Master Thesis

KALMAN FILTER ADAPTATION FOR STABLE AUTONOMOUS DRONE NAVIGATION

Lucas Pirlet
Supervisor : Prof. Skaloud Jan
External Expert: Charlaix Jean-Christophe Assistant Laupré Gabriel
Geodetic Engineering Laboratory EPFL

Fall Semester 2020

Acknowledgements

I wish to show my gratitude to Professor Jan Skaloud, who encouraged me to explore the possibilities on my own while always being present to redirect me when my path would approach a wall.
I would also like to pay my special regards to Laupré Gabriel, who was always available when questions or doubts arose. His confidence and support in my ideas motivated me throughout this project. I hope that this work will prove useful to his own work and I am very thankful for the opportunity he has given me to work on this project.
I wish to thank the Geodetic Engineering Laboratory team, which welcomed me with open arms.
I want also to thank my friends and family, who supported me for this work and provided happy distractions outside of this work.

Abstract

Kalman filters are commonly used in the navigation of unmanned aerial vehicles. A new form of navigation system based on a dynamic model has shown strong potential to increase performance compared to current navigation systems. However the complex process model leads to an implementation which currently can not be used as is in an embedded system. This work tackles different aspects of the current implementation in an effort to render the filtering process more stable. Different methods to remedy the existing problems are explored, applied and analysed. On the numerical side ill-conditioning of the process is analysed. By scaling different states and measures the ill-conditioning is remedied, assuring stability of the calculation and reducing the risk of computation failures due to singular matrices. Three initialization methods are analysed: initialization with a quasi-diagonal matrix, reusing correlations from a previous calculation and a "Warmup" using partial-updates. A novel approach using a Schmidt-Kalman filter to reduce erratic changes in the trajectory due to an increase in uncertainty is explored. To reduce computational complexity of the algorithm, a lumped model is created. Strongly correlated states are replaced by linear relations. The reduced model shows negligible differences in performance with respect to the complete model.

Résumé

Les filtres de Kalman sont couramment utilisés dans la navigation de drônes. Une nouvelle forme de système de navigation basée sur un modèle dynamique a montré un fort potentiel d'amélioration des performances par rapport aux systèmes de navigation actuels. Toutefois, la compléxité du modèle du système mène à une implementation qui ne peut actuellement pas être utilisée telle quelle dans un système embarqué. Ce travail aborde différents aspects de l'implementation actuelle afin de rendre le processus de filtrage plus stable. Différentes méthodes pour remédier aux problèmes existants sont explorées, appliquées et analysées. Sur le plan numérique, le mauvais conditionnement du processus est analysé. En modifiant l'échelle des différents états et mesures, le mauvais conditionnement est corrigé, ce qui garantit la stabilité du calcul et réduit le risque d'échec du calcul dû à des matrices singulières. Trois méthodes d'initialisation sont analysées : initialisation avec une matrice quasi-diagonale, réutilisation des corrélations d'un calcul précédent et un "Warmup" utilisant des mises à jour partielles. Une nouvelle approche utilisant un filtre de Schmidt-Kalman pour réduire les changements erratiques de la trajectoire dus à une augmentation de l'incertitude est explorée. Pour réduire la complexité de calcul de l'algorithme, un modèle réduit est créé. Les états fortement corrélés sont remplacés par des relations linéaires. Le modèle réduit présente des différences de performance négligeables par rapport au modèle complet.

Contents

1 Introduction

1.1 Context

The Kalman filter, developed in 1960 [1], is widely used in different domains, be it in control systems, navigation or even meteorology [2] [3] [4] . Therefore, it currently plays an important role in many applications.
Historically, its first usage in navigation is attributed to Stanley F. Schmidt, in the trajectory determination of the Apollo program, using an onboard sextant as only sensor [5].
Since then, navigation techniques have strongly increased in complexity, using a variety of sensors such as inertial measurement units (IMU), GPS receivers (GNSS) and velocity sensors.
Although we have much larger computational power than for example in the first Apollo missions, the usage of high frequency sensors and more complex models can still bring these systems to their limits. Especially in embedded systems where the available memory and processing power are restrained.
Current navigation systems are often based on IMU inputs for the computation of the process model in the filter[6]. But the considerable errors on these sensors, make it necessary to add secondary sensors to the system such as GNSS. This limits the usage of such systems to environments where GNSS is available at all times.
In 2016 Dr. Khaghani [7] laid down the theoretical background for the usage of a vehicle dynamic model (VDM) as the process model of a navigation system. The usage of a VDM based navigation system greatly increases the accuracy of positioning. In particular it greatly outperforms classical inertial navigation systems (INS) during periods where no GNSS measurements are available [7]. An important drawback of this method is that it adds a multitude of states that are estimated by the filter. A direct consequence of this are more expensive computations. Another consequence is that the states allow for a lot of freedom, increasing the the risks of convergence to a local minimum.

1.2 Outline

The goal of this project is to continue the development of the VDM based navigation system for an implementation as an embedded system. Currently the computations are performed in post-processing with data logged during flights of a small UAV.
The usage of so called online computation, i.e. an in-flight usage of the navigation system, requires further development of the technology. Due to the restricted processing power of an embedded system, a valuable improvement is the reduction of the computational complexity of the filter. Another important factor in the implementation of a live system is the general stability of the filtering algorithm. Divergence or even failures to compute values can result in a false position or even in an involuntary stop of the system. Consequences of this can be bad estimations of the pose or in the worst case physical harm. The stability of the filter is especially important in the case where the system is used in autonomous navigation. In this work different aspects of the filter stability are examined and for each aspect possible solutions are explored.
In a first part the computation has to be made more stable as currently the estimator is

subject to some roundoff errors due to large differences in the scale of some filter states. To assure stability of the filter during calculation, the numerical health of the system is assessed. By scaling different variables we reduce the risk of round off errors.
In a second part the stability of the VDM parameters is assessed and reinforced. Two problematic phases have been identified: the initialization and GNSS outage. Due to the complex process model, initialization with correct values becomes very complex. An initialization with a sparse covariance matrix will lead to missing information, which in turn will provoke some poor quality or even false updates at the start. In the absence of GNSS updates the uncertainty of position grows considerably. As consequence, these states are updated very aggressively by the filter. Strong updates on position lead to erratic movements which can lead to problems with the autopilot. For this the usage of a Schmidt-Kalman filter proves to be a valuable tool. Also called the "consider" filter, it allows to not update certain states. At initialization a warm-up phase is proposed. In this phase the updates on the VDM coefficients are reduced. Leading to a slower convergence of these coefficients at the beginning and possibly reducing the effects of initialization errors as well as lacking correlations between coefficients. During outages the position states are removed from the IMU observations. Therefore the position becomes only a result of the integration of velocity from the VDM.
The final part focuses on the reduction of the computational complexity by reducing the number of parameters used in the VDM. Strong correlations between coefficients show that it might be possible to simplify the dynamic model. Highly correlated states are identified and analysed. Coefficients showing a linear relation are reduced by using a regression. The root mean square error in velocity and orientation are evaluated during outage to quantify the effects on navigation performance of the simplifications. Finally a combination of all simplifications is computed and evaluated.

2 Vehicle Dynamic model

Vehicle dynamic navigation as proposed by Mehran Kaghani in 2016 [7] is a new approach to drone navigation. The main principle is to use a dynamic model of the UAV as process model opposed to inertial sensor based navigation where the process model is controlled by the readings of IMUs. A similar approach can be found in odometry of ground based robots, where by reading the actuator movements, such as wheel velocity inputs, the displacement of the vehicle can be calculated.
The principle of VDM navigation uses the flight control inputs to compute the forces and the moments which should apply to the UAV [7]. But contrary to ground based odometry, which are mostly based on solid mechanics, the dynamics of a flying vehicle are explained by fluid mechanics. This implies more complex interactions between the vehicle and its environment. The model uses the actuator input values of the control surfaces, i.e. ailerons, ruder and elevator, and the propeller speed as input to compute the forces and angular accelerations of the vehicle. Additional information that has to be provided are the velocity compared to the wind frame as well as a multitude of parameters to compute friction, lift and the induced moments.

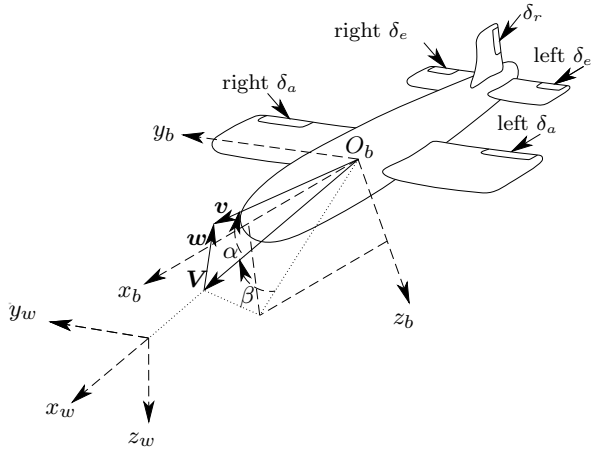


Figure 1: Body and wind frames, with airspeed $\boldsymbol{V}$, wind velocity $\boldsymbol{w}$ and UAV velocity $\boldsymbol{v}$. Adapted from [8]

The governing equations for the VDM are the following:
The thrust force is defined in the body frame along the x-axis.

$$F_T = \rho D^2 (C_{F_T 1} n^2 D^2 + C_{F_T 2} \frac{Vn}{D\pi} + C_{F_T 3} \frac{V^2}{\pi^2}) \tag{1}$$

V is the norm of airspeed i.e the velocity at which the drone moves through the air. D is the diameter of the propeller and n is the rotation speed of the propeller.
The following forces correspond to the drags in the directions of the wind frame axes.

$$F_x^w = \bar{q} S (C_{F_x 1} + C_{F_x \alpha} \alpha + C_{F_x \alpha 2} \alpha^2 + C_{F_x \beta 2} \beta^2) \tag{2}$$

$$F_y^w = \bar{q} S (C_{y1} \beta) \tag{3}$$

$$F_z^w = \bar{q}S(C_{F_z1} + C_{F_z\alpha}\alpha) \tag{4}$$

The angles α and β are the angle of attack and sideslip of the UAV, respectively. $\bar{q}$ is the dynamic air pressure ($\bar{q} = \frac{\rho V^2}{2}$, with ρ being air density) and S is the wing surface.
The moments in the body frame are defined by the following equations:

$$M_x^b = \bar{q}Sb(C_{M_xa}\delta a + C_{M_x\beta}\beta + C_{M_x\omega_x}\omega_x + C_{M_x\omega_z}\omega_z) \tag{5}$$

$$M_y^b = \bar{q}S\bar{c}(C_{M_y1} + C_{M_ye}\delta_e + C_{M_y\omega_y}\omega_y + C_{M_y\alpha}\alpha) \tag{6}$$

$$M_z^b = \bar{q}Sb(C_{M_zr}\delta_r + C_{M_z\omega_z}\omega_z + C_{M_z\beta}\beta) \tag{7}$$

Some geometric properties of the drone can be determined by simple measurements, the variables b and $\bar{c}$ correspond to the wingspan and the mean aerodynamic chord of the drone, respectively. The position of the ailerons, the elevators and the rudder are the variables δ_a, δ_e and δ_r, respectively. ω is the rotation velocity of the UAV around the axis corresponding to its subscript, it is part of the estimated navigation states.

2.1 Advantages of VDM-based navigation

The usage of a dynamic model to determine the UAV's attitude and position globally increases the precision of the navigation states [9]. Essentially, all sensor inputs can be used as measurements, which is not the case for example in INS where the IMU measures are part of the process model [6].
Therefore the implementation allows for more information to be used during a flight. The usage of a physical model to determine the evolution of the states additionally constrains partially the evolution of the navigation states to movements that are possible for the UAV. Especially during GNSS, outage the accuracy of computed positions increases considerably [7]. Because in VDM based navigation IMUs are part of the measurements, the errors of these sensors are directly linked to measures.

