Master's thesis

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

Andreas Haugland

Factorized Kalman filter implementation for aided inertial navigation systems

Master's thesis in Cybernetics and Robotics
Supervisor: Torleiv H. Bryne
January 2022

NTNU
Norwegian University of Science and Technology

Andreas Haugland

Factorized Kalman filter implementation for aided inertial navigation systems

Master's thesis in Cybernetics and Robotics
Supervisor: Torleiv H. Bryne
January 2022

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

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

MSc THESIS DESCRIPTION SHEET

Name:	Andreas Haugland
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Faktorisert Kalman filter implementasjon for integrerte treghetsnavigasjonssystemer
Thesis title (English):	Factorized Kalman filter implementation for aided inertial navigation systems

Thesis Description: The NTNU UAVlab has developed a navigation systems toolbox to run inertial navigations (INS) offline used for experimentation, development, and validation. The toolbox has proven to be useful, but further developments are beneficial to increase performance, utility, and robustness, especially if the state space is large and the number of aiding measurements available is heigh.

The following tasks for the specialization project thesis should be considered.

1. Perform a literature review on
    a. Standard/batchwise and sequential EKF implementations for aided INS
    b. Kalman implementations based on covariance factorization such as QR, LU or UD/UDU
2. Decide on a factorization method and implement a factorized Kalman implementations for aided INS. Justify your choice.
3. Reuse and update the motion simulator and sensor simulator developed in the specialization project.
4. Evaluate performance and runtime using simulated and benchmark with the non-factorized implementations from the specialization project thesis. Include potential improvements of the specialization project work.
5. Present and discuss your results.
6. Conclude your results and suggest further work.

Start date:	2021-08-16
Due date:	2022-01-10
Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Associate professor Torleiv H. Bryne, Dept. of Eng. Cybernetics, NTNU

Summary

To examine if the NTNU UAVlabs navigation system toolbox for offline inertial navigation could yield a better run time performance for cases with large measurement vectors, we tested three variations of the Error-State Kalman Filter. The performance metrics measured were run time, state root square mean error and state covariance.

The three implementations we tested were a batch wise implementation of the error-state Kalman filter which were used as the reference filter. The second variation were the sequential Kalman filter where the measurement update is done sequentially to avoid a matrix inversion when computing the Kalman gain. Lastly, a third filter variation which uses a strategy involving UDU-factorization of the covariance matrix in both the time-update and measurement-update step were chosen as a factorized filter variation.

To test the hypothesis we use a 16 state simulation model with simulated IMU- and aiding measurements, where m generic ranges were generated between beacons spread around the simulated trajectory and the simulated vessel. Each filter variation were separated into three timing categories: total-, time update- and measurement update run time. The timing were performed by running the simulation n = 200 times for a set of simulation durations before the average of the n measured run times were used to compare the variations. This experiment were done with m = 15 and 30 measurement separately.

This thesis showed that the sequential filter strategy yielded a improvement in measurement update run time with a speed up increase of 4.25% for m= 15 measurements and 81.65% for m = 30. Additionally, the reference measurement update run time increased at a near cubic rate when increasing m from 15 to 30. The UD-factorized variation were 137% slower in the time update and 1161% slower in the measurement update relative to the reference for m = 15, and 146% and 200% for m = 30 in the respective update modules.

The sequential variation displayed a numerical identical accuracy and performance in terms of RMSE and final estimated covariance for the longest simulation duration in this experiment. The UD-variation displayed a similar RMSE as the reference, and a small improvement in terms of magnitude in the final covariance estimation compared to the reference.

Sammendrag

For å undersøke om NTNU UAVlabs navigasjonssystemsprogramvare for simulering av treghetsnavigasjon kunne forbedres i tilfeller der det er mange målinger, utførte vi et eksperiment der vi sammenlignet tre variasjoner av et *error-state Kalman filter*. I denne testen sammenlignet vi kjøretid, feiltilstandens kvadratisk gjennomsnittsfeil og den akkumulerte tilstandsvariansen ved endt simulering.

De tre filterimplementasjonene som ble testet var den vanlige implementasjonen av filteret som oftest blir presentert i lærebøker, og som er basert på matriseinvertering under beregningen av Kalman-forsterkingen. Denne variasjonen av filteret ble også brukt som referansegrunnlag for å sammenligne de to andre. Den andre variasjonen var det sekvensielle Kalman-filteret der måleoppdateringen blir utført sekvensielt for å unngå matriseinverteringen under beregningen av Kalman-forsterkningen. Det siste og tredje filteret var en faktorisert variant av filteret basert på en strategi der kovariansmatrisen blir faktorisert gjennom UDU-faktorisering i både tids- og måleoppdateringen.

For å teste hvilket av de tre filterene som var best i forhold til testparameterene, implementerte vi en 16 tilstanders simuleringsmodell med simulerte bevegelses- og hjelpemålinger. Her ble m antall simulerte sendetårn spredd rundt den simulerte banen til fartøyet i modellen. Hver filtervariasjon ble oppdelt i tre tidskategorier separert i total kjøretid, kjøretid som tidsoppdateringen bruker av total kjøretid, samt kjøretid som måleoppdateringen bruker av total kjøretid. Tidsmålingen ble utført ved å kjøre simuleringen $N = 200$ ganger for et gitt sett med simuleringskjøretid. Av de 200 målte dataene, ble det beregnet en gjennomsnittlig kjøretid for de tre kategoriene som ble brukt for å sammenligne de tre variasjonene. I tillegg ble filterenes kvadratiske gjennomsnittsfeil og varians målt og sammenlignet mot hverandre. Dette eksperimentet ble gjort med henholdsvis $m = 15$ og 30 hjelpemålinger hver for seg.

Denne oppgaven resulterte i at den sekvensielle filterstrategien gav en forbedring i kjøretid for måleoppdateringen med en relativ forbedring til referansen på 4.25% for $m = 15$ hjelpemålinger, og 81.65% forbedring for $m = 30$. I tillegg så økte referansens kjøretid med en nær kvadratisk rate for denne økningen av m. Den UD-faktoriserte variasjonen viste en 137% nedgang i relativ kjøretid for tidsoppdateringen og 1161% nedgang i måleoppdateringen relativ til referansen for $m = 15$, samt en 146% og 200% nedgang for $m = 30$ i de respektive modulene.

Den sekvensielle variasjonen viste en numerisk identisk presisjon og ytelse når det kommer til den kvadratiske gjennomsnittsfeilen og den endelige estimerte kovariansen for den lengste simuleringstiden utført i dette eksperimentet. UD-variasjonen viste en lignende kvadratisk gjennomsnittsfeil relativt til referansen og en liten forbedring i størrelsen til kovariansen.

Preface

The work on this thesis was done in the fall of 2021 at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology. The work amounts to 30 study credits, and is a continuation of my project thesis done in the spring of 2021 [1].

The master thesis builds on the simulation model and the standard error-state Kalman filter implemented in the project thesis. The thesis also patches and re-implements the sequential variation into a properly working variation after uncovering a couple of errors in the former implementation. In addition to the standard error-state Kalman Filter, and the variation with sequential measurement update, this thesis implements and investigates a variation where the covariance matrix is factorized in to a UDU-shape in the time- and measurement update called the UD-factorized variation. The standard variation of the filter is used as a reference filter for which the sequential and UD-factorized variation were compared to. The simulation model and the filter implementations in this thesis were all implemented and run with Python 3.9.

In this thesis I compare the three variations in regards of run time, root mean square error of the error state and final estimated covariance for a given set of simulation duration on the same simulation model with m = 15 and m = 30 measurements. The comparison test were standardized to be as reproducible as possible.

I would like to thank my supervisor Torleiv H. Bryne for the advice given throughout this semester as well as the prior semester, and a special thanks for the feedback given throughout both theses.

Andreas Haugland,
Trondheim, Januar 2022

Contents

List of Figures

List of Tables

Listings

Part I

Introduction and problem background

Chapter 1

Introduction

1.1 Problem background

The NTNU UAVlab has developed a navigation systems toolbox to run inertial navigations offline used for experimentation, development and validation. The toolbox has proven to be useful, but further developments are beneficial to increase performance, utility and robustness.

An item for examination is the choice of error-state Kalman filter strategy to use in the toolbox. Depending on the INS design, the environment of operation, and the platform architecture, there are multiple considerations to be made regarding the time update and measurement update design optimization.

Groves [2] (p. 75) states that the processing load for Kalman filter implementations depend on the number of components in the state vector, measurement vector and system noise vector. If the number of states n are large, the covariance propagation step becomes increasingly more computationally demanding, while if the size of the measurement vector with m measurements increases, then the load increase becomes more significant in the Kalman gain computation in the measurement update. This establishes two important sub-problems for the filtering optimization, the time update steps and the measurement update steps.

The computational load in the measurement update comes largely from the required matrix inversion when computing the Kalman gain. The gain matrix $\boldsymbol{K}_k$ is given by the equation $\boldsymbol{K}_k = \boldsymbol{P}_{k-1}^{-}\boldsymbol{H}_k^T\boldsymbol{S}_k^{-1}$, where the size of $\boldsymbol{S}_k$ is $\in \mathbb{R}^{m\times m}$, where m is the number of measurements in the measurement vector. The inversion of this matrix have the computational complexity $O(m^3)$ when the inversion is performed with Gauss-Jordan elimination. This means that the computational load from scaling the measurement vector up from the minimum required to compute a 3D user position ramps up fast. How to improve on the measurement update then becomes a question of either finding a new way to optimize the inversion function, or to find another variation of the measurement update. Dan Simon [3] (p. 150) suggests the sequential Kalman filtering as a alternative filter strategy for larger number of measurements to ease the computational load in the Kalman filter algorithm. Instead of handling the aiding measurements all at once in a batch-wise manner such as is common in the default filter variation, the sequential handles the measurements one at the time. This results in a measurement update step where the computation of the Kalman gain depend on scalar multiplications if each of the aiding measurement are scalar. If the measurements have higher dimensions the maximum matrix inversion will the same size as the largest dimension vector measurement. In addition to this, by removing the need of a matrix inversion in the update, one also opens up for using this filter variation on architectures that does not have a matrix library with matrix inversion functions

built in. The sequential measurement update is also needed as a basis for the measurement update in the UD-factorized variation discussed below.

In NASAs *Navigation filter best practices* [4] (p. 63), D'Souza repeats Groves claim about the increasing computational load with increasing state space, and adds to it that for the usual Kalman filter, an enlargement of the state-space will also augment the nonlinear effects leading to filter divergence and non-positive definiteness of the covariance matrix. To fix this issue, D'Souza suggests a factorization method first discussed by Thornton in *"Triangular Covariance Factorization for Kalman Filtering"* [5]. Here she suggests factorizing the covariance matrix $\mathbf{P}$ into the combination $\mathbf{UDU}^T$ where $\mathbf{U}$ is a upper-triangular matrix, and $\mathbf{D}$ is a diagonal matrix. In this dissertation, she states that the U-D filter algorithm she derived both "combines the numerical precision of square root filtering techniques with an efficiency comparable to that of Kalman's original formula", and that "the method involves no more computer storage than the Kalman algorithm" [5] (p. i). Simon adds to this and states that in addition to bettering the numerical properties of the covariance matrix, that the U-D filtering should be some middle point between the faster sequential implementation, and the more numerically stable square root filtering methods [3] (p. 162 p.174). Regarding additional computational considerations, Salzmann suggests the U-D covariance factorization filter if "reduction of computational burden is of utmost importance" [6] (p. 77-78) as well as the benefit of not needing any explicit square root computations as seen in square root filter types [6] (p. 34). The issue of computational load versus accuracy and robustness becomes a delicate balancing problem when designing Kalman filter algorithms. Should the design be fast on the cost of numerical stability and accuracy, slow but robust, or somewhere in the middle. What is needed depends heavily on the application of the filter. A system with fast dynamics such as a UAV may demand both. Based on these statements, I have chosen to investigated three Error State Kalman Filter implementations. The first is the standard ESKF-implementation, presented by Solà [7] (Ch. 5), The second filter variation is the sequential variation presented by Simon [3] (p.150) which changes the measurement update to handle the update one measurement at the time. The third variation is the UDU-factorized variation as presented by D'Souza and Simon, based on Thornton [5] and *"Factorization Methods for Discrete Sequential Estimation"* by Bierman [8]. Here, the UD-variation is chosen over other alternatives such as the QR and LU factorized variation due to the advantages mentioned in the literature such as it being similar in accuracy and numerical stability as the computational demanding square root filters, while being more similar to the standard error-state Kalman filter in terms of computational load.

The end goal is to evaluate run time and to compare the accuracy of the three implementations, where both combined run time for time-update and measurement-update will be compared with individual run time for time-update and measurement update. Furthermore, the number of arithmetic operations mentioned by D'Souza will not be investigated here, but rather I will use this claim to see if the the supposedly fewer operations will result in a faster run time when performing simulations.

Due to this focus, we decided on implementing a simulation model that is able to estimate the trajectory and attitude of a simulated vessel, and to limit the states to as few as possible. An inertial measurement unit consisting of a 3-axis accelerometer and a 3-axis rate-gyro with all axis are perfectly aligned were chosen. We also assume that the dynamic and static IMU-biases can be combined into a total bias. This removes the need to add a 6 misalignment vectors for the IMU and 6 additional biases in the state space. This yields a state space consisting of $n = 15$ states in our model. The states are positions, velocities, attitudes, accelerometer bias and gyro rate bias. Furthermore, We've chosen to generate the aiding measurement through a generic range between the, i.e our simulated vessel, and a constellation of beacons scattered around the trajectory of the vessel. This aiding measurement could be obtained through a array

of alternatives, such as ultra wide band, Bluetooth, or more typically a GNSS measurement, but the beacon constellation were chosen due to how easy one may scale up the measurement vector by adding or removing beacons to the constellation. In addition it also removes the need to estimate states such as GNSS clock error as a state and to model atmospheric noises for a high fidelity GNSS signal.

Since the matrix inversion has the complexity of $\mathcal{O}(m^3)$, whereas adding an iteration to a for-loop should have complexity $\mathcal{O}(m)$, the run time increase should be squared or even cubed when increasing the measurement vector. From this, I believe there is reason to to state that the sequential update will run faster than the batch-wise update. In addition to this, I believe the difference between the elapsed run times between the variations will increase as the size of the measurement vector increases. Furthermore, I want to examine if the benefit of a faster measurement update comes with a cost of system accuracy, which I will measure with the state root mean square-error (RMSE), and a examine if the covariance matrices are larger in one of the filter variations. Both the batch-wise and sequential variations should be similar in total time update run time since they share the code base.

The design for the UD-factorized variation presented in the literature requires changes to both the time update and the measurement update through multiple factorization functions. These factorization functions adds to the computational load by adding multiple for-loops and matrix multiplications, which are listed in Section 5.4.3. The time update gets two additional nested for-loops for each time step, while the measurement update adds two nested for loop for each measurement. Thus I believe the total time update run time should be slower than time update shared between the two other variations, while the measurement update should be significant slower than the sequential variation. Comparing the UD-measurement update to the standard update, I believe these should be somewhat comparable in run time. The final point I want to examine is if we obtain a better covariance and a smaller RMSE between the UD-variation and the two other. Based on Thorntons argument that the UD-variation should result in similar results as the standard variation, I believe that the UD-factorized variation should be at least as good as the standard variation, possible better in terms of accuracy performance.

1.2 Main contributions

This master thesis uses the majority of the simulation model created in my own project thesis, *"Efficient and accurate implementation of inertial navigation systems"*[1], thus some of the contributions have their origin from there.

Following is a list of main contributions for the master thesis

- Review and validation of the results from my project thesis.
- Re-implementation of the simplified 16 state motion simulator and inertial measurement units and beacon range simulator created in my project thesis to fit the scope of this thesis and to patch errors from the prior experiment.
- Re-implementation of a standard Error-State Kalman Filter and recreate the true states from the simulation data through the ESKF estimation algorithm to patch out errors.
- Implementation of a sequential inertial navigation system aiding strategy for the ESKF.
- Implementation of a factorized inertial navigation system aiding strategy for the ESKF.
- Scripting a standardized method of bench marking and measure the filter variations.
- Evaluate performance and run time of the three aided inertial navigation system strategies.

The code used in the master thesis can be found in this GitHub repository [9]. The code for the project thesis can be found in this repository [10].

1.3 Thesis organization

In addition to the introduction and problem background the thesis is structured into the following chapters and parts:

- **Mathematical preliminaries**: A brief presentation of mathematical preliminaries used throughout this thesis.
- **Inertial Navigation System preliminaries**: Contains a presentation of the Inertial Measurement Units (IMU) and the measurement equations used in the Inertial Navigation System (INS). This section also contains the IMU process noise and bias design and how this noise design were used to tune the error-state Kalman filter. Lastly, this chapter contains the system dynamics equations and the system and derives the relation between the measurement units, true kinematics, nominal kinematics and the error-state kinematics as well as the discretization used for these equations in the simulation model.
- **Simulation model**: In this chapter I present how the simulation model and how the simulated measurement data were generated. This chapter is largely kept the same as in the project thesis. Some changes have been done regarding data-generation and the placement of the beacons, but the model is otherwise kept similar in terms of the equations used.
- **Error State Kalman & Aided INS designs**: Contains the equations used in the time and measurement updates for the batch-wise reference ESKF, the sequential ESKF and the UD-factorized ESKF. Section 5.4 contains the algorithm variations and displays the flow in the filter variations.
- **Results**: Chapter 6 the results and the evaluation of the results for m = 15 and m = 30 respectively. The filter variation and module run times are here presented in scatter plot and table form, and the relative speed difference between the variations can be seen in table form for the chosen set of simulation durations. The corresponding root mean square errors for the error states as well as the final estimated covariance is also found here.
- **Conclusion and further work**: Contains the summarized conclusion from the findings and the discussion as well as a suggestion for further work.

Some chapters in this thesis report is based on the theory used in the project thesis and are thus similar in content. Following is a list of the chapters that share the most similarities between the two thesis'. Other sections may have similarities, but are not listed here as they largely have been rewritten.

- Parts of the Inertial Navigation System preliminaries in Chapter 3.
- The derivation of the equations of the system dynamics and the equations for the reference Error-State Kalman Filter in chapter Section 3.3 and Section 5.1.
- The design of the reference error-state Kalman filter.
- The derivation of the equations used in the simulation model in Chapter 4

Chapter 2

Mathematical preliminaries

This section contains some of the general mathematical preliminaries making it easier to follow the equations written in the upcoming sections.

2.1 Rotation matrix

The rotation matrix between two frames a and b is denoted as $\boldsymbol{R}_b^a$. The rotation matrix is an element in $SO(3)$ and has the properties of that special orthogonal group of order 3 for which

$$SO(3) = \left\{\boldsymbol{R} | \boldsymbol{R} \in \mathbb{R}^{3\times 3}, \boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{I}_3\right\} \tag{2.1.1a}$$

$$\det(\boldsymbol{R}) = 1 \tag{2.1.1b}$$

In this report, we adapt the following notation

$$\boldsymbol{v}^{to} = \boldsymbol{R}_{from}^{to}\boldsymbol{v}^{from} \tag{2.1.2}$$

Here, the subscript denotes which frame we rotate from, and the superscript denotes which frame we rotate to for the rotation matrix. For the vectors, the superscript denotes which frame the corresponding vector or variable is in.

The euler angle transformation is given as

$$\boldsymbol{R}_b^n = \boldsymbol{R}(\boldsymbol{\Theta}_{nb}) \tag{2.1.3}$$

where

$$\boldsymbol{\Theta} = \begin{bmatrix}\phi & \theta & \psi\end{bmatrix}^T \tag{2.1.4}$$

The one-axis rotations are given by

$$\boldsymbol{R}_{x,\phi} = \begin{bmatrix}1 & 0 & 0\\ 0 & c\phi & -s\phi\\ 0 & s\phi & c\phi\end{bmatrix}, \boldsymbol{R}_{y,\theta} = \begin{bmatrix}c\theta & 0 & s\theta\\ 0 & 1 & 0\\ -s\theta & 0 & c\theta\end{bmatrix}, \boldsymbol{R}_{z,\psi} = \begin{bmatrix}c\psi & -s\psi & 0\\ s\psi & c\psi & 0\\ 0 & 0 & 1\end{bmatrix} \tag{2.1.5}$$

where s($\cdot$) = sin($\cdot$) and c($\cdot$) = cos($\cdot$). The combined, full rotation from the body frame to NED frame is given by multiplying these three one-axis rotations as seen in Equation (2.1.6)

$$\boldsymbol{R}_b^n = \boldsymbol{R}_{z,\psi}\boldsymbol{R}_{y,\theta}\boldsymbol{R}_{x,\phi} = \begin{bmatrix}c\psi c\theta & -s\psi c\psi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\phi s\theta\\ s\psi c\theta & c\psi c\phi + s\phi s\theta s\psi & -c\psi s\phi + s\theta s\psi\phi\\ -s\theta & c\theta s\phi & c\theta c\phi\end{bmatrix} \tag{2.1.6}$$

2.2 Skew symmetric matrix

The skew symmetric matrix $\boldsymbol{S}$ is a matrix of order n which holds the property

$$\boldsymbol{S} = -\boldsymbol{S}^T$$

and is defined as

$$\boldsymbol{S}(\boldsymbol{\lambda}) = -\boldsymbol{S}^T(\boldsymbol{\lambda}) = \begin{bmatrix} 0 & -\lambda_3 & \lambda_2 \\ \lambda_3 & 0 & -\lambda_1 \\ -\lambda_2 & \lambda_1 & 0 \end{bmatrix} \tag{2.2.1}$$

where

$$\boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 \end{bmatrix}^T$$

2.3 Quaternions

An alternative attitude representation to the Euler angle representation is the quaternion representation. A unit quaternion is defined as a complex number with one real part η, and three imaginary parts $\}m$ and is given by

$$\boldsymbol{q} = \begin{bmatrix} \eta & \epsilon_1 & \epsilon_2 & \epsilon_3 \end{bmatrix}^T \tag{2.3.1}$$

The unit quaternion satisfies $\boldsymbol{q}^T\boldsymbol{q} = 1$.

The product of two quaternions is defined as

$$\boldsymbol{q}_1 \otimes \boldsymbol{q}_2 = \begin{bmatrix} \eta_1\eta_2 - \epsilon_1^T\epsilon_2 \\ \eta_1\epsilon_2 + \eta_{21} + \boldsymbol{S}(\epsilon_1)\epsilon 2 \end{bmatrix} \tag{2.3.2}$$

Unit quaternion rotation matrix

The unit quaternion rotation matrix from BODY to NED is given by

$$\begin{aligned} \boldsymbol{R}(\boldsymbol{q}_b^n) &= \boldsymbol{I}_3 + 2\eta\boldsymbol{S}(\epsilon) + 2\boldsymbol{S}^2(\epsilon) \\ &= \begin{bmatrix} 1 - 2(\epsilon_2^2 + \epsilon_3^2) & 2(\epsilon_1\epsilon_2 - \epsilon_3\eta) & 2(\epsilon_1\epsilon_3 + \epsilon_2\eta) \\ 2(\epsilon_1\epsilon_2 + \epsilon_3\eta) & 1 - 2(\epsilon_1^2 + \epsilon_3^2) & 2(\epsilon_2\epsilon_3 - \epsilon_1\eta) \\ 2(\epsilon_1\epsilon_3 - \epsilon_2\eta) & 2(\epsilon_2\epsilon_3 + \epsilon_1\eta) & 1 - 2(\epsilon_1^2 + \epsilon_2^2) \end{bmatrix} \end{aligned} \tag{2.3.3}$$

2.4 Euclidian norm

The euclidian norm is defined as

$$||\boldsymbol{x}||_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \tag{2.4.1}$$

2.5 Noise processes

In both the following subsections b denotes the bias state, p denotes a static component, and w denotes a dynamic noise component.

2.5.1 Gauss-Markov

A Gauss-Markov model is given by the equation

$$\dot{b} = -pb + w \tag{2.5.1}$$

where

$$w \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma^2 \delta t - \tau})$$

2.5.2 Wiener

A wiener process is given by

$$\dot{b} = w \tag{2.5.2}$$

where

$$w \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma^2 \delta t - \tau})$$

Chapter 3

INS preliminaries

The inertial navigation system is a system consisting of an inertial measurement unit (IMU) and a computational unit. The IMU is typically made up of a 3-axis accelerometer, a 3-axis gyroscope and in some cases further augmented with a 3-axis magnetometer. Both the accelerometer and the gyroscope are mounted orthogonally to each other, and allows the user to obtain specific forces and angular rates in six degrees of freedom (6-DOF).

