Development of an Autopilot for a Small Fixed-Wing Unmanned Aircraft

Special Course

Mads Friis Bornebush
Kristian Sloth Lauszus
June 2016

Supervisors:
José M.G. Merayo
Roberto Galeazzi

DTU Space
National Space Institute

DTU Space
National Space Institute
Technical University of Denmark
Elektrovej, building 327
DK - 2800 Kgs. Lyngby
Tel (+45) 4525 9500
Fax (+45) 4525 9575
www.space.dtu.dk

Front cover image: https://upload.wikimedia.org/wikipedia/commons/2/28/Aerosonde.jpg

Summary (English)

The goal of this project is to develop the navigation and guidance system for an autopilot for a small fixed-wing unmanned aircraft. The navigation system, which is implemented as a Kalman filter, will use the attitude and sensor measurements from accelerometer, GPS, airspeed sensor and barometer to estimate the position and velocity of the aircraft along with the wind speed. The guidance system will be implemented as a Dubins path and vector fields which takes a set of waypoints and an estimate of the position and outputs set-points for the low level controllers on the airplane in order to make it follow the desired path. The full system is tested in simulation and the results are analysed. The final navigation and guidance systems are implemented in C and C++.

Summary (Danish)

Formålet med dette projekt er at udvikle navigations- og guidancesystemet til en autopilot beregnet til et mindre ubemandet fastvingefly. Navigationssystemet, som er implementeret som et Kalmanfilter, bruger orienteringen og sensorinputs fra accelerometer, GPS, pitotrør og barometer til at estimere fartøjets position og hastighed samt vindhastigheden. Guidancesystemet vil blive implementeret som Dubinskurver og vektorfelter. Det får et sæt navigationspunkter samt den estimerede position som inputs og giver referencerne til fartøjets grundlæggende regulatorer for at få det til at følge den planlagte rute. Det samlede system er simuleret og resultaterne af dette er analyseret. Det endelige navigations- og guidancesystem er implementeret i C og C++.

Preface

This report was composed by Kristian Sloth Lauszus (s123808) and Mads Friis Bornebusch (s123627) at DTU Space at the Technical University of Denmark (DTU). The project was supervised by José M.G. Merayo from DTU Space and Roberto Galeazzi from DTU Electrical Engineering. The duration of the project was from 1st of February 2016 to 24th of June 2016. The workload of the project was 10 ECTS-points per person. The development was carried out at the offices of Danish Aviation Systems. The detailed plan of the project with milestones and weekly tasks is shown in Appendix A.

In the first part of the report the theory for the navigation system will be presented and implemented. The next part deals with the theory and implementation of a guidance system that allows a fixed-wing aircraft to fly between different waypoints.

In the final part the two systems will be combined and simulated. The simulation results will then be presented, future work proposed and a conclusion to the project outlined.

Kristian Sloth Lauszus

Mads Friis Bornebusch

Contents

Nomenclature

$\hat{\mathbf{x}}(k)$	State estimate
$\mathbf{a}$	Acceleration vector
$\mathbf{A}$, $\mathbf{A}(t)$	System matrix in continuous time
$\mathbf{B}$, $\mathbf{B}(t)$	Input matrix in continuous time
$\mathbf{b}$, b	Bias state, subscript denotes where it is present
$\mathbf{B}_v$, $\mathbf{B}_v(t)$	Noise input matrix
$\mathbf{C}$, $\mathbf{C}(t)$, $\mathbf{C}(k)$	Output matrix
$\mathbf{D}$, $\mathbf{D}(t)$	Direct term matrix
$\mathbf{e}$, e	Noise process, subscript denotes where it is present
$\mathbf{F}$, $\mathbf{F}(k)$	System matrix in discrete time
$\mathbf{G}$, $\mathbf{G}(k)$	Input matrix in discrete time
$\mathbf{G}_v$, $\mathbf{G}_v(k)$	Noise input matrix in discrete time
$\mathbf{I}$	Identity matrix
$\mathbf{i}(k)$	Innovation process
$\mathbf{L}(k)$	Kalman gain matrix
$\mathbf{P}$	Position vector
$\mathbf{Qd}$	Process noise covariance matrix in discrete time

$\mathbf{Q}(k)$	State covariance matrix
$\mathbf{Rd}$	Measurement noise covariance matrix in discrete time
$\mathbf{R}$	Rotation matrix
$\mathbf{u}$	Input vector
$\mathbf{V}$	Velocity vector
$\mathbf{V}_1$	Process noise covariance matrix in continuous time
$\mathbf{v}_1$	Process noise vector
$\mathbf{V}_2$	Measurement noise covariance matrix in continuous time
$\mathbf{v}_2$	Measurement noise vector
$\mathbf{V}_R$	Right eigen vector column matrix
$\mathbf{W}$	Wind velocity vector
$\mathbf{x}$	State vector
$\mathbf{y}$	Output vector
dir	Turning direction of the Dubins path i.e. 1 when turning CCW, -1 when turning CW and 0 when going straight
μ	Mean
ω	Turning rate
ϕ, θ, ψ	Roll, pitch and yaw of aircraft. Also called bank, elevation and heading
ϕ_{max}	Maximum allowed roll angle of the UAV
ψ_d	Desired heading angle
ψ_e	Entry heading angle for the line vector field
ρ	Turning radius
σ	Standard deviation
σ_a^2	Variance of measurement noise noise for the accelerometer
σ_P^2	Variance of the process noise for the position
σ_V^2	Variance of measurement noise for the GPS velocity
σ_W^2	Variance of the process noise for the wind

σ^2_{air}	Variance of measurement noise for the airspeed sensor
σ^2_{baro}	Variance of measurement noise for the barometer
σ^2_{bias}	Variance of the process noise for the biases
σ^2_{GPS}	Variance of measurement noise for the GPS position
τ	Transition region boundary distance for the line vector field
θ_{max}	Maximum allowed pitch angle of the UAV
φ, λ	Latitude and longitude
g_0	The gravitational constant
k_1	Convergence gain for the circular vector field
k_2	Transition gain for the line vector field
L_{air}	Length of the Dubins path in the x,y,z-plane
L_{car}	Length of the Dubins path in the x,y-plane
R	Earth radius
s	Normalized distance along a straight line segment on a Dubins path
t, p, q	Length of first, second and third segment for a Dubins path
T_s	Sample time
T_{GPS}	GPS sample time
T_{meas}	Measurement sample time
V_{air_d}	Desired airspeed
V_{air}	Airspeed
w_1, w_2	First and last waypoint in a Dubins path
w_{min_dist}	Minimum threshold distance between UAV and a waypoint
w_{sub1}, w_{sub2}	The two sub-waypoints between the two given waypoints w_1 and w_2 in a Dubins path
z_d	Desired z-position

Chapter 1

Introduction

The use of unmanned aerial vehicles (UAVs) are becoming more and more widespread. UAVs are found in all forms from hobby drones such as multirotors or model airplanes of various sizes to full size airplanes used for surveillance and warfare. The low cost of sensors is opening up the market for small autonomous drones used for commercial purposes such as aerial photography, videography, farming, goods delivery, meteorological surveying, traffic monitoring or search and rescue operations (Geographic, 2013). The demand for cheap UAVs is therefore high. The type of UAV determines the properties of it. When considering small UAVs there are two main types. There is the fixed-wing UAVs and multirotors, each of them having their own advantages and disadvantages. The main advantages of fixed-wing UAVs is the fact that they have long endurance and generally higher payload capability while the advantages of multirotors are that they can land and takeoff vertically and hover.

For a UAV to fulfil its purpose it is crucial that it can be controlled and programmed to fly a specific route autonomously. This requires sensors and software on-board the UAV that can estimate the orientation, position and velocity of it. These estimates can be used by other software to control the actuators on the UAV making it follow the desired path.

1.1 Project aim

The purpose of this project is to create an autopilot for a small fixed-wing UAV. The project was carried out for the company Danish Aviation Systems[1] who are developing a flight controller for a range of different UAVs. A flight controller has to do a multitude of tasks in order to control a UAV. It has to take sensor inputs, communicate with a ground station, keep track of the mission and pilot the aircraft. The system which is responsible for the latter is the autopilot and it consists of a number of subsystems. A block diagram of a simple autopilot for a UAV is shown on Figure 1.1.

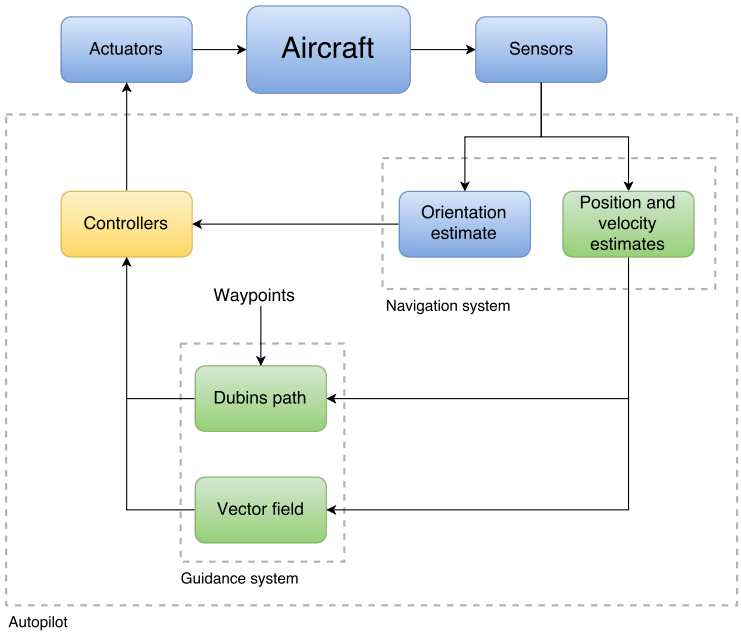


Figure 1.1: Block diagram of an autopilot on a UAV. This project will focus on the implementation of the subsystems shown in green. The controllers which are shown in yellow were implemented in simulation to be able to test the full system

The subsystems shown in green, the navigation system and the guidance system, is what this project will focus on, while the controllers, shown in yellow, will be implemented in order to simulate the full system. The navigation system is the part that takes the measurements from the sensors and estimates the orientation, velocity and position of the aircraft. This project will be limited

[1] `http://www.danishaviationsystems.dk`

to the part of the navigation system that estimates the position and velocity. The guidance system calculates the desired path from a set of waypoints and determines the low level controller set-points given the position estimate. The guidance system will be limited to not consider autonomous takeoff and landing.

As the guidance and navigation systems are supposed to be used as part the full flight controller, the implementation of the systems will be the main focus of the project. To keep development time down, all testing will be done in Matlab and only the final algorithms will be implemented in C and C++ to be compatible with the full flight controller.

1.2 Requirements

The requirements of the system is based on the following being provided by Danish Aviation Systems:

- Sensor measurements in SI units
- Orientation estimate as Euler angles or quaternions
- Controllers
- Airframe and flight controller hardware

The airframe which is expected to be provided is a flying wing type UAV which flies slower than $50\,\mathrm{km/h}$. The product of this project should be the following two systems:

- Navigation system: Given sensor measurements and a orientation estimate it should estimate the position and velocity
- Guidance system: Given waypoints and position estimate it should calculate the desired path and give set-points for the controllers

Since the required airframe and flight controller hardware was not provided by Danish Aviation Systems in due time, we decided to focus on implementing the navigation and guidance system in a simulated environment. The implemented systems were then converted to C and C++ code, thus allowing the navigation and guidance system to be implemented on an embedded platform in the future.

The project agreement with Danish Aviation Systems which contains the initial requirements is shown in Appendix B.

The source code for the project will not be included in the written report, but can be requested for research purposes by contacting the authors.

Chapter 2

Navigation system theory

This chapter will briefly explain the theory necessary for designing the navigation system for a fixed-wing UAV. The reference frames used in the navigation system design will be explained along with the necessary transformations between reference frames. This is necessary since the different sensor measurements are in different reference frames. Dynamical systems will be explained, as will how to get the system description on linear state space form in both continuous and discrete time. This will be the basis for the explanation of the Kalman filter; an optimal observer for state estimation of a dynamic system which is a critical part in the UAV navigation system.

2.1 Reference frames

Two different reference frames will be used in this project. One of them is the body frame and the other is the inertial frame. These will be described in the following sections.

2.1.1 Body frame

The body frame is the frame of reference which is aligned with the UAV body. It is a rectangular coordinate system where the origo is at the center of gravity of the UAV with the x-axis going through the nose, the z-axis going through the bottom and the y-axis completing a right handed coordinate system. The body frame coordinate system is shown in Figure 2.1. This coordinate system will be referred to as body frame and coordinates in this frame of reference will have x, y and z subscripts.

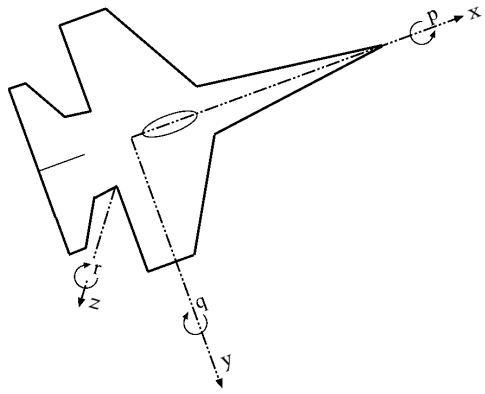


Figure 2.1: Body frame coordinate system. Figure is from Farrell (2008, page 26)

When considering a specific implementation of a system, two other reference frames might need to be considered. One of those is the platform frame at which the sensors are attached. Introducing this reference frame can compensate for misalignments in the sensor placement with respect to the body frame. However, the transformation from this frame to body frame can be determined in the design stage and the sensor measurements transformed accordingly, or if the difference between this and the body frame is small, it can be ignored. Another frame of reference is the instrument frame. Introducing this, compensates for misalignement between the instrument axes. If high quality sensor data is required from sensors of low quality this frame of reference will definitely have to be considered. For the purpose of this project it will be ignored.

2.1.2 Inertial frame

An inertial frame is a frame of reference where Newtons laws of motion applies. This means that an inertial frame can be in constant motion but it does not accelerate. The inertial frame in this project will be the local geodetic frame, which is a rectangular coordinate system with origin in a point on the surface of the Earth. The x-axis of the coordinate system points in the direction of the north pole, the z-axis points down and the y-axis points east and completes the right handed coordinate system. This coordinate system is shown in Figure 2.2. Coordinates in this frame will be denoted with N, E and D subscripts. The advantage of this coordinate system is that the coordinates are easy to understand intuitively as the origo is on the surface of the Earth. When a map projection is used to project points on the surface of Earth into this coordiante system, the disadvantage of this system is a high distortion when the distance to the origo is large.

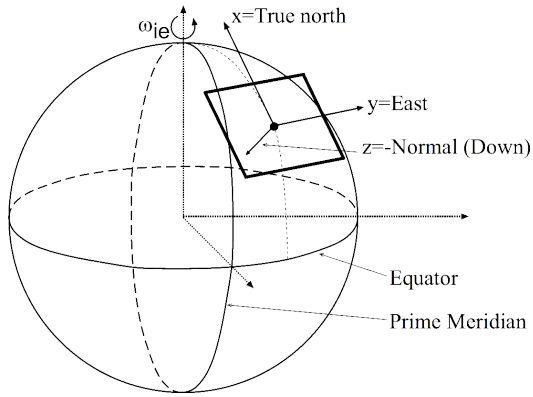


Figure 2.2: Local geodetic coordinate system in relation to the ECEF coordinate system. Figure is from Farrell (2008, page 25)

To calculate the NED-coordinates in the local geodetic frame from latitude, longitude and altitude GPS coordinates, a stereographic projection is used. The stereographic projection is a projection of a sphere on a tangent plane. An example of this projection is shown in Figure 2.3. The figure shows the projection of a point P on the sphere into P' in the plane. The origo of the stereographic projection S is the intersection of the plane and the sphere. Points are projected on a straight line from the point on the sphere opposite the origo into the plane.

The formula relating the latitude and longitude coordinates to the x and y

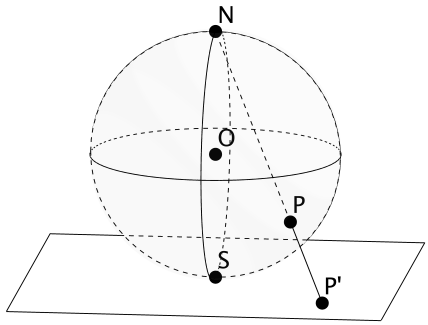


Figure 2.3: Stereographic projection. The origo S is the intersection of the plane and sphere. The point P on the sphere is projected into P' in the plane

coordinates in the projection plane is given in Snyder (1987, page 157) as:

$$\begin{aligned} x &= Rk\left(\cos(\varphi_0)\sin(\varphi) - \sin(\varphi_0)\cos(\varphi)\cos(\lambda - \lambda_0)\right) \\ y &= Rk\cos(\varphi)\sin(\lambda - \lambda_0) \\ k &= \frac{2k_0}{1 + \sin(\varphi_0)\sin(\varphi) + \cos(\varphi_0)\cos(\varphi)\cos(\lambda - \lambda_0)} \end{aligned} \tag{2.1}$$

where R is the Earth radius, φ is the latitude, λ is the longitude and $k_0 = 1$ is a scaling factor. The zero subscript on the latitude and longitude denotes the coordinates to the intersection between the plane and the sphere. When the projection is performed according to the formulas above, the x axis is pointing north and the y axis is pointing east. The stereographic projection is conformal which means that that angles between intersecting curves are unchanged (Snyder, 1987, page 154). However it is clear that there is distortion in distance which increases as a function of the distance to origo of the projection. A plot of this is shown in Figure 2.4. This figure shows that for distances below 100 km from the origin the error is less than 5 m. However, for large distances from the origin the error will be substantial. A common aproach to avoid this problem for UAVs with large operating ranges is to move the local geodetic plane along with the airplane (Farrell, 2008, page 25).

2.1.3 Rotations

Rotations in this project will be represented as Euler angles. Euler angles are rotations around the principal axes in a specific sequence. Rotation around the x axis is roll, ϕ; around the y axis is pitch, θ and around the z axis is yaw,

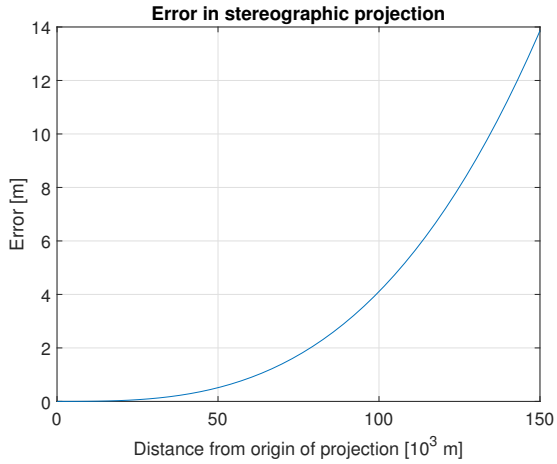


Figure 2.4: Distance error in the stereographic projection compared to great circle distance

ψ. The rotations will be represented by a rotation matrices. Rotation with an angle around an axis is realised with the following rotation matrices (Goldstein et al., 2001, page 134-144) (Diebel, 2006):

$$\mathbf{R_x}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \tag{2.2}$$

$$\mathbf{R_y}(\theta) = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.3}$$

$$\mathbf{R_z}(\psi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The subscript on $\mathbf{R}$ denotes the axis and the argument is the angle. A rotation sequence is a series of rotations and the rotation sequence operator is found as a matrix product of the above matrices. The rotation sequence used here is XYZ rotations where the rotation is around the x axis first, then around the y axis and in the end around the z axis.

The rotation matrix for rotating a vector from the body to initial frame is found as:

$$\mathbf{R} = [\mathbf{R_x}(\phi)\mathbf{R_y}(\theta)\mathbf{R_z}(\psi)]^T \tag{2.5}$$

which gives:

$$\mathbf{R} = \begin{bmatrix} \cos(\psi)\cos(\theta) & \cos(\psi)\sin(\phi)\sin(\theta) - \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\psi) + \cos(\phi)\cos(\psi)\sin(\theta) \\ \cos(\theta)\sin(\psi) & \cos(\phi)\cos(\psi) + \sin(\phi)\sin(\psi)\sin(\theta) & \cos(\phi)\sin(\psi)\sin(\theta) - \cos(\psi)\sin(\phi) \\ -\sin(\theta) & \cos(\theta)\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix} \tag{2.6}$$

The inverse (or transpose) of this rotation matrix will rotate the vector from inertial frame into body frame.

2.2 Dynamic systems

Dynamic systems are systems whose beaviour, states and output change dynamically as a function of time. Mathematically these systems are usually modelled as coupled ordinary differential equations. Dynamic systems can be deterministic or stochastic if they are subject to noise. In the following both types will be explained briefly.

2.2.1 Deterministic dynamic systems

The mathematical description of a dynamic system can be written as (Hendricks et al., 2008, page 28):

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{2.7}$$

where $\mathbf{x}(t)$ is the system state vector, $\mathbf{u}(t)$ is the input to the system and $\mathbf{f}$ is the system function. The function $\mathbf{f}$ of the state can be a time variant linear or nonlinear function. The output of such a system is (Hendricks et al., 2008, page 29):

$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), t) \tag{2.8}$$

2.2.2 Stochastic dynamic systems

All realistic systems will be subject to noise. Noise is stochastic or random processes and they will in this project be assumed normally distributed and white. They can therefore be characterized by a mean μ and standard deviation σ (Farrell, 2008, page 112) (Hendricks et al., 2008, page 360). In the analysis of the noise the notions of covariance, correlation and autocorrelation will be used. A more thorough explanation of the concepts are given in Appendix C.

A system subject to noise can be written as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}_1(t), t) \tag{2.9}$$

where $\mathbf{v}_1(t)$ is the so called process noise. The output of the system is:

$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), t) + \mathbf{v}_2(t) \tag{2.10}$$

where $\mathbf{v}_2(t)$ is the measurement noise.

2.2.3 Linearization

The system is linearised by a first order Taylor expansion around a stationary point (Hendricks et al., 2008, page 30-31) (Farrell, 2008, page 72) (Gustafsson, 2012, page 318). The linear system matrices are thus found as the Jacobians:

$$\begin{aligned}
\mathbf{A}(t) &= \left.\frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{v}_1, t)}{\partial \mathbf{x}}\right|_{\mathbf{x}_0(t), \mathbf{u}_0(t)} \\
\mathbf{B}(t) &= \left.\frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{v}_1, t)}{\partial \mathbf{u}}\right|_{\mathbf{x}_0(t), \mathbf{u}_0(t)} \\
\mathbf{B}_v(t) &= \left.\frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{v}_1, t)}{\partial \mathbf{v}_1}\right|_{\mathbf{x}_0(t), \mathbf{u}_0(t)}
\end{aligned} \tag{2.11}$$

The linear output matrices are found in a similar way:

$$\begin{aligned}
\mathbf{C}(t) &= \left.\frac{\partial \mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{v}_2, t)}{\partial \mathbf{x}}\right|_{\mathbf{x}_0(t), \mathbf{u}_0(t)} \\
\mathbf{D}(t) &= \left.\frac{\partial \mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{v}_2, t)}{\partial \mathbf{u}}\right|_{\mathbf{x}_0(t), \mathbf{u}_0(t)}
\end{aligned} \tag{2.12}$$

The linear system can now be written as (Hendricks et al., 2008, page 13):

$$\begin{aligned}
\dot{\mathbf{x}}(t) &= \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) + \mathbf{B}_v(t)\mathbf{v}_1(t) \\
\mathbf{y}(t) &= \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t) + \mathbf{v}_2(t)
\end{aligned} \tag{2.13}$$

This system is linear and time variant as can be seen from the time dependence of the system matrices.