2.2 Difficulties related to VDM-based navigation

Although VDM-based navigation has certainly potential to increase the overall performance of navigation systems, its implementation is linked with various difficulties. Most of these difficulties are due to the increase of the number of states to be estimated and the fact that most of the VDM coefficients are unknown.
This makes tuning the filter noises especially tedious as a multitude of parameters have to be tracked to assure convergence. Additionally, the parameters of the VDM are strongly correlated, this implies that errors in one state might be compensated by corrections of other states. The latter creates an estimation model that has a lot of flexibility and that can have multiple solutions that can closely fit the measurements.
Another concern due to the elevated number of states is computational complexity. The model used in this work contains 47 states. If other models, such as misalignments or more

complex sensor error models , are added, the number of states can double. Computational complexity has to be taken into account because the system will have to be adapted for an embedded system, which will have reduced computational power.

3 Data and materials

For this work, two main programs have been used. The Simulator is a simulation to create artificial data of drone flights. The Estimator is a Matlab implementation of the navigation system.

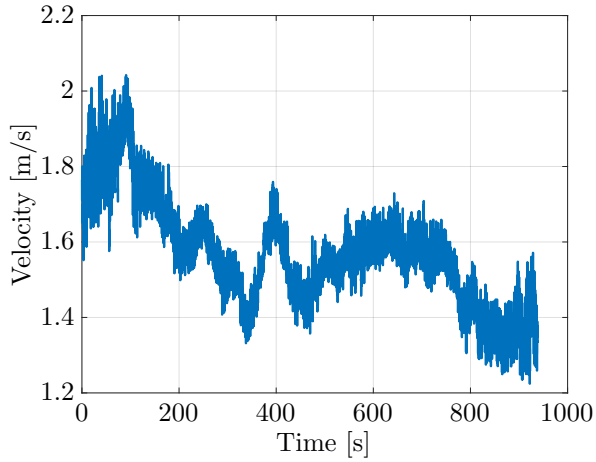
3.1 Simulator

For the generation of data, a simulator coded in Matlab is used. A simple guidance system allows to compute the necessary actuator inputs for the UAV to move from one waypoint to the following. The simulator uses the same equations as the VDM to compute the forces and moments acting on the drone.
From the computed navigation states and forces the corresponding sensor inputs are generated. In this work the sensors used are one IMU unit with 3 accelerometers and 3 gyroscopes and a GNSS sensor which measures position and velocity. These measures are later on corrupted with noises with the characteristics given in Tab. 1. They are composed of a bias, a first-order Gauss Markov and white noise for the IMU sensor. GNSS position and velocity measures are only affected by white noise. The values originate from [7], the values are unmodified with exception of the GNSS position noise which corresponds to 10 % of standalone GNSS precision.

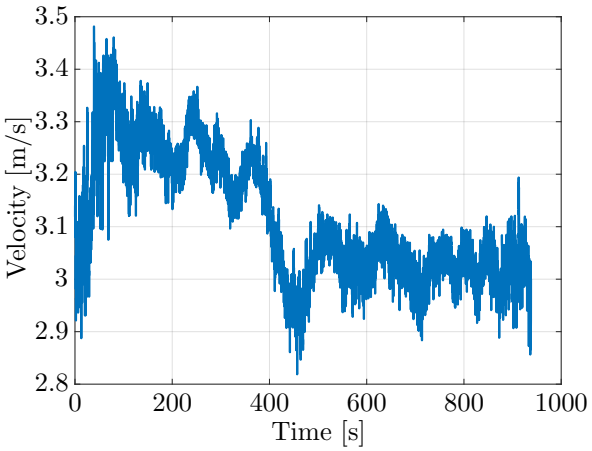
Sensor	Type	Value	
Accelerometer	Bias	0.0784	$[m/s^2]$
	GM σ	0.05	$[mg]$
	GM T	200	$[s]$
	WN σ	50	$[\mu g/\sqrt{Hz}]$
Gyroscope	Bias	720	$[deg/h]$
	GM σ	0.0028	$[deg/s]$
	GM T	200	$[s]$
	WN σ	0.18	$[deg/\sqrt{h}]$
GNSS position	WN σ hor.	0.25	$[m]$
	WN σ vert.	0.3	$[m]$
GNSS velocity	WN σ hor	0.04	$[m/s]$
	WN σ vert.	0.05	$[m/s]$

Table 1: Values of noise used in the simulation. GM stands for a Gauss-Markov process and WN for white noise

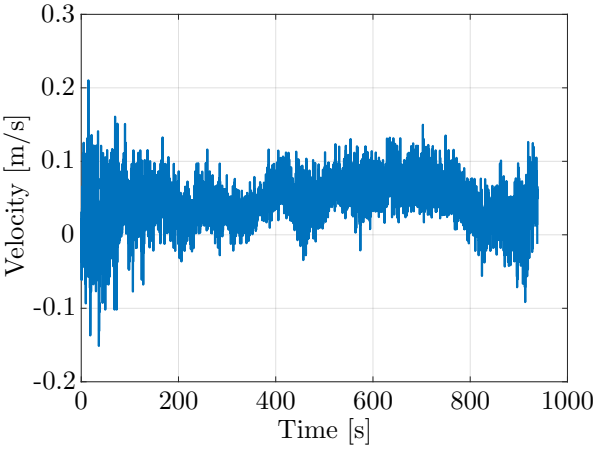
Additionally, it is possible to include wind in the computation of the trajectory. The wind values used originate from real wind measurements. A representation of these values and corresponding statistics can be seen in Fig. 2.



(a) Wind velocity north



(b) Wind velocity east



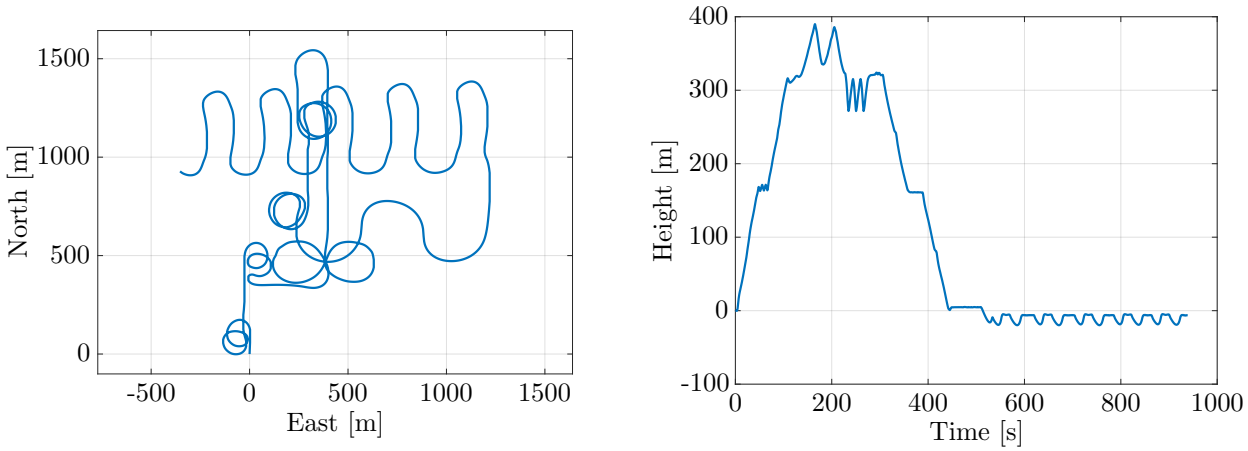
(c) Wind velocity down

Component	Mean [m/s]	σ [m/s]
North	1.576	0.135
East	3.119	0.120
Down	0.044	0.029

(d) Wind velocity statistics

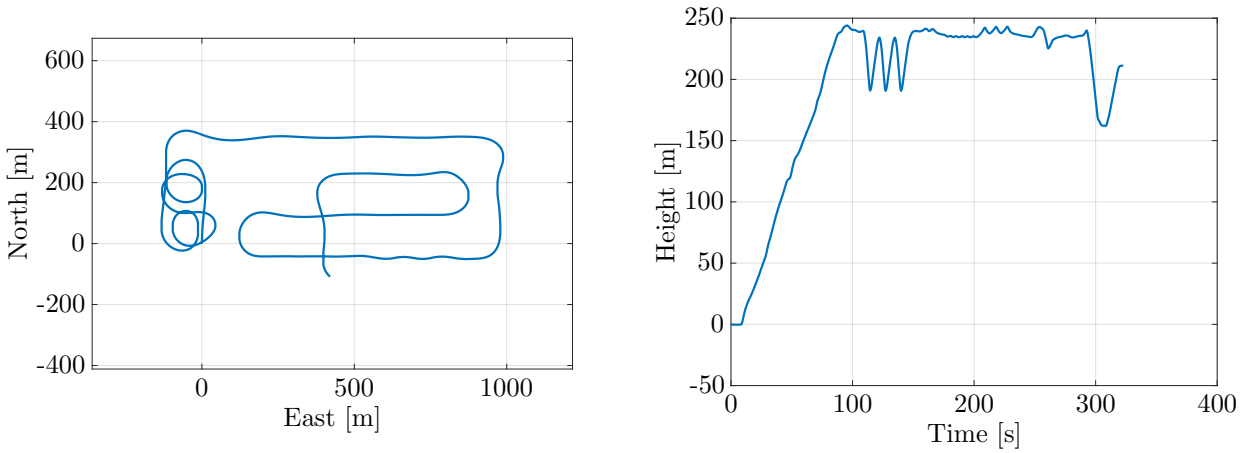
Figure 2: Visualization and statistics of the used wind values

For the usage in the Estimator, two trajectories have been defined (Fig. 3, Fig. 4). For both trajectories the starting point is the origin of a local reference frame. To ensure observability of all states the trajectories are highly dynamic. Different dynamics excite distinct groups of parameters, increasing their estimation [8]. Depending on the experiment GNSS is made unavailable at the end of the flight.



(a) Horizontal compontents of first trajectory (b) Relative change in height of first trajectory

Figure 3: First trajectory



(a) Horizontal components of trajectory 2 (b) Relative change in height of second trajctory

Figure 4: Second trajectory

3.2 Estimator

The navigation estimator is based on a Kalman filter, which estimates the states necessary to navigation, as well as VDM coefficients, actuator states and IMU biases.
The main algorithm can be decomposed in two parts, a prediction step and an update step. In the prediction step the evolution in time of the states is predicted using a predefined process model. The corresponding covariance matrix is also propagated during this step. The process model Φ describes the evolution in time of the states x. To account for the error between the modeled evolution and reality a so called process noise Q is introduced.

$$\tilde{x}_k = \Phi_k \hat{x}_{k-1} \tag{8}$$

$$\tilde{P}_k = \Phi_k \hat{P}_{k-1} \Phi_k^T + Q_k \tag{9}$$

When measurements are available a prediction of the states is made. The filter compares the accuracy of the predicted states $\tilde{P}$ with the measurements supposed accuracy R to correct the predicted states with the new information. A central part is the gain K which is computed in the following manner.

$$K_k = \tilde{P}_k H_k^T (H_k \tilde{P}_k H_k^T + R_k)^{-1} \tag{10}$$

The gain is then used to update the states.

$$\hat{x}_k = \tilde{x}_k + K_k(z_k - H_k \tilde{x}_k) \tag{11}$$

Because additional information is available the covariance matrix is adapted to correspond to the update states.

$$\hat{P}_k = (I - K_k H_k)\tilde{P}_k \tag{12}$$

In our case we use an Extended Kalman Filter (EKF) with linearized measurement functions. In addition to the static VDM parameters and the actuators, there are 13 navigation states describing the movement of the UAV. Three states represent the wind velocity in the local frame, these velocities are modeled as random walks. To take into account sensor errors there are 6 states representing the IMU errors. These errors are as well modeled as random walks. The errors in GNSS position and velocity are considered as white noises, therefore no additional state is needed for the errors of these measures.
We can distinguish the states at different levels:

State group	Level 1	Level 2	Symbol
X_n	Navigation states	Position	x
		Velocity	v
		Attitude	θ
		Rotation velocity	ω
X_p	VDM parameters	Forces	P
		Moments	
X_a	Actuators	Propeller speed	U
		Control surfaces	
X_w	Wind		W
X_e	Noises	Accelerometer	b_a
		Gyroscope	b_g

Table 2: Distinction of the estimated states

The main equations used in the process model are defined in section 2.
Initial values for the navigation states are taken from the first values of the simulation, depending on the experiment their values are corrupted according to their initial variance. Wind and IMU biases are initiated at 0.
The initial standard deviation of the navigation states is defined in Tab. 3. For the position and attitude states their variance is propagated to ellipsoidal coordinates and into quaternion space. Velocity is given in the body frame and propagated into the local frame.

