Nonlinear Model Predictive Path-Following Control for fixed-wing Unmanned Aerial Vehicles

Thomas Leirfall

A thesis presented for the degree of
Master of Science

Supervisors:
Professor Tor Arne Johansen, Dept. of Eng. Cybernetics, NTNU
PhD fellow Dirk Reinhardt, Dept. of Eng. Cybernetics, NTNU

Department of Engineering Cybernetics
Norwegian University of Science and Technology
Norway
June 2021

Abstract

This master's thesis studies the path following problem. The path-following problem refers to steering a vehicle and then keeping it close to a predefined geometric curve in the Euclidean space. In contrast to trajectory tracking, the position along the path is a degree of freedom such that the magnitude of the velocity vector is in most cases controlled independently. Nonlinear model predictive controller (NMPC) can be used to follow geometrically challenging curves, and at the same time performing optimally with respect to user-defined cost function and constraints.

This thesis builds on the specialization project[1] which explored a straight-line path problem with two different approaches. The first was a vector field-based path-following algorithm for controlling the course and height in a successive loop closure. And an NMPC with a simplified kinematic model. The kinematic model will be further developed in this thesis and include a low-level autopilot in the inner-loop. There will also be a complete dynamic model of the unmanned aerial vehicle (UAV) X8 Skywalker in the NMPC.

Will explored two ways to parameterize the path where the path is a straight line and a curved path in the Euclidean space $\mathcal{R}^3$. Further, there will be a simulation study and comparing the simplified kinematic and dynamic model NMPC to geometric controllers vector field-based (VFB) for straight-line path and nonlinear differential geometric path-following (NDGPFG) for the curved path.

A simulation of the results are seen at https://www.youtube.com/watch?v=SYCMKUfa-mk.

Preface

This thesis marks the end of a 2-year master's degree in cybernetics and robotics at NTNU Trondheim. The project assignment was specified and written by Tor Arne Johansen and Dirk Reinhardt at the Department of Engineering Cybernetics NTNU.

I will use theory and methods on optimization, airplane kinematics, and dynamics and simulations to develop controllers for fixed-wing aerial vehicles in my thesis on subjects I have acquired during the cybernetics and robotics courses at NTNU Trondheim.

I want to thank my supervisors Tor Arne Johansen and Dirk Reinhardt. Especially Dirk for weekly feedback and to guide me to the path on theory and implementations of controllers. Also, to all the contributors on the repository UAVlab.

Table of contents

1 Introduction

The applications for uses of drones and UAVs, both for private and professional service, are growing. In the summer of 2020, Equinor completed the world's first logistics operation with a drone to an offshore installation [2]. During the terrible quick clay landslide in Gjerdrum around New Year's Eve in 2020, drones were used for surveillance and to find people. Even though both these examples are multirotor drones, this shows the scope of unmanned Aircraft System (UAS) and fixed-wing UAVs.

In this section, there will be a small overview of existing controllers that could solve the path following problem. For path following, time is neglected, and the objective is to converge to the path and follow. This makes it different from trajectory following which is time-dependent on the path. Looking at a UAV under the influence of wind, the magnitude of the gust wind will be significant to the speed of a small UAV. Consider this, the path following formulation is preferred, where speed can be controlled separately.

The path following problems is solved in the literature with different controllers and approaches. Here are some examples.

As the name proposes, geometric controllers look at the geometric to define the differential equations where the states are global. In [3] a controller make the UAV follow a nominal path using a control algorithm that uses a Special Orthogonal group for the formulation of the attitude control problem to avoid the singularity.

Lyapunov based design methods use a control-Lyapunov function (CLF) to design the control stable. Lyapunov stability is a well know tool for engineers to design system, and the reader can look into [4] for more. In [5] a CLF is proposed to select control input which is feasible with respect to the CLF in a trajectory tracking problem.

Backstepping technique is used to remove unwanted nonlinear terms, making the system unstable when checking for Lyapunov stability. In [6] a robust recursive design technique is presented when looking at the nonlinearity.

Virtual target is, as the name says, a targeting approach. In [7] a Serret-Frenet frame is defined as the target, where the error dynamics is the error between target and UAV and the Lyapunov-stability is proven.

Simply, a vector field can be described as a plane where there are vectors at each point in the field with different directions and magnitude. Vector field-based controllers use this principle and guide the UAV to the path using the direction and magnitude of the vectors. Example are seen in [8] and [9]. Later in this thesis, the vector field-based controller from [9] is used for comparison. In [10] is a survey of different guidance techniques for fixed-wing UAV.

Guidance controllers can not solve for optimality on converging to the path, introducing the optimal controller model predictive controller (MPC). Here the controller solves the optimal control problem (OCP) for a time horizon and only uses the first input and discharges the rest. Then the optimal trajectory to the path appears as a solution to the OCP. The benefit about the MPC is that the input and states are explicitly constrained. Meaning that can directly control the use of actuators through constraints.

Using the dynamic model of the UAV requires theoretical and observing the parameters of the UAV. This process can be time-consuming and challenging because of experimental studies (for example, wind tunnels). Therefore, a common solution is to use a simplified parameterless and parameterised (explored later in this thesis) kinematic model in the MPC solving the optimization problem in the outer loop, with an autopilot in the inner loop which tracks the high-level commands.

The inner-loop of [11] and [12] is based on the coordinated turn [9]. Here the MPC solves the optimal problem and sends the Euler angles reference to the lower-level autopilot to track. The optimal problem for [11] in the MPC is position and course angle error.

In [13] a virtual state controlling the behavior on the path is introduced in an augmented system. The path parameterization can be readily be changed. This approach is also implemented in [14]. Other methods in the inner-loop are proposed in [15] a backstepping in the longitudinal and integral-LQR in the lateral direction.

In [16] an auxiliary control law is proposed with an arbitrary small tube for which the UAV should be within. Then the position error is used in OCP. The line-of-Sight controller uses a lookahead distance to calculate the desired course angle for which the vehicle converges to the path. Adding an integral, integral line-of-sight (ILOS), removes the steady-state offsets. In [17] the lookahead Δ is a decision variable in the OCP to give an optimal converge. In [18] a virtual target approach is used with a MPC where the cost of the OCP is the error between virtual target vehicle and the UAV.

Using MPC requires relatively fast computation time. Because at every time step, the OCP is solved for an N time horizontal steps. For advanced models, this could be a time issue. With the development of faster solvers, the use of MPC is more available. An example of available solvers is ACADOS [19] and YALMIP.

There are different ways to defined the path. A straight-line path is a line between two points. With more points, could use path planning with a Dubins path for which the most effective route is found [20]. In [21] collision and obstacle avoidance, where an obstacle is avoided on a straight line. The path could also be curved, with a nonlinear line between two or more points. Bézier curves or B-spline are examples of creating a curved path. In [22] a dynamic path planning with B-spline is proposed.

1.1 Project plan

In this thesis, the following plan is defined

1. Identify a kinematic model with a low-level autopilot in the inner loop.
2. Extend the kinematic model with a dynamic model and implement a path-following NMPC using the full model.
3. Implement a mechanism to initialize the path variable and for switching between path segments.
4. Implement an additional path parameterization and test them with the developed controllers.
5. Conduct a simulation study and compare the performance of the NMPC based on the approximated kinematic model and the full dynamic model on the implemented paths.
6. Further compare both NMPCs to the vector field-based controllers implemented in the specialization project[1].

1.2 Limitations

This thesis is limited to only computer simulation where full feedback is expected (no estimation). Program libraries for aircraft dynamics and PID controler are used. ACADOs is used to solve the NMPC.

1.3 Structure of the thesis

The thesis is divided into 7 chapters. Starting in chapter 3 with a focus on theory and methods that are important for the concept. Then in chapter 4, a control algorithm design is presented, with methods of the controllers. In chapter 5 the result is presented with a discussion on how the controllers performed with a conclusion in chapter 6. The solutions used are presented in Attachments.

2 Notation and abbreviations

The use of boldface symbols for vectors and matrices, where vectors are lowercase and matrices are uppercase letter.

Acronyms

$\mathcal{F}^i$	NED frame
$\mathcal{F}^v$	Vehicle frame
$\mathcal{F}^{v1}$	Vehicle-1 frame
$\mathcal{F}^{v2}$	Vehicle-2 frame
$\mathcal{F}^b$	Body frame
$\mathcal{F}^s$	Stability frame
$\mathcal{F}^w$	Wind frame
ϕ	Roll
θ	Pitch
ψ	Yaw
Θ	Euler angles $[\phi \quad \theta \quad \psi]^T$
α	Angle of attack
β	Side-slip
$S(\cdot)$	Skew symmetric matrix
δ_a	Aileron
δ_e	Elevation
δ_r	Rudder
δ_t	Throttle
$Q \succeq 0$	Positive definite matrix Q
$\|x\|^2$	2-norm of vector
$\|x\|_Q^2$	x^TQx
n_x	Number of state
n_u	Number of input
n_y	Number of output
$z_{\{i\}}$	Virtual state
v	Virtual input
λ	Path parameter
$\mathcal{W}$	Sequence of waypoints
$\mathbf{w}_{\{i\}}$	Waypoint index i

3 Theory

In this section, some basic theory about UAVs and path following is presented. A theoretical description in section 3.1 presents the kinematic and dynamic differential equations with frames and angles. Included in this are the actuators and the forces and moments. The definition of the path following problem is in section 3.2. Theory of control method including successive loop closure in section 3.3, optimizing in section 3.4 which is the basics for the MPC seen in section 3.5.

3.1 Kinematics and dynamics

Kinematics is looking at the motion of the UAV without the forces, While dynamics includes the forces when looking at the motion. In this subsection the kinematic and dynamic equations will be explained. To do this, fist look at the frames where the states are express in and the angles between the frames. These angles and the position builds the kinematic equations. Then forces and moments together with the actuators gives the dynamic equations.

The orientation of the UAV with respect to the world is expressed in frames. The important frames, in this thesis, are the north east down (NED)-, body- and stability- frame, see figure 1 to 4.

NED
In NED x pointing to the north, y east, and z down. The positive direction for z is down. This frame is on the curve of the earth and is a tangential frame. Assume that this frame is an inertial frame, which is where the forces occur. The frame is denoted $\mathcal{F}^i$. Also denote i, j and k to be x-,y- and z-axis, respectively.

Vehicle
Located at the center of UAV, and does not rotate, which means that this has the same orientation as the NED-frame. This frame is seen in figure 1.

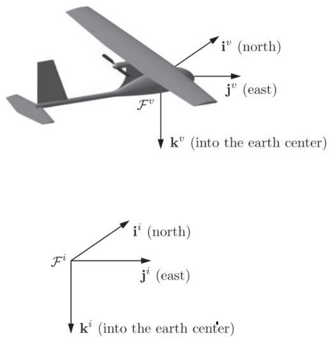
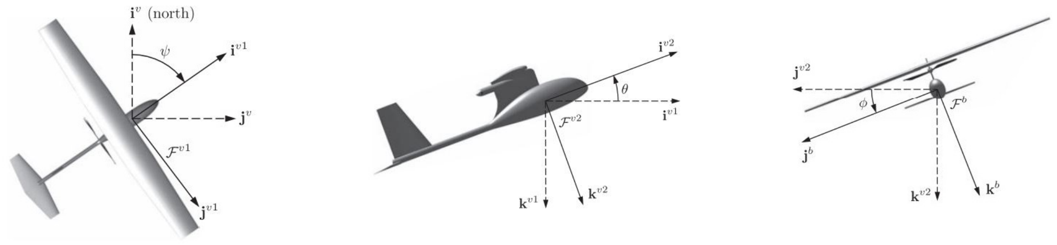


Figure 1: NED and Vehicle frame [9]

Body
Rotation the Vehicle frame with the Euler angles yaw, pitch and roll, denoted ψ, θ and ϕ, respectively, gives the body frame. This rotates are seen in figure 2. Denote this frame as $\mathcal{F}^b$ and seen in figure 2c.



(a) Yaw rotate around k^v [9] (b) Pitch rotate around j^{v1} [9] (c) Roll rotate around i^{v2} [9]

Figure 2: Rotation steps from $\mathcal{F}^v$ to $\mathcal{F}^b$

The total rotation, seen in figure 2 has the rotation matrix

$$\mathbf{R}_v^b(\mathbf{\Theta}) = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}}_{\text{Around } i^{v2}} \underbrace{\begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}}_{\text{Around } j^{v1}} \underbrace{\begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Around } k^{v}} \tag{1}$$

Note that this has a singularity if the pitch angle is $\pm$ 90°and motivates the use of quaternion, which has no singularity.

Stability and wind

The stability- and wind frame are for calculation purposes. Stability frame is used to calculate the decoupled lateral forces and wind frame to define airspeed denoted as V_a. Note that when talking about the airspeed, this is the magnitude if not express otherwise. The vector is denoted $\mathbf{v_a}$.

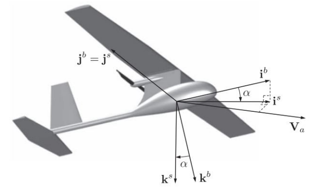


Figure 3: Stability frame [9]

Stability is the frame when rotating around $\mathcal{F}^b$ y-axis with angle, α, called the angle of attack. Denote this frame as $\mathcal{F}^s$ and seen in figure 3. If the positive angle of attack, the wings will create lift, and the plane elevates and descend if negative. It is the airspeed that acts on the wings to create lift.

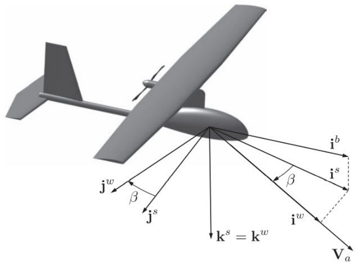


Figure 4: Wind frame [9]

Wind frame is the frame when rotating around $\mathcal{F}^s$ z-axis with angle, β, called sideslip angle. Denote this frame as $\mathcal{F}^w$ and seen in figure 4. The sideslip angle can be seen as an angle the UAV *sliding* out. After this rotation, the i^w is aligned with V_a. Gives the trivial representation of the airspeed vector denoted in the wind frame.

$$\mathbf{v_a^w} = \begin{bmatrix} V_a & 0 & 0 \end{bmatrix}^T \tag{2}$$

The two rotation matrices are defined as

$$\mathbf{R}_b^s(\alpha) = \underbrace{\begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}}_{\text{Around } j^b} \tag{3}$$

$$\mathbf{R}_s^w(\beta) = \underbrace{\begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Around } k^s} \tag{4}$$

The Euler angles, defined above, represent the orientation of the UAV, and are

$$\mathbf{\Theta} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T \tag{5}$$

The yaw angle, ψ, can be replaced by the course angel, χ. In the wind triangle in figure 5 can see that the wind is acting on the UAV. UAV is crabbing into the wind, and the crab angle is defined as the difference between the course and yaw angle.

$$\chi_c = \chi - \psi \tag{6}$$

Also seen in figure 5 is the speed vectors. Airspeed is denoted as $\mathbf{v_a}$, windspeed $\mathbf{v_w}$, and groundspeed $\mathbf{v_g}$. The relationship between this speed are

$$\mathbf{v_a} = \mathbf{v_g} - \mathbf{v_w} \tag{7}$$

The groundspeed vector is defined in the $\mathcal{F}^b$

$$\mathbf{v_g^b} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{8}$$

which are typically states in the system. In equation 2, the airspeed is denoted in $\mathcal{F}^w$, rotating this to $\mathcal{F}^b$ gives

$$\mathbf{v_a^b} = \begin{bmatrix} u_r \\ v_r \\ w_r \end{bmatrix} = \mathbf{R}_w^b(\alpha,\beta)\mathbf{v_a^w} \tag{9}$$

where u_r, v_r and w_r as the relative speed in the $\mathcal{F}^b$. The rotation from $\mathcal{F}^b$ to $\mathcal{F}^w$ is

$$\mathbf{R}_b^w(\alpha,\beta) = \mathbf{R}_s^w(\beta)\mathbf{R}_b^s(\alpha) \tag{10}$$

The relationship gives

$$\begin{aligned} V_a &= \sqrt{u_r^2 + v_r^2 + w_r^2} \\ \alpha &= \arctan\frac{w_r}{u_r} \\ \beta &= \arcsin\frac{v_r}{V_a} \end{aligned} \tag{11}$$

In figure 5 the airspeed, groundspeed, and windspeed in shown. Both course and heading angles can describe the direction of the UAV. The course angle is used to navigate airplanes because of measures groundspeed with, for example, GPS. When using a compass, typically on boats, heading angle is preferred. In this thesis, the vector field-based controller uses course angle, and the NMPC controllers use heading angle.

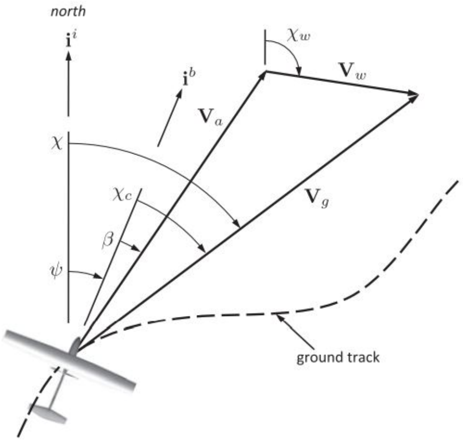


Figure 5: Horizontal wind triangle [9]

Typically overview of the states and description is seen in table 1.

Name	Description
p_n	Postion along i^i in $\mathcal{F}^i$
p_e	Postion along i^i in $\mathcal{F}^i$
p_d	Postion along k^i in $\mathcal{F}^i$
u	Velocity along i^b in $\mathcal{F}^b$
v	Velocity along j^b in $\mathcal{F}^b$
w	Velocity along k^b in $\mathcal{F}^b$
ϕ	Roll angle defined with respect to $\mathcal{F}^{v2}$
θ	Pitch angle defined with respect to $\mathcal{F}^{v1}$
ψ	Heading (yaw) angle defined with respect to $\mathcal{F}^v$
p	Roll rate measured along i^b in $\mathcal{F}^b$
q	Pitch rate measured along j^b in $\mathcal{F}^b$
r	Yaw rate measured along k^b in $\mathcal{F}^b$

Table 1: States

Kinematic states

The kinematic states are the position of the UAV in $\mathcal{F}^i$ and the Euler angles, seen in table 1. In the next paragraph, the kinematic states are defined. It is also common to use simplified kinematic models to represent the UAV, where the angle rates are considered as input. The simplified models use geometric representation.

First start be looking at the translation states $[p_n \quad p_e \quad p_d]^T$. Time differential the position gives, where $[u \quad v \quad w]^T$ is in $\mathcal{F}^v$, gives

$$\frac{d}{dt}\begin{bmatrix} p_n \\ p_e \\ p_d \end{bmatrix} = \mathbf{R}_b^v \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{12}$$

With the properties of the rotation, the inverse is the same as the transpose matrix. The expression for $\mathbf{R}_v^b$, see equation 1. Then

$$\begin{bmatrix} \dot{p_n} \\ \dot{p_d} \\ \dot{p_d} \end{bmatrix} = (\mathbf{R}_v^b)^T \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{13}$$

Next is the rotation states. In the appendix 7.3 it is seen that time differentiate a rotation matrix is

$$\boldsymbol{\omega} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \mathbf{R}_{v2}^b(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \mathbf{R}_{v2}^b(\phi)\mathbf{R}_{v1}^{v2}(\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}$$

By some rearranging, and defined that $\boldsymbol{\omega} = [p \quad q \quad r]^T$ then:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{14}$$

Gives the state explicit

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & -\sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{15}$$

Equations 13 and 15 is the six full kinematic equations.

Dynamic states

The dynamic equation is affected by forces, moments and atmospheric disturbances. Finding those equation uses the rigid-body dynamics. Starting by forces and newtons second law gives

$$\begin{aligned} \sum f &= ma \\ \sum \mathbf{f}^b &= m\frac{d\mathbf{v}_g}{dt_i} \end{aligned} \tag{16}$$

Where the typical *a* is the groundspeed vector time differentiated. Example of time differentiation a vector is seen in appendix 7.4. Note also that the force $\mathbf{f}^b$ is defined in $\mathcal{F}^b$. Writing out the time differentiating gives for equation 16

$$\sum \mathbf{f}^b = m(\frac{d}{dt_b}\mathbf{v}_g^b + \boldsymbol{\omega}_{b/i}^b \times \mathbf{v}_g) \tag{17}$$

Write that $\mathbf{v}_g^b = [u \quad v \quad w]^T$ and $\boldsymbol{\omega}_{b/i}^b = [p \quad q \quad r]^T$. Here $\boldsymbol{\omega}_{b/i}^b$ is the angular rotation between $\mathcal{F}^i$ and $\mathcal{F}^b$ denoted in $\mathcal{F}^b$. Writing this out, and rearrange for the explicit state gives:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \frac{1}{m}\sum \mathbf{F}^b \tag{18}$$

Which is the dynamic equation for the translation. Next is the dynamic rotation equation. Starting again with the newtons law for sum of moments, which is

$$\sum \mathbf{m} = \mathbf{I}\boldsymbol{\alpha} \tag{19}$$

Where $\mathbf{m}$ is moments, $\mathbf{I}$ is interia of moments and α is the angular acceleration. This can be rewritten as

$$\sum \mathbf{m}^b = \frac{d\mathbf{h}}{dt_i} \tag{20}$$

Where h is the angular momentum. The moments $\mathbf{m}$ is defined in $\mathcal{F}^b$. The time differential of $\mathbf{h}$ is:

$$\frac{h}{dt_i}\mathbf{h} = \frac{h}{dt_b}\mathbf{h} + \boldsymbol{\omega}_{b/i} \times \mathbf{h} \tag{21}$$

Insert this into equation 20 gives

$$\sum \mathbf{m}^b = (\frac{d}{dt_b}\mathbf{h}^b + \boldsymbol{\omega}_{b/i}^b \times \mathbf{h}^b) \tag{22}$$

Where $\mathbf{h}^b = \mathbf{J}\boldsymbol{\omega}_{b/i}^b$ is defined, where $\mathbf{J}$ is the interia of moments. This is equivalent as in equation 19. Insert this gives with the moments defined in $\mathcal{F}^b$

$$\sum \mathbf{m}^b = \frac{d}{dt_b}\mathbf{J}\boldsymbol{\omega}_{b/i}^b + \boldsymbol{\omega}_{b/i}^b \times \mathbf{J}\boldsymbol{\omega}_{b/i}^b \tag{23}$$

With the interia matrix constant this can write as

$$\sum \mathbf{m}^b = \mathbf{J}\frac{d}{dt_b}\boldsymbol{\omega}_{b/i}^b + \boldsymbol{\omega}_{b/i}^b \times \mathbf{J}\boldsymbol{\omega}_{b/i}^b \tag{24}$$

Can get the states from $\frac{d}{dt_b}\boldsymbol{\omega}_{b/i}^b = [\dot{p} \quad \dot{q} \quad \dot{r}]^T$, this gives

$$\sum \mathbf{m}^b = \mathbf{J}\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \boldsymbol{\omega}_{b/i}^b \times \mathbf{J}\boldsymbol{\omega}_{b/i}^b \tag{25}$$

Rearranging, then the state is explicit

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \mathbf{J}^{-1}\left(\sum \mathbf{m}^b - \boldsymbol{\omega}^b_{b/i} \times \mathbf{J}\boldsymbol{\omega}^b_{b/i}\right) \tag{26}$$

Which is the dynamics of the rotation. Since the aircraft is assumed symmetric about z^b and x^b, the inertia matrix $\mathbf{J}$ can be written as, with $J_{XY} = J_{YZ} = 0$

$$\mathbf{J} = \begin{bmatrix} J_X & 0 & -J_{XZ} \\ 0 & J_Y & 0 \\ -J_{XZ} & 0 & J_Z \end{bmatrix} \tag{27}$$

Actuators
The actuators, except throttle, manipulated the air to create force and moments to control the UAV.

Aileron, denoted δ_a, controls the roll, and it is on the wings. To control pitch, the elevators, denoted δ_e, moves the wing to make the noise go up or down. Rudder, denoted δ_r, is used to controlling the yaw and is on the tail. Last, the throttle, denoted δ_r, controls the UAVs speed.

The configuration of these control surfaces are many, but in figure 6 it is an example.

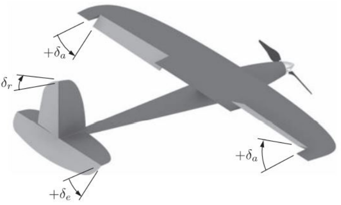


Figure 6: Example of control surfaces on a UAV [9]

Forces and moments
From the states equation 18 and 26 the sum of forces and moments are:

$$\begin{aligned} \sum \mathbf{f}^b &= \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \mathbf{f}_g + \mathbf{f}_a + \mathbf{f}_p \\ \sum \mathbf{m}^b &= \begin{bmatrix} l \\ m \\ n \end{bmatrix} = \mathbf{m}_a + \mathbf{m}_p \end{aligned} \tag{28}$$

Where subscript g is gravity, a is aerodynamics, and p is propulsion. The aerodynamics forces are lift and drag, which again gives a moment.