2.2.4 Discretization

A linear time invariant system can be discretized as(Hendricks et al., 2008, page 77) (Farrell, 2008, page 80):

$$\begin{aligned}
\mathbf{F} &= e^{\mathbf{A}T_s} \\
\mathbf{G} &= \int_0^{T_s} e^{\mathbf{A}t}\mathbf{B}dt
\end{aligned} \tag{2.14}$$

Where T_s is the sampling time. For a slowly time varying system where $\mathbf{A}$ can be considered constant within the sampling time interval $t_0 \leq t \leq t_0 + T_S$ the discrete system matrix can be found by the Taylor expansion of the matrix exponential (Farrell, 2008, page 144 and 241-242):

$$e^{\mathbf{A}T} = \mathbf{I} + \mathbf{A}T + \frac{1}{2}(\mathbf{A}T)^2 + \frac{1}{3!}(\mathbf{A}T)^3 + \dots \tag{2.15}$$

This approach of first linearizing the system and then discretizing it is also described by Gustafsson (2012, page 318). Here, another method is also shown where the nonlinear integral equation relating the current state to the one, which is one sample time into the future is solved. This equation is:

$$\mathbf{x}(t + T_s) = \mathbf{x}(t) + \int_t^{t+T_s} \mathbf{f}(\mathbf{x}(\tau), \mathbf{u}(\tau))d\tau \tag{2.16}$$

This will directly give the discretization of the nonlinear system which is then linearized. This method is claimed by (Gustafsson, 2012) to give more accurate results than the method where the system is linearized first and then discretized. However, if it can be assumed that $\mathbf{A}$ and $\mathbf{u}$ are constant between samples, the method in (2.14) with the matrix exponential calculated as the Taylor series in (2.15) is preferable due to the simplicity of it. If the system linearization and discretization can be solved analytically the end result is a set of time varying discrete time matrices as in Farrell (2008, page 242).

2.2.5 Discrete time system

A discrete time variant system is given by:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{F}(k)\mathbf{x}(k) + \mathbf{G}(k)\mathbf{u}(k) + \mathbf{G}_v(k)\mathbf{v}_1(k) \\ \mathbf{y}(k) &= \mathbf{C}(k)\mathbf{x}(k) + \mathbf{D}(k)\mathbf{u}(k) + \mathbf{v}_2(k) \end{aligned} \tag{2.17}$$

where $\mathbf{F}(k)$ is the system matrix, $\mathbf{G}(k)$ is the input matrix, $\mathbf{G}_v(k)$ is the noise input matrix, $\mathbf{v}_1$ is the process noise vector, $\mathbf{C}(k)$ is the state output matrix, $\mathbf{D}(k)$ is the direct output matrix and $\mathbf{v}_2(k)$ is the measurement noise. For this system it is assumed that all inputs and matrices can be considered constant between samples. The time variance comes from the matrices being found with analytic linearization and discretization as described previously. Depending on the system that is linearised a more precise way to state the matrices could be:

$$\mathbf{F}(\mathbf{x}(k), \mathbf{u}(k), k) \tag{2.18}$$

and similar for the other matrices. This shows that inserting the values at which the linearization is carried out will give a constant matrix.

The noise in the system is given by:

$$\begin{aligned} \mathbf{v}_1(k) &\in N(0, \mathbf{V}_1) \\ \mathbf{v}_2(k) &\in N(0, \mathbf{V}_2) \end{aligned} \tag{2.19}$$

where the discretization of $\mathbf{V}_1$ and $\mathbf{V}_2$ should be done as:

$$\begin{aligned} \mathbf{Qd}(k) &\approx \mathbf{B}_v(k)\mathbf{V}_1\mathbf{B}_v^T(k)T_s \\ \mathbf{Rd} &\approx \frac{\mathbf{V}_2}{T_s} \end{aligned} \tag{2.20}$$

where T_s is the sample time (Farrell, 2008, page 141-142) (Hendricks et al., 2008, 414-415). For the purpose of this project, the noise matrices will be constant as the noise is assumed to be stationary. It can be seen that the matrix $\mathbf{Qd}(k)$ will be time variant due to the time variance of $\mathbf{G}_v(k)$. It is common to have the outputs of a system measurd by sensors at different sampling rates. If this is the case, the matrix $\mathbf{Rd}$ can not be found by dividing by a single sample time as shown above. If $\mathbf{V}_2$ is diagonal, $\mathbf{Rd}$ is instead found as:

$$\mathbf{Rd} \approx \mathbf{V}_2 \begin{bmatrix} T_{s1}^{-1} & & \\ & \ddots & \\ & & T_{sm}^{-1} \end{bmatrix} = \mathbf{V}_2\mathbf{T}^{-1} \tag{2.21}$$

where all empty space is zeros, m is the number of measurements and $T_{s1}, \cdots, T_{sm}$ are the sample times for the measurements and $\mathbf{T} = \text{diag}(T_{s1}, \cdots, T_{sm})$.

2.3 The Kalman filter

The Kalman filter is an observer for a dynamic system. We will consider a discrete time system like the one in (2.17) with the only exception that there is no direct term from input to output which means that $\mathbf{D}(k) = 0 \ \forall \ k$. This is done to simplify the equations as the system is not expected to have a direct term. A Kalman filter for a system with a direct term is presented in Gustafsson (2012, page 154). A dynamical system with no direct term from input to output can be written as:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{F}(k)\mathbf{x}(k) + \mathbf{G}(k)\mathbf{u}(k) + \mathbf{G}_v(k)\mathbf{v}_1(k) \\ \mathbf{y}(k) &= \mathbf{C}(k)\mathbf{x}(k) + \mathbf{v}_2(k) \end{aligned} \tag{2.22}$$

An observer uses a model of the system and measurements of the system output to find a state estimate.

The open form Kalman filter (also called ordinary Kalman filter) consists of two steps; the time update of the state estimate and the measurement update of the state estimate. The time update is where the system model is used to predict the system states. This prediction is then corrected with the output measured from the system in the measurement update. The state estimate is accompanied by a state covariance matrix which also has a time update and measurement update step. The correction of the state in the measurement update is weighted by a gain. This gain is called the Kalman gain and it is calculated for each time instant. This calculation is what sets the Kalman filter apart from other observers as the Kalman gain is shown to give the minimum variance and least square error estimates of the states (Hendricks et al., 2008, page 439).

In the following the Kalman filter equations will be presented mainly using the notation of Hendricks et al. (2008). Denoting the state estimate $\hat{\mathbf{x}}$, the time update can be written as:

$$\hat{\mathbf{x}}(k)^- = \mathbf{F}(k-1)\hat{\mathbf{x}}(k-1) + \mathbf{G}(k-1)\mathbf{u}(k-1) \tag{2.23}$$

This equation can be recognized as a prediction of the state using the system model. The minus superscript on $\hat{\mathbf{x}}(k)$ denotes that data up to time $t-1$ is used for this estimate. The covariance time update can be written as:

$$\mathbf{Q}(k)^- = \mathbf{F}(k-1)\mathbf{Q}(k-1)\mathbf{F}^T(k-1) + \mathbf{Q}\mathbf{d}(k-1) \tag{2.24}$$

Again the minus superscript denotes that data up to time $t-1$ has been used to calculate the state covariance matrix. After the time update step has been performed, the Kalman gain can be calculated:

$$\mathbf{L}(k) = \mathbf{Q}(k)^-\mathbf{C}^T(k)\left[\mathbf{C}(k)\mathbf{Q}(k)^-\mathbf{C}^T(k) + \mathbf{R}\mathbf{d}\right]^{-1} \tag{2.25}$$

This gain is now used to adjust the state estimate using the measurements in the measurement update:

$$\hat{\mathbf{x}}(k) = \hat{\mathbf{x}}(k)^- + \mathbf{L}(k)\left[\mathbf{y}(k) - \mathbf{C}(k)\hat{\mathbf{x}}(k)^-\right] \tag{2.26}$$

The Kalman gain is also used to update the state covariance matrix in the measurement update:

$$\mathbf{Q}(k) = \left[\mathbf{I} - \mathbf{L}(k)\mathbf{C}(k)\right]\mathbf{Q}(k)^- \tag{2.27}$$

The result of running the Kalman filter is at each time instant an estimate of the state and a covariance matrix for the state estimate which shows how certain the estimate is.

2.3.1 Assumptions

There are a number of necessary assumptions in the Kalman filter. For a Kalman filter to exist for a system, the system needs to be detectable, which means that all unobservable states are asymptotically stable. For all states to be reconstructed by the Kalman filter at all times, the system needs to be fully observable (Hendricks et al., 2008, page 432). Another assumption is that the process noise $\mathbf{v}_1$ and the measurement noise $\mathbf{v}_2$ are uncorrelated. This means that $E\{\mathbf{v}_1(j)\mathbf{v}_2^T(k)\} = 0$ for all j,k. It is also assumed that there is noise on all measurements (Hendricks et al., 2008, page 433).

2.3.2 Iterated Kalman filter

When implementing a Kalman filter, calculating the matrix inverse in (2.25) is usually a time consuming task. However, this can be avoided if the measurements are independent, that is if $\mathbf{Rd}$ is a diagonal matrix (Farrell, 2008, page 191) (Gustafsson, 2012, page 170). The measurement update can then be done sequentially where the matrix inversion is a division by a scalar. A Kalman filter implemented like this is referred to as an iterated Kalman filter. We define the following at time k:

$$\begin{aligned} \mathbf{Q}_1 &= \mathbf{Q}(k)^- \\ \hat{\mathbf{x}}_1 &= \hat{\mathbf{x}}(k)^- \\ \mathbf{C}(k) &= \begin{bmatrix} \mathbf{C}_1 \\ \vdots \\ \mathbf{C}_m \end{bmatrix} \end{aligned} \tag{2.28}$$

where m is the number of measurements and $\mathbf{C}_1, \cdots, \mathbf{C}_m$ are the rows in $\mathbf{C}$. The measurement update equations for this implementation is then, for $i = 1$ to m:

$$\begin{aligned} \mathbf{L}_i &= \frac{\mathbf{Q}_i \mathbf{C}_i^T}{\mathbf{C}_i \mathbf{Q}_i \mathbf{C}_i^T + \mathbf{Rd}_{ii}} \\ \hat{\mathbf{x}}_{i+1} &= \hat{\mathbf{x}}_i + \mathbf{L}_i \left[\mathbf{y}_i - \mathbf{C}_i \hat{\mathbf{x}}_i\right] \\ \mathbf{Q}_{i+1} &= \left[\mathbf{I} - \mathbf{L}_i \mathbf{C}_i\right] \mathbf{Q}_i \end{aligned} \tag{2.29}$$

where $\mathbf{Rd}_{ii}$ is the diagonal elements in the matrix $\mathbf{Rd}$. At the end of the iterations we have:

$$\begin{aligned} \hat{\mathbf{x}}(k) &= \hat{\mathbf{x}}_{m+1} \\ \mathbf{Q}(k) &= \mathbf{Q}_{m+1} \end{aligned} \tag{2.30}$$

which is equal to what was found in (2.26) and (2.27).

2.3.3 Handling different measurement rates

If the measurements to the Kalman filter arrives at different rates, $\mathbf{C}$ will be time varying, as to reflect at a given time k the measurements that are available (Poulsen, 2007, page 210). The matrix $\mathbf{C}$ is the matrix for a time where all measurements are available. For a time k where some measurements are unavailable, the corresponding rows in the matrix $\mathbf{C}(k)$ will be zero. In this way, only the measurements that are available will provide information to the Kalman filter at a given time.

2.3.4 Kalman filter performance

When implementing a Kalman filter there are several issues that has to be adressed. Some of them are of the numerical nature and some of them are of the analytical or theoretical nature. The numerical issues will not be adressed in this project. The issues which will be adresed is the observability of the system and the whiteness of the innovation process. These issues are important as they are signs of errors or omissions in the design of the Kalman filter.

2.3.4.1 Observability analysis

One of the assumptions stated above for a Kalman filter to exist is that all states are observable or at least detectable. For a system to be observable, the observability Gramian has to be regular (Hendricks et al., 2008, page 148). The observability Gramian, $\mathbf{W}_0(k_0, k_f)$ can be calculated as:

$$\mathbf{W}_0(k_0, k_f) = \sum_{i=k_0}^{k_f-1} \Phi^T(i, k_0)\mathbf{C}^T(k)\mathbf{C}(k)\Phi(i, k_0) \tag{2.31}$$

where Φ is the state transistion matrix defined as (Hendricks et al., 2008, page 82):

$$\Phi(l, m) = \mathbf{F}(l-1)\mathbf{F}(l-2)\dots\mathbf{F}(m) \tag{2.32}$$

where $\Phi(l, l) = \mathbf{I}$ and $\Phi(l+1, l) = \mathbf{F}(l)$. Due to the assumption that the state and inputs are constant between samples, the Gramian takes the form of a time invariant system when the observability is considered at a certain time instant. If the observability test is done for every single sample, the test can be carried out as for a time invariant system. In this case there are several methods to test the observability. One of them is calculating the observability matrix, which for

an observable system must have full rank. The observability matrix is given by:

$$\mathbf{M}_O = \begin{bmatrix} \mathbf{C} \\ \mathbf{CF} \\ \mathbf{CF}^2 \\ \vdots \\ \mathbf{CF}^{n-1} \end{bmatrix} \tag{2.33}$$

where n is the rank of $\mathbf{F}$. The system is observable if and only if $rank(\mathbf{M}_O) = n$. However, this test is sensitive to the matrix being ill-conditioned (Paige, 1981). Another way to test the observability is by the right eigenvectors of the system matrix $\mathbf{F}$. The matrix containing the right eigenvectors of $\mathbf{F}$ as columns will be denoted $\mathbf{V}_R$. The system is then observable if and only if:

$$\mathbf{CV}_R \neq \mathbf{0} \tag{2.34}$$

That is is there are no zero columns in the product between $\mathbf{C}$ and the right eigenvector matrix $\mathbf{V}_R$.

The two previous equations only applies to linear time invariant systems. In this project they will be used on linear time variant systems where the time variance is given symbolically. This will lead to equations for each column in (2.34) that can be solved to find the cases in which the system is not observable.

2.3.4.2 Innovation process

The innovation process is defined as:

$$\mathbf{i}(k) = \mathbf{y}(k) - \mathbf{C}(k)\hat{\mathbf{x}}(k) \tag{2.35}$$

which is recognized as the part that adjusts the state estimate in (2.26). When a perfect model of the system is used, the innovation process will be white noise with the same intensity as the measurement noise sources (Hendricks et al., 2008, page 446). The innovation process is the difference between the measured output and the output predicted by the system model. If there is dynamics in the system which are note modelled this will be present in the innovations which means that they will not be white. Examining the whiteness of the residuals is therefore a way to assess if a Kalman filter is designed properly or not. If not, the system model in (2.22) has to be changed to take the neglected dynamics into account. The innovation process can also give information which can be used to adjust the noise matrices **Qd** and **Rd** in the Kalman filter but this approach will be left as possible future work (Hajiyev et al., 2015, page 67).

Chapter 3

Navigation system design

We want to design a navigation system which is able to estimate the states of a small fixed-wing UAV. We assume that the orientation of the aircraft or an estimate thereof is given. The states we want to estimate are the position and the speed of the aircraft along with the wind velocity. We also assume that we are given a range of noisy sensor measurements in SI units. The sensors which are assumed to be present is an accelerometer, a GPS outputting position and velocity, a barometer and an airspeed sensor.

A dynamic model of the airplane, relating the control inputs (through forces and torques) with the angular and linear acceleration, could be used in the design of the navigation system. The navigation system would then be based on both the kinetics and kinematics of the airplane. If designed properly, this could lead to a very precise estimate of the system states due to the detailed system model. However, such a model is not trivial to obtain and the model parameters would have to be found for each airframe individually (Hajiyev et al., 2015, page 9-23). Another approach is to just consider the kinematics of the system as in Farrell (2008). This is the approach which will be used in this project and the design methodology will be the ones from Farrell (2008, page 10 and 235-247) which can be summarized as:

- Derive kinematic model of the system

- Develop sensor models
- Augment system with sensor models
- Linearize and discretize system
- Design state estimator (Kalman filter)
- Analyse system performance

This chapter will go through the necessary steps in the design of the navigation system and the above steps will be explained in the following sections. The analysis of the system performance will be explained in Section 5.1 in the chapter concerned with the results of this project.

3.1 Kinematic model of the system

The model for the system is a simple kinematic model relating the acceleration with the velocity and position. The input to the system is the measured acceleration vector, $\mathbf{u} = \mathbf{a}$. The states are the velocity, position and wind velocity, $\mathbf{x} = \begin{bmatrix}\mathbf{P} & \mathbf{V} & \mathbf{W}\end{bmatrix}^T$. The continuous time system model is then:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{v}_1, t) = \begin{bmatrix}\dot{\mathbf{P}} \\ \dot{\mathbf{V}} \\ \dot{\mathbf{W}}\end{bmatrix} = \begin{bmatrix}\mathbf{V} \\ \mathbf{a} \\ \mathbf{0}\end{bmatrix} + \begin{bmatrix}\mathbf{e}_P \\ \mathbf{e}_a \\ \mathbf{e}_W\end{bmatrix} \tag{3.1}$$

It can be seen that the wind is simply modelled as a random walk process. All noise in the system is assumed to be white Gaussian noise. The system measurements are:

$$\mathbf{y} = \begin{bmatrix}\mathbf{P}_{GPS} \\ \mathbf{V}_{GPS} \\ P_{D,baro} \\ V_{air}\end{bmatrix} \tag{3.2}$$

The types of sensors providing the measurements are given in the subscripts. There are a position and ground velocity measurement from a GPS, a barometer measurement of the altitude and an airspeed measurement. In the following section the sensor models will be developed and the system will be augmented with the sensor models.

3.2 Sensor models

Sensors will provide both the input to the system (high rate sensor) and the outputs or measurements of the system (mix of high and low rate sensors). Each of the sensor models will be explained in the following. The sensor models are simplified theoretical models of sensors found in the literature. They are therefore subject to a number of assumptions which means that these models needs to be revisited and adjusted if the Kalman filter is to be implemented on a specific hardware platform. For such an implementation the sensor data from the implementation needs to be analysed and appropriate sensor models have to be derived in case the initial assumptions does not hold.

3.2.1 Accelerometer

An accelerometer will be used to provide a high rate measurement used as input to the system. The acceleration measurements comes from a sensor which is rigidly attached to the platform. This is known as a strapdown system and the sensor measurements are therefore in platform frame (or body frame if possible misalignment is ignored). As the velocity and position is in the inertial frame, we need to rotate the measured acceleration into the inertial frame. Recalling that $\mathbf{R}$ found in (2.6) is the rotation matrix from body to inertial frame and letting a leading subscript denote the reference frame, we can write:

$$ {}_I\mathbf{a} = \mathbf{R}_B\mathbf{a} \tag{3.3} $$

The acceleration in inertial frame has to be compensated for gravity which gives the following equation:

$$ {}_I\mathbf{a}_{comp} = \mathbf{R}_B\mathbf{a} - \mathbf{g}_0 \tag{3.4} $$

with $\mathbf{g}_0$ being the specific gravity which is $\mathbf{g}_0 = \begin{bmatrix} 0 & 0 & 9.81 \end{bmatrix}^T {}^m/_{s^2}$. Measurements from an accelerometer are never perfect and there are several errors that can be modelled. Examples of these errors are biases (adding 3 states), scale factors and misalignments (adding 9 states) and nonlinear effects (adding 18 states) (Farrell, 2008, page 408-410). In this project only the biases will be considered. When this system is to be implemented on a hardware platform, data from the accelerometer should be analysed in order to see if this simplification is justified or if more accelerometer errors should be modelled. Modelling the accelerometer biases and adding the measurement noise, the acceleration measurement, $\mathbf{u}_a$, is:

$$ \mathbf{u}_a = {}_B\mathbf{a} - \mathbf{b}_a + \mathbf{R}^T\mathbf{g}_0 + \mathbf{e}_a \tag{3.5} $$

where $\mathbf{a}$ is the true acceleration in body frame, $\mathbf{b}_a$ is the bias, $\mathbf{g}_0$ is the gravity vector and $\mathbf{e}_a \in N_{iid}(0, \sigma_a^2)$ is the measurement noise. This equation gives the acceleration measured by the accelerometer in body frame. If this is rotated into inertial frame we have:

$$\mathbf{R}\mathbf{u}_a = {}_I\mathbf{a} - \mathbf{R}\mathbf{b}_a + \mathbf{g}_0 + \mathbf{R}\mathbf{e}_a \tag{3.6}$$

The compensated acceleration in inertial frame can now be isolated:

$${}_I\mathbf{a} = \mathbf{R}\mathbf{u}_a + \mathbf{R}\mathbf{b}_a - \mathbf{g}_0 + \mathbf{R}\mathbf{e}_a \tag{3.7}$$

This is the linear acceleration that can be integrated to get the velocity and position.

3.2.2 GPS position and velocity

A GPS is used for measuring the position and velocity. Modelling the GPS position errors is not a simple task. The reason for this is that most GPS receivers filter the satellite measurements and produce an output which might be correlated in time and where the noise on the axes might be correlated (Farrell, 2008, page 421). To realistically model the GPS errors a large number of factors has to be taken into account (Farrell, 2008, page 280-302). However, a simpler model of the errors can also be used where only the clock bias is considered as in Hajiyev et al. (2015, page 44) or in Wada and Hashimoto (2004). Similar to this the GPS measurement will in this project be modelled as a true position and a bias. The model including the noise is:

$$\mathbf{y}_{GPS} = \mathbf{P} - \mathbf{b}_{GPS} + \mathbf{e}_{GPS} \tag{3.8}$$

where $\mathbf{P}$ is the true position, $\mathbf{b}_{GPS}$ is the bias and $\mathbf{e}_{GPS} \in N_{iid}(0, \sigma_{GPS}^2)$ is the measurement noise. The modelling of GPS velocity is equally complicated as GPS position (Gaglione, 2015) (Serrano et al., 2004). The model that will be used in this project is with an additive bias as for the GPS position. The measurement model for the GPS velocity including noise is:

$$\mathbf{y}_V = \mathbf{V} - \mathbf{b}_V + \mathbf{e}_V \tag{3.9}$$

where $\mathbf{V}$ is the true velocity, $\mathbf{b}_V$ is the bias and $\mathbf{e}_V \in N_{iid}(0, \sigma_V^2)$ is the measurement noise.

3.2.3 Barometer

A barometer measures the pressure which can be converted to an altitude measurement. A formula is given in Jan et al. (2008) which assumes knowledge of

the temperature gradient as a function of altitude:

$$h = \frac{T_0}{T_{grad}} \left[1 - \left(\frac{p}{p_0} \right)^{\left(\frac{T_{grad} R}{g_0} \right)} \right] \tag{3.10}$$

where h is barometric altitude, T_0 is sea level temperature, T_{grad} is the temperature gradient, p is the pressure at altitude h, p_0 is the pressure at sea level, R is the gas constant and g_0 is the gravity. This model may be necessary if the platform is to perform at high altitudes as well as low altitudes. In Magnusson (2013, page 38) this temperature gradient is modelled as an multiplicative constant which is estimated in the Kalman filter. In Bosch (2015) the altitude is calculated using the typical values for T_0/T_{grad} and $T_{grad}R/g_0$. In this project we will assume that the barometric altitude is calculated fairly accurately and the unmodelled factors will be modelled as an additive bias. The measurement model is then:

$$y_{baro} = P_D - b_{baro} + e_{baro} \tag{3.11}$$

where P_D is the true altitude, b_{baro} is the barometer bias and $e_{baro} \in N_{iid}(0, \sigma^2_{baro})$ is the measurement noise.