Position	horizontal	0.02	[m]
	vertical	0.03	[m]
Velocity	horizontal	0.04	[m/s]
	vertical	0.05	[m/s]
Attitude	roll/pitch	0.5	[deg]
	yaw	1	[deg]
Angular velocity	roll/pitch	1	[deg/s]
	yaw	2	[deg/s]

Table 3: Initial standard deviation of navigation states

Initial variance for the error states is equivalent to the biases defined in Tab. 1. The wind variances are initialized to 2 [m/s] and 1 [m/s] for the horizontal and vertical directions respectively.
All measurement uncertainties are tuned to the white noises found in Tab. 1.

4 Numerical Stability

A value stored in the memory of a computer will only have a fixed number of bits available. This implies that numbers can only be stored with a finite precision. The finite precision of stored numbers leads in general to roundoff errors, which means that the part below machine precision is rounded or the additional bits are chopped [10]. In general, every numeric resolution of problems is subject to approximations in the computation, i.e in iterative processes and roundoff errors. Depending on the nature of the problem these can seriously limit the precision obtained through the calculation.

4.1 Roundoff errors and stability of Kalman filters

Already in the 1960 numerical stability issues have been discovered when using Kalman filters and already problems with only 6 states showed the presence of numerical instabilities[1].
Roundoff errors become problematic when a problem is ill-conditioned, i.e the problem combines numbers with very different scales. This implies that when roundoff errors occur, it is possible that a matrix becomes singular and can therefore not be inverted, leading to a failure in the computation. Examples of these problems can be found in [10] and [5]
A good indicator of ill-conditioning is the the condition number of a matrix [10]:

$$cond(A) = \frac{|\lambda_{max}|}{|\lambda_{min}|} \tag{13}$$

Where λ_{max} and λ_{min} are the largest and the smallest eigenvalue of A, respectively. In the case of a singular matrix, there is at least one eigenvalue that is equal to 0, leading to an infinite condition number. For a well conditioned problem $cond(A)$ should stay relatively low as a function of the machine precision.
We can use the following rule of thumb to ensure a well conditioned matrix[6]:

$$cond(P) << \frac{1}{2^{-N}} \tag{14}$$

With N being the number of bits used in the mantissa. For the current project we work in Matlab in a 64 bits architecture, which uses 52 bits for its mantissa [11]. Therefore the condition number should be in our case well below 10^{15}.
Additionally we can bound the propagation of roundoff errors with the following formula [12]:

$$||\overline{(A^{-1}b)} - (A^{-1}b)|| \leq \epsilon \; cond(A)||(A^{-1}b)|| \tag{15}$$

Where $||\cdot||$ is the 2-norm.
Eq. (15) shows that the errors in the resolution of a linear system is directly proportional to the condition number of the inverted matrix. A problem with more or less homogenous scales will be less affected by roundoff errors.
Although numerical errors will not necessarily lead to a divergence of the filter, it is possible that the filter becomes momentarily unstable, which in turn will lead to a slower convergence[10]. Small values in the matrices may be rounded to 0 during computations, this can lead to changes in sign of the final result. In case of the Kalman gain the filter can through that update wrongly certain states.

4.2 Problem statement

The initial implementation of the VDM based navigation used WGS84 ellipsoidal coordinates as position states, namely as latitude, longitude and height. Additionally to use these states in the preprogrammed functions the angles have to be given in radians. This choice leads to the fact that when computing displacements the values quickly become very close to machine precision. The following rule of thumb is quite useful for people handling geographical data and gives a good estimation of the order of magnitude of a displacement on the earths surface:

$$30[m] \approx 1" = 2.7 * 10^{-4}[°] = 4.8 * 10^{-6}[rad] \tag{16}$$

The standard deviation of a position can be assimilated as a bound of displacement of a measure according to its true value. With todays advances in GNSS technology the accuracy increased strongly, which in turn decreases the variance of position in the filter. For example centimeter accuracy will lead to a standard deviation of approximately 10^{-9}, which creates an element in the covariance matrix of 10^{-18}.

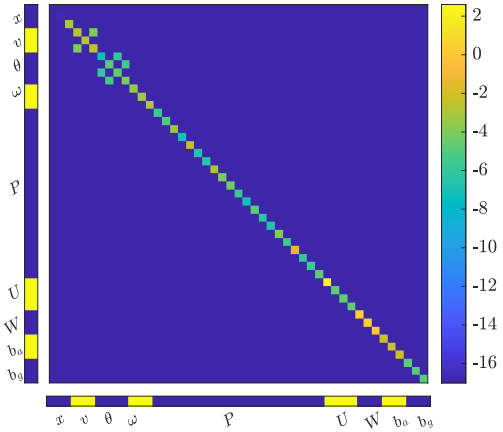


Figure 5: Initial Covariance matrix

Fig. 5 shows the order of magnitude of each element of the initial covariance matrix. It is to note here, that the initial covariance matrix is quasi diagonal, with exception of the attitude, the velocity and the position states, which are propagated to their corresponding frame / space. We can already see that the elements corresponding to latitude and longitude are very low, they are hardly distinguishable with the zero elements that are present off-diagonal. In consequence the initial condition number is very high (10^{24}), which is already strongly above the limit defined by Eq. 14. Although this value sharply drops sharply during the first few iterations of the filter, it stabilizes close to 10^{15}. This still leads to a high probability of significant roundoff errors.

4.3 Solution approaches

In a first step, problematic states have been identified by analyzing the initial covariance matrix. It has been therefore decided to change the units of position, as these states showed the smallest variance (10^{-18}). Also the propeller velocity has been changed as it showed a variance in the order of 10^2. This should create a more homogenous covariance matrix. The chosen scaling factors are 10^8 for latitude and longitude and 10^{-2} for the propeller velocity. We then proceed to reduce the difference in the orders of magnitude of the states and their variance by rescaling outliers manually before recomputing the corresponding model. This rescaling should reduce the condition number of the covariance matrix.
The rescaling on a Kalman filter can be shown as follows:

If we scale X_u by using a diagonal matrix D, which can be seen as a change of units, we obtain a scaled vector X_s

$$X_s = D_x X_u \tag{17}$$

The other matrices have to be adapted in the following way:

$$P_s = D_x P_u D_x^T \tag{18}$$
$$\Phi_s = D_x \Phi_u D_x^{-1} \tag{19}$$
$$H_s = H_u D_x^{-1} \tag{20}$$
$$Q_s = D_x Q_u D_x^T \tag{21}$$

Although one can simply scale the states and not bother with the other matrices used in the filter, this will quickly lead to just moving the scaling problem to other parts of the filter.
As seen in Eq. (15) the propagation of roundoff error in an inverse is proportional to the condition number. In the current implementation an inversion occurs in the computation of the Kalman gain, specifically the expression $HPH^T + R$ is inverted.
By only scaling the states the condition number of the matrix $S = HPH^T + R$ will not change:

$$H_s P_s H_s^T + R = H_u D_x^{-1} D_x P_u D_x^T \left(H_u D_x^{-1}\right)^T + R = H_u P_u H_u^T + R \tag{22}$$

Essentially, the units of this matrix are controlled by R and thus correspond to the measurement units. These measurements have to be scaled to fit the new state vector. Scaling measures only affects the measurement matrix H and the measurement noise R in the following way:

$$R_s = D_z R_u D_z^T \tag{23}$$
$$H_s = D_z H_u \tag{24}$$

In this case the matrix S becomes:

$$S_s = D_z \left(H_u P H_u^T + R\right) D_z^T = D_z S_u D_z^T \tag{25}$$

By carefully choosing the elements of D_z the elements of S can be brought to a more homogenous level. Effectively lowering the condition number of the matrix.

4.4 Results

After scaling the states corresponding to latitude and longitude, as well as the propeller speed the condition number of P during the flight diminishes considerably.

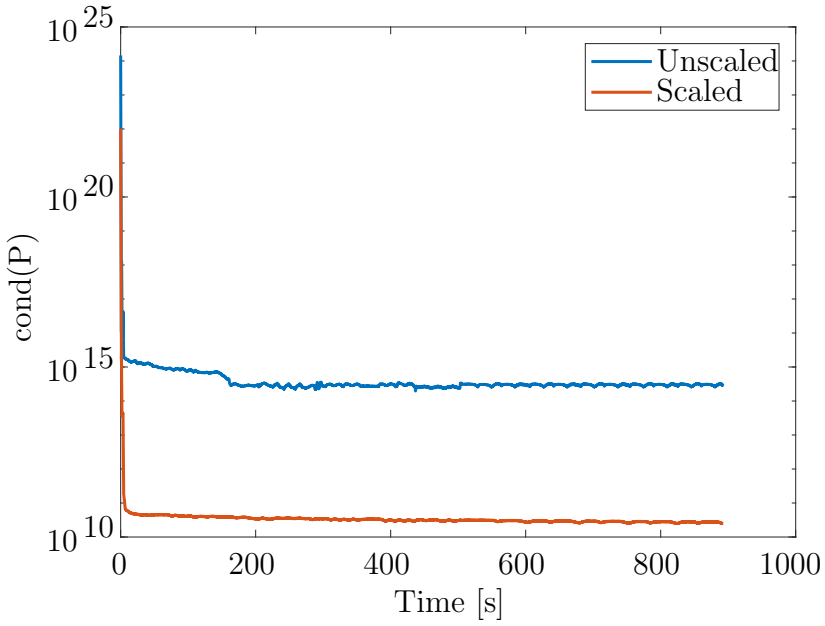


Figure 6: Condition number of the covariance matrix with and without scaling of lat/long and n

Fig. 6 shows that without scaling the condition number of the covariance matrix is very elevated during the whole flight. The sharp drop at the beginning is due to the initialization with a quasi diagonal matrix, this issue will be adressed in Sec. 5. After rescaling the condition number is considerably lower.

