoting the initial state as a tuning variable may be somewhat incorrect, as one usually has control over the initial conditions in the experiment, and will therefore not need to tune this variable. If the filter initially converges fast towards the true state, this variable will also not affect the remainder of the experiment. However, if one were to initialize the filter at a completely wrong initial state, the filter may never converge at all [32], especially if there are large modeling errors. Setting a good initial state may therefore be crucial for the performance of the filter.

The process and measurement model covariances, $\mathbf{Q}$ and $\mathbf{R}$, respectively, are usually the main tuning variables for a Kalman filter. For the ESKF, which here internally discretizes the continuous-time covariance $\boldsymbol{\Sigma}_n$ into $\mathbf{Q}$, the continuous-time covariance is tuned instead. These variables capture the uncertainty in the

models, and are given values that are consistent with both the model uncertainty and the dataset being tuned for. Although there are methods that give rough estimates of Σ_n and $\mathbf{R}$, the process of tuning is often done by trial and error on a given dataset. For the ESKF, where both the process and measurement model use sensory input, the values of Σ_n and $\mathbf{R}$ need only be consistent with the noise of the given sensors, greatly simplifying the tuning process. The same goes for the bias covariances, as these are properties of the specific IMU used in the experiment.

Although using the characteristics of the sensors for choosing the respective covariances may seem sufficient, the filter may still not perform satisfactory in terms of consistency. This is because the modeling assumptions, namely that the process and measurement models are Gaussian, are nothing more than *models* of the true processes. An analysis in terms of NIS, and NEES and RMSE if the true states are available, is then usually done to ensure that the filter is consistent. One will typically observe that too large values of $\mathbf{Q}$ and $\mathbf{R}$ drive the ANEES and ANIS below the lower bounds of the confidence intervals, while too low values drive them above the upper bounds. The two cases are denoted as the filter being *underconfident* and *overconfident*, respectively. Underconfident typically means that the associated covariance is too large, and that the filter makes better predictions than it believes itself. A consequence of this is for example slower convergence to the true state of the system. Overconfident is the counterpart of underconfident in the sense that the filter now makes worse predictions than it is aware of itself. An underconfident filter can work in practice, although not optimally, while an overconfident filter is susceptible to divergence, and is a much more critical problem to correct.

If the true states are available, it may seem obvious that the filter should be tuned to give the lowest possible RMSE. This may, however, have negative consequences in terms of consistency, where the performance of the filter may be satisfactory on the dataset being tuned on, but completely off on another. A middle ground between filter consistency and RMSE is thus often desired, so that the filter is generalized while still performing satisfactory.

10 | Results

As this thesis was written as a cooperation with the student organization Ascend NTNU, an implementation in C++, shown in Appendix C, was written that interfaces with the *Robot Operating System* (ROS). ROS is a middleware that significantly simplifies communication between different *nodes*, which are just processes on a computer executing a single task. In this context, the ESKF node is a part of a pipeline of nodes to provide the autonomous drone with state estimation. The performance of this implementation has been benchmarked using two datasets, and the results together with discussion is presented in the following chapter. Additionally, the tuning process is explained, with how initial model parameters are chosen and how they are tuned in accordance to the given results.

10.1 The datasets used

As mentioned, two different datasets are used to inspect the performance of the ESKF. Both datasets were used as part of the second graded assignment in the course TTK4250 Sensor Fusion, fall 2019, at NTNU. The first dataset is a simulated dataset generated in Matlab, but the underlying parameters and models used in the simulator are not known. The second dataset is real data from a flight with a UAV conducted by the UAV lab at the ITK Department of Engineering Cybernetics at NTNU. See Figure 10.1 for flight path.

For the real dataset, the IMU used was a Sensonor STIM300 [34], while the GNSS modules were two uBlox NEO-M8T receiver modules [35], with the front receiver placed close to the IMU and the rear receiver placed approximately 0.7 m behind. The dataset, however, only provides one GNSS measurement that is assumed on top of the IMU, and as such, no lever arm is used to compensate for displacement. The measurements are timestamped with the *SenTiBoard*, which

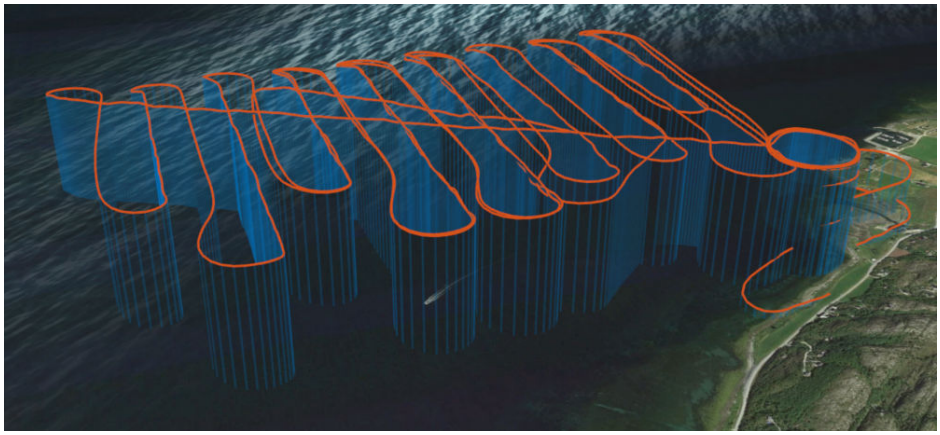


Figure 10.1: *Flight path of the real dataset. Picture taken from [33].*

is a sensor timing board developed at the UAV lab at the ITK Department of Engineering Cybernetics at NTNU [36]. This is used to synchronize the arrival of all measurements beforehand and is therefore not done internally in the ESKF node.

10.1.1 Benchmarking with both simulated and real data

In general, benchmarking the performance of ESKF with both simulated and real data has several benefits. Using simulated data will give access to what is called *ground truth*, which is the true state, at any desired point in time. This allows for directly measuring the error in the estimate and to use metrics such as NEES from Chapter 9.2 and RMSE from Chapter 9.3. Additionally, simulated data is made from a simulator written in software, meaning it is available at a much earlier stage than the actual hardware required for real data. This is crucial for debugging in the early stages of an iterative development process, where the functionality of the filter may be tested on perfect data, that is, data without noise, that may otherwise hide faults in the system. However, a simulator is no more realistic than the underlying, implemented model. Ultimately, the filter has to be tested with data collected with the actual hardware it will be used in conjunction with to validate that it functions as intended. This also tests the entire pipeline of data gathering, transport and processing.

10.2 The simulated dataset

Before going into the tuning process, a brief overview of the data is in order. The trajectory that the UAV follows in the simulated data can be seen in Figure 10.3. The dataset has a brief period in the start where the UAV is at rest, before being launched into flight, where it then follows many sharp maneuvers. The entire set is 15 minutes long, with IMU measurements arriving at 100 Hz and GNSS at 1 Hz. GNSS measurements are given in a local tangent frame, called *north-east-down* (NED) frame, where the x-axis is aligned to point north, the y-axis east and z-axis downwards, towards the center of Earth. The origin is placed at the start of the trajectory.

With ground truth available, the initial nominal state $\hat{\boldsymbol{\mu}}_0$ was chosen equal to ground truth at the first time step and error covariance $\delta\mathbf{P}_0$ small. Thus, the main tuning challenge was the tuning of the continuous-time process noise covariance matrix $\boldsymbol{\Sigma}_n$ and discrete-time measurement noise covariance matrix $\mathbf{R}$.

One large tuning benefit for filters used in INS, such as the ESKF, is the fact that both process noise and measurement noise originate from sensors. In the more general case, $\mathbf{Q}$ has to be deduced from the process dynamics, where characteristics such as process transients are used for making an initial guess, and must therefore also be known. In the ESKF, however, both $\boldsymbol{\Sigma}_n$ and $\mathbf{R}$ are obtainable from the datasheets of the respective sensors. As the parameters used in the simulated data were not known, this was obviously not possible.

For tuning the white noise in $\boldsymbol{\Sigma}_n$, it was exploited that the IMU is at rest for the first period of the simulation. This period is, however, less than a minute, which is not long enough to give good estimates of the sensor bias as it accumulates so slowly. Bias variance was therefore tuned by trial and error using NEES and estimation errors. Sensor bias tends to be magnitudes lower than white noise, where around a tenth of the white noise standard deviation gave good results.

As ground truth was available, $\mathbf{R}$ was estimated by subtracting the ground truth from the GNSS measurements and calculating the empirical standard deviation of the resulting signal. This gave reasonable results, with equal variance in the plane and higher variance in altitude, as expected for a GNSS. The final parameters are shown in Table 10.1, with the ESKF performance shown in Appendix A.1.

NIS and NEES were used for tuning by validating that the filter is not under- or overconfident and with as many estimates inside the confidence interval as possible, choosing $\alpha = 0.05$ as confidence level. The metrics, however, ended up

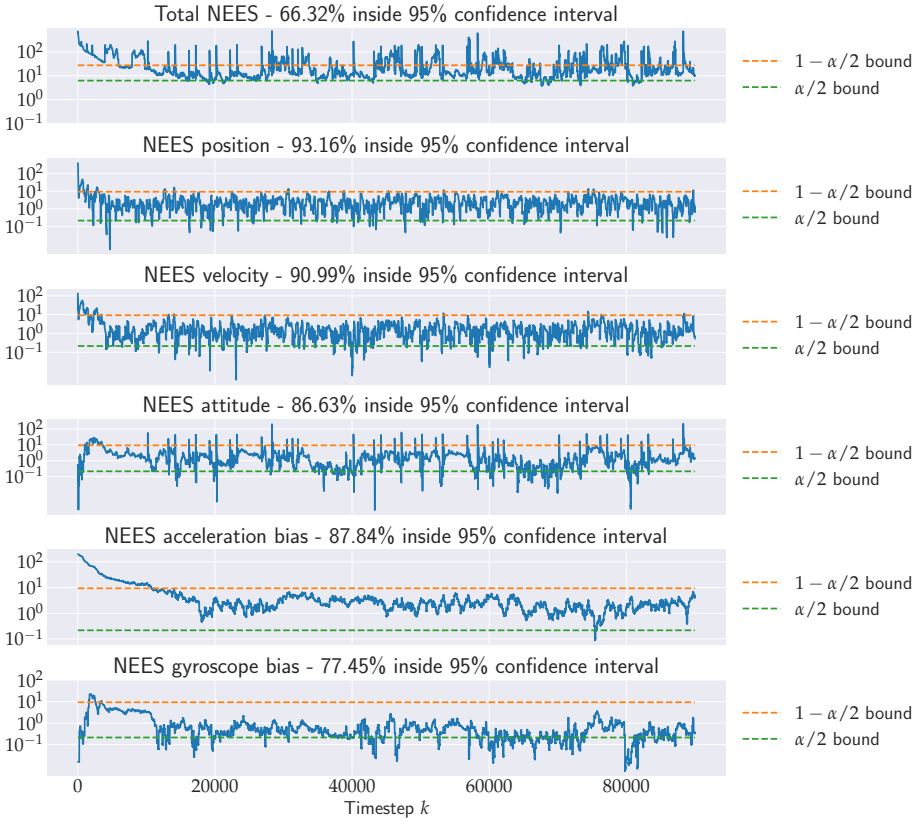


Figure 10.2: *NEES from simulated data. Note that the y-axis is logarithmic.*

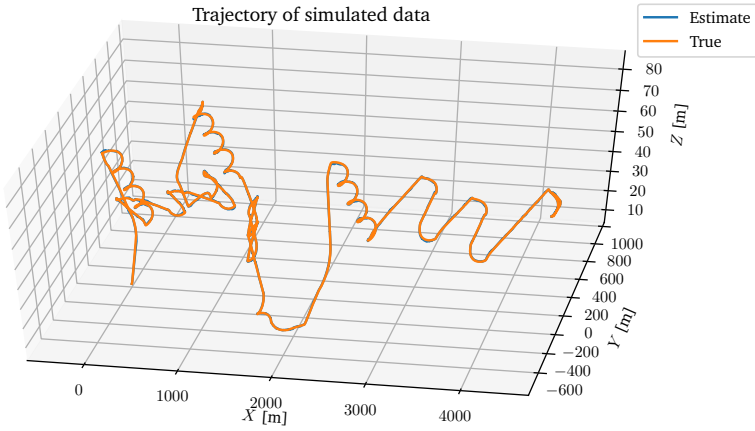


Figure 10.3: *Trajectory of simulated data. Note the different scales of the axes.*

being used for only a few of the states when arriving at the values in Table 10.1. This is explained by the discussion above, were parameter values were mostly estimated based on a purely statistical approach instead of trial and error. This is not guaranteed to give the best results, and the results in Appendix A.1 are most likely not depicting the optimal solution, but further tuning by trial and error did not show significant increase in performance.