3.2.4 Airspeed sensor

An airspeed sensor is usually realized as a differential pressure sensor. The measurement is the pressure from a pitot tube pointing in the direction of travel and a static pressure. By applying Bernoulli's equation, these two pressures can be used to calculate the flow velocity (NASA, 2015) (Microbridge, 2009):

$$v = \sqrt{\frac{2(p_t - p_s)}{\rho}} \tag{3.12}$$

where v is the velocity, ρ is the air density, p_t is the total pressure measured from the pitot tube and p_s is the static pressure. In this project we will assume that the airspeed is already converted and is given in $^m/_s$. The measurement model for the airspeed sensor is:

$$y_{air} = \mathbf{N} \cdot \mathbf{R}^T (\mathbf{V} + \mathbf{W}) + e_{air} \tag{3.13}$$

where $\mathbf{V}$ is the ground speed, $\mathbf{W}$ is the wind velocity, $\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ and $e_{air} \in N_{iid}(0, \sigma^2_{air})$. The cause for the N-matrix is that the measurement is only in one dimension and that the velocity and wind velocity vectors that we are measuring are in three dimensions. The noise free airspeed for the UAV would be:

$$V_{air} = \mathbf{N} \cdot \mathbf{R}^T (\mathbf{V} + \mathbf{W}) \tag{3.14}$$

The airspeed in three dimensions in inertial frame is:

$$ {}_I\mathbf{V}_{air} = \mathbf{V} + \mathbf{W} \tag{3.15} $$

It can be seen from this that a wind coming from North is positive in the North-coordinate. The wind vector $\mathbf{W}$ therefore gives the direction of travel where there is headwind. The other sensors has so far been modelled with an additive bias, but this will not be the case for the airspeed sensor. When an airspeed bias state was included, the Kalman filter was not able to estimate the sensor biases for the other measurements correctly. This led to a wrong state estimate and a bad overall performance of the filter. To avoid this, the airspeed measurement is therefore modelled without a bias.

3.3 Augmented system

The system will now be augmented with the sensor models found above. This means that the bias states will be added to the state vector and the system equations will be updated. The new states of the system are:

$$ \mathbf{x} = \begin{bmatrix} \mathbf{P}^T & \mathbf{V}^T & \mathbf{W}^T & b_{baro} & \mathbf{b}_{GPS}^T & \mathbf{b}_a^T & \mathbf{b}_V^T \end{bmatrix}^T \tag{3.16} $$

where $\mathbf{P}$, $\mathbf{V}$, $\mathbf{W}$, $\mathbf{b}_{GPS}$, $\mathbf{b}_a$ and $\mathbf{b}_V$ are vectors given by:

$$ \begin{aligned} \mathbf{P} = \begin{bmatrix} P_N \\ P_E \\ P_D \end{bmatrix} \mathbf{V} = \begin{bmatrix} V_N \\ V_E \\ V_D \end{bmatrix} \mathbf{W} = \begin{bmatrix} W_N \\ W_E \\ W_D \end{bmatrix} \\ \mathbf{b}_{GPS} = \begin{bmatrix} b_{GPS_N} \\ b_{GPS_E} \\ b_{GPS_D} \end{bmatrix} \mathbf{b}_a = \begin{bmatrix} b_{a_X} \\ b_{a_Y} \\ b_{a_Z} \end{bmatrix} \mathbf{b}_V = \begin{bmatrix} b_{V_N} \\ b_{V_E} \\ b_{V_D} \end{bmatrix} \end{aligned} \tag{3.17} $$

We can see that the system has 19 states. The system function, $\mathbf{f}$, in vector form is now given by:

$$ \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}_1(t), t) = \begin{bmatrix} \mathbf{V} \\ \mathbf{R}\mathbf{b}_a \\ 0 \\ \vdots \\ 0 \\ \vdots \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{R}\mathbf{u}_a - \mathbf{g}_0 \\ 0 \\ \vdots \\ 0 \\ \vdots \end{bmatrix} + \begin{bmatrix} \mathbf{e}_P \\ \mathbf{R}\mathbf{e}_a \\ \mathbf{e}_W \\ \vdots \\ e_{bias} \\ \vdots \end{bmatrix} \tag{3.18} $$

We can see that the first vector is the part of $\mathbf{f}$ that comes from the states, the second vector is the part that comes from the input and the last vector is the

process noise. The terms in the process noise vector are given by:

$$\begin{aligned} \mathbf{e}_P &\in N_{iid}(0, \sigma_P^2) \\ \mathbf{e}_W &\in N_{iid}(0, \sigma_W^2) \\ e_{bias} &\in N_{iid}(0, \sigma_{bias}^2) \end{aligned} \tag{3.19}$$

The biases are modelled as integrated white noise which will give a random walk process. The input to the system is given by:

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_a \\ \mathbf{g}_0 \end{bmatrix} \tag{3.20}$$

The reason for g_0 being in the input is that the acceleration is corrected for gravity inside the Kalman filter. If a gravity compensated acceleration is used instead, this term would vanish from $\mathbf{f}$ and $\mathbf{u}$.

The system measurements are given by:

$$\mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), t) + \mathbf{v}_2(t) = \begin{bmatrix} \mathbf{P} - \mathbf{b}_{GPS} \\ \mathbf{V} - \mathbf{b}_V \\ P_D - b_{baro} \\ \mathbf{N} \cdot \mathbf{R}^T(\mathbf{V} + \mathbf{W}) \end{bmatrix} + \begin{bmatrix} \mathbf{e}_{GPS} \\ \mathbf{e}_V \\ \mathbf{e}_{baro} \\ \mathbf{e}_{air} \end{bmatrix} \tag{3.21}$$

In the measurement equation we can recognize all the measurement models found previously.

3.4 Discrete time linear system

The system function is now to be linearized and discretized. From the system function in (3.18) we can see that the time variance in the system comes from the rotation matrix $\mathbf{R}$. The values in this matrix will change over time when the Euler angles at a certain time instant is inserted. The linearization of the system is done analytically and the system matrix $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are obtained. The coefficients in the matrices are functions of the orientation, which is a function of time. The sizes of the matrices are 19×19 for $\mathbf{A}$, 19×4 for $\mathbf{B}$ and 8×19 for $\mathbf{C}$. All matrices are shown in Appendix D.

For the purpose of this project, the assumption of $\mathbf{A}$ and $\mathbf{u}$ being constant between samples is justifiable as the orientation of the UAV which gives rise to the time variance of $\mathbf{A}$ and the acceleration which is the input are sampled at the highest rate. The method for discretizing a time invariant matrix shown in (2.14) is therefore the one that will be used. These equations are solved

analytically resulting in time varying matrices $\mathbf{F}$ and $\mathbf{G}$. The time variance in these matrices are again from the time variance of the orientation. The sizes of the matrices are 19×19 for $\mathbf{F}(k)$, 19×4 for $\mathbf{G}(k)$ and are shown in Appendix D. At a certain time step, k, the constant matrices are found by inserting the orientation at that time step. The Kalman filter design can now be carried out using these matrices.

3.5 Kalman filter design

When the discrete time system model is found, the Kalman filter can be designed. To ease the computational burden of running the Kalman filter, it will be implemented as an iterated Kalman filter as described in Section 2.3.2. To do this, it has to be assumed that the measurements are independent which means that the measurement noise matrix $\mathbf{Rd}$ is diagonal. The Kalman filter will receive sensor data at different rates from the GPS, airspeed sensor and barometer. To accommodate this, the rows in $\mathbf{C}$ which corresponds to unavailable sensors at a time k will be set to zero, as described in Section 2.3.3. What is left of the Kalman filter design after these choices is to make sure that the system is observable and to find the noise matrices. This will be covered in the following sections.

3.5.1 Observability

The observability is tested for the time varying symbolic matrices found above. The observability is tested by symbolically finding the matrix $\mathbf{CV}_R$ from (2.34). This matrix is given as:

$$\mathbf{CV}_R = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & CV_{84} & CV_{85} & CV_{86} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.22}$$

where CV_{84}, CV_{85} and CV_{86} are given by:

$$\begin{aligned} CV_{84} &= \frac{\cos(\psi)\cos(\theta)}{\cos(\psi)^2\cos(\theta)^2 + \cos(\psi)^2\sin(\theta)^2 + \cos(\theta)^2\sin(\psi)^2 + \sin(\psi)^2\sin(\theta)^2} \\ CV_{85} &= \frac{\cos(\theta)\sin(\psi)}{\cos(\psi)^2\cos(\theta)^2 + \cos(\psi)^2\sin(\theta)^2 + \cos(\theta)^2\sin(\psi)^2 + \sin(\psi)^2\sin(\theta)^2} \\ CV_{86} &= -\frac{\sin(\theta)}{\cos(\theta)^2 + \sin(\theta)^2} \end{aligned} \tag{3.23}$$

For the system to be observable, the matrix $\mathbf{CV}_R$ can not have any zero columns. The only variables the matrix depends on is the pitch θ and the heading ψ. We can find the orientations at which the system is unobservable by setting (3.23) equal to zero and solving for the angles. This is easily done as this is the solution to the sine and cosine functions being equal to zero. The result is that the system is unobservable under the following conditions:

$$\begin{aligned} \psi &= \begin{cases} \frac{\pi}{2} + \pi p & \forall\theta,\ p \in \mathbb{Z} \\ \pi p & \forall\theta,\ p \in \mathbb{Z} \end{cases} \\ \theta &= \begin{cases} \frac{\pi}{2} + \pi p & \forall\psi,\ p \in \mathbb{Z} \\ \pi p & \forall\psi,\ p \in \mathbb{Z} \end{cases} \end{aligned} \tag{3.24}$$

We can see that when the heading or pitch is 0, 90, 180 or 360 degrees, the system will be unobservable. This test is also performed numerically for the system which gives the same result. The result of the numerical test is shown in Figure 3.1. From examining this plot we can see that the observability depends on θ and ψ and that the system is unobservable at 0, $\pm\pi/2$ and $\pm\pi$ which corresponds with what was found analytically in (3.24). An observable subspace decomposition could be carried out in order to find which states were observable (Hendricks et al., 2008, page 157). However, due to an ill-conditioned system matrix, this does not give consistent results in Matlab. An argumentation for which states that will be unobservable can instead be given by examining the measurement equations. The airspeed sensor only measures the airspeed in one direction, which is the direction of flight. With only a kinematic model of the airplane, the wind velocity is only observable through the airspeed sensor. This can also be seen from the measurement equations in (3.21) where the states $\mathbf{W}$ are only in the airspeed measurement equation. This means that the components of the wind velocity which are perpendicular to the one airspeed measurement will be unobservable. When revisiting the cases where the system was unobservable, we see that they can all be explained by the unobservability of the wind.

Most of the time, the system is working with no GPS input. This means that the rows in $\mathbf{C}$ that corresponds to the GPS measurements are zero. In this case, the system is not fully observable. The states which are observable through the

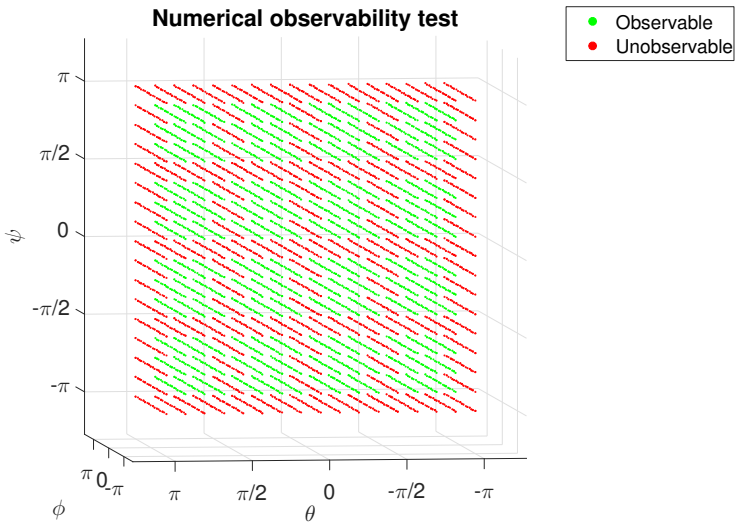


Figure 3.1: Result of the numerical observability test for a number of positions. Note that this is a 3D-plot. We can see that the observability depends on θ and ψ, but not on ϕ

barometer and airspeed sensor are P_D, b_{baro}, $\mathbf{V}$ and $\mathbf{W}$ but the states may be indistinguishable from each other when there is no GPS measurement.

3.5.2 Noise matrices

The process noise covariance matrix for the Kalman filter was set to:

$$\mathbf{V_1} = \text{diag}((\sigma_P^2)_{\times 3}, (\sigma_a^2)_{\times 3}, (\sigma_W^2)_{\times 3}, (\sigma_{bias}^2)_{\times 10}) \tag{3.25}$$

Where $\sigma_P = 0.1$, $\sigma_a = 0.3$, $\sigma_W = 0.1$ and $\sigma_{bias} = 0.005$. The prefix below the variances is used to indicate the number of times the term is repeated, thus the process noise covariance matrix $\mathbf{V_1}$ is a 19×19 matrix with the variances along its diagonal.

The measurement noise covariance matrix was set to:

$$\mathbf{V_2} = \text{diag}((\sigma_{GPS}^2)_{\times 3}, (\sigma_V^2)_{\times 3}, \sigma_{baro}^2, \sigma_{air}^2) \tag{3.26}$$

Where $\sigma_{GPS} = 1$, $\sigma_V = 0.05$, $\sigma_{baro} = 0.2$ and $\sigma_{air} = 0.2$. Since there is eight measurements the measurement noise covariance matrix $\mathbf{V_2}$ is an 8×8 matrix.

The sampling time used for the discretization of the process- and measurements noise covariance matrices according to (2.20) was:

$$\begin{aligned} T_s &= 0.02\,\text{s} \\ T_{GPS} &= 1\,\text{s} \\ T_{meas} &= 0.02\,\text{s} \end{aligned} \tag{3.27}$$

Where T_s is the sample time for the process, thus the it is the sample time for the time update step in the Kalman filter. The GPS is assumed to have a update rate of only 1 Hz, thus the sample time of the GPS was set to $T_{GPS} = 1\,\text{s}$. Both the barometer and airspeed sensor were assumed to have equal sample time at $T_{meas} = 0.02\,\text{s}$.

To discretize the process noise matrix we use (2.20):

$$\mathbf{Qd}(k) \approx \mathbf{B}_v(k)\mathbf{V}_1\mathbf{B}_v^T(k)T_s \tag{3.28}$$

where $\mathbf{V}_1$ is given by (3.25), T_s is given above and $\mathbf{B}_v$ is given as:

$$\mathbf{B}_v = \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & R_{11} & R_{12} & R_{13} & & \\ & & & R_{21} & R_{22} & R_{23} & & \\ & & & R_{31} & R_{32} & R_{33} & & \\ & & & & & & 1 & \\ & & & & & & & \ddots \end{bmatrix} \tag{3.29}$$

This matrix is a 19×19 matrix with ones in the diagonal except for where the rotation matrix $\mathbf{R}$ is. All empty space is zeros. The resulting $\mathbf{Qd}$ is given in Appendix D.

The measurement noise matrix $\mathbf{Rd}$ is given by (2.21):

$$\mathbf{Rd} \approx \mathbf{V}_2\mathbf{T}^{-1} \tag{3.30}$$

where

$$\mathbf{T} = \text{diag}((T_{GPS})_{\times 6}\,, T_{meas}\,, T_{meas}) \tag{3.31}$$

The matrix is given in Appendix D.

Chapter 4

Guidance system

In this chapter, the guidance system for a UAV will be discussed. The guidance system consists of a path planning algorithm, which is used to calculate the path between two waypoint under some given constraints. Based on the current position of the UAV the guidance system will then output set-points used by the controllers in order to follow the planned path.

4.1 Dubins path

The Dubins path is named after the American mathematician *Lester Dubins*, who published a paper back in 1957 (Dubins, 1957) showing that the shortest path between two points with a constraint on the curvature and final tangents, for a particle that can only travel forward, consist of circular arcs and straight line segments. He showed that the shortest path only consists of three segments of the type CCC or CSC where C denotes either a left or right circle arc segment and S is a straight line segment. Thus the end segments will always be a circle arc with either another circle arc or straight line in-between. This is very useful for UAVs as it gives an optimal way of calculating the shortest path between two waypoints given by their x,y-coordinates and heading angle ψ constrained by the maximum turning radius ρ.

In the paper by Shkel and Lumelsky (2001) the following three operators are defined for each type of segment:

$$\begin{aligned} L_v(x,y,\psi) &= (x+\sin(\psi+v)-\sin\psi, y-\cos(\psi+v)+\cos\psi, \psi+v) \\ R_v(x,y,\psi) &= (x-\sin(\psi-v)+\sin\psi, y+\cos(\psi-v)-\cos\psi, \psi-v) \\ S_v(x,y,\psi) &= (x+v\cos\psi, y+v\sin\psi, \psi) \end{aligned} \tag{4.1}$$

where x and y are the x,y-coordinates, ψ is the heading of the vehicle and the subscript v denotes the length of that particular segment. By introducing a rectangular coordinate system with its origin in the initial point P_i and the x-axis pointing in the direction of the final point P_f. The coordinates of these two points can be simplified to $P_i = (0,0)$ and $P_f = (d,0)$. Furthermore the angle of each point is α and β respectively.

If a left and right circular arc are denoted by L and R respectively and a straight line is denoted S, the six possible combinations of segments are: *LSL*, *LSR*, *RSR*, *RSL*, *RLR* and *LRL*. Thus by using the boundary conditions for the two points the path *LSR* is given by $R_q(S_p(L_t(0,0,\alpha))) = (d,0,\beta)$. Where t, p and q denotes the length of each segment. By solving these three equations for t, p and q one can calculate the total length of the Dubins path in the x,y-plane:

$$\mathcal{L} = t + p + q \tag{4.2}$$

Thus the path length of all these six combinations can be determined by inserting (4.1) into one another and solving for the boundary conditions. The shortest path is then simply given by the minimum path length found. The derivations of all six combination are shown in (Shkel and Lumelsky, 2001) and the results are summarized in Appendix E. This approach is very useful, as it calculates the length of each segment directly, which is beneficial when limited computation time is available, as it will be on a UAV.

Note that the above results are only valid for a so-called *long path case*, which is constrained by $d > \sqrt{4-(|\cos\alpha|+|\cos\beta|)^2}+|\sin\alpha|+|\sin\beta|$, thus if this condition is not fulfilled a sub-optimal solution is found using the equations presented.

A UAV will be limited by the maximum allowed roll angle ϕ_{max}. For a given airspeed V_{air} the turning rate ω is given by:

$$\omega = \frac{g_0}{V_{air}} \tan\phi_{max} \tag{4.3}$$

The turning radius ρ is given by:

$$\rho = \frac{V_{air}}{\omega} \tag{4.4}$$

By inserting (4.3) into (4.4):

$$\rho = \frac{V_{air}^2}{g_0 \tan \phi_{max}} \tag{4.5}$$

Thus the turning radius ρ of a UAV is given by the airspeed V_{air}, the gravitational constant g_0 and the maximum roll angle ϕ_{max} (Briod, 2008, Eq. (3.4-3.6)).

The results from (Shkel and Lumelsky, 2001) where implemented by Walker (2008–) and was used as means of calculating the length and type of the shortest path in *C* and *Matlab*. Figure 4.1 shows an example of a Dubins path for the points $P_1 = (0, 0)$ with heading angle $\psi_1 = 0$ and $P_2 = (10, 15)$ with heading angle $\psi_2 = -\pi/4$. The airspeed V_{air} was set to $5\,\mathrm{m/s}$, the maximum roll angle ϕ_{max} was $\pm 30°$ and the gravitational constant g_0 was $9.81\,\mathrm{m/s^2}$. By inserting into (4.5) the turning radius was calculated to be $\rho = 4.4140\,\mathrm{m}$, as it can be seen the resulting shortest path between these point is a *LSR* path.

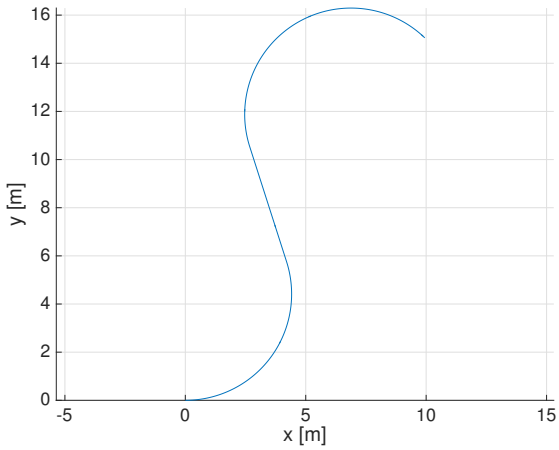


Figure 4.1: Dubins path (*LSR*) for $P_1 = (0, 0)$ and $P_2 = (10, 15)$ with a heading angle of $\psi_1 = 0$ and $\psi_2 = -\pi/4$ respectively. The turning radius was set to $\rho = 4.4140\,\mathrm{m}$

4.2 Vector field guidance

In the work by Nelson et al. (2006) a solution for calculating a converging vector fields for circles and straight lines are given. The direction of the vector fields can be used as a desired heading angle, ψ_d, allowing the UAV to reach the desired path asymptotically.

4.2.1 Circular vector field

The algorithm for calculating the circular vector field when the UAV needs to stay at a circular segment will be describe in the following section.

The distance from UAV to the center of the circle is given by:

$$d = ||z - c|| \tag{4.6}$$

where $z = (x, y)^T$ is the x,y-coordinates of the current location of the UAV and $c = (c_x, c_y)^T$ is the x,y-coordinates of the origin of the circle.

The current angle of the UAV with relation to the origin of the circle is given by:

$$\gamma = \text{atan2}(y - c_y, x - c_x) \tag{4.7}$$

where atan2 is the four quadrant inverse tangent function.

If the distance from the center of the circle is larger than twice the radius of the circle r i.e. $d > 2r$ the angle of the vector is given by:

$$\psi_d = \gamma + \text{dir} \left[\pi - \sin^{-1}\left(\frac{r}{d}\right)\right] \tag{4.8}$$

where dir is the turning direction following the right-hand rule i.e. 1 when turning counter-clockwise (CCW) and -1 when turning clockwise (CW).

If the distance is less or equal twice the radius of the circle the angle of the vector is given by:

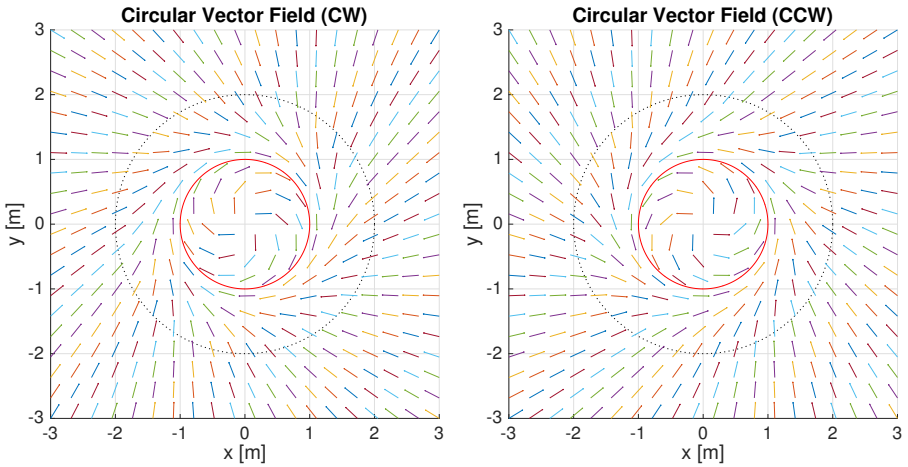
$$\psi_d = \gamma + \text{dir} \left[\frac{\pi}{2} + \frac{\pi}{3}\left(\frac{d - r}{r}\right)^{k_1}\right], \; k_1 >= 1 \tag{4.9}$$

where k_1 is the convergence gain and can be tuned by the user.

Figure 4.2 shows a vector field generated for a circle in the clockwise (CW) and counter-clockwise (CCW) direction respectively. Where the dotted line indicates the $d > 2r$ region.

4.2.2 Line vector field

When the UAV needs to stay at a straight line connecting the two waypoints $w_1 = (w_{1_x}, w_{1_y}), w_2 = (w_{2_x}, w_{2_y})$ a different algorithm is needed.