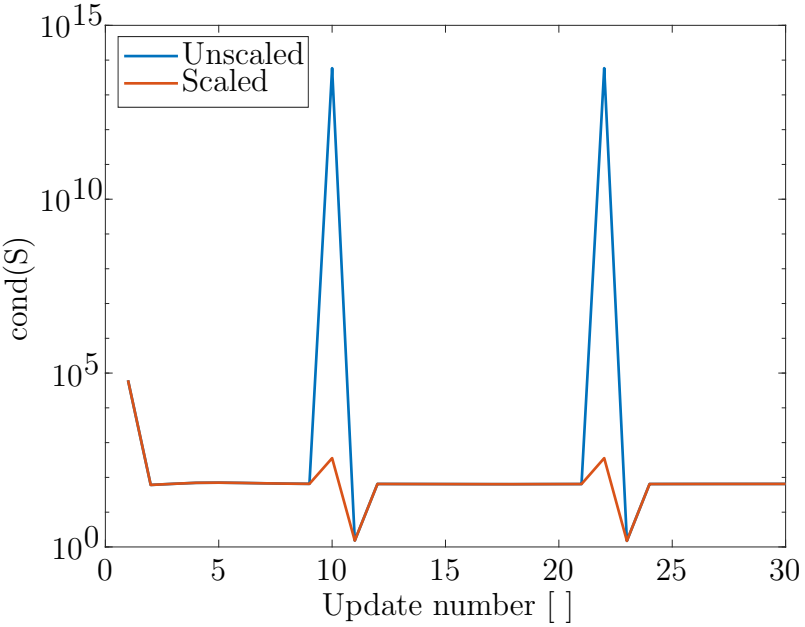


Figure 7: Condition number of $S = HPH^T + R$ of all updates (IMU, GNSS position and GNSS velocity) with and without scaling of GNSS position measures with a factor of 10^8.

Fig. 7 shows the condition number of the matrix $S = HPH^T + R$ for each update for scaled and unscaled position measures. S is the matrix that is inverted in Eq. (10), this operation is responsible for an important part of the propagation of roundoff errors. GNSS velocity measures correspond to the low values with $cond(S) \approx 10^0$. Values with $cond(S)$ at approximately 10^2 correspond to IMU measures. The peaks in the unscaled measures correspond to position measures, by scaling the GNSS measures the condition number almost drops to the same level as IMU measures.
Conservatively, we have a decrease in 10 orders of magnitude during the GNSS update. According to Eq. (15) we diminish the propagation of roundoff errors by a factor of 10^{10}.
The application of scaling did not show any significant changes in the performance of the filter. The very complex model used for the filter makes it quasi impossible to detect and observe the changes in numerical stability. Probably the errors disperse themselves across the states and are mitigated through the following updates. Nevertheless it could be interesting to verify on the final product if differences occur between the scaled and unscaled version.

4.5 Intermediate conclusion

Potential sources of roundoff errors have been identified. By rescaling specific states the ill-conditioned nature of the initial covariance matrix has been diminished. This should reduce the occurrence of problematic roundoff errors during prediction. Because only rescaling states does not change the inverted part of the Kalman gain, GNSS measurements have also been rescaled. Leading to a significant diminution in the propagation of roundoff errors.
Due to the complexity of the model and the nature of roundoff errors the potential increase in accuracy could not be quantified. Especially since the analysis of the results would be done

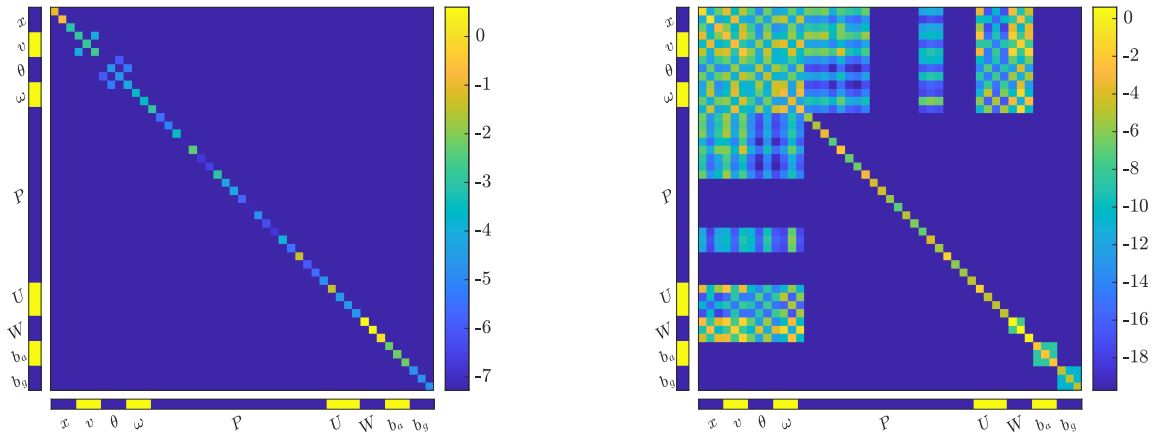
with the same precision as the computation, differences in results at this level are almost impossible to detect.
But as already explained by Biermann [13]: "Even when catastrophic illness does not occur there is diminished health". As in our case no clear filter divergence due to roundoff errors was detected, the unscaled version was probably still working in a less optimal way.
Should there still be a suspicion of numerical instability in the filter, one may resort to additional stabilization methods. In that case implementations such as square-root filtering [14] or singular value decomposition Kalman filters [15] could be used.

5 Stabilization of the parameters

This part will focus on different phases of the estimation where the current build up of the filter shows some particular behaviors. Specifically the initialization and GNSS outage are analysed.

5.1 Problem statement

In the current VDM implementation, 47 states are estimated using 12 observations: 3 accelerations, 3 rotation velocities, 3 local velocities, 3 absolute positions. From these states 21 are parameters that define constants in the aerodynamic model used in the prediction step of the filter.
At the start of the estimation the covariance matrix undergoes strong changes, due to the lack of initial covariance between the different states. The strong changes in P_0 can already be suspected when inspecting Fig. 6, where in the beginning the condition numbers drop sharply. The absence of correlation information in the beginning of the filtering has as consequence that the estimator ignores links between certain states. During the first few updates the filter builds up the correlation between states while simultaneously estimating the states. This leads that during the first updates many states are updated as if they where independent. In further timesteps the filter may tend to a local minimum, which can lead to reduced navigation accuracy.



(a) Initial covariance matrix

(b) Covariance matrix after one prediction

Figure 8: Covariance matrix after one prediction

In the current implementation the covariance matrix is initialized with a quasi diagonal matrix (Fig. 8a).
We can note in Fig. 8b that the covariance of the block corresponding to the VDM parameters is not changed in the first prediction step. Because of the static nature of the VDM parameters the process model will not add any correlation between them. The blank in the parts corresponding to the covariance between parameters and navigation states correspond

to parameters of the roll and yaw moment respectively. Due to the initialization to 0 of the wind velocity and the absence of actuator inputs for the control surfaces as well as no rotation rates in the first iteration of the filter these parts do not contribute to the prediction of the states at this point. Therefore the covariance values between these parts will stay small or close to zero at the beginning.

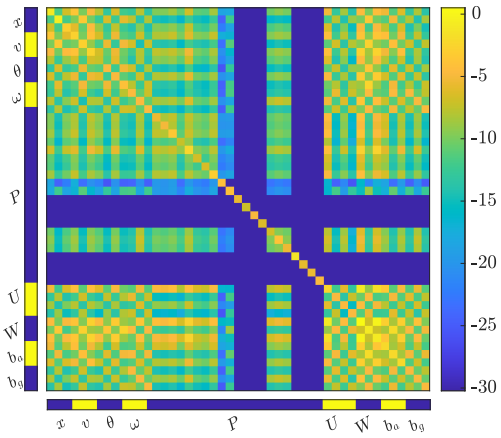


Figure 9: Covariance matrix after one prediction and update

After the first IMU update (Fig. 9) one can observe that correlations between parameters that showed covariance with the navigation states were added. Essentially these correlations are induced by the filter through the update. The main component responsible for the induced covariance is the linearization of the measurement function.

Example 5.1 *States that are combined to obtain the innovation will have a modification in their covariance through the update. Let there be the following matrices used in an update:*

$$P = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_x^2 \end{bmatrix}, R = \sigma_m^2 \text{ and } H = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

The gain becomes:

$$K = \frac{\sigma_x^2}{2\sigma_x^2 + \sigma_m^2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = C \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

And the covariance update yields:

$$\hat{P} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - C \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right) \tilde{P} = \begin{bmatrix} 1-C & -C \\ -C & 1-C \end{bmatrix}$$

We can observe that if the measure can be written as a combination of 2 states these will show negative correlation after an update.

The moments are never directly observable as the gyroscopes only measure rotation velocities. The updates of these states will only be controlled by their covariance with other states.

Example 5.2 *Correlated states that have zero elements in the measurement function H will not contribute to the innovation and its uncertainty, but will be updated according to their covariance.*
Let there be a problem with:

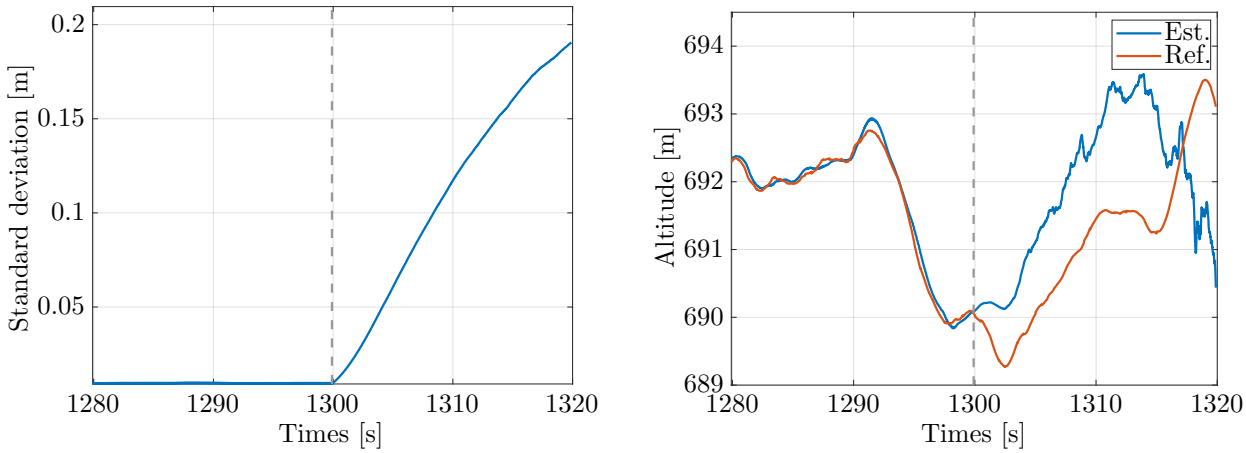
$$P = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix}, R = \sigma_m^2 \text{ and } H = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The gain becomes:

$$K = \begin{bmatrix} \sigma_x^2 \\ \sigma_{xy} \end{bmatrix} \frac{1}{\sigma_x^2 + \sigma_m^2}$$

We can note here that the second state is only updated as a function of its covariance.

The covariance that will control the updates of the parameters corresponding to the moments will be added through the process model.
Initialization errors in navigation states or for example the wind can also lead to strong updates and to unwanted corrections in the parameters. Due to the absence of process noise on the VDM parameters false updates at the start can be hard to recover from. The absence of process noise leads to a very rapid lowering of variance on these states, as a consequence the coefficients will later on be considered as very precise and receive less corrections. If it can be assumed that the parameters are known well enough and that the main error source will come from other states, it could be beneficial to first let the filter roughly estimate such states and build up the covariance matrix before correcting the VDM parameters.
During outages it can be observed that the estimated trajectory is subject to some erratic movements. This is linked to the fact that the navigation parameter's variance becomes unbounded or at least increases strongly when no GNSS corrections are available. A direct consequence of this is that these states will be updated more aggressively from sensor measures. This behavior can be seen in Fig. 10, along with how the uncertainty grows. This phenomena can not be observed in INS based navigation as here the IMU measures are directly integrated through the process model, which is independent of the uncertainty of the states.