The true errors of the state estimates are all low, as seen from Figure 10.4. Although the position and velocity error are slightly larger in the beginning, with v_x spiking a couple of times, these errors are otherwise small and show a white noise looking behavior. This is expected from their strong observability from the GNSS position measurements. Roll and pitch errors and horizontal accelerometer biases are only observed as linear combinations from the GNSS measurements under constant acceleration and attitude. To separate the estimates, turning and forward acceleration of the UAV is required. The error states are then observable by the ESKF by tracking the changes to their impact on the velocity error through the off-diagonal elements in $\delta\mathbf{P}$ [37]. From Figure 10.4, the roll φ and pitch θ error converges reasonably fast to zero, with the horizontal accelerometer biases $b_{a,x}$ and $b_{a,y}$ lagging more behind. This could be a consequence of suboptimal tuning of bias variance, where a larger variance is more fit. This is consistent with NEES for accelerometer bias in Figure 10.2, as the filter shows overconfident behavior in the start before converging to consistent estimates. Heading, both for yaw ψ and gyroscope bias $b_{g,z}$, requires more significant maneuvers to be properly observed, resulting in a usually slower convergence [37], and the heading error in Figure 10.4 never really settles on zero in the same way as roll and pitch. This is subsequently a weakness of this filter with only position as GNSS measurement with no lever arm. For example using a compass or a second GNSS antenna for measuring heading would therefore provide better estimates of heading for the whole trajectory.

10.3 The real dataset

As for the simulated dataset, an overview of the real data is in order before delving into the tuning process. The first part of the data the UAV is stationary on ground level. It is then launched into flight, where it first builds height following a helix trajectory, before following a zig-zag trajectory, with an intermediate flight back to approximately the initial x- and y-coordinate, but keeping its altitude.

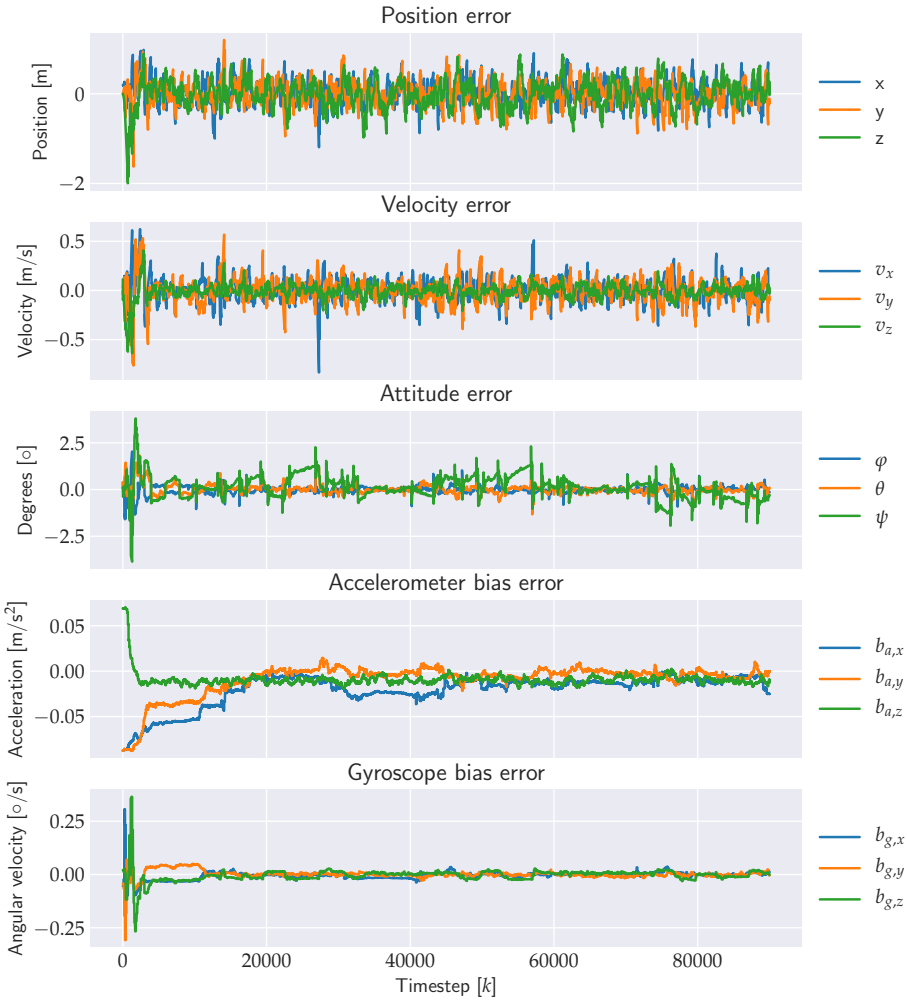


Figure 10.4: *Plots over true error state from simulated data.*

Parameter	Value
σ_a	0.02 m/s$\sqrt{s}$
σ_{b_a}	0.002 m/s$\sqrt{s}$
σ_g	0.0008 rad/$\sqrt{s}$
σ_{b_g}	0.00008 rad/$\sqrt{s}$
ρ_x	0.3 m
ρ_y	0.3 m
ρ_z	0.5 m

Table 10.1: *Table over final parameters for the ESKF on simulated data. σ indicates the scalar value used in the diagonal of the respective $\mathbf{\Sigma}$ covariance matrix. Recall that the white noise of the IMU is assumed isotropic, such that the same variance is used for all axes. ρ is used for the diagonal element of $\mathbf{R}$.*

The entire dataset is approximately 1 hour long, where IMU measurements arrive at 250 Hz and the GNSS at 1 Hz. Although the original GNSS measurements came in ECEF frame, it has here been transformed into NED frame beforehand. Lastly, the uBlox receiver is able to estimate the accuracy of its own position estimate, and these estimates were provided with the dataset.

An initial test of the filter performance with the parameters from Chapter 10.2 was done. However, this resulted in suboptimal results, and a new investigation of parameters was done. With the sensors used now given, an initial guess on noise parameters was derived from data in the respective datasheets. The velocity accuracy for the used GNSS sensor is given as 0.05 m/s [35], so an initial guess for position noise was set to

$$\rho = \frac{0.05 \text{ m/s}}{1 \text{ Hz}} = 0.05 \text{ m} \tag{10.1}$$

where 1 Hz is the sampling frequency and ρ is the position standard deviation in x-, y- and z-direction.

For the IMU, the white noise parameters were found to be 0.06 m/s/$\sqrt{\text{hr}}$ for acceleration and 0.15 °/$\sqrt{\text{hr}}$ for angular velocity [34]. The standard deviation for the acceleration white noise σ_a was calculated to be

$$\begin{aligned}
\sigma_a &= 0.06 \text{ m/s}/\sqrt{\text{hr}} \\
&= 0.06 \text{ m/s}/\sqrt{\text{hr}} \cdot \sqrt{\frac{1 \text{ hr}}{3600 \text{ s}}} \\
&= 1.0 \cdot 10^{-3} \text{ m/s}\sqrt{\text{s}}
\end{aligned} \tag{10.2}$$

and the standard deviation for the gyroscope white noise σ_g to be

$$\begin{aligned}\sigma_g &= 0.15\ ^\circ/\sqrt{\text{hr}} \\ &= 0.15\ ^\circ/\sqrt{\text{hr}} \cdot \frac{\pi \text{ rad}}{180^\circ} \cdot \sqrt{\frac{1 \text{ hr}}{3600 \text{ s}}} \\ &\approx 4.36 \cdot 10^{-5} \text{ rad}/\sqrt{\text{s}}\end{aligned} \tag{10.3}$$

where the units of $\text{m/s}\sqrt{\text{s}}$ and $\text{rad}/\sqrt{\text{s}}$ are a result of computing standard deviations in *continuous-time*. After squaring σ_a and σ_g, inserting them into Σ_n in their appropriate places and discretizing the covariance matrix, the units will rest assured be correct.

No numeric value for the bias, or random walk, variance is given in the datasheet for the STIM300. The datasheet does, however, provide plots of the *Allan variance* for the gyroscope and accelerometer. Allan variance is a systematic method for estimating the white noise, bias instability and random walk variance of IMUs, and was originally invented by David W. Allan for estimating frequency stability in oscillators during the 1960s [38]. The bias variance can be read from such a plot by the intersection between the tangent of the log curve where the slope is 1/2 with where $\tau = 3$ s [39], [40]. Random walk can be observed from such a plot as an inflection of the graph, where, after some time of decreasing, the variance starts to increase. The time of inflection is dependent on the quality of the IMU. However, due to the high quality of the STIM300 and the short time scale of only 10^4 s $\approx$ 2.7 hours in the plots, extracting the bias variance can unfortunately not be done without extrapolating the curve, and was instead estimated from the white noise variance as a thousandth of its magnitude. For a real application, where the exact IMU that will be used is available, such a plot can be made from raw measurements of said IMU at perfect rest in a temperature constant environment to minimize estimation errors from external sources of noise. The time period required is again dependent on the IMU, but in general, the longer time, the better. In [39], measurements are sampled for seven days to ensure stable data with little influence from external errors on average.

With these initial parameter values, the filter performs decent. The saw-tooth looking behaviour of the x- and y-coordinate in the plot shown in Figure 10.6 is explained by the shape of the trajectory, as the UAV is zig-zagging back and forth in the x-direction before returning to the origin, twice. The y-coordinate increases linearly when the UAV turns, and remains stationary while making the stint where the x-coordinate changes. This causes the staircase looking plot.

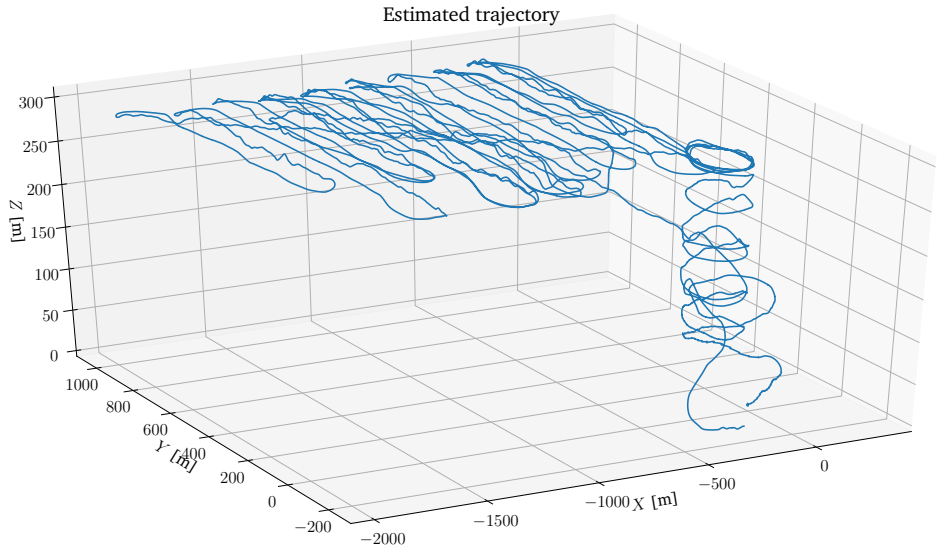


Figure 10.5: *Estimated trajectory of real dataset.*

The attitude estimates in Figure 10.7 appear stable and consistent with what is expected from the trajectory. Thus, having a sensor give control input to the filter has therefore been shown to ease the tuning process, as the process noise is well explained by the properties of the sensor and not only in the uncertainty of the maneuvers of the object being estimated, which can be difficult to assess.