The gravity force is defined in $\mathcal{F}^v$, which is oriented the same as $\mathcal{F}^i$. The $\mathbf{f}^v_g$ is defined as

$$\mathbf{f}^v_g = \begin{bmatrix} 0 \\ 0 \\ m_{UAV}g \end{bmatrix} \tag{29}$$

Using the rotation matrix defined in equation 1 the gravity force can be expressed in $\mathcal{F}^b$ as

$$\mathbf{f}_g^b = \mathbf{R}_v^b(\Theta)\mathbf{f}_g^v \tag{30}$$

The lift, drag and moment m in longitudinal are

$$\begin{aligned} f_{lift} =& \frac{1}{2}\rho V_a^2 S C_L(\alpha, q, \delta_e) \\ f_{drag} =& \frac{1}{2}\rho V_a^2 S C_D(\alpha, q, \delta_e) \\ m =& \frac{1}{2}\rho V_a^2 S c C_m(\alpha, q, \delta_e) \end{aligned} \tag{31}$$

where C_L, C_D and C_m is function of aerodynamic coefficients, S is platform area of wing and c is mean chord of wing. C_L, C_D functions are nonlinear, but can be linearization to a given range of α to avoid stall conditions.

In figure 7 the lift and drag forces and moments are seen. Both lift and drag are expressed in $\mathcal{F}^s$.

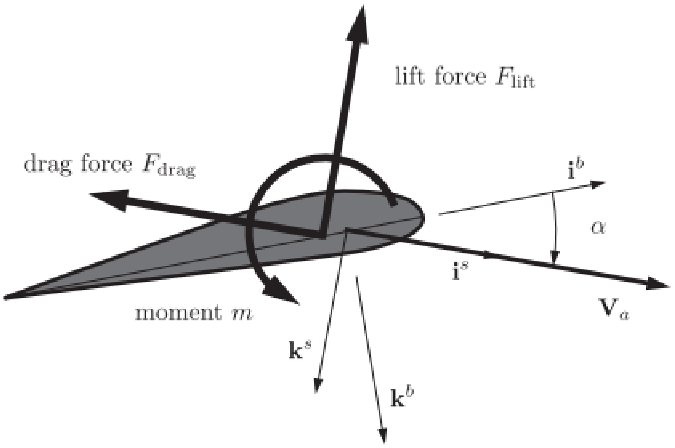


Figure 7: Lift and drag force and moment created on the UAV wing[9]

Using the rotation matrix in equation 3 to express the forces is $\mathcal{F}^b$ gives the forces in longitudinal to be

$$\begin{bmatrix} f_x \\ f_z \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} -f_{drag} \\ -f_{lift} \end{bmatrix} \tag{32}$$

Note that $\mathcal{F}^b$ and $\mathcal{F}^s$ has the same y-axsis, therefore the rotation matrix can be simplify to just rotation for x-z direction. In lateral the force and moments are

$$\begin{aligned} f_y =& \frac{1}{2}\rho V_a^2 S C_Y(\beta, p, r, \delta_a, \delta_r) \\ l =& \frac{1}{2}\rho V_a^2 S b C_l(\beta, p, r, \delta_a, \delta_r) \\ n =& \frac{1}{2}\rho V_a^2 S b C_n(\beta, p, r, \delta_a, \delta_r) \end{aligned} \tag{33}$$

where C_Y, C_l and C_n is function of aerodynamic coefficients and b is the wingspan. In [23] the aerodynamic functions are defined and also parameters of the X8 which is used later in this thesis.

Propulsion forces are from the propeller thrust with a propeller torque which gives a moment. This thesis will not go deeper into these.

From equation 32 and 33 the forces and moments are function of the angle of attack α and sideslip angle β. Therefor the states $[u \quad v \quad w]^T$ from equation 18 can be change to be $[\alpha \quad \beta \quad V_a]^T$ whcih is a good representation of the UAV and will be exploited in this thesis.

Last in this subsection, is the wind. There are two types of wind. Steady ambient wind in $\mathcal{F}^i$ and stochastic (gust) wind which is expressed $\mathcal{F}^b$. The wind vector is defined as, $\mathbf{v_w^b}$

$$\mathbf{v_w^b} = \begin{bmatrix} u_w \\ v_w \\ w_w \end{bmatrix} \tag{34}$$

$$\mathbf{v_w^b} = \mathbf{R}_v^b(\Theta) \begin{bmatrix} w_{n_s} \\ w_{e_s} \\ w_{d_s} \end{bmatrix} + \begin{bmatrix} w_{n_g} \\ w_{e_g} \\ w_{d_g} \end{bmatrix} \tag{35}$$

The way to get values for gust wind is to filter white noise through a Dryden transfer function. Here the user can adjust for altitude (height of the airplane) and strength of turbulence. The steady ambient wind is constant.

Trim conditions

Trim conditions means when the UAV is in a subset of states, and the dynamics are in equilibrium. In this thesis, this means constant-altitude, wings-level steady flight.

3.2 Path following problem

The path is denoted $\mathcal{P}$, and the goal is

$$\underset{t \to \infty}{lim} \left\| \mathbf{y}(t) - \mathcal{P}(\lambda(t)) \right\| = 0 \tag{36}$$

where $\mathbf{y}$ is the position state of the UAV and path parameter λ. The error between the position and the path is defined as

$$\mathbf{e}(t) = \mathbf{y}(t) - \mathcal{P}(\lambda(t)) \tag{37}$$

Throughout this thesis, the path is defined in the 3D-space, $\mathcal{P} \in \mathcal{R}^3$. The path can be expressed in different ways. Examples are linear-, quadratic- and cubed Bézier curves, Dubins paths, and B-splines. In this thesis, a straight-line path and B-spline curve are used.

3.2.1 Straight line path

A straight-line can be defined with two coordinates, $\mathbf{P}_i \in \mathcal{R}^3$ or waypoints $\mathbf{w}_i \in \mathcal{R}^3$, and a path parameter λ which track the location on the path. A linear Bézier is defined as

$$\begin{aligned} \mathbf{B}(\lambda) = &\mathbf{P}_0 + \lambda(\mathbf{P}_1 - \mathbf{P}_0) \\ = &(1-\lambda)\mathbf{P}_0 + \lambda\mathbf{P}_1 \end{aligned} \tag{38}$$

where $0 \leq \lambda \leq 1$. In this thesis this linear line is formulated in two different ways, depending on the properties associated with the controller.

In figure 8, it can be seen as a visual overview of the goal. In the beginning, the UAV aims at w_1, meaning $\lambda = \lambda_0$. As time goes, $\lambda \to \lambda_1$ and the UAV aims more and more at w_2. These are the green lines in the figure. So for each step, the aiming is more and more at w_2.

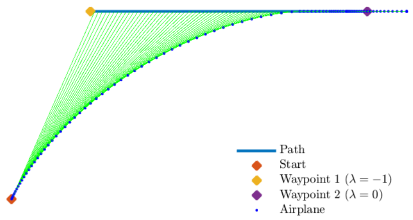
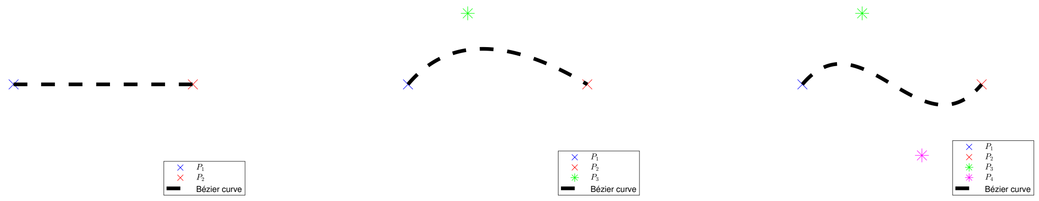


Figure 8: Illustration of path convergence

3.2.2 Curved path

B-spline is a set of Bézier curves joined end on end. A Bézier curve can be linear, quadratic or cubed. Examples of this can be seen in figure 9. In figure 9a there is two points (x), start and end. Adding one points (star), gives quadratic curve seen in figure 9b. It can be seen that the path do not go through the point, but gives a bend to the curve. Last is the cubed Bézier which is four points. Also here the path do not go through the two middle points(star),



(a) (b) (c)

Figure 9: Example of linear, quadratic and cubed Bézier curves

B-spline is k degree curved with $n+1$ points. So for n points will be $n-k+1$ Bézier curves joint in the B-spline curve. The degree have to be at least two, which is quadratic. In this thesis the degree will be $k=3$. The B-spline is continuous through the joint of the Bézier curves, meaning that the end of first Bézier curve is the same coordinate as the start of the next Bézier curve. Same as for the derivatives of the last of one, and first of second. The B-spline curve is defined as

$$\mathbf{C}(\lambda) = \sum_{i=0}^{n} N_{i,p}(\lambda)\mathbf{P}_i \tag{39}$$

where $\mathbf{P}_i$ is control points, or waypoints. As seen later in section 4.2.5, the set of control points used in the B-spline are a combination of internal control points and waypoints making a desirable curve. The basis functions are

$$\mathbf{N}_{i,0}(\lambda) = \begin{matrix} 1 & if \lambda_i \leq \lambda \leq \lambda_{i+1} \\ 0 & else \end{matrix} \tag{40}$$

$$\mathbf{N}_{i,p}(\lambda) = \frac{\lambda - \lambda_i}{\lambda_{i+p} - \lambda_i} N_{i,p-1}(\lambda) + \frac{\lambda_{i+p+1} - \lambda}{\lambda_{i+p+1} - \lambda_{i+1}} N_{i+1,p-1}(\lambda) \tag{41}$$

here λ is the path parameter. The range of this is defined in the knot vector.

$$\mathbf{u}_{knot} = \begin{bmatrix} \lambda_0 & \dots & \lambda_m \end{bmatrix} \tag{42}$$

where $\lambda \in [\lambda_0, \lambda_m]$. Number of knots, m, is defined as $m = k + n + 1$. By chosen values for the knot vector, the curve on the B-spline can be decided [22].

3.3 Successive loop closure

Successive loop closure is a principle for which the inner loop is fast enough to reach reference, that the outer loop can see this as a gain of 1. Typically the bandwidth of the inner loop is 5-10 times faster than the outer loop[9].

This principle is in both the vector field-based controller and the autopilot in the inner loop of the NMPC with a simplified kinematic model. In both of them, there is a reference on yaw. From yaw, a PI controller is used to find the commanded roll. With a PD controller, the commanded input on the aileron. This is seen in equation 44 and 43

$$\phi^c = k_{p_\chi}(\chi^c - \chi) + \frac{k_{i_\chi}}{s}(\chi^c - \chi) \tag{43}$$

$$\delta_a = k_{p_\psi}(\psi^c - \psi) - k_{d_\psi}\dot{\psi} \tag{44}$$

The inner loop is roll to the aileron. This loop is closed, with a fast enough bandwidth, so $\phi^c = \phi$. Then the outer loop from course to roll, threat the inner loop as a gain constant of 1. An example of this in figure 10.

It is not wanted to have an integral effect in the inner loop because having an integral effect inside a loop can cause bandwidth problems. Because an integral is time-consuming, and this can slow down the inner loop. Meaning that the inner loop could possibly not have a gain of 1. The integral effect on the outer loop can correct the steady-state error in both loops.

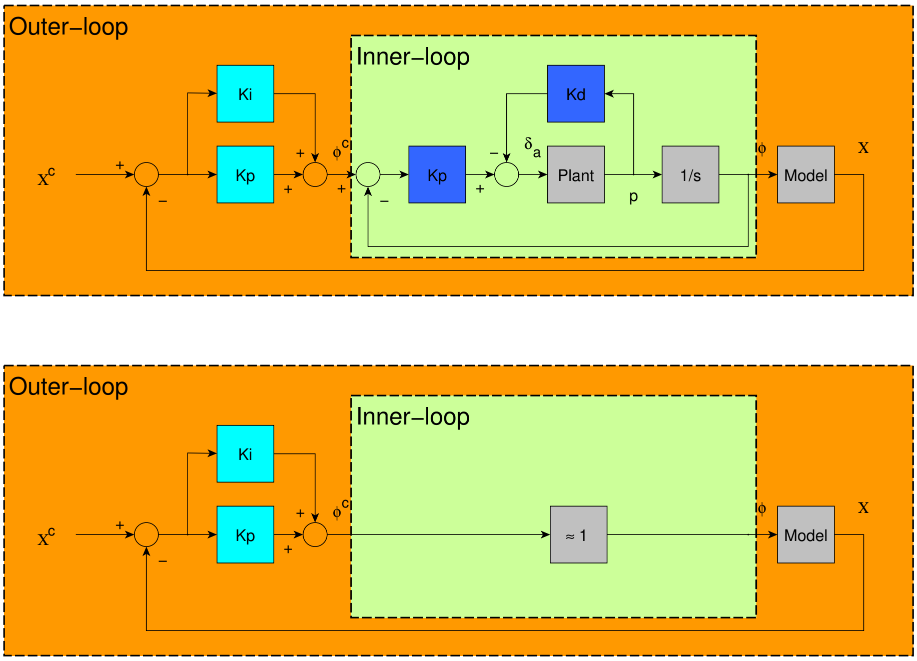


Figure 10: Control architecture of a successive loop closure [9]

3.4 Optimizing

Optimizing is to minimize or maximise a cost function. In this context, minimize. With this objective function, there is some constrains, which needs to be respected. There are equality constrains, which means that something must be equal. And inequality, which means greater or smaller. Typically denote:

$$\underset{z \in \mathbb{R}^{n_z}}{min} \quad f(z) \tag{45}$$

subject to:

$$\begin{aligned} \mathbf{c}_i(z) &= 0, \quad i \in \mathcal{E} \\ \mathbf{c}_i(z) &\geq 0, \quad i \in \mathcal{I} \end{aligned} \tag{46}$$

Here can see that $\mathcal{E}$ are the equality constraints, and $\mathcal{I}$ are the inequality constraints. $f(z)$ is the cost function, and z is the decision variable.

Some important terms

1. Feasible area: an area where all the constraints hold and can find a solution.
2. Constrains set: if the function is convex, then any two points on the function can connect with a line that does not cross the function itself. Convexity can say if the solution is a global or only local minimum. If convex, then global, and if not, then local.

A matrix, $\mathbf{Q}$, is said to be positive definite if

$$\mathbf{x}^T\mathbf{Q}\mathbf{x} \geq 0 \quad \forall x \tag{47}$$

From here, the notation of this is $\mathbf{Q} \succeq 0$.

3.5 Nonlinear Model Predictive Control

As seen in section 3.4, optimization is to minimize. In a model predictive control, the goal is to minimize an objective function, which can be the error between the wanted position and actual position.

The MPC solve the optimizing problem over a time horizontal. When all is solved, the controller only applies the first input and disregards the rest, which means that for every time step, the MPC calculates the optimal solution—illustrated in figure 11. Here the upper part of the figure shows how the MPC calculate the trajectory and only apply the first step to the plant, the lower part.

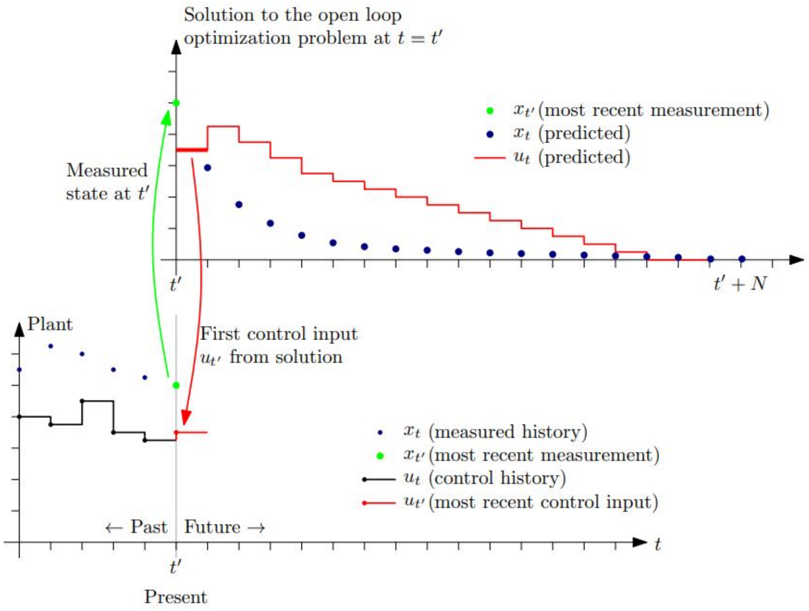

Solution to the open loop optimization problem at $t = t'$
$x_{t'}$ (most recent measurement)
x_t (predicted)
u_t (predicted)
Measured state at t'
t'
$t' + N$
Plant
First control input $u_{t'}$ from solution
x_t (measured history)
$x_{t'}$ (most recent measurement)
u_t (control history)
$u_{t'}$ (most recent control input)
← Past
Future →
t
t'
Present

Figure 11: Illustration of how the MPC works [24]

A NMPC can be formulated based on the OCP.

$$\underset{\mathbf{z} \in \mathbb{R}^n_z}{min} \quad \mathbf{f}(\mathbf{z}) \tag{48}$$

where

$$\mathbf{f}(\mathbf{z}) = \sum_{t=0}^{N-1} \frac{1}{2}\mathbf{x}_{t+1}^T\mathbf{Q}_{t+1}\mathbf{x}_{t+1} + \mathbf{d}_{xt+1}\mathbf{x}_{t+1} + \frac{1}{2}\mathbf{u}_t^T\mathbf{R}_t\mathbf{u}_t + \mathbf{d}_{ut}\mathbf{u}_t + \frac{1}{2}\Delta\mathbf{u}_t^T\mathbf{R}_{\Delta t}\mathbf{u}_t \tag{49}$$

subject to

$$\mathbf{x}_{t+1} = \mathbf{g}(\mathbf{x}_t, \mathbf{u}_t) \tag{50}$$
$$\mathbf{x}_o = given \tag{51}$$
$$\mathbf{x}^{low} \leq \mathbf{x}_t \leq \mathbf{x}^{high} \tag{52}$$
$$\mathbf{u}^{low} \leq \mathbf{u}_t \leq \mathbf{u}^{high} \tag{53}$$
$$\Delta\mathbf{u}^{low} \leq \mathbf{u}_t \leq \Delta\mathbf{u}^{high} \tag{54}$$

where

$$\begin{aligned} \mathbf{Q}_t &\succeq 0 \\ \mathbf{R}_t &\succeq 0 \\ \mathbf{R}_{\Delta t} &\succeq 0 \end{aligned} \tag{55}$$

The OCP in equation 49 is a quadratic function, with linear terms. If the OCP is ex the error, this can be seen as computing the sum of all errors from time 0 to time N-1, and then finding the optimal input to the system which gives the least error.

The last term in the OCP, is to control input change, in this thesis, this is not relevant and hence removed. This also includes the constraints on the change of input in 54.

50 - 54 is the constraints for the system. The 50 is the nonlinear equation. 51 is the start condition for state and input. Notice that index 0 on the state, and -1 on the input. 52 and 53 is the constraint on the state and input.

The 55 is the gain matrix. Note that $\mathbf{R}_{\Delta t}$ is removed (see above). All this matrix needs to be positive invariant. This matrix is here time-variant, and in this thesis, this matrix is considered time-invariant, meaning constant.

4 Control Algorithm Design

This section will investigate the methods of solving the problems formulated in section 1.1. Begin in section 4.1 by looking at the different path parameterizations used throughout this thesis. This overview will give the reader an understanding of the different path formulations in the controllers.

The main focus of this thesis is the two optimal controllers seen in section 4.2. Begin in section 4.2.1 by looking at the augmented system proposed by [13] together with the timing law. Then the two models are formulated in section 4.2.2 and 4.2.3 together with the control architecture. The path in this thesis will be both straight and curved. For the straight line, there needs to be a switch mechanism between paths. This mechanism is described in section 4.2.4 together with the initialize of the path parameter. The curved path is seen in section 4.2.5.

Both NMPC will be compared against a geometric controller (gc). For straight line, a VFB controller from [9] is used. This controller is seen in section 4.4.2. In the curved path, a NDGPFG controller from [25] is used. This is described in seen in section 4.4.3.

Last, in section 4.5, the algorithm for choosing PI/PD gains for the lateral- and longitudinal directional autopilot plus an airspeed controller. Both the VFB and NDGPFG uses this autopilot. In the low-level autopilot in the NMPC the lateral autopilot is the same, but in longitudinal, there is a pitch controller.

4.1 Path parameterization

For the straight line, the approach for the two different NMPC and the VFB is a bit different. For the curved path, the two NMPC and the NDGPFG use the same B-spline function.

Straight-line
For the optimal controler NMPC, the approach is to treat the path parameter as an error, which should be driven to zero. Then the path is defined as [13]

$$\mathcal{P} = \{y \in \mathcal{R}^3 | \lambda \in \Big[\lambda_0, \lambda_1\Big]\} \rightarrow \mathcal{P}(\lambda) \tag{56}$$

$$\mathcal{P}(\lambda) = \mathbf{w}_{i+1} - \lambda(\mathbf{w}_i - \mathbf{w}_{i+1}) \tag{57}$$

where the path parameter $\lambda \in [\lambda_0, \lambda_1] = [-1, 0]$. Then the path parameter could be included in the OCP as a virtual state to be minimized, shown in section 4.2.1. So this gives

$$\begin{matrix} lim \\ \lambda \rightarrow 0 \end{matrix} \quad \begin{bmatrix} p_n & p_e & p_d \end{bmatrix}^T \rightarrow \mathbf{w}_{i+1} \tag{58}$$

where $[p_n \quad p_e \quad p_d]^T$ is the UAV position in $\mathcal{F}^i$.

In the VFB controller the path parameter is $\lambda \in [\lambda_0, \lambda_1] = [0, 1]$.

$$\begin{matrix} lim \\ \lambda \rightarrow 1 \end{matrix} \quad \begin{bmatrix} p_n & p_e & p_d \end{bmatrix}^T \rightarrow \mathbf{w}_{i+1} \tag{59}$$

This same straight-line as the linear Bézier seen in section 3.2. Then the path is defined as [9]

$$\mathcal{P}_{line}(\mathbf{r}, \mathbf{q}) = \{\mathbf{x} \in \mathcal{R}^3 : \mathbf{x} = \mathbf{r} + \lambda \mathbf{q}, \quad \lambda \in \mathcal{R}\} \rightarrow \mathcal{P}(\lambda) \tag{60}$$

Where the $\mathbf{q} \in \mathbf{R}^3$ is defined as the difference between two waypoints

$$\mathbf{q} = \mathbf{w}_{i+1} - \mathbf{w}_i \tag{61}$$

and $\mathbf{r} \in \mathbf{R}^3$ is the first of the two waypoints, $\mathbf{w}_i \in \mathbf{R}^3$. This can be written out as

$$\mathcal{P}(\lambda) = \mathbf{w}_i + \lambda(\mathbf{w}_{i+1} - \mathbf{w}_i) \tag{62}$$

Curve

Just using the waypoints as points could mean that the path takes shortcuts. This shortcutting is not wanted, and an example of this in figure 12.

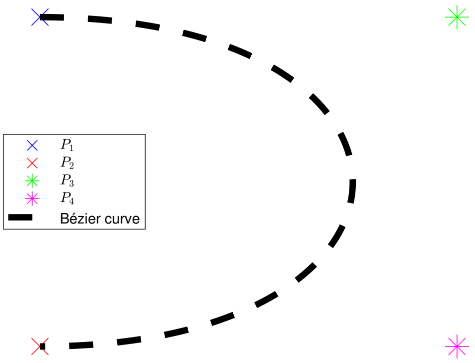


Figure 12: Example of Bézier curve with short cutting without internal control points

In [22] the approach is to add two middle points in a cubed Bézier curve, which means that from waypoint k to $k+1$, two internal control points force the path through the waypoint $k+1$. Opposite from seen in figure 12 where two waypoints are untouched. An example of this in figure 13.