(a) Standard deviation of altitude

(b) Estimated altitude

Figure 10: Altitude and corresponding uncertainty, the grey line is the last GNSS measure

5.2 Solution approach

A possible approach to stabilize the VDM parameters is the usage of the so called Schmidt-Kalman filter or 'consider' filter. The main usage of this method is to reduce the computational complexity by only implicitly estimating biases or constant parameters [16]. This could be for example the case in gyroscope biases or misalignments. The Schmidt-Kalman filter ignores the corrections of the 'consider' states. But, to account for the uncertainty of these states, their variance and covariance is still propagated in the prediction step [3].
A technic which is often used is to remove the state completely and include process noise to account for the increased modelling error[3]. This reduces the computational complexity further but does only partially account for the ignored parameter in the gain calculation.
In 2017, Brink [16] showed that simply resetting the states and the corresponding block of the covariance matrix results in an application of the Schmidt-Kalman filter. A partial update is also possible by using the following equations developed in [16]:
In a Schmidt-Kalman filter the states are separated in two groups or blocks, x is updated as usually and y are the considered states

$$X = \begin{bmatrix} x \\ y \end{bmatrix} \tag{26}$$

$$P = \begin{bmatrix} P_{xx} & P_{xy} \\ P_{yx} & P_{yy} \end{bmatrix} \tag{27}$$

A normal update is applied to all the states.

$$\hat{P} = (I - KH)\tilde{P} \tag{28}$$

$$\hat{X} = \tilde{X} + K(z - H\tilde{X}) \tag{29}$$

After the update the considered states are recomputed as a function of γ. γ is a weighting factor between 0 and 1, which can ponderate between a full and a consider update.

$$\hat{y}^c = \gamma\tilde{y} + (1-\gamma)\hat{y} \tag{30}$$

$$\hat{P}^c_{yy} = \gamma^2\tilde{P}_{yy} + (1-\gamma^2)\hat{P}_{yy} \tag{31}$$

$$\hat{P} = \begin{bmatrix} \hat{P}_{xx} & \hat{P}_{xy} \\ \hat{P}_{yx} & \hat{P}^c_{yy} \end{bmatrix} \tag{32}$$

$$\hat{X} = \begin{bmatrix} \hat{x} \\ \hat{y}^c \end{bmatrix} \tag{33}$$

The advantage of this approach is mainly that this allows for an easy implementation and a quick switch between full and consider filter. Although this implementation does not present any improvements in computational complexity, it allows at this state to quickly and simply modify the considered states.
The additional code snippet used for the implementation can be found in listing 1. With the variable 'gamma' it is possible to adjust the fraction of used update.

```
1
2  c_states = [1:3]; % indexes of consider states
3  gamma = 0; %weighting function
4
5  if obj.gnssOutage
6
7      dX_k(c_states) = gamma*dX_k(c_states);
8      Phat(c_states, c_states) = gamma^2*Phat(c_states,c_states)+(1-gamma^2)*
       Ppred(c_states,c_states);
9
10 end
```

Listing 1: Code snippet for Schmidt-Kalman implementation

At startup it is proposed to reduce the correction given to the VDM parameters to reduce strong changes in the parameters at the beginning of the estimations. This "Soft-start" should allow for a reduction of erroneous corrections on a part of the states when the filter still ignores the correlations between them. By still keeping a part of the updates the filter is still capable of restructuring the covariance matrix. To evaluate the effectiveness of this method, this approach is compared to a sparse initialization, i.e where no warm up is performed, as well as a full initialization. For the full initialization the correlations between VDM coefficients are extracted from the final covariance matrix of a previous flight.
We recall quickly the link between covariance σ_{xy} and the correlation ρ_{xy}:

$$\rho_{xy} = \frac{1}{\sigma_x}\sigma_{xy}\frac{1}{\sigma_y} \tag{34}$$

To obtain the correlation matrix through direct operations we can compute:

$$\rho = \Sigma^{-1} P \Sigma^{-1} \tag{35}$$

Where Σ is a diagonal matrix containing the standard deviations of the states.
Using the initial variances it is possible to compute a covariance matrix which takes into account the correlations between VDM coefficients from the previous flight. The usage of this method requires the usage of the coefficients from the same flight.
If a GNSS outage occurs, due to a lack of available satellites or to high uncertainty on the measure, which could be predicted by using for example receiver autonomous integrity monitoring, it is proposed to only consider the positional navigation states. By doing so the position will only be computed by the integration of velocity and should stay smooth. Because these states are the "final" result of the process model, this usage should not influence the other states.

5.3 Results

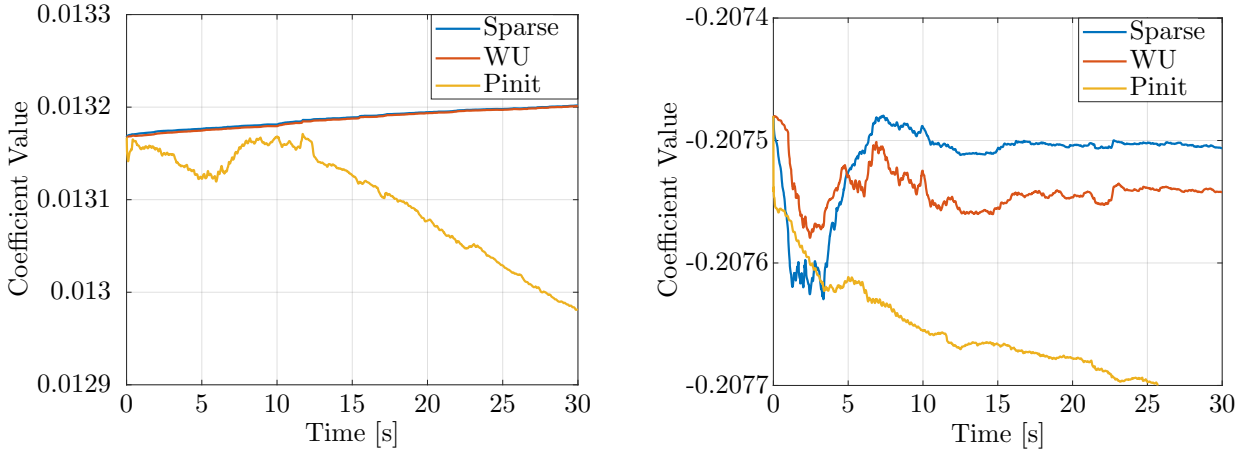
Tab. 4 shows the average RMSE of velocity and orientation for 100 Monte Carlo runs during the first 30 seconds of a straight line with different initialization methods. For the Monte-Carlo analysis, sensor noises and initialization errors for the navigation states have been recomputed for each run. Additional errors that are present are the initialization of wind and bias values to 0. Three different warmups have been tested. "Warmup $\gamma = 0.2$, $T = 1[s]$" and "Warmup $\gamma = 0.2$, $T = 5[s]$" both partially update all VDM coefficients and IMU error states for 1 and 5 seconds respectively. "Warmup $\gamma = 0.2$, $T = 1[s]$ reduced " only applies partial updates on the VDM coefficients. For the full initialization a matrix from a 15 min flight has been used for the correlation between coefficients. All initial states were taken from the previously mentioned flight with a standard deviation corresponding to 1 % of the coefficients value.

Run	RMSE V_{body} $[m/s]$	RMSE *Attitude* $[rad]$
Sparse	0.735	0.034
Full init.	0.363	0.017
Warmup $\gamma = 0.2$, $T = 1[s]$	0.633	0.029
Warmup $\gamma = 0.2$, $T = 5[s]$	0.838	0.039
Warmup $\gamma = 0.2$, $T = 1[s]$ reduced	0.666	0.031

Table 4: Average RMSE of 30 first seconds of a flight

The warmup of a duration of 1 second and a $\gamma = 0.2$ shows the best performance of all warmups. With full updates on error states during the warmup leads to an increase in navigation error. A hypothesis for this behavior is that the error states start to account for errors due to the initial underestimation of the wind. The warmup of 5 seconds also shows an increase in navigation error. Probably restraining the updates to long will increase the errors due to the VDM coefficients, while modifying navigation states and wind to fit the measures. Essentially this means that the filter will adapt these states in a way that the VDM and the measures match. The partial updates reduce the large updates on the VDM

coefficients as can be seen in Fig. 11b. A secondary effect is that, because the variance of the consider states does not diminish, the estimated uncertainty of navigation states will be higher. This means that at start-up the filter will depend slightly more on the sensors than the other initialization methods.



(a) Evolution of coefficent $C_{F_z 1}$

(b) Evolution of coefficent $C_{M_x \omega_x}$

Figure 11: Two examples of parameters for different initialization methods: a sparse initialization, a warmup with 2 seconds of partial updates ($\gamma = 0.2$) on the VDM coefficients and sensor biases and an initialization with correlations from a previous flight. The shown data is the mean of all 100 MC.

In Fig. 11 we can see that, although, the full initialization and the warm-up show a diminution in RMSE, the coefficients evolve in different manners.
Generally, the evolution of the states due to the warm-up is very similar to a sparse initialization. As some correlations will only appear in certain maneuvers later on, the updates in a full initialization will sensibly differ from the other methods, where these correlations are not present at the beginning of the estimation. The effect of those correlations can be seen in the the first updates already, where the states of the full initialization moves away from its initial value more rapidly than the other methods. But we are also able to see that in this case the coefficient $C_{M_x \omega_x}$ is behaving differently for all three methods. In the sparse initialization the coefficient is lowered at start and returns close to its initial value after a few seconds, this phenomena is efficiently damped by the partial updates.
The application of a Schmidt-Kalman on the position states during GNSS outages stopped the erratic behavior of the trajectory. Fig. 12 shows 130 seconds of flight with 2 minutes of outage. It is clearly visible that by computing position only through integration the trajectory becomes smooth.

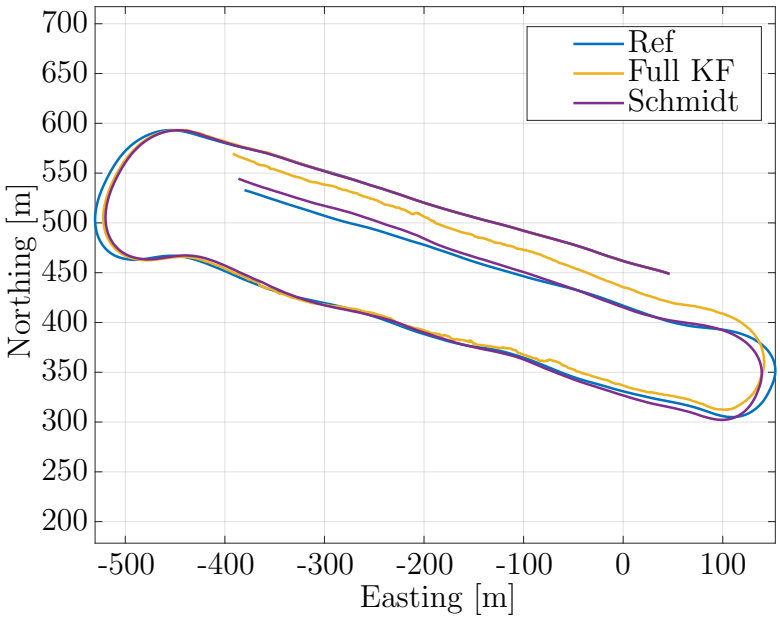


Figure 12: Plane view of a 2 min outage of a 20 min flight with and without updates on the position states.

On this example it seems that the positioning error is lower for the Schmidt-Kalman filter. Tab. 5 shows the RMSE of position and each of its component during outage.