As the real data has no ground truth to calculate NEES with, the only available metric for evaluating filter consistency is NIS. In Figure 10.8a, a large spike can be observed in the beginning of the dataset, before settling down, and is the main contributor to measurements outside the confidence interval. The NIS settles down when the robot starts to move. One possible explanation is that, as this is at the start of the estimation, the filter estimates incorrect bias as it has not converged to the correct estimate yet. This is probably also due to the poor observability of accelerometer bias when the UAV is not in sufficient motion, and the usual slower convergence of gyro bias with respect to time [37]. Thus, as the UAV is otherwise at rest at the start of the estimation, small signal noise and biases dominate the measurement, which the filter incorrectly mistakes for true acceleration which propagates to driving the kinematics. This is consistent with the fact the filter shows overconfident behavior. These errors are only observed and corrected by the periodic GNSS measurements. After a while, the filter man-

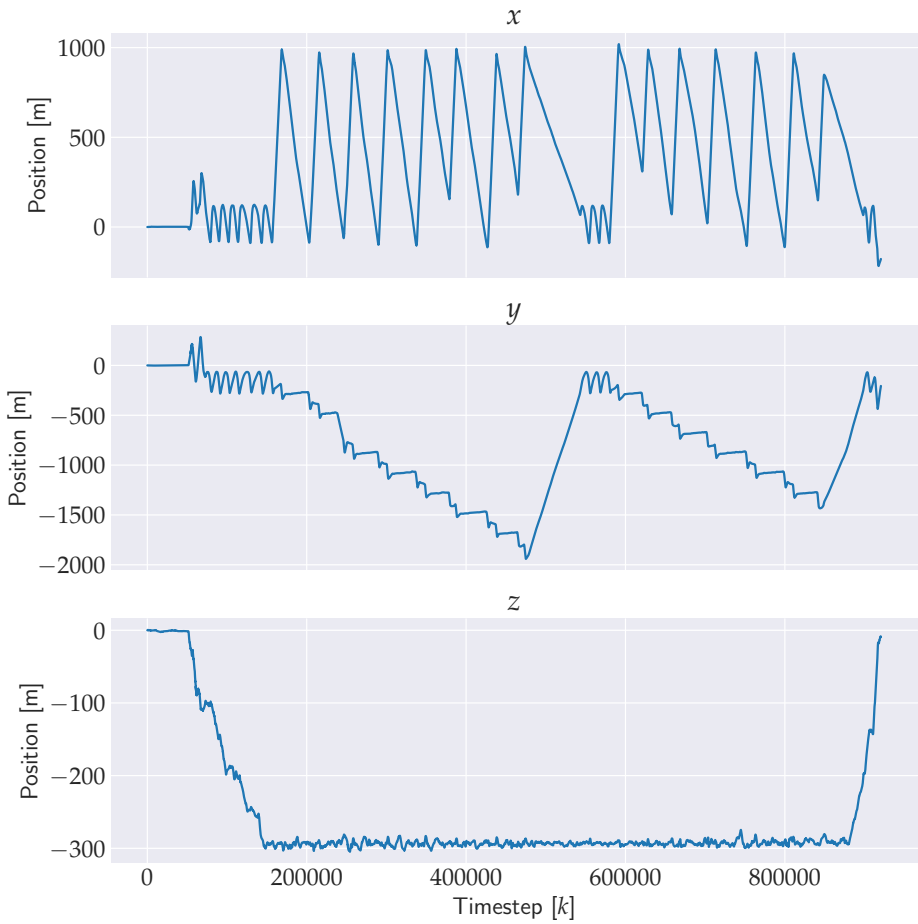


Figure 10.6: *Position estimate from real data.*

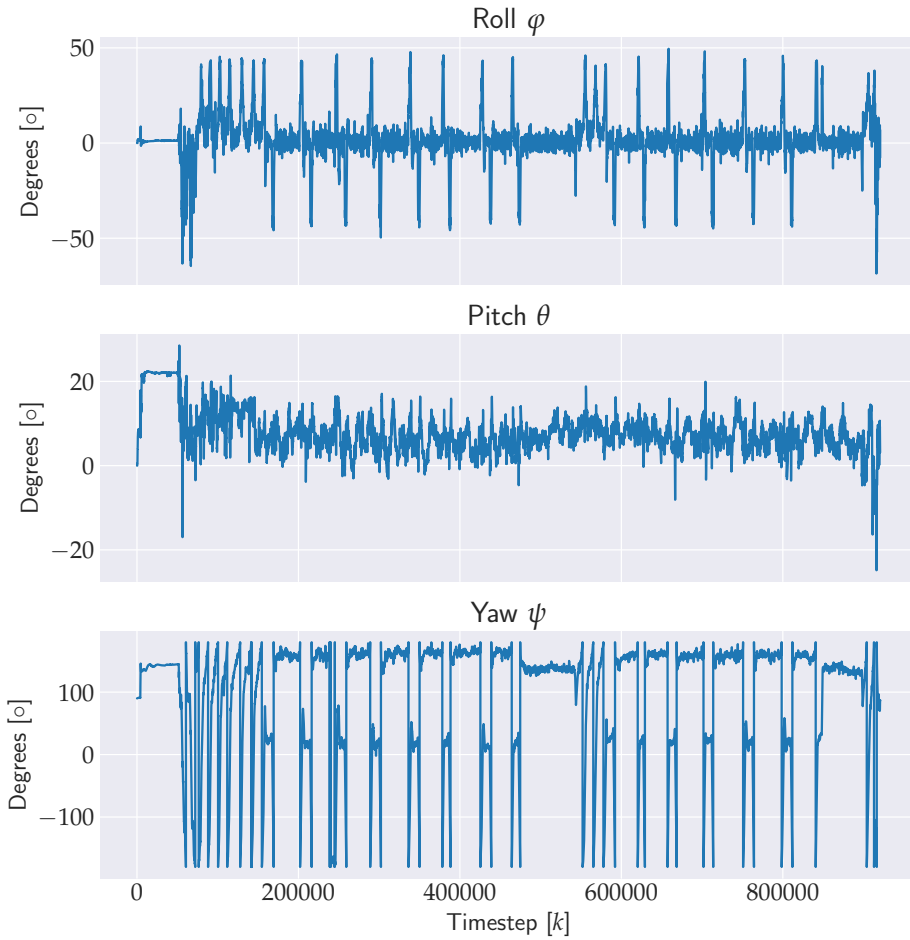


Figure 10.7: *Attitude estimate from real data.*

ages to estimate bias correctly before being launched into flight, which settles down NIS.

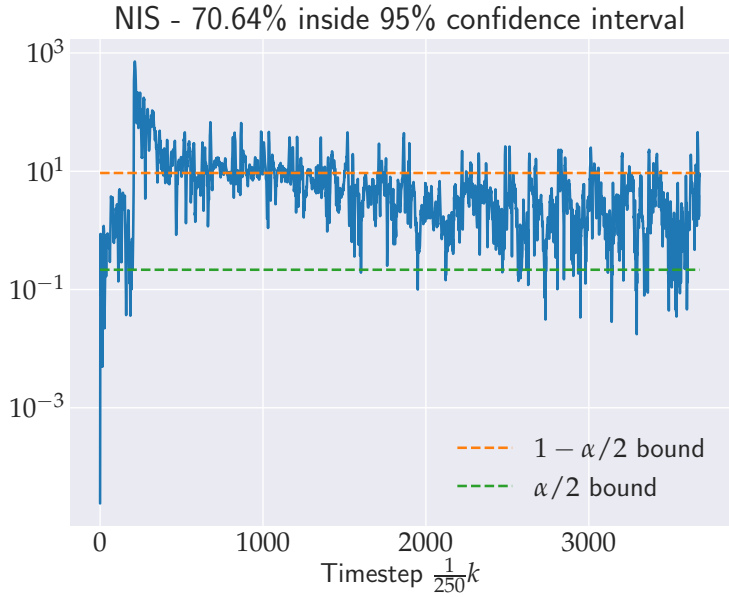
With the initial measurement standard deviation of $\rho = 0.05$ m, only ~10% of the measurements were inside the 95% NIS confidence interval, see Figure 10.8b. A large portion of the measurement estimates were on the upper side of the 95% confidence interval, indicating the filter was overconfident. Because of this, the measurement variance was inspected, and was done by analyzing the uBlox position accuracy estimates that were available. Using this data directly would be wrong considering it is the position precision, and not the accuracy, that is of interest. The standard deviation of the accuracy, however, should be a good

estimate of the precision. Calculating the standard deviation revealed that the given GNSS module had closer to 0.5 m in standard deviation, ten times higher than the value predicted from the datasheet. Applying this correction improved NIS to 70%, see Figure 10.8a.

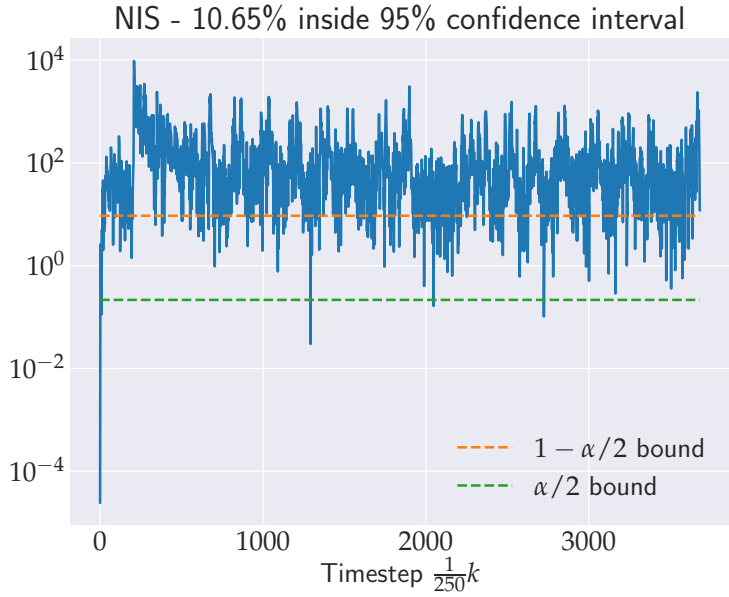
To further improve NIS, implementing the compensation matrices $\mathbf{C}_a$ and $\mathbf{C}_b$ from Chapter 6.3.1 given by (6.30) and (6.31) is an option. Although the filter still is able to estimate the state to a satisfying degree, a filter consistency analysis will usually quickly reveal that the filter in absence of such compensations matrices becomes overconfident. The most intuitive explanation for this is that the compensation matrices present the filter with additional information that the filter otherwise has no means of acquiring. As the filter naively trusts the IMU measurements, this results in it underestimating the actual error, which is the source of the overconfidence. As this was never implemented in software, these results do not exist, unfortunately.

Parameter	Value
ρ_x	0.5 m
ρ_y	0.5 m
ρ_z	0.5 m
σ_a	$1.0{\cdot}10^{-3}$ m/s$\sqrt{\text{s}}$
σ_{b_a}	$1.0{\cdot}10^{-6}$ m/s$\sqrt{\text{s}}$
σ_g	$4.36 \cdot 10^{-5}$ rad/$\sqrt{\text{s}}$
σ_{b_g}	$4.36 \cdot 10^{-8}$ rad/$\sqrt{\text{s}}$

Table 10.2: *Table over final parameters for the ESKF on real data.*



(a) *The final NIS from the real dataset. The sharp peak in the start is probably a consequence of the UAV being at rest.*



(b) *The initial NIS from the real dataset, before increasing measurement noise.*

Figure 10.8: *NIS before and after correcting* **R**. *Note the logarithmic y-axis. Also note that the NIS is is only computed every time a GNSS measurement arrives, which is every* 250^{th} *timestep*

V

Closing remarks

11 | Closing remarks

11.1 Conclusion

This thesis started by introducing theory for understanding the fundamental Bayes filter, together with underlying assumptions, that the Kalman filter is based on. The original linear Kalman filter was then given together with its involved assumptions. Then, the extension for nonlinear systems, the extended Kalman filter, is introduced, together with derivations of how the process and measurement models are linearized to fit with into the original Kalman filter framework. The error-state Kalman filter is then given in a general form, where the concept of the error state and how it can represent the same states as the true and nominal state, but with different parameterizations, is introduced.

After the introduction of the Kalman filters, relevant theory for inertial navigation systems is given, with special emphasis on attitude representations. Then the state spaces and kinematics of the true, nominal and error state are introduced. The error-state Kalman filter is then finally explained in the setting of an inertial navigation system.

The thesis then discusses how the filter can be tuned and how its performance can be measured with relevant metrics. Then an implementation of the filter is tested with two datasets, where its performance is plotted and discussion is included around how to choose initial parameters and how to tune them based on the results. The results show that the error-state Kalman filter is able to perform state estimation in a highly dynamic configuration, and achieves satisfactory performance with regards to the relevant metrics.

11.2 Further work

Many models throughout the thesis are discussed, but were never implemented. Examples are the compensating matrices in Chapter 6.3.1 and leverarm in (6.33). Especially implementing a leverarm is beneficial, as this allows for using multiple GNSS measurements, which allows for directly observing heading through measurements of the baseline between two GNSS antennas. If three or more antennas are used that are not colinear, the full attitude becomes directly observable, improving filter consistency. As mentioned in Chapter 10.3, the compensating matrices can also improve the consistency of the filter.