In the following sections, I will use the notation $(.)_m$ to denote the *measured signal*, $(.)_t$ is the *true signal*, $(.)_n$ as the *measurement noise signal*, and $(.)_{bt}$ is the *bias term*. This notation is adapted from Solà[2] and Brekke [11], with the addition of frame notation superscipts $(.)^b$ or $(.)^n$ denoting if the object at hand is written in the NED-frame or the BODY-frame. In addition to this, the rotation $\boldsymbol{R}_b^n(\boldsymbol{q}_b^n)$ signifies a rotation from frame b to frame n, similar to what is presented in (2.1.2).

3.1 Accelerometer and rate gyro sensor

In a general case, the rate sensor can be defined in any measurement frame, and the accelerometer can be defined in any arbitrary frame, measuring the specific force, i.e the acceleration relative to the gravitational acceleration. In this thesis, both the accelerometer and rate sensor are modeled in the body-frame.

The measurement equations for the IMU accelerometer and gyro rate sensor is given by Fossen in [12] (p. 420). The accelerometer measures three-axis specific force, which is a non-gravitational force per unit mass m, and is denoted as

$$\boldsymbol{f}_{acc}^b = \boldsymbol{f}_{nm_I}^b \tag{3.1.1}$$

where $\boldsymbol{g}^b$ is the body fixed gravitational force, and $\boldsymbol{f}_{nm_I}^b$ is the total force. Subtracting the gravitational forces, we're left with the formula for specific force

$$\boldsymbol{f}_{acc}^b = \boldsymbol{a}_{nm_I}^b - \boldsymbol{g}^b \tag{3.1.2}$$

From (3.1.2) we get the measurement equation for a three axis accelerometer with the corresponding measurement bias equation given as

$$\boldsymbol{a}_m^b = \boldsymbol{R}_n^b(\boldsymbol{q}_b^n)(\boldsymbol{a}_t^n - \boldsymbol{g}_t^n) + \boldsymbol{a}_{bt}^b + \boldsymbol{a}_n^b \tag{3.1.3a}$$

$$\boldsymbol{a}_{bt}^b = -p_{\text{ab}}\boldsymbol{I}a_{bt}^b + a_w^b \tag{3.1.3b}$$

From Fossen we also obtain three axis rate gyro sensor measurement equation with the corresponding measurement bias equation given as

$$\boldsymbol{\omega}_m^b = \boldsymbol{\omega}_t^b + \boldsymbol{\omega}_{bt}^b + \boldsymbol{\omega}_n^b \tag{3.1.4a}$$

$$\dot{\boldsymbol{\omega}}_{bt}^b = -p_{\omega b}\boldsymbol{I}\boldsymbol{\omega}_{bt}^b + \boldsymbol{\omega}_w^b \tag{3.1.4b}$$

Here, both IMU measurement bias equations differ from Fossen where we model the biases $\boldsymbol{a}_{bt}^b$ and $\dot{\omega}_{bt}^b$ as slowly varying Gauss-Markov biases processes instead of Wiener processes. The biases will continue to grow during operation, thus it is necessary to estimate them for feedback purposes, which is done in the ESKF.

In addition to this, Fossen states that the gyro rate sensor equation given in (3.1.4) is only valid for *"low-speed applications such as a marine craft moving on the surface of the Earth since it assumes that $\{n\}$ is nonrotating, that is $\boldsymbol{\omega}_{ib}^b \approx \boldsymbol{\omega}_{nb}^b$. For terrestrial navigation, the Earth rotation will affect the results and it is necessary to use the inertial frame $\{i\}$ instead of the approximate frame $\{n\}$."*. Thus, this thesis uses the assumption that application we are modeling is a low-speed application.

3.2 IMU process noise

This section introduces the process noise vector $\boldsymbol{n}$ which will be more closely derived in (3.3.10). It consists of the continuous-time represented IMU input noises $\boldsymbol{a}_n^b$ and $\boldsymbol{\omega}_n^b$, and the driving noises of the bias processes $\boldsymbol{a}_w^b$ and $\boldsymbol{\omega}_w^b$. Their continuous-time spectral density matrices are denoted $\tilde{\boldsymbol{V}}, \tilde{\boldsymbol{\Theta}}, \tilde{\boldsymbol{W}}$ and $\tilde{\boldsymbol{\Omega}}$, all assumed to be white processes:

$$\boldsymbol{n} \sim \mathcal{N}(\boldsymbol{0}_{12x1}, \tilde{\boldsymbol{Q}}), \quad \tilde{\boldsymbol{Q}} = \text{blkdiag}(\tilde{\boldsymbol{V}}, \tilde{\boldsymbol{\Theta}}, \tilde{\boldsymbol{W}}, \tilde{\boldsymbol{\Omega}}) \tag{3.2.1}$$

Solà states that the noise process are different in nature, which in turn affects how their covariance should be tuned. Here, we base the velocity and attitude prediction equations of a numerical integration of the discrete measurements equations from the IMU, derived from (3.1.3) and (3.1.4). This also comes with a integration of the IMU measurement noise. Thus, by knowing the noise characteristics of the discrete time IMU measurements, we can say something about the power spectral density of the continuous-time IMU measurement noises $\boldsymbol{a}_n^b$ and $\boldsymbol{\omega}_n^b$, or as Solà states *noise levels in the control signals*. The IMU measurement noises $\boldsymbol{a}_n^b$ and $\boldsymbol{\omega}_n^b$ are modeled as additive white noise processes assumed Gaussian. Their continuous time covariances are denoted by $\boldsymbol{V}$ and $\boldsymbol{\Theta}$ and are given in (3.2.2).

$$\boldsymbol{a}_n^b \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{V}\boldsymbol{\delta t} - \boldsymbol{\tau}) \tag{3.2.2a}$$

$$\boldsymbol{\omega}_n^b \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Theta}\boldsymbol{\delta t} - \boldsymbol{\tau}) \tag{3.2.2b}$$

The STIM300 were the IMU of choice for this simulation model. This IMU were chosen to design both the noise in the IMU data generated in the simulation model, as well as the the

parameters for the tuning of the ESKF. The IMU specifications seen in Table 3.2.1 contains the bias instability and random walk parameters for the Sensonor STIM300 [13]. The units in this table is $^\circ$ = *degrees*, h = *hour*, m/s = *meter per second*, g = *gravitational acceleration.*

Table 3.2.1: *Data sheet parameters from STIM300 [13]*

Gyro angle random walk (ARW)	0.15 $\left[^\circ/\sqrt{\text{h}}\right]$
Gyro bias instability	0.3 [°/h]
Accelerometer velocity random walk (VRW)	0.07 $\left[\frac{\text{m/s}}{\sqrt{\text{h}}}\right]$
Accelerometer bias instability	0.05 $\left[g \cdot 10^{-3}\right]$

(3.2.3) - (3.2.6) displays the derivation for finding the standard deviation of the gyro random walk. (3.2.7) and (3.2.8) displays the conversion to SI-units and values used in the tuning of the Kalman filter, and the spectral densities of the bias driving IMU noises can be seen in (3.2.9). The equations used are from [14].

First we define the angular random walk as ARW = $\boldsymbol{\delta\Theta}$, and

$$\boldsymbol{\delta\dot{\Theta}} = \boldsymbol{\omega}_n^b \tag{3.2.3a}$$

$$\mathbb{E}\left[\boldsymbol{\omega}_n^b(t)\boldsymbol{\omega}_n^{bT}(\tau)\right] = \boldsymbol{Q}\boldsymbol{\delta}(t-\tau) \tag{3.2.3b}$$

The next step is to compute the variance $\mathbb{E}\left[\boldsymbol{\Theta}(t)\boldsymbol{\Theta}^T(\tau)\right]$

$$\begin{aligned}
\mathbb{E}\left[\boldsymbol{\Theta}(t)\boldsymbol{\Theta}^T(\tau)\right] &= \mathbb{E}\left[\int_0^t \boldsymbol{\omega}_n^b(\beta)d\beta \int_0^t \boldsymbol{\omega}_n^{bT}(\tau)d\tau\right] \\
&= \int_0^t \int_0^t \mathbb{E}\left[\boldsymbol{\omega}_n^b(\beta)\boldsymbol{\omega}_n^{bT}(\tau)\right] d\beta d\tau \\
&= \int_0^t \int_0^t \boldsymbol{Q}\delta(\beta-\tau)d\beta d\tau \\
&= \int_0^t \boldsymbol{Q}d\tau \\
&= t \cdot \boldsymbol{Q}
\end{aligned} \tag{3.2.4}$$

From here, we can obtain the standard deviation by taking the square root of the variance

$$\boldsymbol{\sigma}_{\delta\boldsymbol{\Theta}} = \sqrt{\mathbb{E}\left[\boldsymbol{\Theta}(t)\boldsymbol{\Theta}^T(\tau)\right]} = \sqrt{t} \cdot \sqrt{\boldsymbol{Q}} \tag{3.2.5}$$

where we have the standard deviation for the gyro measurement noise as

$$\boldsymbol{\sigma}_{\omega_n^b} = \sqrt{\boldsymbol{Q}} = \frac{\boldsymbol{\sigma}_{\delta\boldsymbol{\Theta}}}{\sqrt{t}} = \frac{\boldsymbol{\sigma}_{\text{ARW}}}{\sqrt{t}}$$

(3.2.6)

The next step is to convert this sigma-value to SI-units before we can use them to tune our error-state Kalman filter. By inserting the angular random walk from Table 3.2.1, and letting t = h = 3600 seconds, and converting to radians from degrees, we obtain $\boldsymbol{\sigma}_{\boldsymbol{\omega}_n^b}$ as

$$\begin{aligned}\boldsymbol{\sigma}_{\boldsymbol{\omega}_n^b} &= 0.15 \left[\frac{\circ}{\sqrt{\text{h}}}\right] \\ &= 0.15 \left[\frac{\circ}{\sqrt{3600\text{s}}}\right] \\ &= \frac{0.15}{60} \left[\frac{\circ}{\sqrt{\text{s}}}\right] \\ &= \frac{0.15\pi}{60 \cdot 180} \left[\frac{\text{rad}}{\sqrt{\text{s}}}\right]\end{aligned} \tag{3.2.7}$$

In a similar way we obtain the standard deviation of the accelerometer measurement noise, $\boldsymbol{\sigma}_{\boldsymbol{a}_n^b}$ as

$$\begin{aligned}\boldsymbol{\sigma}_{\boldsymbol{a}_n^b} &= \boldsymbol{\sigma}_{\text{VRW}}/\sqrt{t} = 0.07 \left[\frac{\text{m/s}}{\sqrt{\text{h}}}\right] \\ &= 0.07 \left[\frac{\text{m/s}}{\sqrt{3600\text{s}}}\right] \\ &= \frac{0.07}{60} \left[\frac{\text{m/s}}{\sqrt{\text{s}}}\right]\end{aligned} \tag{3.2.8}$$

The spectral density of the bias driving IMU noises $\boldsymbol{a}_w^b$ and $\boldsymbol{\omega}_w^b$ were designed to match the standard deviation of the simulated bias noise, which can be seen in section Chapter 4 and were set to

$$\tilde{\boldsymbol{W}} = 0.004^2 \quad \frac{m}{s^2} \tag{3.2.9a}$$

$$\tilde{\boldsymbol{\Omega}} = 0.00005^2 \quad \frac{\text{rad}}{s} \tag{3.2.9b}$$

Where the respective standard deviations can be found by taking the square root of (3.2.9).

3.3 System dynamics

The variables and their definitions used in the strapdown equations and in the ESKF are defined by Before moving on to the system equations, Joan Solà[2] can be seen in Table 3.3.1.

The equations for the ESKF are separated into continuous time equations seen in Section 3.3.2 and the discretized equations in section Section 3.3.3.

3.3.1 Continuous time system kinematics

The continuous time system kinematics are divided into true-state, nominal state and error-state kinematics and are shown in the following sections.

Table 3.3.1: *Elements in Solà's ESKF formulation*

Magnitude	True	Nominal	Error	Composition	Measured	Noise
Position	$\mathbf{p}_t$	$\mathbf{p}$	$\boldsymbol{\delta p}_t$	$\boldsymbol{p_t} = \boldsymbol{p} + \boldsymbol{\delta p}$		
Velocity	$\mathbf{v}_t$	$\mathbf{v}$	$\boldsymbol{\delta v}_t$	$\boldsymbol{v_t} = \boldsymbol{v} + \boldsymbol{\delta v}$		
Orientation	$\mathbf{q}_t$	$\mathbf{q}$	$\boldsymbol{\delta q}_t$	$\boldsymbol{q_t} = \boldsymbol{q} \otimes \boldsymbol{\delta q}$		
3 DOF attitude error			$\boldsymbol{\delta\theta}$	$\boldsymbol{\delta q} \approx \begin{bmatrix} 1 \\ \frac{\boldsymbol{\delta\theta}}{2} \end{bmatrix}$		
Accelerometer bias	$\mathbf{a}_{bt}$	$\mathbf{a}_b$	$\boldsymbol{\delta a}_b$	$\boldsymbol{a_{bt}} = \boldsymbol{a}_b + \boldsymbol{\delta a}_b$		$\boldsymbol{a}_w$
Gyro bias	$\boldsymbol{\omega}_{bt}$	$\boldsymbol{\omega}_b$	$\boldsymbol{\delta\omega}_b$	$\boldsymbol{\omega_{bt}} = \boldsymbol{\omega}_b + \boldsymbol{\delta\omega}_b$		$\boldsymbol{\omega}_w$
Acceleration	$\mathbf{a}_t$				$\mathbf{a}_m$	$\mathbf{a}_n$
Angular rate	$\boldsymbol{\omega}_t$				$\boldsymbol{\omega}_m$	$\boldsymbol{\omega}_n$

3.3.2 True-state kinematics

The true kinematic equations containing the model terms are given by Solà and Brekke as seen in (3.3.1).

$$\dot{\boldsymbol{p}}_t^n = \boldsymbol{v}_t^n \tag{3.3.1a}$$
$$\dot{\boldsymbol{v}}_t^n = \boldsymbol{a}_t^n \tag{3.3.1b}$$
$$\dot{\boldsymbol{q}}_{t,b}^n = \frac{1}{2}\boldsymbol{q}_{t,b}^n \otimes \boldsymbol{\omega}_t^b \tag{3.3.1c}$$
$$\dot{\boldsymbol{a}}_{bt}^b = -p_{ab}\boldsymbol{I}\boldsymbol{a}_{bt}^b + \boldsymbol{a}_w^b \tag{3.3.1d}$$
$$\dot{\boldsymbol{\omega}}_{bt}^b = -p_{\omega b}\boldsymbol{I}\boldsymbol{\omega}_{bt}^b + \boldsymbol{\omega}_w^b \tag{3.3.1e}$$

The true IMU biases are modeled as first order Gauss Markov processes with p_{ab} and $p_{\omega b}$, being the inverse time constants.

By substituting the true terms in the velocity and attitude equations with the IMU measurements given by (3.1.3) and (3.1.4), we rewrite (3.3.1) to (3.3.2):

$$\dot{\boldsymbol{p}}_t^n = \boldsymbol{v}_t^n \tag{3.3.2a}$$
$$\dot{\boldsymbol{v}}_t^n = \boldsymbol{R}_{t,b}^n(\boldsymbol{q}_{t,b}^n)(\boldsymbol{a}_m^b - \boldsymbol{a}_{bt}^b - \boldsymbol{a}_n^b) + \boldsymbol{g}_t^n \tag{3.3.2b}$$
$$\dot{\boldsymbol{q}}_{t,b}^n = \frac{1}{2}\boldsymbol{q}_{t,b}^n \otimes (\boldsymbol{\omega}_m^b - \boldsymbol{\omega}_{bt}^b - \boldsymbol{\omega}_n^b) \tag{3.3.2c}$$
$$\dot{\boldsymbol{a}}_{bt}^b = -p_{ab}\boldsymbol{I}\boldsymbol{a}_{bt}^b + \boldsymbol{a}_w^b \tag{3.3.2d}$$
$$\dot{\boldsymbol{\omega}}_{bt}^b = -p_{\omega b}\boldsymbol{I}\boldsymbol{\omega}_{bt}^b + \boldsymbol{\omega}_w^b \tag{3.3.2e}$$

Which can be be written as the system $\dot{\boldsymbol{x}}_t = f_t(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w})$. The system with states, are driven by input $\boldsymbol{u}$ and white Gaussian noise $\boldsymbol{w}$ are listed in (3.3.3).

$$\boldsymbol{x}_t = \begin{bmatrix} \boldsymbol{p}_t \\ \boldsymbol{v}_t \\ \boldsymbol{q}_t \\ \boldsymbol{a}_{bt} \\ \boldsymbol{\omega}_{bt} \end{bmatrix}, \qquad \boldsymbol{u} = \begin{bmatrix} \boldsymbol{a}_m - \boldsymbol{a}_n \\ \boldsymbol{\omega}_m - \boldsymbol{\omega}_n \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} \boldsymbol{a}_w \\ \boldsymbol{\omega}_w \end{bmatrix} \tag{3.3.3}$$

Nominal-state kinematics

The nominal-state kinematics corresponds to the modeled system without noises or perturbations, and is given as:

$$\dot{\boldsymbol{p}}^n = \boldsymbol{v}^n \tag{3.3.4a}$$
$$\dot{\boldsymbol{v}}^n = \boldsymbol{R}_b^n(\boldsymbol{q}_b^n)(\boldsymbol{a}_m^b - \boldsymbol{a}_b^b) + \boldsymbol{g}^n \tag{3.3.4b}$$
$$\dot{\boldsymbol{q}}_b^n = \frac{1}{2}\boldsymbol{q}_b^n \otimes (\boldsymbol{\omega}_m^b - \boldsymbol{\omega}_b^b) \tag{3.3.4c}$$
$$\dot{\boldsymbol{a}}_{bt}^b = -p_{ab}\boldsymbol{I}\boldsymbol{a}_{bt}^b \tag{3.3.4d}$$
$$\dot{\boldsymbol{\omega}}_{bt}^b = -p_{\omega b}\boldsymbol{I}\boldsymbol{\omega}_{bt}^b \tag{3.3.4e}$$

Error-state kinematics

Since the underlying system for the error-state kinematics are nonlinear, it is required to approximate the system to obtain a linear error state model. In (3.3.5), Brekke presents the approximation used by Solà by means of the first-order approximation expressed as follows.

$$\dot{\delta\boldsymbol{p}}^n = \delta\boldsymbol{v}_t^n \tag{3.3.5a}$$
$$\dot{\delta\boldsymbol{v}}^n = -\boldsymbol{R}_b^n(\boldsymbol{q}_b^n)S(\boldsymbol{a}_m^b - \boldsymbol{a}_b^b)\delta\boldsymbol{\theta} - \boldsymbol{R}_b^n(\boldsymbol{q}_b^n)\delta\boldsymbol{a}_b^b - \boldsymbol{R}_b^n(\boldsymbol{q}_b^n)\boldsymbol{a}_n^b \tag{3.3.5b}$$
$$\dot{\delta\boldsymbol{\theta}}^b = -S(\boldsymbol{\omega}_m^b - \boldsymbol{\omega}_b^b)\delta\boldsymbol{\theta} - \delta\boldsymbol{\omega}_b^b - \boldsymbol{\omega}_n^b \tag{3.3.5c}$$
$$\dot{\delta\boldsymbol{a}}_b^b = -p_{ab}\boldsymbol{I}\delta\boldsymbol{a}_{bt}^b + \boldsymbol{a}_w^b \tag{3.3.5d}$$
$$\dot{\delta\boldsymbol{\omega}}_b^b = -p_{\omega b}\boldsymbol{I}\delta\boldsymbol{\omega}_{bt}^b + \boldsymbol{\omega}_w^b \tag{3.3.5e}$$
$$\tag{3.3.5f}$$

The proofs for linear velocity error and orientation error are mentioned in chapter 5.3.3 (page 55-57) in Solà [2], and in Brekke [11] (page 180-181).

3.3.3 Discrete time system equations

The continuous time differential equations given prior needs to be discretized to be put to use. The only predicted state vector is the nominal states, while the error state is used for prediction of the covariance.

The nominal state kinematics

Integration of the nominal state kinematics are done through integration based on Taylor expansion, zero-order-hold and Euler integration, and yields the equations listed in (3.3.6).

In addition to the subscripts seen in table 3.3.1, and frame-superscripts $(.)^{b/n}$, we now introduce the discrete time step notation $(.)_{x,k}^x$, where $[k]$ denotes the state at time step k, and x denotes the subscript and frame superscript as formerly described in 3.3.1.

$$\boldsymbol{p}_{k+1}^n = \boldsymbol{p}_k^n + \boldsymbol{v}_k^n \Delta t + \frac{1}{2}(\boldsymbol{R}_b^n(\boldsymbol{q}_{b,k}^n)(\boldsymbol{a}_{m,k}^b - \boldsymbol{a}_{b,k}^b) + \boldsymbol{g}^n)\Delta t^2 \tag{3.3.6a}$$

$$\boldsymbol{v}_{k+1}^n = \boldsymbol{v}_k^n + (\boldsymbol{R}_b^n(\boldsymbol{q}_{b,k}^n)(\boldsymbol{a}_{m,k}^b - \boldsymbol{a}_{b,k}^b) + \boldsymbol{g}^n)\Delta t \tag{3.3.6b}$$

$$\boldsymbol{q}_{b,k+1}^n = \boldsymbol{q}_{b,k}^n \otimes \boldsymbol{q}_k\{(\boldsymbol{\omega}_{m,k}^b - \boldsymbol{\omega}_{b,k}^b)\Delta t = \boldsymbol{q}_{b,k}^n \otimes \Delta \boldsymbol{q}_k \tag{3.3.6c}$$

$$\boldsymbol{a}_{b,k+1}^b = (1 - \Delta t p_{ab}\boldsymbol{I})\boldsymbol{a}_{bt,k}^b \tag{3.3.6d}$$

$$\boldsymbol{\omega}_{b,k+1}^b = (1 - \Delta t p_{\omega b}\boldsymbol{I})\boldsymbol{\omega}_{bt,k}^b \tag{3.3.6e}$$

The right hand term in the discretized quaternion (3.3.6d), is defined as

$$\Delta q* = \begin{bmatrix} \cos(\frac{\Delta\bar{\Theta}}{2}) \\ \sin(\frac{\Delta\bar{\Theta}}{2})\frac{\Delta\bar{\Theta}}{\Delta\boldsymbol{\Theta}} \end{bmatrix} \tag{3.3.7}$$

For the case where ω goes toward zero, or close to numerical zero for the compiler, then the imaginary part of $\Delta\boldsymbol{q}*$ goes towards the unit quaternion.

The predicted quaternion $\boldsymbol{q}_b^n = \boldsymbol{q}_b^n \otimes \Delta\boldsymbol{q}_b^n$ is then normalized by dividing on the 2 norm of the predicted quaternion

$$\bar{\boldsymbol{q}}_b^n = \frac{\boldsymbol{q}_b^n}{||\boldsymbol{q}_b^n||_2} \tag{3.3.8}$$

The discrete nominal state $\boldsymbol{x}_k$ is propagated as

$$\boldsymbol{x}_k^- = \boldsymbol{\Phi}_{k-1}\boldsymbol{x}_{k-1}^+ \tag{3.3.9}$$

.

All-though, the state propagation is not necessary since the $\boldsymbol{x}^+$ is always zero in a error-state filter since the states are propagated outside the ESKF.