Run	Position [m]	Northing [m]	Easting [m]	Down [m]
Full KF	27.9	19.8	16.1	11.2
Schmidt-Kalman	17.9	8.2	15.6	3.7

Table 5: RMSE with and without considering position during outage

This result is quite unexpected and may only be valid in specific cases. A possible hypothesis in this case is that the wind is badly estimated, leading to updates which creates a drift to the North through the updates. But for this case we have a global decrease of error in all components of position. Altitude seems to be a main component affected by the Schmidt implementation. The difference when considering position is a further increase in position uncertainty, because the IMU update does not decrease the corresponding variances.
But because the elements corresponding to position in the measurement matrix are zero this does not influence the gain computation and the update of the other states. This can be deducted by generalizing Ex. 5.2, where neither the variance nor the covariance of a state that is not contributing to the innovation influences the covariance matrix of the innovations. Tab. 6 shows the RMSE of body velocity and orientation, as suspected these values are similar. It is therefore very likely that the difference in navigation performance is only due to the updates on position states.

Run	V_{body} [m/s]	*Orientation* [rad]
Full KF	1.0533	0.28675
Schmidt-Kalman	1.0533	0.28675

Table 6: RMSE with and without considering position during outage

5.4 Intermediate conclusion

For initialization three methods have been compared: Initialization with a quasi diagonal matrix, the application of partial updates at the beginning and the usage of correlations from a previous flight. Using partial updates during the start of the filter showed a reduction in the RMSE in the beginning that is equivalent to the initialization with correlations from a previous flight. The warmup decreases large updates at the beginning. Its application seems to have repercussions on the later evolution of the states. But its usage requires a careful choice of the time at which to apply partial updates and especially on which states these should be performed. Never the less this is not equivalent to an initialization with a priori computed correlations. Especially as some correlations only become apparent during certain maneuvers. This has both supposed disadvantages and advantages, on one hand the filter will ignore important correlations between parameters that may appear in certain situations only. On the other hand the matrix used for the initialization will be strongly dependant on the flight used to compute it, especially since it is necessary to use the coefficients from the same flight. Because the effects of the two alternatives to a sparse initialization seem to differ in their behaviors a combination of both methods might seem even more beneficial.
Only considering the positions during outage does reduce the erratic changes in trajectory while uncertainty becomes unbounded. Its benefits to navigation accuracy are questionable but a smooth trajectory is beneficial for the usage of an autopilot. For future usage it is necessary to evaluate the effects that this implementation has on the performance of the filter, by using for example Monte-Carlo analysis. Should the performance increase during outage, it may be interesting to investigate the possibility to apply the Schmidt-Kalman filter during the whole flight. This method could also be applied to other states, for example wind velocity. Nevertheless, in such a case it seems necessary to carefully track the evolution of other states which might be influenced by an increasing uncertainty.
An aspect, where partial updates might prove to be also beneficial, is the calibration of the VDM coefficients. Such an approach has showed promising results in the calibration of the lever-arm and attitude offset of a camera-IMU couple [16]. The application of the Schmidt-Kalman filter in its intermittent implementation creates generally a whole new range of possibilities. Especially in multi-sensor systems it allows to redirect quite efficiently the updates on certain states. Such a usage might become beneficial when using data that originates from flights that use additional sensors such as barometers, pitot tubes.

6 Model reduction

VDM navigation currently requires a high number of parameters. This implies a heavy computational load, which may for example reduce the frequency at which the embedded system can compute results. Additionally a high number of parameters makes it difficult for the filter to estimate accurately the different coefficients of the VDM.

6.1 Problem statement

The coefficients of the dynamic model in VDM navigation are strongly correlated. Although correlations between parameters do not pose a problem for the filter, strong correlations show that it is possible to reduce the number of parameters in the model. This possibility has been proposed in [8].
Explicitly the goal of this part is to reduce the number of parameters needed in the vehicle dynamic model while maintaining a acceptable precision in the estimation of vehicle localization and attitude.

6.2 Solution approach

Recall Example 5.2 where it was possible to see that during updates correlated parameters are corrected in the same manner. If the covariance between two states is high enough with respect to the variances and other covariances of the states, the update values of these states will be very similar. Meaning that they will evolve in a more or less linear manner.
To assure the usage of correct parameters this part is done with help of simulations. It is assumed here that the VDM is accurate for a real application and that the errors due to the reduction of the model will behave in the same way in a real application.
To identify potential candidates first the estimated correlation matrix from a 15 min flight is analysed. Once potential candidates are found these are related through a linear relation or a simple scaling factor.

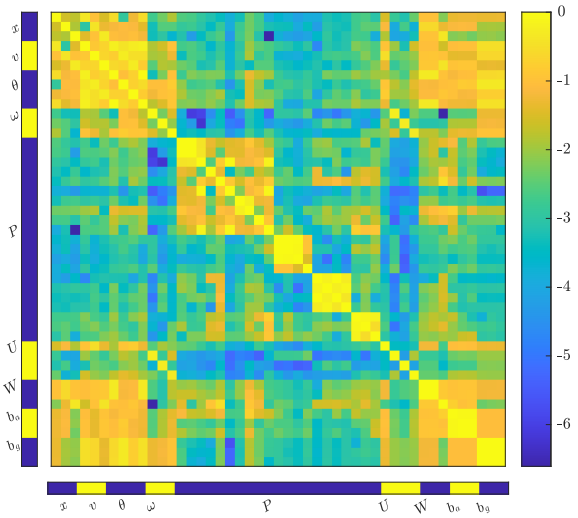


Figure 13: Correlation matrix used for the choice of candidate pairs

Pairs with an elevated correlation are analysed in more detail. For this the joint evolution of the pairs is observed and if possible a linear regression is computed to approximate the second parameter.
By constraining the filter it is expected to see a degradation in the navigation accuracy. Therefore the selection of constraints or reductions will principally be the ones that show a low change in performance.
For each chosen pair of states a new model is computed. It is expected that such changes will also affect the other parameters. To reduce the errors to false initial parameter values the filter is trained on a 15 min flight (Trajectory 1, Fig. 3) without wind and sensor noise. The uncertainties and noises related to wind and sensor errors are lowered for the calibration. The final parameters of this calibration flight are then used on a validation trajectory (Fig. 4), with a lower initial uncertainty (1% of the initial value of the coefficient). For validation sensor noises and wind are added to the simulation. The last 2 minutes of the trajectory are performed with a simulated GNSS outage to test the autonomous navigation performance. To compare directly the VDM performance, process noise on the navigation states was removed during the whole procedure.
This procedure is also applied to an unchanged model to obtain a reference on the errors.
As indicators of performance the root mean square error of attitude and velocity in the body frame during outage are computed.
The choice not to directly compare the position of the UAV is made because these errors are correlated in time and combine both attitude and velocity errors.

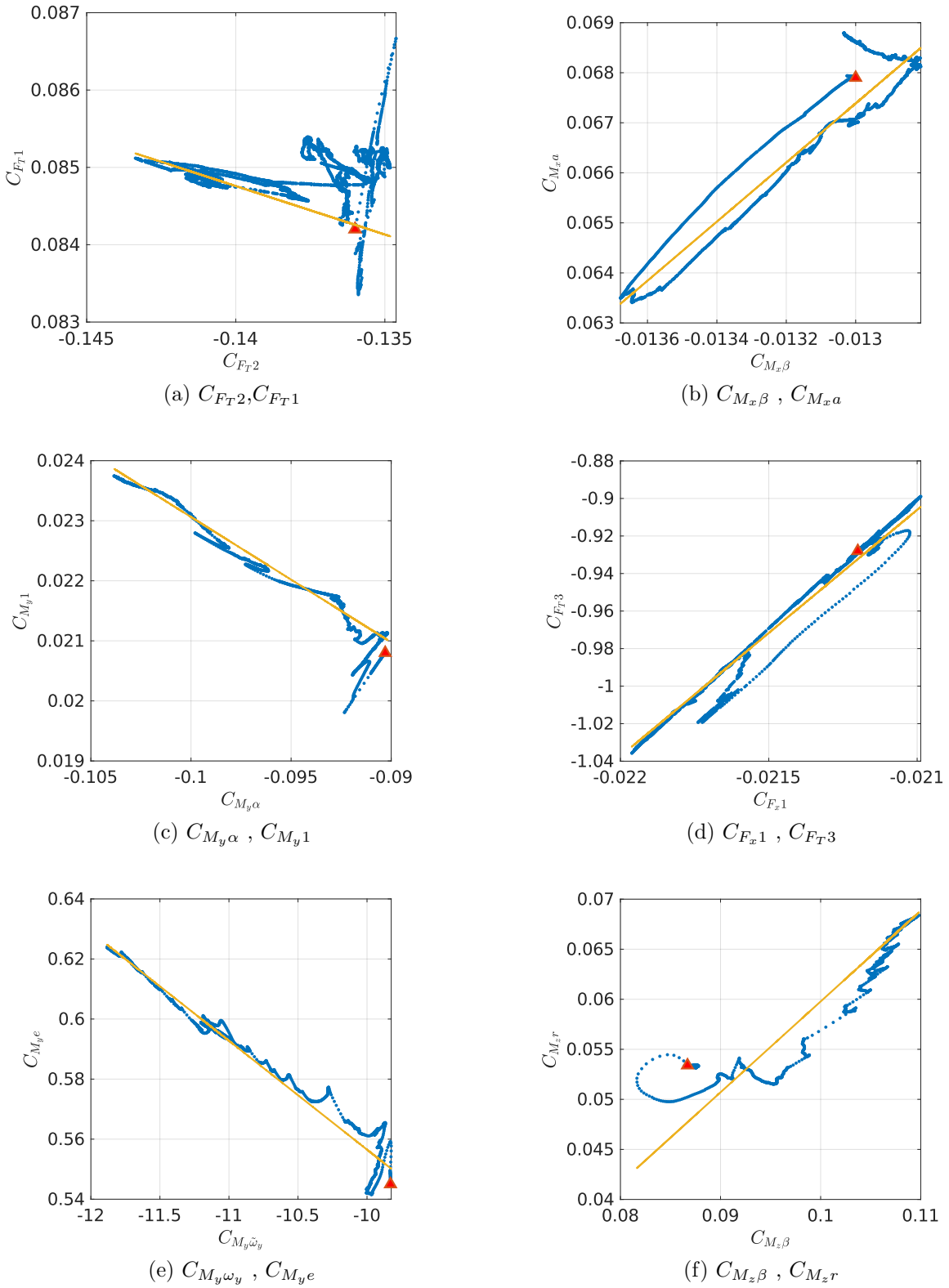
6.3 Identification of potential candidates

To find potential candidates first the correlation matrix after a 15 min flight is examined.

Parameter 1	Parameter 2	Correlation
$C_{F_T 2}$	$C_{F_T 1}$	0.96
$C_{M_x \beta}$	$C_{M_x a}$	0.94
$C_{M_y \alpha}$	$C_{M_y 1}$	0.94
$C_{M_x \omega_x}$	$C_{M_x a}$	0.89
$C_{M_x \omega_x}$	$C_{M_x \beta}$	0.87
$C_{F_x 1}$	$C_{F_T 3}$	0.80
$C_{M_y \omega_y}$	$C_{M_y e}$	0.78
$C_{M_z \beta}$	$C_{M_z r}$	0.75

Table 7: VDM Parameters with a high correlation

Tab. 7 shows pairs with a correlation above 0.75. It is interesting to notice that most parameters are those corresponding to the moments. The roll moment has 3 pairs in the highest correlations. To not constrain the model to much the only the pair $C_{M_x \beta}$ - $C_{M_x a}$ has been retained from this triplet.
Another observation to be made is that only the pair $C_{F_x 1}$ - $C_{F_T 3}$ creates a strong relation between two equations of the VDM. These two states have the same dimensionality (see Eq. (1) and Eq. (2)) and would act the same way if the angle of attack and the sideslip angle would be neglected.
To decide on the coefficients of the linear relation which will link the parameters, a regression on the joint evolution is made. We can observe that most parameters show a very linear trend. These pairs can be replaced by a linear relation.