Additionally, the state space chosen is one of the simplest state spaces possible for an INS, and could be expanded. Bias from using incorrect gravity is a large source of error and possibly divergence of the state estimate from the true state, and as such online estimation of gravity can be beneficial to correct this. The ESKF implemented for this thesis fused GNSS and IMU data in a loosely coupled manner. The estimate can be improved by converting to tightly coupled fusion, although this demands a sophisticated model for error sources in raw GNSS measurements such as pseudorange and Doppler frequency.

The thesis has only considered the theoretical background for ESKF in an INS, and little to no discussion around practical implementation is included. An example is use of the costly matrix exponentiation computation given by Van Loan in Chapter 7.2.1, which has approximations discussed in for example [24] that can be considered for shorter computation time. Perhaps most importantly, time synchronization of sensor data is an incredibly important part of sensor fusion, and has been completely abstracted away in the datasets used. How the filter should cope with measurements that are not valid at the time of arrival should be discussed and implemented, for example by use of rollback, where the update step is performed on a previous, buffered state estimate that is closest in time to when the GNSS measurement is valid, for then to predict back to the current time step. On top of this, outlier rejection of measurements can be considered, either to detect missing measurements or measurements that are not feasible. This can for example be implemented as a hypothesis test with a suitable confidence interval that the measurement must be within.

In Chapter 10.3, the used GNSS module provides position accuracy estimates online. Thus, instead of setting a static covariance based on a datasheet value, updating the measurement covariance online while the filter is running has potential for improvement.