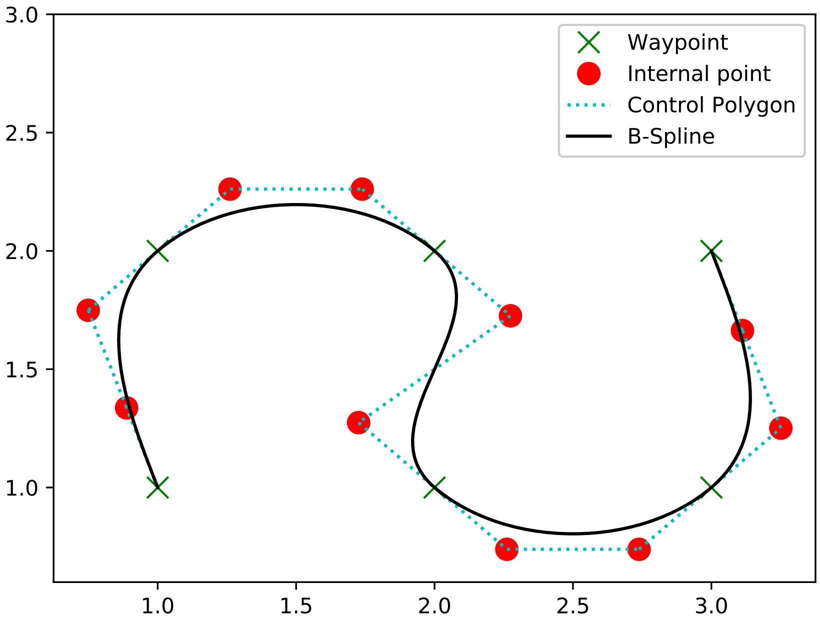


Figure 13: Example B-spline with internal control points [22]

The steps to find the two internal points is that the first internal point is tangential to the first waypoint, and the second internal point is tangential to the second waypoint. An example of the internal points with regards to the waypoints is in figure 13, where the internal points are tangential to the closest waypoint. First, start by defining the waypoints as $\mathbf{Q}_k$. The vector from waypoint $k-1$ to k is

$$\mathbf{q}_k = \mathbf{w}_k - \mathbf{w}_{k-1} \tag{63}$$

The tangential vector to waypoint $\mathbf{Q}_k$ is then given and illustrated in figure 14.

$$\mathbf{V}_k = (1-\alpha_k)\mathbf{q}_k + \alpha_k \mathbf{q}_{k+1} \tag{64}$$

where α_k is given by

$$\alpha_k = \frac{|\mathbf{q}_{k-1} \times \mathbf{q}_k|}{|\mathbf{q}_{k-1} \times \mathbf{q}_k| + |\mathbf{q}_{k+1} \times \mathbf{q}_{k+1}|}, \quad k = 2, \ldots, n-1 \tag{65}$$

$$\begin{aligned} \mathbf{q}_0 =& 2\mathbf{q}_1 - \mathbf{q}_2 \\ \mathbf{q}_{-1} =& 2\mathbf{q}_0 - \mathbf{q}_1 \\ \mathbf{q}_{n+1} =& 2\mathbf{q}_n - \mathbf{q}_{n-1} \\ \mathbf{q}_{n+2} =& 2\mathbf{q}_{n+1} - \mathbf{q}_n \end{aligned} \tag{66}$$

Unit vector of the tangent vector is given by

$$\mathbf{T}_k = \frac{\mathbf{V}_k}{|\mathbf{V}_k|} \tag{67}$$

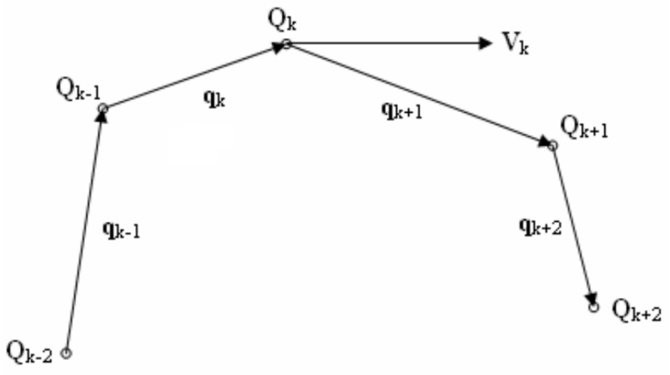


Figure 14: Tangent vector $\mathbf{V}_k$[22]

Defining the intermediate calculations as

$$\begin{aligned} \alpha =& \frac{-b+\sqrt{b^2-4ac}}{2a} \\ a =& 16-|\mathbf{T}_k+\mathbf{T}_{k+1}|^2 \\ b =& 12(\mathbf{P}_{k,3}-\mathbf{P}_{k,0})(\mathbf{T}_k+\mathbf{T}_{k+1}) \\ c =& -36|\mathbf{P}_{k,3}-\mathbf{P}_{k,0}|^2 \end{aligned} \tag{68}$$

Then the two internal points are given by

$$\begin{aligned} \mathbf{P}_{k,1} =& \mathbf{P}_{k,0}+\frac{1}{3}\alpha\mathbf{T}_k \\ \mathbf{P}_{k,2} =& \mathbf{P}_{k,3}-\frac{1}{3}\alpha\mathbf{T}_{k+1} \end{aligned} \tag{69}$$

where $\mathbf{P}_{k,0} = \mathbf{w}_k$ and $\mathbf{P}_{k,3} = \mathbf{w}_{k+1}$. Then, in this cubed Bézier curve is the four points: $\mathbf{w}_k \quad \mathbf{P}_{k,1} \quad \mathbf{P}_{k,2} \quad \mathbf{w}_{k+1}$. Loop through all the waypoints will give all the points

$$\mathbf{w}_0, \mathbf{P}_{0,1}, \mathbf{P}_{0,2}, \mathbf{P}_{1,1}, \ldots, \mathbf{P}_{n-2,2}, \mathbf{P}_{n-1,1}, \mathbf{P}_{n-1,2}, \mathbf{w}_n \tag{70}$$

Next is finding the knot vector. This is given by

$$U = \left\{0,0,0,0,\frac{\bar{u}_1}{\bar{u}_n},\frac{\bar{u}_1}{\bar{u}_n},\frac{\bar{u}_2}{\bar{u}_n},\frac{\bar{u}_2}{\bar{u}_n},\cdots,\frac{\bar{u}_{n-1}}{\bar{u}_n},\frac{\bar{u}_{n-1}}{\bar{u}_n},1,1,1,1\right\} \tag{71}$$

where

$$\bar{u}_{k+1} = \bar{u}_k + 3|\mathbf{P}_{k,1}-\mathbf{P}_{k,0}| \tag{72}$$

Looking at equation 71, the path parameter $\lambda \in [0,1]$. This the gives

$$\begin{matrix} lim \\ \lambda \to 1 \end{matrix} \quad \begin{bmatrix} p_n & p_e & p_d \end{bmatrix}^T \to \mathbf{w}_n \tag{73}$$

With the points in 70 and knots in 71 the B-Spline curve in equation 39 can be calculated. This will be done using the script seen in appendix 7.9.

For the optimal controller, minimizing the error $e_\lambda = \lambda - 1$ is included in the OCP as a virtual state, shown in section 4.2.1.

Summary
In figure 15 is a summary of the convergence of the path parameter λ. It is ready to see the difference between straight-line controllers and how they use the same function in the curved path.

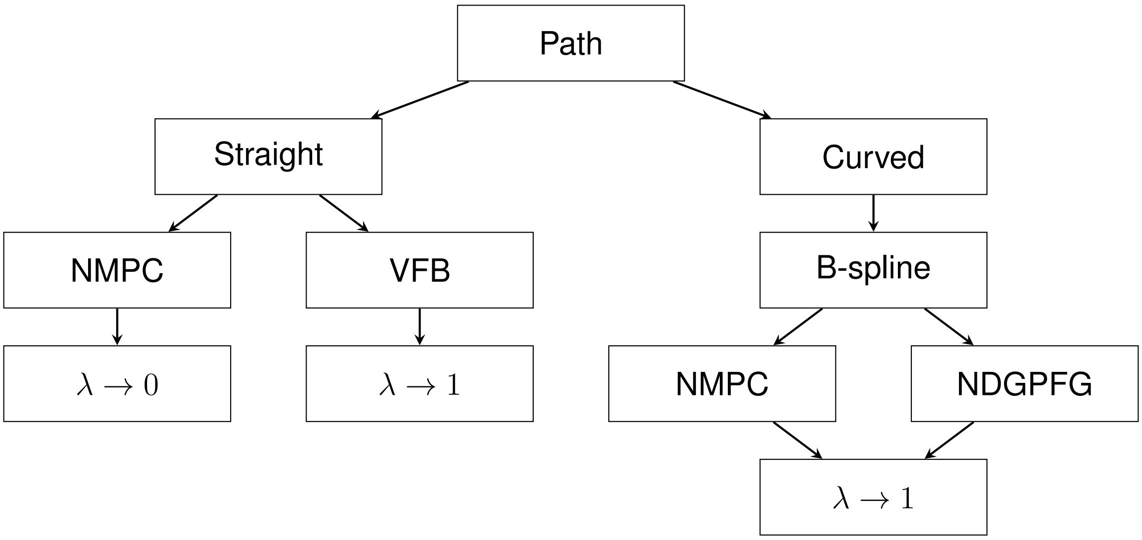


Figure 15: An overview of the path structure

4.2 Nonlinear model predictive path-following controler

This section will focus on the optimal controllers. In an optimal controller, the cost function is minimized with the input as decision variables. Therefore in section 4.2.1, begin by looking at how the OCP is defined to solve the path problem.

In the NMPC there will be a simplified kinematic model and a dynamic model. These models will be investigated in section 4.2.2 and 4.2.3. Regarding the two different path parameterizing from section 4.1, there is a difference in the formulation for the straight-line path, section 4.2.4, and curved path, section 4.2.5.

4.2.1 Optimal control problem

Timing law
The path parameter λ is introduced as a virtual state. To be able to control this parameter,

the timing law is introduce[13].

$$\begin{aligned}&\mathbf{g}\Big(\lambda^{(k)}, \lambda^{(k-1)}, ..., \dot{\lambda}, \lambda, v\Big) = 0 \\ &\forall i \in \{1, ..., k\} : \lambda^{(i)}(t_0) = \lambda_0^{(i)}, \lambda(t_0) = \lambda_0\end{aligned} \tag{74}$$

The first line defined the timing law $g(\cdot)$, the second line defined the initialization condition to the virtual states. The state v is the virtual input. This virtual input can be interpreted as controlling the behavior along the path. Timing law is chosen as [13]:

$$\begin{aligned}&\lambda^{(\hat{r}+1)} = v \\ &\lambda(t_0) = \lambda_0 \quad \forall j \in \{1, ..., \hat{r}\} : \lambda^{(j)}(t_0) = 0\end{aligned} \tag{75}$$

Where

$$\hat{\mathbf{r}} = max\{r_1, ..., r_{n_y}\} \tag{76}$$

Where $\{r_1, ...r_{n_y}\}$ is the relevant degrees.

The relative degree is the number of times it is needed to derive the output y to get input u in the states. Which method to use for this task is up to the reader, but ordinary time differential can be used or Lie derivation. A system has relative degree ρ in a region $\mathbb{D}_0 \subset \mathbb{D} \subset \mathbb{R}^n$ if [4]

$$\left.\begin{array}{ll} L_g L_f^{i-1} = 0 & , \quad 1 \leq i \leq \rho - 1 \\ L_g L_f^{\rho-1} h \neq 0 & \end{array}\right\} \forall x \in \mathbb{D}_0 \tag{77}$$

An example is found in appendix 7.5.

Where the system has more than one output, it is common to write.

$$\begin{aligned}\mathbf{r} =& (r_1, ..., r_{n_y}) \\ \rho =& \sum_{i=1}^{n_y} r_i \leqslant n_x\end{aligned} \tag{78}$$

Where n_y and n_x are numbers of output and states, respectively.

Augmented system

Given that the nonlinear system is defined as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{79}$$

$$\mathbf{y}(t) = \mathbf{h}(\mathbf{x}(t)) \tag{80}$$

The size of the state $\mathbf{x}$ is n_x. Then the function $h(x(t))$ maps from n_x to n_y.

The virtual system is defined as

$$\begin{aligned}\dot{\mathbf{z}} =& \mathbf{A}\mathbf{z} + \mathbf{b}v \\ =& \underbrace{\begin{bmatrix} 0 & \cdots & 1 & \cdots & 0 \\ \vdots & & \vdots & \ddots & \\ 0 & \cdots & 0 & \cdots & 1 \\ 0 & \cdots & 0 & \cdots & 0 \end{bmatrix}}_{\mathbf{A}} z + \underbrace{\begin{bmatrix} 0 \\ \vdots \\ \vdots \\ 1 \end{bmatrix}}_{\mathbf{b}} v\end{aligned} \tag{81}$$

with

$$\begin{aligned}\lambda =& \mathbf{c}\mathbf{z} \\ =& \underbrace{\begin{bmatrix} 1 & \cdots & 0 \end{bmatrix}}_{\mathbf{c}} \mathbf{z} \\ =& z_1\end{aligned} \tag{82}$$

In equation 81 the virtual input v is

$$z^{\hat{r}+1} = v \tag{83}$$

and the virtual states are

$$z^k = z_{k+1} \quad k = \{1, ..., \hat{r}\} \tag{84}$$

Adding the timing law with the virtual state and input, the augmented system is then defined as:

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}} \end{bmatrix} = \begin{bmatrix} \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{l}(\mathbf{z}, v) \end{bmatrix} \tag{85}$$

with the OCP

$$\begin{bmatrix} \mathbf{e} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{h}(\mathbf{x}) - \mathcal{P}(z_1) \\ z_1 \end{bmatrix} \tag{86}$$

The size of the virtual state $\mathbf{z}$ is n_z. Note the the state z is not an algebraic state. Where n_z is given by

$$n_z = \hat{r} + 1 \tag{87}$$

The total size of equation 85 is $n_x + n_z$.

Path Followability

For the system to be exactly followable, Definition 4.2 in [13] is

$$\begin{aligned}\forall j, k \in \{1, \ldots, n_y\}, i = 1, \ldots, r_{k-1} :& \frac{\partial}{\partial u_j} L_f^i h_k(x) = 0 \\ \text{For at least one} j \in \{1, \ldots, n_y\} :& \frac{\partial}{\partial u_j} L^{r_k} i_f h_k(x) \neq 0\end{aligned} \tag{88}$$

and

$$\mathbf{A}(x) = \begin{bmatrix} \frac{\partial}{\partial u_1} L_f^{r_1} h_1(x) & \dots & \frac{\partial}{\partial u_{n_u}} L_f^{r_1} h_1(x) \\ \vdots & & \vdots \\ \frac{\partial}{\partial u_1} L_f^{r_{n_y}} h_1(x) & \dots & \frac{\partial}{\partial u_{n_u}} L_f^{r_{n_y}} h_1(x) \end{bmatrix} \tag{89}$$

is full rank. Here n_y and n_u is the number of outout and input. r_i is the relative degree vector

$$\mathbf{r} = \begin{bmatrix} r_1 & , & \dots & , & r_{n_y} \end{bmatrix} \tag{90}$$

and $L_f^i h(x)$ is the Lie derivative.

Cost function

From section 4.1 and equation 58 and 73 that the path parameter describes the position on the path. Using this together with the error between the UAV and path in equation 36 this is used to solved the path problem. This is seen in equation 86. It is also desirable to control the airspeed, so the airspeed error is included in the cost function. This gives

$$\mathbf{F}(\mathbf{e}) = \|(\mathbf{e})\|_{\mathbf{Q}}^2 + \|\mathbf{u}, v\|_{\mathbf{R}}^2 \tag{91}$$

where $\mathbf{e} = [e_n \quad e_e \quad e_d \quad V_a - V_{a,ref} \quad z_1 - z_{1,ref}]^T$.

From section 3.5, the NMPC is defined (here with a integral) to minimize the cost function, where

$$\min_{\mathbf{x}(\cdot), \mathbf{u}(\cdot)} \quad \int_{t=0}^{T} \mathbf{l}(\mathbf{x}(t), \mathbf{u}(t)) dt + \mathbf{M}(\mathbf{x}(T)) \tag{92}$$

subject to

$$\begin{aligned} & 0 = f(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t)), && t \in [0, T+1] \\ & \mathbf{x}_o = given \\ & \mathbf{x}^{low} \leq \mathbf{x}(t) \leq \mathbf{x}^{high}, && t \in [0, T+1] \\ & \mathbf{u}^{low} \leq \mathbf{u}(t) \leq \mathbf{u}^{high}, && t \in [0, T] \end{aligned} \tag{93}$$

Equation 93 is dependent on which model that is used. Starting at the top, $f(\dot{x}(t), x(t), u(t)) : \mathcal{R}^{n_x}$ is the implicit differential equation of the augmented system. From equation 85 this gives

$$\mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t)) = \begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{z}} \end{bmatrix} - \begin{bmatrix} \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{l}(\mathbf{z}, v) \end{bmatrix} \tag{94}$$

Next line in equation 93 is the initialization. At the beginning of the simulation, this is the trim conditions. After that the previous state is used to initialize the next. The two last lines is the constrains on the states and inputs. This, and the augmented system to the models will be further describe in section 4.2.2 and 4.2.3. The fixed horizon length is denoted T.

Looking at the cost function in equation 92, this is the ACADOs form [19]. This is defined as

$$\mathbf{l}\Big(\mathbf{x}(t), \mathbf{u}(t)\Big) = \|\mathbf{V}_x \mathbf{x} + \mathbf{V}_u \mathbf{u} - \mathbf{y}_{ref}\|_{\mathbf{W}}^2 \tag{95}$$

With the terminal cost

$$\mathbf{M}(x) = \|\mathbf{V}_x^e\mathbf{x} - \mathbf{y}_{ref}^e\|_{\mathbf{W}^e}^2 \tag{96}$$

Here the gain matrix W and W^e

$$\mathbf{W} = diag\{\begin{bmatrix}\mathbf{Q} & \mathbf{R}\end{bmatrix}\} \tag{97}$$

$$W^e = Q \tag{98}$$

The matrix $\mathbf{V}_x$ and $\mathbf{V}_u$ maps from n_x and n_u to n_y^{NMPC}, where n_x, n_u and n_y^{NMPC} is the number of states, inputs and NMPC outputs, respectively. Where $n_y^{NMPC} = n_y + n_u$, meaning including the input as a part of the output. This gives $\mathbf{V}_x\mathbf{x} = [e_n \quad e_e \quad e_d \quad V_a \quad z_1 \quad 0_{1,n_u}]^T$, $\mathbf{V}_x^e\mathbf{x} = [e_n \quad e_e \quad e_d \quad V_a \quad z_1]^T$ and $\mathbf{V}_u\mathbf{u} = [0_{1,n_y} \quad \mathbf{u}]^T$. And $\mathbf{y}_{ref} = [0 \quad 0 \quad 0 \quad V_{a,ref} \quad z_{1,ref} \quad 0_{1\times n_u}]^T$ with $\mathbf{y}_{ref}^e = [0 \quad 0 \quad 0 \quad V_{a,ref} \quad z_{1,ref}]^T$. The matrix $\mathbf{Q}$ and $\mathbf{R}$, which are $\mathcal{R}^{n_x\times n_x}$ and $\mathcal{R}^{n_u\times n_u}$ respectively, is the diagonal positive defined gain matrix penalise the states and inputs, respectively.

The virtual state z_1, which is the path parameter, has a reference $z_{1,ref}$. Seen in section 4.1 this was defined as $z_{1,ref} = 0$ for straight-line path, and $z_{1,ref} = 1$ for curved path.

4.2.2 Simplified kinematic model

In this section the simplified kinematic model will be investigated. Start by looking at the model and propose a augmented system with virtual states defined in section 4.2.1 and then prove followability. This augmented system is used in the NMPC described in equation 92 and the constrains in equation 93 is in this section defined. Then a proposed control architecture where a low-level autopilot is used to track the high-level NMPC.

The kinematic models used is based on the kinematic equation from [7] describe as

$$\begin{aligned}
\dot{p}_n =& V_a \cos\psi\cos\theta + w_n \\
\dot{p}_e =& V_a \sin\psi\cos\theta + w_e \\
\dot{p}_d =& -V_a \sin\theta + w_d \\
\dot{\theta} =& q \\
\dot{\psi} =& r
\end{aligned} \tag{99}$$

Where p_n, p_e, p_d is the UAVs position and the wind, w_n, w_e, w_d, all expressed in $\mathcal{F}^i$. The states θ, ψ,V_a is pitch, yaw and airspeed, respectively. Input is q and r. The airspeed can be measured or be a state. For better control of pitch and yaw, and the assumptions that the states cannot be directly change without any disturbances, these will be changed. In [9] a first order approximated model

$$\dot{x}_1 = b_{x_1}(x_1^c - x_1) \tag{100}$$

where b_{x_1} is a positive constant. In [11] a second order model

$$\begin{aligned}\dot{x}_1 =& x_2 \\ \dot{x}_2 =& b_0 x_1^c - b_1 x_2 - b_2 x_1\end{aligned} \tag{101}$$

where $b_{\{i\}}$ is positive constants.

To see which of the first or second order model is best, both will be explored. Where in section 5.2 a model identifications study will be presented and one of the models will be used.

First order model
For the first order model the pitch, yaw and airspeed are change to

$$\begin{aligned}\dot{V}_a =& b_{V_a}(V_a^c - V_a) \\ \dot{\theta} =& b_\theta(\theta^c - \theta) \\ \dot{\psi} =& b_\psi(\psi^c - \psi)\end{aligned} \tag{102}$$

Adding equation 102 into 99

$$\begin{aligned}\dot{p}_n =& V_a \cos\psi \cos\theta + w_n \\ \dot{p}_e =& V_a \sin\psi \cos\theta + w_e \\ \dot{p}_d =& - V_a \sin\theta + w_d \\ \dot{V}_a =& b_{V_a}(V_a^c - V_a) \\ \dot{\theta} =& b_\theta(\theta^c - \theta) \\ \dot{\psi} =& b_\psi(\psi^c - \psi)\end{aligned} \tag{103}$$

With the input

$$\mathbf{u} = \begin{bmatrix} V_a^c & \theta^c & \psi^c \end{bmatrix} \tag{104}$$

From section 4.2.1 the OCP was defined as the error between position and path. Including the control of airspeed, the output is defined as

$$\mathbf{y} = \begin{bmatrix} p_n - \mathcal{P}(z_1)_0 & p_e - \mathcal{P}(z_1)_1 & p_d - \mathcal{P}(z_1)_2 & V_a \end{bmatrix}^T \tag{105}$$

A timing law of $\hat{r} = 2$ is proposed and gives

$$\begin{aligned}\dot{z}_1 =& z_2 \\ \dot{z}_2 =& v\end{aligned} \tag{106}$$

Which is the virtual state z_1 and z_2 and the virtual input v. Then the augmented system input and output is

$$\mathbf{u} = \begin{bmatrix} V_a^c & \theta^c & \psi^c & v \end{bmatrix} \tag{107}$$

$$\mathbf{y} = \begin{bmatrix} p_n - \mathcal{P}(z_1)_0 & p_e - \mathcal{P}(z_1)_1 & p_d - \mathcal{P}(z_1)_2 & V_a & z_1 \end{bmatrix}^T \tag{108}$$

Where the relative degree vector is

$$\mathbf{r} = \begin{bmatrix} 2 & 2 & 2 & 2 & 1 \end{bmatrix}^T \tag{109}$$

Looking at the values in equation 109 this was the motivation behind chosen the timing law at equation 106.

Next is looking at path followability, seen in section 4.2.1. In appendix 7.6 matrix $\mathbf{A}$ is found to be

$$\mathbf{A}(\mathbf{x}) = \begin{bmatrix} b_{V_a} cos(\psi)cos(\theta) & -V_a b_\theta cos(\psi)sin(\theta) & -V_a b_\psi cos(\theta)sin(\psi) & -\frac{\partial \mathcal{P}(z_1)}{\partial z_1} \\ b_{V_a} cos(\theta)sin(\psi) & -V_a b_\theta sin(\psi)sin(\theta) & V_a b_\psi cos(\psi)cos(\theta) & -\frac{\partial \mathcal{P}(z_1)}{\partial z_1} \\ -b_{V_a} sin(\theta) & -V_a b_\theta cos(\theta) & 0 & -\frac{\partial \mathcal{P}(z_1)}{\partial z_1} \\ 0 & 0 & 0 & 1 \\ b_{V_a} & 0 & 0 & 0 \end{bmatrix} \tag{110}$$

This must be full rank at x_0, which is at trim conditions. Inserting the trim conditions from section 5.3.2 and the two first waypoints, the matrix $\mathbf{A}$ is

$$\mathbf{A}(\mathbf{x} = \mathbf{x_0}) = \begin{bmatrix} 2.0997 & -1.8859 & 0.3698 & 300 \\ -0.1168 & 0.1049 & 6.6465 & 700 \\ -0.0649 & -61.1708 & 0 & 20 \\ 0 & 0 & 0 & 1 \\ 2.1040 & 0 & 0 & 0 \end{bmatrix} \tag{111}$$

Witch is full rank. If $\theta_0 = \pm 90deg$, then matrix $\mathbf{A}$ is rank-deficient. But in this thesis, the pitch angle will not be greater than $\pm 35deg$. Note that $\frac{\partial \mathcal{P}(z_1)}{\partial z_1}$ will be evaluated in section 4.2.4 and 4.2.5 for straight and curved path, respectively.

Then the proposed augmented system is

$$\begin{aligned} \dot{e}_n &= V_a \cos\psi \cos\theta + w_n - \frac{d\mathcal{P}(z_1)_0}{dt} \\ \dot{e}_e &= V_a \sin\psi \cos\theta + w_e - \frac{d\mathcal{P}(z_1)_1}{dt} \\ \dot{e}_d &= -V_a \sin\theta + w_d - \frac{d\mathcal{P}(z_1)_2}{dt} \\ \dot{V}_a &= b_{V_a}(V_a^c - V_a) \\ \dot{\theta} &= b_\theta(\theta^c - \theta) \\ \dot{\psi} &= b_\psi(\psi^c - \psi) \\ \dot{z}_1 &= z_2 \\ \dot{z}_2 &= v \end{aligned} \tag{112}$$

Where $z_{\{i\}}$ is virtual states and v is a virtual input. The system input is

$$\mathbf{u} = \begin{bmatrix} V_a^c & \theta^c & \psi^c & v \end{bmatrix}^T \tag{113}$$

and the output

$$\mathbf{y} = \begin{bmatrix} e_n & e_e & e_d & V_a & z_1 \end{bmatrix}^T \tag{114}$$

The proposed augmented system in equation 112 is explicit. Changing it to implicit, as in equation 94, this is used in equation 93. Next is the constraints on the states and inputs. The error states are e_n, e_e and e_d is chosen arbitrary big enough to not be violated.