(a) CW circular vector field **(b)** CCW circular vector field

Figure 4.2: Circular vector fields. The vector field is generated for the red circle. Outside the dotted circle, the vector fields are calculated according to (4.8) while they are calculated using (4.9) inside this region

The heading between the two waypoints is given by:

$$\psi_f = \text{atan2}(w_{2_y} - w_{1_y}, w_{2_x} - w_{1_x}) \tag{4.10}$$

The normalized distance along the path $s \in [0, 1]$ needs to be calculated:

$$s = \frac{(z - w_1)^T (w_2 - w_1)}{||w_2 - w_1||^2} \tag{4.11}$$

The distance of the UAVs current location $z = (x, y)^T$ from the path is given by:

$$\epsilon = ||z - [s\,(w_2 - w_1) + w_1]\,|| \tag{4.12}$$

The sign of the cross product between the distance between the two waypoints and the distance between the current location and the first waypoint indicates which side of the straight line the UAV is currently on:

$$\eta = \text{sign}\,[(w_2 - w_1) \times (z - w_1)] \tag{4.13}$$

The distance from the path is then redefined:

$$\epsilon = \eta\epsilon \tag{4.14}$$

If the absolute distance is greater than the transition region boundary distance τ i.e. $|\epsilon| > \tau$ the angle of the vector is given by:

$$\psi_d = \psi_f - \eta\psi_e; \tag{4.15}$$

where ψ_e is the entry heading angle and can be tuned by the user.

On the other hand if the absolute distance is less or equal to the transition region boundary distance $|\epsilon| <= \tau$ the angle of the vector is given by:

$$\psi_d = \psi_f - \psi_e \left(\frac{\epsilon}{\tau}\right)^{k_2}, \; k_2 >= 1 \tag{4.16}$$

where k_2 is the transition gain. It should be noted that if $s > 1$, a waypoint has been missed and action has to be taken by the underlying guidance algorithm, this will be discussed more in section 4.6.2.

Figure 4.3 shows the result of plotting the vector field for a straight line. The dotted line indicates the transition region boundary distance. An entry heading angle of $\psi_e = \pi/4$ was chosen. Notice how the angle between the vector field and transition region boundary distance is just equal to $\pm\psi_e$ outside the transition region boundary distance according to (4.15), while the vector fields converges inside this region.

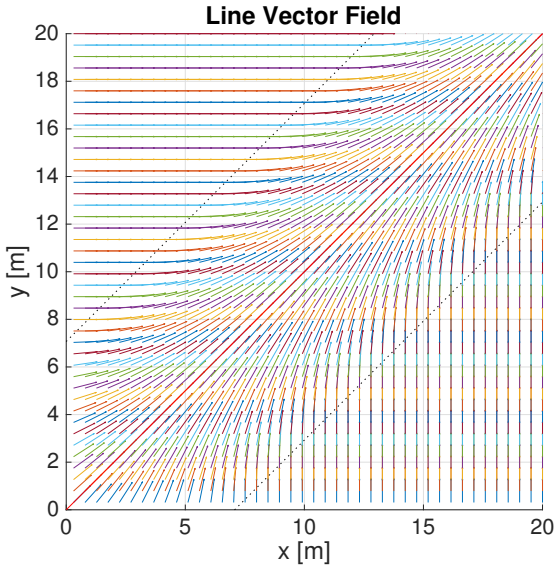


Figure 4.3: Line vector field. The red line is the straight line segment for which the vector field is generated, while the dotted lines indicates the transition region boundary distance

4.3 Sub-waypoint calculation

As the vector field calculation depends on the type of segment that the UAV is on at a given time, it is important to be able to determine this. One way of doing this is to introduce two sub-waypoints between the segments. Figure 4.4 shows an example of the two sub-waypoints found for the *LSR* Dubins path that was shown in Figure 4.1.

The origin of the circle C_1 is given by the following:

$$\begin{aligned} C_1 &= (w_{1_x} - \mathrm{dir}\rho \sin w_{1_\psi}, w_{1_y} + \mathrm{dir}\rho \cos w_{1_\psi}) \\ &= (0 - 4.4140 \sin 0, 0 + 4.4140 \cos 0) \\ &= (0, 4.4140) \end{aligned} \tag{4.17}$$

where w_{1_x}, w_{1_y} and w_{1_ψ} are the x,y-coordinates and heading angle of the first waypoint. dir is the turning direction of the first segment, since it is a left turn, the direction is positive according to the right-hand rule i.e. $\mathrm{dir} = 1$.

The heading at the sub-waypoint w_{sub1} can now be calculated using the length

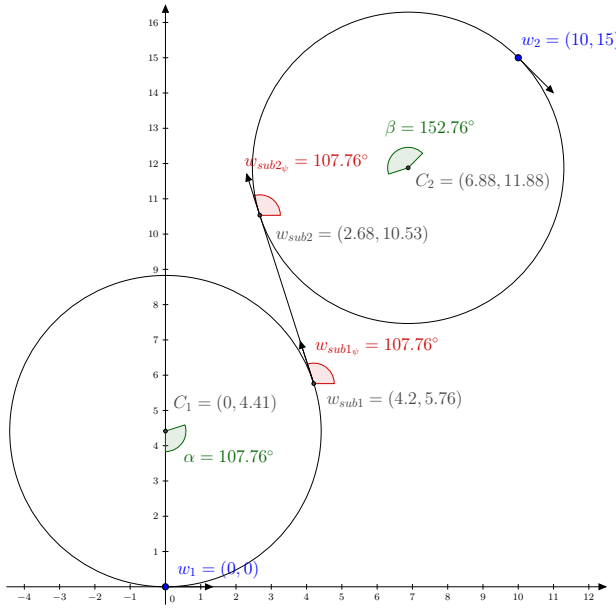


Figure 4.4: Example of sub-waypoint calculation for the Dubins path for $w_1 = (0, 0)$, $w_{1_\psi} = 0$, $w_2 = (10, 15)$, $w_{2_\psi} = -\pi/4$ and $\rho = 4.4140\,\text{m}$

of the first segment t obtained using (E.7) in Appendix E:

$$\begin{aligned} w_{sub1_\psi} &= w_{1_\psi} + \text{dir}\left(\frac{t}{\rho}\right) \mod 2\pi \\ &= 0 + \frac{8.3017}{4.4140} \mod 2\pi \\ &= 1.8808\,\text{rad} = 107.7620° \end{aligned} \tag{4.18}$$

where mod is the modulo operator.

Now the x,y-coordinates of the sub-waypoint w_{sub1} can be determined:

$$\begin{aligned} w_{sub1_x} &= C_{1_x} + \text{dir}\rho \sin w_{sub1_\psi} \\ &= 0 + 4.4140 \sin 107.7620° = 4.2036 \end{aligned} \tag{4.19}$$

$$\begin{aligned} w_{sub1_y} &= C_{1_y} - \text{dir}\rho \cos w_{sub1_\psi} \\ &= 4.4140 - 4.4140 \cos 107.7620° = 5.7605 \end{aligned} \tag{4.20}$$

The origin of the circle C_2 is calculated in a similar way, however the turning direction of the third segment is a right turn, thus the turning direction is

negative i.e. dir $= -1$:

$$\begin{aligned} C_2 &= (w_{2_x} - \text{dir}\rho \sin w_{2_\psi}, w_{2_y} + \text{dir}\rho \cos w_{2_\psi}) \\ &= (10 + 4.4140 \sin\left(\frac{-\pi}{4}\right), 15 - 4.4140 \cos\left(\frac{-\pi}{4}\right)) \\ &= (6.8788, 11.8788) \end{aligned} \tag{4.21}$$

The heading of the second sub-waypoint w_{sub2} can be calculated using the length of the third segment q. Again this is found using (E.7) in Appendix E:

$$\begin{aligned} w_{sub2_\psi} &= w_{2_\psi} - \text{dir}\left(\frac{q}{\rho}\right) \mod 2\pi \\ &= \frac{-\pi}{4} + \frac{11.7684}{4.4140} \mod 2\pi \\ &= 1.8808\,\text{rad} = 107.7620^\circ \end{aligned} \tag{4.22}$$

Notice how the sign has been swapped compared to (4.18) this is because the sub-waypoint w_{sub2} is before the second waypoint w_2 in the path. In this case the heading of the two sub-waypoints are the same, this will always be the case for a CSC path, as the two sub-waypoints will be connected by a tangent line for the two circles C_1 and C_2.

The x,y-coordinates of the second sub-waypoint w_{sub2} can now be calculated:

$$\begin{aligned} w_{sub2_x} &= C_{2_x} + \text{dir}\rho \sin w_{sub2_\psi} \\ &= 6.8788 - 4.4140 \sin 107.7620^\circ = 2.6752 \end{aligned} \tag{4.23}$$

$$\begin{aligned} w_{sub2_y} &= C_{2_y} - \text{dir}\rho \cos w_{sub2_\psi} \\ &= 11.8788 + 4.4140 \cos 107.7620^\circ = 10.5323 \end{aligned} \tag{4.24}$$

4.4 2D guidance

If the length of the first and last segment and the type of Dubins path is known, it is possible to calculate the two sub-waypoints and since the type of Dubins path is known, the type of segment can easily be determined and the corresponding vector field calculated.

Figure 4.5 shows a plot of two Dubins path between three waypoints. In the example the convergence gain and the transition gain was set to $k_1 = 1$ and $k_2 = 1$. The transition region boundary distance was set to $\tau = \rho$ and the entry heading angle and was set to $\psi_e = \pi/4$.

The sub-waypoints are first calculated and the vector field for the path is plotted next to it. The type of segment is determined by assuming it starts at the first segment. In this case it will plot a circular vector field until it gets within a certain distance of the first sub-waypoint, the next sub-waypoint is then set as the objective and a line vector field is calculated until it reaches the second sub-waypoint. This is then repeated until it reaches the last waypoint on the path.

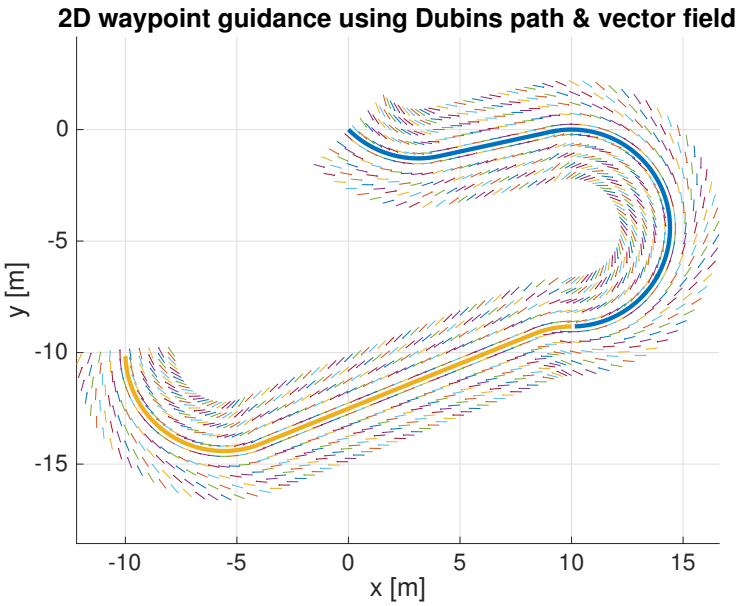


Figure 4.5: Example of two continuous *LSR* Dubins paths with circular and straight line vector fields that converges asymptotically towards the path

4.5 Dubins airplane

Inspired by the work in (Beard and McLain, 2013; Briod, 2008) the Dubins path algorithm was extended for 3D applications, as the Dubins path described so far is only for 2D, thus we need to extend it in the z-axis as well. To differentiate the 2D and 3D Dubins path they are often called the Dubins car and Dubins airplane respectively.

If the maximum allowed pitch angle of the UAV is denoted θ_{max} and the Dubins car path length L_{car} is the one calculated in (4.2), then the UAV will be able

to fly between the two waypoints if the following condition is satisfied:

$$|w_{2_z} - w_{1_z}| \leq L_{car} \tan \theta_{max} \tag{4.25}$$

If the condition is not satisfied the UAV will have to extend the Dubins car path length L_{car} in order to satisfy (4.25). One way of doing it is to gain altitude by spiralling n number of times either down or up, thus increasing L_{car}.

It is desired for the UAV to travel in the highest altitude for as long time as possible in order to avoid any collisions and ground turbulence, thus if the first waypoint is located below the second waypoint i.e. $w_{1_z} < w_{2_z}$ helices should be generated in the beginning of the first segment. On the other hand if it is located above $w_{1_z} > w_{2_z}$ the first two first segments should first be flown and then the UAV should helix down.

The number of helices that should be performed is given by:

$$n = \left\lceil (2\pi\rho)^{-1} \frac{|w_{2_z} - w_{1_z}|}{\tan \theta_{max}} - L_{car} \right\rceil \tag{4.26}$$

where the brackets $\lceil\ \rceil$ is the ceil function which rounds the value up to the nearest integer, thus satisfying (4.2). Note that because of the rounding, we will end up in a higher altitude than in the optimal solution however this suboptimal solutions simplifies the implementation a lot, as the x,y-coordinates of the sub-waypoint does not change.

If the helices are done in the beginning of the path, the length of the first segment, t, will be increased by the following relationship:

$$t = t + 2\pi\rho\, n \tag{4.27}$$

Similarly, when the UAV helixes down, the length of the last segment, q, will be increased accordingly:

$$q = q + 2\pi\rho\, n \tag{4.28}$$

Now the z-coordinate of the two sub-waypoints can be calculated assuming the altitude change is just a linear function:

$$w_{sub1_z} = (w_{2_z} - w_{1_z}) \frac{t}{L_{car}} + w_{1_z} \tag{4.29}$$

$$w_{sub2_z} = (w_{2_z} - w_{1_z}) \frac{t+p}{L_{car}} + w_{1_z} \tag{4.30}$$

Note that L_{car} has to be recalculated in this case according to (4.2).

Assuming the pitch angle is constant throughout the entire path the angle can be calculated:

$$\theta = \text{atan2}(w_{2_z} - w_{1_z}, L_{car}) \tag{4.31}$$

The total length of the Dubins airplane path is then given by:

$$L_{air} = \frac{L_{car}}{\cos\theta} \tag{4.32}$$

4.5.1 Determination of desired altitude

Given a straight line segment S between the two waypoints, w_1 and w_2, and the current position of the UAV, $(x, y, z)^T$, a vector a between the two waypoints is given by:

$$a = w_2 - w_1 \tag{4.33}$$

A vector, b, between the current position and the first waypoint is given by:

$$b = (x, y, z)^T - w_1 \tag{4.34}$$

The projection of b onto a is then given by (Department of Mathematics, Oregon State University, 1996):

$$proj_{ab} = \frac{a \cdot b}{|a|^2} a = \frac{a \cdot b}{a \cdot a} a \tag{4.35}$$

where $\cdot$ denotes the dot product.

By adding the projected vector, $proj_{ab}$, to the first waypoint, w_1, one can obtain the x,y,z-coordinates of the closest intersection between the two vectors:

$$inter = w_1 + proj_{ab} \tag{4.36}$$

The z-component of the closest intersection can be used as the desired z-position, z_d, for the UAV when travelling on a straight line segment, S. On the other hand, when the UAV is located on a circular segment, C, the desired z-position z_d is simply set to the z-coordinate of the waypoint in the end of the current segment.

4.6 3D guidance

Figure 4.6 shows a plot of a Dubins airplane path. The vector fields are plotted next to it in the x,y-plane. A UAV could follow the path shown by calculating the angle of the vector at its current location and using that angle as the desired heading. The desired z-position, z_d, could be calculated, as described above and could be used as the set-point for the altitude controller.

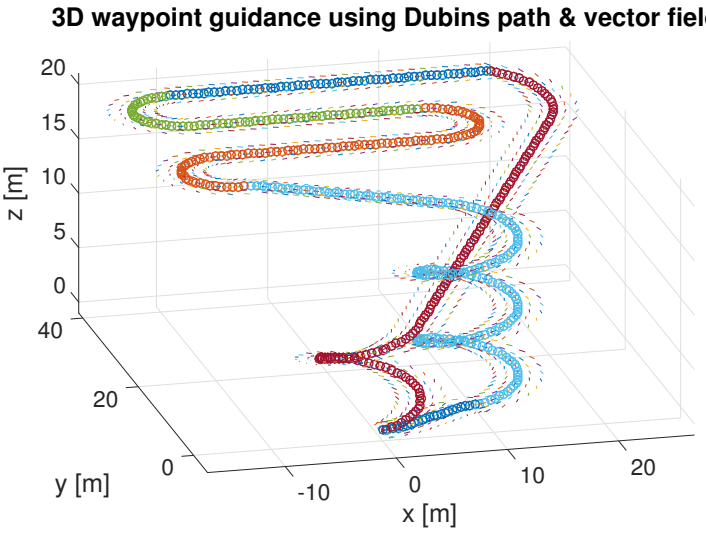


Figure 4.6: Example of six continuous Dubins airplane paths. The vector fields are plotted in the x,y-plane next to the path

4.6.1 Waypoint switching

Initially, the UAV will be at the first waypoint, w_1, and flying along a circular segment C. The UAV reaches the next waypoint, w_{sub1}, when it is within a certain distance, w_{min_dist}, of it:

$$||w_{sub1} - (x, y, z)^T|| < w_{min\_dist} \tag{4.37}$$

The guidance algorithm then automatically determines the type of the next segment and the corresponding vector field. This is repeated until the last waypoint, w_2, is reached. If there are more paths in the queue, the last waypoint will be set as the initial waypoint and the procedure is repeated until the final waypoint is reached. Once that is done, the UAV will simply circle around that waypoint until a new waypoint is given.

4.6.2 Missed waypoint considerations

Due to the possibility of the UAV working in conditions where external factors can affect it, a strategy needs to be considered in case a waypoint is missed. In the case of a circle arc segment, C, nothing special has to be done, as the UAV will simply follow the circular vector field in the x,y-plane and use the desired altitude as the set-point in the z-axis. This will make the UAV circle around until the waypoint is within a certain distance according to (4.37). However, if the UAV is travelling along a straight line segment, S, the underlying guidance algorithm has to detect if the UAV has travelled too far and make the UAV turn around. An easy way of determining when it has travelled to far is to check whenever $s > 1$ in (4.11). If this is the case, action has to be taken by the guidance system. In our implementation, the guidance algorithm simply creates a circular vector field around the missed waypoint until it is reached.

Figure 4.7 shows the results when a waypoint is missed. In this case the second sub-waypoint in the first path is missed and the UAV automatically follows a helix path in order to reach the waypoint.

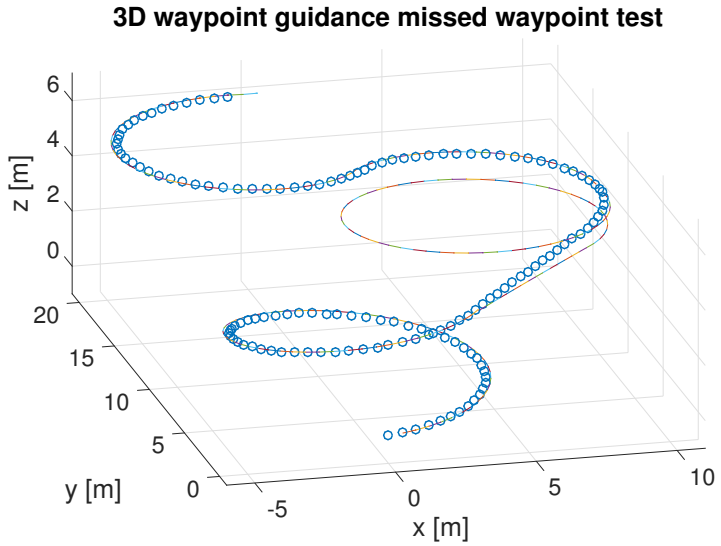


Figure 4.7: Example of the strategy when missing a waypoint on a straight line segment. The UAV will simply start to helix around the missed waypoint until it is reached

The Dubins airplane algoithm developed in this chapter could be expanded to perform a Dubins path in the vertical plane and to use vector field guidance

similar to in the x,y-plane instead of a simple altitude set-point, but since the UAV is assumed to not change its altitude a lot, it has not been implemented. Furthermore, a strategy for autonomous start and landing has not be considered.

Chapter 5

Results & discussion

In this chapter, the results from the simulation of the entire system will first be evaluated and then future improvements will be discussed.

5.1 Simulation results

In order to simulate our system, the 19-state Kalman filter described in chapter 3 was implemented in Matlab. The Dubins airplane and vector field guidance system discussed in chapter 4 was implemented in Matlab as well.

Furthermore, since the customer for the project wants to use the implementations on an embedded device later on, the Kalman filter was implemented in C++, this is describe in more detail in Appendix G. Similarly the Dubins airplane guidance system was implemented in C.

Simulink was used as the simulation environment and the AeroSim Blockset[1] was used to simulate the aircraft dynamics of an Aerosonde UAV[2]. The AeroSim

[1] `http://www.u-dynamics.com/aerosim/default.htm`
[2] `http://www.aerosonde.com`

Blockset also includes realistic environment models such as standard atmosphere, background wind, turbulence and Earth Models such as geoid reference, gravity and magnetic field models. Throughout the simulations, a NED-coordinate system is used. An overview of the Simulink model is shown in Appendix H.

We closed the loop in the model using controllers for the elevator, aileron and throttle inputs. The controllers are manually tuned PID-controllers and are described in more detail in Appendix F. We decided not to use the rudder control surface, as the customer wants to later implement the system on a flying wing.

The Kalman filter was used as an observer for the closed loop, thus providing an estimate of the position, ground velocity, wind speed, barometer bias, GPS biases, accelerometer biases and ground velocity biases, as presented in (3.16). The attitude was assumed to be known and is provided by the model. In a real system, the attitude would be estimated by another algorithm. This estimate would most likely not be perfect and to model this, white noise with a variance of 0.1^2 were added on top of the attitudes given by the Simulink model. The guidance system is used to give set-points for the desired heading, ψ_d, and desired z-position, z_d, based on the current position estimates.

In the case of the simulation, the variances of the different process and measurements are all known, thus the Kalman filter should be able to estimate the different states with high precision. However in a real scenario these would not be known and the measurement data would have to be analysed in order to estimate the variances.

5.1.1 Wind-free simulation

Figure 5.1 shows a plot of the simulation results for simulated flight between the three waypoints:

$$\begin{aligned} w_{1_{xyz\psi}} &= (0, 0, -300, 0) \\ w_{2_{x}yz\psi} &= (1000, 500, -700, 0) \\ w_{3_{x}yz\psi} &= (1000, -500, -570, \pi) \end{aligned} \tag{5.1}$$

with a wind speed set to 0 in all three directions. The desired airspeed, V_{air_d}, was set to $25\,\mathrm{m/s}$, the maximum roll angle, ϕ_{max}, was limited to $\pm 40°$ and the gravitational constant g_0 was set to $9.81\,\mathrm{m/s^2}$. By inserting into (4.5) this gives a turning radius of $\rho = 75.9272$. Furthermore a threshold distance for the waypoint was set to $w_{min\_dist} = 30$.