VI

Appendices

A | Additional plots

A.1 Results from simulated data

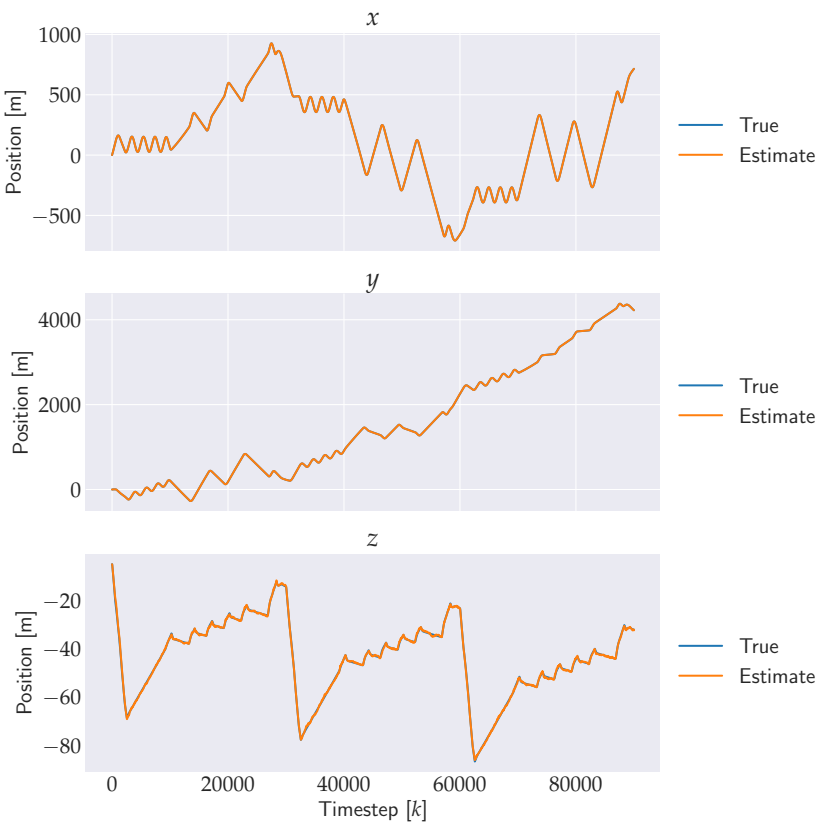


Figure A.1: *Position estimate from simulated data.*

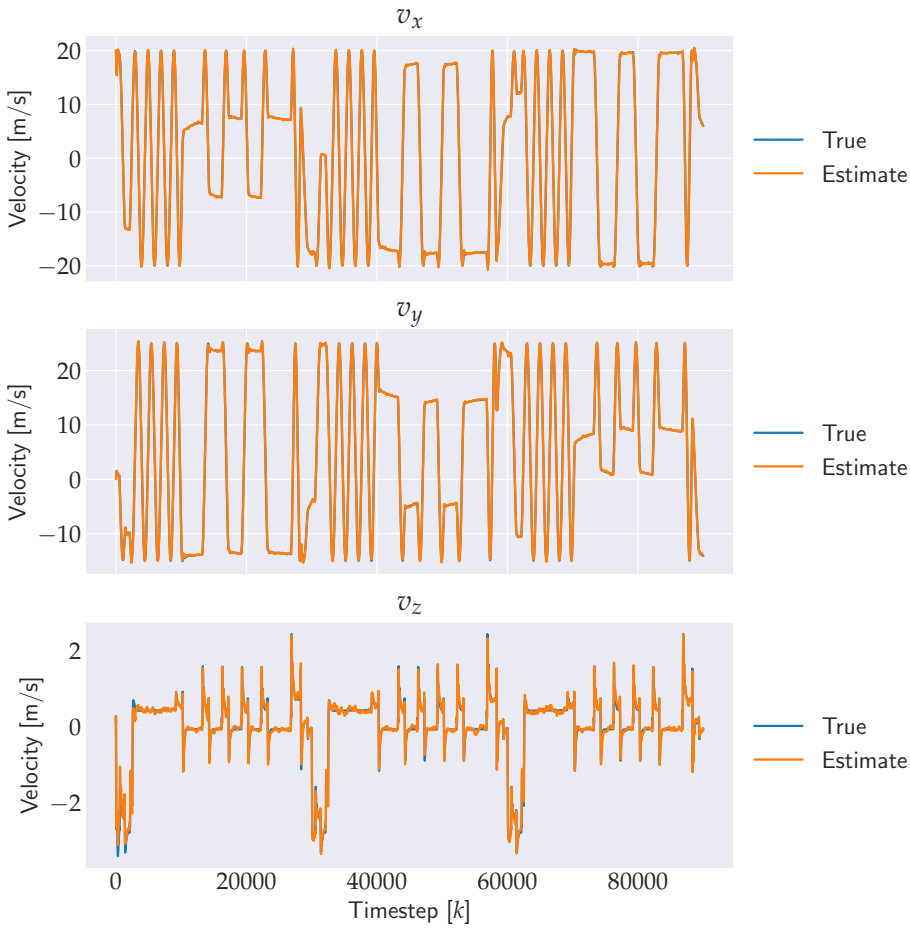


Figure A.2: *Velocity estimate from simulated data.*

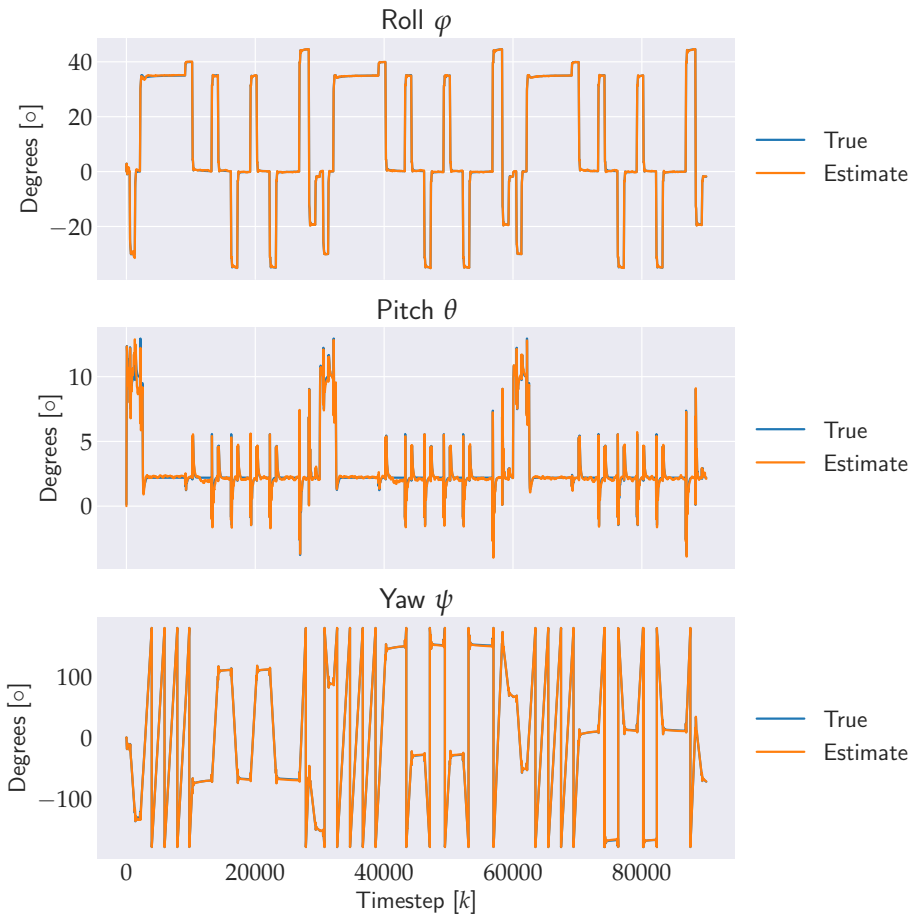


Figure A.3: *Attitude estimate from simulated data.*

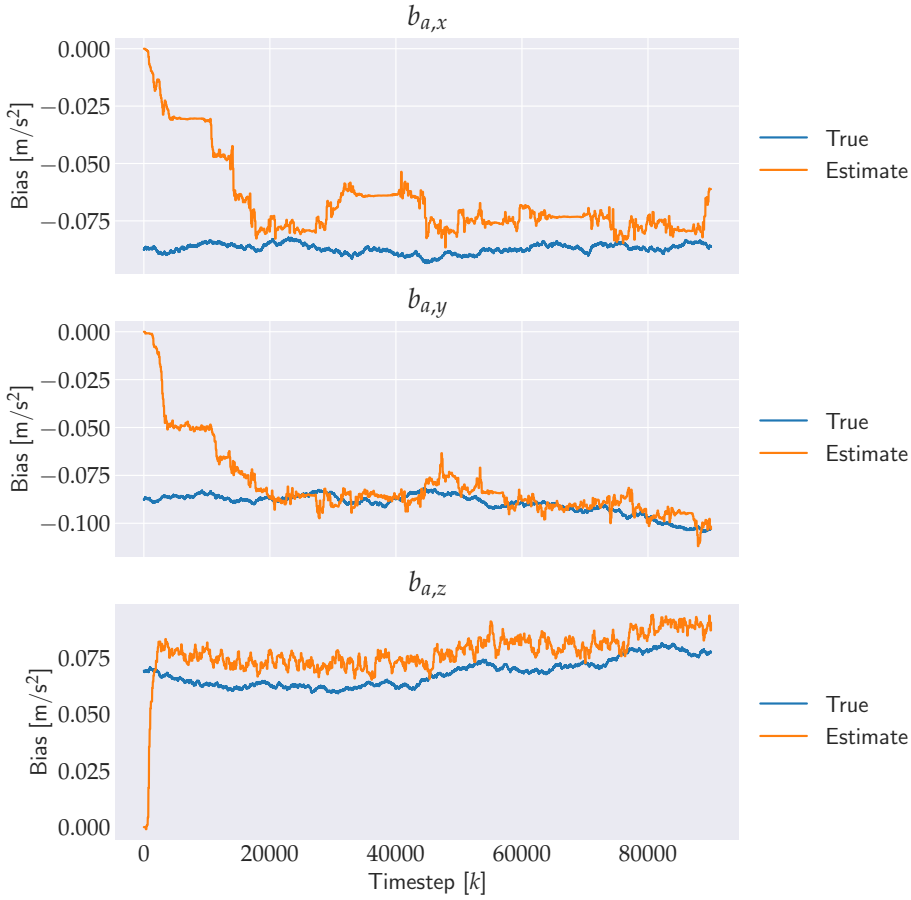


Figure A.4: *Accelerometer bias estimate from simulated data.*

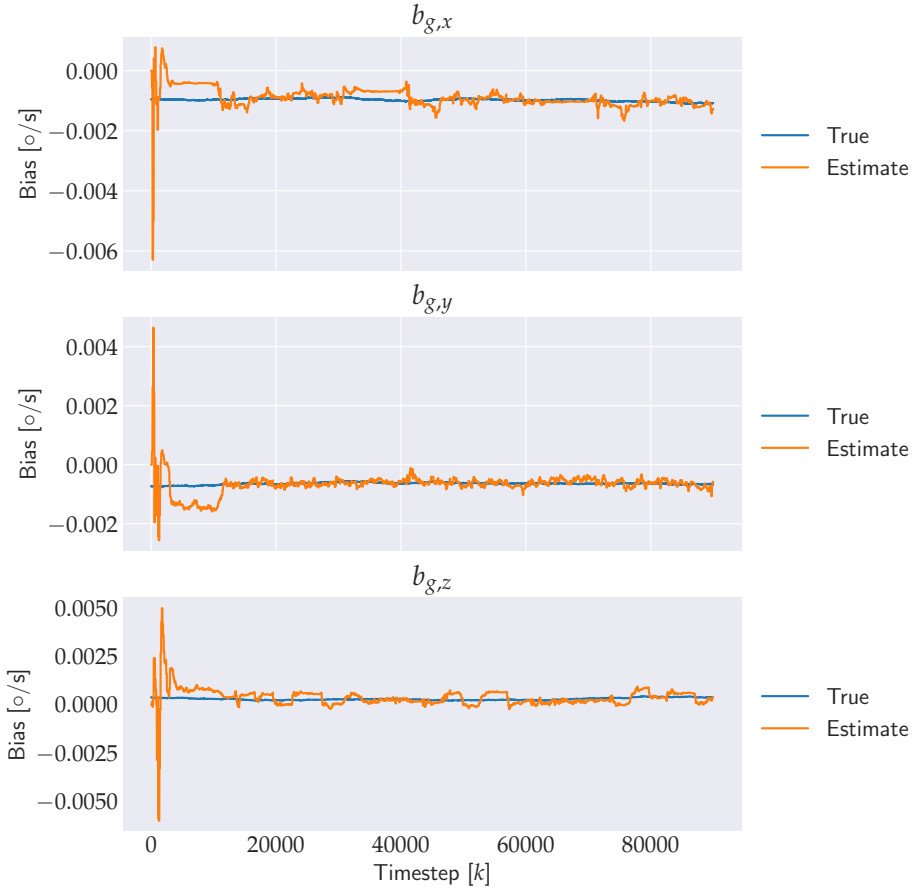


Figure A.5: *Gyroscope bias estimate from simulated data.*

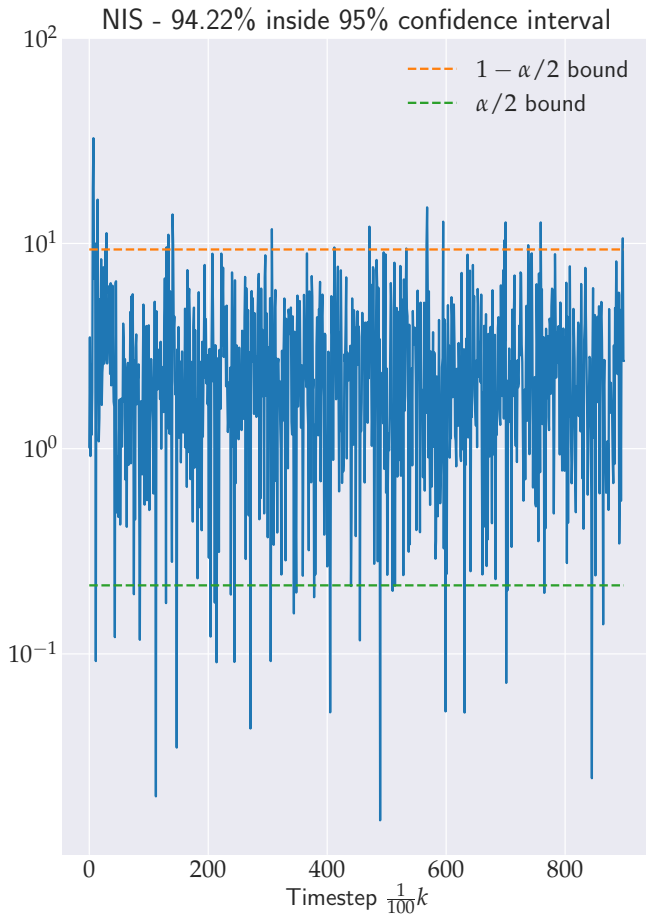


Figure A.6: *NIS from simulated data. Note that the y-axis is logarithmic. NIS is only computed at arrival of a GNSS measurement, so the time scale is every* 100^{th} *timestep* k*.*