The yaw angle can, in principle, be a multiple of 2π since the UAV can rotate around itself. The yaw angle is chosen to be $\pm 360deg$. The feedback yaw angle is mapped in $(-180, 180)$. If the NMPC had this constraint on yaw angle, then instead of going from $-179deg$ to $179deg$, which is $2deg$ the smallest way, it would go $358deg$. So even though the constraints are $\pm 360deg$, the feedback is mapped in $(-180, 180)$. With setting the constraints $\pm 360deg$, the optimal solution in the time horizontal can go beyond $\pm 180deg$, and then the errors between $\psi^{NMPC} - \psi^{UAV}$ is mapped to prevent discontinuous jump. Want to avoid the ψ^{NMPC} close to the constraint, which could make it unstable in the sense of a circular turn on the path to correct the angle.

For the pitch angle, this angle has physical limits and is more constrained. If, for example, the pitch angle is $-90deg$, then the UAVs noise is pointing to the ground. Therefore choose to be $\pm 35deg$. A to high pitch angle can also led to stall conditions.

For airspeed the reference is a constant $V_{a,ref} = 18\frac{m}{s}$, therefor the max airspeed is chosen to be $25\frac{m}{s}$ with min $15\frac{m}{s}$.

The virtual states z_1 are dependent on the path. If straight line then $z_1 \in [0, -1]$ or curved $z_1 \in [0, 1]$, which is seen in section 4.1. The upper limit is chosen a bit higher to prevent constraint violence when approaching the upper limit. This was the case in the specialization project in [1]. Therefor $z_1 \in [-1, 2]$ or $z_1 \in [0, 4]$. For z_2 this is $[0, 10]$, this gives that movement on the path must be positive, and hence always along the path. Then

$$\underbrace{\begin{bmatrix} -1e5 \\ -1e5 \\ -1e5 \\ 15 \\ -35\frac{\pi}{180} \\ -2\pi \\ \underline{z}_1 \\ 0 \end{bmatrix}}_{\mathbf{x}^{low}} \leq \begin{bmatrix} e_n \\ e_e \\ e_d \\ V_a \\ \theta \\ \psi \\ z_1 \\ z_2 \end{bmatrix} \leq \underbrace{\begin{bmatrix} 1e5 \\ 1e5 \\ 1e5 \\ 25 \\ 35\frac{\pi}{180} \\ 2\pi \\ \bar{z}_1 \\ 10 \end{bmatrix}}_{\mathbf{x}^{high}} \tag{115}$$

where $\underline{z}_1 = -1$ and $\bar{z}_1 = 2$ for straight line, and $\underline{z}_1 = 0$ and $\bar{z}_1 = 4$ for curved.

For the input, V_a^c, θ^c and ψ^c is chosen the same as the upper and lower limit of their respective state. The virtual input v is chosen arbitrarily.

$$\underbrace{\begin{bmatrix} 15 \\ -35\frac{\pi}{180} \\ -2\pi \\ -10 \end{bmatrix}}_{\mathbf{u}^{low}} \leq \begin{bmatrix} V_a^c \\ \theta^c \\ \psi^c \\ v \end{bmatrix} \leq \underbrace{\begin{bmatrix} 25 \\ 35\frac{\pi}{180} \\ 2\pi \\ 10 \end{bmatrix}}_{\mathbf{u}^{high}} \tag{116}$$

Second order model
For the second order model, the pitch, yaw and airspeed is

$$\begin{aligned} \dot{V}_a &= a_a \\ \dot{a}_a &= b_{V_a,0}V_a^c - b_{V_a,1}a_a - b_{V_a,2}V_a \\ \dot{\theta} &= q \\ \dot{q} &= b_{\theta,0}\theta^c - b_{\theta,1}q - b_{\theta,2}\theta \\ \dot{\psi} &= r \\ \dot{r} &= b_{\psi,0}\psi^c - b_{\psi,1}r - b_{\psi,2}\psi \end{aligned} \tag{117}$$

where

$$\begin{aligned} \frac{d}{dt}V_a &= a_a \\ \frac{d}{dt}\theta &= q \\ \frac{d}{dt}\psi &= r \end{aligned} \tag{118}$$

and the input and output is the same as for the first order. Propose a timing law of $\hat{r} = 3$

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= z_3 \\ \dot{z}_3 &= v \end{aligned} \tag{119}$$

Then the relative degree is

$$\mathbf{r} = \begin{bmatrix} 3 & 3 & 3 & 3 & 2 \end{bmatrix}^T \tag{120}$$

Looking at the values in equation 120 this was the motivation behind chosen the timing law at equation 119.

Then the matrix $\mathbf{A}$ is found, in the same way as for the first order model. Note that the $\mathbf{A}$ is the same for the first and second order model.

$$\mathbf{A}(\mathbf{x}) = \begin{bmatrix} b_{V_a,0}cos(\psi)cos(\theta) & -b_{\theta,0}V_a cos(\psi)sin(\theta) & -b_{\psi,0}V_a cos(\theta)sin(\psi) & -\frac{d\mathcal{P}(z_1)_0}{dt} \\ b_{V_a,0}cos(\theta)sin(\psi) & -b_{\theta,0}V_a sin(\psi)sin(\theta) & b_{\psi,0}V_a cos(\psi)cos(\theta) & -\frac{d\mathcal{P}(z_1)_1}{dt} \\ -b_{V_a,0}sin(\theta) & -b_{\theta,0}V_a cos(\theta) & 0 & -\frac{d\mathcal{P}(z_1)_2}{dt} \\ 0 & 0 & 0 & 1 \\ b_{V_a,0} & 0 & 0 & 0 \end{bmatrix} \tag{121}$$

which is full rank in the trim conditions (same as the first order). The full augmented system is

$$\begin{aligned} \dot{e}_n =& V_a \cos\psi \cos\theta + w_n - \frac{d\mathcal{P}(z_1)_0}{dt} \\ \dot{e}_e =& V_a \sin\psi \cos\theta + w_e - \frac{d\mathcal{P}(z_1)_1}{dt} \\ \dot{e}_d =& - V_a \sin\theta + w_d - \frac{d\mathcal{P}(z_1)_2}{dt} \\ \dot{V}_a =& a_a \\ \dot{a}_a =& b_{V_a,0}V_a^c - b_{V_a,1}a_a - b_{V_a,2}V_a \\ \dot{\theta} =& q \\ \dot{q} =& b_{\theta,0}\theta^c - b_{\theta,1}q - b_{\theta,2}\theta \\ \dot{\psi} =& r \\ \dot{r} =& b_{\psi,0}\psi^c - b_{\psi,1}r - b_{\psi,2}\psi \\ \dot{z}_1 =& z_2 \\ \dot{z}_2 =& z_3 \\ \dot{z}_3 =& v \end{aligned} \tag{122}$$

Change equation 122 to implicit, this is used in equation 93. In the second order model the additional state constrains are the rate states. This is chosen so that the states do not change too much. The z_3 is the acceleration on the path, and are chosen so the virtual position z_1 do not move to fast for the UAV.

$$\underbrace{\begin{bmatrix} -5 \\ -10\frac{\pi}{180} \\ -10\frac{\pi}{180} \\ 0 \\ \underline{z}_3 \end{bmatrix}}_{\mathbf{x}^{low}} \leq \begin{bmatrix} \dot{V}_a \\ \dot{\theta} \\ \dot{\psi} \\ z_2 \\ z_3 \end{bmatrix} \leq \underbrace{\begin{bmatrix} 5 \\ 10\frac{\pi}{180} \\ 10\frac{\pi}{180} \\ \bar{z}_2 \\ \bar{z}_3 \end{bmatrix}}_{\mathbf{x}^{high}} \tag{123}$$

where $\bar{z}_2 = 0.09$, $\underline{z}_3 = -0.002$ and $\bar{z}_3 = 0.002$ for straight line, and $\bar{z}_2 = 0.009$, $\underline{z}_3 = -0.0002$ and $\bar{z}_3 = 0.0002$ for curved. Which makes sense that the acceleration along the path for curved is less, because this is longer since this includes the whole path, compared to just a line segment for the straight line-

For the second order model

$$\underbrace{\begin{bmatrix} 15 \\ -35\frac{\pi}{180} \\ -2\pi \\ -0.1 \end{bmatrix}}_{\mathbf{u}^{low}} \leq \begin{bmatrix} V_a^c \\ \theta^c \\ \psi^c \\ v \end{bmatrix} \leq \underbrace{\begin{bmatrix} 25 \\ 35\frac{\pi}{180} \\ 2\pi \\ 0.10 \end{bmatrix}}_{\mathbf{u}^{high}} \tag{124}$$

Disturbance observer
To compensate for any modeling error in the simplified kinematic model in down, a disturbance observer is used. This is done adding a disturbance parameter, d_d, on the augmented equation 112 or 122 for e_d. This disturbance is found through.

$$\Delta p_d(t) = \hat{p}_d(t) - p_d(1|t-1) \tag{125}$$

where $\hat{p}_d(t)$ is state of UAV and $p_d(1|t-1)$ is:

$$p_d(1|t-1) = e_d(1|t-1) + p(z_1(1|t-1)) \tag{126}$$

where $e_d(1|t-1)$ is the NMPC state and $p(z_1(1|t-1))$ is the path function at $z_1(1|t-1)$. The $t-1$ means last iteration. The disturbance is then found

$$d_d(t) = d_d(t) + l_F \Delta p_d(t) \tag{127}$$

where the l_F is the learning rate.

Control architecture
In figure 16 the architecture of the kinematic model in the NMPC with a low-level autopilot. Here the pink and purple boxes are the initializing and switch. The autopilot gets the NMPC states ψ and θ as commanded angles.

The feedback to the NMPC is position from the UAV, which are used to calculate the updatet error and the virtual states. Find the values of z_1 and $[e_n, e_e, e_d]^T$ is described later. The rest of the internal states V_a, θ, ψ, $\dot{V}_a$, $\dot{\theta}$, $\dot{\psi}$, z_2 and z_3 are feedback to the NMPC

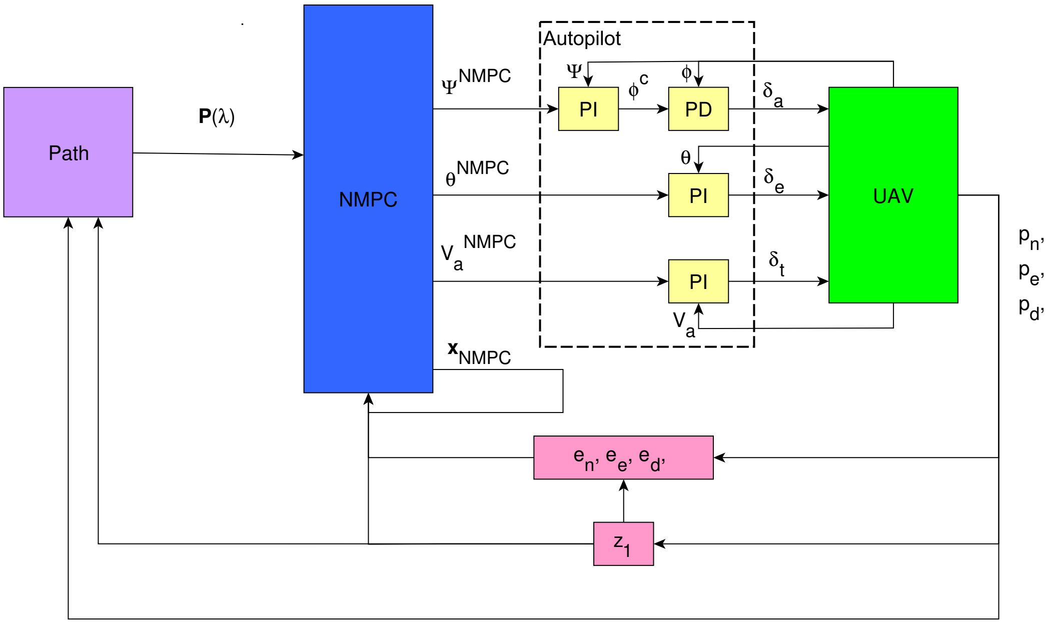


Figure 16: Control architecture of a kinematic model in NMPC in the outer-loop and an autopilot in the inner-loop

Output of the NMPC is

$$\mathbf{y}_{NMPC} = \begin{bmatrix} e_n & e_e & e_d & V_a & z_1 & V_a^c & \theta^c & \psi^c & v \end{bmatrix}^T \tag{128}$$

The following equation describes the autopilot. Note that X8 has no rudder.

$$\delta_t = K_{p_{V_a}}(V_a^{NMPC} - V_a^{UAV}) + \frac{K_{i_{V_a}}}{s}(V_a^{NMPC} - V_a^{UAV}) \tag{129}$$

$$\phi^c = K_{p_\psi}(\psi^{NMPC} - \psi^{UAV}) + \frac{K_{i_\psi}}{s}(\psi^{NMPC} - \psi^{UAV}) \tag{130}$$

$$\delta_a = K_{p_\phi}(\phi^c - \phi^{UAV}) - K_{d_\phi} p \tag{131}$$

$$\delta_e = K_{p_\theta}(\theta^{NMPC} + \theta^{UAV}) - \frac{K_{i_\theta}}{s}(\theta^{NMPC} - \theta^{UAV}) \tag{132}$$

4.2.3 Dynamic model

The differential equation in the full dynamic model is more complex than the kinematic model. For this thesis, the dynamic models of the X8 Skywalker are used. This is retrieved from the

UAV autofly repository, which is a collaborative code network at NTNU. Which frame the states are expressed in, and attitude representation can be chosen. The OCP is position error and airspeed. Therefore it is desirable to choose the states $[\beta \quad \alpha \quad V_a]^T$ instead of $[u \quad v \quad w]^T$, described in section 3.1.

The states are then

$$\mathbf{x} = \begin{bmatrix} \mathbf{p} & V_a & \beta & \alpha & \boldsymbol{\Theta} & \boldsymbol{\Omega} & \boldsymbol{\delta} \end{bmatrix}^T \tag{133}$$

where $\mathbf{p} = [p_n \quad p_e \quad p_d]^T$, $\boldsymbol{\Theta} = [\phi \quad \theta \quad \psi]^T$, $\boldsymbol{\Omega} = [p \quad q \quad r]^T$ and $\boldsymbol{\delta} = [\delta_e \quad \delta_a \quad \delta_r \quad \delta_t]^T$. The frame is the stability and the rotation is *rotmat*, meaning the Euler-angles.

Input to the system is control surfaces rates.

$$\mathbf{u} = \begin{bmatrix} \dot{\delta}_e & \dot{\delta}_a & \dot{\delta}_r & \dot{\delta}_t \end{bmatrix}^T \tag{134}$$

Choose the timing law $\hat{r} = 3$ which gives

$$\begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ v \end{bmatrix} \tag{135}$$

The input and output is

$$\mathbf{u} = \begin{bmatrix} \dot{\delta}_e & \dot{\delta}_a & \dot{\delta}_r & \dot{\delta}_t & v \end{bmatrix}^T \tag{136}$$

$$\mathbf{y} = \begin{bmatrix} p_n - \mathcal{P}(z_1)_0 & p_e - \mathcal{P}(z_1)_1 & p_d - \mathcal{P}(z_1)_2 & V_a & z_1 \end{bmatrix}^T \tag{137}$$

Then the relative degree is

$$\mathbf{r} = \begin{bmatrix} 3 & 3 & 3 & 3 & 2 \end{bmatrix}^T \tag{138}$$

Looking at the values in equation 138, this was the motivation behind chosen the timing law at equation 135.

To prove exactly followable and matrix $\mathbf{A}$ to be full rank is done by the symbolic calculating using CasADi [26] in Python. Matrix $\mathbf{A}$ needs to be full rank at x_0 (trim conditions). The only states that change in simulations is the yaw angle and positions. A configuration of the yaw angle would not affect the full rank—the result when x_0 is trim condition.

$$\mathbf{A}(\mathbf{x} = \mathbf{x}_0) = \begin{bmatrix} 0.106432 & -0.205233 & 0 & 9.50851 & -300 \\ 1.91282 & 0.0114195 & 0 & -0.529068 & -700 \\ 0 & -12.3043 & 0 & -0.294062 & 10 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & -0.20555 & 0 & 9.52322 & 0 \end{bmatrix} \tag{139}$$

Which is not full rank. But the third column is the rudder input, which is basically not an input. So, removing this column makes it full rank.

The proposed augmented system is then

$$
\begin{aligned}
\dot{e}_n =& \dot{p}_n - \frac{d\mathcal{P}(z_1)_0}{dt} \\
\dot{e}_e =& \dot{p}_e - \frac{d\mathcal{P}(z_1)_1}{dt} \\
\dot{e}_d =& \dot{p}_d - \frac{d\mathcal{P}(z_1)_2}{dt} \\
\begin{matrix} \dot{V}_a \\ \dot{\beta} V_a \\ \dot{\alpha}\cos\beta \end{matrix} =& \frac{1}{m}\mathbf{R}_b^w(\mathbf{f}_a^b + \mathbf{f}_T^b) + \mathbf{R}_b^w(\mathbf{R}_b^v)^T\mathbf{f}_g^v - \boldsymbol{\omega}_{nb}^w \times \mathbf{v}_a^w \\
\dot{\phi} =& p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \\
\dot{\theta} =& q\cos\phi - r\sin\phi \\
\dot{\psi} =& -q\sin\phi\sec\theta + r\cos\phi\sec\theta \\
\dot{\boldsymbol{\omega}}_{b/n}^s =& -\boldsymbol{\omega}_{s/b}^s \times \boldsymbol{\omega}_{b/n}^s + (\mathbf{J}^s)^{-1}(\mathbf{R}_b^s\mathbf{m}^b - \boldsymbol{\omega}_{b/n}^s \times \mathbf{J}^s\boldsymbol{\omega}_{b/n}^s) \\
\dot{\delta}_e =& u_1 \\
\dot{\delta}_a =& u_2 \\
\dot{\delta}_r =& u_3 \\
\dot{\delta}_t =& u_4 \\
\dot{z}_1 =& z_2 \\
\dot{z}_2 =& z_3 \\
\dot{z}_3 =& v
\end{aligned}
\tag{140}
$$

[27] where $\boldsymbol{\omega}_{b/n}^b = [p \quad q \quad r]^T$. To calculate the Euler angles differential equation, need to use to rotation matrix to find the angular rotation

$$
\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \mathbf{R}_s^b \omega_{b/n}^s \tag{141}
$$

Further on $\mathbf{v}_a^w = [V_a \quad 0 \quad 0]^T$, $\boldsymbol{\omega}_{s/b}^s = [0 \quad \dot{\alpha} \quad 0]^T$, $\mathbf{J}^s = \mathbf{R}_b^s\mathbf{J}^b(\mathbf{R}_b^s)^T$. Note the subscript on $\boldsymbol{\omega}_{b/n}^s$, this is to keep the angular rotations $\boldsymbol{\omega}_{b/n}^s$ and $\boldsymbol{\omega}_{s/b}^s$ separated. The subscript *b/n* can be read as the body frames angular velocity with respect to NED frame and *s/b* as the stability frame velocity with respect to body frame. Both of them expressed in stability frame. The forces, moments and rotation matrix is defined in section 3.1.

The input and output are.

$$
\mathbf{u} = \begin{bmatrix} \dot{\delta}_e & \dot{\delta}_a & \dot{\delta}_r & \dot{\delta}_t & v \end{bmatrix}^T \tag{142}
$$

$$
\mathbf{y} = \begin{bmatrix} e_n & e_e & e_d & V_a & z_1 \end{bmatrix}^T \tag{143}
$$

Next is the constraints on the states and input. The states e_n, e_e, e_d, yaw, pitch, and airspeed are chosen the same as for the kinematic model. For the roll angle ϕ, sideslip β, and angle of attack α, this is chosen from physical limits. With a too big angle of attack, stall conditions can appear. The value for this is chosen from the repository *alpha_0* $= 27deg$. The control inputs elevator, aileron, and rudder have physical limits, with the throttle response to be from 0 to 1. The angular rates are chosen arbitrarily. The virtual states z_1 and z_2 are the same as for the kinematic model, with z_3 chosen arbitrarily. This can be negative because of the ability to reduce the value of z_2. Note that since there is no rudder, the upper and lower limit is 0.

$$\underbrace{\begin{bmatrix} -1e5 \\ -1e5 \\ -1e5 \\ 15 \\ -\frac{\pi}{2} \\ -27\frac{\pi}{180} \\ -35\frac{\pi}{180} \\ -35\frac{\pi}{180} \\ -2\pi \\ -\pi \\ -\pi \\ -\pi \\ -35\frac{\pi}{180} \\ -35\frac{\pi}{180} \\ 0 \\ 0 \\ \underline{z}_1 \\ 0 \\ -10 \end{bmatrix}}_{\mathbf{x}^{low}} \leq \begin{bmatrix} p_n \\ p_e \\ p_d \\ V_a \\ \beta \\ \alpha \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \\ \delta_e \\ \delta_a \\ \delta_r \\ \delta_t \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} \leq \underbrace{\begin{bmatrix} 1e5 \\ 1e5 \\ 1e5 \\ 25 \\ \frac{\pi}{2} \\ 27\frac{\pi}{180} \\ 35\frac{\pi}{180} \\ 35\frac{\pi}{180} \\ 2\pi \\ \pi \\ \pi \\ \pi \\ 35\frac{\pi}{180} \\ 35\frac{\pi}{180} \\ 0 \\ 1 \\ \bar{z}_1 \\ 0 \\ 10 \end{bmatrix}}_{\mathbf{x}^{high}} \tag{144}$$

where $\underline{z}_1 = -1$ and $\hat{z}_1 = 2$ for straight and $\underline{z}_1 = 0$ and $\hat{z}_1 = 2$ for curved.

The input is the control surfaces rates and is chosen arbitrarily. A more realistic value for the $\dot{\delta}_i$ could be to measured how the actual actuators perform and the rates they change. Although the values in equation 145 are big, the gain matrix $\mathbf{R}$ decided how much the inputs can be used, where a big value means that the input is expensive.

$$\underbrace{\begin{bmatrix} -100 \\ -100 \\ -100 \\ -100 \\ -25 \end{bmatrix}}_{\mathbf{u}^{low}} \leq \begin{bmatrix} \dot{\delta}_e \\ \dot{\delta}_a \\ \dot{\delta}_r \\ \dot{\delta}_t \\ v \end{bmatrix} \leq \underbrace{\begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \\ 25 \end{bmatrix}}_{\mathbf{u}^{high}} \tag{145}$$

Control architecture

This can be seen in figure 17. The NMPC solves the OCP and sends the optimal δ-states to the UAV. The UAV feedback the states to the NMPC and also the position to determine the switch of waypoints. Note in figure 17 that input to the UAV is δ_a, δ_e and δ_t and not δ_r. This is because the X8 do not have a rudder. But in the feedback, the δ_r appears. This is not a mistake, this is simply a constant 0. Since the rudder is included in equation 140 this is also included in the feedback in figure 17.

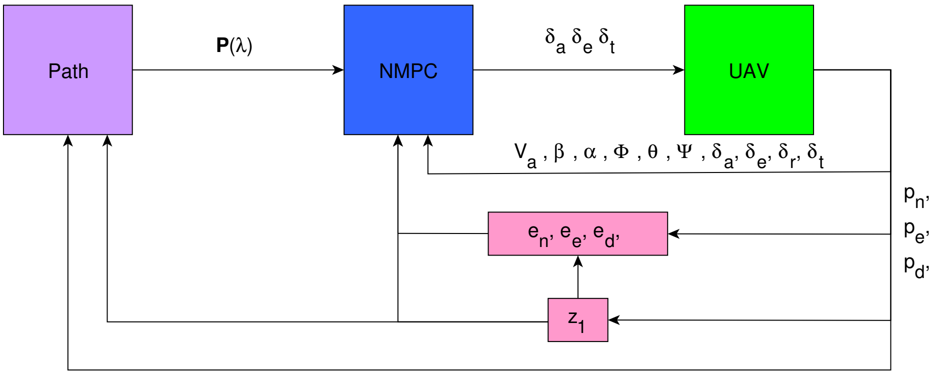


Figure 17: Control architecture of a dynamic model in the NMPC

4.2.4 Straight-line path

Initializing path parameters

The straight-line is in section 4.1 defined as

$$\mathcal{P} = \mathbf{w}_{i+1} - \lambda(\mathbf{w}_i - \mathbf{w}_{i+1}) \tag{146}$$

Seen in section 4.2.1 that $z_1 = \lambda$. This parameter needs to be initialized as the closest point on the path for the UAV. Also, with a set of waypoints, a change of path segments is needed. The z_2 and z_3 can be seen as the movement and acceleration on the path. The start movement and acceleration on the path is zero, hence $z_2 = z_3 = 0$, like the timing law. Choose z_1

$$z_1(t_0) = \underset{z_1 \in [-1,0]}{\arg\min} \left\| \mathbf{p}_{UAV} - \mathcal{P}(z_1) \right\| \tag{147}$$

Too initialize z_1, the along track error seen in figure 21 is used. This error is not used in path following problems, but in trajectory tracking. From [9] the error is describe as

$$e_{px} = \cos(\chi_q)(p_n - r_n) + \sin(\chi_q)(p_e - r_e) \tag{148}$$

where $\mathbf{p}$ and $\mathbf{r}$ is the same as in section 4.4.2.