(a) C_{F_T2},C_{F_T1}

(b) $C_{M_x\beta}$, $C_{M_x a}$

(c) $C_{M_y\alpha}$, C_{M_y1}

(d) C_{F_x1} , C_{F_T3}

(e) $C_{M_y\omega_y}$, $C_{M_y e}$

(f) $C_{M_z\beta}$, $C_{M_z r}$

Figure 14: Joint evolution of different parameter pairs, the triangle shows the initial values.

In Fig. 14 it becomes clear that the suspected linearity between the parameters is present. It is also observable that initially certain parameters don't show a linear trend, this can be explained by the fact that correlations in between them appear only during certain maneuvers. This could be seen in Fig. 9, where parameters related to the roll and yaw moments did not show any correlations after the first update. Some temporary trends can be explained by the influence of other correlations in the model.
Tab. 8 shows the chosen pairs with the empirical relation. The coefficients have been computed using linear regressions. Intervals where the relations seem linear were chosen manually, to exclude the influence of parts were correlations are not present.

Name	Chosen pair	Relation
Full model	-	-
FT1-FT2	C_{F_T1} - C_{F_T2}	$C_{F_T1} = -0.1235\ C_{F_T2} + 0.0672$
FT-Fx	C_{F_T3} - C_{F_x1}	$C_{F_T3} = 131.2\ C_{F_x1} + 1.85$
Mxa-Mxb	$C_{M_x\beta}$ - C_{M_xa}	$C_{M_xa} = 5.905\ C_{M_x\beta} + 0.144$
My1-Mya	$C_{M_y\alpha}$ - C_{M_y1}	$C_{M_y1} = -0.209\ C_{M_y\alpha} + 0.002$
Mye-Mywy	$C_{M_y\omega_y}$ - C_{M_ye}	$C_{M_ye} = -0.036\ C_{M_y\omega_y} + 0.194$
Mzb-Mzr	$C_{M_z\beta}$ - C_{M_zr}	$C_{M_zr} = -0.907\ C_{M_z\beta} - 0.031$

Table 8: Proposed reductions

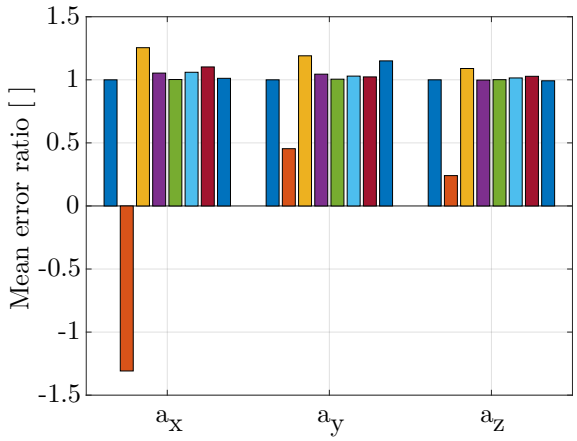
6.4 Results

The results of the proposed procedure are shown in Tab. 9. Because the reduction on the coefficients C_{F_T1} - C_{F_T2} showed the highest errors it has been decided to not use it in the combination.

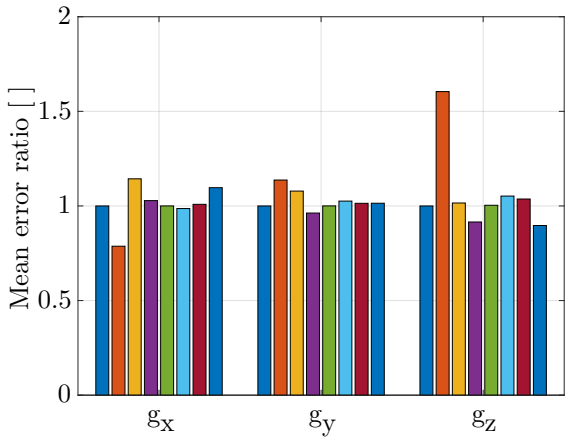
Model	RMSE V_{body} [m/s]	RMSE *Attitude* [*rad*]
Full model	0.329	0.372
FT1-FT2	0.340	0.409
FT-Fx	0.331	0.372
Mxa-Mxb	0.328	0.366
My1-Mya	0.329	0.366
Mye-Mywy	0.329	0.372
Mzb-Mzr	0.330	0.372
Combination	0.326	0.372

Table 9: Root mean square errors of different models

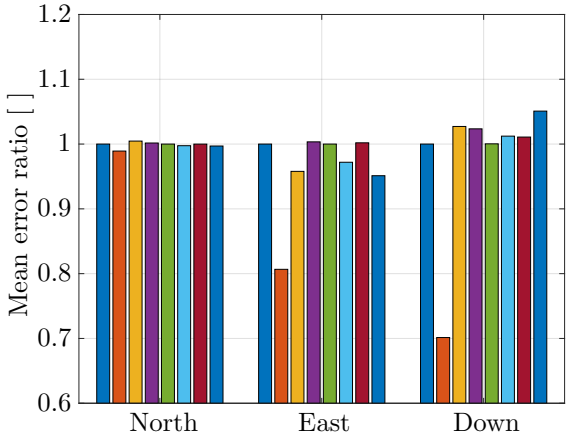
In Tab. 9 we see that the errors compared to the full model stay in a very low range. Which is not surprising as the chosen parameters showed a very linear joint evolution. This means that even by constraining the model the filter is able to model well enough the behavior of the UAV. The model C_{F_T1} - C_{F_T2} creates the largest difference with respect to the full model. This is surprising as this couple showed a correlation of 0.95.

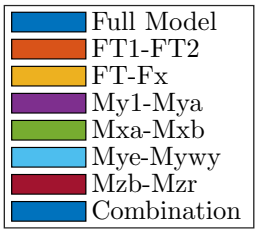


(a) Error ratio in estimated accelerometer biases



(b) Error ratio in estimated gyroscope biases





(c) Error ratio in estimated wind velocity

Figure 15: Ratio of errors in estimated biases and wind values during outage with respect to the complete model

Biases and wind are modeled as random walks, therefore they are more prone to take up other errors due to a higher uncertainty. Variations in these values were observed during the comparison of the models, suggesting that they compensate the differences in the process model to fit the measurements. This implies that they will probably not reflect the real values but act as a sort of buffer for the small differences in the unmodeled states. These errors in the estimation manifest themselves as biases in those values. The mean error allows to compare these differences while removing the zero-mean white noise in sensors and rapid variations in wind velocity. Fig. 15 shows the mean errors of sensor biases and wind values during outage, the values are divided through the mean of the full model to allow for a simpler comparison.
We can see that the simplification on the couple C_{F_T1} - C_{F_T2} shows the largest differences again with the other models. Especially we can see that the errors in the estimated wind are lower in this case. This seems to be related to the lower errors in the accelerometer biases

on the y and z axis.
Reductions linked to moments show only very light differences with respect to the full model. Their updates being mainly driven by their covariance with other states it seems reasonable that in this case the covariance is adapted. Therefore even more simplifications might be possible for these parts.
Because the same uncertainties and noises were used for each model. Specific tuning for each model might show different results in the navigation error.
It is suspected that the initialization of the filter will also benefit from such reductions. As seen in the previous part some correlations only appear during certain maneuvers, in the reduced models this information is already present, although exaggerated, through the regression.

6.5 Intermediate conclusion

It has been shown that some parameters are very strongly correlated. This leads to a joint evolution that can be well approximated by linear regressions. Empirical relations have been computed and used to replace certain coefficients of the model. The navigation performance of the reduced models showed little to no degradation. In some cases a small increases could even be observed.
The elements mainly affected by this change where the wind and the sensor error states. It could be observed that the reductions related to moments, generally showed lower differences than simplifications on the force components.
By linking states through different empirical relations the computational complexity has been reduced.
Although these results look very promising, the coefficients of the linear regression are computed from simulation. Variations in the tuning or the usage in a real application may not show the same relations.
It is to note here also that for the purpose of this test all models were used with the same process noises as well as with a similar initial uncertainty. Individual tuning of the models for calibration and evaluation may result in different results.

7 Conclusion

This work assessed the computational performance of VDM-based navigation as proposed by [7]. Particularly the stability of the filtering aspects were analysed, to mitigate the risk of failures or divergence.
In a first part the numerical stability of the problem was investigated. The problem as originally posed showed strong signs of ill-conditioning. The position states in latitude and longitude were identified as main contributors to a very high condition number. To reduce the ill-conditioning of the covariance matrix it has been decided to rescale the units of the horizontal components of position as well as the state corresponding to propeller speed. This change led to a much more acceptable condition number in this matrix. By scaling the GNSS position measures the ill-conditioning of these updates has been remedied. This aspect is crucial as a major contributor to round-off errors is the inversion of the expression HPH^T+R. Overall no change in filter performance could be observed. But it seems reasonable to keep these changes as they should reduce the risk of computation errors or failures.
A second part analysed two particular situations during filtering: the initialization and GNSS outage. When initializing the filter with a sparse matrix the filter ignores links between states at the beginning. The presence of initial errors as well as the missing correlations in the initial covariance matrix lead to problematic updates, which can have repercussions on the navigation performance later on. A first method to remedy the problem, consisted of applying a partial-update Schmidt-Kalman to the VDM coefficients and the sensor errors. A second method, which consisted of reusing part of the correlation matrix from a previous flight, to reduce the effects of missing correlations was examined. Monte-Carlo analysis showed that the navigation performance is increased in the beginning of the flight when applying either of these two methods. Nevertheless the behavior of the filter differed sensibly between these two methods. Indicating that it might be possible to benefit from both methods simultaneously. During GNSS outage the uncertainty of navigation states increases considerably, leading to very aggressive updates. These manifest themselves amongst other things by erratic changes in the position states. Applying a Schmidt-Kalman filter on position during outages, therefore only computing these states through integration of velocity, allowed to obtain a smooth trajectory. On the studied case the position precision increased considerably without having any effects on the other states. The diminution in position error might be a particular case. For further usage of this implementation it seems necessary to further investigate the effects of this phenomena.
In a third part the possibility to reduce the model was analysed. Strongly correlated coefficients showed the possibility to reduce the number of parameters used with a minimal loss in accuracy. By using the estimated correlation of the filter potential candidates have been identified. Further investigation of these pairs that very clear linear relations were present between them. By linking the parameters with linear regressions the number of parameters has been reduced. An assessment of estimation accuracy showed that the errors due to the reduction are mostly negligible. Especially the change in model modified the estimation of wind and sensor errors. It is suspected that these reductions also stabilize the filter at the beginning of the flight. During initialization the missing correlations of the full model are already present through the linear relation.

References

[1] M. S. Grewal and A. P. Andrews, "Applications of kalman filtering in aerospace 1960 to the present [historical perspectives]," *IEEE Control Systems Magazine*, vol. 30, no. 3, pp. 69–78, Jun. 2010, ISSN: 1941-000X. DOI: 10.1109/MCS.2010.936465.

[2] S. S. Alex, A. E. Daniel, and B. Jayanand, "Reduced order extended kalman filter for state estimation of brushless DC motor," in *2016 Sixth International Symposium on Embedded Computing and System Design (ISED)*, Dec. 2016, pp. 239–244. DOI: 10.1109/ISED.2016.7977089.

[3] J. A. Farrell, *Aided Navigation: GPS with High Rate Sensors*. McGraw Hill Professional, Apr. 3, 2008, 554 pp., ISBN: 978-0-07-164266-8.

[4] P. Muraca and C. Picardi, "A reduced order extended kalman filter algorithm for parameter and state estimation of an induction motor1," *IFAC Proceedings Volumes*, IFAC Workshop on Algorithms and Architectures for Real-Time Control, Seoul, Korea, 31 August - 2 September, vol. 25, no. 20, pp. 225–230, Sep. 1, 1992, ISSN: 1474-6670. DOI: 10.1016/S1474-6670(17)49866-1.