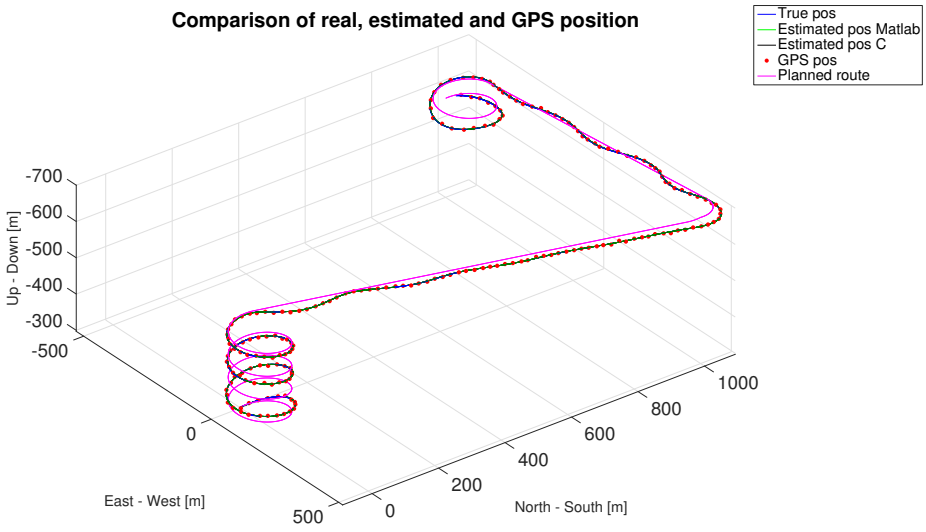


Figure 5.1: Simulation results for no wind conditions

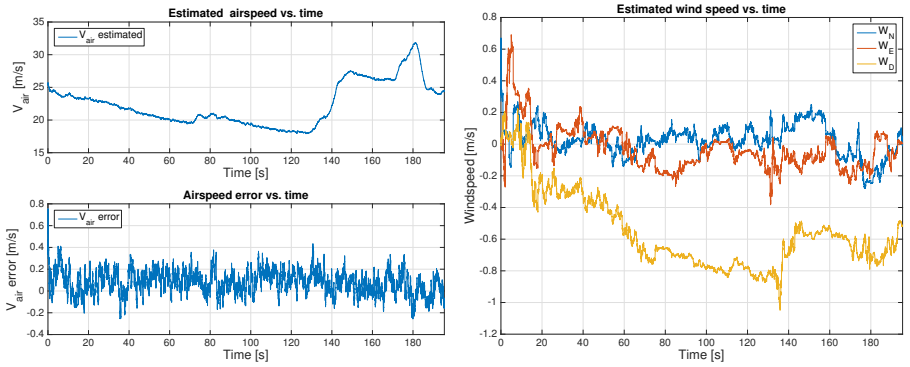
As it can be seen on Figure 5.1, the UAV can successfully fly along the planned path. However, some oscillations can be seen in the x,y-plane. This is most likely due to the controllers not being perfectly tuned as this was not the main focus in this project. Also note how the number of helices is automatically determined in both ends.

5.1.1.1 Airspeed and wind estimates

Figure 5.2a shows the estimated airspeed and airspeed estimation error vs. time. As it can be seen it can successfully estimate the airspeed with an error of approximately $-0.2\,\mathrm{m/s}$ to $0.4\,\mathrm{m/s}$. It is noticed that the airspeed varies quite a bit from the set-point of $25\,\mathrm{m/s}$. The undershoot is caused when the Aerosonde is pitching upward, thus decreasing its speed to approximately $20\,\mathrm{m/s}$ even though the throttle stays at 100 % in the simulation, thus we might have to use a lower climb angle if we want to stay closer to the desired airspeed V_{air_d}. Similarly the airspeed increases to approximately $32\,\mathrm{m/s}$ when the UAV is doing the final downward helix even though the throttle stays at 0 %. In some cases this could make the UAV miss the waypoints. One way of solving this would be to limit the capabilities of the pitching angle, as described earlier, another approach would be to recalculate the the Dubins path for the new airspeed periodically or if the difference between the desired airspeed and the estimated airspeed was larger than a certain value. The most appealing of these solution would be the former

as this would keep the airplane closest to the desired airspeed set-point.

Figure 5.2b shows a plot of the estimated wind speed vs. time. This shows that the UAV can successfully estimate the wind in the north and east direction. However, the estimate of the wind speed in the down direction is not as close to 0 as the other estimates. The reason is that the Kalman filter has no way of estimating the wind speed in the down direction unless the UAV pitches up and down, as the pitot tube is facing forward along the x-axis in the body frame.



(a) **(b)**

Figure 5.2: To the left is estimated airspeed and airspeed error and to the right is estimated wind speed vs. time

5.1.1.2 Bias estimates

Figure 5.3 shows all the bias estimates as a functions of time. In the current simulation all biases were set to 0. As it can be seen, all biases converge. However, none of the bias estimates for the barometer or for the GPS are converging to the true values.

Another thing to notice is that the barometer bias and the bias for the GPS in the downward direction looks to be correlated. The correlation coefficient between the two signals is -0.7898, thus they show sign of inverse correlation. This might indicate that the two states are indistinguishable from one another as an increase in one bias is causing a decrease in the other.

The velocity and accelerometer biases all converge to the true value rather quickly, however it seems unlikely that the biases for the velocity and accelerometer would be so well estimated in a real scenario. For instance the biases of the

accelerometer could be time and/or temperature dependent which would cause the biases to drift. However, since the sensor is modelled exactly as described in section 3.2.1 it is obvious that the biases will converge to the right values.

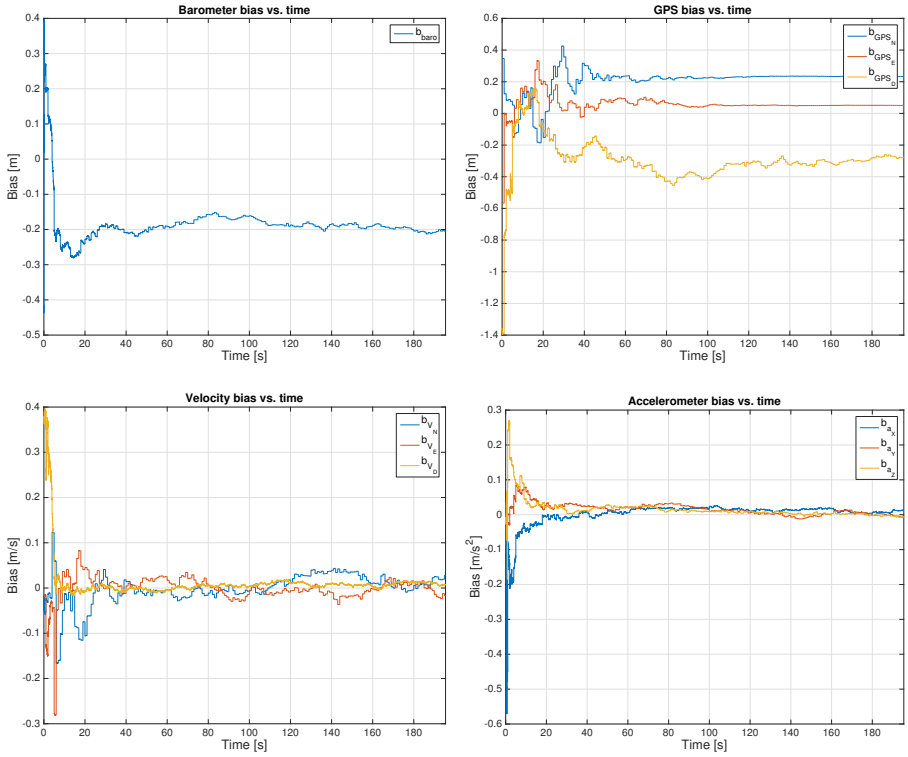


Figure 5.3: Estimation of biases for the various sensors vs. time

5.1.1.3 State errors

Figure 5.4a shows a plot of the difference between the estimated position and the true position. It can be seen that the navigation system can estimate the position with an error of approximately $\pm 1.5\,\mathrm{m}$. The difference between the estimated ground velocity and true ground velocity is plotted in Figure 5.4b, thus it was able to determine the ground velocity with an error at roughly $\pm 0.4\,\mathrm{m/s}$. The difference between the estimated position and the route planned by the Dubins path can be seen in Figure 5.4c. It is calculated as the 2-norm of the current location to the closest point on the planned path. From the figure it can be seen that the UAV can follow the planned path with no more than 37 m

error in the x,y,z-space. The oscillations when flying on a straight line which are also present in Figure 5.1 can be seen here as well. The high peaks in the end of the graph is caused by the overshoot in the last helix caused by the increased airspeed, as shown in Figure 5.2a. It is worth noticing that even though the distance to the path might seem large at times, it is most of the time less than the distance that could be covered by the UAV in 1-2 seconds.

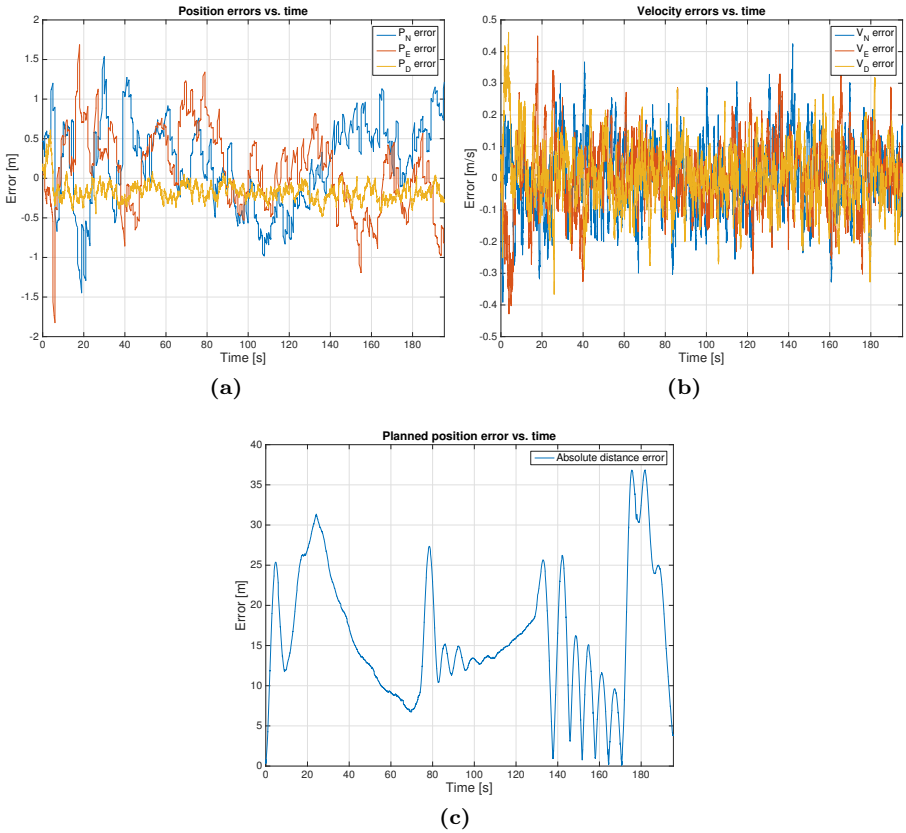


Figure 5.4: (a) and (b) shows the difference between estimated position and ground velocity and their true values. (c) shows the difference between the estimated position of the UAV and planned Dubins path

5.1.1.4 Innovation processes

Figure 5.5 shows the autocorrelation of the innovation process. The green lines in each plot represent the 99 % confidential intervals. The whiteness of the innovation process is an important mean of telling if the Kalman filter is designed properly, as the innovation process should be white noise, as described in section 2.3.4.2.

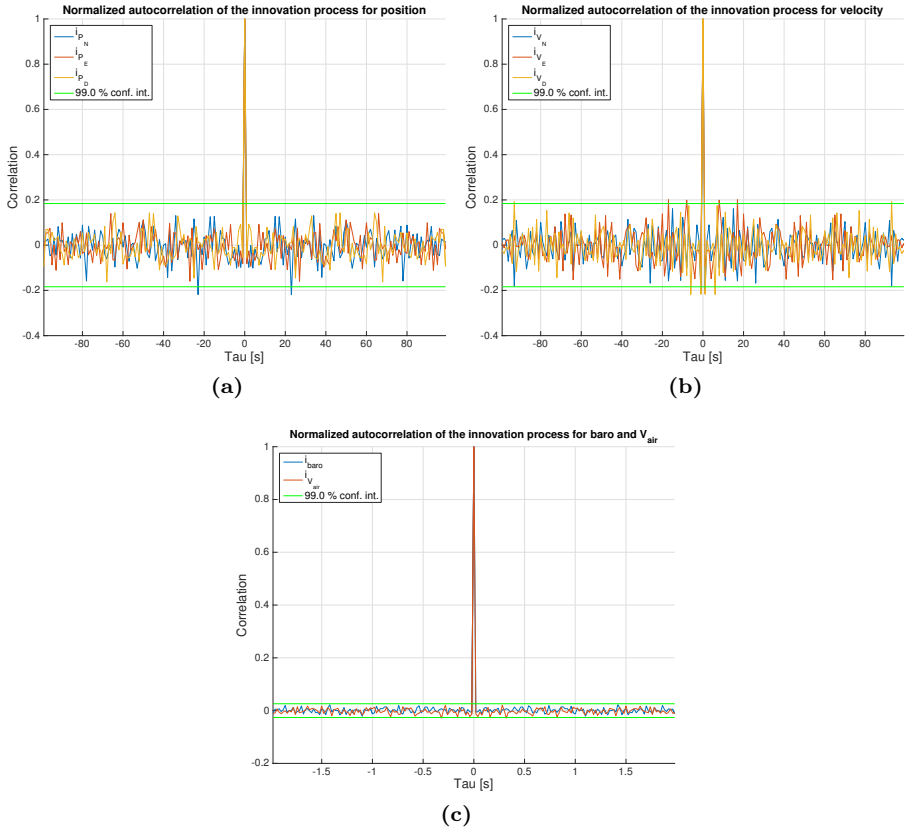


Figure 5.5: The plots show the autocorrelation for the innovation process. On (a) it is shown for the position, on (b) for the velocity and on (c) for the barometer and airspeed sensor. The green lines in all plots are the 99 % confidence intervals

As it can be seen, the innovation process for the position shown in Figure 5.5a looks to be white with only the peaks at roughly $\tau = \pm 22$ crossing the 99 % confidential interval. As the plot shows 100 values of the autocorelation

function, we are expecting one point to be outside the confidence interval on a 99 % level. Figure 5.5b shows the innovation process for the velocity. In this plot there is 1 ponit outside for V_N, 2 points for V_E and 3 points for V_D. This is a bit more problematic as there are too many points outside for two of the innovation processes. It is also problematic for V_N where there is only one point outside as this point is the one that correponds to a lag of one. This can be a sign that the model order is too low. However, finding the cause for this and correcting the Kalman filter model is left as future work. The innovation process for the barometer and airspeed sensors is shown on Figure 5.5c. These plots look as expected as there are no peaks outside the confidential interval for the barometer and only one outside for the airspeed sensor.

5.1.1.5 State estimate variances

Figure 5.6 shows a plot of the variances for all the state estimates. Notice how it takes a little while for the Kalman filter to converge, thus the error is larger when the UAV is first started. In a real application this could be solved by having the UAV sitting on the ground for a little while until the variances have stabilized. The high frequency sawtooth shape displayed by the position variances is due to the variance increasing when there is no GPS input and falling as soon as there is a GPS measurement. The low frequency sawtooth shape of the variance of the wind estimates is caused by the bad observability conditions when flying in a straight line, as explained in section 3.5.1. The variances of the wind speed estimate in the downward direction W_D increases noticeable doing the interval 70 s - 130 s and again at 140 s - 170 s. This is caused by the fact that the UAV is flying straight during those intervals, as it can be seen from Figure 5.7. It can also seen that the variance of the wind speed in the eastern direction W_E rises more than the northern W_N during the interval 70 s - 130 s, as the UAV is flying almost north. The opposite is the case during the interval 140 s - 170 s when the UAV is flying in a more eastern direction.

As the wind estimates generally raises when going in a straight line and converges when doing helices while pitching either up and down. One could imagine a strategy where the UAV would automatically do those manoeuvres if the variances rose above a certain threshold. Since the wind estimates are especially important for autonomous landing it would probably be a good idea for the UAV to helix around the landing site at least once before approaching the runway in order to have a good estimate of the wind velocity.

Figure 5.7 shows the heading of the UAV as a function of time. The figure shows that there are oscillations on the heading when the airplane is flying the straight line segments. This is due to the controllers not being perfectly

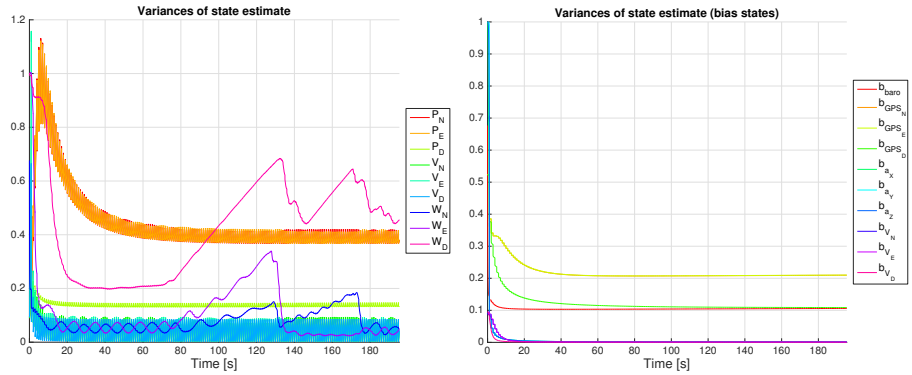


Figure 5.6: Variances of the state estimates

tuned as described earlier. When comparing this figure to the variances of the wind estimate shown in Figure 5.6 it is instantly visible that the variance of the estimate decreases when the UAV is turning but increases when it is flying in a straight line.

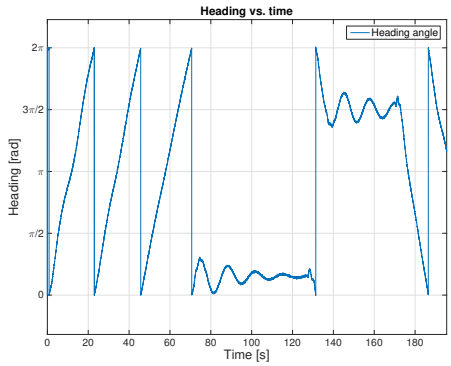


Figure 5.7: UAV heading as a function of time

5.1.2 Wind simulation

Another simulation was performed with the mean wind components set to the following:

$$\mathbf{W} = \begin{bmatrix} 6 \\ 2 \\ 0 \end{bmatrix} \text{m/s} \tag{5.2}$$

The rest of the settings were the same as the ones used previously. This section will explain the results that differ from those of the wind free simulation.

Figure 5.8 shows the resulting flightpath, and it can be seen that it is very similar to the flightpath in the wind free simulation . However, the last straight line path is clearly shifted due to the northern wind component, W_N. This is also noticeable on Figure 5.9a, which shows the distance of the UAV current position to the planned path. Comparing this to 5.4c it can be seen that the error is larger in windy conditions.

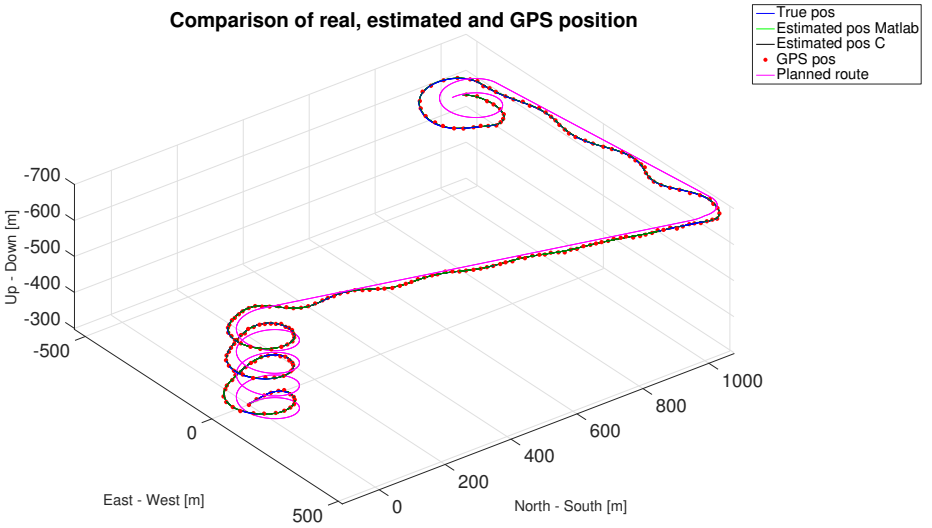
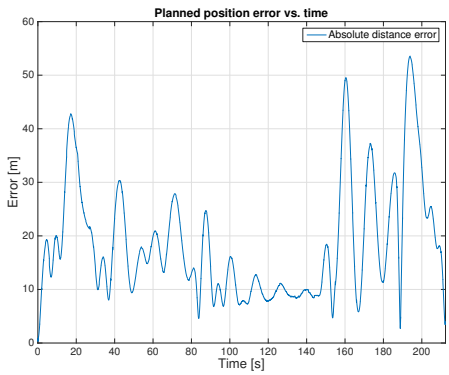


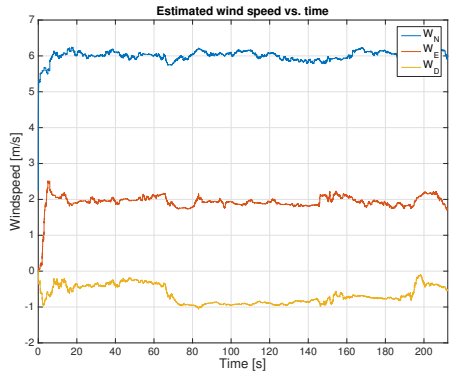
Figure 5.8: Simulation results for wind conditions

Figure 5.9b shows a plot of the estimated wind speed. It can be seen that the Kalman filter successfully converges at the true values for W_N and W_E after roughly 10 s. The downward wind speed components W_D does not converge exactly at 0. This could be caused by the fact that the UAV does not pitch much up and down during the flight so that this component is only weakly observable. Another possible reason is that the state converges to a wrong value due to the angle of attack of the airplane. Comparing the wind estimates to the heading of the airplane which is shown on Figure 5.10a, we see that the estimates for W_N and W_D has converged when the airplane has completed half a helix. This shows that the wind can be estimated relatively fast and even without doing a full helix.

The variances for the estimates states excluding the bias states can be seen in Figure 5.10b. By comparing the variances to the ones for the wind-free simulation in Figure 5.6 it noticeable that the variances are a bit larger for the



(a) Difference between the estimated position of the UAV and planned Dubins path



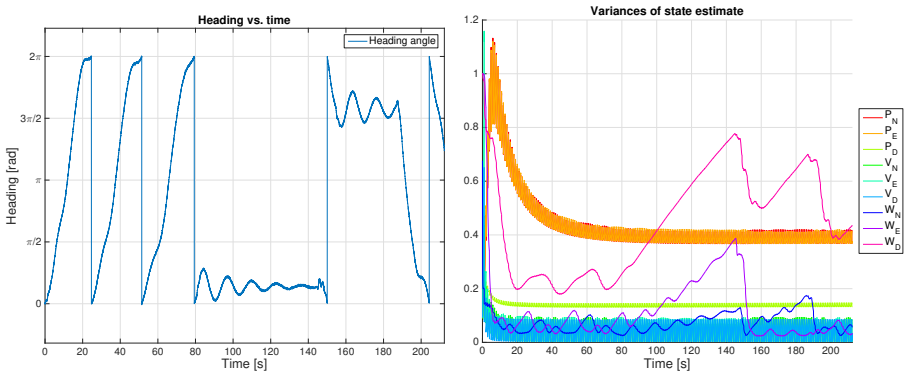
(b) Estimated wind speed vs. time

Figure 5.9

simulation with wind. When looking at the variance for the wind speed we can see that it is at its lowest when it has completed a full helix.

It is important to estimate the wind, especially if autonomous takeoff and landing has to be implemented in the future. It is important for the UAV to takeoff and land against the wind as this will slow its ground speed down if it is flying with constant airspeed. The airspeed is calculated using the wind speed estimate, as seen in (3.14). A wrong estimated airspeed could therefore make the UAV fly too slow and potentially below its stall speed, which in the worst case scenario would stall and crash the UAV.