The e_{px} is given in meters. Therefor normalized with the length of the path, w_i to w_{i+1}. Hence

$$z_{1_{init}} = \mathit{min}\Big(-1, -1 + \frac{e_{px}}{q_{norm}}\Big) \tag{149}$$

where

$$q_{norm} = \frac{\mathbf{w}_{i+1} - \mathbf{w}_i}{\|\mathbf{w}_{i+1} - \mathbf{w}_i\|} \tag{150}$$

The function can be seen in appendix 7.7.

A switch mechanism NMPC
In [9] inserting a fillet between waypoint, segments can be used to find a suitable point for which a switch can be made. The design parameter for this is the radius of the circle R, see figure 18.

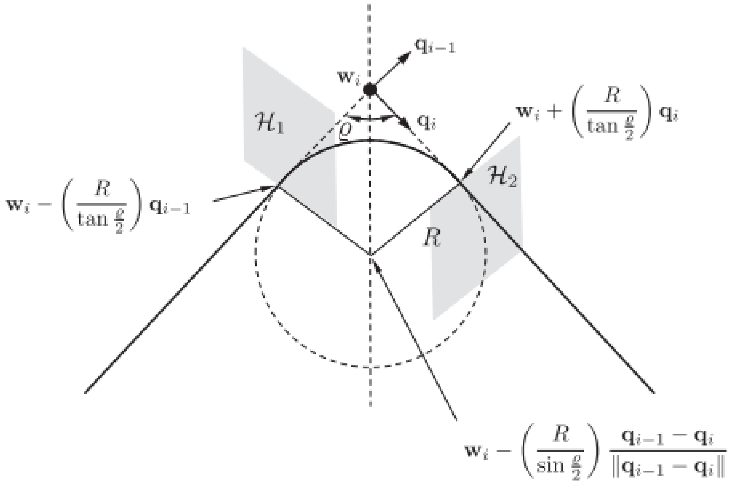

$\mathbf{q}_{i-1}$
$\mathbf{w}_i$
$\mathcal{H}_1$
ϱ
$\mathbf{q}_i$
$\mathbf{w}_i + \left(\frac{R}{\tan\frac{\varrho}{2}}\right)\mathbf{q}_i$
$\mathcal{H}_2$
$\mathbf{w}_i - \left(\frac{R}{\tan\frac{\varrho}{2}}\right)\mathbf{q}_{i-1}$
R
$\mathbf{w}_i - \left(\frac{R}{\sin\frac{\varrho}{2}}\right)\frac{\mathbf{q}_{i-1} - \mathbf{q}_i}{\|\mathbf{q}_{i-1} - \mathbf{q}_i\|}$

Figure 18: Illustration on the fillet between straight line path[9]

The angle ρ between the two lines, defined as

$$\rho = \arccos(-\mathbf{q}_{i-1}^T \mathbf{q}_i) \tag{151}$$

Where

$$\mathbf{q}_i = \frac{\mathbf{w}_{i+1} - \mathbf{w}_i}{||\mathbf{w}_{i+1} - \mathbf{w}_i||} \tag{152}$$

Define $\mathcal{S}^+$ as the stopping point and $\mathcal{S}^-$ as the start point. From figure 18 these two points are

$$\begin{aligned} \mathcal{S}_i^+ =& \mathbf{w}_i - \left(\frac{R}{\tan(\frac{\rho}{2})}\right)\mathbf{q}_{i-1} \\ \mathcal{S}_i^- =& \mathbf{w}_i + \left(\frac{R}{\tan(\frac{\rho}{2})}\right)\mathbf{q}_i \end{aligned} \tag{153}$$

The $\mathcal{S}^+$ vector is normalized

$$\mathcal{Z}_i = min\Big(-1, -1 + \frac{\mathcal{S}^+}{\mathbf{q}_i}\Big) \tag{154}$$

Where $\mathcal{Z}$ is threshold value for a switch.

$$\begin{matrix} if \mathcal{Z}_i \leq z_1 & \text{switch} \\ else & \text{pass} \end{matrix} \tag{155}$$

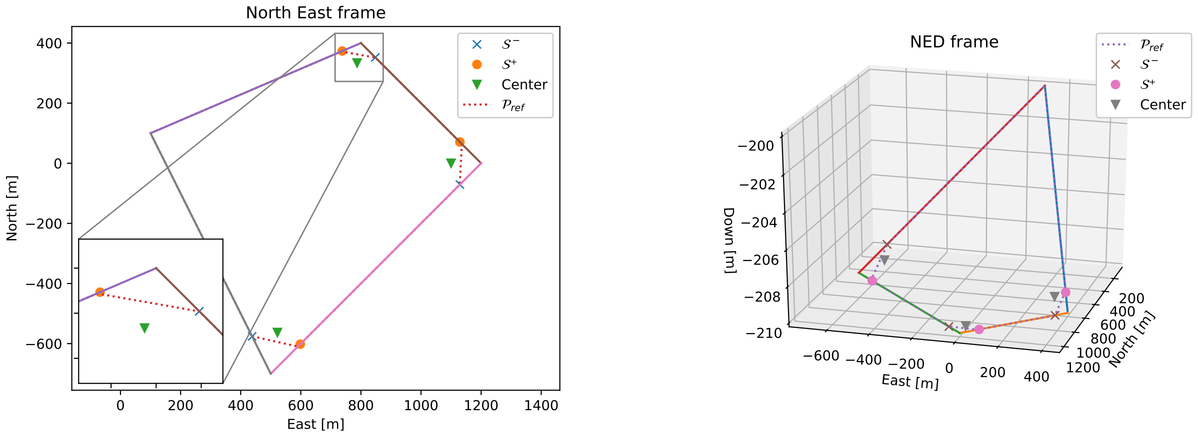
$\mathcal{S}_i^-$ is set as the new starting point for the path, replacing $\mathbf{w}_i$. Meaning that the new path to follow is

$$\mathcal{P}(z_1) = \mathbf{w}_{i+1} - z_1(\mathcal{S}_{i-1}^- - \mathbf{w}_{i+1}) \tag{156}$$

Algorithm 1 Switch

```
Waypoints index $idx_w = 0$
Find switch parameter $\mathcal{Z}_{idx_w}$
Find new start $\mathcal{S}_{idx_w}^-$
for Simulation do
    NMPC $\leftarrow (\mathbf{w}_{idx_w}, \mathbf{w}_{idx_w+1})$
    Get path parameter $z_1$ from NMPC
    if $\mathcal{Z}_{idx_w} \leq z_1$ then
        $idx_w \leftarrow idx_w + 1$
        $\mathcal{S}_{idx_w}^- \leftarrow$ eq 153
        $\mathcal{Z}_{idx_w} \leftarrow$ eq 154
        $\mathbf{w}_{idx_w} \leftarrow \mathcal{S}_{idx_w-1}^-$
        $z_1 \leftarrow$ eq 149
    else
        Pass
```

In figure 19 an example of where the switch points are. It is also seen how the reference for the UAV cut corners. The script can be seen in appendix 7.8



(a) NE-frame

(b) NED-frame

Figure 19: Example on switch points on a rectangular pattern of straight line path

Cost function

In the augmented systems 112, 122 and 140 the position error dynamics

$$\dot{\mathbf{e}} = \dot{\mathbf{p}}_{UAV} - \frac{d\mathcal{P}}{dt} \tag{157}$$

The time derivative of the path is:

$$\begin{aligned}
\frac{d\mathcal{P}}{dt} &= \frac{\partial \mathcal{P}}{\partial z_1}\dot{z}_1 \\
&= \begin{bmatrix} \frac{\partial \mathcal{P}_0}{\partial z_1} \\ \frac{\partial \mathcal{P}_1}{\partial z_1} \\ \frac{\partial \mathcal{P}_2}{\partial z_1} \end{bmatrix} \dot{z}_1 \\
&= \begin{bmatrix} \frac{\partial \left(w_{i+1_0} - z_1 \Delta w_0 \right)}{\partial z_1} \\ \frac{\partial \left(w_{i+1_1} - z_1 \Delta w_1 \right)}{\partial z_1} \\ \frac{\partial \left(w_{i+1_2} - z_1 \Delta w_2 \right)}{\partial z_1} \end{bmatrix} \dot{z}_1 \\
&= \begin{bmatrix} -\Delta w_0 \\ -\Delta w_1 \\ -\Delta w_2 \end{bmatrix} \dot{z}_1 \\
&= - \begin{bmatrix} \Delta w_0 \\ \Delta w_1 \\ \Delta w_2 \end{bmatrix} \dot{z}_1
\end{aligned} \tag{158}$$

Where

$$\begin{bmatrix} \Delta w_0 \\ \Delta w_1 \\ \Delta w_2 \end{bmatrix} = \mathbf{w}_i - \mathbf{w}_{i+1} \tag{159}$$

4.2.5 Curved path

Initializing path parameters
The curved path is described in section 4.1 and with the B-spline curve defined from equation 39, the curved path is

$$\mathcal{P}(\lambda) = \mathbf{C}(\lambda) \tag{160}$$

Note again that $z_1 = \lambda$. As for the straight-line, the path parameters $z_2 = z_3 = 0$ her as well. Finding z_1

$$z_1(t_0) = \underset{z_1 \in [0,1]}{\arg\min} \quad \left\| \mathbf{p}_{UAV} - \mathcal{P}(z_1) \right\| \tag{161}$$

this is solved using the function *project_point_to_curve* in *NurbsCurve* which solved it as a NLP.

Cost function
The error dynamics in position is still

$$\dot{\mathbf{e}} = \dot{\mathbf{p}}_{UAV} - \frac{d\mathcal{P}}{dt} \tag{162}$$

The time derivative of the path is:

$$\begin{aligned} \frac{d\mathcal{P}}{dt} =& \frac{\partial \mathcal{P}(z_1)}{\partial z_1} \dot{z}_1 \\ =& \frac{\partial \mathbf{C}(z_1)}{\partial z_1} \dot{z}_1 \end{aligned} \tag{163}$$

where $\mathbf{C}(z_1)$ is the B-spline function. This calculation is done using jacobian mxfunction in CasADi [26].

4.3 Model identification

For the model identification a cost function

$$J = \| y_{measured} - y_{model} \|_2^2 \tag{164}$$

is defined, where $y_{measured}$ is measurements from the UAV. The y_{model} is defined as

$$y_{model} = x_1 \tag{165}$$

which is the state looking at. This is V_a, θ and ψ. For the second order, chosen one of V_a, θ or ψ in equation 117, gives $f(x)$ to be

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} x_2 \\ b_0 u - b_1 x_2 - b_2 x_1 \end{bmatrix} \tag{166}$$

where input u is same input as for the UAV. The goal is to fit the nonlinear model to measurements. The constant, $b_{\{i\}}$, are decision variables and the cost function minimized. Then the optimizing problem is

$$\underset{b_0, b_1, b_2}{min} \quad \|y_{measured} - y_{model}\|_2^2 \tag{167}$$

subject to

$$\begin{gathered} \mathbf{f}(\mathbf{x}) \\ \mathbf{x}_0 \\ b_{0_0}, b_{1_0}, b_{2_0} \end{gathered} \tag{168}$$

where b_{i_0} is the initial guess of the constants.

The solver is a Gauss-Newton solver with Runge Kutta 4 integrator. Then a single shooting strategy is chosen to find the parameter value that fits the nonlinear model [26].

4.4 Geometric controllers

The two geometric controllers are included to be used for comparing the performance of the kinematic and dynamic model NMPC. Start by looking at the VFB which is used for compared the straight line path NMPC. For curved path, a NDGPFG is used.

4.4.1 Autopilot

Both geometric controllers uses a autopilot to track the commanded course/heading and height. In this section the lateral and longitudinal autopilot and a airspeed controler is presented. The control architecture for the controllers are seen in figure 20 and 25 for the VFB and NDGPFG, respectively.

Lateral autopilot
The outer-loop is to control the course/heading angle of the UAV. The high-level controler gives the commanded course/heading angle. The PD controler is

$$\phi^c = k_{p_\chi}(\chi^c - \chi) + \frac{k_{i_\chi}}{s}(\chi^c - \chi) \tag{169}$$

The inner-loop is a roll PD controler, defined as

$$\delta_a = k_{p_\phi}(\phi^c - \phi) - k_{d_\phi} p \tag{170}$$

Longitudinal autopilot
The outer-loop is to controller the height of the UAV. The high-level controler gives the commanded height. The controller is defined as the PI controler

$$\theta^c = k_{p_{p_d}}(p_d^c - p_d) + \frac{k_{i_{p_d}}}{s}(p_d^c - p_d) \tag{171}$$

The inner-loop is to control the elevator input δ_e. The PD controller is defined as: beginequation

$$\delta_e = k_{p_\theta}(\theta^c - \theta) - k_{d_\theta} q \tag{172}$$

Airspeed controler
The airspeed is set at a constant reference. The PI controller is defined as

$$\delta_t = k_{p_{V_a}}(V_a^c - V_a) + \frac{k_{i_{V_a}}}{s}(V_a^c - V_a) \tag{173}$$

The gains are described in section 4.5.

4.4.2 Path-following with vector-field based controller

This controller is a vector field-based controller, where the approach is to use successive loop closure. In the lateral direction, the aileron controls the yaw to roll, and in the longitudinal direction, the elevator controls the height to pitch. The airspeed uses a PI controller. An overview of the system in figure 20. Note that the feedback to the PI controler in the course segment is the UAV course angle, and for the PD controler the UAV roll angle. For the down segment (midle), the PI controler is feedback on UAV altiude and for the PD controler the pitch angle is feedback. For the airspeed PI controler the feedback is the UAV airspeed.

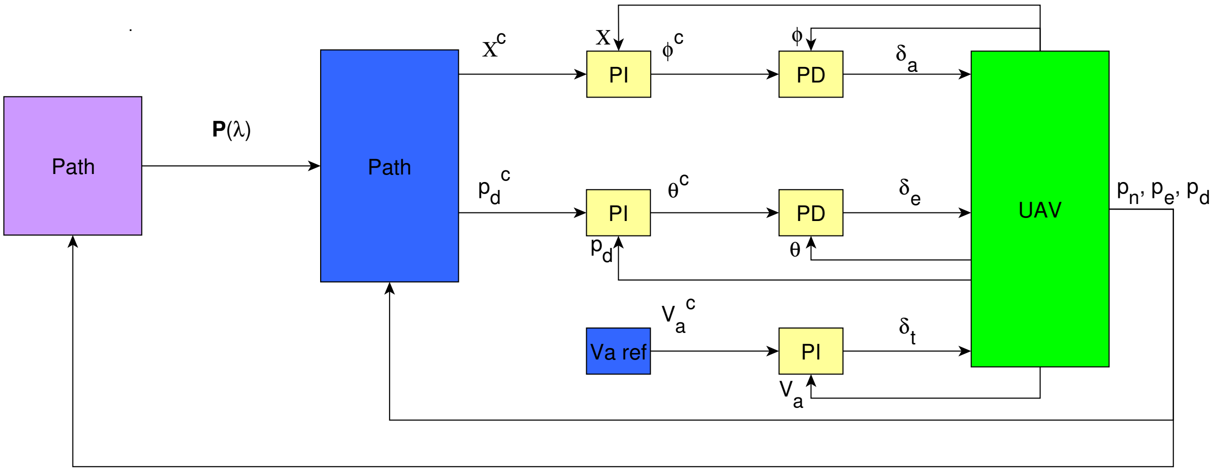


Figure 20: Control architecture of VFB controler for straight line path

Path

The path algorithm is based on vector field [9]. The output is the commanded course angle and height. Since this is a path following, only consider the error along with the cross-track error, defined as

$$e_{py} = -\sin{(\chi_q)}(p_n - r_n) + \cos{(\chi_q)}(p_e - r_e) \tag{174}$$

where this is trigonometry from figure 21.

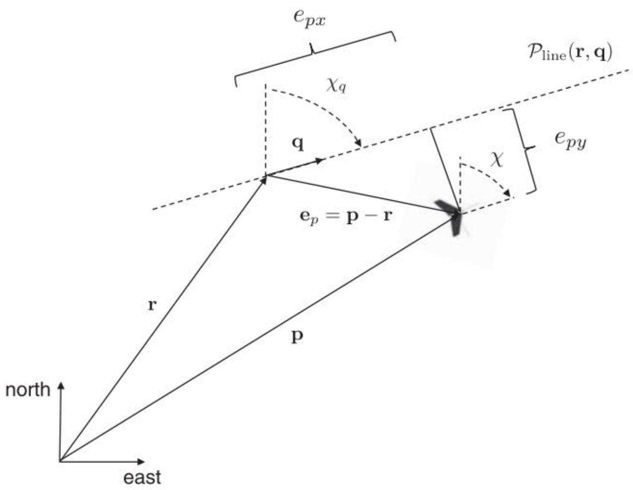

e_{px}
$\mathcal{P}_{\text{line}}(\mathbf{r},\mathbf{q})$
χ_q
$\mathbf{q}$
e_{py}
χ
$\mathbf{e}_p = \mathbf{p} - \mathbf{r}$
$\mathbf{r}$
$\mathbf{p}$
north
east

Figure 21: The cross track error e_{py} and along track error e_{px} for the UAV with respect to the path $\mathcal{P}(\lambda(t))$ [9]

Beard and McLain[9] prove stability, when driving the cross track error to zero, with the following controller

$$\chi_d = x_q - \chi^{\infty}\frac{2}{\pi}\arctan\left(k_{path}e_{py}\right) \tag{175}$$

Here the χ^{∞} is the commanded coruse angle when the aircraft is far from the line, see figure 22a . The gain k_{path} is a parameter to tune how fast the sigmoid function, $\arctan(\cdot)$, curve around zero, see figure 22b.

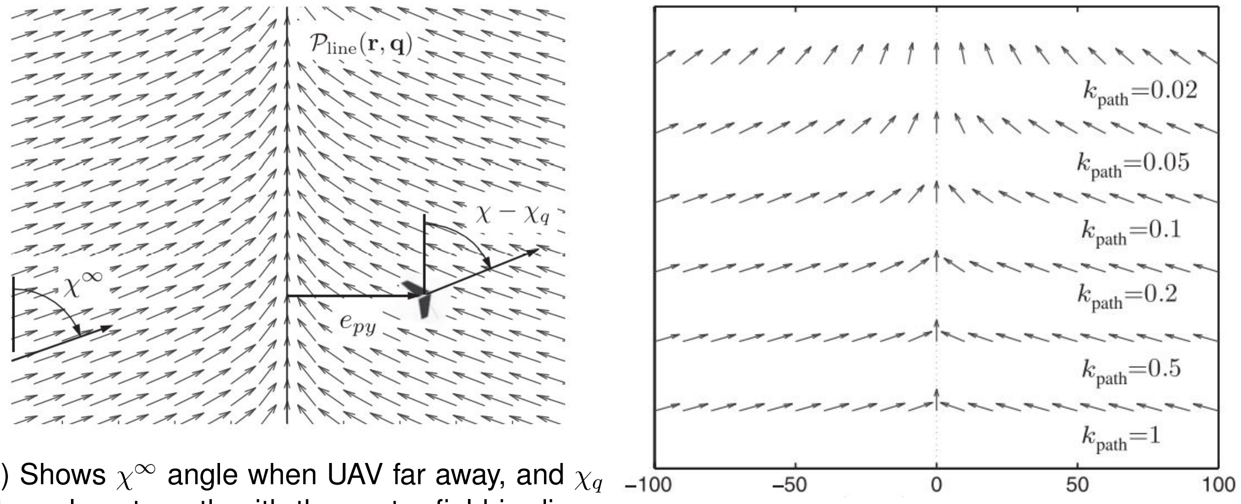
The correction of course, χ_q is defined as

$$\chi_q = atan2(q_e, q_n) \tag{176}$$

Where the $\mathbf{q}$ vector is defined as

$$\mathbf{q} = \mathbf{w}_{i+1} - \mathbf{w}_i \tag{177}$$

where $\mathbf{w}$ is waypoints.



(a) Shows χ^{∞} angle when UAV far away, and χ_q when close to path with the vector field in direction to path

(b) Gain k_{path} which effects how quick to converge

Figure 22: Visually showed how vector field guides the UAV to path [9]

To get the commanded height, the trigonometry from figure 23b is used. Figure 23b is a representation from where the $\mathbf{q}$ and $\mathbf{k}$ axis is merge. Then

$$\frac{-s_d}{\sqrt{s_n^2 + s_e^2}} = \frac{-q_d}{\sqrt{q_n^2 + q_e^2}} \tag{178}$$

Rearrange this equation to

$$s_d = -\sqrt{s_n^2 + s_e^2}\frac{q_d}{\sqrt{q_n^2 + q_e^2}} \tag{179}$$

Where the s vector is

$$\mathbf{s}^i = \mathbf{e}_i^p - (\mathbf{e}_i^p \mathbf{n})\mathbf{n} \tag{180}$$

Were the normal vector $\mathbf{n}$ is

$$\mathbf{n} = \frac{\mathbf{n} \times \mathbf{k}^i}{\|\mathbf{n} \times \mathbf{k}^i\|} \tag{181}$$

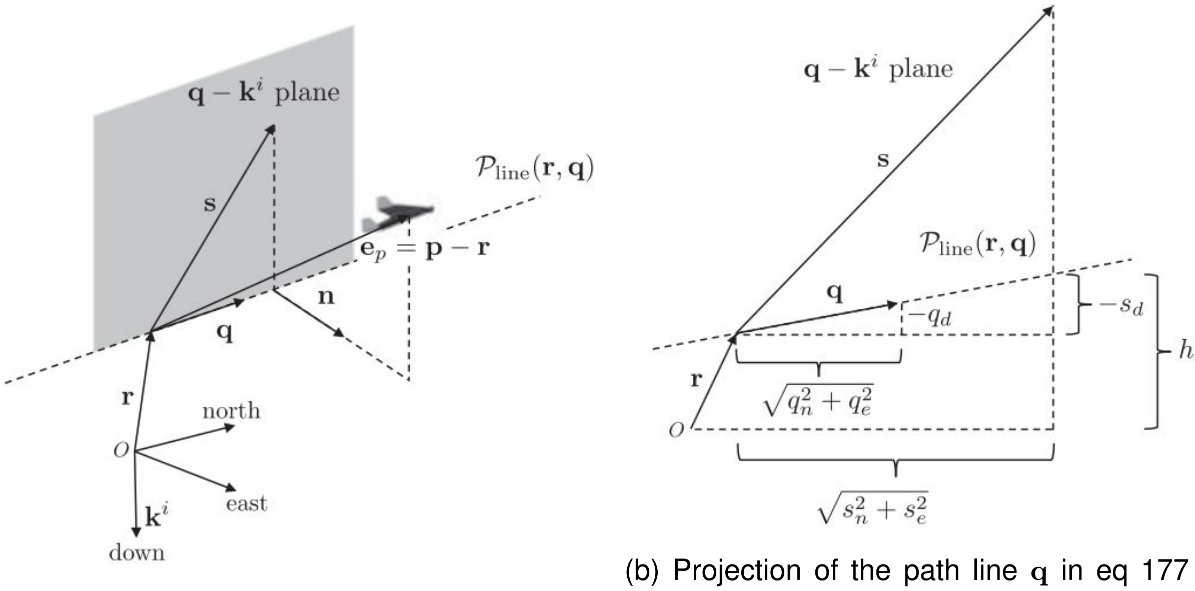
and the error e_i^p is

$$\mathbf{e_i^p} = \begin{bmatrix} e_{pn} \\ e_{pe} \\ e_{pd} \end{bmatrix} = \mathbf{p}^i - \mathbf{r}^i = \begin{bmatrix} p_n - r_n \\ p_e - r_e \\ p_d - r_d \end{bmatrix} \tag{182}$$

Where the $\mathbf{r}^i$ is the first waypoint, and $\mathbf{p}^i$ is the position for the aircraft.

The commanded height is then

$$p_d^c = h^c = -r_d - s_d \tag{183}$$


$\mathbf{q} - \mathbf{k}^i$ plane
$\mathbf{s}$
$\mathcal{P}_{\text{line}}(\mathbf{r}, \mathbf{q})$
$\mathbf{e}_p = \mathbf{p} - \mathbf{r}$
$\mathbf{n}$
$\mathbf{q}$
$\mathbf{r}$
north
O
east
$\mathbf{k}^i$
down
$\mathbf{q} - \mathbf{k}^i$ plane
$\mathbf{s}$
$\mathcal{P}_{\text{line}}(\mathbf{r}, \mathbf{q})$
$\mathbf{q}$
$-q_d$
$-s_d$
h_d
$\mathbf{r}$
O
$\sqrt{q_n^2 + q_e^2}$
$\sqrt{s_n^2 + s_e^2}$

(a) NED frame with path and UAV

(b) Projection of the path line $\mathbf{q}$ in eq 177 to a local frame with shared z-axis as NED

Figure 23: Frame for which the commanded height can be found [9]

Note then the smallest sign angle (SSA) function is used in the algorithm, which maps the angle from $[-\pi, \pi)$ to prevent discontinuous jumps in angles [28]. The function is called $SSA(\cdot)$ where the input is angle in radians.