[5] M. S. Grewal and A. P. Andrews, "Application of variants of the kalman filter to various programs," *ION 58th Annual Meeting*, p. 4, 2002.

[6] M. S. Grewal, L. R. Weill, and A. P. Andrews, *Global Positioning Systems, Inertial Navigation, and Integration*. John Wiley & Sons, Apr. 5, 2004, 416 pp., ISBN: 978-0-471-46386-3.

[7] M. Khaghani, "Vehicle dynamic model based navigation for small UAVs," Infoscience, PhD thesis, EPFL, 2018. DOI: 10.5075/epfl-thesis-8494.

[8] G. F. Laupré and J. Skaloud, "On the self-calibration of aerodynamic coefficients in vehicle dynamic model-based navigation," Drones, Jul. 12, 2020. DOI: 10.3390/drones4030032.

[9] M. Khaghani and J. Skaloud, "Application of vehicle dynamic modeling in uavs for precise determination of exterior orientation," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 41B3, pp. 827–831, Jun. 1, 2016. DOI: 10.5194/isprs-archives-XLI-B3-827-2016.

[10] M. Grewal and A. Andrews, *Kalman filtering: theory and practice. Incl. 1 disk*. Prentice Hall, Jan. 1, 1993, 381 pp., ISBN: 0-13-211335-X.

[11] Floating-point numbers - MATLAB & simulink, [Online]. Available: https://www.mathworks.com/help/matlab/matlab_prog/floating-point-numbers.html (visited on 01/08/2021).

[12] M. Verhaegen and P. V. Dooren, "Numerical aspects of different kalman filter implementations," *IEEE Transactions on Automatic Control*, vol. 31, no. 10, pp. 907–917, Oct. 1986, ISSN: 1558-2523. DOI: 10.1109/TAC.1986.1104128.

[13] G. Bierman and C. Thornton, "Numerical comparison of kalman filter algorithms: Orbit determination case study," *Automatica*, vol. 13, no. 1, pp. 23–35, Jan. 1, 1977, ISSN: 0005-1098. DOI: 10.1016/0005-1098(77)90006-1.

[14] M. Morf and T. Kailath, "Square-root algorithms for least-squares estimation," *IEEE Transactions on Automatic Control*, vol. 20, no. 4, pp. 487–497, Aug. 1975, ISSN: 0018-9286. DOI: 10.1109/TAC.1975.1100994.

[15] M. V. Kulikova, "Numerically robust SVD-based kalman filter implementations," in *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*, Oct. 2018, pp. 170–175. DOI: 10.1109/ICSTCC.2018.8540648.

[16] K. M. Brink, "Partial-update schmidt–kalman filter," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 9, pp. 2214–2228, May 18, 2017. DOI: 10.2514/1.G002808.

[17] Dimitry Savaransky, *UDFactor*, 2008. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/20600-udfactor (visited on 10/15/2020).

8 Appendix

8.1 Additional changes in the code

This part lists shortly other small changes or additions in the code made during the project.

8.1.1 Changes in Simulator

The guidance system has been modified to take into account distance to the next waypoint. This reduces rapid changes in the input command for propeller speed. A script which only computes the necessary data for the estimator, without running the Kalman filter of the simulator has been added.

8.1.2 Changes in the Estimator

Addition of the variable `filter_method` to change between methods i a switch-case statement. Current methods are:

- `normal`: Classic Kalman filter.
- `sym`: Forcing symmetry with : $\frac{P+P^T}{2}$, method used throughout this work.
- `serial`: Serial processing of measurements.
- `tobi`: Thornton-Bierman factorization, based on [10], needs to be checked.
- `svd`: SVD-based mechanization, based on [15], needs to be checked.

8.2 Extracts of code changes

Addition to method EKF() in EKF.m

```
%Initialising filter methods

if isfield(EKFoptions,'filter_method')
    obj.filter_method = EKFoptions.filter_method;
else
    warning("filter method not defined, defaulting to sym");
end

switch obj.filter_method
    case 'tobi'
        %Compute initial decomposition for Thornton-Bierman
        [obj.U_BT,obj.D_BT] = UD_decomp(obj,obj.Pnow);

    case 'svd'
        %Initial SVD for SVD based filtering
        [obj.U_BT,obj.D_BT,~] = svd(obj.Pnow);
        obj.D_BT = sqrt(obj.D_BT);


    otherwise

end

%Initialising Schmidt Kalman

if isfield(EKFoptions,'consider_states')
    obj.c_states = EKFoptions.consider_states;
else
    fprintf("No consider states defined")
end
```

Addition to method KFpredict() in EKF.m

```
%different implementations of the filter
switch obj.filter_method
    case 'sym'
        %forcing symmetry
        obj.Pnow = (tmp_P + tmp_P.')/2;

    case 'normal'

        obj.Pnow = tmp_P;

    case 'tobi'
        %Bierman-Thornton factorization

        [obj.U_BT,obj.D_BT] = thornton(obj,G);

        %reconversion for common comparison
        obj.Pnow = obj.U_BT * obj.D_BT * obj.U_BT';

    case 'svd'
        %SVD-SRKF
        PA = [obj.D_BT*obj.U_BT'*obj.Phi_km1'; chol(obj.MDL.Wmat)'*G'*
      sqrt(dt)];
        [~,D,V] = svd(PA,'econ');
        obj.D_BT = D(1:obj.numStates,1:obj.numStates);
        obj.U_BT = V;
        obj.Pnow = obj.U_BT * obj.D_BT^2 *obj.U_BT';

    otherwise
        obj.Pnow = (tmp_P + tmp_P.')/2;
end
```

Addition to method KFupdate() in EKF.m

```
%consider states that can be set externally
if obj.t<t0+1
    gamma = 0.2;%weighting function
    dX_k(obj.c_states) = gamma*dX_k(obj.c_states);
    PT_k(obj.c_states,obj.c_states) =gamma^2*PT_k(obj.c_states,obj.
      c_states)+(1-gamma^2)* obj.Pnow(obj.c_states,obj.c_states);
end

%consider position during outage states
if  obj.gnssOutage
    c_states = [1:3];
    gamma = 0; %weighting function
    dX_k(c_states) = gamma*dX_k(c_states);
    PT_k(c_states,c_states) =gamma^2*PT_k(c_states,c_states)+(1-gamma^2)
      * obj.Pnow(c_states,c_states);
end

%variants of filtering methods
switch obj.filter_method
    case 'sym'
        obj.Pnow = (PT_k + PT_k.')/2;
        obj.Xnow = obj.Xnow + dX_k;

    case 'normal'
        obj.Pnow = PT_k;
        obj.Xnow = obj.Xnow + dX_k;

    case 'tobi'
        %Biermann update
        for j = 1:length(dZk)
            dz = dZk(j);
            H = H_k(j,:);
            R = R_k(j,j); %needs to be diagonal
            dx =bierman(obj,H,dz,R);
            obj.Xnow = obj.Xnow +dx;
            dZk = snsr.data_out(2:end) - obj.h_func{ii}(obj.Xnow, tmpU,
     obj.MDL.Param, obj.tVect(obj.k));
            H_k = obj.H_func{ii}(obj.Xnow, tmpU, obj.MDL.Param, obj.
     tVect(obj.k));
        end

        obj.Pnow = obj.U_BT * obj.D_BT * obj.U_BT';

    case 'svd'
        %SVD update
        l_k = inv(chol(R_k));
        PA = [l_k * H_k * obj.U_BT; inv(obj.D_BT)];
        [~, D,V] = svd(PA,'econ');
        obj.U_BT = obj.U_BT * V;
        obj.D_BT = inv(D(1:obj.numStates,1:obj.numStates));
        K_k = obj.U_BT * obj.D_BT^2 * obj.U_BT' * H_k'*(l_k*l_k');
        dX_k = K_k * dZk;
        obj.Xnow = obj.Xnow + dX_k;
        obj.Pnow = obj.U_BT * obj.D_BT^2 *obj.U_BT';

    case 'serial'
        %Serial update
        X = obj.Xnow;
        P = obj.Pnow;
        for j = 1:length(snsr.data_out(2:end))
            dz = dZk(j);
            H = H_k(j,:);
            R = R_k(j,j);

            K_bar = P*H'/(H*P*H'+R);
            X = X+ K_bar * dz;
            P = P-K_bar*H*P;
            dZk = snsr.data_out(2:end) - obj.h_func{ii}(X, tmpU, obj.MDL
     .Param, obj.tVect(obj.k));
            H_k = obj.H_func{ii}(obj.Xnow, tmpU, obj.MDL.Param, obj.
     tVect(obj.k));
        end

        obj.Xnow = X;
        obj.Pnow = (P + P')/2;


    otherwise
        obj.Pnow = (PT_k + PT_k.')/2;
        obj.Xnow = obj.Xnow + dX_k;

end
```

UD decomposition for Bierman-Thornton factorization, adapted from [17]

```
function [U,D] = UD_decomp(obj,mat)
    %UD decomposition of mat
    %mat = UDU'
    UD_TOL = 1e-18;
        noerror = '';
    [n m] = size(mat);
    if (n ~= m)
        error('Input matrix must be square');
    end
    U = zeros(n);
    D = zeros(n);
    for j = n:-1:1
        for i = j:-1:1
            sum = mat(i,j);
            for k = (j+1):n
                sum = sum - U(i,k) * D(k,k) * U(j,k);
            end
            if (i == j)
                if (sum <= UD_TOL)
                    if ~strcmpi(noerror,'noerror')
                        warning('Input matrix is not positive definite')
      ;
                    end
                    D(j,j) = 1;
                    U(j,j) = 0;
                else
                    D(j,j) = sum;
                    U(j,j) = 1;
                end
            else
                U(i,j) = sum / D(j,j);
            end
        end
    end
end
```

Bierman update

```
function [dx]=bierman(obj,H,dz, R)
    %Biermann update
    a = obj.U_BT' * H';
    b= obj.D_BT * a;
    alpha = R;
    gamma = 1./alpha;
    for j = 1:length(obj.Xnow)
        beta = alpha;
        alpha = alpha + a(j)*b(j);
        lambda = -a(j) * gamma;
        gamma = 1/alpha;
        obj.D_BT(j,j) = beta * gamma * obj.D_BT(j,j);
        for i = 1:j-1
            beta = obj.U_BT(i,j);
            obj.U_BT(i,j) = beta + b(i) * lambda;
            b(i) = b(i) + b(j) * beta;
        end
    end
    dzs = gamma*dz;
    dx = dzs *b;
end
```

Thornton prediction

```
function [U,D] = thornton(obj,Gin)
    %Thornton method for prediction
    dt = obj.tVect(obj.k) - obj.tVect(obj.k-1);
    [n,r] = size(Gin);
    G=Gin;
    W = obj.MDL.Wmat*dt;
    U= eye(n);
    PhiU = obj.Phi_km1 * obj.U_BT;
    for i = n:-1:1
        sigma = 0;
        for j = 1:n
            sigma = sigma + PhiU(i,j)^2 * obj.D_BT(j,j);
            if (j<=r)
                sigma = sigma + G(i,j)^2 *W(j,j);
            end
        end
        D(i,i) = sigma;
        for j = 1:i-1
            sigma = 0;
            for k = 1:n
                sigma = sigma + PhiU(i,k)*obj.D_BT(k,k)*PhiU(j,k);
            end
            for k= 1:r
                sigma = sigma + G(i,k) *W(k,k)*G(j,k);
            end
            U(j,i) = sigma/D(i,i);
            if isnan(U(j,i))
                fprintf("Error Nan found")
            end
            for k = 1:n
                PhiU(j,k) = PhiU(j,k)-U(j,i)*PhiU(i,k);
            end
            for k= 1:r
                G(j,k) = G(j,k) - U(j,i)*G(i,k);
            end
        end
    end

end
```