Furthermore, the wind estimate could be used for a more advanced planning algorithm that could use the estimated airspeed when estimating the optimal path. It could also be used by the vector field generation to increase the transition gain, k_2, for the straight line vector field. This could improve the performance when there is a side-wind, thus decreasing the offset shown in Figure 5.8.



(a) Heading angle of the UAV vs. time

(b) Variance of the state estimates excluding the bias states

Figure 5.10

5.1.3 Wind simulation with biases

In this simulation, the following biases were added to the sensors:

$$
\begin{aligned}
b_{baro} &= -1.3\,\mathrm{m} \\
\mathbf{b}_{GPS} &= \begin{bmatrix} -1.0 \\ 1.1 \\ -1.2 \end{bmatrix} \mathrm{m} \\
\mathbf{b}_{V} &= \begin{bmatrix} 0.4 \\ -0.5 \\ 0.6 \end{bmatrix} \mathrm{m/s} \\
\mathbf{b}_{a} &= \begin{bmatrix} 0.1 \\ -0.15 \\ 0.2 \end{bmatrix} \mathrm{m/s^2}
\end{aligned}
\tag{5.3}
$$

The values of the biases were chosen so as to be realistic, to be of the same magnitude on all three components on a sensor and to be distinguishable from each other. The wind components were set to the same as the ones in (5.2). This section will cover the results that differ from the previous simulations.

Figure 5.11 shows the result of the simulated flight and it can be seen that there is no noticeable differences compared to the simulation with wind and no biases shown in Figure 5.8.

Figure 5.12 shows the estimated biases vs. time. It can be seen that all the

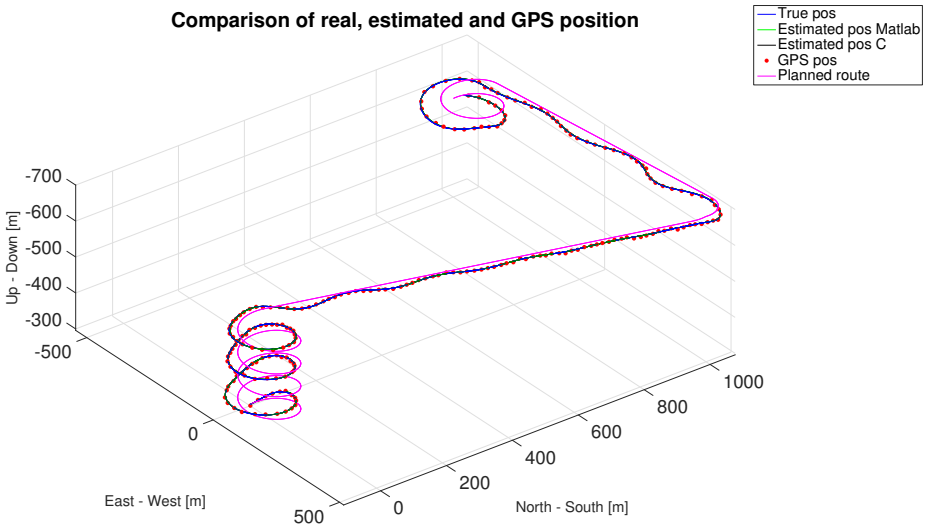


Figure 5.11: Simulation results with both wind and biases

velocity and accelerometer biases are successfully estimated after approximately 10 s. However, the barometer does not converge exactly to the true value of -1.3 m. Worse is the offset in the northern and eastern GPS biases, which was also present in the simulation without wind, as shown in Figure 5.3. Similarly to this simulation it also looks like b_{baro} and b_{GPS_D} are inverse correlated. In this simulation the correlation coefficient between the two signal was calculated to be -0.6841.

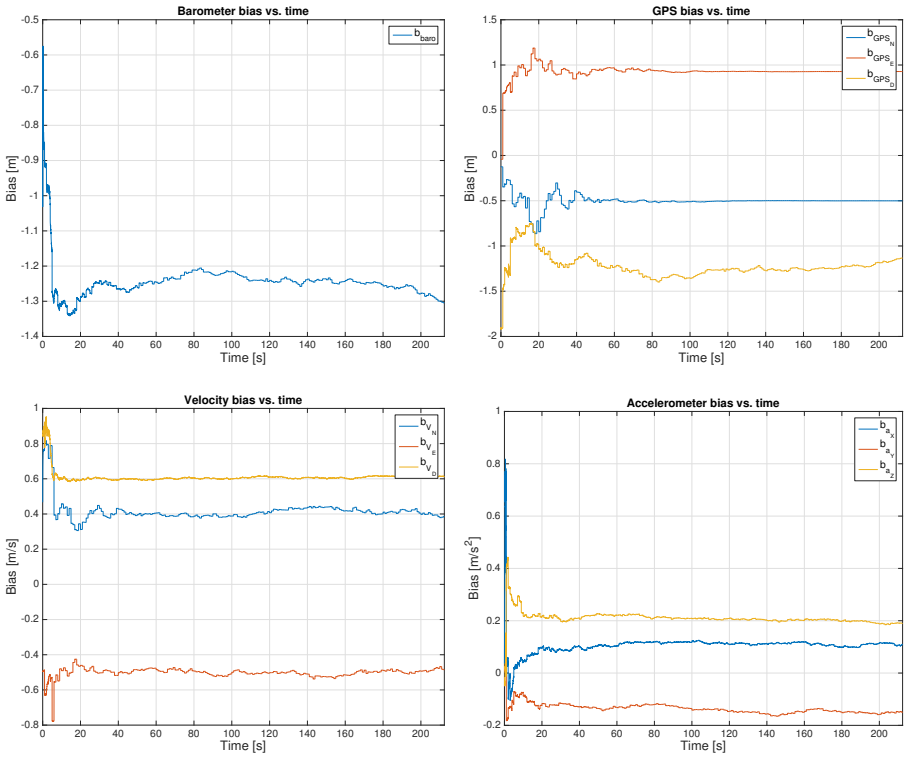


Figure 5.12: Estimation of biases for the various sensors vs. time

5.1.4 Wind simulation with GPS dropout

In this simulation all biases where set back to 0, however this time a GPS dropout was simulated at the interval 80 s - 110 s. This means that no position or velocity measurements were available from the GPS during this interval.

The simulation result is shown on Figure 5.13. The GPS dropout can be seen as the missing dots during the beginning of the first straight line segment. As shown, the UAV can successfully navigate during this interval but with larger position and velocity errors, as seen in Figure 5.14 on 5.14a and 5.14b. The GPS dropout interval is shown with vertical lines. Small oscillations are also visible in the planned position error during the dropout interval, as shown in Figure 5.14c. It can be seen on 5.14d that the wind estimate is updated very little during the GPS dropout. This is due to the lack of observability of the wind states.

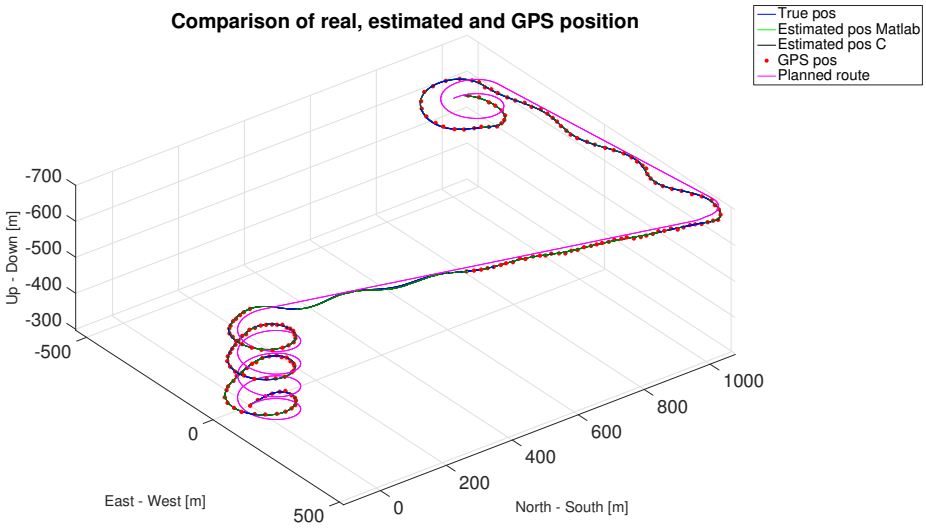


Figure 5.13: Simulation results with GPS dropout. The lack of dotted lines in the GPS position indicates the interval for the GPS dropout

Figure 5.15 shows the variances for the position and velocity estimates. The variances increase during the GPS dropout interval. However, the downward estimate does not rise as much, because of the barometer. This is also the reason for the small error in the downward position and velocity estimate shown in Figure 5.14.

One aspect that might need to be added in the future is resetting of the covariances matrix in the case of a GPS dropout, as the variance will quickly rise, which will cause a delay before the variance drops back to its original value. In this case, the response is very quick, however it might lead to numerical issues if some of the elements in the covariance matrix grow too large, thus the variances might needed to be limited to a certain value.

Figure 5.16 shows a plot of the variances for the wind speed and bias state estimates. It can be seen that the variance of the wind speed estimate rises due to the lack of GPS velocity measurements in (3.13). The variances of the bias states have all converged to an constant during the GPS dropout. Looking at 5.17 it is clear that the bias estimates for the barometer, GPS position and velocity all stay constant during the interval, thus it looks like the GPS dropout makes them unobservable.

Overall, the simulation results for the GPS dropout looks rather good, however it requires very good acceleration estimates, which will probably look worse

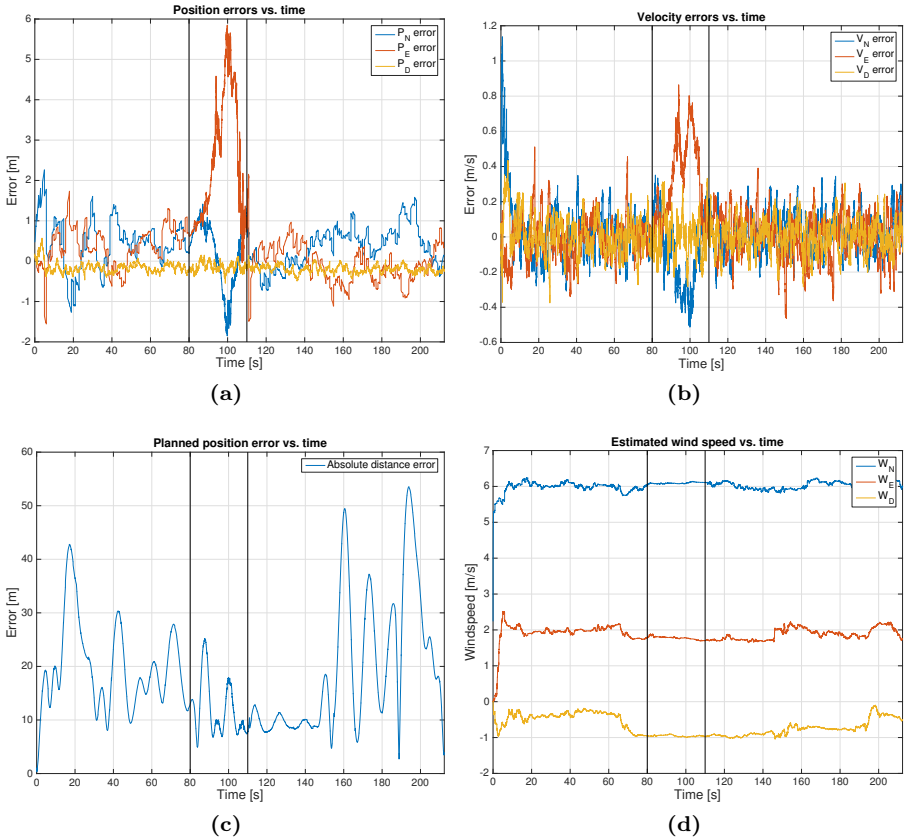


Figure 5.14: Position error, velocity error, planned position error and estimated wind speed. The GPS dropout interval is indicated by the vertical lines

in a real scenario. Furthermore, the results might look worse if the UAV was doing helices during the GPS dropout. In a realistic application, the GPS measurements are critical and the autopilot would probably need to take action in order to prevent the UAV from crashing. One approach would be to start making large helices when a GPS dropout was detected and then simply wait for a GPS fix. If that failed, a last resort would to be deploy a parachute or similar in order to bring the UAV safely down.

One other option would be to redesign the Kalman filter such that it would estimate the ground velocity based on only the airspeed sensor. The wind speed components would then not be updated at all during the GPS dropout such

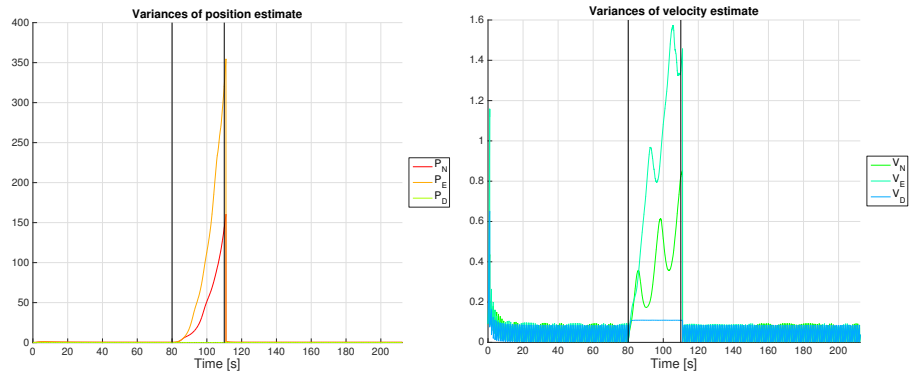


Figure 5.15: Variances of the position and velocity estimates. The GPS dropout interval is indicated by the vertical lines

that the values remained at the last estimated value. Making sure that only the velocity were updated would solve the potential problem of the wind estimate becoming indistinguishable from the accelerometer biases. This could provide a good estimate of the velocity which would give an estimate of the position that could be acceptable for a short period of time until a GPS fix was reestablished.

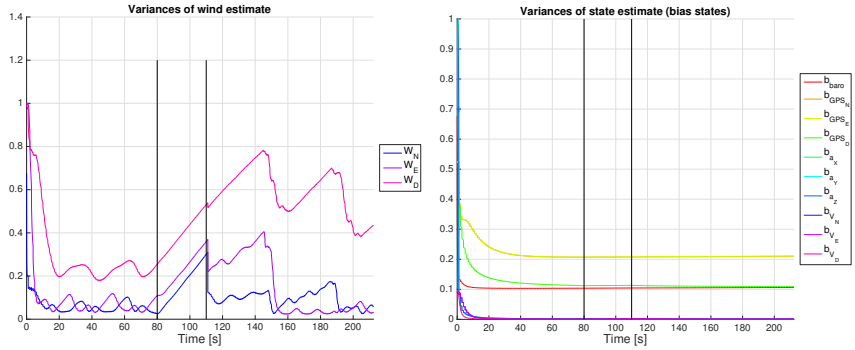


Figure 5.16: Variances of the wind speed and bias state estimates. The GPS dropout interval is indicated by the vertical lines

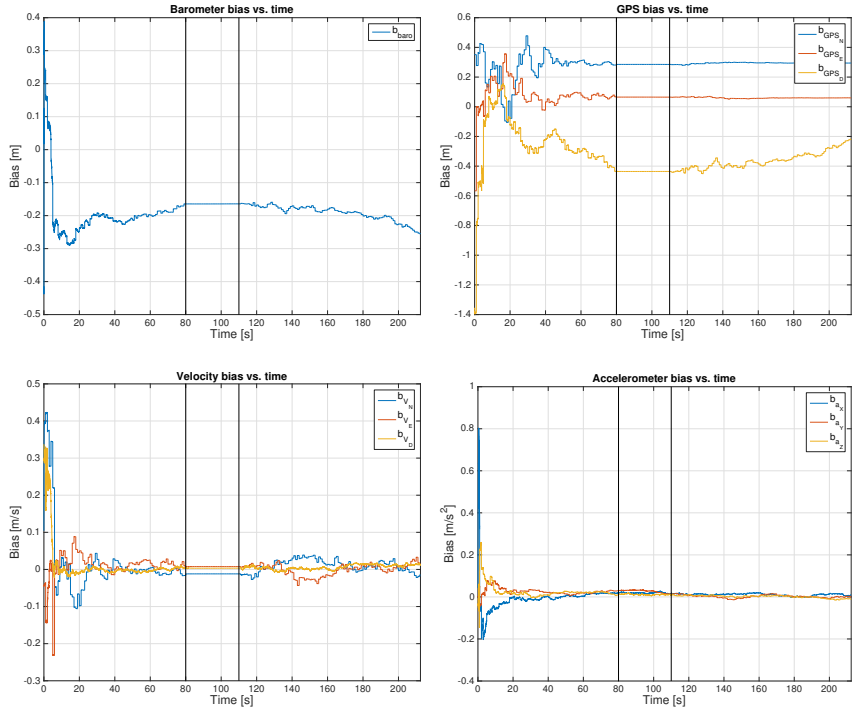


Figure 5.17: Estimation of biases for the various sensors vs. time. The GPS dropout interval is indicated by the vertical lines. Note that the biases all seem to be constant during the GPS dropout

5.2 Discussion

In general, the simulation results were as expected but there was a number of issues that were mentioned. These will be discussed in the following.

5.2.1 Navigation system

As described in section 5.1, the Kalman filter estimates did not converge to the true values for the biases of the barometer and GPS position. At the time of writing the reason is still unknown. It is worth mentioning, that starting the simulation with different initial conditions would result in different errors for b_{GPS_N} and b_{GPS_E}. The simulation was initialised with the plane flying e.g. east instead of north but with the initial condition of the Kalman filter set to the true value. The bias estimates, b_{GPS_N} and b_{GPS_E}, would in this case converge on another set of values. This could be a sign that these bias errors are due to a problem in the simulation. It was also seen that the barometer bias, b_{baro}, and downward position bias, b_{GPS_D}, were inverse correlated. This might be a sign that they are indistinguishable from the Kalman filters point of view. To solve this, the bias for the GPS down position could be modelled differently or removed.

5.2.2 Guidance system

The problems that were present for the guidance system in the simulations in Section 5.1 were an offset from the planned path when subject to wind and problems with following the path when descending due to an increased airspeed. Currently the Dubins path is only calculated before the actual flight and if the airspeed is not kept constant, the minimum turning radius of the UAV would change according to (4.5). When ascending, the airspeed would increase even though the throttle was at 0 %. One way of solving this problem would be to recalculate the Dubins path periodically or if the error in airspeed was too large. It could also be solved by making the Dubins path algorithm even more conservative so that the manoeuvres of the UAV would be gentle enough to let the airspeed controller keep the airspeed closer to the set-point.

An error in position from the planned path was seen when the UAV was subject to wind from the side. If one imagines that the desired heading set by the vector field at a certain distance from the path is exactly the heading that is needed to counter the side wind it becomes apparent that the UAV will not reach the

desired path. This problem could be solved by changing the transition gain, k_2, in the vector field guidance algorithm. However, to get good results from doing this, the transition gain might have to be changed dynamically and a strategy for this will have to be developed.

5.2.3 Future improvements

Overall we are content with the final results, as we were able to successfully create an autopilot for a UAV in a simulated environment. Before the system is ready to perform in a real environment, tests need to be performed on data from a real flight. In the following a number of improvements to the systems will be discussed.

5.2.3.1 Navigation system

A number of improvements are possible for the Kalman filter. One thing that could be done is to improve the sensor models. Tests on real data would be needed to see if this was necessary. Some ways the models could be improved would be by including a scaling factor on the accelerometer or by modelling the airspeed sensor with a scaling factor or a bias. When the sensor models are realistic, tuning of the measurement noise covariance matrix, **Rd**, and process noise covariance matrix, **Qd**, would be needed. Two approaches could be to use innovation-based or residual-based adaptive estimation, as described in (Hajiyev et al., 2015, page 67-68). In this project, the process noise for all biases was set to the same intensity. The process noise is making up for the unmodelled parts of the processes. It would therefore make sense for the process noise to have different magnitudes for the different sensors. The system could also be expanded with a filter to estimate the orientation of the UAV. All these additions would need an extended Kalman filter (EKF), as the system will become non-linear.

Another improvement would be to expand the filter to use quaternions instead of Euler angles. This could be done relatively easy as the rotation matrix could be expressed by quaternions instead of Euler angles.

5.2.3.2 Numerical issues

The main numerical issues are concerning the symmetry and positive definiteness of the covariance matrix, $\mathbf{Q}(k)$. The issues with symmetry are easy to correct as the matrix can be resymmetrized as (Farrell, 2008, page 196) (Gustafsson, 2012, page 181):

$$\mathbf{Q} = \frac{1}{2}(\mathbf{Q} + \mathbf{Q}^T) \tag{5.4}$$

The covariance matrix is by definition positive semi-definite (Farrell, 2008, page 197). It is therefore problematic if the covariance matrix loses its positive semi-definiteness during the calculations. If this happens, a solution is to factorize $\mathbf{Q}$ as:

$$\mathbf{Q} = \mathbf{U}\mathbf{D}\mathbf{U}^T \tag{5.5}$$

Another algorithm, known as a square root implementation of the Kalman filter is then necessary (Farrell, 2008, page 197). Alternatively, the above factorization of $\mathbf{Q}$ can be done at each iteration and negative singular values in $\mathbf{D}$ can be replaced with zero or small positive numbers (Gustafsson, 2012, page 181). Finally the Joseph form of the Kalman filter measurement update could be performed, as it is always symmetric (Farrell, 2008, page 187).

However, these problems are normally due to a lack of precision in the implementation (Farrell, 2008, page 196). In this project, the Kalman filter was implemented with floating point arithmetic. This gives a high precision and the above issues were not present for the short time the simulation was running. However, if the system is to operate for long durations of time, these issues would have to be addressed

5.2.3.3 Guidance system

The main improvement that could be done for the guidance system would be to extend the Dubins airplane algorithm, so it would use a Dubins path in the vertical plane as well, as the current approach is just to use a linear path. Furthermore, it would be desirable to calculate the corresponding vector fields along the z-axis instead of just using a set-point.

A future extension would be to implement an algorithm that would have constrains on the time and speed as well, thus it could reach a waypoint at a particular time and speed. The guidance system would then be realised by trajectory tracking instead of path following.

Chapter 6

Conclusion

This report covered the theory and implementation of the navigation and guidance system for an autopilot for a small fixed-wind unmanned aircraft. The navigation system consisted of a 19 state time-variant Kalman filter that was able to estimate the position, velocity, wind speed and various sensor biases of a UAV in a simulated environment. All states could be estimated successfully except the bias of the barometer and the three biases for the GPS position, as they did not converge to the true values. The reasons for this were discussed and will need further investigation. However, these issues did not seem to have a noticeable effect on the estimates of the states. The guidance system was implemented as a Dubins path with vector fields that generated the setpoints for the low level controllers in order to make the UAV follow the desired path. Simulations of the full system showed that the combination of the navigation and guidance systems were able to make the UAV fly between several waypoints autonomously even during a 30 s GPS dropout.
The purpose of the project was therefore fulfilled by the implementation of the navigation and guidance system in simulation.

Bibliography

Randal W. Beard and Timothy W. McLain. *Handbook of Unmanned Aerial Vehicles*, chapter Implementing Dubins Airplane Paths on Fixed-wing UAVs, pages 1677–1701. Springer, 2013.

Bosch. Digital pressure sensor BMP180. `https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP180-DS000-121.pdf`, 2015.

Adrien Briod. Waypoint navigation with an mav. Technical report, École Polytechnique Fédérale de Lausanne, 2008.

Department of Mathematics, Oregon State University. Dot products and projections. `http://math.oregonstate.edu/home/programs/undergrad/CalculusQuestStudyGuides/vcalc/dotprod/dotprod.html`, 1996.

James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58:1–35, 2006. ISSN 14602431. doi: 10.1093/jxb/erm298.