The whole algorithm in attachments 7.7.

Switch
The loop going from UAV to waypoints is the switch mechanism. This is if the UAV is within a radius of the waypoint.

$$\text{BOOL} = \|\mathbf{p} - \mathbf{w}_{i+1}\| \leq R \tag{184}$$

Where $\mathbf{p}$ is the position of UAV and $\mathbf{w}_{i+1}$ is the targeting waypoint, both expressed in $\mathcal{F}^n$. This gives the statement

$$\begin{array}{cc} if\text{BOOL} & \text{switch} \\ else & \text{pass} \end{array} \tag{185}$$

4.4.3 Nonlinear differential geometric path-following controller

A NDGPFG from [25] proposed a guidance law design which is built on the principal of the look-ahead point-based path-following guidance law. Here the normal guidance commanded input is.

$$\mathbf{a}^{N}_{M_{cmd}} = k(\mathbf{v}_{M_I} \times \hat{\mathbf{L}} \times \mathbf{v}_{M_I}) \tag{186}$$

where $k > 0$ is the a gain, $\mathbf{v}_{M_I}$ the inertial velocity of the UAV. The look-ahead vector $\hat{\mathbf{L}}$ is defined as

$$\hat{\mathbf{L}} = \cos(\theta_L)\hat{\mathbf{d}} + \sin(\theta_L)\hat{\mathbf{T}}_P \tag{187}$$

where θ_L is the look-ahead angle and $\hat{\mathbf{d}}$ is normalize shift error vector, both seen in figure 24. The tangent vector to the path is denoted $\hat{\mathbf{T}}_P$. If $\|\mathbf{d}\| = 0$, then

$$\hat{\mathbf{L}} = \hat{\mathbf{T}}_P \tag{188}$$

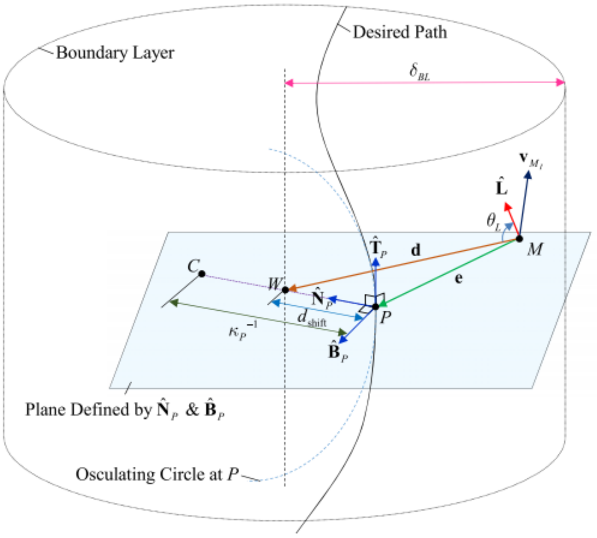


Figure 24: Guidance geometry [25]

The normalize shift error vector is

$$\begin{aligned}\mathbf{d} =& \mathbf{e} + d_{shift}\text{sign}(\kappa_P)\hat{\mathbf{N}}_P \\ \hat{\mathbf{d}} =& \frac{\mathbf{d}}{\|\mathbf{d}\|}\end{aligned} \tag{189}$$

Where there error $\mathbf{e} = \mathcal{P}(\lambda) - \mathbf{P}_{UAV}$, d_{shift} is the radially shifted distance, seen in figure 24 as the distance between $\mathbf{P}$ and $\mathbf{W}$. The variable κ_P is the curvature of the path, and $\hat{\mathbf{N}}_P$ is unitary normal vector to the path. The radially shifted distance is defined as

$$d_{shift} = \left(1 - \left(\frac{2}{\pi}\arccos\frac{|\kappa_P|}{k}\right)^2\right)\frac{\delta_{BL}}{1-\epsilon} \tag{190}$$

where δ_{BL} is the boundary-layer thickness, which is a design parameter, and ϵ is a arbitrary small value.

The look-ahead angle in equation 187 is defined as

$$\theta_L = \frac{\pi}{2}\sqrt{1-(1-\epsilon)\mathrm{sat}\left(\frac{\|\mathbf{d}\|}{\delta_{BL}}\right)} \tag{191}$$

where the saturation term is

$$\begin{aligned} \mathrm{sat}(x) =& x \quad && \text{if} \quad |x| \leq 1 \\ \mathrm{sat}(x) =& \mathrm{sgn}(x) \quad && \text{if} \quad |x| \geq 1 \end{aligned} \tag{192}$$

The $\mathbf{a}^N_{M_{cmd}} \in \mathcal{R}^3$ in equation 186 can be used in a minimising problem where the decision variables are the UAV states expressed in $\mathcal{F}^i$. From this, the p_d and ψ can be readily collected and used as commanded input. Then a successive loop closure is used, in the same way as described in section 4.4.2. The overview can be seen in figure 25.

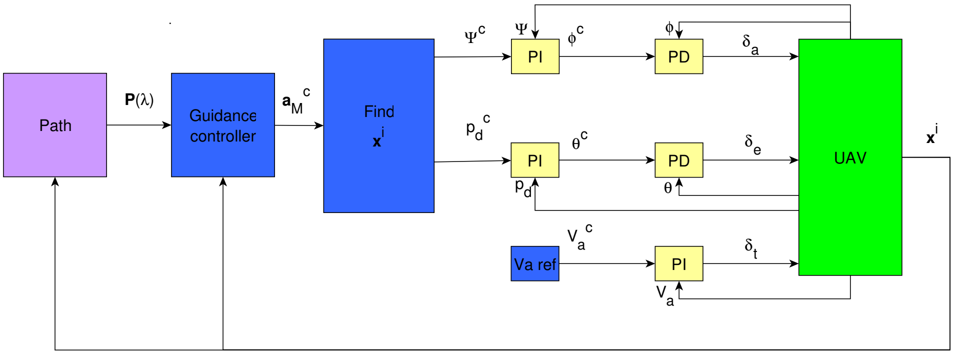


Figure 25: Control architecture of NDGPFG controler for curved path

4.5 Gains for the autopilot controller

In the lateral direction, there is a controller for heading/course and roll. The roll controller is in the inner loop, with the heading/course in the outer, using successive loop closer from section 3.3. For the longitudinal direction, the inner loop is the pitch, with the height in the outer. The last controller is for the airspeed. An overview can be seen in table 2. Note that the pitch controller for the low-level autopilot is a PI controller.

	Lateral		Longitudinal		Speed
	Yaw/course	Roll	Height	Pitch	Airspeed
Guidance controller	PI	PD	PI	PD	PI
Low level	PI	PD	-	PI	PI

Table 2: Overview PI/PD controllers

The following two subsections used the gain algorithm from [9], where the parameters of the Skywalker X8 are from [23], see appendix 7.2. The script for the gains is seen in appendix 7.10.

4.5.1 Lateral-directional autopilot gains

Heading/course gains

The gains for the course controller are based on the concept of *bank to turn*. Here the sideslip angle, β, is zero. If the aircraft has a rudder, then the sideslip angle can be kept at zero. The X8 does not have a rudder, and the sideslip angle is not zero while turning. If the sideslip angle is significant, the UAV can *fall* into the turning circle. Regardless, the gain setup works and will therefore be used. For the vector-based controller, the course angle is used. Note that the difference between the gains for the course and heading angle is that for the course, the groundspeed, V_g is used, and for heading the airspeed, V_a is used. The proportional gain is

$$k_{p_\psi} = \frac{2\zeta_\psi \omega_\psi V_a}{g} \tag{193}$$

Where g is gravity. The integral gain is

$$k_{i_\psi} = \frac{\omega_\psi^2 V_a}{g} \tag{194}$$

where

$$\omega_\psi = \frac{1}{W_\psi}\omega_\phi \tag{195}$$

The bandwidth separation, W_ψ, between the inner and outer loop and ζ_ϕ are the design parameters.

Roll gains
For the lateral inner-loop, the proportional gain is

$$k_{p_\phi} = \frac{\delta_a^{max}}{e_\phi^{max}} sign(a_{\phi_2}) \tag{196}$$

where the max error, e_ϕ^{max}, is a design parameter. This is the chosen max error between state and reference. The natural frequency is

$$\omega_\phi = \sqrt{a_{\phi_2} k_{p_\phi}} \tag{197}$$

The derivative gain is

$$k_{d_\phi} = \frac{2\zeta_\phi \omega_\phi - a_{\phi_1}}{a_{\phi_2}} \tag{198}$$

Where the damping ratio ζ_ϕ is the design parameter. The $a_{\psi_{\{i\}}}$ is based on parameters of the X8.

$$a_{\phi_1} = -\frac{1}{2}\rho V_a^2 SbC_{p_p}\frac{b}{2V_a} \tag{199}$$

$$a_{\phi_2} = \frac{1}{2}\rho V_a^2 SbC_{p_p} \tag{200}$$

Where ρ is the air density, S is the wing area, b is the wingspan of the aircraft, and

$$C_{p_p} = \Gamma_3 C_{l_p} + \Gamma_4 C_{n_p} \tag{201}$$

Where C_{l_p} is roll damping derivative and C_{n_p} is nondimensional aerodynamic coefficients for pitch and the inertia parameters

$$\Gamma_3 = \frac{J_z}{\Gamma} \tag{202}$$

$$\Gamma_4 = \frac{J_{xz}}{\Gamma} \tag{203}$$

$$\Gamma = J_x J_z - J_{xz}^2 \tag{204}$$

All parameters can be seen in table 20.

4.5.2 Longitudinal-directional autopilot gains

Height gains
The natural frequency is

$$\omega_{p_d} = \frac{1}{W_{p_d}}\omega_\theta \tag{205}$$

Where bandwidth separation, W_{p_d}, is a design parameter. The controller for the height is a PI controller. Where the integral effect removes the steady-state pitch error. In a successive loop closure, the inner loop is treated as a gain of 1. In the pitch case, the DC gain is not 1, but

$$K_{\theta_{DC}} = \frac{k_{p_\theta} a_{\theta_3}}{a_{\theta_2} + k_{p_\theta a_{\theta_3}}} \tag{206}$$

Taking advantage of this, the proportional and integral gain are

$$k_{p_{p_d}} = \frac{2\zeta_{p_d}\omega_{p_d}}{K_{\theta_{DC}} V_a} \tag{207}$$

$$k_{i_{p_d}} = \frac{\omega_{p_d}^2}{K_{\theta_{DC}} V_a} \tag{208}$$

Where the damping ratio, ζ_h, is a design parameter.

Pitch gains
With the same idea as for the proportional gain for roll, the proportional gain for pitch is

$$k_{p_\theta} = \frac{\delta_e^{max}}{e_\theta^{max}} sign(a_{\theta_3}) \tag{209}$$

where the max error, e_θ^{max}, is a design parameter. The natural frequency is

$$\omega_\theta = \sqrt{a_{\theta_2} k_{p_\theta}} \tag{210}$$

And the derivative gain is

$$k_{d_\theta} = \frac{2\zeta_\theta \omega_\theta - a_{\theta_1}}{a_{\theta_3}} \tag{211}$$

Where the damping ration, ζ_θ, is a design parameter.

The $a_{\theta_{\{i\}}}$ is based on the parameters of the X8.

$$a_{\theta_1} = \frac{\rho V_a^2 c S}{2 J_y} C_{m_q} \frac{c}{2 V_a} \tag{212}$$

$$a_{\theta_2} = \frac{\rho V_a^2 c S}{2J_y} C_{m_\alpha} \tag{213}$$

$$a_{\theta_3} = \frac{\rho V_a^2 c S}{2J_y} C_{m_{\delta_e}} \tag{214}$$

Where c is the mean chord of the wing, J_y is the y element of the inertia matrix, C_{m_q} is pitch damping derivative, C_{m_α} stability derivative and $C_{m_{\delta_e}}$ is the aerodynamic coefficients of the elevator. All parameters can be seen in table 20.

4.5.3 Airspeed controller gains

The proportional and integral gains are

$$k_{p_{V_a}} = \frac{2\zeta_{V_a}\omega_{V_a} - a_{v1}}{a_{v2}} \tag{215}$$

$$k_{i_{V_a}} = \frac{\omega_{V_a}^2}{a_{v2}} \tag{216}$$

where the natural frequency, ω_v, and damping ratio, ζ_v, are design parameters. The $a_{v_{\{i\}}}$ is found

$$a_{v1} = \frac{\rho V_a^* S}{m}\left[C_{D_0} + C_{D_\alpha}\alpha^* + C_{D_{\delta_e}}\delta_e^*\right] + \frac{\rho S_{prop}}{m} C_{prop} V_a^* \tag{217}$$

$$a_{v2} = \frac{\rho V_a^* S}{m} C_{prop} k_{motor}^2 \delta_t^* \tag{218}$$

Where subscript * is the trim conditions, C_{D_0} is the drag coefficient predicted by the linear model at zero angle of attack, C_{D_α} and $C_{D_{\delta_e}}$ are stability derivatives, S_{prop} is the area swept out by the propeller, C_{prop} is a coefficient of the propeller, k_{motor} is a gain of the motor and m is the mass of the X8.

5 Results

In this section, the results using the control algorithm design described in section 4 will be presented. Start by looking at the gains used in the low-level autopilot for both geometric controllers and the kinematic model NMPC. This is seen in section 5.1. In section 5.2 is the results of the identification of the simplified kinematic model. Then in section 5.3 an introduction to the simulation where the hard-and software is presented. Last is the results of the simulation. Start with the straight-line in section 5.4 and the curved path in section 5.5.

The names of the simulation scripts is seen in appendix 7.1 and can be found in the repository UAVlab autofly in the branch *path_following_thomas*.

5.1 Tune controller

In this section the design parameters from section 4.5 will be tuned. All the gains are functions of V_a, meaning they are adaptive. In table 3 is an overview of tunable parameters. The tuning was done by looking at the step responses in a one-by-one state. Starting by looking at the inner-loops, which is the roll and pitch. Then the outer-loop with yaw/course and height. When tuning the outer-loop, it was also important to consider the inner-loop. Especially when deciding the bandwith separation . Start by tuning without wind in the simulation. When all the parameters for both inner and outer-loop were found, the wind was included in the simulation. Then some small adjustment was made to compensate for the wind. Acceptable criteria are fast convergence to reference and also as small overshoot as possible.

	Damping ration	Bandwith separation	Max error	Natural frequency
	$\zeta_{\{i\}}$	$W_{\{i\}}$	$e_{\{i\}}^{max}$	ω_i
ψ	ζ_ψ	W_ψ		
ϕ	ζ_ϕ		e_ϕ^{max}	
p_d	ζ_{p_d}	W_{p_d}		
θ	ζ_θ		e_θ^{max}	
V_a	ζ_{V_a}			ω_{V_a}

Table 3: Design parameters

5.1.1 Lateral-directional design parameters

The lateral-directional includes the yaw/course and roll. With the actuator aileron to control the UAV. Started by tuning ζ_ϕ and e_ϕ^{max} in the roll-controler. For the two geometric controllers this is equation 170 and for the NMPC this is 131. This roll controller gives the aileron input.

Finding e_{ϕ}^{max} was intuitively selected, and having in mind that this is the only tunable parameter in the proportional gain, see equation 196. Whereas the ζ_{ϕ} influence the derivative gain, see 198. So when the proportional gain was acceptable, the derivative gain through ζ_{ϕ} was chosen.

Next, the ζ_{ψ} and W_{ψ} in the yaw/course controller. The gains for the yaw and course controllers are the same. So for the two geometric controllers, this is equation 169. Note that the VFB uses course angle, while NDGPFG uses heading. For the NMPC this is equation 130. Important to remember here is to have the W_{ψ} big enough. The bandwidth is used to decide the natural frequency of the yaw/course controler, which is the only tunable parameter in the integral gain, see equation 194. So started with tuning the ζ_{ψ} which, together with the W_{ψ}, influence the proportional gain, see equation 193. So a value greater than 5 was applied to the W_{ψ}, then tuning the ζ_{ψ}. When the proportional gain was acceptable, the integral gain through the W_{ψ} was chosen. When decided the W_{ψ} it was important to also look at the response for the roll. Because this inner-loop is assumed to be a gain of 1, see section 3.3.

The results in figure 27 without wind with a step input on heading. Through this, the commanded roll angle and aileron are found. Here the commanded yaw angle is set in figure 27a(top). At the beginning of the step response, the yaw angle is going oppositely. This behavior is highlighted in figure 26. This happens because there is a zero at the right half-plane, which makes it unstable. From [9] the transfer function for the course

$$H_{\chi} = \frac{2\zeta_{\chi}\omega_{n_{\chi}}s + \omega_{n_{\chi}}^{2}}{s^{2} + 2\zeta_{\chi}\omega_{n_{\chi}}s + \omega_{n_{\chi}}^{2}} \tag{219}$$

So, when the bandwith is close to the zero, it is unstable. Throughout the actual path simulations this has not been a problem. And it does not happen in figure 27b where there is wind.

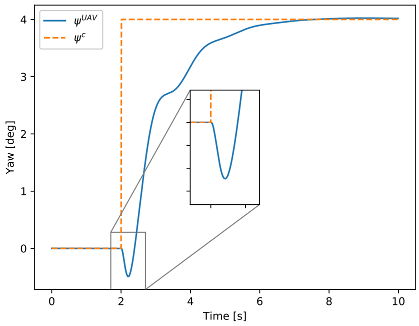
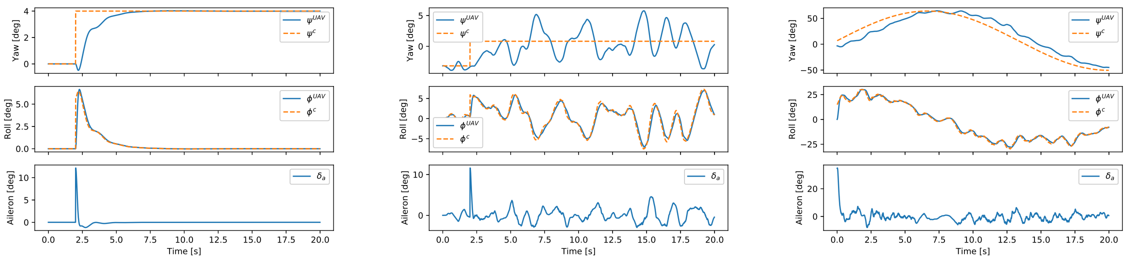


Figure 26: Minimum phase

It can be seen in figure 27a that both the yaw and roll controller are converging to reference. In figure 27b it is simulated with wind. Here, the controller can not follow the reference. The yaw controller is better to follow reference when it is not constant. This is seen figure 27c where the commanded yaw angle is a sinus curved. Since it is not likely that the yaw reference is constant for this thesis, this is accepted. Can also see in figure 27a that the roll controller has some overshoot, but this is small. The design parameters can be seen in table 4.



(a) Without wind

(b) With wind

(c) With wind

Figure 27: Lateral-directional - Yaw to roll with aileron response

Parameter	Value
W_{ψ}	20
ζ_{ψ}	0.5
ζ_{ϕ}	1.8
e_{ϕ}^{max}	15

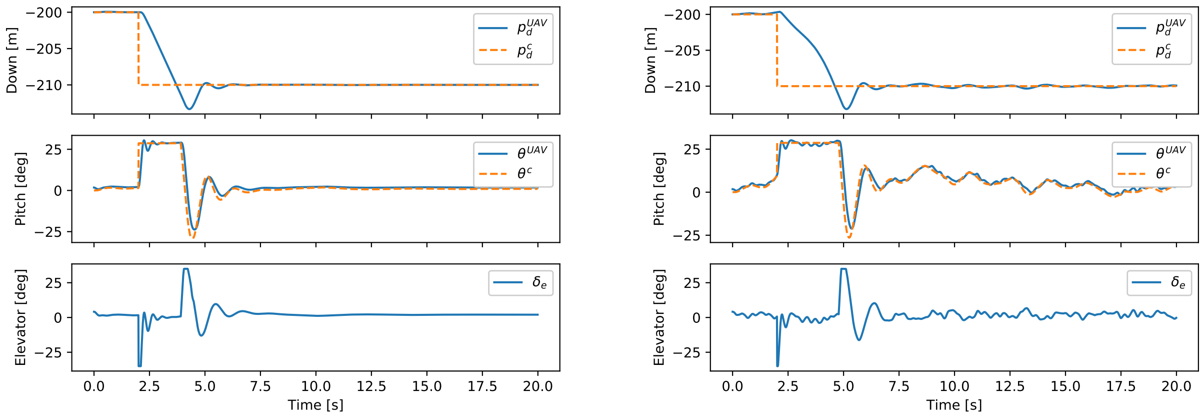
Table 4: Lateral-directional design parameters

5.1.2 Longitudinal-directional design parameters

The longitudinal-directional includes the height and pitch with the actuator elevator to control the UAV. This controler is used in the two geometric controllers, see equation 172. Start by tuning the inner-loop, which is the pitch controler, and the parameter ζ_{θ} and e_{θ}^{max}. Also, here e_{θ}^{max} is the only tunable parameter in the proportional gain, see equation 209. When the proportional gain was acceptable, the derivative gain, see equating 211, through ζ_{θ} was tuned.

Next was the height controler, and the parameters ζ_{p_d} and W_{p_d}. This is used in the two geometric controllers, see equation 171. Started by tuning the proportional gain, which is influence by the two tuneable parameter, see equation 207. So a value greater than 5 was applied to W_{p_d}, then tuning the ζ_{p_d}. Then the integral gain, see equation 208, through W_{p_d} was tuned.

The result in figure 28 simulated without wind. For this, a step input on the height. Through this, the commaned pitch angle and elevator are found. It can be seen in figure 28a that the height is given a step input to 210 meters. This is achieved with a small overshoot, and both the height controller and pitch controller are acceptable performance. In figure 28a the DC gains in the pitch are seen(the middle). This steady-state error is removed with an integral action in the height controller. It is not want to have an integral action in the inner loop. The design parameters can be seen in table 5.



(a) Without wind

(b) With wind

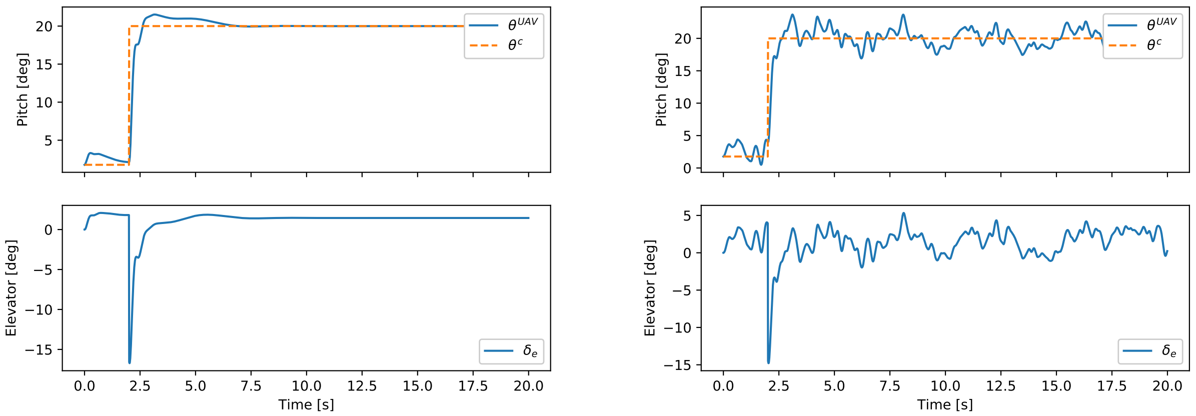
Figure 28: Longitudinal-directional - Height to pitch with elevator response

Parameter	Value
W_{p_d}	10
ζ_{p_d}	0.707
ζ_θ	1
e_θ^{max}	15

Table 5: Longitudinal-directional design parameters

Pitch controller

In the low-level controller of NMPC, there is no commanded height, but pitch commaned. Therefore, an integral effect is proposed on the pitch controller and makes it a PI controller, see equation 132. The gain of the integral is found through trial and error. The result can be seen in figure 29. In figure 29a is simulated without wind. Here the steady-state error is integrated away. With wind can be seen in figure 29b. The pitch angle is oscillating around the reference, but this is excepted. The values can be seen in table 6.