The error-state kinematics

Similar to the true state system given in (3.3.3), we can write the nominal and error state system on compact form as

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{v} \\ \boldsymbol{q} \\ \boldsymbol{a}_b \\ \boldsymbol{\omega}_b \end{bmatrix}, \qquad \boldsymbol{\delta x} = \begin{bmatrix} \boldsymbol{\delta p} \\ \boldsymbol{\delta v} \\ \boldsymbol{\delta\theta} \\ \boldsymbol{\delta a}_b \\ \boldsymbol{\delta\omega}_b \end{bmatrix}, \qquad \boldsymbol{u}_m = \begin{bmatrix} \boldsymbol{a}_m \\ \boldsymbol{\omega}_m \end{bmatrix}, \qquad \boldsymbol{n} = \begin{bmatrix} \boldsymbol{a}_n^b \\ \boldsymbol{\omega}_n^b \\ \boldsymbol{a}_w^b \\ \boldsymbol{\omega}_w^b \end{bmatrix} \tag{3.3.10}$$

The error state kinematics given in (3.3.5) are approximated to a linear time-varying (LTV) system. Brekke explains this due to the δ-terms always occurring as a vector in a matrix-vector product, where the matrices depend on the nominal state vector through $\boldsymbol{q}$ and $\boldsymbol{\omega}_b^b$. This gives us the error state system on the form seen in (3.3.11), where denotes the error state transition matrix, and $\boldsymbol{G}$ denotes the system error input matrix.

$$\begin{aligned} \delta\boldsymbol{x} &\leftarrow f(\boldsymbol{x}, \delta\boldsymbol{x}, \boldsymbol{u}_m, \boldsymbol{n}) \\ &= \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{u}_m)\delta\boldsymbol{x} + \boldsymbol{G}(\boldsymbol{x})\boldsymbol{n} \end{aligned} \tag{3.3.11}$$

Thus (3.3.5) can be rewritten as the continuous LTV system given in (3.3.12)

$$
\begin{aligned}
\dot{\boldsymbol{\delta x}} &= \boldsymbol{F}(\boldsymbol{x})\boldsymbol{\delta x} + \boldsymbol{G}(\boldsymbol{x})n \\
&= \begin{bmatrix}
\mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & -\boldsymbol{R}_b^n(\boldsymbol{q}_b^n)S(\boldsymbol{a}_m^b - \boldsymbol{a}_b^b) & -\boldsymbol{R}_b^n(\boldsymbol{q}_b^n) & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & -S(\boldsymbol{\omega}_m^b - \boldsymbol{\omega}_b^b) & \mathbf{0} & -\boldsymbol{I} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & -p_{ab}\boldsymbol{I} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & -p_{\omega b}\boldsymbol{I}
\end{bmatrix} \boldsymbol{\delta x} + \begin{bmatrix}
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
-\boldsymbol{R}_b^n(\boldsymbol{q}_b^n) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & -\boldsymbol{I} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I}
\end{bmatrix} \boldsymbol{n}
\end{aligned}
\tag{3.3.12}
$$

Chapter 4

Simulation model

I made a simulation model to test and verify the two implementations of the Error-State Kalman Filter. The model contains a set of simulated true states, simulated IMU measurements with bias and noise, simulated GNSS measurements and beacon location over a given set of time.

4.1 Simulation model design

1. **True path, velocity, acceleration and beacon locations**
   Both to ensure that the accelerometer input was a rich enough signal, and to verify that the filter were able to handle various manouvers, a path starting with a straight line, before entering into an eight-figure was designed through the parameterization given in (4.1.1)

$$\begin{bmatrix} x^n \\ y^n \\ z^n \end{bmatrix} = \begin{bmatrix} 10 \\ 2t \\ 1 \end{bmatrix} \quad \text{for t} \leq 20s, \quad \text{and} = \begin{bmatrix} r\cos(\omega t) \\ \frac{1}{10}r^2\sin(2\omega t) + 40 \\ r\omega\cos(\omega t) \end{bmatrix} \quad \text{for t} > 20s \tag{4.1.1a}$$

$$\begin{bmatrix} u^n \\ v^n \\ w^n \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix} \quad \text{for t} \leq 20s, \quad \text{and} = \begin{bmatrix} -r\omega\sin(\omega t) \\ \frac{1}{5}r^2\omega\cos(2\omega t) \\ -r\omega^2\sin(\omega t) \end{bmatrix} \quad \text{for t} > 20s \tag{4.1.1b}$$

$$\begin{bmatrix} \dot{u}^n \\ \dot{v}^n \\ \dot{w}^n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{for t} \leq 20s, \quad \text{and} = \begin{bmatrix} -r\omega^2\cos(\omega t) \\ \frac{2}{5}r^2\omega^2\sin(2\omega t) \\ r\omega^3\cos(\omega t) \end{bmatrix} \quad \text{for t} > 20s \tag{4.1.1c}$$

   The acceleration in (4.1.1c), is the two times time differentiated true trajector. This acceleration is fed into the accelerometer as the term a_t in (3.1.3) and (3.1.4).

   Fifteen beacons were also scattered around the trajectory of the vessel. This design choice were made to ensure a desirable geometry at all time, where the beacons always have a wide angle relative to at least some of the beacons. If the vessel were to move outside the beacon configuration, would worsen this accuracy and lead to a increasingly worse geometry as the vessels distance increases from the beacon constellation, due to the angle between the beacon configuration and vessel becomes more narrow. An additional 15 beacons were added to increase the measurement vector to obtain the results in section 6.2 where the timing experiment were performed with m = 30 measurements.

   The path plot and 3D-path with beacons can be seen in figure Figure 4.1.1 and Figure 4.1.2

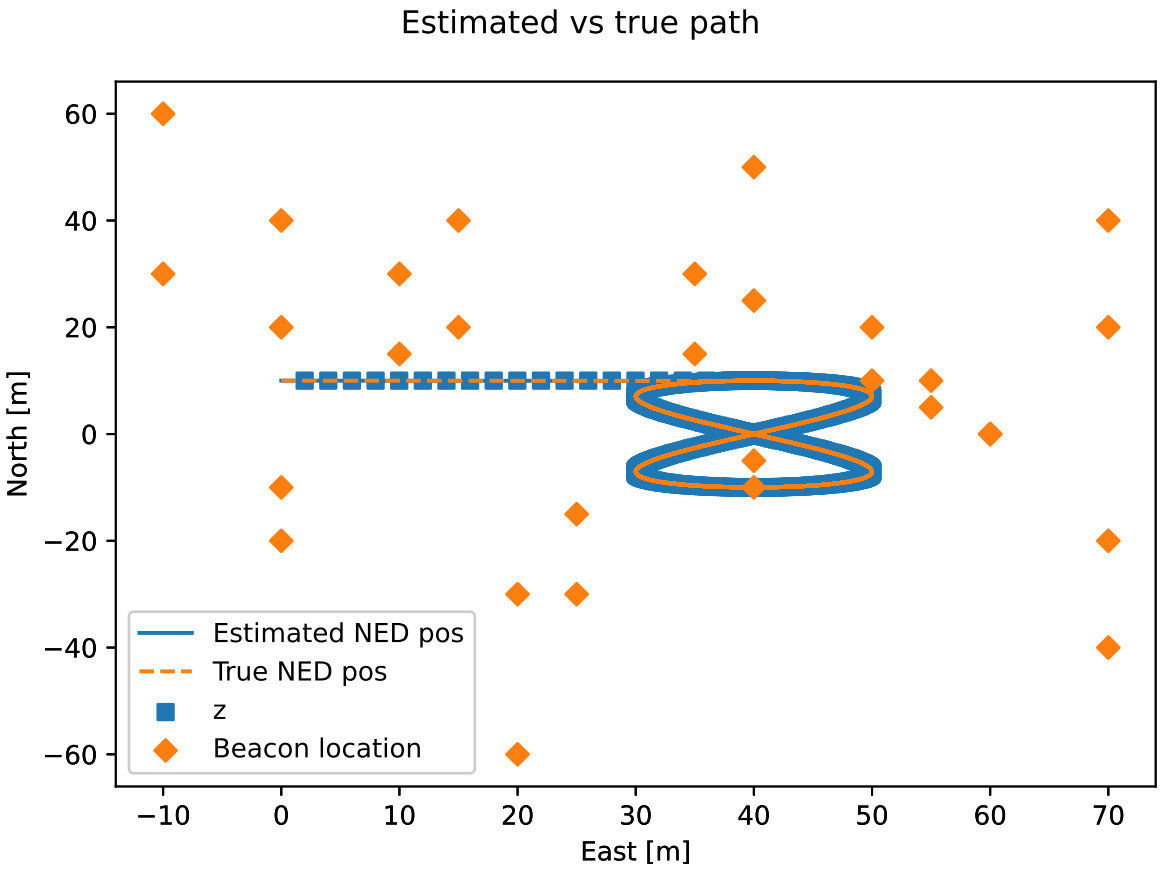


Figure 4.1.1: *True vs estimated 3D trajectory with beacon location and position measurements*

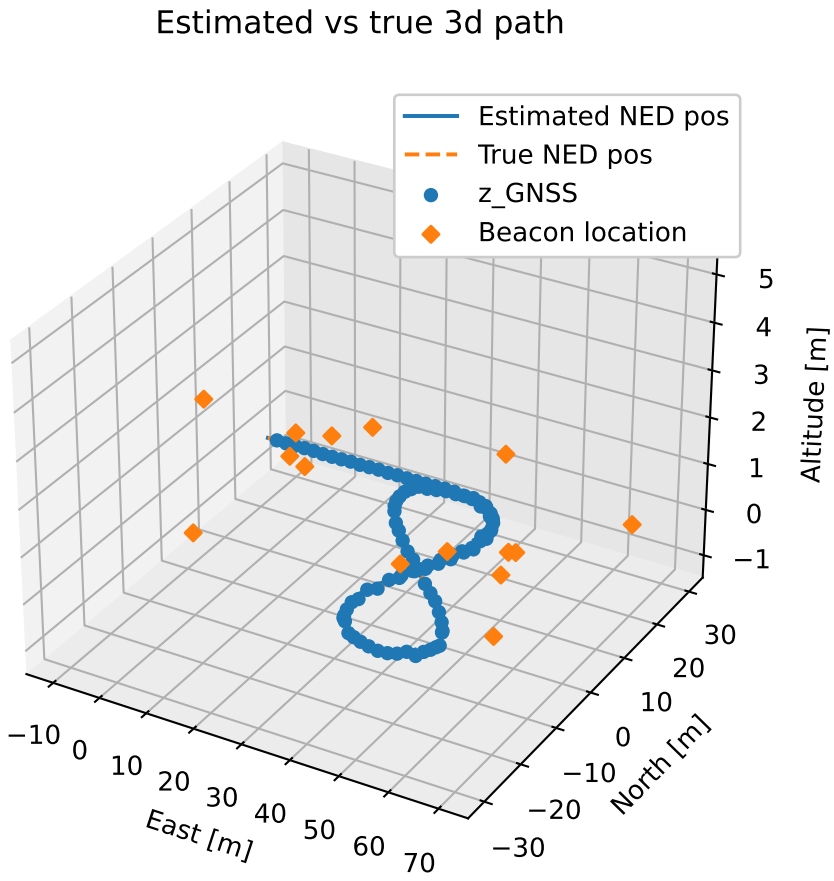


Figure 4.1.2: *True vs estimated 3D trajectory with beacon location and generic position measurements*

2. **Generic position measurements:**
   The generic position measurements were generated similar to (4.1.1b), but sampled at 1Hz.

3. **IMU measurements:**
   The accelerometer and gyroscope measurements were generated according to equation (3.1.3) and (3.1.4). The standard deviation of the measurement noise was based on the standard deviation found in section Section 3.2.

4. **Attitude:**
   The simulated gyro readings are not designed to match the kinematics of any kind of a vessel, but rather designed to verify that the filter is able to reproduce the *"true"* attitude given by the euler angle approximation from the differential equation represented by (4.1.2) through the equations given in section Section 3.3.

$$\dot{\boldsymbol{\Theta}} = \boldsymbol{T}(\boldsymbol{\Theta_{nb}})\boldsymbol{\omega}_{nb}^{b}, \quad \text{where} \quad \boldsymbol{T}(\boldsymbol{\Theta_{nb}}) = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \frac{\sin(\phi)}{\cos(\theta)} & \frac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} \tag{4.1.2}$$

The attitude given by (4.1.2) was compared to the discretized quaternion attitude described in (3.3.6) - (3.3.8)

Due to the nature of error propagation in the Kalman filter, which can be seen in matrix A in (3.3.12), we note that the attitude error will start to drift if the measured signal

is not rich enough. Figure Figure 4.1.3 displays the drift caused by the gyro readings not being rich enough signal after t = 300s, where the gyro measurements were set to be zero. Instead, a sine gyro reading were set to oscillate after t = 300s. The resulting measurement and attitude estimation can be seen in figure Figure 4.1.4, and Figure 4.1.5 where the simulation time is set to 1 hour.

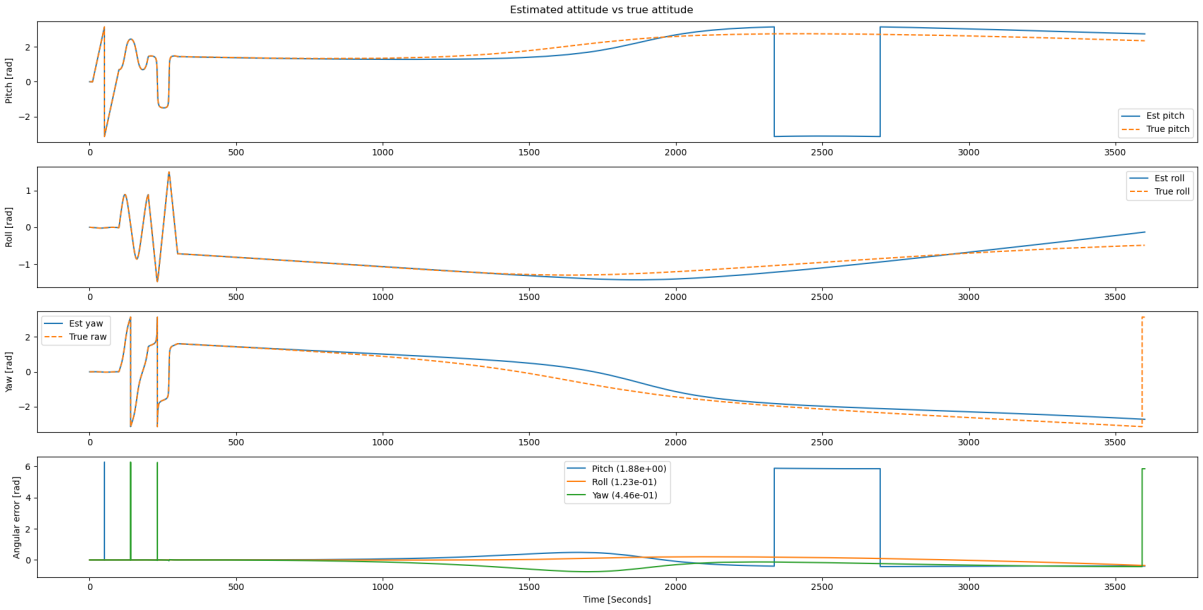


Figure 4.1.3: *Attitude estimation with steady state gyro readings*

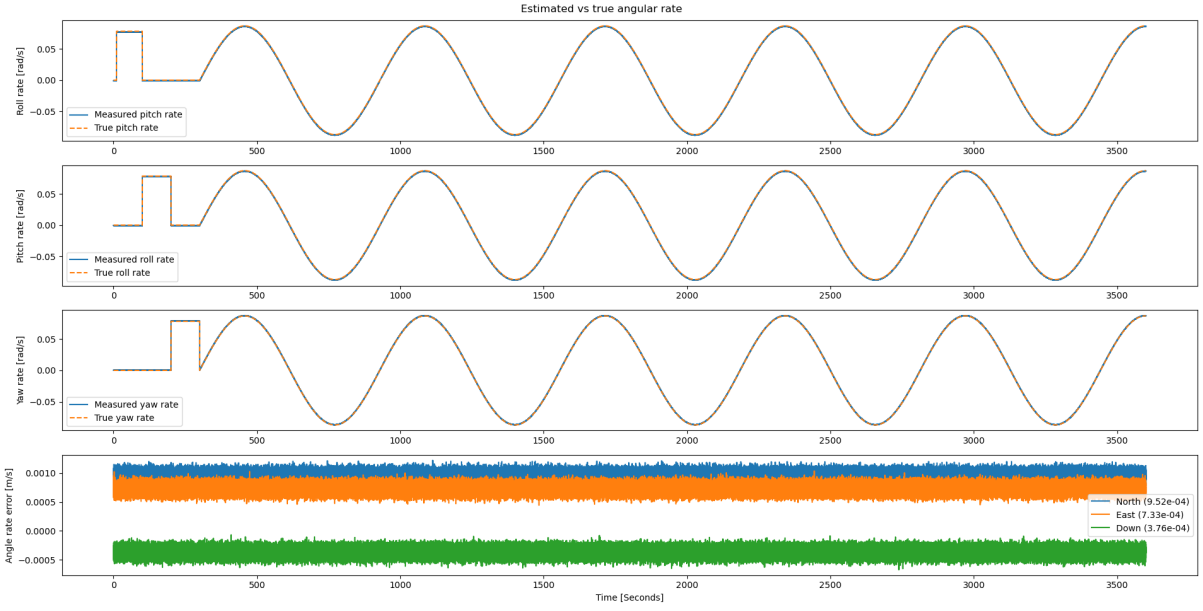


Figure 4.1.4: *Simulated gyro readings for t = 1 hour*

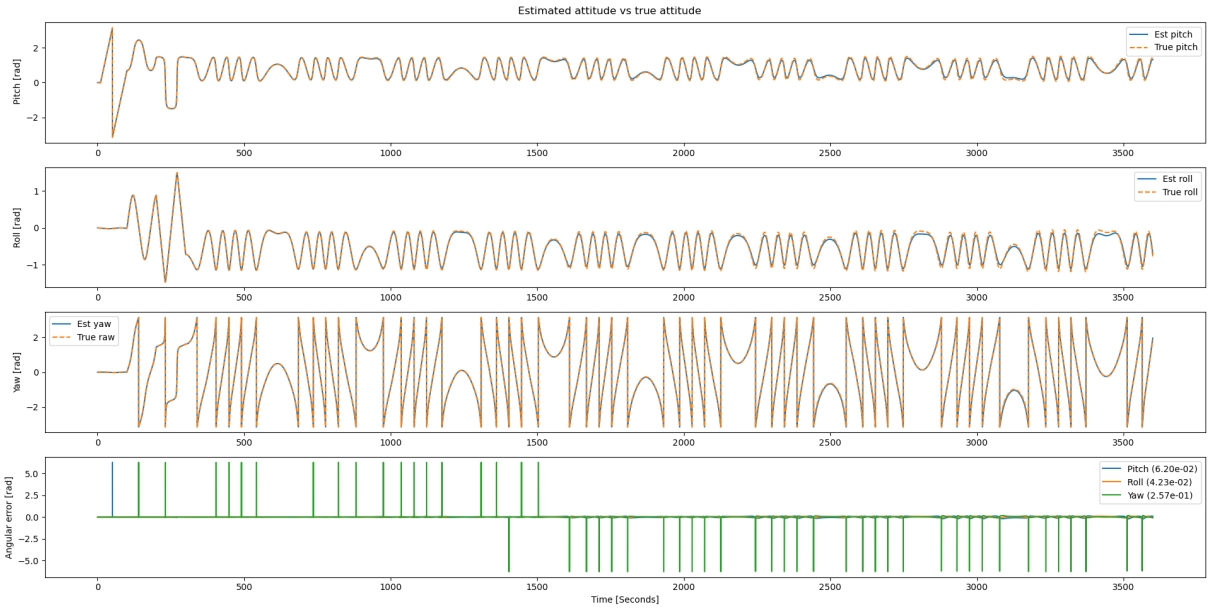


Figure 4.1.5: *Estimated attitude from* (4.1.2) *vs ESKF estimation*

5. **Accelerometer and gyro biases:**
   The accelerometer and gyro biases simulation data were modeled as a first-order Gauss Markov process given by (4.1.3). The same bias model can be seen in (3.3.2).

$$\begin{bmatrix} \dot{a}_{bt}^{b} \\ \dot{\omega}_{bt}^{b} \end{bmatrix} = \begin{bmatrix} -\frac{1}{T} a_{b}^{b} + a_{w}^{b} \\ -\frac{1}{T} \omega_{b}^{b} + \omega_{w}^{b} \end{bmatrix} \tag{4.1.3}$$

   The driving noises a_w^b and ω_w^b are white Gaussian noises with standard deviations set to have a standard deviation in milli-gs for the accelerometer and in a order of magnitude smaller for the gyroscope bias noise standard deviation.

Part II

Error State Kalman Filter Updates & Aided INS Designs

Chapter 5

The Error State Kalman Filter

This chapter presents the derivation of the discrete time equations in the update steps based on the system dynamics in Section 3.3. The derivation is based of the Error-State Kalman Filter from Solà [7], which Brekke [11] presents in chapter 10 of *Fundamentals of Sensor Fusion*. In Section 5.2 and Section 5.3, the sequential and UD-factorized filter variations presented by Simon [3], D'Souza [4], Thornton [5] and Bierman [8] are presented, with the following variation algorithm flow presented in Section 5.4.1 - Section 5.4.3.

Both the sequential and the UD-filtered variation of the ESKF shares the same kinematics and model as the batch-wise ESKF derived in Section 5.1. The sequential variation differ only in the measurement update, while the UDU-factorized variation differ in both the time-update and the measurement update.

5.1 The standard batch-wise ESKF

The error-state Kalman filter derived in this section is the filter variation we later also call the *"Batch-wise filter variation"*, or the standard text-book version of the filter. This variation of the filter is the one used to benchmark the other two filter variations. A special requirement in this variation requires is a built in matrix library for matrix multiplication and the matrix inversion in the Kalman gain computation in the measurement update.

5.1.1 Discretization of covariance and covariance propagation in the time update

A reasonable approximation of the covariance prediction can be achieved by using a standard method for discretizing the LTV system with the same frequency as the IMU measurement, where we in this implementation use Van Loans formula to discretize the transition matrix $\boldsymbol{\Phi}(\boldsymbol{x})$ and the continuous time covariance matrix $\boldsymbol{Q}(\boldsymbol{x})$. Van Loan's formula [15] as defined in theorem 4.5.2 by Brekke [11] (p. 65), and are given in (5.1.1) - (5.1.4). An alternative discretization method is to evaluate the transition matrix corresponding to $\boldsymbol{\Phi}(\boldsymbol{x})$ through a Taylor series truncation, suggested by Sola.

To discretize the matrices we first define the Van Loan matrix $\boldsymbol{V}$:

$$\boldsymbol{V} = \begin{bmatrix} -\boldsymbol{\Phi} & \boldsymbol{G}\boldsymbol{Q}\boldsymbol{G}^T \\ \mathbf{0} & \boldsymbol{\Phi}^T \end{bmatrix} \tag{5.1.1}$$

where G is the continous time error state noise input matrix defined as

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{R}_b^n(\boldsymbol{q}_b^n)S(\boldsymbol{a}_m^b - \boldsymbol{a}_b^b) & -\boldsymbol{R}_b^n(\boldsymbol{q}_b^n) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & -S(\boldsymbol{\omega}_m^b - \boldsymbol{\omega}_b^b) & \boldsymbol{0} & -\boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & -p_{ab}\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & -p_{\omega b}\boldsymbol{I} \end{bmatrix}, G = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{R}_b^n(\boldsymbol{q}_b^n) & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{5.1.2}$$

Then take the matrix exponential of this, which yields

$$\exp^{\boldsymbol{V}\Delta t} = \begin{bmatrix} \times & \boldsymbol{V_2} \\ \boldsymbol{0} & \boldsymbol{V_1} \end{bmatrix} \tag{5.1.3}$$

The discretized transition matrix $\boldsymbol{\Phi_k}$ is then the lower right matrix $\boldsymbol{V_1}$, and the discrete covariance matrix $\boldsymbol{Q}_k$, is the discretized transmition matrix multiplied by the upper right matrix $\boldsymbol{V_2}$, as shown in (5.1.4). The subscript $(.)_k$ denotes the object at time step k.

$$\boldsymbol{\Phi_k} = \boldsymbol{V_1}^T \tag{5.1.4a}$$

$$\boldsymbol{Q}_k = \boldsymbol{V_1}^T \boldsymbol{V_2} \tag{5.1.4b}$$

From this, the covariance propagation $\boldsymbol{P}_k^-$ is given as

$$\boldsymbol{P}_k^- = \boldsymbol{\Phi}_k \boldsymbol{P}_{k-1}^+ \boldsymbol{\Phi}_k^T + \boldsymbol{Q}_k \tag{5.1.5}$$

5.1.2 Measurement update in the ESKF

A general approach to generating the measurement jacobian for the measurement update in the ESKF is given by Brekke as seen in (5.1.8).