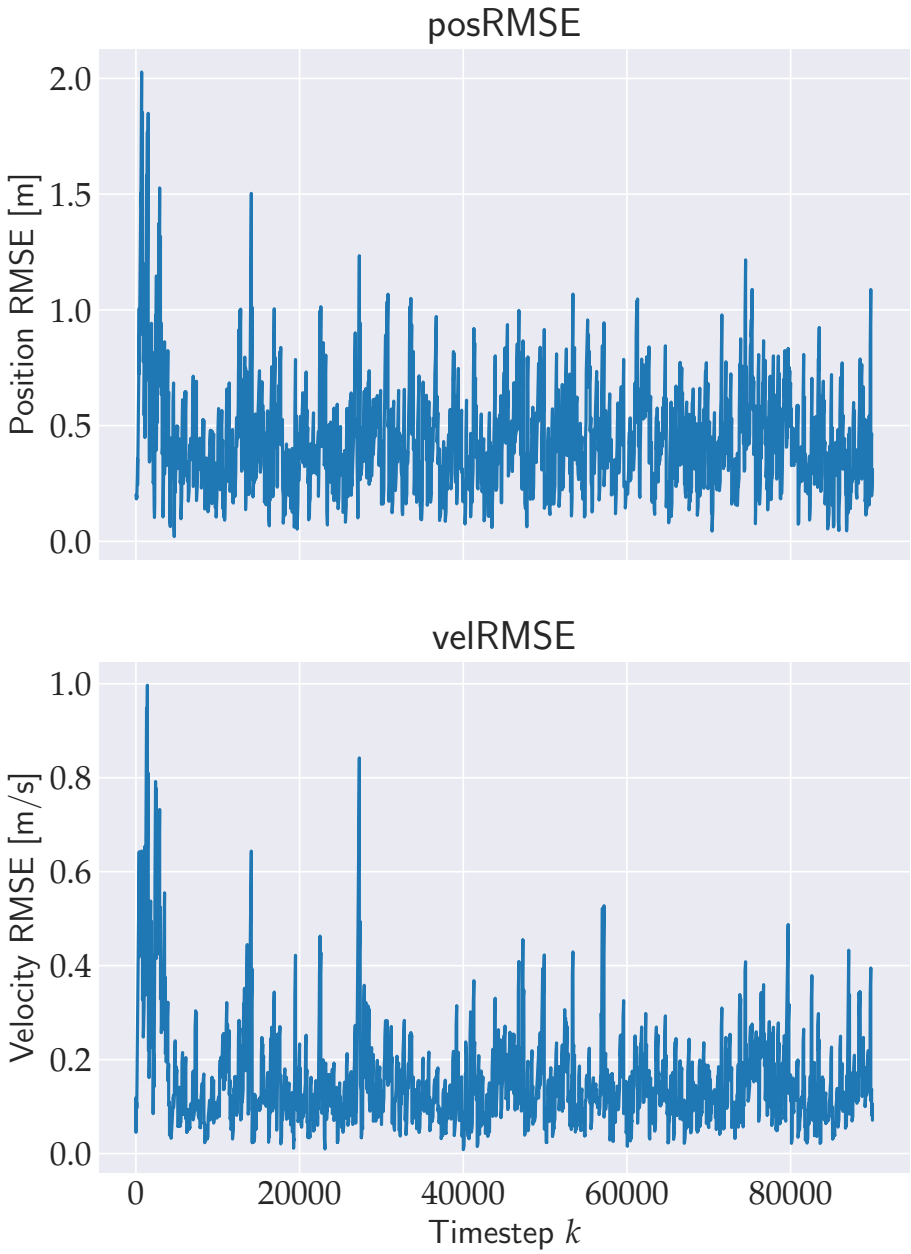


Figure A.7: *RMSE of position and velocity from simulated data.*

A.2 Results from real data

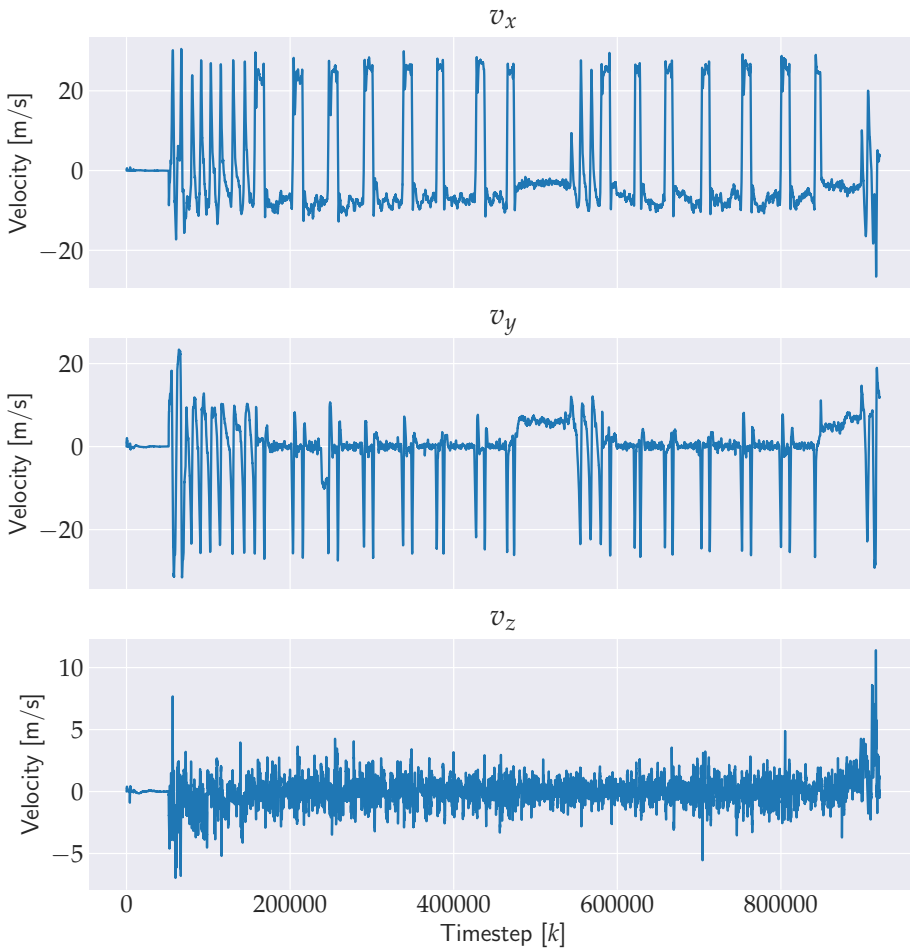


Figure A.8: *Velocity estimate from real data.*

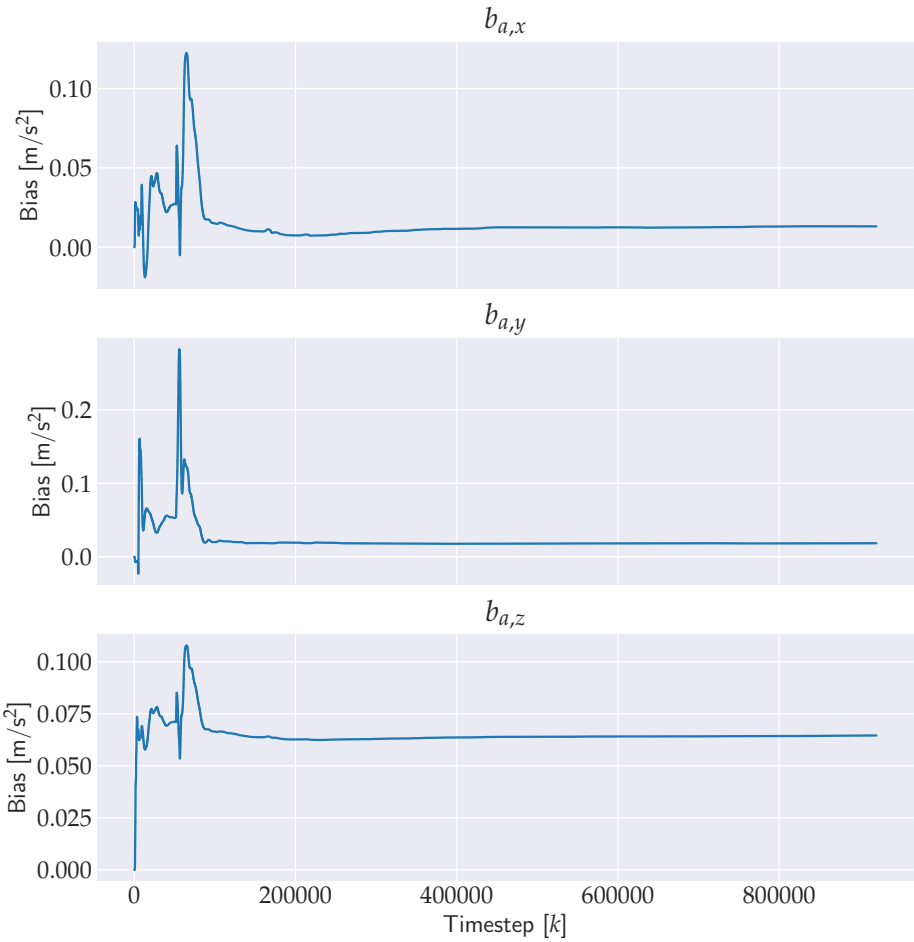


Figure A.9: *Accelerometer bias estimate from real data.*

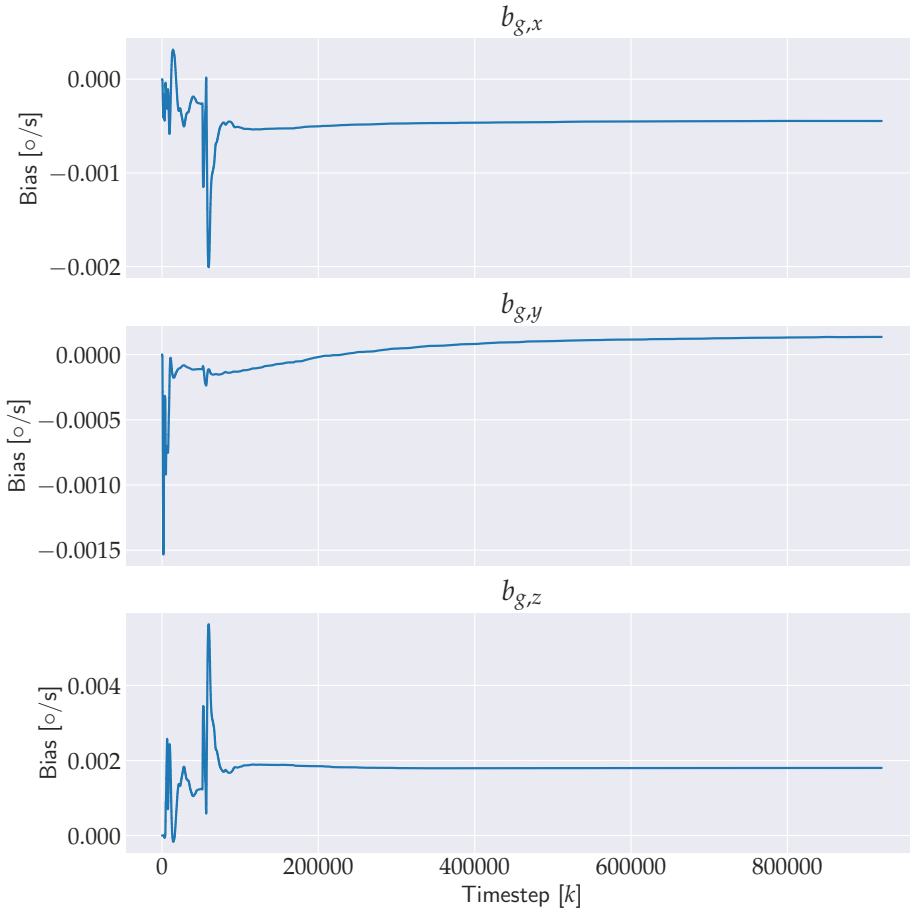


Figure A.10: *Gyroscope bias estimate from real data.*

B | Attitude related mathematics

B.1 Arithmetics with quaternions

To use quaternions, the arithmetic operations must be defined. Addition and subtraction are trivially defined as

$$q_1 + q_2 = \begin{bmatrix} \eta_1 \\ \epsilon_1 \end{bmatrix} + \begin{bmatrix} \eta_2 \\ \epsilon_2 \end{bmatrix} = \begin{bmatrix} \eta_1 + \eta_2 \\ \epsilon_1 + \epsilon_2 \end{bmatrix}$$

and

$$q_1 - q_2 = \begin{bmatrix} \eta_1 \\ \epsilon_1 \end{bmatrix} - \begin{bmatrix} \eta_2 \\ \epsilon_2 \end{bmatrix} = \begin{bmatrix} \eta_1 - \eta_2 \\ \epsilon_1 - \epsilon_2 \end{bmatrix}.$$

which follows intuitively from (6.5) and (6.8).

Multiplication is less intuitive, and can be derived from the formulation in (6.5), as

$$q_1 \otimes q_2 = \begin{bmatrix} \eta_1\eta_2 - \epsilon_1^\mathsf{T}\epsilon_2 \\ \eta_2\epsilon_1 + \eta_1\epsilon_2 + \epsilon_1 \times \epsilon_2 \end{bmatrix}, \tag{B.1}$$

with $\otimes$ defined as the *quaternion multiplication operator*. Some remarks regarding (B.1) are in order. Firstly, with the presence of the cross product in the vector part of the product, it is evident that quaternion multiplication is not commutative. That is,

$$q_1 \otimes q_2 \neq q_2 \otimes q_1$$

in general. However, it is associative and distributive, meaning that

$$q_1 \otimes q_2 \otimes q_3 = (q_1 \otimes q_2) \otimes q_3 = q_1 \otimes (q_2 \otimes q_3)$$

and

$$\boldsymbol{q}_1 \otimes (\boldsymbol{q}_2 + \boldsymbol{q}_3) = \boldsymbol{q}_1 \otimes \boldsymbol{q}_2 + \boldsymbol{q}_1 \otimes \boldsymbol{q}_3$$

for all $\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3 \in \mathbb{R}^4$. Additionally, the quaternion product is linear in both its arguments, or *bilinear*, meaning (B.1) can be rewritten as a matrix-vector product as

$$\begin{aligned} \boldsymbol{q}_1 \otimes \boldsymbol{q}_2 &= L(\boldsymbol{q}_1)\boldsymbol{q}_2 \\ &= R(\boldsymbol{q}_2)\boldsymbol{q}_1 \end{aligned}$$

where $L(\cdot)$ and $R(\cdot)$ are the left and right quaternion product matrix operators, respectively. By inspecting (B.1), these are found to be

$$L(\boldsymbol{q}) = \begin{bmatrix} \eta & -\epsilon_1 & -\epsilon_2 & -\epsilon_3 \\ \epsilon_1 & \eta & -\epsilon_3 & \epsilon_2 \\ \epsilon_2 & \epsilon_3 & \eta & -\epsilon_1 \\ \epsilon_3 & -\epsilon_2 & \epsilon_1 & \eta \end{bmatrix}, \tag{B.2a}$$

$$R(\boldsymbol{q}) = \begin{bmatrix} \eta & -\epsilon_1 & -\epsilon_2 & -\epsilon_3 \\ \epsilon_1 & \eta & \epsilon_3 & -\epsilon_2 \\ \epsilon_2 & -\epsilon_3 & \eta & \epsilon_1 \\ \epsilon_3 & \epsilon_2 & -\epsilon_1 & \eta \end{bmatrix}. \tag{B.2b}$$

The matrices in (B.2) are useful when differentiating quaternion products, for example

$$\frac{\partial(\boldsymbol{q}_1 \otimes \boldsymbol{q}_2)}{\partial \boldsymbol{q}_1} = \frac{\partial(R(\boldsymbol{q}_2)\boldsymbol{q}_1)}{\partial \boldsymbol{q}_1} = R(\boldsymbol{q}_2),$$

which may not otherwise be intuitive to compute. Lastly, as for ordinary functions, differentiating a quaternion product obeys the product rule, meaning that

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{q}_1(t) \otimes \boldsymbol{q}_2(t)\right) = \dot{\boldsymbol{q}}_1(t) \otimes \boldsymbol{q}_2(t) + \boldsymbol{q}_1(t) \otimes \dot{\boldsymbol{q}}_2(t). \tag{B.3}$$

It is important to remember that the quaternion product still is not commutative, such that

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{q}^2(t) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{q}(t) \otimes \boldsymbol{q}(t)\right) = \dot{\boldsymbol{q}}(t) \otimes \boldsymbol{q}(t) + \boldsymbol{q}(t) \otimes \dot{\boldsymbol{q}}(t) \neq 2\boldsymbol{q}(t) \otimes \dot{\boldsymbol{q}}(t).$$

The inverse of a quaternion $\boldsymbol{q}$ can be defined from its *conjugate* $\boldsymbol{q}^*$, which

itself is defined to be

$$\boldsymbol{q}^* = \begin{bmatrix} \eta \\ -\boldsymbol{\epsilon} \end{bmatrix}. \tag{B.4}$$

Note that the conjugate of a quaternion product is the product of each quaternion conjugated in the opposite order,

$$\left(\boldsymbol{q}_1 \otimes \boldsymbol{q}_2\right)^* = \boldsymbol{q}_2^* \otimes \boldsymbol{q}_1^*. \tag{B.5}$$

The inverse $\boldsymbol{q}^{-1}$ is then given as

$$\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{||\boldsymbol{q}||^2}$$

where $||\boldsymbol{q}||$ is the familiar vector norm applied to a quaternion,

$$\begin{aligned} ||\boldsymbol{q}|| &= \sqrt{\eta^2 + \boldsymbol{\epsilon}^\mathsf{T}\boldsymbol{\epsilon}} \\ &= \sqrt{\eta^2 + \epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2}. \end{aligned}$$

Notice that when $\boldsymbol{q}$ is a *unit quaternion*, that is, $||\boldsymbol{q}|| = 1$, then $\boldsymbol{q}^{-1} = \boldsymbol{q}^*$. In other words, the conjugate and inverse are equal.

Multiplication of a quaternion with its inverse results in the *identity quaternion*

$$\boldsymbol{q}_\mathbf{I} = \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix}, \tag{B.6}$$

which holds commutatively. Specifically,

$$\boldsymbol{q} \otimes \boldsymbol{q}^{-1} = \boldsymbol{q}^{-1} \otimes \boldsymbol{q} = \boldsymbol{q}_\mathbf{I}.$$

The identity quaternion has the property

$$\boldsymbol{q} \otimes \boldsymbol{q}_\mathbf{I} = \boldsymbol{q}_\mathbf{I} \otimes \boldsymbol{q} = \boldsymbol{q}$$

for all $\boldsymbol{q}$.

B.2 Composing quaternions

Composing quaternions, that is, chaining together consecutive transformations, is done by simply multiplying together the different quaternions in the correct order. Given $\boldsymbol{q}_a^b$ that transforms from frame a to frame b and $\boldsymbol{q}_b^c$ that transforms from frame b to c, the quaternion $\boldsymbol{q}_a^c$ that transforms from frame a to frame c directly is given by

$$\boldsymbol{q}_a^c = \boldsymbol{q}_b^c \otimes \boldsymbol{q}_a^b. \tag{B.7}$$

B.3 R as member of $SO(3)$

The name *Special Orthogonal* in $SO(3)$ can be made clear by inserting (6.14) into (6.1). This yields

$$\begin{aligned}\boldsymbol{u}^{a\mathsf{T}}\boldsymbol{u}^a &= \boldsymbol{u}^{b\mathsf{T}}\boldsymbol{u}^b \\ &= (\mathbf{R}_a^b\boldsymbol{u}^a)^\mathsf{T}(\mathbf{R}_a^b\boldsymbol{u}^a) \\ &= \boldsymbol{u}^{a\mathsf{T}}\mathbf{R}_a^{b\mathsf{T}}\mathbf{R}_a^b\boldsymbol{u}^a\end{aligned}$$

which implies

$$\mathbf{R}_a^{b\mathsf{T}}\mathbf{R}_a^b = \mathbf{I} \implies \mathbf{R}_a^{b\mathsf{T}} = (\mathbf{R}_a^b)^{-1} \tag{B.8}$$

meaning that $\mathbf{R}_a^b$ is *orthonormal*, or *Orthogonal* with unit column vectors. The *Special* in $SO(3)$ comes from taking the determinant of (B.8), as

$$\begin{aligned}\det(\mathbf{R}_a^{b\mathsf{T}}\mathbf{R}_a^b) &= \det(\mathbf{I}) \\ \det(\mathbf{R}_a^{b\mathsf{T}})\det(\mathbf{R}_a^b) &= 1 \\ \det(\mathbf{R}_a^b)^\mathsf{T}\det(\mathbf{R}_a^b) &= 1 \\ \det(\mathbf{R}_a^b)^2 &= 1 \\ \det(\mathbf{R}_a^b) &= \pm 1.\end{aligned}$$

Choosing $\det(\mathbf{R}_a^b) = -1$ results in a *reflection* in $\mathbb{R}^3$, which are transformations that do not form a Lie group and is of no interest here. Thus, all rotation matrices have determinant equal to unity, making them *Special*.