(a) Without wind

(b) With wind

Figure 29: PI pitch controler with elevator response

Parameter	Value
ζ_θ	0.707
e_θ^{max}	35
k_{i_θ}	-0.8

Table 6: PI controller pitch parameters

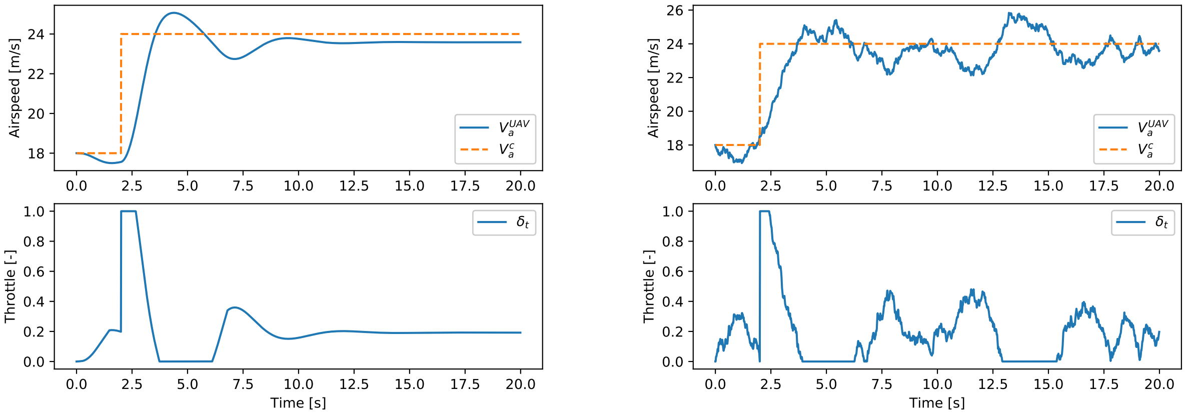
5.1.3 Airspeed

The airspeed controller was the hardest to tune because of the wind. This controler is used in both geometric controler, see equation 173 and in the low-level autopilot in the kinematic NMPC, see equation 129. The approach for this tuning was a trial and error, since both tuneable parameters, ζ_{V_a} and ω_{V_a}, influence the proportion gain, see equation 215. The integral gain can be seen in equation 216.

The airspeed controller has a steady-state error, seen in figure 30a. This is because the anti-windup threshold removes the integral effect that should drive the steady-state error to zero. If the windup threshold were higher, there would be much more oscillation. Therefore, in choosing an oscillation around the reference or a steady-state error, the choice was a small steady-state error. This error is about $0.05[m/s]$, which is not that much.

It is hard for a small UAV to follow a constant reference speed when wind is included. It can be seen in figures 30b that the airspeed is oscillating around the reference. It is the throttle that controls the airspeed. The throttle can not slow down the UAV, only increase the speed. Therefore it is hard to follow the reference airspeed exact with gust wind. So it is expected that the speed will oscillate around the reference when there is wind. So, this is acceptable.

The design parameters can be seen in table 7. Also, note that the steady-state error is gone when the wind is added.



(a) Without wind

(b) With wind

Figure 30: PI airspeed controler with throttle response

Parameter	Value
ζ_{V_a}	0.707
ω_{V_a}	$\frac{\pi}{2}$

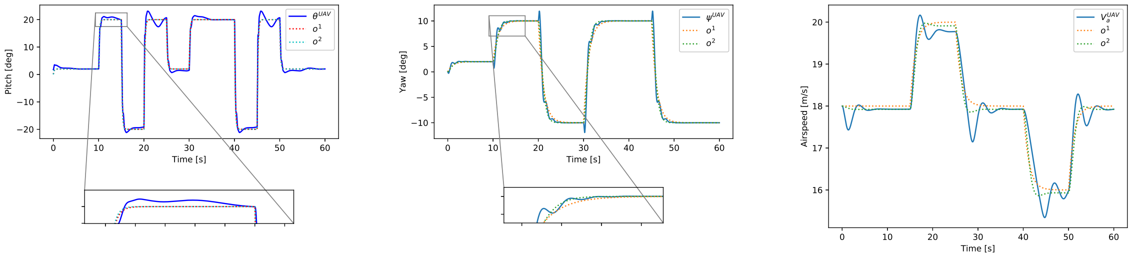
Table 7: Airspeed design parameters

5.2 Model identification

In this section the model parameters in the two simplified kinematic model seen in equations 112 and 122 identified.

The UAV was given different step input on pitch, see figure 31a. Then the closed-loop pitch dynamics of the UAV were measured. The simulation is without wind. This process was repeated for yaw in figure 31b and airspeed in figure 31c, measuring the closed-loop loop dynamics of the yaw and airspeed, respectively. Then the method of fitting a nonlinear model to measurements described in section 4.3 is used to identify the pitch, yaw, and airspeed model. For the first-order model this is equation 102, and here there is one parameter, b_i, for each state. The second-order model is equation 117, where there is three-parameter, $a_{i,0}$, $a_{i,1}$ and $a_{i,2}$, for each state.

The results can be seen in figure 31. Where the approximated models should follow the UAV response to the different step inputs. The validation of the models can be seen in table 8.



(a) Pitch (b) Yaw (c) Airspeed

Figure 31: First order vs second order kinematic equations vs actual UAV response

Using the formula for average mean squared error to validate.

$$\text{validation} = 1 - \text{mean}\left(\left[\left\|\frac{x_0-u_0}{u_0}\right\| \quad \cdots \quad \left\|\frac{x_i-u_i}{u_i}\right\|\right]\right) \quad i \in [0, \text{Nsim}] \tag{220}$$

where NSIM is the number of simulations steps and u_i is the UAV state. The validation of the models are seen in table 8. Here the second order model is following the reference best for all. Therefor, the second order kinematic model is used in the NMPC.

	O^1	O^2
V_a	98.83 %	99.07 %
θ	83.29 %	83.43 %
ψ	87.49 %	91.20 % %

Table 8: Validation on model

The parameters found is seen in table 9. Note that for airspeed, the $b_{i,0} \neq b_{i,2}$, which is the case for pitch and yaw. Reason for this is the steady state error seen in figure 30a.

	1 order	2 order		
i	$b_{i,0}$	$b_{i,0}$	$b_{i,1}$	$b_{i,2}$
V_a	0.858938	1.833	1.98789	1.84107
θ	5.66121	189.444	33.0477	189.444
ψ	0.911763	3.51349	3.59127	3.51349

Table 9: Approximated model parameter in the simplified kinematic models

5.3 Introduction to simulation

This section is an introduction to the simulation. Look at some software choices and which frequency the controllers run on. Also, the initial value of the UAV used in the simulation.

5.3.1 Controller configuration

All the simulations are ran using operative system Ubuntu 18.04 using Oracle VirtualBox Manager. The frequency of the controller on the UAV is $100Hz$, and the NMPC frequency is $20Hz$. For the NMPC the prediction horizon $Tf = 10$, the number of discretization steps $N = 50$. The maximum simulation time is $200[s]$, with a break statement if the UAV reaches the end point. In table 10 is the options chosen for the OCP solver.

Quadratic programming solver	*PARTIAL_CONDENSING_HPIPM*
nonlinear problem solver type	*SQP_RTI*
Hessian approx	*GAUSS_NEWTON*
Integrator type	*IRK*
Sim method number stages	4
Sim method number steps	3
tolerance	$1e-4$

Table 10: OCP solver options

5.3.2 Intial states

At every simulation, the UAV starts in trim condition, where the values are seen in table 11

From the repository a function to find the trim conditions for the UAV is used. Input is wanted circle radius, airspeed, course angle and sideslip angle. The turning circle is infinity, meaning no turning, airspeed is $18[m/s]$ and course and gamma is set to zero. This gives the following initialized states:

state	value	
u	18.9	[m/s]
v	1.05	[m/s]
w	0.584	[m/s]
Roll	0.00	[deg]
Pitch	1.76	[deg]
Yaw	-3.18	[deg]
p	0.00	[deg/s]
q	0.00	[deg/s]
r	0.00	[deg/s]
Va	18.0	[m/s]
beta	0.00	[deg]
alpha	1.76	[deg]
aileron	0.00	[deg]
elevator	2.10	[deg]
rudder	0.00	[deg]
throttle	0.121	[-]

Table 11: Trim condition

5.3.3 Wind

All the simulations are simulated with wind at a turbulence level 2.

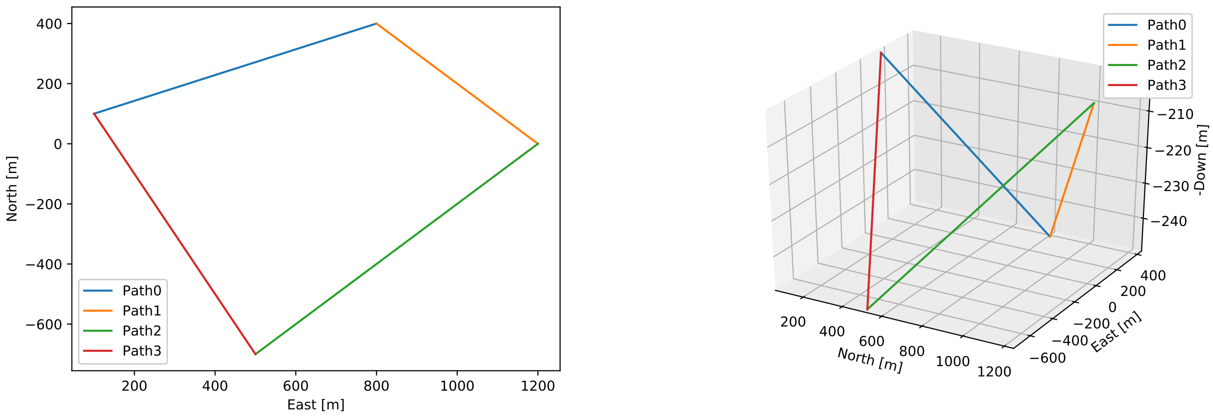
5.4 Straight line path

In this section, the results for the straight-line path are presented. The straight-line path, seen in section 4.2.4, is four waypoints that make up a rectangular pattern. These four waypoints are also used in the making of the curved path, seen in section 4.2.5. Waypoints $\mathbf{w}_{\{i\}}$ are expressed in $\mathcal{F}^i$, is defined in table 12 and are used for all the straight-line path simulations.

i	N	E	D
1	100	100	-200
2	400	800	-250
3	0	1200	-200
4	-700	500	-250

Table 12: Waypoints

In figure 32 the path is shown



(a) NE-frame

(b) NED-frame

Figure 32: Straight-line pattern

For the straight line, the VFB controller is used to compare against the two NMPC. Start with the two NMPC then the VFB. The simulation results of all the controller's performance in NE- and NED frame is seen in figure 42.

The switch and initializing for the two NMPC can be seen in section 4.2.4.

5.4.1 Kinematic model in NMPC with low-level autopilot

This controller uses the second-order kinematic model described in section 4.2.2 with the model parameter found in table 9 with the NMPC defined in section 4.2.1. From equation 91 the two gain matrix $\mathbf{Q}$ and $\mathbf{R}$ must be defined. The higher the value on element in matrix $\mathbf{Q}$, the more the system punishing an error. Finding the values was done by trial and error. The overall goal is to reduce the error on the path. Intuitive this would mean high value on the element penalize the e_n, e_e and e_d. However, seen in equation 221 the element on error airspeed is the highest. The airspeed is the only error state that should hold a constant value. Moreover, have the smallest error compared to the other errors. For example, the error in north and east has over $100m$ as intial error, with error in down constantly oscillation around $0m$, see figure 33a. The path parameter z_1 has an initial error of 1, see figur 33b. This means that to minimize the cost function, eqation 91, described in section 4.2.1, the controller should focus on the other errors. Also, note that the feedback to the kinematic model NMPC is the internal airspeed state, meaning that the airspeed is not affected by the wind as the airspeed for the UAV. Therefore it should be relatively easy for control to keep a constant value for airspeed. There, this element on $\mathbf{Q}$ is so high. Same argument for why the element on e_d is higher than for e_n and e_d. The element on the path parameter is reasonably high. The gain matrix on the states are

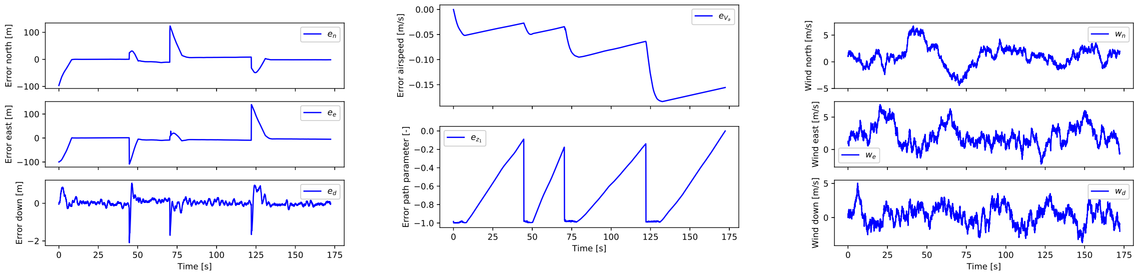
$$\mathbf{Q} = diag\begin{bmatrix}1e1 & 1e1 & 1e3 & 5e6 & 1e1\end{bmatrix} \tag{221}$$

which penalize the e_n, e_e, e_d, $V_a - V_{a,ref}$ and z_1, respectively. For the matrix $\mathbf{R}$ high value on elements would mean that this input is expensive to use. In equation 222 shows that the input on pitch and yaw is costly. This high price is because if it were cheaper, these angles would oscillate. So these high values are there to prevent this high oscillation. The last element in equation 222 is the virtual input. This input is the acceleration for the path parameter on the path. If this value is too high, the path parameter will move too slow compared to the UAV, expensive to travel. Therefore it is set as reasonably cheap. The gain matrix on the input is.

$$\mathbf{R} = diag\begin{bmatrix}1e1 & 1e4 & 1e3 & 1e-2\end{bmatrix} \tag{222}$$

penalize the V_a^c, θ^c, ψ^c and υ,respectively. For the kinematic model a disturbance observer, describes on section 4.2.2, where the learning rate is 0.002. The control architecture can be seen in figure 16.

The errors in the OCP is seen in figure 33. In figure 33a the error is between UAV and the path. Here the spikes are because of a change in waypoints. It can be seen that the errors converge to zero. In figure 33b is the airspeed and path parameter errors. For the airspeed, the error is not converging to zero. To avoid this, could increase the airspeed element on the gain matrix $\mathbf{Q}$ even more. This is not done because of two reasons. First, the UAV dos not manage to follow the airspeed reference exactly, so an error of $0.15\frac{m}{s}$ is not that important. Second, by increase, the element in $\mathbf{Q}$, the position error is less prioritized. Looking at the gain matrix in equation 221 this element is already high. Overall, following the path is considered more important. For the path parameter, from section 4.1, is driven to zero. This is achieved in figure 33b. Note that the value only reaches zero at the end. This is because the switch mechanism uses this z_1 value to change waypoints. In figure 33c is the wind used in the simulation.



(a) North, east and down error

(b) Airspeed and path parameter error

(c) Simulated wind

Figure 33: Simulation results for the kinematic model in the NMPC for straight line path

5.4.2 Dynamic model in NMPC

This model is described in section 4.2.3. Also, here the gain matrix needs to be defined. Here the element on e_n and e_d are low compared to the others. If these elements were higher, the constraint would be violated. They are therefore reasonably low. The element on e_d is the highest and is therefore prioritized the most. Compared to the kinematic model, the element on airspeed error is much less. In the dynamic model, the UAV airspeed is feedback, and hence not constant, see figure 34b. Therefore this is also reasonably low together with the element on the path parameter. The gain matrix on the states are

$$\mathbf{Q} = diag\begin{bmatrix}1e-2 & 1e-2 & 1e1 & 1 & 1\end{bmatrix} \tag{223}$$

which penalize the e_n, e_e, e_d, $V_a - V_{a,ref}$ and z_1, respectively.

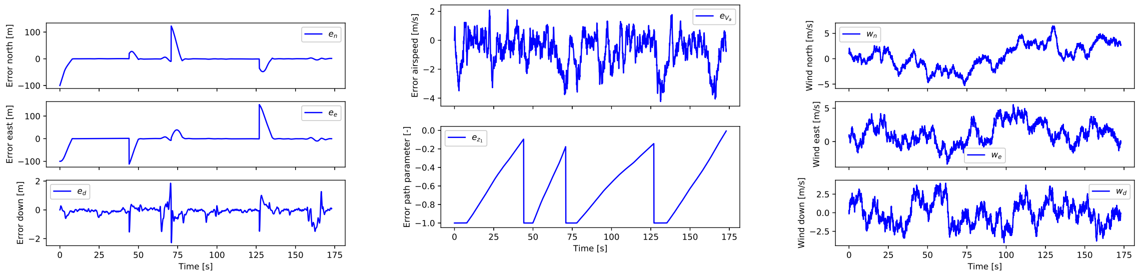
For the matrix $\mathbf{R}$ the element is resonable cheap. The rudder is not used.

$$\mathbf{R} = diag\begin{bmatrix}1 & 1 & 1e-3 & 1e-1 & 1e-1\end{bmatrix} \tag{224}$$

penalize the $\dot{\delta}_e$, $\dot{\delta}_a$, $\dot{\delta}_r$, $\dot{\delta}_t$ and v, respectively. The control architecture can be seen in figure 17.

The OCP errors can be seen in figure 34. Also here the position errors converge to zero, seen in figure 34a. There are some oscillation for errors in north and east.

For the airspeed error in figure 34b this oscillates much more than seen in the second-order kinematic model NMPC. This is because here, the UAV airspeed is used in feedback to the NMPC, and hence the NMPC uses this to find the optimal throttle input which is applied to the UAV. The kinematic model uses internal NMPC states as feedback and is, therefore, oscillates much less. To compare the airspeed error for the kinematic and dynamic, the low-level airspeed control must be used for the kinematic model. In figure 34c is wind used in the simulation.



(a) North, east and down error

(b) Airspeed and path parameter error

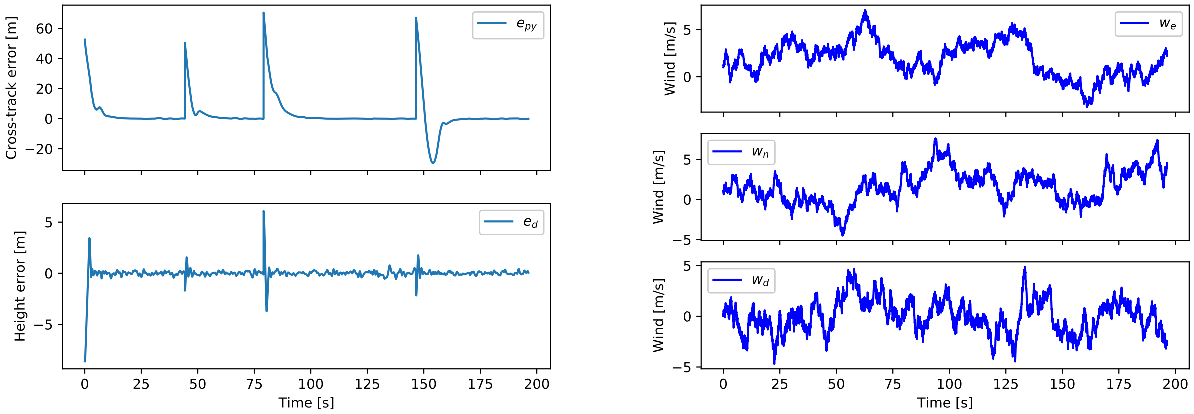
(c) Simulated wind

Figure 34: Simulation results for the dynamic model in the NMPC for straight line path

5.4.3 VFB controller

This controller is described in section 4.4.2 and is used to compare against the two NMPC for the straight-line path. The control architecture can be seen in figure 20.

The error can be seen in figure 35a. Here the error is the cross-track error defined in figure 21 which is the orthogonal distance to the path. This error can be seen converging to zero. The spikes are because of the waypoints change. Can also see the error in height converging and oscillating around zero. In figure 35b is with the simulated wind.



(a) Cross-track and height error

(b) Simulated wind

Figure 35: Simulation results for VFB controler for straight line path

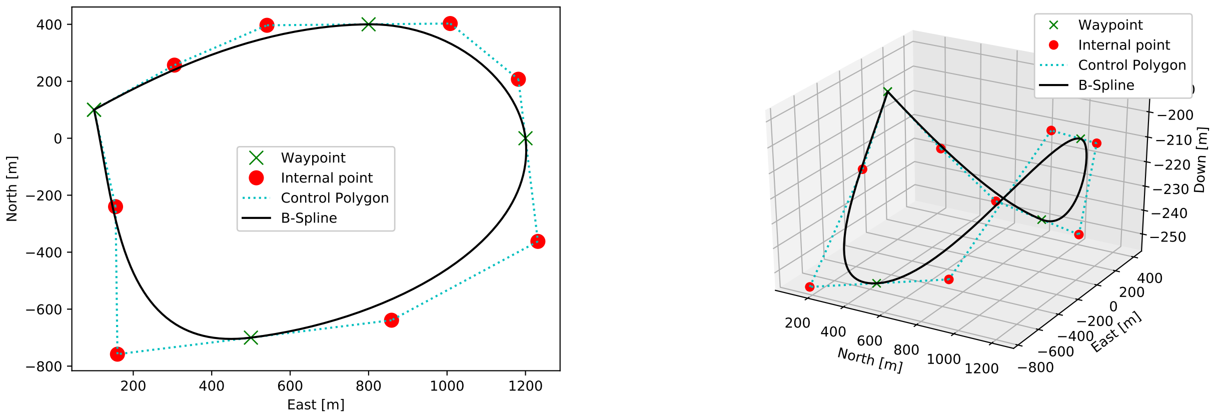
5.5 Curved path

Here the curved path results are presented. The curved path is defined by a B-spline where the internal control points are

i	N	E	D
1	256.74	304.97	-224.11
2	396.57	540.89	-246.28
3	402.75	1008.09	-252.97
4	207.21	1181.91	-207.32
5	-362.13	1231.61	-187.19
6	-638.83	858.50	-243.86
7	-758.06	159.67	-255.82
8	-240.20	154.87	-221.94

Table 13: Internal controll points

This is used for all the curved path simulations. In figure 36 the path is shown



(a) NE-frame

(b) NED-frame

Figure 36: Curved path

For the curved path, a NDGPFG is used to compare against the two NMPC. Start by looking at the gc, and then the two NMPC. The simulation results of all the controllers performance in NE- and NED frame is seen in figure 47.

The initializing for the two NMPC can be seen in section 4.2.5.

5.5.1 Kinematic model in NMPC with low-level autopilot

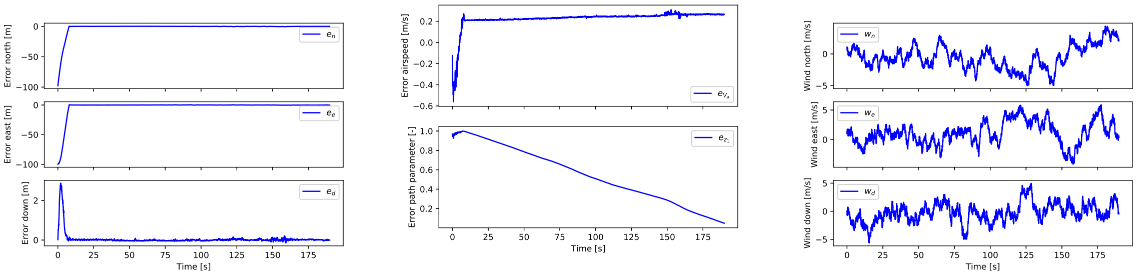
This controller uses the second-order kinematic model. For the gain matrix $\mathbf{Q}$, the elements on e_n and e_e are much higher compared to the straight line. This is because it is expected that the errors will be smaller in the north and east because of the continuous path. See figure 33a compared to 37a. The element on the path parameter is less, and this is because the whole path is longer, meaning it included the whole path compared to the straight line, which is just segments. To drive the path parameter to zero is achieved after a longer time. The gain matrix on states are

$$\mathbf{Q} = diag\begin{bmatrix}1e3 & 1e3 & 1e4 & 1e5 & 1e-1\end{bmatrix} \tag{225}$$

which penalize the e_n, e_e, e_d, $V_a - V_{a,ref}$ and $z_1 - z_{1,ref}$, respectively. For the gain matrix, $\mathbf{R}$ the value on the elements pitch and yaw are less, meaning that it is a more aggressive controller, with more oscillation. Also, the virtual input v is more expensive compared to the straight line. This is because it is wanted to have less acceleration on path parameter.

$$\mathbf{R} = diag\begin{bmatrix}1e1 & 1e3 & 1e2 & 1\end{bmatrix} \tag{226}$$

penalize the V_a^c, θ^c, ψ^c and v. The error plots can be seen in figure 37. Note that for the curved path, there are no spikes because of changes in waypoints. This is seen for the errors in figure 37a and the path parameter in figure 37b. For the errors in north and east, the error starts with a big error and converges to zero. For the error in down (height), there is a spike of over $2m$ in the beginning. For the NMPC to find an optimal solution, the NMPC minimize the cost function more by reducing the errors in north and east, compared to the error in down. It can be seen that almost all errors converge to zero at once. So this could be the reason for the spike at the beginning for the down error. Again, the kinematic model airspeed error does not converge to zero, see figure 37b. The argument is the same here since this value is so small and that the low-level airspeed control does not exactly follow the reference, this steady-state error is accepted. The path parameter is driven to 1 for the curved path. So the error, $z_1 - 1$ is driven to zero, seen in figure 37b. In figure 37c is the wind used in the simulation.