Aiding measurements can be related to the state vector using models of the form

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{h}(\boldsymbol{x_t}) + \boldsymbol{w}, \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}) \\ \Rightarrow \boldsymbol{z} &= \boldsymbol{h}(\boldsymbol{x} \otimes \boldsymbol{\delta x}) + \boldsymbol{w}, \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}) \end{aligned} \tag{5.1.6}$$

When linearizing around $\boldsymbol{\delta x} = 0$, equation (5.1.6) yields

$$\boldsymbol{z} \approx \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{H}\boldsymbol{\delta x} + \boldsymbol{w} \tag{5.1.7}$$

The Jacobian H can be expresesd as a product of the Jacobians of $\boldsymbol{h.}$ and $\boldsymbol{x} \otimes \boldsymbol{\delta x}$.

$$\boldsymbol{H} = \frac{\partial}{\partial \boldsymbol{x_t}} \boldsymbol{h}|_{\boldsymbol{x_t}=\boldsymbol{x}} \cdot \frac{\partial}{\partial \boldsymbol{\delta x}} \boldsymbol{x_t} \boldsymbol{H_x} \boldsymbol{X_{\delta x}} \tag{5.1.8}$$

and $\boldsymbol{H_{\delta x}}$ is defined by Brekke as:

$$\begin{aligned}\boldsymbol{X}_{\delta \boldsymbol{x}} &= \begin{bmatrix} \boldsymbol{I_6} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q}_{\delta\boldsymbol{\theta}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I_6} \end{bmatrix}, \qquad \text{where} \\ \boldsymbol{Q}_{\delta\boldsymbol{\theta}} &\approx \frac{\partial}{\partial \boldsymbol{\delta\theta}}(\boldsymbol{q} \otimes \boldsymbol{\delta q})|_q = \frac{\partial}{\partial \boldsymbol{\delta}}(\boldsymbol{q} \otimes \boldsymbol{\delta q})|_q \frac{\partial}{\partial \boldsymbol{\delta\theta}}(\boldsymbol{\delta q})|_{\delta\theta=0} \\ &= \frac{1}{2}\begin{bmatrix} -\boldsymbol{\epsilon_1} & -\boldsymbol{\epsilon_2} & -\boldsymbol{\epsilon_3} \\ \boldsymbol{\eta} & -\boldsymbol{\epsilon_3} & \boldsymbol{\epsilon_2} \\ \boldsymbol{\epsilon_3} & \boldsymbol{\eta} & -\boldsymbol{\epsilon_1} \\ -\boldsymbol{\epsilon_2} & \boldsymbol{\epsilon_1} & \boldsymbol{\eta} \end{bmatrix}\end{aligned} \tag{5.1.9}$$

The matrix $\boldsymbol{H_x}$ depends on the measurement model, and were chosen as $[\boldsymbol{I_3} \qquad \boldsymbol{0_{3\times12}}]$ when verifying the correction part of the filter with generic ranges during development.

As an example, if the measurement model would have been based on GNSS measurements as the aiding measurement range equation would have been modeled as

$$\boldsymbol{\rho} = \rho_{\alpha,s} + c(\delta t_\alpha - \delta t_s) + I_{\alpha,s} + T_{\alpha,s} + \epsilon_p \tag{5.1.10}$$

where ρ is the measured pseudorange, δt_α and δt_s are receiver and satellite (in this case beacon) errors, and $I_{\alpha,s}$ and $T_{\alpha,s}$ are ionospheric and tropospheric delays. ϵ_p is the remaining noise and unmodelled errors. By choosing GNSS for the aiding measurement, we would in other words be required to include the sender and receiver clock errors as a state in our state space

Instead we simplify the measurement model by making a few assumptions which will lead to the beacon range measurement update this model is based on. The first assumption is that true ranges are generated between beacons and the user receivers instead of originating from a GNSS constellation. This assumption neglects the receiver clock error and error contributions from the ionosphere and troposphere. As a consequence, the pseudorange given in (5.1.10) reduces to the geometric range plus noise and errors given in (5.1.11).

$$\rho \approx \rho_{\alpha,s} = \|\boldsymbol{p_u} - \boldsymbol{p_i}\|_2 + \epsilon_p \tag{5.1.11}$$

Where the Euclidean norm is given by (2.4.1). This leads to the matrix $\boldsymbol{H_x}$ changing to (5.1.15). In addition, the receivers position is assumed to be located in the center of mass of the point mass simulating the vessel in this model. In the case where the receivers are not located in the center of mass, it would be necessary to compensate with a lever arm term to accurately compensate for the difference between vessel location and signal receiver location.

To determine the receiver position from the ranges we generate the design matrix that connects the measurements to the beacons, the residual range measurement and the measurement matrix H. This design matrix contains the *user-to-satellite-*, or in this case *user-to-beacon Line of Sight* (LOS) vectors, which are defined in (5.1.13).

1. The estimated residual range measurement, or innovation vector, is defined as

$$\nu = \boldsymbol{z} - \hat{\boldsymbol{z}} \tag{5.1.12}$$

   where $z = ||p_u - p^{(i)}||$ and $\hat{\boldsymbol{z}} = ||\hat{p}_u - p^{(i)}||$

2. The user-to-satellite Line Of Sight (LOS) vector is defined as

$$\boldsymbol{h^{(i)}} = \frac{\boldsymbol{p_u} - \boldsymbol{p^{(i)}}}{\|\boldsymbol{p_u} - \boldsymbol{p_i}\|} \tag{5.1.13}$$

where $\boldsymbol{p}^{(i)}$ is beacon number {i} location, and $\hat{\boldsymbol{p_u}}$ is the estimated user position, while $\boldsymbol{p_u}$, which is also denoted as $\boldsymbol{y_{meas}}$, or $\boldsymbol{y_{mk}}$ in later sections, is the measured user position.

3. The geometry matrix connecting measurements to the beacons are given by (5.1.14)

$$\boldsymbol{D} = \begin{bmatrix} (\boldsymbol{h^{(1)}} \\ (\boldsymbol{h^{(2)}} \\ \vdots \\ (\boldsymbol{h^{(i)}} \end{bmatrix} \in \mathbb{R}^{ix3} \tag{5.1.14}$$

4. The measurement matrix H are then designed as

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{D} & \boldsymbol{0_{i\times 12}} \end{bmatrix} \in \mathbb{R}^{ix15} \tag{5.1.15}$$

The ESKF measurement update steps are done by the ESKF measurement equations given in (5.1.16).

$$\text{Covariance of innovation:} \quad \boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_{k-1}^- \boldsymbol{H}_k^T + R_k \tag{5.1.16a}$$

$$\text{Kalman gain:} \quad \boldsymbol{K}_k = \boldsymbol{P}_{k-1}^- \boldsymbol{H}_k^T \boldsymbol{S}_k^{-1} \tag{5.1.16b}$$

$$\text{Innovation vector:} \quad \boldsymbol{\nu}_k = (\boldsymbol{z}_k - \boldsymbol{h}(\boldsymbol{x_k})) = \boldsymbol{z}_k - \hat{\boldsymbol{z}}_k \tag{5.1.16c}$$

$$\text{Posterior state estimate:} \quad \boldsymbol{\delta \hat{x}}_k = \boldsymbol{K}_k \cdot \boldsymbol{\nu}_k \tag{5.1.16d}$$

$$\text{Posterior covariance:} \quad \boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k-1}^- (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k)^T + \boldsymbol{K}_k \boldsymbol{R}_k \boldsymbol{K}_k^T \tag{5.1.16e}$$

In (5.1.16e), the Joseph form variant of the time update step is used to strengthen the numerical stability of the covariance update computation.

5.1.3 Injection and reset

After obtaining an estimate $\boldsymbol{\delta \hat{x}}$ of the error state, the nominal state gets updated with the observed error state. This is denoted as the injection step, and is given in (5.1.17), where $\otimes$ means sum or quaternion product.

$$\hat{\boldsymbol{x}}_k^+ = \boldsymbol{x}_k^- \otimes \boldsymbol{\delta \hat{x}}_k^+ \tag{5.1.17}$$

After this step is done, all the information contained in $\boldsymbol{\delta \hat{x}}$ is moved to the nominal state, ensuring the expectation of the error state to become zero until the next measurement update occurs.

The nominal orientation $\boldsymbol{q}$ changes due to the injection, so the covariance of $\boldsymbol{\delta \theta}$ must be modified through the ESKF covariance reset [11] (p.185). The covariance reset can be seen in (5.1.18).

$$\boldsymbol{P}_{k,inj} = \boldsymbol{G} \boldsymbol{P}_k^+ \boldsymbol{G}^T \tag{5.1.18a}$$

$$\text{where} \quad \boldsymbol{G} = \begin{bmatrix} \boldsymbol{I}_6 & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} - S(\frac{1}{2}\boldsymbol{\delta \hat{\theta}}) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_6 \end{bmatrix} \tag{5.1.18b}$$

5.2 Sequential ESKF

As stated in the introduction of this section, the sequential variation shares the time update step and the kinematics of the standard batch-wise implementation. The difference between the models is how the measurements are handled in the measurement update, which requires certain conditions to be met. The first is regarding how the measurements are handled. Here, the measurement are handled one by one to avoid the matrix inversion when computing the Kalman gain. This requires that if we obtain multiple measurements at the same time such as the measurement vector $\boldsymbol{y}_k$, we need to separate them into individual measurements $y_k(m)$, where m denotes the measurement number. The second requirement is that the measurement covariance matrix $\boldsymbol{R}_k$ is either a diagonal matrix, or a constant covariance factor R.

From here we define

$$\boldsymbol{R}_k = \begin{bmatrix} R_{1k} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & R_{mk} \end{bmatrix} \tag{5.2.1}$$

We also define H_mk as the mth row of $\boldsymbol{H}_k$ and let

$$y_{mk} = H_{mk}\boldsymbol{x}_k + v_{mk} \tag{5.2.2a}$$
$$v_{mk} = (0, R_{mk}) \tag{5.2.2b}$$

Before we begin the measurement update, we need to initialize the partial posterior state and covariance. Here the left hand sides denotes the variables before any measurements have been handled and are equal to the a priori estimate.

$$\hat{\boldsymbol{x}}_0 k^+ = \hat{\boldsymbol{x}}_k^- \tag{5.2.3a}$$
$$\boldsymbol{P}_{0k}^+ = \boldsymbol{P}_k^- \tag{5.2.3b}$$

To not loose any information during this step, we have to ensure that the information from the other partual measurement iterations are not lost throughout the loop. This is done by accumulating both the prior partial error state into both $\hat{\boldsymbol{z}}$ and $\boldsymbol{\delta}\hat{\boldsymbol{x}}_{k,m+1}$, and the partial covariance computation into the next partial covariance computation as seen in (5.2.4).

$$\boldsymbol{H}_{mk} = \left[(\hat{p}_u - p^{(m)})/\hat{\boldsymbol{z}}_r, \quad \mathbf{0_{12}}\right] \tag{5.2.4a}$$
$$\boldsymbol{S}_{mk} = \boldsymbol{H}_{mk}\boldsymbol{P}_{mk}^+\boldsymbol{H}_{mk}^T + \boldsymbol{R}_{mk} \tag{5.2.4b}$$
$$\boldsymbol{K}_{mk} = \frac{\boldsymbol{P}_m^+\boldsymbol{H}_{mk}^T}{\boldsymbol{S}_{mk}} \tag{5.2.4c}$$
$$\boldsymbol{\delta}\hat{\boldsymbol{x}}_{mk+1} = \boldsymbol{\delta}\hat{\boldsymbol{x}}_{mk} + \boldsymbol{K}_{mk} \cdot (\boldsymbol{z} - \hat{\boldsymbol{z}}) \tag{5.2.4d}$$
$$\boldsymbol{P}_{m+1k}^+ = (\boldsymbol{I} - \boldsymbol{K}_{mk}\boldsymbol{H}_{mk})\boldsymbol{P}_{mk}^+(\boldsymbol{I} - \boldsymbol{K}_{mk}\boldsymbol{H}_{mk})^T \tag{5.2.4e}$$

The same equations can be seen in the sequential algorithm described in Section 5.4.2.

After all measurements have been handled, we set the updated state error and covariance as

$$\boldsymbol{\delta\hat{x}}_k^+ = \boldsymbol{\delta\hat{x}}_{km}^+ \tag{5.2.5a}$$
$$\boldsymbol{P}_k^+ = \boldsymbol{P}_{mk}^+ \tag{5.2.5b}$$

5.3 UDU-factorized ESKF

As stated in the introduction, the UD-factorized variation differs from the standard variation in both the time- and measurement update. In Section 5.3.1 I will present the factorization functions created, Section 5.3.2 and Section 5.3.3 will present the structures and the most important equations used, while Section 5.4.3 will display the equations and algorithm flow in its completeness. The implementations are made based on D'Souzas summary of the UDU-factorization in [4] (Ch. 7), and Simons chapter about U-D filtering in [3]. Both D'Souza and Simons notes are largely based on Bierman and Thorntons "IEEE conference on Decision and control" (1975), "JPL Tehcnical Memorandum" (1976), as well as Biermans *"Factorization Methods for Discrete Sequential Estimation"* [8]".

The filter variation is another way to increase the numerical precision of the Kalman Filter at the cost of an increased computational load when comparing it to the standard and sequential variations. The algorithm factorizes a n by n, symmetric, positive matrix $\boldsymbol{P}$, and factorizes to a upper triangular n by n matrix $\boldsymbol{U}$, as well as a diagonal n by n matrix $\boldsymbol{D}$. By factorizing the covariance into a upper triangular matrix factor and a diagonal factor one may easily investigate the positive definiteness of the covariance matrix through the diagonal factor $\boldsymbol{D}$ by examining the matrix.

5.3.1 Factorization Code

As stated, we need a n by n, symmetric positive matrix to make this work. The following code snippet displays the function "$UDU\_factorization(\boldsymbol{P})$", which takes the covariance matrix as input argument, and returns the upper triangular $\boldsymbol{U}$ and diagonal $\boldsymbol{D}$. In addition to "$UDU\_factorization(\boldsymbol{P})$", we also need a code which solves the ladder problem of the time update in [4] (p. 66-67), the Modified Gram-Schmidt Algorithm, here named as "$mod\_gram\_NASA(\boldsymbol{Y}, \tilde{\boldsymbol{D}})$, which completes the covariance propagation through the structure $\boldsymbol{Y}$ seen in (5.3.2),

In [4], D'Souza uses the notation $\bar{\boldsymbol{P}}$ to denote the propagated covariance, which we have written as $\boldsymbol{P}_k^-$, and I'll continue to do so with the exception of the code snippet for modified gram Schmidt in this section.

```python
def UDU_factorization(P: np.ndarray):
    """
    Args:
    -----------
    P (np.ndarray): The covariance matrix. Must be symmetric and positive semidefinite, shape ((15,15))

    Returns:
    -----------
    U, D np.ndarray: The upper triangular matrix U, and Diagonal matrix D
    """

    n = len(P)
    U = np.zeros((n,n))
    d = np.zeros(n)

    U[:,-1] = P[:,-1] / P[-1,-1]
    d[-1] = P[-1, -1]

    for j in range(n-2,-1,-1):
        d[j] = P[j,j] - np.sum(d[j+1:n] * (U[j,j+1:n])**2)

        U[j,j] = 1.0

        for i in range(j-1,-1,-1):

            U[i,j] = (P[i,j] - np.sum(d[j+1:n] * (U[i,j+1:n]) * (U[j,j+1:n]))) / d[j]

    D = np.diag(d)

    return U, D
```

Listing 5.1: *UDU-factorization algorithm*

```python
def mod_gram_NASA(Y, D_tilde):

    (n, m) = np.shape(Y)
    b = np.zeros((n, m))
    f = np.zeros((n, m))

    D_bar = np.zeros((n,n))
    U_bar = np.zeros((n,n))

    for k in range(n-1, -1,-1):
        #Copy the row Y[k] into b[k]. This can be done more efficient and does not need to be its own loop
        b[k,:] = Y[k,:]

    for j in range(n-1, 0,-1):

        U_bar[j,j] = 1

        f[j,:] = D_tilde @ b[j,:]

        D_bar[j,j] = b[j,:].T @ f[j,:]

        f[j,:] = f[j,:] / D_bar[j,j]

        for i in range(0, j):

            U_bar[i,j] = b[i,:].T @ f[j,:]
            b[i,:] = b[i,:] - U_bar[i,j] * b[j,:]
```

```
28
29      U_bar[0,0] = 1
30      f[0,:] = D_tilde @ b[0,:]
31      D_bar[0,0] = b[0,:] @ f[0,:]
32      return U_bar, D_bar
```

Listing 5.2: *Modified Gram-Schmidt algorithm*

5.3.2 The time update problem of the UDU variation

First, we start with the propagation of the covariance $\boldsymbol{P}_k^-$, which we want to find based on the prior observation $\boldsymbol{P}_{k-1}^+$, the discrete transition matrix $\boldsymbol{\Phi}_k$ and the covariance of the process noise matrix$\boldsymbol{Q}_k$.

$\boldsymbol{P}_{k-1}^+$ is sent through the aforementioned factorization script which returns the covariance factors $\boldsymbol{U}_{k-1}^+$ and $\boldsymbol{D}_{k-1}^+$.

$$\begin{aligned} \boldsymbol{U}_k^- \boldsymbol{D}_k^- \boldsymbol{U}_k^{-T} &= \boldsymbol{\Phi}_k \boldsymbol{U}_{k-1}^+ \boldsymbol{D}_{k-1}^+ \boldsymbol{U}_{k-1}^+ \boldsymbol{\Phi}_k^T + \boldsymbol{Q}_k \\ &= \begin{bmatrix} \boldsymbol{\Phi}_k \boldsymbol{U}_{k-1}^+ & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_{k-1}^+ & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q}_k \end{bmatrix} \begin{bmatrix} \boldsymbol{\Phi}_k \boldsymbol{U}_{k-1}^+ & \boldsymbol{I} \end{bmatrix}^T \end{aligned} \tag{5.3.1}$$

From here we define $\boldsymbol{Y}_k$ as

$$\boldsymbol{Y}_k = \begin{bmatrix} \boldsymbol{\Phi}_k \boldsymbol{U}_{k-1}^+ & \boldsymbol{I} \end{bmatrix} \tag{5.3.2}$$

and $\tilde{\boldsymbol{D}}_k$ as

$$\tilde{\boldsymbol{D}}_k = \begin{bmatrix} \boldsymbol{D}_{k-1}^+ & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q}_k \end{bmatrix} \tag{5.3.3}$$

To complete the covariance propagation we now seek a matrix $\boldsymbol{T}_k$ which renders

$$\boldsymbol{Y}_k \boldsymbol{T}_k^{-1} = \begin{bmatrix} \boldsymbol{U}_k & \boldsymbol{0} \end{bmatrix} \tag{5.3.4}$$

To do this, we need the Gram Schmidt function which takes $\boldsymbol{Y}_k$ and $\tilde{\boldsymbol{D}}_k$ as arguments, and returns $\boldsymbol{U}_k^-$ and $\boldsymbol{D}_k^-$, from which we can compute $\boldsymbol{P}_k^-$ from through

$$\boldsymbol{P}_k^- = \boldsymbol{U}_k^- \boldsymbol{D}_k^- \boldsymbol{U}_k^{-T} \tag{5.3.5}$$

5.3.3 Measurement update of the UDU-variation

The same requirements for the sequential measurement still holds for the UDU measurement update. We are required to handling the measurements one at the time, and the measurement covariance being either a diagonal matrix or a scalar factor. We will also be needing to initialize the measurement update in a similar fashion as the sequential measurement update.

Thus we have

$$\boldsymbol{R}_k = \begin{bmatrix} R_{1k} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & R_{mk} \end{bmatrix} \tag{5.3.6}$$

$\boldsymbol{H}_{mk}$ is defined as the m-th row of $\boldsymbol{H}_k$ and

$$y_{mk} = H_{mk}\boldsymbol{x}_k + v_{mk} \tag{5.3.7a}$$
$$v_{mk} = (0, R_{mk}) \tag{5.3.7b}$$

The partial posterior error state and covariance are initialized as

$$\boldsymbol{\delta}\hat{\boldsymbol{x}}_0 k^+ = \boldsymbol{\delta}\hat{\boldsymbol{x}}_k^- \tag{5.3.8a}$$
$$\boldsymbol{P}_{0k}^+ = \boldsymbol{P}_k^- \tag{5.3.8b}$$

The equations for $\hat{\boldsymbol{z}}_r$, $\boldsymbol{H}_{mk}$ and $\boldsymbol{S}_{mk}$ can be seen in both the algorithm for sequential and UD-variations in Section 5.4.2 and Section 5.4.3. As for $\boldsymbol{P}_k^+$, we need to take a detour through some partial computations.

When handling the measurements, the first step is to factorize the partial covariance $\boldsymbol{P}_{mk}^+$ to get the initial partial covariance factors $\boldsymbol{U}_{mk}^-$ and $\boldsymbol{D}_{mk}^-$. Next we define $\tilde{\boldsymbol{P}}_{mk}$ as

$$\tilde{\boldsymbol{P}}_{mk} = \boldsymbol{D}_{mk}^- - \frac{1}{\boldsymbol{S}_{mk}}(\boldsymbol{D}_{mk}^-\boldsymbol{U}_{mk}^{-T}\boldsymbol{H}_{mk}^{-T})(\boldsymbol{D}_{mk}^-\boldsymbol{U}_{mk}^{-T}\boldsymbol{H}_{mk}^{-T})^T \tag{5.3.9}$$

Then we factorize $\tilde{\boldsymbol{P}}_{mk}$ through the factorization function to get $\tilde{\boldsymbol{U}}_{mk}$ and $\tilde{\boldsymbol{D}}_{mk}$, and define the partial covariance factors $\boldsymbol{U}_{mk}^+$ and $\boldsymbol{D}_{mk}^+$ as

$$\boldsymbol{U}_{mk}^+ = \boldsymbol{U}_{mk}^-\tilde{\boldsymbol{U}}_{mk} \tag{5.3.10a}$$
$$\boldsymbol{D}_{mk}^+ = \tilde{\boldsymbol{D}}_{mk} \tag{5.3.10b}$$

From here we can obtain the Kalman gain through

$$\boldsymbol{K}_{mk} = \frac{\boldsymbol{U}_{mk}^+\boldsymbol{D}_{mk}^+\boldsymbol{U}_{mk}^{+T}\boldsymbol{H}_{mk}^T}{\boldsymbol{S}_{mk}} \tag{5.3.11}$$

And finally the partial covariance computation with Joseph form as $\boldsymbol{P}_{m+1k}$ as

$$\boldsymbol{P}_{m+1k}^+ = (\boldsymbol{I} - \boldsymbol{K}_{mk}\boldsymbol{H}_{mk})\boldsymbol{U}_{mk}^+\boldsymbol{D}_{mk}^+\boldsymbol{U}_{mk}^{T+}(\boldsymbol{I} - \boldsymbol{K}_{mk}\boldsymbol{H}_{mk})^T + \boldsymbol{K}_{mk}\boldsymbol{R}_{mk}\boldsymbol{K}_{mk}^T \tag{5.3.12}$$

which the posterior covariance $\boldsymbol{P}_k^+$ is set to be equal to when all the measurements have been handled i.e

$$\boldsymbol{P}_k^+ = \boldsymbol{P}_{mk}^+ \tag{5.3.13}$$

The error state is computed similar as before with the exception of using the new Kalman gain displayed in (5.3.11).

5.4 Filter algorithms

As stated in the introduction, the aim of this thesis is to compare the runtime and performance of the selected variations of the Error-State Kalman Filter. In the following sections the algorithms are divided into the time update step and a measurement update step. The algorithms listed in Section 5.4.1, Section 5.4.2 and section 5.4.3 are supported by the flowcharts given by Figure 5.4.1, Figure 5.4.2 and Figure 5.4.3. The equations presented in the prior sections about the standard, sequential and UDU-factorized ESKF variations are written out in the respective time update- and measurement update steps.

The following list offers a refreshment of variable notation for sub- and superscripts:

- n = number of states in the state space
- m = number of measurements in measurement vector
- When m is used as subscripts, such as $(.)_{mk}$, it denotes a iteration variable indicating that we're working with measurement number m for time step k.
- When m is used as superscript, such as $(\mathbb{R})^{m\times m}$, it denotes the number of measurements in the measurement vector.

5.4.1 Batch-wise algorithm

The batch wise algorithm follows the text book implementation of the update steps, where the all measurements are processed at once. The Kalman gain computation is based on a matrix inversion of the matrix $\boldsymbol{S}_k \in \mathbb{R}^{m\times m}$.

Time update: For each time step k, do:

1. Nominal state prediction/priori state estimation

$$\hat{\boldsymbol{x}}_k^- \leftarrow f(\hat{\boldsymbol{x}}_{k-1}, u, \Delta t) \tag{5.4.1}$$

   If no new measurements are recieved from the generic beacon measurement: Set the posteriori state estimate to be equal to the priori prediction $\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^-$.