L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

Gabriel Hugh Elkaim, Fidelis Adhika Pradipta Lie, and Demoz Gebre-Egziabher. Principles of Guidance, Navigation and Control of UAVs. *Handbook of Unmanned Aerial Vehicles*, pages 347–380, 2015. doi: 10.1007/978-90-481-9707-1-56.

Jay A. Farrell. *Aided Navigation: GPS with High Rate Sensors*. McGraw-Hill, 2008. ISBN 0071493298 9780071493291. doi: 10.1036/0071493298.

Salvatore Gaglione. How does a GNSS receiver estimate velocity? *Inside GNSS*, pages 38–41, 2015.

National Geographic. 5 Surprising Drone Uses (Besides Amazon Delivery). `http://news.nationalgeographic.com/news/2013/12/131202%2Ddrone%2Duav%2Duas%2Damazon%2Doctocopter%2Dbezos%2Dscience%2Daircraft%2Dunmanned%2Drobot`, December 2013. Visited: 20. June 2016.

Herbert Goldstein, Charles Poole, and John Safko. *Classical Mechanics*. Addison Wesley, 3rd edition, 2001. ISBN 0201657023. URL `papers://6d732309-80c9-4f91-90fd-4ffdb7258f93/Paper/p6006`.

Fredrik Gustafsson. *Statistical Sensor Fusion*. Studentlitteratur, 2012. ISBN 9789144077321.

Chingiz Hajiyev, Halil Ersin Soken, and Sitki Yenal Vural. *State Estimation and Control for Low-cost Unmanned Aerial Vehicles*. Springer, 2015. ISBN 978-3-319-16416-8. doi: 10.1007/978-3-319-16417-5.

Elbert Hendricks, Ole Jannerup, and Paul Haase Sørensen. *Linear Systems Control*. Springer, 2008. ISBN 978-3-540-78485-2. doi: 10.1007/978-3-540-78486-9.

Shau-shiun Jan, Demoz Gebre-Egziabher, Todd Walter, and Per Enge. Improving GPS-Based Landing System Performance using an Empirical Barometric Altimeter Confidence Bound. *IEEE Transactions on Aerospace and Electronic Systems*, 44(1):127–146, 2008.

Thom Magnusson. State estimation of uav using extended kalman filter. Master's thesis, Linkopings Universitet, 2013.

Microbridge. Air Speed Measurement Using MB-LPS ΔP Sensors. `http://www.servoflo.com/sensors-by-application/download/372/787/17`, 2009.

NASA. Bernoulli's Equation. `http://www.grc.nasa.gov/WWW/K-12/airplane/bern.html`, 2015. Visited: 2016-06-19.

Derek R. Nelson, D. Blake Barber, Timothy W. McLain, and Randal W. Beard. Vector field path following for small unmanned air vehicles. *Proceedings of the American Control Conference*, 2006.

Chris C Paige. Properties of Numerical Algorithms Related to Computing Controllability. *IEEE Transactions on Automatic Control*, AC-26(1):130–138, 1981.

Niels Kjølstad Poulsen. *Stokastisk Adaptiv Regulering: Stokastiske systemer, Identifikation og Regulering*. DTU Informatics, 2007.

Luis Serrano, Donghyun Kim, Richard B Langley, Kenji Itani, and Mami Ueno. A GPS Velocity Sensor: How Accurate Can It Be? - A First Look. *Proceedings of the 2004 National Technical Meeting of The Institute of Navigation*, pages 875–885, January 2004.

Andrei M. Shkel and Vladimir Lumelsky. Classification of the dubins set. *Robotics and Autonomous Systems*, 34(4):179–202, 2001.

John P Snyder. Map Projections: A Working Manual. *U.S. Geological Survey Professional Paper 1395*, pages 154–163, 1987. ISSN 00941689. doi: 10.2307/1774978.

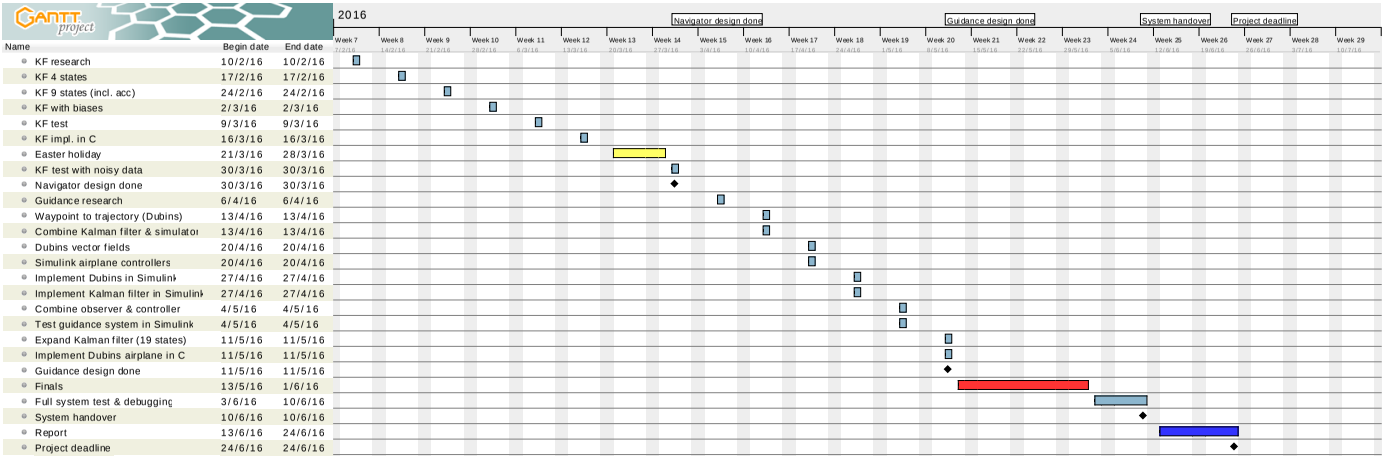
M. Wada and H. Hashimoto. Nonlinear iterative algorithm for GPS positioning with bias model. *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*, pages 684–689, 2004.

Andrew Walker. Dubins-curves: an open implementation of shortest paths for the forward only car. `https://github.com/AndrewWalker/Dubins-Curves`, 2008–.

Appendix A

Project plan

Figure A.1 shows the project plan for the assignment.



Name	Begin date	End date
KF research	10/2/16	10/2/16
KF 4 states	17/2/16	17/2/16
KF 9 states (incl. acc)	24/2/16	24/2/16
KF with biases	2/3/16	2/3/16
KF test	9/3/16	9/3/16
KF impl. in C	16/3/16	16/3/16
Easter holiday	21/3/16	28/3/16
KF test with noisy data	30/3/16	30/3/16
Navigator design done	30/3/16	30/3/16
Guidance research	6/4/16	6/4/16
Waypoint to trajectory (Dubins)	13/4/16	13/4/16
Combine Kalman filter & simulator	13/4/16	13/4/16
Dubins vector fields	20/4/16	20/4/16
Simulink airplane controllers	20/4/16	20/4/16
Implement Dubins in Simulink	27/4/16	27/4/16
Implement Kalman filter in Simulink	27/4/16	27/4/16
Combine observer & controller	4/5/16	4/5/16
Test guidance system in Simulink	4/5/16	4/5/16
Expand Kalman filter (19 states)	11/5/16	11/5/16
Implement Dubins airplane in C	11/5/16	11/5/16
Guidance design done	11/5/16	11/5/16
Finals	13/5/16	1/6/16
Full system test & debugging	3/6/16	10/6/16
System handover	10/6/16	10/6/16
Report	13/6/16	24/6/16
Project deadline	24/6/16	24/6/16

Figure A.1: Project plane

Appendix B

Project requirements

Figure B.1 shows the project requirements that was agreed upon with Danish Aviation Systems (DAS).

5. April 2016

DTU

Project agreement

Development of an autopilot for a small fixed-wing unmanned aircraft

DAS provides

DAS will provide sensor measurements in SI units. The sensor measurements will at least consist of data from a 3-axis accelerometer, 3-axis gyroscope, 3 axis magnetometer, air speed, barometer and GPS. The current attitude of the wing will be provided as an input in the form of Euler angles or quaternions. External controllers for maintaining a desired attitude and airspeed will be provided as well. The hardware specification and implementation is to be provided. This includes an airframe and all the necessary flight controller hardware for testing.

Project contribution

Navigation system: Given sensor measurements in SI units this system will estimate position and velocity in NED coordinates.

Guidance system: Given waypoints this system will calculate the desired trajectory. It will output setpoints for the desired air speed and attitude, which will be used by external controllers provided by DAS.

Success criteria

The navigation system should be able to estimate the position of the aircraft within ±2.5 m in the static case and estimate the ground speed within ±0.1 m/s in the static case. The pitch and roll should be estimated with an accuracy of ±2 degrees and yaw should be estimated within ±5 degrees.

The guidance system should be able stabilize the aircraft and make it follow a certain path within ±2 m.

Working code will be delivered in C or C++ as a software module.

Workload

The workload will be at least 10 hours a week during the period 1st of February to 12th of May and full time during the period 3rd of June to 24th of June excluding the time needed for the documentation in form of a report.

The deadline of the project is the 24th of June 2016.

Date & signature

4/5-16 4/5-16 4/5-16

Kristian Sloth Lauszus
Kristian Sloth Lauszus
Lauszus Consulting

Mads Friis Bornebusch

Steven Friberg
Danish Aviation Systems Aps

Figure B.1

Appendix C

Stochastic processes

A random variable is a variable which take values after no particular pattern. It can be described by a probability density function which specifies the probability of the variable taking certain values. The expected value or the mean of a random variable, X, is the integral or sum of all values a random value can take weighted by the probability of the variable taking this value. For a discrete random variable this is:

$$E\{X\} = \sum_{i=1}^{N} x_i p_i \tag{C.1}$$

where p_i is the probability of the random variable taking the value x_i. The variance of a random variable is given by:

$$\sigma^2 = \sum_{i=1}^{N} p_i \left(x_i - E\{X\}\right)^2 \tag{C.2}$$

The square root of the variance is called the standard deviation (Hendricks et al., 2008, page 357-361).

C.1 The Normal distribution

A common distribution for a random variable is a Gaussian or normal distribution. This distribution is given by the probability density function:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{C.3}$$

where μ is the mean and σ is the standard deviation (Farrell, 2008, page 112) (Hendricks et al., 2008, page 360). The Gaussian distribution is particularly useful due to a theorem known as the central limit theorem. It states that the distribution of a sum of n independent random variables will approach a Gaussian distribution as n approaches infinity. This result is independent of the individual distributions of the random variables in the sum. When a random variable is a sum of many random effects, which is the case in many applications, the distribution will be approximately normal making the normal distribution particularly useful (Farrell, 2008, page 117) (Hendricks et al., 2008, page 369).

C.2 Correlation and covariance

The notion of random variables can be extended to vectors of random variables. An assumption for such a vector that is frequently in this project is that the random variables in the vector are independent and identically distributed (iid). For vectors of random variables which are not independent the dependence can be quantified through the covariance and the correlation (Farrell, 2008, page 117). The covariance is given by:

$$cov(\mathbf{v}, \mathbf{w}) = E\{(\mathbf{v} - \mu_v)(\mathbf{w} - \mu_w)^T\} \tag{C.4}$$

where $\mathbf{v}$ and $\mathbf{w}$ are vectors of random variables. The correlation is given by:

$$corr(\mathbf{v}, \mathbf{w}) = E\{\mathbf{v}\mathbf{w}^T\} \tag{C.5}$$

C.3 Autocorrelation

Random or stochastic processes can be seen as a random variable indexed by the time parameter (Farrell, 2008, page 121). The covariance and correlation is also defined for a random process and so is the autocorrelation where the correlation of the function is taken with itself at different times:

$$\mathbf{R}_\mathbf{v}(t_1, t_2) = corr(\mathbf{v}(t_1), \mathbf{v}(t_2)) = E\{\mathbf{v}(t_1)\mathbf{v}(t_2)^T\} \tag{C.6}$$

There are a couple of properties which are useful for stochastic processes. A process is called stationary if the distribution is independent of time and wide sense stationary if the mean and variance of the process is independent of time. A process is called ergodic if the mean of the process over a certain time is equal to the mean at a specific time of different ensembles of the process (Farrell, 2008, page 122-123) (Hendricks et al., 2008, page 378-382).

C.4 White noise

The power spectral density (PSD) of a wide sense stationary random process is the Fourier transform of the autocorrelation function (Farrell, 2008, page 122):

$$S_v(j\omega) = \int_{-\infty}^{\infty} \mathbf{R}_{\mathbf{v}}(\tau) e^{-j\omega\tau} d\tau \tag{C.7}$$

where $\mathbf{R}_{\mathbf{v}}(\tau) = corr(\mathbf{v}(t_1), \mathbf{v}(t_2))$, ω is the frequency and j is the imaginary unit. Noise, which is a random process, is usually characterised by the frequency contents. Noise which has a flat PSD and therefore the same amount of power at all frequencies is called white noise, while noise where this is not the case is called coloured noise (Farrell, 2008, page 123) (Hendricks et al., 2008, page 400). When considering noise on sensors of which the output is integrated, another important term is a random walk which is integrated white noise. This is also called Brownian motion (Farrell, 2008, page 134).

Appendix D

System matrices

In this chapter the following short forms will be used:

$$\cos(x) \equiv \mathrm{c}x$$
$$\sin(x) \equiv \mathrm{s}x$$

D.1 Linear continuous time system matrices

$$
\mathbf{A} = \begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c\psi c\theta & c\psi s\phi s\theta - c\phi s\psi & s\phi s\psi + c\phi c\psi s\theta & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c\theta s\psi & c\phi c\psi + s\phi s\psi s\theta & c\phi s\psi s\theta - c\psi s\phi & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s\theta & c\theta s\phi & c\phi c\theta & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{D.1}
$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathrm{c}\psi\mathrm{c}\theta & \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta - \mathrm{c}\phi\mathrm{s}\psi & \mathrm{s}\phi\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi\mathrm{s}\theta & 0 \\ \mathrm{c}\theta\mathrm{s}\psi & \mathrm{c}\phi\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\psi\mathrm{s}\theta & \mathrm{c}\phi\mathrm{s}\psi\mathrm{s}\theta - \mathrm{c}\psi\mathrm{s}\phi & 0 \\ -\mathrm{s}\theta & \mathrm{c}\theta\mathrm{s}\phi & \mathrm{c}\phi\mathrm{c}\theta & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{D.2}$$

Given here is:

$$
\begin{aligned}
a_c &= \mathrm{c}\psi\mathrm{c}\theta \\
b_c &= \mathrm{c}\theta\mathrm{s}\psi \\
c_c &= -\frac{\mathrm{s}\theta}{\mathrm{c}\theta^2 + \mathrm{s}\theta^2} \\
d_c &= \mathrm{c}\psi^2\mathrm{c}\theta^2 + \mathrm{c}\psi^2\mathrm{s}\theta^2 + \mathrm{c}\theta^2\mathrm{s}\psi^2 + \mathrm{s}\psi^2\mathrm{s}\theta^2
\end{aligned}
\tag{D.3}
$$

$$
\mathbf{C} = \left[\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & c_c \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{b_c}{d_c} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{a_c}{d_c} \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & c_c \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \frac{b_c}{d_c} \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \frac{a_c}{d_c} \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right]
$$

D.2 Linear discrete time system matrices

$$
\mathbf{F} = \begin{bmatrix}
1 & 0 & 0 & T_s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{{T_s}^2 \mathrm{c}\psi\mathrm{c}\theta}{2} & \frac{{T_s}^2 \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta}{2} - \frac{{T_s}^2 \mathrm{c}\phi\mathrm{s}\psi}{2} & \frac{{T_s}^2 \mathrm{c}\phi\mathrm{c}\psi\mathrm{s}\theta}{2} + \frac{{T_s}^2 \mathrm{s}\phi\mathrm{s}\psi}{2} & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & T_s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{{T_s}^2 \mathrm{c}\phi\mathrm{c}\psi}{2} + \frac{{T_s}^2 \mathrm{s}\phi\mathrm{s}\psi\mathrm{s}\theta}{2} & \frac{{T_s}^2 \mathrm{c}\phi\mathrm{s}\psi\mathrm{s}\theta}{2} - \frac{{T_s}^2 \mathrm{c}\psi\mathrm{s}\phi}{2} & \frac{{T_s}^2 \mathrm{c}\phi\mathrm{s}\psi}{2} - \frac{{T_s}^2 \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta}{2} & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & T_s & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{{T_s}^2 \mathrm{s}\theta}{2} & \frac{{T_s}^2 \mathrm{c}\theta\mathrm{s}\phi}{2} & \frac{{T_s}^2 \mathrm{c}\phi\mathrm{c}\theta}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & T_s & 0 & 0 & 0 & 0 & 0 & T_s \mathrm{c}\psi\mathrm{c}\theta & T_s \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta - T_s \mathrm{c}\phi\mathrm{s}\psi & T_s \mathrm{s}\phi\mathrm{s}\psi + T_s \mathrm{c}\phi\mathrm{c}\psi\mathrm{s}\theta & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & T_s & 0 & 0 & 0 & 0 & T_s \mathrm{c}\phi\mathrm{c}\psi + T_s \mathrm{s}\phi\mathrm{s}\psi\mathrm{s}\theta & T_s \mathrm{c}\phi\mathrm{s}\psi\mathrm{s}\theta - T_s \mathrm{c}\psi\mathrm{s}\phi & T_s \mathrm{c}\phi\mathrm{s}\theta\mathrm{s}\psi - T_s \mathrm{c}\psi\mathrm{s}\phi & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T_s & 0 & 0 & 0 & T_s \mathrm{c}\theta\mathrm{s}\phi & T_s \mathrm{c}\phi\mathrm{c}\theta & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T_s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T_s & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & T_s & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1
\end{bmatrix} \tag{D.4}
$$

$$\mathbf{G} = \begin{bmatrix} \frac{{T_s}^2 \mathrm{c}\psi\mathrm{c}\theta}{2} & -\frac{{T_s}^2 \mathrm{c}\phi\mathrm{s}\psi - \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta}{2} & \frac{{T_s}^2 \mathrm{s}\phi\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi\mathrm{s}\theta}{2} & 0 \\ \frac{{T_s}^2 \mathrm{c}\theta\mathrm{s}\psi}{2} & \frac{{T_s}^2 \mathrm{c}\phi\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\psi\mathrm{s}\theta}{2} & -\frac{{T_s}^2 \mathrm{c}\psi\mathrm{s}\phi - \mathrm{c}\phi\mathrm{s}\psi\mathrm{s}\theta}{2} & 0 \\ -\frac{{T_s}^2 \mathrm{s}\theta}{2} & \frac{{T_s}^2 \mathrm{c}\theta\mathrm{s}\phi}{2} & \frac{{T_s}^2 \mathrm{c}\phi\mathrm{c}\theta}{2} & -\frac{{T_s}^2}{2} \\ T_s \mathrm{c}\psi\mathrm{c}\theta & T_s \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta - T_s \mathrm{c}\phi\mathrm{s}\psi & T_s \mathrm{s}\phi\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi\mathrm{s}\theta & 0 \\ T_s \mathrm{c}\theta\mathrm{s}\psi & T_s \mathrm{c}\phi\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\psi\mathrm{s}\theta & T_s \mathrm{c}\phi\mathrm{s}\psi\mathrm{s}\theta - T_s \mathrm{c}\psi\mathrm{s}\phi & 0 \\ -T_s \mathrm{s}\theta & T_s \mathrm{c}\theta\mathrm{s}\phi & T_s \mathrm{c}\phi\mathrm{c}\theta & -T_s \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{D.5}$$

D.3 Noise matrices

The short form $k_{T\sigma} = T_s {\sigma_{bias}}^2$ is used.

$$\Phi d = \begin{bmatrix}
{T_s\sigma_P}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & {T_s\sigma_P}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & {T_s\sigma_P}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & Qd_{44} & Qd_{45} & Qd_{46} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & Qd_{45} & Qd_{55} & Qd_{56} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & Qd_{46} & Qd_{56} & Qd_{66} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & {T_s\sigma_W}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & {T_s\sigma_W}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & {T_s\sigma_W}^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{T\sigma}
\end{bmatrix} \tag{D.6}$$

Where:

$$
\begin{aligned}
Qd_{44} &= T_s(\sigma_a{}^2(\mathrm{s}\phi\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi\mathrm{s}\theta)^2 + \sigma_a{}^2(\mathrm{c}\phi\mathrm{s}\psi - \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta)^2 + \sigma_a{}^2\mathrm{c}\psi^2\mathrm{c}\theta^2)\\
Qd_{45} &= -T_s(\sigma_a{}^2(\mathrm{c}\phi\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\psi\mathrm{s}\theta)(\mathrm{c}\phi\mathrm{s}\psi - \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta) + \sigma_a{}^2(\mathrm{s}\phi\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi\mathrm{s}\theta)(\mathrm{c}\psi\mathrm{s}\phi - \mathrm{c}\phi\mathrm{s}\psi\mathrm{s}\theta) - \sigma_a{}^2\mathrm{c}\psi\mathrm{c}\theta^2\mathrm{s}\psi)\\
Qd_{46} &= -T_s(\sigma_a{}^2\mathrm{c}\theta\mathrm{s}\phi(\mathrm{c}\phi\mathrm{s}\psi - \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta) - \sigma_a{}^2\mathrm{c}\phi\mathrm{c}\theta(\mathrm{s}\phi\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi\mathrm{s}\theta) + \sigma_a{}^2\mathrm{c}\psi\mathrm{c}\theta\mathrm{s}\theta)\\
Qd_{54} &= -T_s(\sigma_a{}^2(\mathrm{c}\phi\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\psi\mathrm{s}\theta)(\mathrm{c}\phi\mathrm{s}\psi - \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta) + \sigma_a{}^2(\mathrm{s}\phi\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi\mathrm{s}\theta)(\mathrm{c}\psi\mathrm{s}\phi - \mathrm{c}\phi\mathrm{s}\psi\mathrm{s}\theta) - \sigma_a{}^2\mathrm{c}\psi\mathrm{c}\theta^2\mathrm{s}\psi)\\
Qd_{55} &= T_s(\sigma_a{}^2(\mathrm{c}\phi\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\psi\mathrm{s}\theta)^2 + \sigma_a{}^2(\mathrm{c}\psi\mathrm{s}\phi - \mathrm{c}\phi\mathrm{s}\psi\mathrm{s}\theta)^2 + \sigma_a{}^2\mathrm{c}\theta^2\mathrm{s}\psi^2)\\
Qd_{56} &= -T_s(\sigma_a{}^2\mathrm{c}\phi\mathrm{c}\theta(\mathrm{c}\psi\mathrm{s}\phi - \mathrm{c}\phi\mathrm{s}\psi\mathrm{s}\theta) - \sigma_a{}^2\mathrm{c}\theta\mathrm{s}\phi(\mathrm{c}\phi\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\psi\mathrm{s}\theta) + \sigma_a{}^2\mathrm{c}\theta\mathrm{s}\psi\mathrm{s}\theta)\\
Qd_{64} &= -T_s(\sigma_a{}^2\mathrm{c}\theta\mathrm{s}\phi(\mathrm{c}\phi\mathrm{s}\psi - \mathrm{c}\psi\mathrm{s}\phi\mathrm{s}\theta) - \sigma_a{}^2\mathrm{c}\phi\mathrm{c}\theta(\mathrm{s}\phi\mathrm{s}\psi + \mathrm{c}\phi\mathrm{c}\psi\mathrm{s}\theta) + \sigma_a{}^2\mathrm{c}\psi\mathrm{c}\theta\mathrm{s}\theta)\\
Qd_{65} &= -T_s(\sigma_a{}^2\mathrm{c}\phi\mathrm{c}\theta(\mathrm{c}\psi\mathrm{s}\phi - \mathrm{c}\phi\mathrm{s}\psi\mathrm{s}\theta) - \sigma_a{}^2\mathrm{c}\theta\mathrm{s}\phi(\mathrm{c}\phi\mathrm{c}\psi + \mathrm{s}\phi\mathrm{s}\psi\mathrm{s}\theta) + \sigma_a{}^2\mathrm{c}\theta\mathrm{s}\psi\mathrm{s}\theta)\\
Qd_{66} &= T_s(\sigma_a{}^2\mathrm{c}\phi^2\mathrm{c}\theta^2 + \sigma_a{}^2\mathrm{c}\theta^2\mathrm{s}\phi^2 + \sigma_a{}^2\mathrm{s}\theta^2)
\end{aligned}
\tag{D.7}
$$

$$\mathbf{Rd} = \begin{bmatrix} \frac{{\sigma_{GPS}}^2}{T_{GPS}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{{\sigma_{GPS}}^2}{T_{GPS}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{{\sigma_{GPS}}^2}{T_{GPS}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{{\sigma_V}^2}{T_{GPS}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{{\sigma_V}^2}{T_{GPS}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{{\sigma_V}^2}{T_{GPS}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{{\sigma_{baro}}^2}{T_{meas}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{{\sigma_{air}}^2}{T_{meas}} \end{bmatrix} \tag{D.8}$$

Appendix E

Dubins path segment lengths

This is a summary of the results from (Shkel and Lumelsky, 2001) where the length of each type of segment is calculated directly.