B.4 The skew operator

The skew operator $S(\cdot)$ is defined as

$$S(\boldsymbol{\omega}) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{B.9}$$

and provides a mapping $\mathbb{R}^3 \rightarrow \mathfrak{so}(3)$, where $\mathfrak{so}(3)$ is the *Lie algebra* of the Lie group $SO(3)$ and contains all derivatives of $\mathbf{R}$ at its origin, where $\mathbf{R}(0) = \mathbf{I}$.

Remarkably, $S(\cdot)$ also provides an alternative way of computing the cross product between two vectors, as

$$S(\boldsymbol{u})\boldsymbol{v} = \boldsymbol{u} \times \boldsymbol{v} \tag{B.10}$$

by construction of $S(\cdot)$. Note that this implies that

$$\begin{aligned} S(\boldsymbol{u})\boldsymbol{v} &= \boldsymbol{u} \times \boldsymbol{v} \\ &= -(\boldsymbol{v} \times \boldsymbol{u}) \\ &= -S(\boldsymbol{v})\boldsymbol{u} \end{aligned} \tag{B.11}$$

and, where $\alpha, \beta \in \mathbb{R}$,

$$\begin{aligned} S(\alpha\boldsymbol{u} + \beta\boldsymbol{v})\boldsymbol{w} &= (\alpha\boldsymbol{u} + \beta\boldsymbol{v}) \times \boldsymbol{w} \\ &= \alpha(\boldsymbol{u} \times \boldsymbol{w}) + \beta(\boldsymbol{v} \times \boldsymbol{w}) \\ &= \alpha S(\boldsymbol{u})\boldsymbol{w} + \beta S(\boldsymbol{v})\boldsymbol{w} \\ &= \left(\alpha S(\boldsymbol{u}) + \beta S(\boldsymbol{v})\right)\boldsymbol{w}, \end{aligned}$$

implying that

$$S(\alpha\boldsymbol{u} + \beta\boldsymbol{v}) = \alpha S(\boldsymbol{u}) + \beta S(\boldsymbol{v}). \tag{B.12}$$

B.5 Composing rotation matrices

Composing rotation matrices is done identically to quaternions. Given $\mathbf{R}_a^b$ that transforms from frame a to frame b and $\mathbf{R}_b^c$ that transforms from frame b to c, the rotation matrix $\mathbf{R}_a^c$ that transforms from frame a to frame c directly is given

by

$$\mathbf{R}_a^c = \mathbf{R}_b^c \mathbf{R}_a^b. \tag{B.13}$$

B.6 Calculating R from other attitude representations

While quaternions are useful for storing an attitude in a state vector due to only needing four parameters, rotation matrices are the most useful representation in a practical application for actually transforming vectors. This is because the involved transformation is just a simple matrix-vector product in $\mathbb{R}^3$. As such, formulas for calculating $\mathbf{R}$ from parameters of other attitude representations are desirable.

The first and most basic way is to compose the rotation matrix from individual rotations about each axis,

$$\mathbf{R}(\varphi, \theta, \psi) = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\varphi) \tag{B.14}$$

where the order x, then y, then z is convention and φ, θ and ψ denote roll, pitch and yaw, respectively. The three matrices $\mathbf{R}_x$, $\mathbf{R}_y$ and $\mathbf{R}_z$ are given as

$$\mathbf{R}_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{bmatrix}, \tag{B.15a}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, \tag{B.15b}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{B.15c}$$

Another way is from the angle-axis representation, where the angle-axis $\boldsymbol{\phi} = \phi \boldsymbol{n}$ was a vector with direction parallel to the rotation axis and magnitude equal to the rotation angle. Such a rotation matrix is constructed from *Rodrigues rotation formula* as

$$\mathbf{R}(\phi, \boldsymbol{n}) = \mathbf{I} + \sin(\phi)S(\boldsymbol{n}) + (1 - \cos(\phi))S(\boldsymbol{n})^2 \tag{B.16}$$

where $S(\cdot)$ is the skew operator defined in (B.9). Lastly, the rotation matrix can be constructed from a unit quaternion $\boldsymbol{q} = [\eta \quad \boldsymbol{\epsilon}^T]^T$ as

$$\mathbf{R}(\boldsymbol{q}) = (\eta - \boldsymbol{\epsilon}^T\boldsymbol{\epsilon})\mathbf{I} + 2\eta S(\boldsymbol{\epsilon}) + 2\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T. \tag{B.17}$$

C | Snippet of C++ implementation

The following code is a snippet of the C++ implementation which performance is discussed in Chapter 10. Only the high-level part of the code is included, where the actual algorithm is implemented, and more low-level code that interfaces with ROS and the external sensors is left out. The library *Eigen* that is used is a linear algebra library in C++ that abstracts away the complicated low-level arithmetics involved in matrix and vector arithmetics, in addition to providing many convenient functions for other matrix operations. More information can be found in [41].

```
#include "eskf/eskf.h"
#include <cmath>
#include <Eigen/Core>

ESKF::ESKF()
{
    nom_state_.quat = Eigen::Quaterniond(1,0,0,0);
    nom_state_.vel = Eigen::Vector3d(0,0,0);
    nom_state_.pos = Eigen::Vector3d(0,0,0);
    nom_state_.acc_bias = Eigen::Vector3d(0,0,0);
    nom_state_.gyro_bias = Eigen::Vector3d(0,0,0);

    Qerr_ = Eigen::MatrixXd::Zero(12, 12);
    R_ = Eigen::Matrix3d::Zero(3, 3);
    P_ = Eigen::MatrixXd::Zero(15, 15);
}

void ESKF::initialize(const double Qacc, const double Qacc_bias, const double
↪  p_acc,
                      const double Qgyro, const double Qgyro_bias, const double
                      ↪  p_gyro,
```

```
                    const double RGNSSx, const double RGNSSy, const double
                    ↪  RGNSSz,
                    const NominalState& init_state, const Eigen::MatrixXd&
                    ↪  init_covariance, double g,
                    const double imu_sample_rate)
{
    Eigen::Matrix3d I = Eigen::Matrix3d::Identity();

    Qerr_.block<3,3>(0,0) = Qacc * I;
    Qerr_.block<3,3>(3,3) = Qgyro * I;
    Qerr_.block<3,3>(6,6) = Qacc_bias * I;
    Qerr_.block<3,3>(9,9) = Qgyro_bias * I;

    R_(0,0) = RGNSSx;
    R_(1,1) = RGNSSy;
    R_(2,2) = RGNSSz;

    p_acc_ = p_acc;
    p_gyro_ = p_gyro;

    nom_state_ = init_state;
    P_ = init_covariance;

    G_ = Eigen::Vector3d(0,0,g);

    imu_sample_rate_ = imu_sample_rate;
    IMUdt_ = 1.0 / imu_sample_rate_;
}

void ESKF::readIMU(const Eigen::Vector3d& acc, const Eigen::Vector3d& gyro)
{
    acc_raw_ = acc;
    gyro_raw_ = gyro;
}

void ESKF::readGNSS(const Eigen::Vector3d& gnss)
{
    gnss_raw_ = gnss;
}

Eigen::Quaterniond ESKF::getAttitude() const
{
    return nom_state_.quat;
}

Eigen::Vector3d ESKF::getVelocity() const
{
```

```
65      return nom_state_.vel;
66  }
67
68  Eigen::Vector3d ESKF::getPosition() const
69  {
70      return nom_state_.pos;
71  }
72
73  void ESKF::predict()
74  {
75      nom_state_  = predictNominal();
76      P_          = predictCovariance();
77  }
78
79  NominalState ESKF::predictNominal()
80  {
81      Eigen::Vector3d acc, ang;
82      Eigen::Matrix3d I = Eigen::Matrix3d::Identity();
83      Eigen::Quaterniond dq = Eigen::Quaterniond::Identity();
84      NominalState nom_state_pred;
85
86      // Predict nominal orientation (quaternion)
87      ang = (gyro_raw_ - nom_state_.gyro_bias)*IMUdt_;
88      double ang_norm = ang.norm();
89      if (abs(ang_norm) > 1e-8)
90      {
91          dq.w() = cos( ang_norm/2.0 );
92          dq.vec() = sin( ang_norm/2.0 ) * ang/ang_norm;
93      }
94
95      nom_state_pred.quat = nom_state_.quat * dq;
96      nom_state_pred.quat.normalize();
97
98      acc = nom_state_.quat.toRotationMatrix() * (acc_raw_ - nom_state_.acc_bias) +
        ↪  G_;
99
100     // Predict nominal position, velocity, bias
101     nom_state_pred.pos = nom_state_.pos + nom_state_.vel*IMUdt_ +
        ↪  0.5*acc*IMUdt_*IMUdt_;
102     nom_state_pred.vel = nom_state_.vel + acc*IMUdt_;
103     nom_state_pred.acc_bias = nom_state_.acc_bias -
        ↪  p_acc_*I*nom_state_.acc_bias*IMUdt_;
104     nom_state_pred.gyro_bias = nom_state_.gyro_bias -
        ↪  p_gyro_*I*nom_state_.gyro_bias*IMUdt_;
105
106     return nom_state_pred;
107 }
```

```
108
109 Eigen::MatrixXd ESKF::predictCovariance()
110 {
111     Eigen::MatrixXd P_pred(15,15);
112
113     Eigen::MatrixXd A = Aerr();
114     Eigen::MatrixXd G = Gerr();
115     auto [Ad, GQGd] = discretizeErrorMats(A, G);
116
117     P_pred = Ad * P_ * Ad.transpose() + GQGd;
118
119     return P_pred;
120 }
121
122 Eigen::MatrixXd ESKF::Aerr()
123 {
124     // Error-state vector: x = [d_pos.T, d_vel.T, d_ang.T, d_acc_bias.T,
    ↪  d_gryo_bias.T].T (15x1)
125     Eigen::MatrixXd A = Eigen::MatrixXd::Zero(15,15);
126
127     Eigen::Matrix3d I = Eigen::Matrix3d::Identity();
128     Eigen::Matrix3d Rq = nom_state_.quat.toRotationMatrix();
129     Eigen::Matrix3d Sa = skewSymmetricMatrix(acc_raw_ - nom_state_.acc_bias);
130     Eigen::Matrix3d Sw = skewSymmetricMatrix(gyro_raw_ - nom_state_.gyro_bias);
131
132     A.block<3,3>(0,3)   =  I;
133     A.block<3,3>(3,6)   = -1.0 * Rq * Sa;
134     A.block<3,3>(3,9)   = -1.0 * Rq;
135     A.block<3,3>(6,6)   = -1.0 * Sw;
136     A.block<3,3>(6,12)  = -1.0 * I;
137     A.block<3,3>(9,9)   = -1.0 * p_acc_ * I;
138     A.block<3,3>(12,12) = -1.0 * p_gyro_ * I;
139
140     return A;
141 }
142
143 Eigen::MatrixXd ESKF::Gerr()
144 {
145     // Process noise vector: n = [acc_n.T, gyro_n.T, acc_w.T, gyro_w.T].T (12x1)
146     Eigen::MatrixXd G = Eigen::MatrixXd::Zero(15,12);
147
148     Eigen::Matrix3d I = Eigen::Matrix3d::Identity();
149     Eigen::Matrix3d Rq = nom_state_.quat.toRotationMatrix();
150
151     G.block<3,3>(3,0)  = -1.0 * Rq;
152     G.block<3,3>(6,3)  = -1.0 * I;
153     G.block<3,3>(9,6)  = I;
```

```
154     G.block<3,3>(12,9) = I;
155 
156     return G;
157 }
158 
159 std::tuple<Eigen::MatrixXd, Eigen::MatrixXd> ESKF::discretizeErrorMats(const
    ↪  Eigen::MatrixXd& A, const Eigen::MatrixXd& G)
160 {
161     // Allocate
162     Eigen::MatrixXd Ad(15,15);
163     Eigen::MatrixXd GQGd(15,15);
164     Eigen::MatrixXd V = Eigen::MatrixXd::Zero(30,30);
165     Eigen::MatrixXd Vexp(30,30);
166     Eigen::MatrixXd V1(15,15);
167     Eigen::MatrixXd V2(15,15);
168 
169     // Using Van Loan's formula
170     V.block<15,15>(0,0) = -1.0 * A;
171     V.block<15,15>(0,15) = G * Qerr_ * G.transpose();
172     V.block<15,15>(15,15) = A.transpose();
173     V = V * IMUdt_;
174 
175     Vexp = V.exp();
176     V1 = Vexp.block<15,15>(15,15);
177     V2 = Vexp.block<15,15>(0,15);
178 
179     Ad = V1.transpose();
180     GQGd = V1.transpose() * V2;
181 
182     return {Ad, GQGd};
183 }
184 
185 void ESKF::update()
186 {
187     // Allocate matrices for error state update
188     Eigen::MatrixXd Hx = Eigen::MatrixXd::Zero(3,16);
189     Hx.block<3,3>(0,0) = Eigen::Matrix3d::Identity();
190 
191     Eigen::MatrixXd X_dx = Eigen::MatrixXd::Zero(16,15);
192     Eigen::MatrixXd Q_dtheta(4,3);
193     double qx = nom_state_.quat.x();
194     double qy = nom_state_.quat.y();
195     double qz = nom_state_.quat.z();
196     double qw = nom_state_.quat.w();
197     Q_dtheta << -qx, -qy, -qz,
198                  qw, -qz,  qy,
199                  qz,  qw, -qx,
```

```
                -qy,  qx,  qw;
    Q_dtheta = Q_dtheta * 0.5;

    X_dx.block<6,6>(0,0) = Eigen::MatrixXd::Identity(6,6);
    X_dx.block<4,3>(6,6) = Q_dtheta;
    X_dx.block<6,6>(10,9) = Eigen::MatrixXd::Identity(6,6);

    Eigen::MatrixXd H = Hx * X_dx;
    Eigen::MatrixXd S = H*P_*H.transpose() + R_;
    Eigen::VectorXd x_nom(16);
    x_nom << nom_state_.pos,
             nom_state_.vel,
             nom_state_.quat.w(),
             nom_state_.quat.vec(),
             nom_state_.acc_bias,
             nom_state_.gyro_bias;
    Eigen::MatrixXd I = Eigen::MatrixXd::Identity(15,15);

    // Update error state/covariance (assumes GNSS measurements in NED-frame)
    Eigen::MatrixXd K = P_*H.transpose()*S.inverse();
    Eigen::VectorXd delta_x = K*(gnss_raw_ - Hx*x_nom);

    err_state_.pos = delta_x.block<3,1>(0,0);
    err_state_.vel = delta_x.block<3,1>(3,0);
    err_state_.ang = delta_x.block<3,1>(6,0);
    err_state_.acc_bias = delta_x.block<3,1>(9,0);
    err_state_.gyro_bias = delta_x.block<3,1>(12,0);

    P_ = (I - K*H)*P_*(I - K*H).transpose() + K*R_*K.transpose();

    // Inject error into nominal state
    inject();
}

void ESKF::inject()
{
    // Get quaternion from euler angles in error state
    double ang_x = err_state_.ang(0);
    double ang_y = err_state_.ang(1);
    double ang_z = err_state_.ang(2);
    Eigen::Vector3d ang_err(ang_x, ang_y, ang_z);
    Eigen::Quaterniond q_err(1, 0.5*ang_x, 0.5*ang_y, 0.5*ang_z);

    // Inject error state into nominal state
    nom_state_.pos = nom_state_.pos + err_state_.pos;
    nom_state_.vel = nom_state_.vel + err_state_.vel;
    nom_state_.quat = nom_state_.quat * q_err;
```

```
    nom_state_.quat.normalize();
    nom_state_.acc_bias = nom_state_.acc_bias + err_state_.acc_bias;
    nom_state_.gyro_bias = nom_state_.gyro_bias + err_state_.gyro_bias;

    // Reset error
    err_state_.pos = Eigen::Vector3d::Zero();
    err_state_.vel = Eigen::Vector3d::Zero();
    err_state_.ang = Eigen::Vector3d::Zero();
    err_state_.acc_bias = Eigen::Vector3d::Zero();
    err_state_.gyro_bias = Eigen::Vector3d::Zero();

    Eigen::MatrixXd G = Eigen::MatrixXd::Identity(15,15);
    Eigen::Matrix3d Serr = skewSymmetricMatrix(0.5*ang_err);
    G.block<3,3>(6,6) = Eigen::Matrix3d::Identity() - Serr;

    P_ = G * P_ * G.transpose();
}

// Helpers

Eigen::Matrix3d ESKF::skewSymmetricMatrix(const Eigen::Vector3d& v)
{
    Eigen::Matrix3d skew = Eigen::Matrix3d::Zero();
    skew <<    0, -v[2],  v[1],
            v[2],     0, -v[0],
           -v[1],  v[0],    0;

    return skew;
}
```