2. Compute the discretized, linearized transition matrix $\boldsymbol{\Phi}_k$ and noise coviariance matrix $\boldsymbol{Q}_k$ through Van Loan as described in section (3.3.12)

3. Compute the priori covariance

$$\boldsymbol{P}_k^- = \boldsymbol{\Phi}_k \boldsymbol{P}_{k-1}^+ \boldsymbol{\Phi}_k^T + \boldsymbol{Q}_k \tag{5.4.2}$$

4. Average $\boldsymbol{P}_k^-$

$$\boldsymbol{P}_k^- = \frac{(\boldsymbol{P}_k^- + \boldsymbol{P}_k^{-T})}{2} \tag{5.4.3}$$

Measurement update: For each time step k when a new measurement is available, do:

1. Compute the measured and estimated ranges $\boldsymbol{z}_k$ and $\hat{\boldsymbol{z}}_k$

2. Compute $\boldsymbol{\nu}_k$

$$\boldsymbol{\nu}_k = (\boldsymbol{z}_k - \boldsymbol{h}(\hat{\boldsymbol{x}}_k^-)) = (\boldsymbol{z}_k - \hat{\boldsymbol{z}}_k) \tag{5.4.4}$$

3. Compute the measurement jacobian $\mathrm{H} \in \mathbb{R}^{m\times n}$ from equations (5.1.13) - (5.1.15) in section

4. Compute $\boldsymbol{S}_k$

$$\boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_{k-1}^{-} \boldsymbol{H}_k^T + \boldsymbol{R}_k \in \mathbb{R}^{m \times m} \tag{5.4.5}$$

5. Compute the kalman gain

$$\boldsymbol{K}_k = \boldsymbol{P}_{k-1}^{-} \boldsymbol{H}_k^T \boldsymbol{S}_k^{-1} \in \mathbb{R}^{m \times m} \tag{5.4.6}$$

6. Compute the estimated error δx through the innovation

$$\boldsymbol{\delta \hat{x}}_k^{+} = \boldsymbol{W}_k \cdot \boldsymbol{\nu}_k \in \mathbb{R}^{n \times 1} \tag{5.4.7}$$

7. Correct the covariance with the Joseph form of the measurement update.

$$\boldsymbol{P}_k^{+} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k-1}^{-} (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k)^T \in \mathbb{R}^{n \times n} \tag{5.4.8}$$

8. Inject the estimated error in the nominal state and correct the covariance through (5.1.18)

$$\hat{\boldsymbol{x}}_k^{+} = \hat{\boldsymbol{x}}_k^{-} \otimes \boldsymbol{\delta \hat{x}}_k^{+} \tag{5.4.9}$$

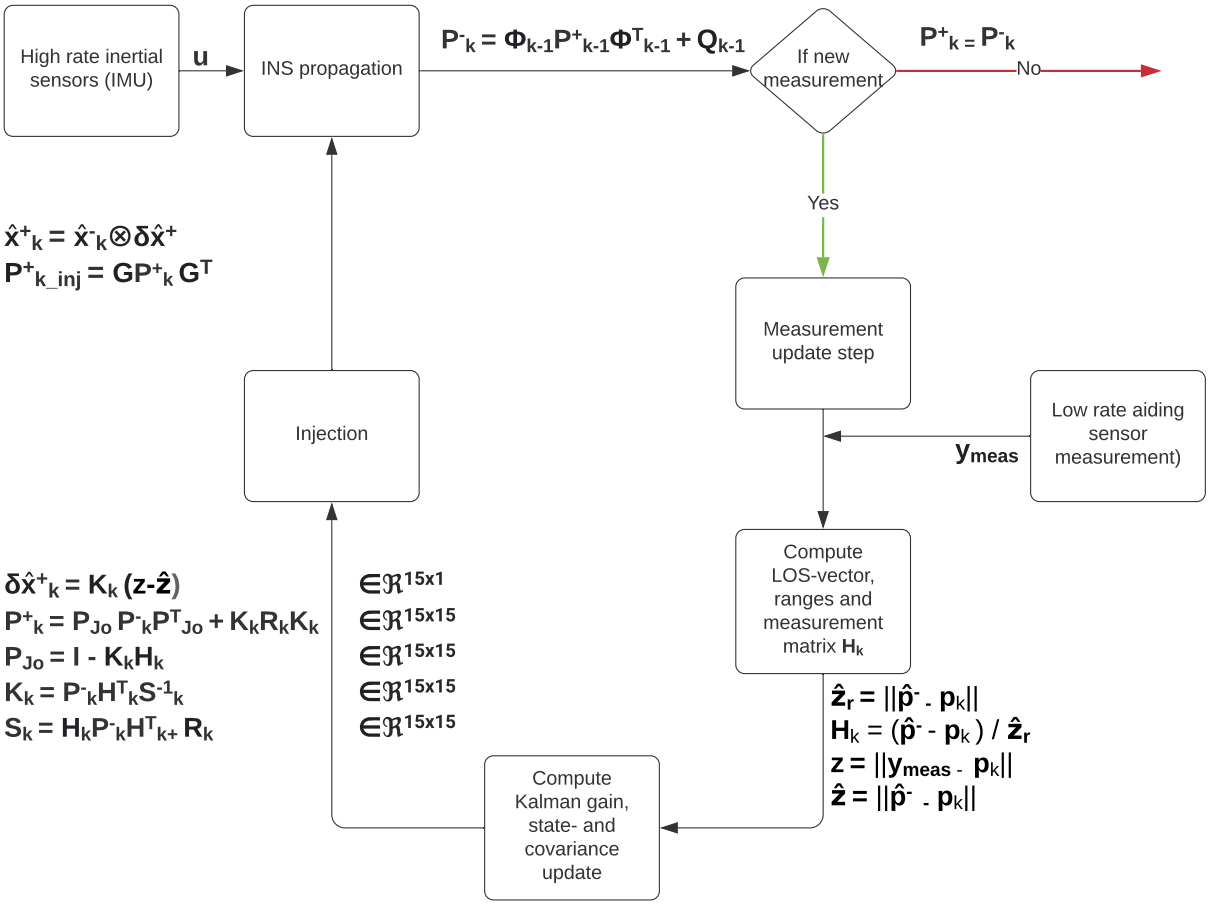


Figure 5.4.1: *Flowchart for the batchwise ESKF algorithm*

5.4.2 Sequential algorithm

Time update: The sequential algorithm time update are identical to the time update for the batch-wise variation.

Measurement update: For each time step k when a new measurement is available, do:

1. Error state and measurement matrix $\boldsymbol{H}$ is set to zero and the "partial", or "initial" covariance matrix $\boldsymbol{P}_{0k}^{+} = \boldsymbol{P}_{k}^{-}$

$$\boldsymbol{\delta\hat{x}}_{0k} = \mathbf{0} \tag{5.4.10}$$

$$\boldsymbol{P}_{0k}^{+} = \boldsymbol{P}_{k}^{-} \tag{5.4.11}$$

2. **for m in range(number of measurements):**

   (a) Compute the estimated user-beacon range for this beacon

$$\hat{\boldsymbol{z}}_r = ||\hat{p}_u - p^{(m)}|| \tag{5.4.12}$$

   (b) Compute $\boldsymbol{H}_{mk}$:

$$\boldsymbol{H}_{mk} \in \mathbb{R}^{1xn} = \left[(\hat{p}_u - p^{(m)})/\hat{\boldsymbol{z}}_r, \quad \mathbf{0_{12}}\right] \tag{5.4.13}$$

   (c) Compute $\hat{\boldsymbol{z}}$ for this beacon

$$\hat{\boldsymbol{z}} = ||\hat{p}^{u} - p^{(m)}|| + \boldsymbol{H}_{mk}\boldsymbol{\delta x}_{mk} \tag{5.4.14}$$

   (d) Compute $\boldsymbol{z}$ for this beacon

$$\boldsymbol{z} = ||\boldsymbol{y}_{meas} - p^{(m)}|| \tag{5.4.15}$$

   (e) Compute $\boldsymbol{S}_{mk}$

$$\boldsymbol{S}_{mk} = \boldsymbol{H}_{mk}\boldsymbol{P}_{mk}^{+}\boldsymbol{H}_{mk}^{T} + \boldsymbol{R}_{mk} \in \mathbb{R}^{1} \tag{5.4.16}$$

   (f) Compute the kalman gain

$$\boldsymbol{K}_{mk} = \frac{\boldsymbol{P}_{mk}^{+}\boldsymbol{H}_{mk}^{T}}{\boldsymbol{S}_{mk}} \in \mathbb{R}^{n\times 1} \tag{5.4.17}$$

   (g) Compute the estimated error $\boldsymbol{\delta\hat{x}}_{k+1,m}$ through the innovation of this one beacon and accumulate it

$$\boldsymbol{\delta\hat{x}}_{mk+1} = \boldsymbol{\delta\hat{x}}_{mk} + \boldsymbol{K}_{mk} \cdot (\boldsymbol{z} - \hat{\boldsymbol{z}}) \in \mathbb{R}^{n\times 1} \tag{5.4.18}$$

   (h) Compute the next partial covariance with the Joseph form of the measurement update from this beacon and accumulate it with the other

$$\boldsymbol{P}_{m+1k}^{+} = (\boldsymbol{I} - \boldsymbol{K}_{mk}\boldsymbol{H}_{mk})\boldsymbol{P}_{mk}^{+}(\boldsymbol{I} - \boldsymbol{K}_{mk}\boldsymbol{H}_{mk})^{T} \in \mathbb{R}^{n\times n} \tag{5.4.19}$$

3. When all measurements have been handled, compute the final covariance and error state

$$\boldsymbol{\delta\hat{x}}_{k}^{+} = \boldsymbol{\delta\hat{x}}_{mk}^{+} \tag{5.4.20}$$

$$\boldsymbol{P}_{k}^{+} = \boldsymbol{P}_{m} \tag{5.4.21}$$

4. Inject the estimated error in the nominal state and correct the covariance through (5.1.18)

$$\hat{\boldsymbol{x}}_{k}^{+} = \hat{\boldsymbol{x}}_{k}^{-} \otimes \boldsymbol{\delta\hat{x}}_{k}^{+}$$

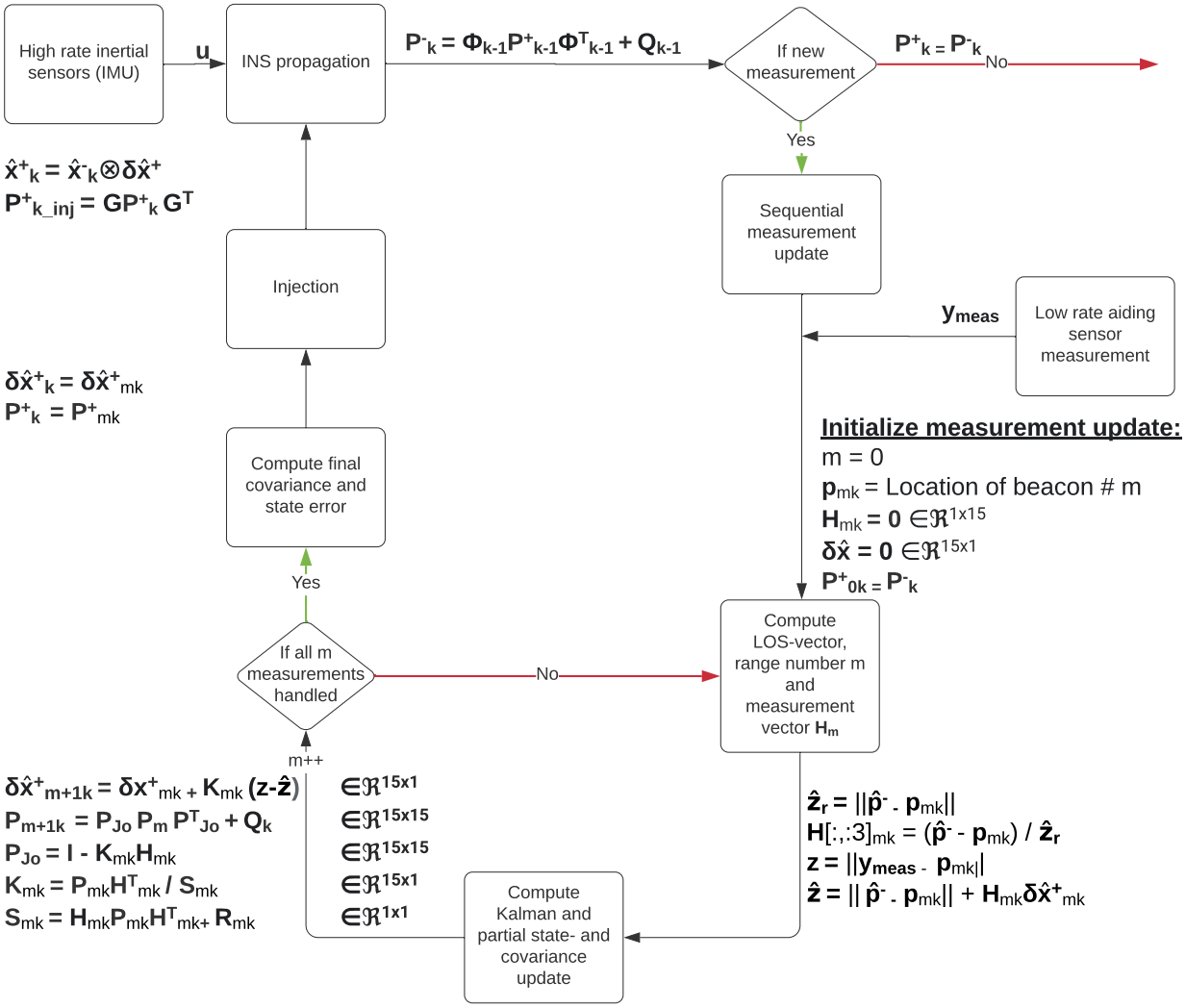


Figure 5.4.2: *Flowchart for the sequential ESKF algorithm*

5.4.3 UDU-factorized algorithm

Time update: The UD-filtered algorithm follows the same steps as the sequential and standard algorithm for state propagation, but differs in covariance propagation and measurement update. The discretization of the transition matrix and noise covariance matrix is done through Van Loans method, similar to the two other variations of the filter.

For each time step k, do:

1. Obtain $\boldsymbol{U}_{k-1}^{-}$ and $\boldsymbol{D}_{k-1}^{-}$ through UDU_factorization($\boldsymbol{P}_{k-1}^{+}$)
2. Define $\boldsymbol{Y_k}$ and $\tilde{\boldsymbol{D}}_k$

$$\boldsymbol{Y}_k = \begin{bmatrix} \boldsymbol{\Phi}_k \boldsymbol{U}_{k-1}^{+} & \boldsymbol{I} \end{bmatrix} \tag{5.4.22}$$

$$\tilde{\boldsymbol{D}}_k = \begin{bmatrix} \boldsymbol{D}_{k-1}^{+} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q}_k \end{bmatrix} \tag{5.4.23}$$

3. Obtain $\boldsymbol{U}_k^{-}$ and $\boldsymbol{D}_k^{-}$ from mod_gram_NASA($\boldsymbol{Y}_k, \tilde{\boldsymbol{D}}_k$)
4. Compute $\boldsymbol{P}_k^{-}$

$$\boldsymbol{P}_k^{-} = \boldsymbol{U}_k^{-} \boldsymbol{D}_k^{-} \boldsymbol{U}_k^{-T} \tag{5.4.24}$$

Measurement update: For each time step k when a new measurement is available, do:

1. Error state and measurement matrix $\boldsymbol{H}$ is set to zero and the "partial", or "initial" covariance matrix $\boldsymbol{P}_{0k}^{+} = \boldsymbol{P}_k^{-}$

$$\boldsymbol{\delta}\hat{\boldsymbol{x}}_{0k} = \boldsymbol{0} \tag{5.4.25}$$

$$\boldsymbol{P}_{0k}^{+} = \boldsymbol{P}_k^{-} \tag{5.4.26}$$

2. **for m in range(number of beacons:)**
   (a) Compute the estimated user-beacon range for this beacon

$$\hat{\boldsymbol{z}}_r = ||\hat{p}_u - p^{(m)}|| \tag{5.4.27}$$

   (b) Compute $\boldsymbol{H}_{mk}$:

$$\boldsymbol{H}_{mk} \in \mathbb{R}^{1x15} = \begin{bmatrix} (\hat{p}_u - p^{(m)})/\hat{\boldsymbol{z}}_r, & \boldsymbol{0_{12}} \end{bmatrix} \tag{5.4.28}$$

   (c) Compute $\hat{\boldsymbol{z}}$ for beacon m

$$\hat{\boldsymbol{z}} = ||\hat{p}^{u} - p^{(m)}|| + \boldsymbol{H}_{mk} \boldsymbol{\delta x}_{mk} \tag{5.4.29}$$

   (d) Compute $\boldsymbol{z}$ for beacon m

$$\boldsymbol{z} = ||\boldsymbol{y}_{meas} - p^{(m)}|| \tag{5.4.30}$$

   (e) Compute $\boldsymbol{S}_{mk}$

$$\boldsymbol{S}_{mk} = \boldsymbol{H}_{mk} \boldsymbol{P}_{mk}^{+} \boldsymbol{H}_{mk}^{T} + \boldsymbol{R}_{mk} \in \mathbb{R}^{1} \tag{5.4.31}$$

   (f) Factorize the partial covariance $\boldsymbol{P}_{mk}^{+}$ through UDU_factorization($\boldsymbol{P}_{mk}^{+}$) to obtain $\boldsymbol{U}_{mk}^{-}$ and $\boldsymbol{D}_{mk}^{-}$

(g) Compute $\tilde{\boldsymbol{P}}_m$

$$\tilde{\boldsymbol{P}}_m = \boldsymbol{D}_{mk}^- - \frac{1}{\boldsymbol{S}_{mk}}(\boldsymbol{D}_{mk}^- \boldsymbol{U}_{mk}^{-T} \boldsymbol{H}_{mk}^{-T})(\boldsymbol{D}_{mk}^- \boldsymbol{U}_{mk}^{-T} \boldsymbol{H}_{mk}^{-T})^T \tag{5.4.32}$$

(h) Factorize $\tilde{\boldsymbol{P}}_m$ through UDU_factorization($\tilde{\boldsymbol{P}}_m$) to obtain $\tilde{\boldsymbol{U}}_m$ and $\tilde{\boldsymbol{D}}_m$

(i) Compute $\boldsymbol{U}_{mk}^+$ and $\boldsymbol{D}_{mk}^+$

$$\boldsymbol{U}_{mk}^+ = \boldsymbol{U}_k^- \tilde{\boldsymbol{U}}_m \tag{5.4.33a}$$

$$\boldsymbol{D}_{mk}^+ = \tilde{\boldsymbol{D}}_m \tag{5.4.33b}$$

(j) Compute the kalman gain

$$\boldsymbol{K}_{mk} = \frac{\boldsymbol{U}_{mk}^+ \boldsymbol{D}_{mk}^+ \boldsymbol{U}_{mk}^{+T} \boldsymbol{H}_{mk}^T}{\boldsymbol{S}_{mk}} \in \mathbb{R}^{1x15} \tag{5.4.34}$$

(k) Compute the estimated error $\boldsymbol{\delta\hat{x}}_{mk+1}$ through the innovation of the given beacon and accumulate it to the estimated state error

$$\boldsymbol{\delta\hat{x}}_{mk+1} = \boldsymbol{\delta\hat{x}}_{mk} + \boldsymbol{K}_{mk} \cdot (\boldsymbol{z} - \hat{\boldsymbol{z}}) \in \mathbb{R}^{15\times 1} \tag{5.4.35}$$

(l) Compute the next partial covariance with the Joseph form of the measurement update from this beacon and accumulate it with the other

$$\boldsymbol{P}_{m+1k}^+ = (\boldsymbol{I} - \boldsymbol{K}_{mk}\boldsymbol{H}_{mk})\boldsymbol{U}_{mk}^+ \boldsymbol{D}_{mk}^+ \boldsymbol{U}_{mk}^{+T} (\boldsymbol{I} - \boldsymbol{K}_{mk}\boldsymbol{H}_{mk})^T + \boldsymbol{K}_{mk}\boldsymbol{R}_{mk}\boldsymbol{K}_{mk}^T \tag{5.4.36}$$

3. When all measurements have been handled, compute the final covariance and error state

$$\boldsymbol{\delta\hat{x}}_k^+ = \boldsymbol{\delta\hat{x}}_{mk}^+ \tag{5.4.37}$$

$$\boldsymbol{P}_k^+ = \boldsymbol{P}_{mk}^+ \tag{5.4.38}$$

4. Inject the estimated error in the nominal state and correct the covariance through (5.1.18)

$$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- \otimes \boldsymbol{\delta\hat{x}}_k^+$$

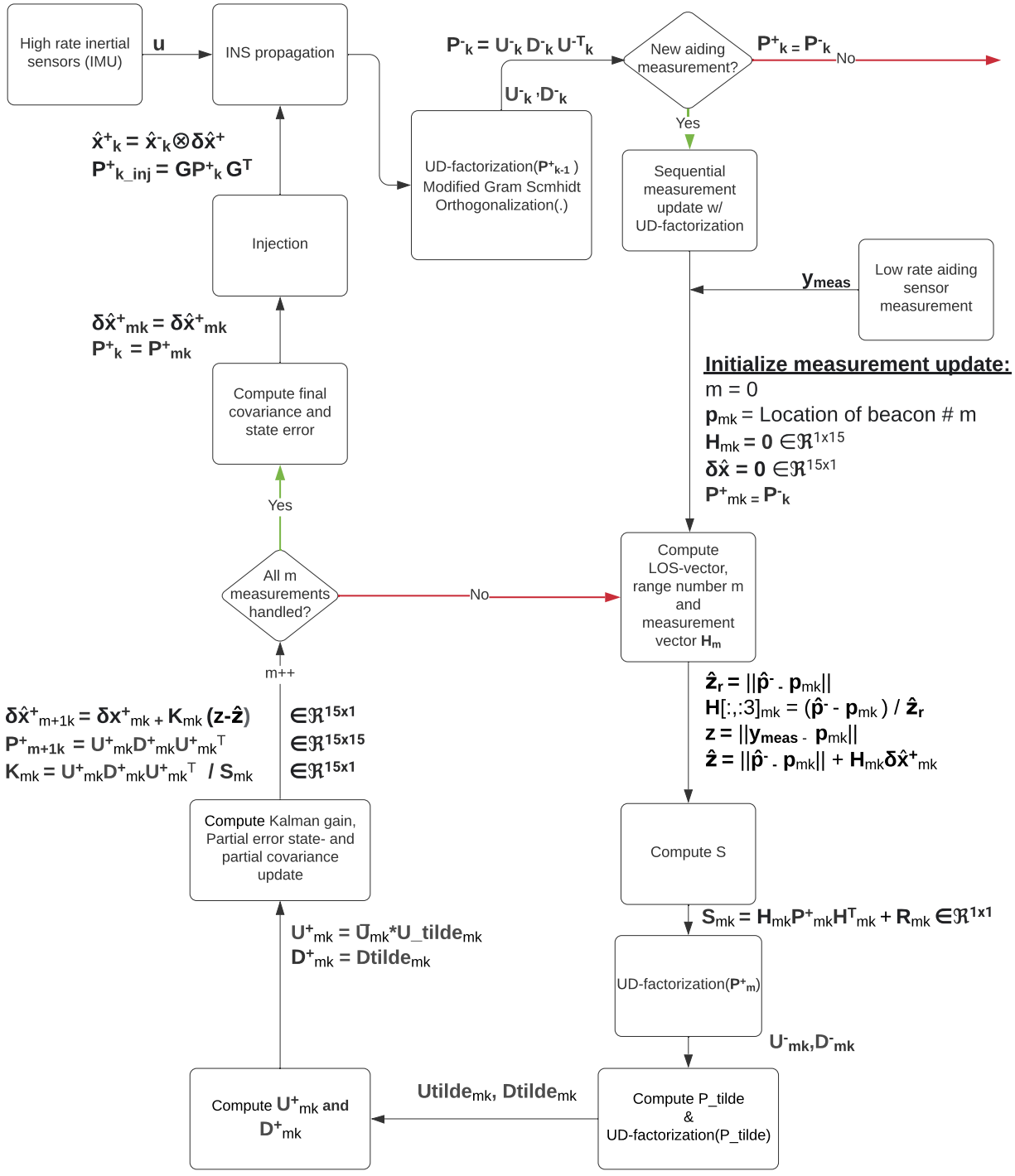


Figure 5.4.3: *Flowchart for the UD-factorized ESKF algorithm*

Part III

Results

Chapter 6

Evaluation

The run time evaluation and bench marking relative to the batch-wise measurement aiding strategy was performed using the python library *timeit* [16] to time the duration of the total simulation, the time-update steps and the measurement update steps. Other steps and modules such as initialization and returning of variables were not timed. The simulations run to evaluate the performance of the filters were run on a work station with the following relevant hardware:

Table 6.0.1: *Computer hardware*

CPU:	Intel i7-8700, 3.20GHz
RAM:	32.00 GB

Different hardware may result in a different run time result.