Lets first introduce the normalized distance d with respect to the turning radius ρ between the two waypoints w_1 and w_2:

$$d = \frac{|w_2 - w_1|}{\rho} \tag{E.1}$$

The angle at the first waypoint α and second waypoint β is given by:

$$\theta = \text{atan2}(w_{2_y} - w_{1_y}, w_{2_x} - w_{1_x}) \mod 2\pi \tag{E.2}$$
$$\alpha = w_{1_\psi} - \theta \mod 2\pi \tag{E.3}$$
$$\beta = w_{2_\psi} - \theta \mod 2\pi \tag{E.4}$$

The length of each type of segment can then be calculated accordingly:

***LSL* Dubins path:**

$$\begin{aligned} t_{lsl} &= -\alpha + \arctan\frac{\cos\beta - \cos\alpha}{d + \sin\alpha - \sin\beta} \mod 2\pi \\ p_{lsl} &= \sqrt{2 + d^2 - 2\cos(\alpha-\beta) + 2d(\sin\alpha - \sin\beta)} \\ q_{lsl} &= \beta - \arctan\frac{\cos\beta - \cos\alpha}{d + \sin\alpha - \sin\beta} \mod 2\pi \end{aligned} \tag{E.5}$$

***RSR* Dubins path:**

$$\begin{aligned} t_{rsr} &= \alpha - \arctan\frac{\cos\alpha - \cos\beta}{d - \sin\alpha + \sin\beta} \mod 2\pi \\ p_{rsr} &= \sqrt{2 + d^2 - 2\cos(\alpha-\beta) + 2d(\sin\beta - \sin\alpha)} \\ q_{rsr} &= -\beta + \arctan\frac{\cos\alpha - \cos\beta}{d - \sin\alpha + \sin\beta} \mod 2\pi \end{aligned} \tag{E.6}$$

***LSR* Dubins path:**

$$\begin{aligned} t_{lsr} &= -\alpha + \arctan\frac{-\cos\alpha - \cos\beta}{d + \sin\alpha + \sin\beta} - \arctan\frac{-2}{p_{lsr}} \mod 2\pi \\ p_{lsr} &= \sqrt{-2 + d^2 + 2\cos(\alpha-\beta) + 2d(\sin\alpha + \sin\beta)} \\ q_{lsr} &= -\beta + \arctan\frac{-\cos\alpha - \cos\beta}{d + \sin\alpha + \sin\beta} - \arctan\frac{-2}{p_{lsr}} \mod 2\pi \end{aligned} \tag{E.7}$$

***RSL* Dubins path:**

$$\begin{aligned} t_{rsl} &= \alpha - \arctan\frac{\cos\alpha + \cos\beta}{d - \sin\alpha - \sin\beta} + \arctan\frac{2}{p_{rsl}} \mod 2\pi \\ p_{rsl} &= \sqrt{d^2 - 2 + 2\cos(\alpha-\beta) - 2d(\sin\alpha + \sin\beta)} \\ q_{rsl} &= \beta - \arctan\frac{\cos\alpha + \cos\beta}{d - \sin\alpha - \sin\beta} + \arctan\frac{2}{p_{rsl}} \mod 2\pi \end{aligned} \tag{E.8}$$

***RLR* Dubins path:**

$$\begin{aligned} t_{rlr} &= \alpha - \arctan\frac{\cos\alpha - \cos\beta}{d - \sin\alpha + \sin\beta} + \frac{p_{rlr}}{2} \mod 2\pi \\ p_{rlr} &= \arccos\frac{1}{8}\left(6 - d^2 + 2\cos(\alpha-\beta) + 2d(\sin\alpha - \sin\beta)\right) \mod 2\pi \\ q_{rlr} &= \alpha - \beta - t_{rlr} + p_{rlr} \mod 2\pi \end{aligned} \tag{E.9}$$

***LRL* Dubins path:**

$$
\begin{aligned}
t_{lrl} &= -\alpha + \arctan\frac{-\cos\alpha + \cos\alpha}{d + \sin\alpha - \sin\beta} + \frac{p_{lrl}}{2} \mod 2\pi \\
p_{lrl} &= \arccos\frac{1}{8}\left(6 - d^2 + 2\cos(\alpha-\beta) + 2d(\sin\alpha - \sin\beta)\right) \mod 2\pi \\
q_{lrl} &= \beta - \alpha + 2p_{lrl} \mod 2\pi
\end{aligned}
\tag{E.10}
$$

Note the equations above calculates the normalised segment lengths with respect to the turning radius ρ, thus in order to get the true length of each segment we will have to multiply the segment lengths by the turning radius ρ:

$$t = \rho\, t_{nom} \tag{E.11}$$

$$p = \rho\, p_{nom} \tag{E.12}$$

$$q = \rho\, q_{nom} \tag{E.13}$$

The final Dubins path length is then given by:

$$\mathcal{L} = t + p + q \tag{E.14}$$

Notice that it will be beneficial to substitute all arctan function with the four quadrant inverse tangent function atan2 when implementing the equations above.

Appendix F

Speed, heading and altitude controllers

The controllers used for the simulation is based on the work by Elkaim et al. (2015). An overview of the controllers can be seen in Figure F.1 where the desired heading angle ψ_d, desired z-position z_d (denoted $PosD_D$ in the simulation) and desired airspeed V_{air_d} are used as set-points for the controllers. The saturation blocks are used to limit the output from the controllers to the valid input values for the aileron, throttle and elevator.

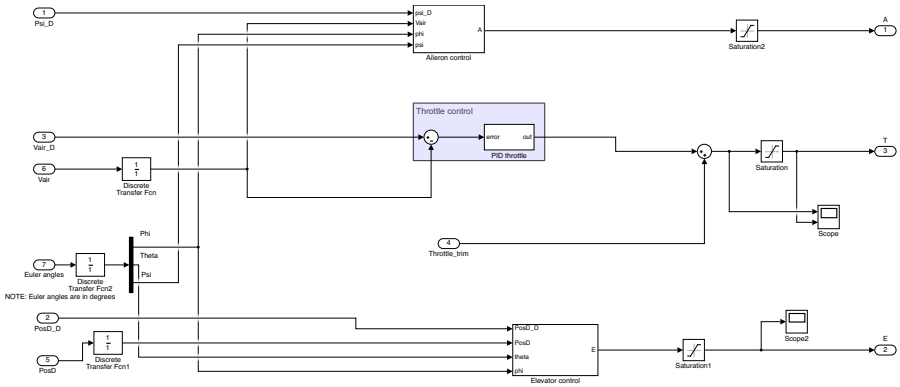


Figure F.1: Controller overview

Figure F.2 shows an overview of the aileron controller. The first block shown in Figure F.3 converts the heading error to a desired heading turning rate $\dot{\psi}_d$. The heading turning rate is the converted to the desired roll angle according to the following equation:

$$\phi_d = \text{atan2}\left(\dot{\psi}_d V_{air}, g_0\right) \tag{F.1}$$

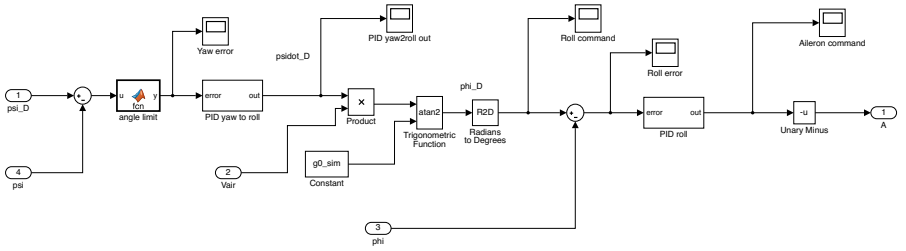


Figure F.2: Aileron controller overview

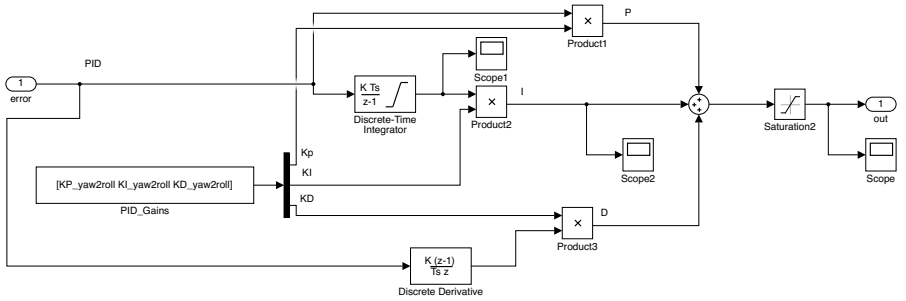


Figure F.3: Yaw to roll controller

Finally a PID-controller, shown in Figure F.4, is used as the output to the aileron input.

The final PID-values used for the aileron controllers were:

$$\begin{aligned} Kp_{yaw2roll} &= 0.0049 \\ Ki_{yaw2roll} &= 0.0035 \\ Kd_{yaw2roll} &= 5.5851 \times 10^{-4} \\ Kp_{roll} &= 0.0349 \\ Ki_{roll} &= 0.0873 \\ Kd_{roll} &= 0 \end{aligned} \tag{F.2}$$

Even though the Aerosonde UAV has a rudder input, we decided not to use it, as the system is later to be implemented on a flying wing, which does not have a rudder control surface.

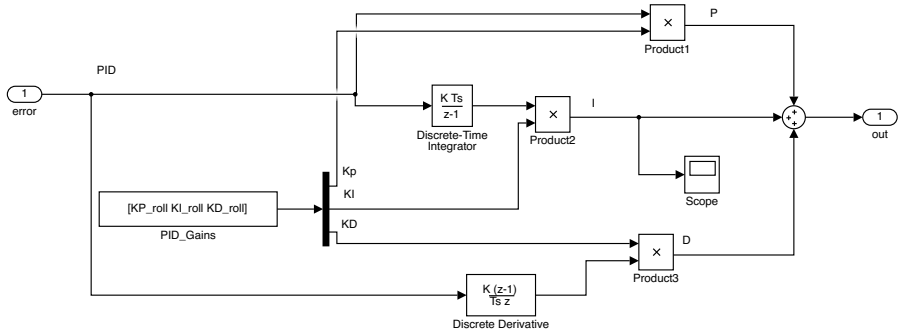


Figure F.4: Roll controller

An overview of the elevator controller can be seen in Figure F.5. The bottom one is a feed forward controller that is used to prevent the UAV from loosing altitude when turning. The feed forward term is a simple P-controller given by:

$$Kp\left(\frac{1}{\cos\phi}-1\right) \tag{F.3}$$

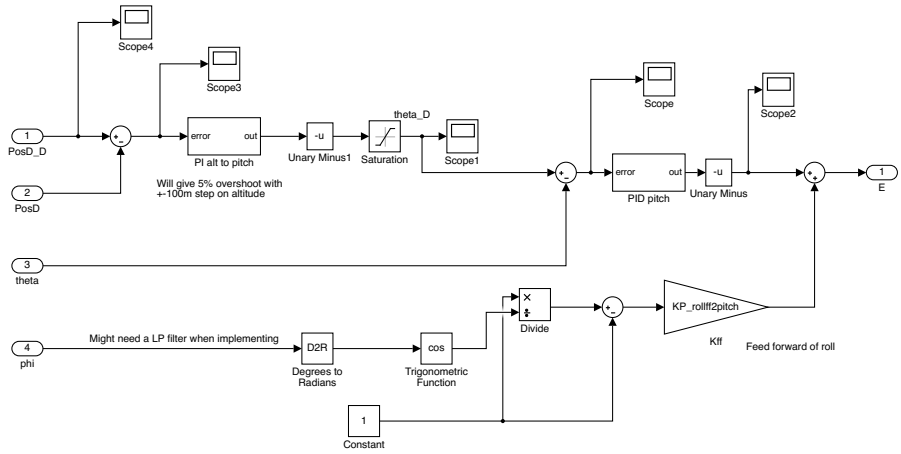


Figure F.5: Elevator controller overview

Figure F.6 shows the controller that converts the error in altitude to a desired pitch angle θ_d. The error in pitch angle is then used as the input to a second

controller, shown in Figure F.7. The output from this controller is added to the output from the feed forward controller. The combined output is then used as input to the elevator.

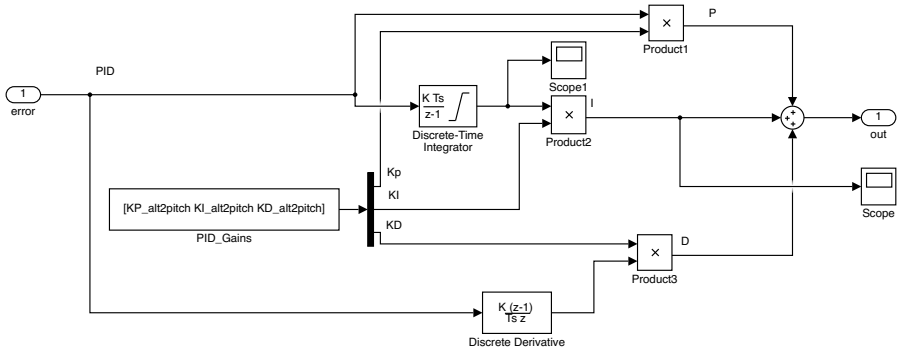


Figure F.6: Altitude to pitch controller

The final PID-values used for the simulation were:

$$\begin{aligned} Kp_{rollff2pitch} &= -0.5000 \\ Kp_{alt2pitch} &= 1 \\ Ki_{alt2pitch} &= 0.1800 \\ Kd_{alt2pitch} &= 0 \\ Kp_{pitch} &= 0.0087 \\ Ki_{pitch} &= 0.0087 \\ Kd_{pitch} &= 0.0014 \end{aligned} \tag{F.4}$$

Figure F.8 shows the throttle controller. This controller is used to keep a constant airspeed doing the flight. The output from this controller is added to a constant $Throttle_{trim}$, which is the trimmed throttle value needed for level flight.

The final PID-values used for the throttle controller were:

$$\begin{aligned} Kp_{thr} &= 0.8000 \\ Ki_{thr} &= 0.1000 \\ Kd_{thr} &= 0.0320 \end{aligned} \tag{F.5}$$

The output of all integrators were limited in order to acts as a simple anti-windup mechanism. For instance this was needed when doing helices, as the

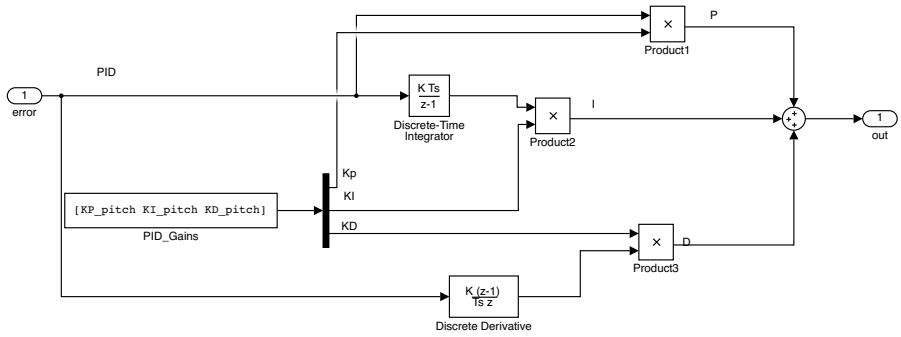


Figure F.7: Pitch controller

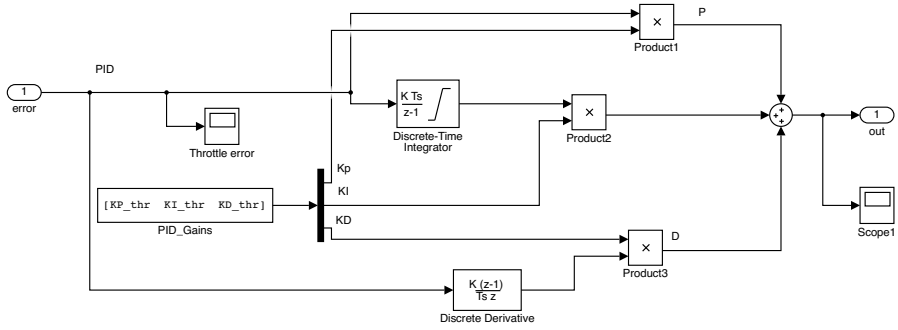


Figure F.8: Throttle controller

time for it to reach the desired z-position could take quite some time, which would cause a huge overshoot due to the integral term, if they were not limited.

Appendix G

Exporting the Kalman filter from Matlab to C++

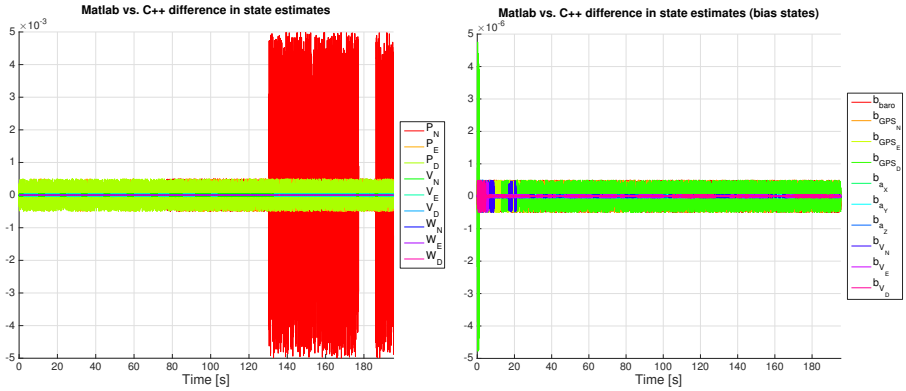
The matrices used for the Kalman filter were first solved analytically and implemented in Matlab. Since the designed Kalman filter is time-varying the system model matrices needs to be updated in each loop. However the matrix assignments can easily be exported to C code using the Matlab function *ccode*. The code were then further refined using a custom bash script. Thus the tedious process of writing down these terms can be done automatically, this is especially useful when the order of the system is large, as it becomes almost impossible not to make any mistakes when writing down the equations manually in C or C++. The vector and matrix operations in the C++ code for the Kalman time update and measurement update steps were done using the Eigen library[1]. The entire code was then put into a C++ class. This procedure allowed us to automate the process of solving the system model matrices for the Kalman filter and export it to C++ code. This made is easy to expand the model during development and include more states in the system.

By comparing the differences between the estimated states from the Matlab and C++ implementation one can determine if the conversion from Matlab to C++ is correct.

[1] `http://eigen.tuxfamily.org`

Figure G.1 shows a plot of the difference between the state estimates for the Matlab and C++ implementation of the Kalman filter for the first simulation presented in section 5.1. As it can be seen the difference between the two is very small and is possible just due to differences in the implementation of matrix operators and precision of the floating-point numbers used for the two systems. Even the oscillations shown in Figure G.1a is only about 5 mm which is much smaller than the precision of the GPS anyway.

All the other simulations had similar plots as well, thus the implementation in Matlab and C++ were assumed to be the same.



(a) **(b)**

Figure G.1: (a) and (b) shows the difference between the state estimates for the Matlab and C++ implementation

Appendix H

Simulink model

Figure H.1 shows an overview of the Simulink model used in this report. The Aerosonde UAV block provided by the Aerosom Blockset[1] is a blue block seen in the upper left corner. The output from this block is then run through the "Noise and bias system" block (middle right) that adds noise and biases on the measurements. The measurements are then discretized and run through the designed Kalman filter (bottom right). The estimated position of the UAV is then used as the input for the Dubins path algorithm which outputs the desired heading and the desired z-position (bottom left). The set-points and estimated states are then used as the input to the controller block (middle left, just below the Aerosonde UAV block) which stabilises the system. All yellow and turquoise blocks are showing various values and are used for debugging. Orange blocks are scopes which are also used for debugging. Pink blocks are constants set in an external file.

[1] `http://www.u-dynamics.com/aerosim/default.htm`

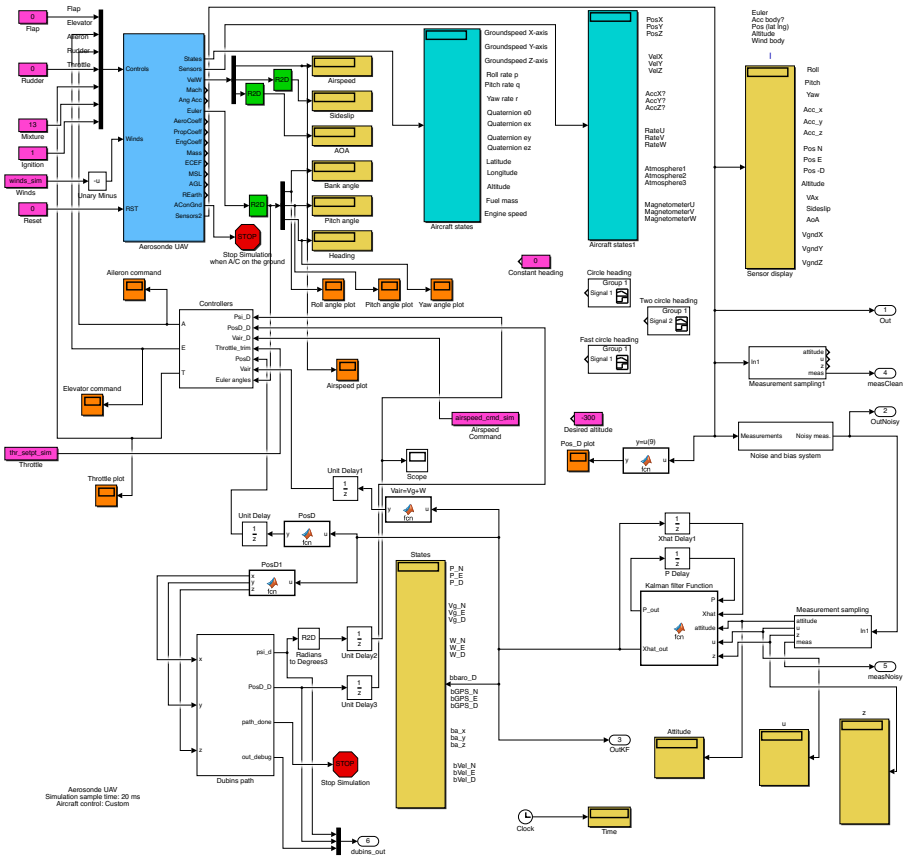


Figure H.1: Overview of the top-level Simulink model

DTU Space
National Space Institute
Technical University of Denmark

Elektrovej, building 327
DK - 2800 Kgs. Lyngby
Tel (+45) 4525 9500
Fax (+45) 4525 9575

www.space.dtu.dk