Bibliography

[1] S. Thrun, W. Burgard and D. Fox, *Probabilistic Robotics*, 1st ed. The MIT Press, 2005, ISBN: 0262201623,9780262201629.

[2] E. F. Brekke. Fundamentals of Sensor Fusion, [Online]. Available: http://folk.ntnu.no/edmundfo/msc2019-2020/sf13chapters.pdf (visited on 3rd May 2020).

[3] S. Särkkä, *Bayesian filtering and smoothing*, 1st ed. Cambridge University Press, 2013.

[4] H. Wolfgang and S. Léopold, *Applied multivariate statistical analysis*. Springer, 2015.

[5] R. Wang. (Nov. 2006). Marginal and conditional distributions of multivariate normal distribution, [Online]. Available: http://fourier.eng.hmc.edu/e161/lectures/gaussianprocess/node7.html (visited on 24th Mar. 2020).

[6] B.-N. Vo and W.-K. Ma, 'The Gaussian Mixture Probability Hypothesis Density Filter', *IEEE Transactions on Signal Processing*, vol. 54, no. 11, 4091–4104, 2006. DOI: 10.1109/tsp.2006.881190.

[7] Y. Ho and R. Lee, 'A Bayesian approach to problems in stochastic estimation and control', *IEEE Transactions on Automatic Control*, vol. 9, no. 4, 333–339, 1964. DOI: 10.1109/tac.1964.1105763.

[8] B. Ristic, S. Arulampalam and N. Gordon, *Beyond the Kalman filter: Particle Filters for Tracking Applications*. Artech House, 2004.

[9] E. F. Brekke and E. F. Wilthil, 'Suboptimal Kalman filters for target tracking with navigation uncertainty in one dimension', *2017 IEEE Aerospace Conference*, 2017. DOI: 10.1109/aero.2017.7943601.

[10] L.-C. N. Tokle, 'Multi target tracking - Using random finite sets with a hybrid state space and approximations', Master's thesis, Norwegian University of Science and Technology, Aug. 2018.

[11] T.-T. Lu and S.-H. Shiou, 'Inverses of 2 × 2 block matrices', *Computers & Mathematics with Applications*, vol. 43, no. 1-2, 119–129, 2002. DOI: 10.1016/s0898-1221(01)00278-4.

[12] R. S. Bucy and P. D. Joseph, *Filtering for Stochastic Processes with Applications to Guidance*. John Wiley & Sons, 1968.

[13] J. Farrell, *Aided navigation: GPS with high rate sensors*. McGraw-Hill, 2008.

[14] R. Zanetti and K. J. Demars, 'Joseph formulation of unscented and quadrature filters with application to consider states', *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 6, 1860–1864, 2013. DOI: 10.2514/1.59935.

[15] C.-T. Chen, *Linear system theory and design*, 3rd. Oxford University Press, 1999.

[16] S. Julier and J. Uhlmann, 'Unscented Filtering and Nonlinear Estimation', *Proceedings of the IEEE*, vol. 92, no. 3, 401–422, 2004. DOI: 10.1109/jproc.2003.823141.

[17] T. I. Fossen, *Handbook of Marine Mraft Hydrodynamics and Motion Control*. Wiley, 2014.

[18] J. Solà. (Oct. 2017). Quaternion kinematics for the error-state KF, [Online]. Available: http://www.iri.upc.edu/people/jsola/JoanSola/objectes/notes/kinematics.pdf (visited on 23rd Jan. 2020).

[19] O. Egeland and J. T. Gravdahl, *Modeling and Simulation for Automatic Control*. Marine Cybernetics, 2003.

[20] M. D. Shuster, 'A survey of attitude representations', *The Journal of the Astronautical Sciences*, vol. 41, no. 4, pp. 439–517, 1993.

[21] A. Baker, *Matrix groups: an introduction to Lie group theory*. Springer, 2002.

[22] W. R. Hamilton, 'On Quaternions; or on a new System of Imaginaries in Algebra', *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science*, vol. xxv, pp. 489–495, 1844.

[23] M. L. Sollie, T. H. Bryne and T. A. Johansen, 'Pose Estimation of UAVs Based on INS Aided by Two Independent Low-Cost GNSS Receivers', *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019. DOI: 10.1109/icuas.2019.8797746.

[24] P. Maybeck, *Stochastic models, estimation, and control*. Academic Press, 1979, vol. 3.

[25] J. R. Carpenter and C. N. D'Souza. (Apr. 2018). Navigation Filter Best Practices - NASA, [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20180003657.pdf (visited on 12th May 2020).

[26] P. Misra and P. Enge, *Global positioning system: signals, measurements, and performance*. Ganga-Jamuna Press, 2012.

[27] E. D. Kaplan and C. J. Hegarty, *Understanding GPS: principles and wexxd*. Artech House, 2006.

[28] R. Wang. (Jul. 2014). National geospatial-intelligence agency (nga) standardization document - Department of defense World Geodetic System 1984, [Online]. Available: http://w3.uch.edu.tw/ccchang50/NGA.STND.0036_1.0.0_WGS84.pdf (visited on 4th May 2020).

[29] J. G. Balchen, T. Andresen and B. A. Foss, *Reguleringsteknikk*, 6th ed. Institutt for teknisk kybernetikk, NTNU, 2016.

[30] 'Computing integrals involving the matrix exponential', *IEEE Transactions on Automatic Control*, vol. 23, no. 3, 395–404, 1978. DOI: 10.1109/tac.1978.1101743.

[31] Y. Bar-Shalom, X. Li and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*. John Weily & Sons, 2001.

[32] K. Reif, S. Gunther, E. Yaz and R. Unbehauen, 'Stochastic stability of the discrete-time extended Kalman filter', *IEEE Transactions on Automatic Control*, vol. 44, no. 4, 714–728, 1999. DOI: 10.1109/9.754809.

[33] M. L. Sollie, 'Estimation of UAV Position, Velocity and Attitude Using Tightly Coupled Integration of IMU and a Dual GNSS Receiver Setup', Master's thesis, Norwegian University of Science and Technology, 2018.

[34] *STIM300 Inertia Measurement Unit*, TS1524, Rev. 25, Sensonor, Feb. 2020.

[35] *NEO/LEA-M8T*, UBX-15025193, Rev. 3, uBlox, Jun. 2016.

[36] SenTiBoard - About, [Online]. Available: https://sentiboard.com/about.html (visited on 9th May 2020).

[37] P. D. Groves, *Principles of GNSS, inertial, and multisensor integrated navigation systems*, 2nd. Artech House, 2013.

[38] D. Allan, 'Statistics of atomic frequency standards', *Proceedings of the IEEE*, vol. 54, no. 2, 221–230, 1966. DOI: 10.1109/proc.1966.4634.

[39] N. El-Sheimy, H. Hou and X. Niu, 'Analysis and Modeling of Inertial Sensors Using Allan Variance', *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 1, 140–149, 2008. DOI: 10.1109/tim.2007.908635.

[40] MathWorks. Inertial sensor noise analysis using allan variance, [Online]. Available: https://se.mathworks.com/help/nav/ug/inertial-sensor-noise-analysis-using-allan-variance.html (visited on 9th May 2020).

[41] Eigen - Main Page, [Online]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page (visited on 19th May 2020).

Fagerli, A. T.; Severinsen, O. A.

The Error-State Kalman Filter for Singularity-Free State Estimation in Inertial Navigation Systems