To remove discrepancies caused by the CPU being used by anything else, a minimum of background software were run at the same time as the simulation were performed. In addition to this, the number of samples to perform a proper time series analysis needs to be large enough that outliers will not dominate the outcome. Salzmann suggests in his experiments [6] (p.75) that the number of samples can be chosen with consideration of time constraints caused by hardware limitations and a rich enough sample size to perform the analysis on. With access of a work station with hardware as seen in Table 6.0.1, I decided on performing N = 200 simulations for the simulations durations seen below. For N lower than 100, variations caused by a small sample size started to show show.

The simulation durations chosen were

- 10 seconds
- 100 seconds
- 600 seconds
- 1000 seconds

The durations were chosen with consideration of the following factors. I wanted to see if shorter durations had any start up cost contributions which made the variations slower when comparing to the scaled up simulations. 100s were chosen as a duration close to when the model completes one loop around the eight-figure. 600s were chosen as a mid-range number illustrating a short 10 minute flight, and 1000s were chosen as a scale up comparison of 10 and 100s. 1000s were also chosen as the maximum due to the required time to perform the simulations and time them, which added up to roughly 25 real time hours for all three filters when using 200 simulations.

Each configuration were run on the hardware listed in Table 6.0.1. The final results for m = 15 and m = 30 measurements are presented in Section 6.1 and Section 6.2. All run times can be found in the text file *"runtimes.txt"* available in the GitHub repository.

6.1 Results for m = 15 measurements

The following figures displays the distribution of the results illustrated by scatter plot of the runs with some jitter and a overlaying box-plot on top for m = 15 measurements.

6.1.1 Distribution of run times

Figure 6.1.1 illustrates the respective timing of the various duration for the three filter variations; the total run time, time-update run time and measurement-update run time. Figure 6.1.2 presents the results from the two most similar filter variations.

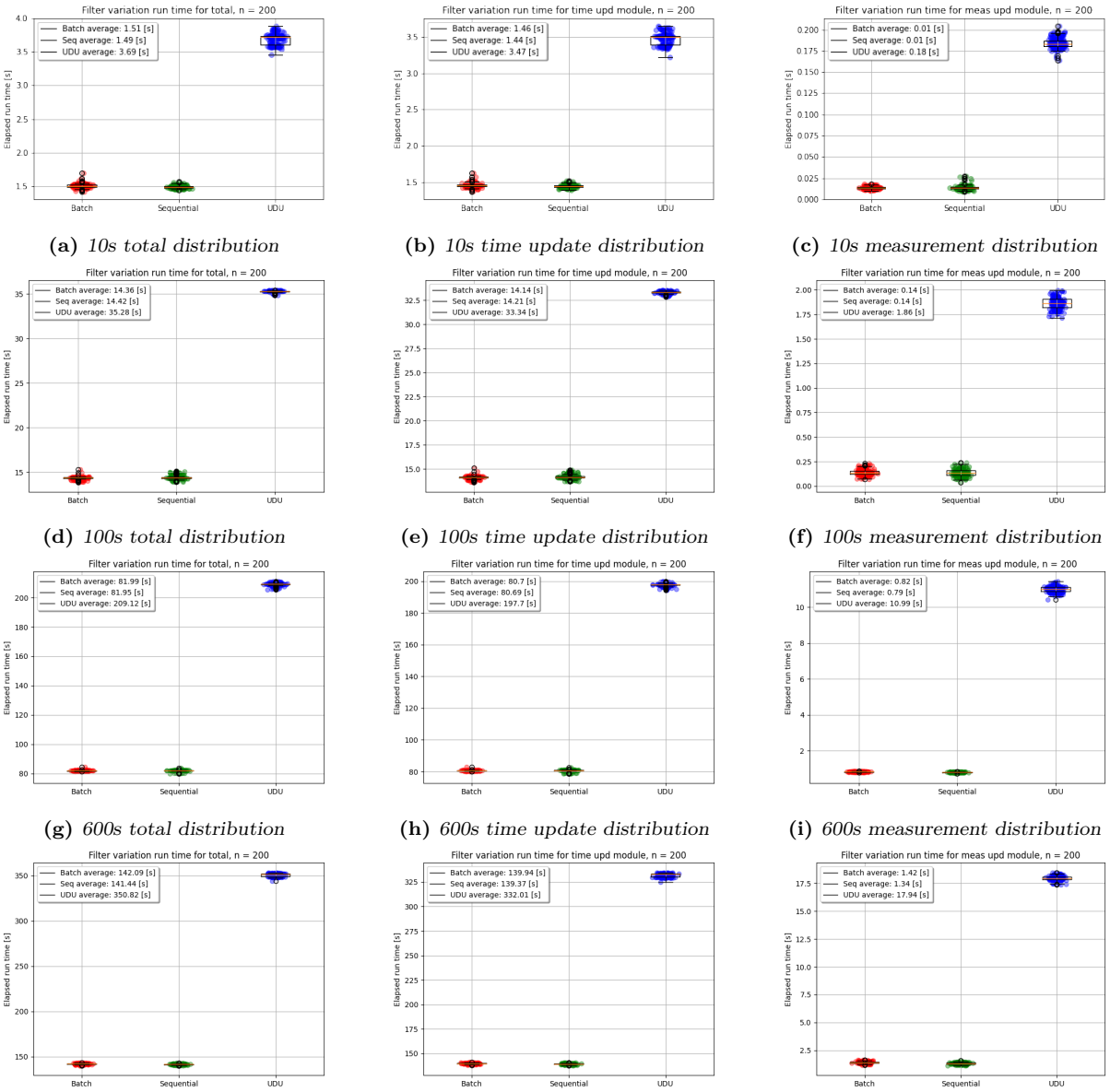
Each row of subfigures represents one duration with the plots in the order of total run time, time-update run time and measurement-update run time. The number n in the subplots titles represents number of simulations done and in turn the number of data points in each scatter and box plot for each filter variation. The number m represents the number of aiding measurements in the measurement vector. The legend displays the average for that module, computed with a two point decimal rounding.

The figures shows that the simulation durations are directly related to the elapsed run time for all modules in all variations. Both the time update run time and measurement update run time occupies a similar portion of the total elapsed run time for each duration. This portion is roughly 98% for the prediction module in the batch wise and sequential variation, 95% for the time update module in the UDU-variation, and 1% for the measurement update in the batch wise and sequential variation and roughly 5% for the UDU-measurement update. Figure 6.1.1 shows that the UDU-variation are slower in all three measured timings, while the batch wise and sequential variations are similar in all three measured timings. Examining Figure 6.1.2, we observe a more nuanced difference between the batch wise and the sequential variation. Both are similar in total run time and the elapsed run time for the time update, the ladder which they share the same equations for. While for the measurement update there is a decrease in average elapsed run time for the sequential variation.

Table 6.1.1 displays the average time elapsed for the specified duration the average percent that the time step module occupies when the simulation is run, and the average percent that the measurement step module occupies of the n simulations performed.

The simulation durations presented are the scaled duration 10, 100, 600 and 1000 seconds. From Table 6.1.1 we observe the same trend as in Figure 6.1.1 and Figure 6.1.2, where the elapsed simulation run times are directly proportional to the simulation run time. Increasing the simulation duration with a factor of 10 yields a run time increased with a factor of 10.

From Table 6.1.1 we observe a similar average elapsed run time for the batch wise and sequential variation in total run time and time update run time. From the same table, we also see that measurement update run time for the sequential variation is faster than the run time for the batch wise update by roughly 3.5%.



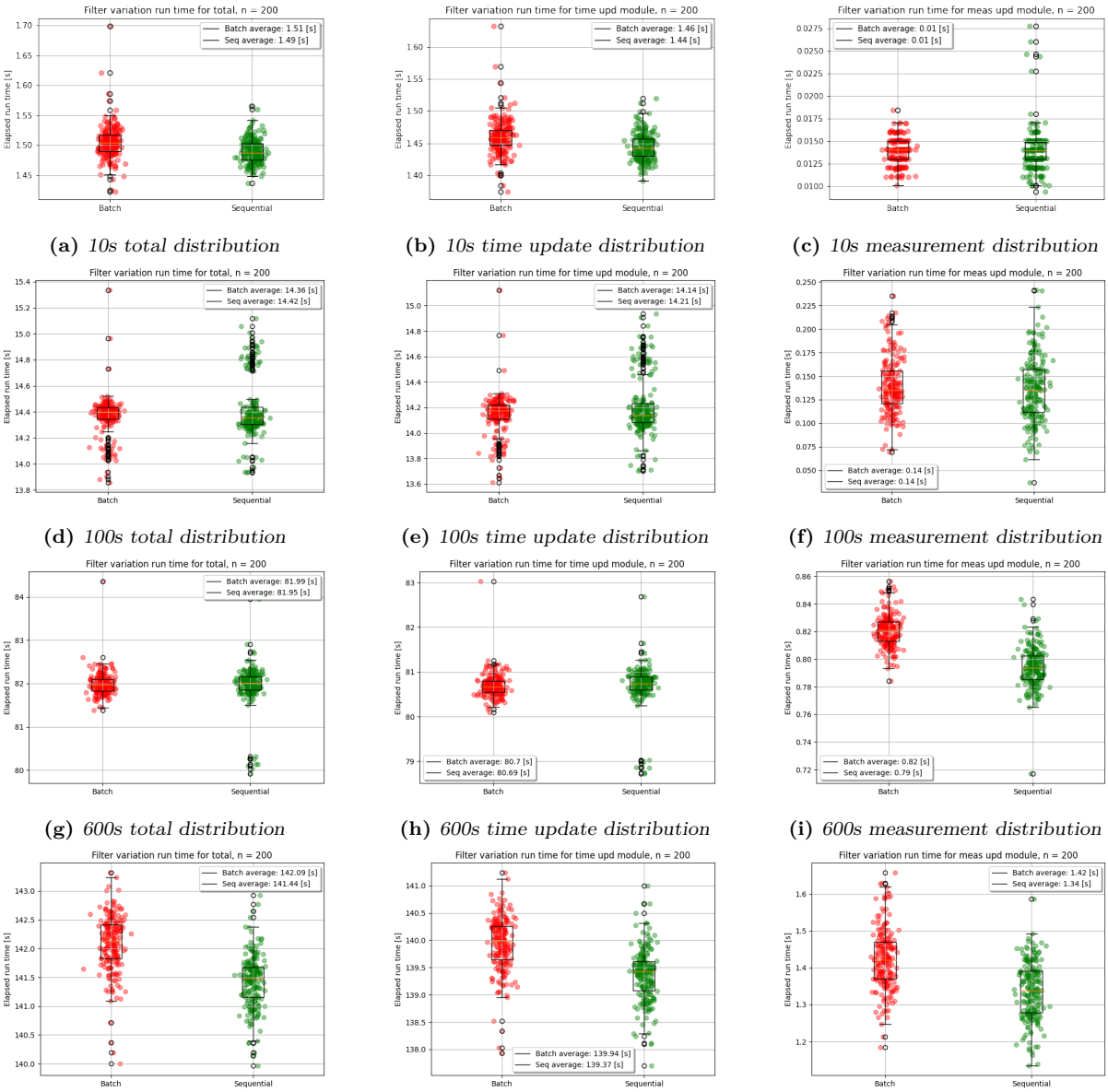
(a) *10s total distribution* **(b)** *10s time update distribution* **(c)** *10s measurement distribution*

(d) *100s total distribution* **(e)** *100s time update distribution* **(f)** *100s measurement distribution*

(g) *600s total distribution* **(h)** *600s time update distribution* **(i)** *600s measurement distribution*

(j) *1000s total distribution* **(k)** *1000s time update distribution* **(l)** *1000s measurement distribution*

Figure 6.1.1: *Comparison of batch, sequential and UDU-variation run time distribution*



(a) *10s total distribution* **(b)** *10s time update distribution* **(c)** *10s measurement distribution*

(d) *100s total distribution* **(e)** *100s time update distribution* **(f)** *100s measurement distribution*

(g) *600s total distribution* **(h)** *600s time update distribution* **(i)** *600s measurement distribution*

(j) *1000s total distribution* **(k)** *1000s time update distribution* **(l)** *1000s measurement distribution*

Figure 6.1.2: *Comparison of batch and sequential run time distribution*

Table 6.1.1: *Timing results for simulation of batch-wise and sequential computation using 15 beacons with varying duration.*

	Duration [s]	**Avg time [s]**	**Time upd [s]**	**Meas upd [s]**
Batch	10	1.51	1.46	0.014
	100	14.36	14.14	0.14
	600	81.99	80.70	0.82
	1000	142.09	139.94	1.42
Seq	10	1.49	1.445	0.014
	100	14.42	14.21	0.14
	600	81.95	80.69	0.79
	1000	141.43	139.37	1.34
UDU	10	3.69	3.47	0.18
	100	35.28	33.34	1.86
	600	209.12	197.70	10.99
	1000	350.82	332.01	17.94

6.1.2 Numerical evaluation

This section contains the numerical run-time evaluation in table form for the case with m = 15 measurements.

Table 6.1.2 displays the relative speedup in percentage between the filter variations. As reflected in Figure 6.1.1, we observe that the batch wise and sequential variation share a similar run time, while the UDU-variation is 145 to 155 % slower than the other two variations. Examining Table 6.1.3, which displays the relative speed up of the average elapsed run time for the time update, we again see the same similarity reflected in the subfigures representing the time update in Figure 6.1.1. Here the sequential variation is very close to the batch-wise reference, while the UDU-variation is about 1.4 times slower relative to the reference. Table 6.1.4 displays the relative speedup in percentage of the time update module and measurement update respectively. This module displays the largest spread between the variations. The UDU-variation is roughly 1250% slower than the batch wise and sequential variation. The sequential measurement update is similar for the 10 second duration, which is caused by rounding errors due to the number of decimals used when computing the average elapsed run time, causing the longer durations to be more representative. The longer durations display up to a 4.25% relative speed up for the sequential variation, meaning that the sequential measurement update is faster than the batch wise measurement update with a measurement vector consisting of 15 aiding measurements.

Table 6.1.2: *Relative speedup comparison between the three variations given in percentage [%]. Negative value signifies that the left hand method is faster than the cross-listed variation on top.*

Duration [s]		**Batch** [%]	**Sequential**[%]	**UDU** [%]
10			0.997	-145.50
100			-0.40	-145.61
600	**Batch**	N/A	0.043	-144.64
1000			0.46	-146.90
10		-0.997		-147.92
100		0.4		-144.64
600	**Sequential**	0.043	N/A	-155.17
1000		-0.46		-148.04
10		145.50	147.92	
100		145.61	144.64	
600	**UDU**	144.64	155.17	N/A
1000		146.90	148.04	

Table 6.1.3: *Relative speedup comparison between the time update modules of the three variations given in percentage [%]. Negative value signifies that the left hand method is faster than the cross-listed variation on top.*

Duration [s]		**Batch** [%]	**Sequential**[%]	**UDU** [%]
10			1.03	-137.95
100			-0.43	-135.72
600	**Batch**	N/A	0.012	-145.00
1000			-0.078	-137.26
10		-1.03		-140.42
100		0.43		-134.71
600	**Sequential**	-0.012	N/A	-145.03
1000		0.078		-138.22
10		137.95	140.42	
100		135.72	134.71	
600	**UDU**	145.00	145.03	N/A
1000		137.26	138.22	

Table 6.1.4: *Relative speedup comparison between the measurement update modules of the three variations given in percentage [%]. Negative value signifies that the left hand method is faster than the cross-listed variation on top.*

Duration [s]		**Batch** [%]	**Sequential**[%]	**UDU** [%]
10			0.0	-1214.29
100			3.57	-1229.29
600	**Batch**	N/A	3.29	-1238.86
1000			4.25	-1160.79
10		0.0		-1214.29
100		-3.57		-1278.52
600	**Sequential**	-3.29	N/A	-1284.38
1000		-4.25		-1242.89
10		1214.29	1214.29	
100		1229.29	1278.52	
600	**UDU**	1238.86	1284.38	N/A
1000		1160.79	1242.89	

6.1.3 Filter variation performance and accuracy for m = 15

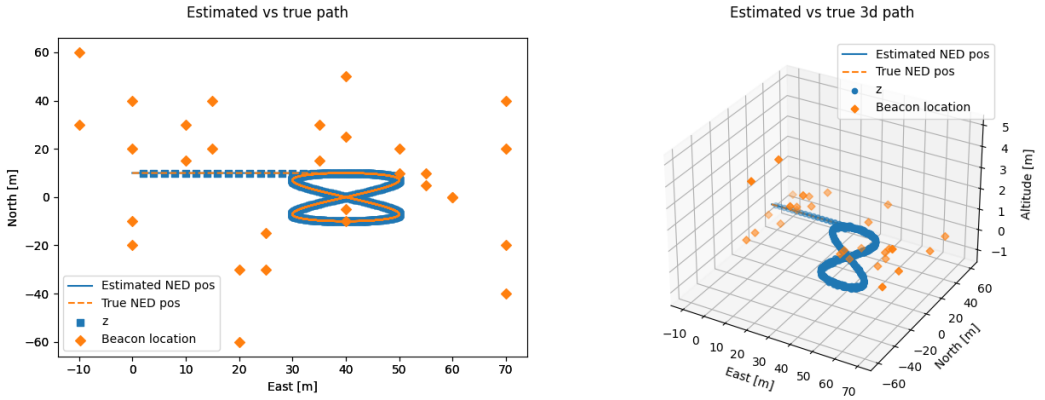
This section displays the performance and accuracy of the Kalman filter variations for m = 15. All three subsections shares the same setup with plots displaying their paths, and plots displaying the state error of the 15 states in the model. The following have been made with the results after running one simulation with the duration of 1000 seconds for m = 15 measurements. In Section 6.1.3, Table 6.1.5 the RMSE comparison for the 15 states in the three variations can be seen, together with a comparison of the covariance in Table 6.1.6. In the trajectory figures, all beacons are plotted, but only the 15 closest were toggled to give the position aiding measurement. The relative comparison r, seen in Table 6.1.6, is computed with (6.1.1) with a 3 point float rounding.

$$r = \frac{\text{Variation} - \text{Reference}}{\text{Reference}} \cdot 100\% \tag{6.1.1}$$

Standard ESKF

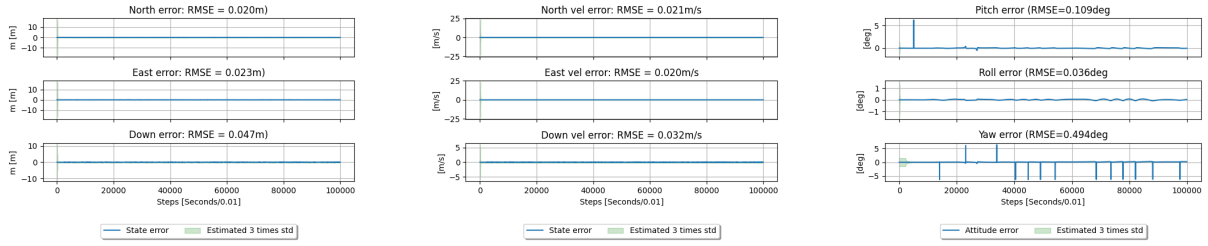
Figure 6.1.3 displays the 2d and 3d path for the standard batch-wise ESKF, and Figure 6.1.4 displays the state estimation error with the states three times standard deviation plotted on top.

Examining the trajectory plots we observe that the standard ESKF manages to estimate the trajectory with close to 0 error. This is further supported by the RMSE in Figure 6.1.4, which displays a 0.02m to 0.047m root mean square error for the NED position. All other state errors are within the expected three times standard deviation of where the values should be, with the exception of the gyro rate bias error.

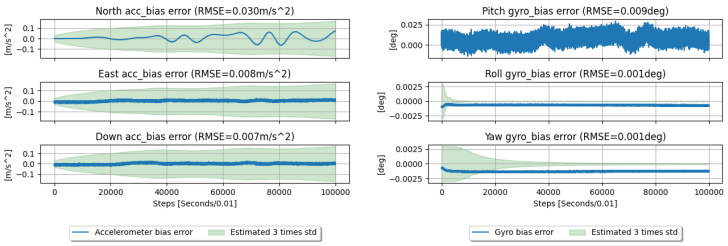


(a) *2D path with beacons and measurement* **(b)** *3D path with beacons and measurement*

Figure 6.1.3: *2D and 3D path for standard ESKF*



(a) *Position error* **(b)** *Velocity error* **(c)** *Attitude error*



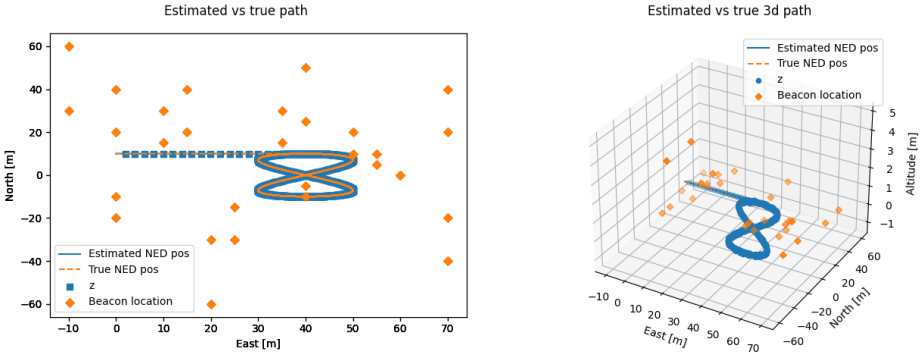
(d) *Accelerometer bias error* **(e)** *Gyro rate bias error*

Figure 6.1.4: *State error plot with 3x sigma for standard ESKF*

Sequential variation

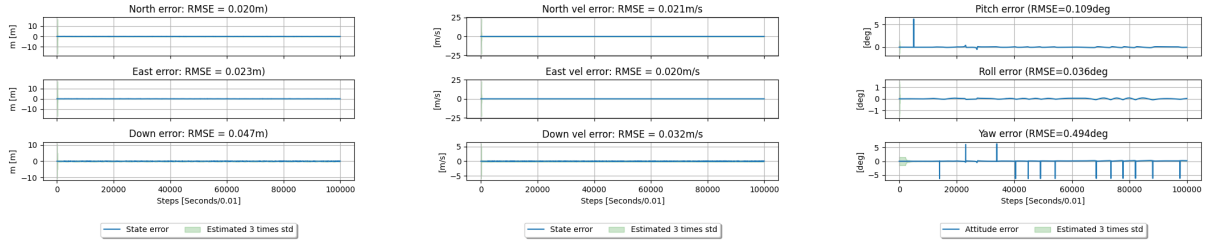
Figure 6.1.5 displays the 2d and 3d path for the sequential ESKF, and Figure 6.1.6 displays the state estimation error with the states three times standard deviation plotted on top.

Examining the trajectory plots we observe that the standard ESKF manages to estimate the trajectory with close to 0 error. This is further supported by the RMSE in Figure 6.1.6, which displays a 0.02m to 0.047m root mean square error for the NED position. All other state errors are within the expected three times standard deviation of where the values should be, with the exception of the gyro rate bias error.

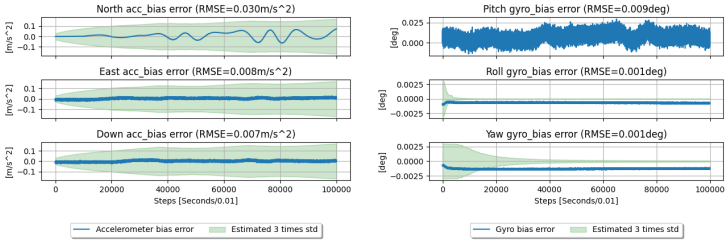


(a) *2D path with beacons and measurement* **(b)** *3D path with beacons and measurement*

Figure 6.1.5: *2D and 3D path for sequential ESKF*



(a) *Position error* **(b)** *Velocity error* **(c)** *Attitude error*



(d) *Accelerometer bias error* **(e)** *Gyro rate bias error*

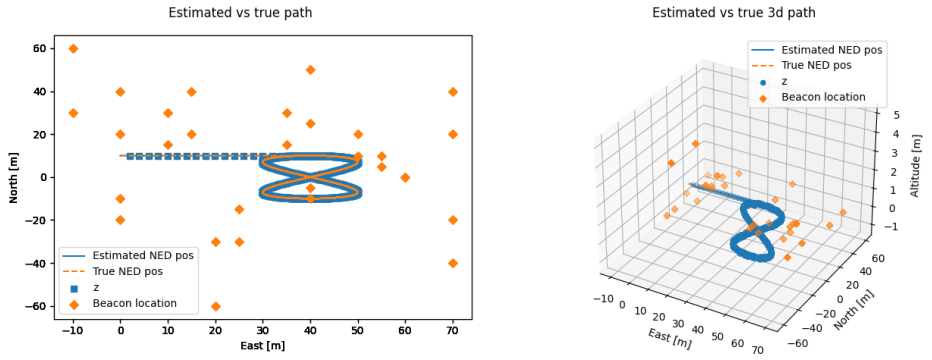
Figure 6.1.6: *State error plot with 3x sigma for sequential ESKF*

UDU-factorized variation

Figure 6.1.7 displays the 2d and 3d path for the UDU-factorized ESKF, and Figure 6.1.8 displays the state estimation error with the states three times standard deviation plotted on top.

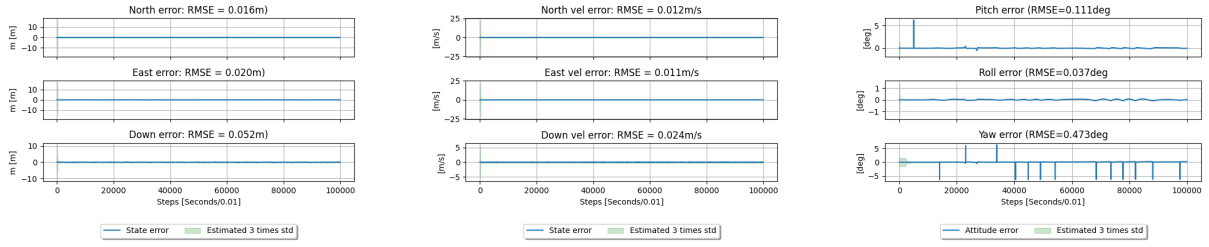
Examining the trajectory plots we observe that the standard ESKF manages to estimate the trajectory with close to 0 error.

This is further supported by the RMSE in Figure 6.1.8, which displays a 0.02m to 0.047m root mean square error for the NED position. All other state errors are within the expected three times standard deviation of where the values should be, with the exception of the gyro rate bias error.

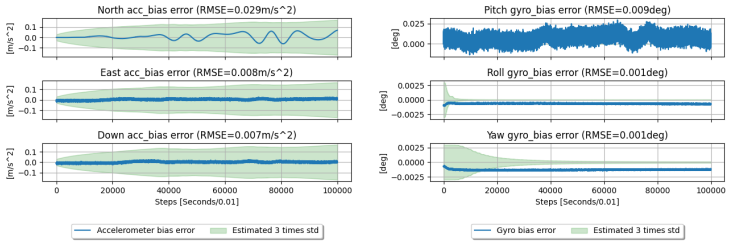


(a) *2D path with beacons and measurement* **(b)** *3D path with beacons and measurement*

Figure 6.1.7: *2D and 3D path for sequential ESKF*



(a) *Position error* **(b)** *Velocity error* **(c)** *Attitude error*



(d) *Accelerometer bias error* **(e)** *Gyro rate bias error*

Figure 6.1.8: *State error plot with 3x sigma for UD-factorized ESKF*

Numerical performance comparison, m=15

Extracting the RMSE values from Figure 6.1.4, Figure 6.1.6 and Figure 6.1.8 we get the result seen in Table 6.1.5 with a 3 point float rounding. Here, the sequential and UDU-variations have been compared against the batch-wise state RMSEs to observe the difference between the

variations. The sequential variation displayed a numerical identical result versus the batch vise variation. The UDU-variation displayed a small difference with a magnitude of 10e-3 between the state RMSEs. Seven states displayed a smaller state RMSE, three states displayed a larger error, and fire displayed a numerical identical RMSE to versus the batch-wise variation.

The final covariance result in Table 6.1.6 shows that the sequential and batch-wise variations also are identical in the final covariance diagonal after a simulation duration of 1000 seconds. The UDU-factorized variation covariance is similar in magnitude to the batch-wise reference, but displays a large improvement when comparing the difference between the reference in relative difference given by (6.1.1). The 9 first states displays a improvement ranging from 0.05% to 89.63% percent between the UDU-variation and the reference.

Table 6.1.5: *RMSE values for states in filter variations with m = 15 measurements*

State number	**Batch**	**Sequential**	Vs Batch	**UDU**	vs Batch	**[Unit]**
1	0.020	0.020	0	0.016	0.004	[m]
2	0.023	0.023	0	0.020	0.003	[m]
3	0.047	0.047	0	0.052	-0.005	[m]
4	0.021	0.021	0	0.012	0.009	[m/s]
5	0.020	0.020	0	0.011	0.009	[m/s]
6	0.032	0.032	0	0.024	0.008	[m/s]
7	0.109	0.109	0	0.111	-0.002	[deg]
8	0.036	0.036	0	0.037	-0.001	[deg]
9	0.494	0.494	0	0.473	0.021	[deg]
10	0.030	0.030	0	0.029	0.001	[m/s^2]
11	0.008	0.008	0	0.008	0	[[m/s^2]
12	0.007	0.007	0	0.007	0	[[m/s^2]
13	0.009	0.009	0	0.009	0	[deg/s]
14	0.001	0.001	0	0.001	0	[deg/s]
15	0.001	0.001	0	0.001	0	[deg/s]

The final covariance diagonal elements for the run with 15 measurements displayed similar results.

Table 6.1.6: *Final state variance values for states in filter variations with m = 15 measurements*

Element number	**Batch**	**Sequential**	Vs Batch	**UDU**	vs Batch	%
1	2.285e-5	2.285e-5	0	1.205e-5	1.108e-2	-89.63
2	1.879e-5	1.879e-5	0	1.099e-5	7.8e-4	-70.97
3	1.762e-3	1.762e-3	0	9.872e-4	7.75e-3	-78.48
4	1.083e-3	1.083e-3	0	1.043e-3	4.0e-5	-3.69
5	1.076e-3	1.076e-3	0	1.045e-3	3.1e-5	-40.79
6	3.095e-3	3.095e-3	0	2.538e-3	0.557e-3	-18.00
7	1.621e-6	1.621e-6	0	1.619e-6	2e-9	-0.12
8	1.647e-6	1.647e-6	0	1.646e-6	1e-9	-0.06
9	2.012e-4	2.012e-4	0	2.011e-4	1e-7	-0.05
10	3.189e-3	3.189e-3	0	3.189e-3	0	0
11	3.070e-3	3.070e-3	0	3.070e-3	0	0
12	3.058e-3	3.058e-3	0	3.058e-3	0	0
13	4.456e-10	4.456e-10	0	4.456e-10	0	0
14	4.474e-10	4.47e-10	0	4.473e-10	0	0
15	1.59e-9	1.592e-9	0	1.592e-9	0	0

6.2 Results for m = 30 measurements

The experiment were performed with n = 200 simlations for the simulation durations 100 and 1000 seconds. Figure 6.2.1 in Section 6.2.1 displays the distribution of the results illustrated by scatter plot of the runs with jitter and a overlaying box-plot on top. The average run time values are written in the sub-plot legend. Table 6.2.1 shows the module elapsed run times for 100 and 1000 second duration.

Table 6.2.2, Table 6.2.3 and Table 6.2.3 in Section 6.2.1 displays the relative speed difference between the filter variations.

The diagonal covariance elements in Table 6.2.6 displays the estimated state variance from the final time step in a 1000s simulation.

6.2.1 Distribution of run times

In Figure 6.2.1, the rows illustrates the following:

In row 1, Figure 6.2.1a, Figure 6.2.1b and Figure 6.2.1c displays the distribution of elapsed total-, time update- and measurement update run time between all three variations of the ESKF for a simulation duration of 100 seconds.

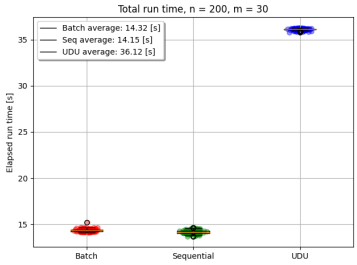
In row 2, Figure 6.2.1d, Figure 6.2.1e and Figure 6.2.1f displays the distribution of elapsed total-, time update- and measurement update run time between the batch-wise and sequential variations for a simulation duration of 100 seconds.

In row 3, Figure 6.2.1g, Figure 6.2.1h and Figure 6.2.1i displays the distribution of elapsed total-, time update- and measurement update run time between all three variations of the ESKF for a simulation duration of 1000 seconds.

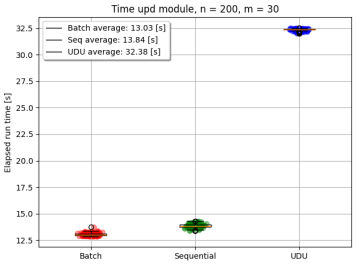
In row 4, Figure 6.2.1j, Figure 6.2.1k and Figure 6.2.1l displays the distribution of elapsed total-, time update- and measurement update run time between all filters for a simulation duration of 1000 seconds.

The simulation durations presented are the scaled duration 100 and 1000 seconds. From Table 6.1.1 we observe the same trend as in Figure 6.2.1, where the elapsed simulation run times are directly proportional to the simulation run time. Increasing the simulation duration with a factor of 10 yields a run time roughly increased with a factor of 10.

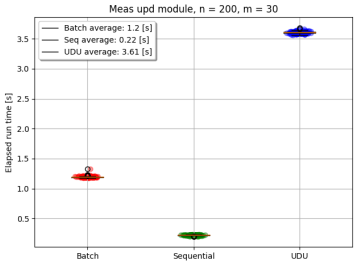
From Table 6.2.1 we observe a similar average elapsed run time for the batch wise and sequential variation in total run time and time update run time. The measurement update run time for the sequential variation is faster than the run time for the batch wise update by roughly 4%.



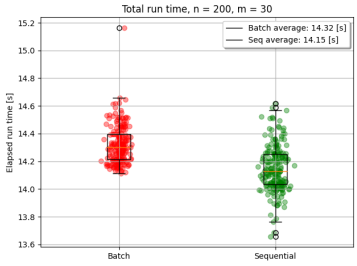
(a) *100s total distribution*



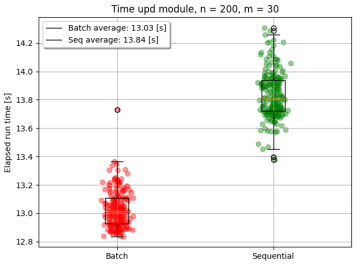
(b) *100s time update distribution*



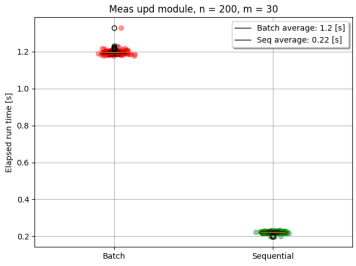
(c) *100s meas. upd. distribution*



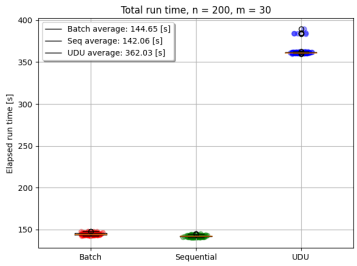
(d) *100s total distribution*



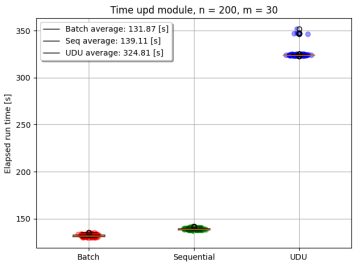
(e) *100s time update distribution*



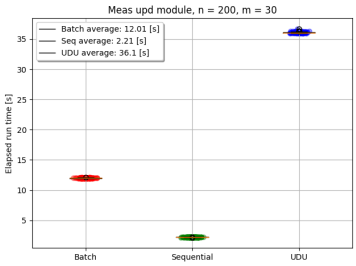
(f) *100s meas. upd. distribution*



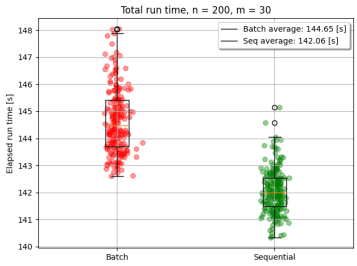
(g) *1000s total distribution*



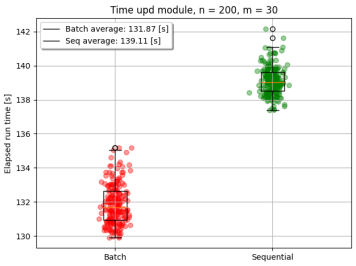
(h) *1000s time update distribution*



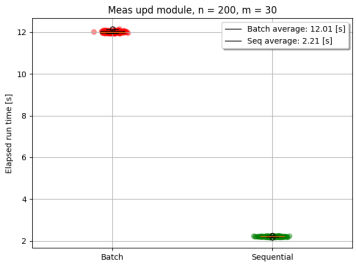
(i) *1000s meas. upd. distribution*



(j) *1000s total distribution*



(k) *1000s time update distribution*



(l) *1000s meas. upd. distribution*

Figure 6.2.1: *Comparison of batch, sequential and UDU-variation run time distribution for m = 30*

Table 6.2.1: *Timing results for simulation of batch-wise and sequential computation using 30 beacons.*

	Duration [s]	Avg time [s]	Time upd [s]	Meas upd [s]
Batch	100	14.32	13.03	1.20
	1000	144.64	131.87	12.01
Seq	100	14.15	13.84	0.22
	1000	142.06	139.11	2.21
UDU	100	36.12	32.38	3.61
	1000	362.03	324.81	36.00

6.2.2 Numerical evaluation

This section contains the findings in table form for the case with m = 15 measurements.

Table 6.2.2 displays the relative speedup in percentage between the filter variations. As reflected in Figure 6.2.1, we observe that the batch wise and sequential variation share a similar run time, while the UDU-variation is 145 to 155 % slower than the other two variations. Examining Table 6.2.3, which displays the relative speed up of the time update run time, we note the same trend as for total run time.

Table 6.2.2: *Relative speedup comparison between the three variations given in percentage [%]. Negative value signifies that the left hand method is faster than the cross-listed variation on top.*

Duration [s]		**Batch** [%]	**Sequential**[%]	**UDU** [%]
100	**Batch**	N/A	1.21	-152.21
1000			1.79	-150.28
100	**Sequential**	-1.21	N/A	-154.84
1000		-1.79		-155.27
100	**UDU**	152.21	155.30	N/A
1000		150.28	154.84	

Table 6.2.4 displays the relative speedup in percentage of the time update module and measurement update respectively. This module displays the largest spread between the variations. The UDU-variation is roughly 1250% slower than the batch wise and sequential variation. The sequential measurement update is similar for the 10 second duration, which is caused by rounding errors due to the number of decimals used when computing the average elapsed run time. More representative is the longer durations, illustrating a 3.5% relative speed up for the sequential variation, meaning that the sequential measurement update is faster than the batch wise measurement update.

Table 6.2.3: *Relative speedup comparison between the time update modules of the three variations given in percentage [%]. Negative value signifies that the left hand method is faster than the cross-listed variation on top.*

Duration [s]		**Batch** [%]	**Sequential**[%]	**UDU** [%]
100	**Batch**	N/A	-6.17	-148.50
1000			-5.49	-146.32
100	**Sequential**	6.17	N/A	-134.06
1000		5.67		-133.50
100	**UDU**	148.50	134.06	N/A
1000		146.32	133.50	

Table 6.2.4: *Relative speedup comparison between the measurement update modules of the three variations given in percentage [%]. Negative value signifies that the left hand method is faster than the cross-listed variation on top.*

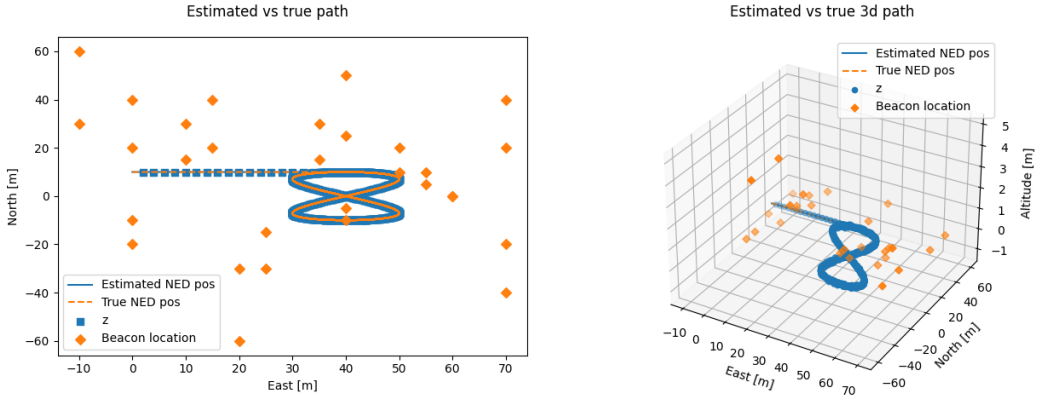
Duration [s]		**Batch** [%]	**Sequential**[%]	**UDU** [%]
100	**Batch**	N/A	81.65	-200.50
1000			80.84	-201.96
100	**Sequential**	-81.21	N/A	-1532.13
1000		- 81.65		-1536.96
100	**UDU**	201.59	1532.13	N/A
1000		200.50	1536.96	

6.2.3 Filter variation accuracy for m = 30

Batch-wise ESKF

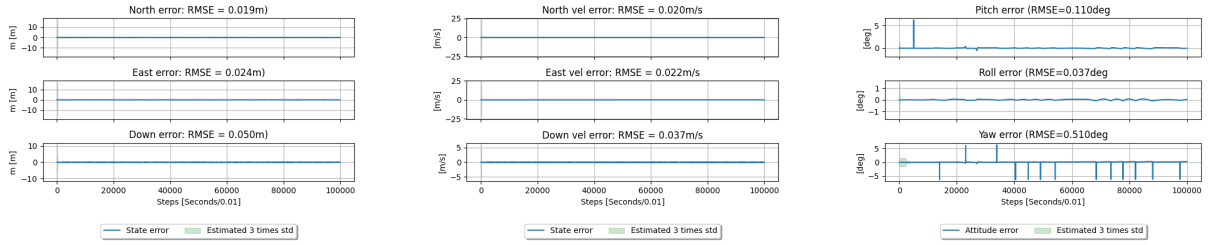
Figure 6.2.2 displays the 2d and 3d path for the standard batch-wise ESKF, and Figure 6.2.3 displays the state estimation error with the states three times standard deviation plotted on top.

Examining the trajectory plots we observe that the standard ESKF manages to estimate the trajectory with close to 0 error. This is further supported by the RMSE in Figure 6.1.4, which displays a 0.02m to 0.047m root mean square error for the NED position. All other state errors are within the expected three times standard deviation of where the values should be, with the exception of the gyro rate bias error.

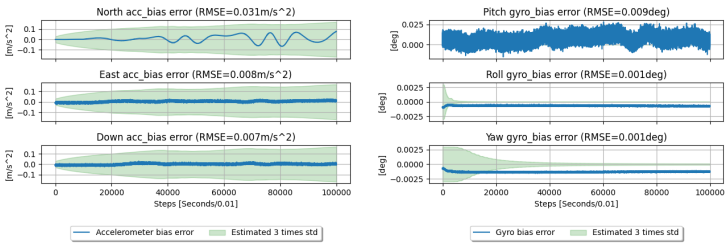


(a) *2D path with beacons and measurement* **(b)** *3D path with beacons and measurement*

Figure 6.2.2: *2D and 3D path for standard ESKF with 30 measurements in update*



(a) *Position error* **(b)** *Velocity error* **(c)** *Attitude error*



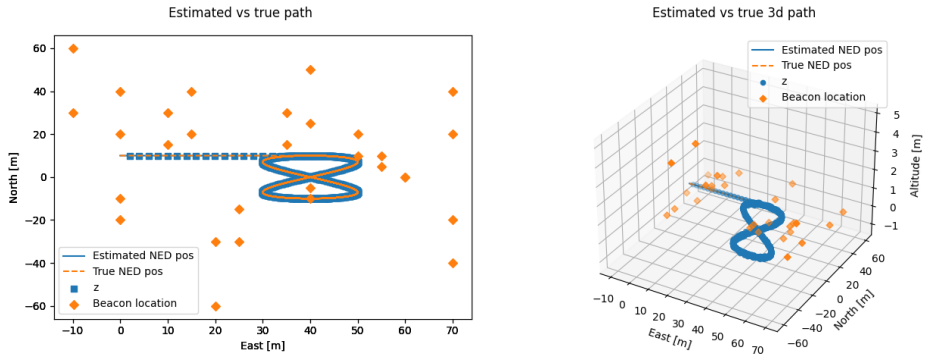
(d) *Accelerometer bias error* **(e)** *Gyro rate bias error*

Figure 6.2.3: *State error plot with 3x sigma for standard ESKF with 30 measurements in update*

Sequential variation

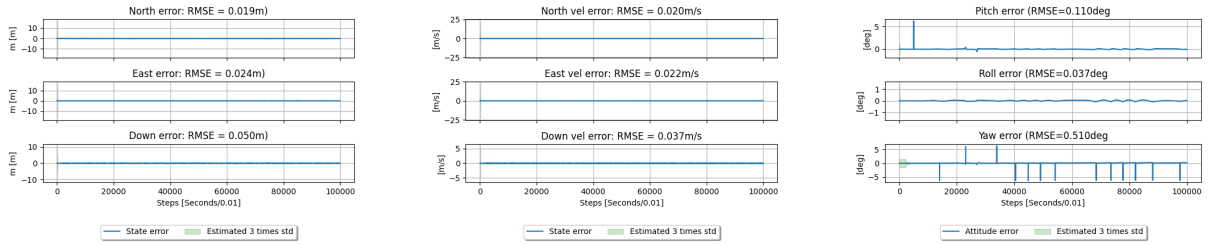
Figure 6.2.4 displays the 2d and 3d path for the sequential ESKF, and Figure 6.2.5 displays the state estimation error with the states three times standard deviation plotted on top.

Examining the trajectory plots we observe that the standard ESKF manages to estimate the trajectory with close to 0 error. This is further supported by the RMSE in Figure 6.2.5, which displays a 0.02m to 0.047m root mean square error for the NED position. All other state errors are within the expected three times standard deviation of where the values should be, with the exception of the gyro rate bias error.

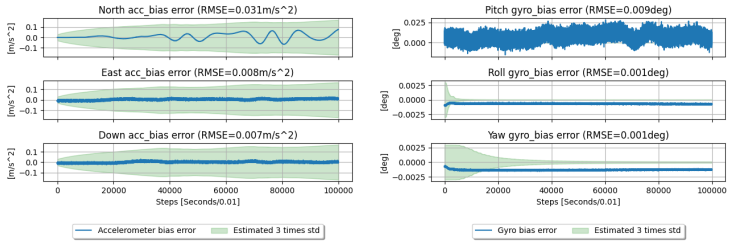


(a) *2D path with beacons and measurement* **(b)** *3D path with beacons and measurement*

Figure 6.2.4: *2D and 3D path for sequential ESKF with 30 measurements in update*



(a) *Position error* **(b)** *Velocity error* **(c)** *Attitude error*



(d) *Accelerometer bias error* **(e)** *Gyro rate bias error*

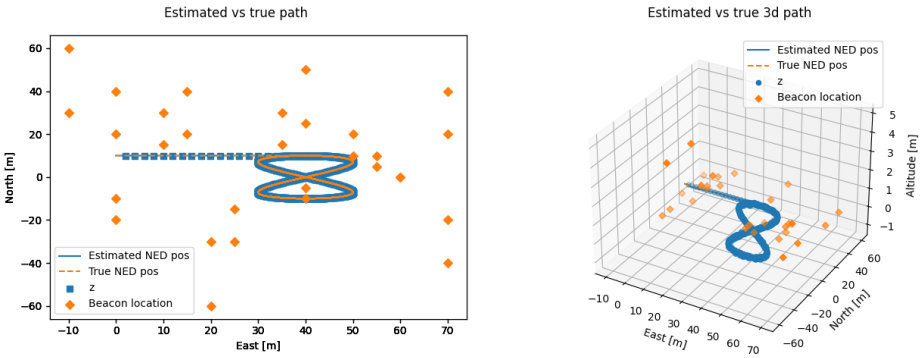
Figure 6.2.5: *State error plot with 3x sigma for sequential ESKF with 30 measurements in update*

UDU-factorized variation

Figure 6.2.6 displays the 2d and 3d path for the UDU-factorized ESKF, and Figure 6.2.7 displays the state estimation error with the states three times standard deviation plotted on top.

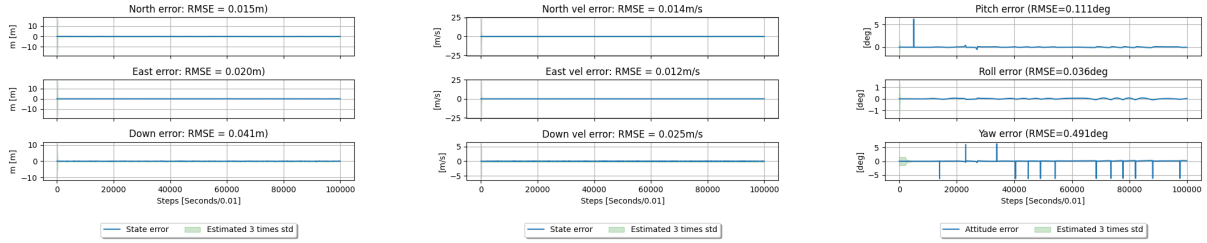
Examining the trajectory plots we observe that the standard ESKF manages to estimate the trajectory with close to 0 error. This can be seen again by the RMSE in Figure 6.2.7, which

displays a 0.015m to 0.041m root mean square error for the NED position. All other state errors are within the expected three times standard deviation of where the values should be, with the exception of the gyro rate bias error.

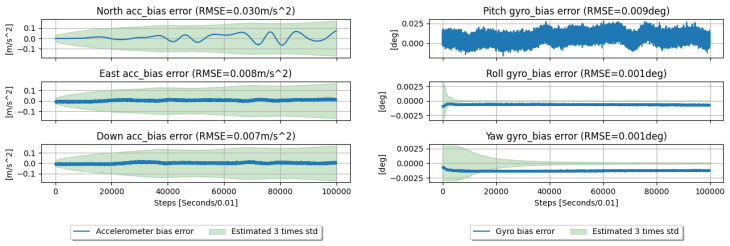


(a) *2D path with beacons and measurement* **(b)** *3D path with beacons and measurement*

Figure 6.2.6: *2D and 3D path for sequential ESKF*



(a) *Position error* **(b)** *Velocity error* **(c)** *Attitude error*



(d) *Accelerometer bias error* **(e)** *Gyro rate bias error*

Figure 6.2.7: *State error plot with 3x sigma for UD-factorized ESKF with 30 measurements in update*

Numerical performance comparison, m=30

Extracting the RMSE values from Figure 6.2.3, Figure 6.2.5 and Figure 6.2.7 we get the result seen in Table 6.2.5 with a 3 point float rounding. Here, the sequential and UDU-variations have been compared against the batch-wise state RMSEs to observe the difference between the variations. The sequential variation displayed a numerical identical result versus the batch vise variation. The UDU-variation displayed a small difference with a magnitude of 10e-3 between the state RMSEs. Seven states displayed a smaller state RMSE, 1 states displayed a larger error, and nine displayed a numerical identical RMSE to versus the batch-wise variation. The rest were identical.

The final covariance result in Table 6.2.6 shows that the sequential and batch-wise variations also are identical in the final covariance diagonal after a simulation duration of 1000 seconds.

The UDU-factorized variation covariance is similar in magnitude to the batch-wise reference, but displays a large improvement when comparing the difference between the reference in relative difference given by (6.1.1). The 9 first states displays a improvement ranging from 49% to 100% percent between the UDU-variation and the reference.

Table 6.2.5: *RMSE values in for states in filter variations with m = 30 measurements*

State number	**Batch**	**Sequential**	Vs Batch	**UDU**	vs Batch	**[Unit]**
1	0.019	0.019	0	0.015	0.004	[m]
2	0.024	0.024	0	0.020	0.004	[m]
3	0.050	0.050	0	0.041	0.009	[m]
4	0.020	0.020	0	0.014	0.006	[m/s]
5	0.022	0.022	0	0.012	0.010	[m/s]
6	0.037	0.037	0	0.025	0.012	[m/s]
7	0.110	0.110	0	0.111	-0.001	[deg]
8	0.037	0.037	0	0.036	0.001	[deg]
9	0.510	0.510	0	0.491	0.019	[deg]
10	0.031	0.031	0	0.030	0.001	$[m/s^2]$
11	0.008	0.008	0	0.008	0	$[[m/s^2]]$
12	0.007	0.007	0	0.007	0	$[m/s^2]$
13	0.009	0.009	0	0.009	0	[deg/s]
14	0.001	0.001	0	0.001	0	[deg/s]
15	0.001	0.001	0	0.001	0	[deg/s]

Table 6.2.6: *Final state variance values for states in filter variations with m = 30 measurements*

Element number	**Batch**	**Sequential**	Vs Batch	**UDU**	vs Batch	% vs Batch
1	9.354e-6	9.354e-6	0	4.763e-6	4.591e-6	-49.08
2	1.088e-5	1.087e-5	0	5.935e-6	4.936e-6	-54.63
3	1.001e-3	1.001e-3	0	5.477e-4	4.532e-4	-54.73
4	1.034e-3	1.034e-3	0	1.015e-3	1.900e-5	-98.16
5	1.045e-3	1.045e-3	0	1.024e-3	2.100e-5	-97.99
6	2.558e-3	2.558e-3	0	2.091e-3	4.667e-4	-81.76
7	1.619e-6	1.619e-6	0	1.618e-6	1.000e-9	-99.94
8	1.646e-6	1.646e-6	0	1.645e-6	1.000e-9	-99.94
9	2.011e-4	2.011e-4	0	2.011e-4	0	0
10	3.189e-3	3.189e-3	0	3.189e-3	0	0
11	3.070e-3	3.070e-3	0	3.070e-3	0	0
12	3.058e-3	3.058e-3	0	3.058e-3	0	0
13	4.456e-10	4.456e-10	0	4.456-10	0	0
14	4.473e-10	4.473e-10	0	4.473e-10	0	0
15	1.592e-9	1.592e-9	0	1.592e-9	0	0

6.3 Discussion

In this thesis I found that the sequential measurement update is faster than the batch-wise measurement update for both cases where m = 15 and m = 30 measurements. In addition to this, by comparing the batch-wise reference in Table 6.1.4 to Table 6.2.4 we have that for a doubling of m measurements, the average elapsed time increased by 8.26 times, from 1.42 to 11.73 seconds, while the sequential variation increased with 1.66 times, from 1.34 to 2.22 seconds. This also suggests that the increased run time for the measurement update is reflected in the computational complexity of a matrix inversion, $O(m^3)$, where a doubling of m resulted in $2 \times m^3 = 8$ times longer run time. For the sequential variation, the increase of m is loosely reflected in the complexity of the for-loop, $O(m)$, but not quite as large as resulting in doubling in run time. This supports the starting hypothesis, where I stated that the not only would the sequential variation be faster, but the difference between the batch-wise reference would also increase with the enlargement of the measurement vector.

From Table 6.1.1 and Table 6.1.3 we observe that the time update comparison between the batch-wise and sequential variations are similar in run time and have a close to 0% speed up difference between the module. This is to be expected since they share the same equations and steps in the module. Furthermore, from Table 6.1.1 we can observe that there is a linear relation between the simulation duration and the elapsed average total run time, average run time for time update and average run time for measurement update. The same scaling relation is observed when increasing m from 15 to 30 measurements as well as seen in Table 6.2.1. But even though the scaling factor is roughly directly proportional to the run time, there are signs that the shorter durations are more affected by start up costs caused by starting the script than the longer durations, making the longer durations being more representative for the final results than the shorter durations. This may also suggest that more or even fewer differences between the filter variations could have been revealed by repeating the experiment for an even longer simulation duration. This were not done due to the required time a longer simulation duration experiment would require to perform, but it can be performed by increasing the simulation duration to the maximum of the generated simulation data set, or even generating a longer simulation data set.

In the introduction I stated that I believed the UDU-variation would be comparable to the reference for both m = 15 and m = 30, which was not the case here. The UDU-factorized variation displayed a large increase in run time for all modules. Examining Figure 6.1.1 and Figure 6.2.1 we observe that the samples are roughly 2.5 to almost 13 times larger than the values for the respective modules from the batch-wise and sequential filter variations, the same which is reflected in corresponding tables containing the numerical values and relative differences in Table 6.1.1 Table 6.1.2, Table 6.1.3 and Table 6.1.4 for m = 15, and Table 6.2.1 Table 6.2.2, Table 6.2.3 and Table 6.2.4 for m = 30. This increase is without doubt caused by UD-factorized variations utilisation of the factorization functions in both the time update and the measurement update.

In total, the UD-time update consists of two double for loops, where the outer loop has up to 15 iterations and the inner loop has up to 14 loops, where both the double loops are carried out on every time step. In addition to this, there are also a double for loop carried out in each iteration of the sequential measurement update of the same size as the one in the time update step. This contributes to m times double for loops in the measurement update, where m is equal to the measurements available. There are probably some run time % decrease to be gained by optimizing the loops to be more efficient through methods such as loop jamming since the loops are repeated and used in every time step. When doubling the number of measurements, we observe a 10% increase in relative differences between the UDU-variation and the batch-wise

reference in total run time and time update run time. What is worth noting is that for the measurement update case with m = 30, the relative difference between the reference and the UDU-variation is only a increase by 202% as opposed to the 1200% increase for m = 15. The added for-loops that comes with the UDU-variation increases the elapsed measurement update run time, but the relative difference between the reference decreases as m increases in size. It could be interesting to see if there is a cut off where the batch-wise filter variation overtakes the UDU-variation in measurement update run time by increasing m further.

When examining the numerical precision of the filters I chose to examine the RMSEs of the error states in the three variations, and the diagonal of the final covariance matrix for the longest simulation duration. The results shows that all three filter variations had state errors in a similar magnitude and did not alone imply that one variation were better than the others. The figures accompanying figures in Section 6.1.3 and Section 6.2.3also displays that the state errors were inside a 3 times standard deviation for all states with the exception of the gyro rate bias error, which lies outside the deviation with a magnitude of 0.0010 degrees. For the sequential variation and the batch-wise reference variation, we note that the results are numerically identical. In other words there are no performance losses measurable by those two metrics in this experiment, and the faster sequential variation is equally as accurate for both cases. When examining the UD-variation the results becomes somewhat suggestive in terms of forming a conclusion about the results. The RMSE displayed similar levels of error for the states, while the covariance were lower for the UD-variation with a magnitude ranging from 10e-4 to 10e-9 for m = 30, and 10e-2 to 10e-9 for m = 15. Translating this to a relative difference we have that the UD-variation displayed between a 0% to 100% lower covariance for some states in the filter. All though this result is something that should be examined closer. Since the magnitudes of the covariance already are so small, the relative difference becomes large in % even when the variance value difference is only 0.00001. Simon states [3] (p.174) that the UD-variation increases the computational cost of the filter when comparing it to the batch-wise and sequential filter, but not as much as in other variations such as the square root filter. Even though it seems to be the consensus that the UD-variation of the Kalman filter should be a good replacement and not too costly when comparing it to the other two variations, this experiment suggests that the UD-factorized variation should not be used instead of the standard of sequential variation in a filter where real time computation is important in the operation.

When investigating why the state errors were so similar in magnitude, a couple of hypothesis emerged. One of them being that both the measurement and system noises and biases could be set too low, even for artificially low standards in the simulation model and thus are not large enough to cause any drift that "challenges" the filters. Another factor could be the development platform this thesis is based on. The model and filter implementation is written in Python, which uses a 64-bit resolution for numerical operations using floats, and 32-bit resolution for integers unless otherwise has been specified. This may have resulted in no significant numerical errors caused by rounding and other floating point operation limitations. Combining this with the artificially low noises, it can't be determined if there are any differences regarding the numerical stability of the variations based on state error and trajectory accuracy. A final option can be that all three variations share a strong and robust performance and state estimation accuracy.

It is worth mentioning that these findings contradicts the findings in my project thesis [1], where I found that sequential filter strategy were no faster than the standard filter strategy. This could be due to the number of simulations performed to control deviations between the runs and hardware limitations. Performing only 20 runs on the hardware that were used yielded a large spread in the measured times, where some of the experiments gave results in favour of the hypothesis stating that the sequential method were faster, and some in the opposite direction. Increasing the number of simulations per run according to parameters set in Chapter 6, the

sample size were big enough that we could perform a proper time analysis where the between run variations and outliers were reduced in their significance and ability to influence the sample set. Another reason for why the results were different is due to the separation of the time- and measurement update module in the timing processes. Since the measurement update occurs at a rate of 1Hz and occupies less than 1% of the total elapsed run time, the measurement update speed up would be reflected in the total run time as a 0.005% difference. Another way this difference might have been observed in the project thesis is if I had performed a simulation with more than 15 measurements and compared those.

This argument of how little the measurement update contributes to the total run time brings in a new angle in the discussion regarding error state Kalman filter optimization. If one desires a faster Kalman filter strategy, there could be more to gain from optimizing the time update module as it contributes to over 98% of the total run time in the standard- and sequential variation, and 95% of the total run time for the UD-factorized variation. The Van Loan discretization done in the time update module in all three variations is a safe discretization option, but comes with the cost of being computationally expensive. A substitution for this may improve the run time on cost of other performance metrics such as state error and accuracy.

Part IV

Conclusion and further work

Chapter 7

Conclusion

The results of this thesis leads to the conclusion that the sequential measurement update is faster than the standard error-state Kalman filter measurement update for cases with a large measurement vector. Additionally, the difference between the sequential measurement update and the reference measurement update increased at a close to cubic rate when increasing the number of aiding measurements. The final improvement in run time for the measurement update were a speed up increase of 4.25% for m = 15 measurements and 81.65% for m = 30. Increasing the measurement vector from 15 to 30 yielded a 1.66 increased run time for the sequential variation as opposed to a 8.26 times increase for the batch-wise reference. This suggests that the increase in run time is proportional to the computational complexity of the matrix inversion with $O(m^3)$ where m corresponds to the number of measurements in the measurement vector, implying that the run time increase is increasing at a cubic rate proportional to the number of aiding measurements in the measurement vector.

Regarding performance, I found that the sequential variation is numerical identical to the standard batch-wise variation used for reference in terms of root mean square error and covariance estimation, meaning no accuracy or performance loss cane be observed between these two variations from these metrics.

The UD-factorized variation were significantly slower in time-update run time and measurement update run time compared to the batch-wise reference. For m = 15, the UD-variation were 137% slower in the time update and 1161% slower in the measurement update. For m = 30, the difference were a 146% and 200% speed increase for the UD-variation. Comparing the UD-variation to the sequential variation, the difference is larger. For m = 15, the relative speed up difference is 1443% for m = 15 and 1537% for m = 30.

The RMSE of the UD-variation were similar in magnitude to the reference variation.

The covariance of the UD-variation were lower than for the reference variation with magnitude difference between 1e-4 to 1-e9, yielding a relative difference varying between 0% to 100%. The numerical improvements yielded by a UD-factorized filter strategy suggested by D'Souza can not be determined to be present in this experiment due to the low magnitude values and high relative improvement values.

Chapter 8

Further work

To both contiue the study of efficient implementations of the error-state Kalman filter, there are several options that could be examined when it comes to run time optimization. In addition to this, the findings of this master thesis should be verified on other platforms in addition to the python based platform used.

Some suggestions for further work are listed below

1. **Increase system and measurement noises for verification of the covariance estimation improvement for the UD variation**: In this thesis I found that the UD-variation yielded a up to 100% lower covariance for some states in the state estimation, but this should be verified in a model where the system and measurement noise and biases are more realistic. Since the values are so low in magnitude, but high in relative difference, they might give a misleading result.
2. **Optimization of time update step:** In this implementation of the ESKF, the time update occupied roughly 98% of the total elapsed run time for a simulation, which leads to the idea of examining optimization possibilities for the time update. Using the Van Loan method of discretizing the transition and system noise matrix is computationally costly due to the matrix exponential used when solving "la.expm(V)" to obtain the Van Loan matrix. Different strategies may result in a faster time update module, but it may come at the cost of other things such as accuracy.
3. **Micro-controller**: To verify the results and to examine the state accuracy, a micro controller with lower bit-resolution than Python could be used. By using a lower bit-resolution one may also uncover larger numerical differences between the models.
4. **Real life application**: The model could be changed to use more realistic system- and measurement noise, or tested on experimental data.
5. **Composite Kalman implementations such as LDL, QR and LU:** Examine if other factorization based strategies such as the LDL, QR and LU composition of the covariance matrix are faster than the UDU-factorization.
6. **Optimize the UD-factorization loops:** Timing and bench-marking the for loops in the factorization variation to examine if the loops can be written more efficient through. This could yield a decrease in run time for the UDU-variation
7. **Batch-wise UDU-filtering:** Compare the sequential UDU measurement update in the UDU-variation to D'Souza and Zanettis paper on "*Information Formulation of the UDU Kalman filter*" [17]. This paper suggests a vectorial measurement update which avoids the

need for having a diagonal measurement noise matrix and to handle the measurements one at the time. Since the sequential UD-variation requires so many additional for-loops when handling the measurements one at a time, there is reason to believe that the vectorial and sequential UD-variation should be similar in run time for large measurement vectors.

8. **Increase the fidelity of the simulation model:** Implement functionality that handles
   - Clock error in pseudorange.
   - Atmospheric disturbances such as ionospheric and tropospheric noise, multipath and ephemeris error.
   - Position sensors and recievers and compensate the system for the lever arms introduced.
   - Add noise to both the GNSS and beacon measurements.
   - More complex IMU error modeling.

References

[1] Haugland Andreas. *Efficient implementation of inertial navigation systems*. 2021.

[2] Joan Solà. *Quaternion kinematics for the error-state Kalman filter*. **volume** abs/1711.02508. 2017. arXiv: 1711.02508. URL: http://arxiv.org/abs/1711.02508.

[3] Dan Simon. *Alternate Kalman filter formulations*. John Wiley Sons, Ltd, 2006. ISBN: 9780470045343. DOI: https://doi.org/10.1002/0470045345. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0470045345. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0470045345.

[4] Christpher N. D'Souza J. Russell Carpenter. *Navigation Filter Best Practices*. 2017. URL: https://ntrs.nasa.gov/api/citations/20180003657/downloads/20180003657.pdf.

[5] Catherine Thornton. *Triangular Covariance Factorization for Kalman Filtering*. 1976. URL: https://ntrs.nasa.gov/api/citations/19770005172/downloads/19770005172.pdf.

[6] Martin A. Salzmann. *Some Aspects of Kalman Filtering*. University of New Brunswick, Canada, 1988. URL: http://www2.unb.ca/gge/Pubs/TR140.pdf.

[7] Paul Groves. *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems, First Edition*. 2008. ISBN: 978-1-58053-255-6.

[8] G. Bierman. *Factorization Methods for Discrete Sequential Estimation*. 1977.

[9] URL: https://github.com/aHaugl/TTK4900-Master.

[10] URL: https://github.com/aHaugl/TTK4550.

[11] Edmund Brekke. *Fundamentals of Sensor Fusion, Target tracking, navigation and SLAM*, **pages** 51–74, 167–187.

[12] Thor I. Fossen. *Handbook of marine craft hydrodynamics and motion control, 2nd edition*.

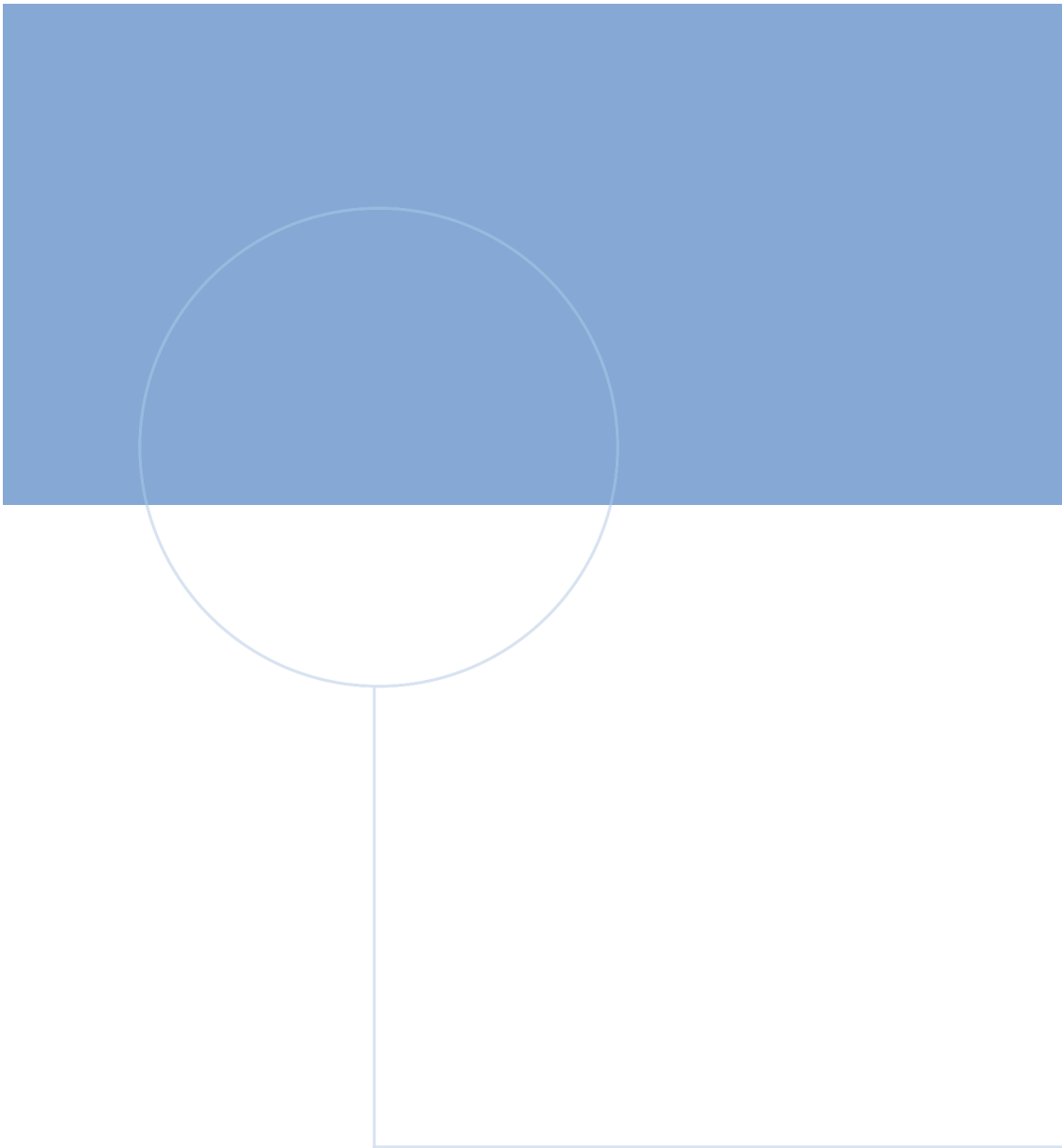
[13] URL: https://www.sensonor.com/products/inertial-measurement-units/stim300/.

[14] Torleiv H. Bryne. *TK8109 Navigation - Lecture 1, Introduction to Inertial Navigation and Inertial Sensors*. 2021.

[15] C. Van Loan. *Computing integrals involving the matrix exponential*. **volume** 23. 3. 1978, **pages** 395–404. DOI: 10.1109/TAC.1978.1101743.

[16] URL: https://docs.python.org/3/library/timeit.html.

[17] Christopher D'souza **and** Renato Zanetti. *Information Formulation of the UDU Kalman Filter*. **volume** 55. 1. 2019, **pages** 493–498. DOI: 10.1109/TAES.2018.2850379.



NTNU

Norwegian University of
Science and Technology