(a) North, east and down error

(b) Airspeed and path parameter error

(c) Simulated wind

Figure 37: Simulation results for the kinematic model in the NMPC for curved path

5.5.2 Dynamic model in NMPC

Comparing to the straight-line path the elements on e_n and e_e is higher. This to prevent oscillation that occurs on the path. The element on the path parameter is less compared to the straight-line. Here the same arguments as for the reason on the kinematic model for the length of the path compared to the straight line. The gain matrix on states are

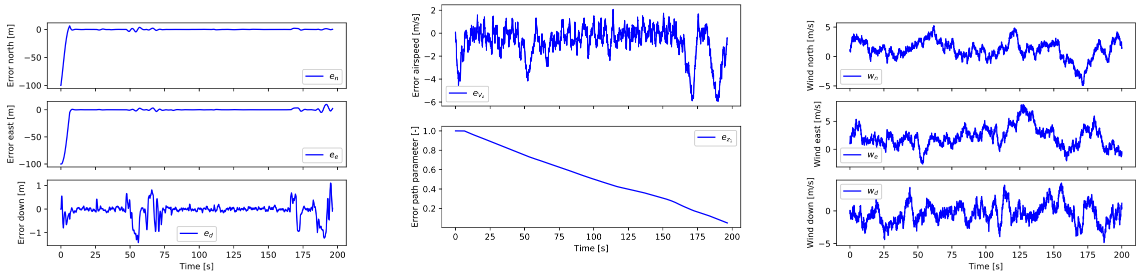
$$\mathbf{Q} = diag\begin{bmatrix}3e-2 & 3e-2 & 1e1 & 1 & 1e-1\end{bmatrix} \tag{227}$$

which penalize the e_n, e_e, e_d, $V_a - V_{a,ref}$ and $z_1 - z_{1,ref}$, respectively. For the gain matrix $\mathbf{R}$ this is almost the same as for the straight line, with the virtual input v more expensive. If wanted less acceleration on the path, the same argument as above about the length of the path compared to the straight line.

$$\mathbf{R} = diag\begin{bmatrix}1 & 1 & 1e-3 & 1e-1 & 1\end{bmatrix} \tag{228}$$

penalize the $\dot{\delta}_e$, $\dot{\delta}_a$, $\dot{\delta}_r$, $\dot{\delta}_t$ and v,respectively. The error plots can be seen in figure 38. There is some oscillation for the errors in north, east and down. The straight-line path with the dynamic model also had this oscillation. The error in airspeed, in figure 38b, is excepted.

Even though the two big spikes at around $175s$ and $190s$ are a bit high. In figure 38c is the wind used in the simulation.

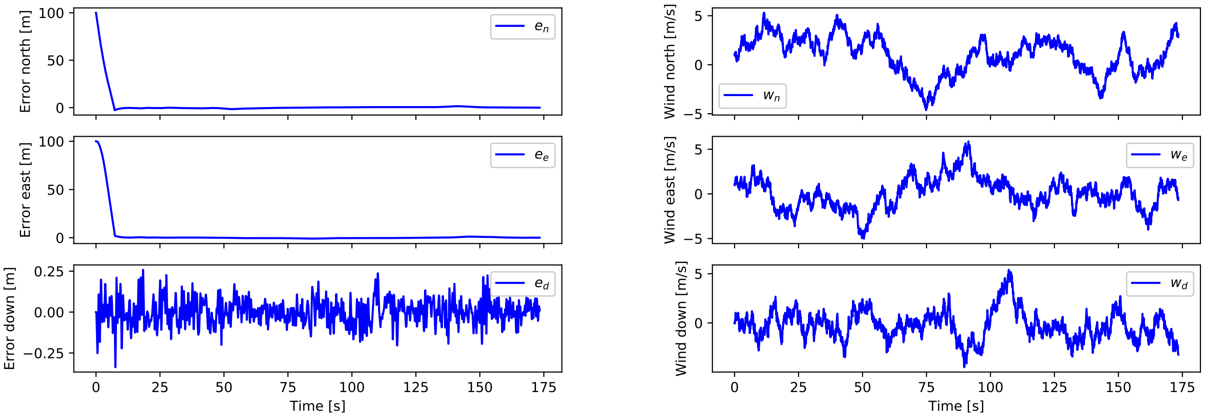


(a) North, east and down error (b) Airspeed and path parameter error (c) Simulated wind

Figure 38: Simulation results for the dynamic model in the NMPC for curved path

5.5.3 NDGPFG

This controller is described in section 4.4.3 with the control architecture seen in figure 25. It is used to compare against the two NMPC for curved path. The boundary-laywer thickness $\delta_{BL} = 50$ and guidance gain $k = 0.09$. The error plot can be seen in figure 39a, where all the errors converge to zero. In figure 39b is the wind used in the simulation.



(a) North, east and down error (b) Simulated wind

Figure 39: Simulation results for the gc for curved path

5.6 Discussion

5.6.1 Why use internal state vs. UAV states in the kinematic model NMPC

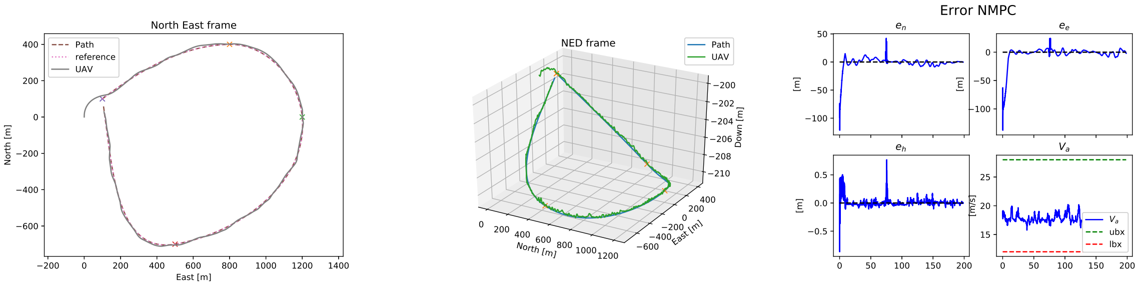
In this control architecture, there is an autopilot in the lower-level loop to track the reference from the NMPC. Feedback to the NMPC is the updated error and virtual state z_1. This means that the position of the UAV is feedback to the controler, which again updates the virtual states and errors.

For pitch and yaw, the internal NMPC states are feedback. The reason for this is because the frequency used in the NMPC and autopilot is not the same. This means that there is an overshoot when using the UAV state as feedback, see figure 40. Looking at the NE-frame at figure 40a the UAV is oscillating on the path, where the yaw angle has overshoot. The error in down is good, which can be seen in figure 40b and the errors in figure 40c.

The errors e_n and e_e are oscillating around zero. Note that the big spike is because of the mapping of the yaw angle in $\psi \in (-180, 180)$, because the UAV range is $\psi^{UAV} \in (-180, 180)$. Therefore, this error spikes happening at $80s$ is more a cosmetic error.

Looking at the figure 27b the yaw-controler can not follow the reference precisely because of the wind, where the yaw angle is oscillating. Because of that, this motivates the internal NMPC state feedback instead of UAV state feedback.

To not have state feedback to the control is only valid when there is assumed no parametric disturbance.



(a)

(b)

(c)

Figure 40: UAV state feedback

5.6.2 Performance measurements

When looking at the performance, the following error measurements are used:

$$J_{e_i} = \frac{1}{T} \sum_{0}^{T} |e_i| dt \tag{229}$$

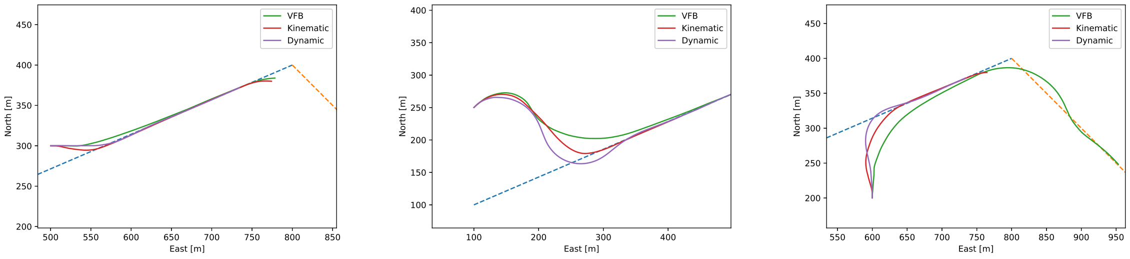
where dt is the time step of the simulation and T is simulation time. This is repeated for both the use of actuators and Euler angles error for the autopilot. Since the NMPC do not have a height controler, this is not used for comparison. The two following equation are for the actuators and Euler angles

$$J_{\delta_i} = \frac{1}{T}\sum_{0}^{T} |\delta_i| dt \tag{230}$$

$$J_{e_{\Theta_i}} = \frac{1}{T}\sum_{0}^{T} |e_{\Theta_i}| dt \tag{231}$$

5.6.3 Straight line

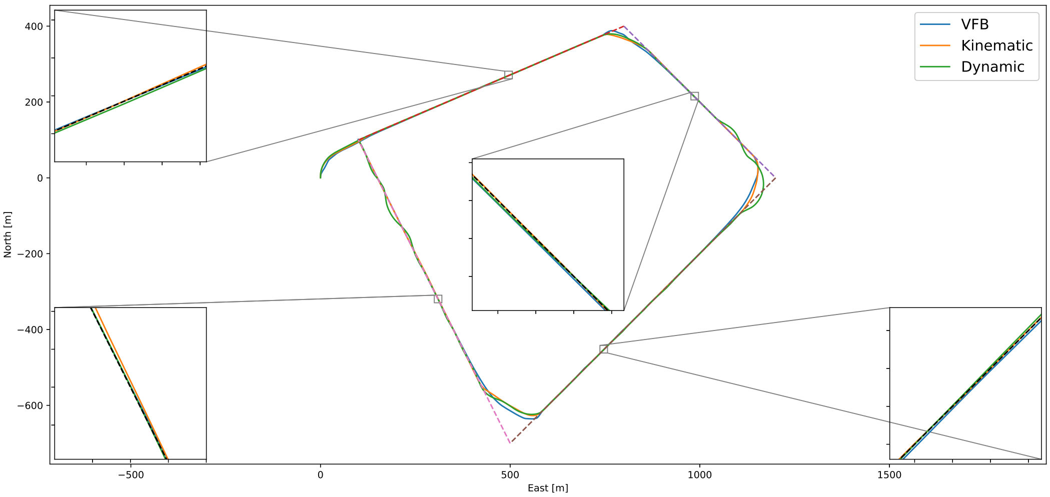
Converge to path with different start position
Test all the controllers to converge to path from different start positions. This can be seen in figure 41. Here, the VFB converge the slowest for all the starts and the kinematic the fastest. The dynamic has overshoot at start 2 and 3 compared to the kinematic and VFB that has no overshoot. The VFB for start 3 does not converge to path because it changes path segment before touching onto the path.



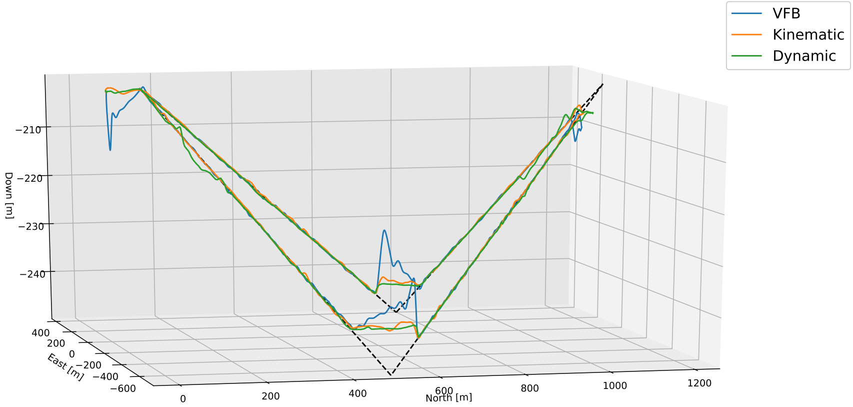
(a) Start 1 (b) Start 2 (c) Start 3

Figure 41: Converge to straight-line path for different start configurations

Simulation study and compare performance
For the straight-line path, all the controler performed well. The simulated results for the path can be seen in figure 42. In figure 42a it can be seen in the zoom windows that all the controllers are very close to the path. Looking at the NED frame, in figure 42b, the VFB controller is off in height at the switch compared to the two NMPC who performs better at the switch. It can also be seen that the dynamic controler is oscillating just before waypoint 3 and at the end of the simulation. This can also be seen in figure 42b and 38a.



(a) NE-frame

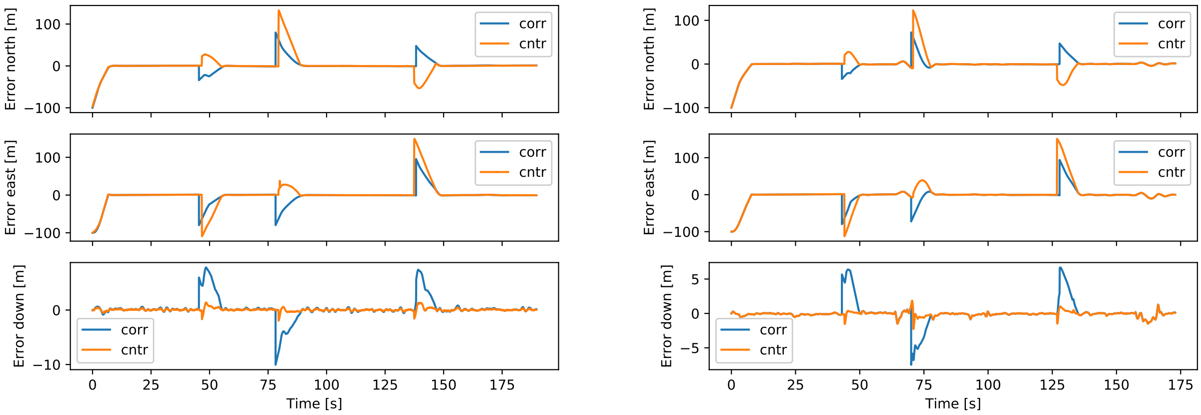


(b) NED-frame

Figure 42: Simulation results for straight-line path for VFB and the two NMPC

Comparing the error between VFB and the two NMPC is difficult because the VFB use the cross-track error while the NMPC uses error in north, east and down. Another point is that at the switch, the start error for the VFB is less than the start for the two NMPC. This is seen in figure 33a , 34a and 35a. This is because the start point on the line for the NMPC is placed further onto the path. Which is described in section 4.2.4 for the switch. Therefore, this initial error after switch for the two NMPC is bigger. This is done so that the path is cut within the predetermine fillet radius for a smoother convergence.

To have a better error comparison, looping through the data and using the path parameter initializing to find the closest point to path, the error in north, east and down can be found. Comparing the controller error and corrected error for the two NMPC can be seen in figure 43. One can see for both the kinematic and dynamic that the initial error after the switch is much smaller. Note that since the VFB do not have an error in north and east, this comparison is not included in figure 43.

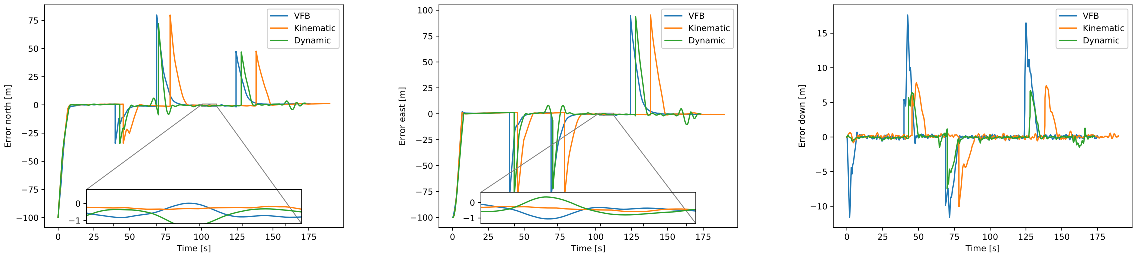


(a) Kinematic model NMPC

(b) Dynamic model NMPC

Figure 43: The control error vs the corrected error for straight-line path

Now comparing the corrected kinematic and dynamic error with the VFB corrected error can be seen in figure 44. Here the initial errors after the switch are the same, and the comparison is much better.



(a) Error north

(b) Error east

(c) Error down

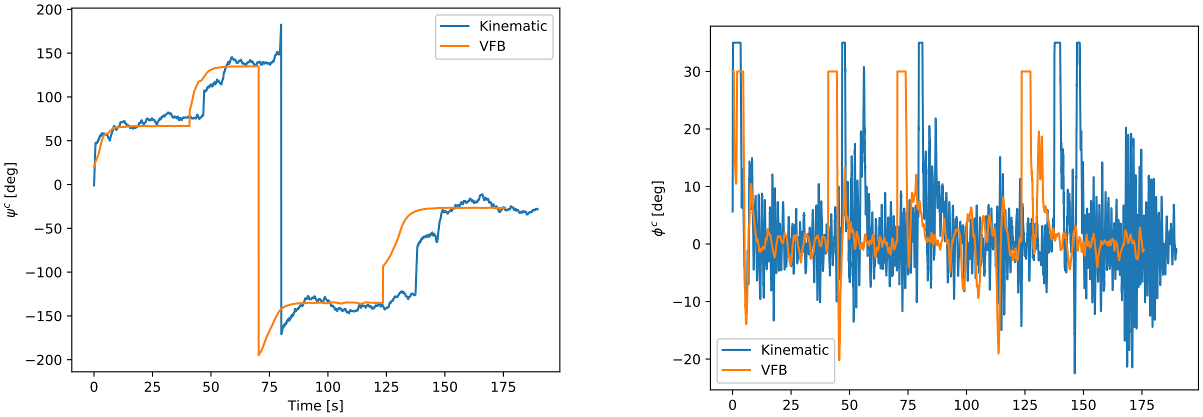
Figure 44: Comparing the corrected error for VFB and the two NMPC for straight-line path

Looking at the performance error, J_{e_i}, in table 14 the errors for the VFB is the highest for north, east and down. The dynamic has the smallest error in all.

Var	VFB	NMPC Kinematic	NMPC Dynamic
J_{e_n}	6.436	5.924	5.250
J_{e_e}	8.642	8.234	7.798
J_{e_d}	1.397	0.8241	0.6523

Table 14: Position error performance for the controllers at straight-line path in m

Both the VFB and kinematic model NMPC use the same lateral autopilot, while different for the longitudinal. Comparing the lateral, the VFB has a course command, while NMPC heading. In figure 45a the two different commanded angels given to the autopilot is seen. For the VFB the course angel is more stable, with constant reference. For the NMPC the reference is oscillating more, and this affects the commanded roll angle in figure 45b, where the oscillating is much more than the VFB. This is also seen in table 15. Her error values are much less for the VFB. Also, note that the error value for yaw/course is much higher compared to the pitch and roll. This is also seen in figure 27b and 28b that the autopilot is struggling more to follow reference on yaw.



(a) Commanded course angle for VFB and heading for NMPC

(b) Commanded roll angle

Figure 45: Comparing the commanded angles from the VFB and the kinematic model NMPC for straight line path

Var	VFB	NMPC Kinematic
J_{e_θ}	1.532	6.270
J_{e_ψ}	13.55	22.30
J_{e_ϕ}	0.7815	9.839

Table 15: Autopilot error performance for straight line path in deg

In table 16 is the use of actuators. The dynamic use less throttle and elevator, while the VFB uses less aileron. For the kinematic NMPC the high oscillating commanded angles for the kinematic is the reason for the much more use of aileron and also an elevator.

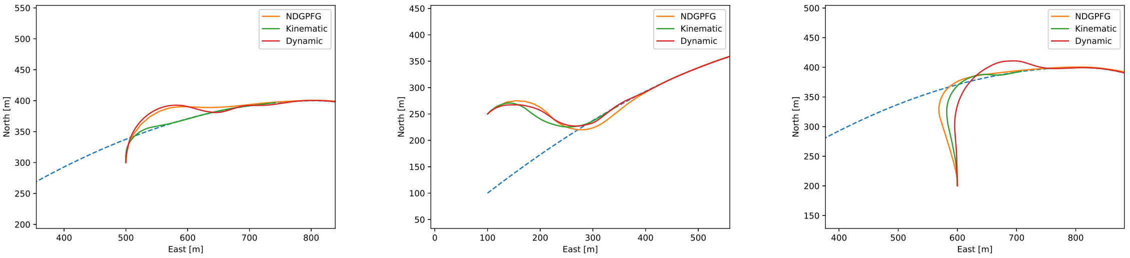
Var	VFB	NMPC Kinematic	NMPC Dynamic
$J_{u_{\delta_a}}$	1.603	23.15	2.076
$J_{u_{\delta_e}}$	3.233	12.64	2.717
$J_{u_{\delta_t}}$	0.1528	0.6100	0.1137

Table 16: Use of actuators for straight-line path in deg for aileron and elevator and dimensionless throttle

5.6.4 Curved path

Converge to path with different start position

In figure 46 the controllers all converge to path. Can see that for start 1, both the NDGPFG and the dynamic overshoot, while the kinematic has just a small overshoot onto the path. This is also repeated in start 2 where the kinematic first converge, then the dynamic, and last the NDGPFG with overshoot. In start 3, the NDGPFG goes almost perpendicular to the path. While the dynamic aims further to the right. Also, here the kinematic converge first, then the NDGPFG and last the dynamic.



(a) Start 1

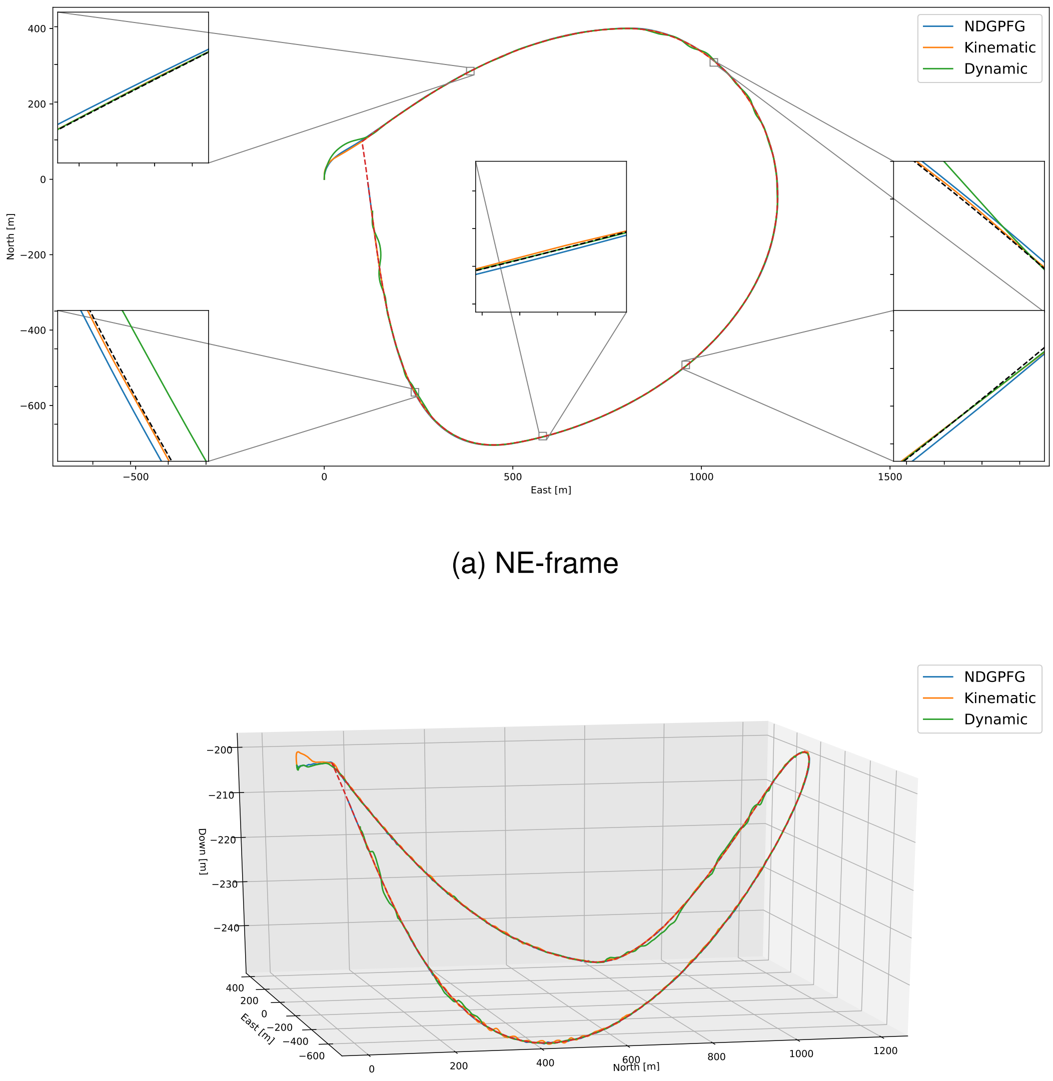
(b) Start 2

(c) Start 3

Figure 46: Converge to curved path for different start configurations

Simulation study and compare performance

Looking at the curved path and how the controllers worked can be seen in figure 47. Can be seen in figure 47a and figure 38a that the dynamic is oscillating on path on multiple cases. Simulation without wind removes this oscillation, so the effect of wind is causing this phenomenon. Both the kinematic and the NDGPFG is handling the wind better and are much tighter to the path. This is also seen in figure 48. Also, note that the dynamic is taking a long turn at the start, compared to the kinematic and NDGPFG. This is the same that occurred in figure 46 where the dynamic has a bigger turning than compared to the kinematic. This start can also be seen in error figure 48a where there is an overshoot for the dynamic.

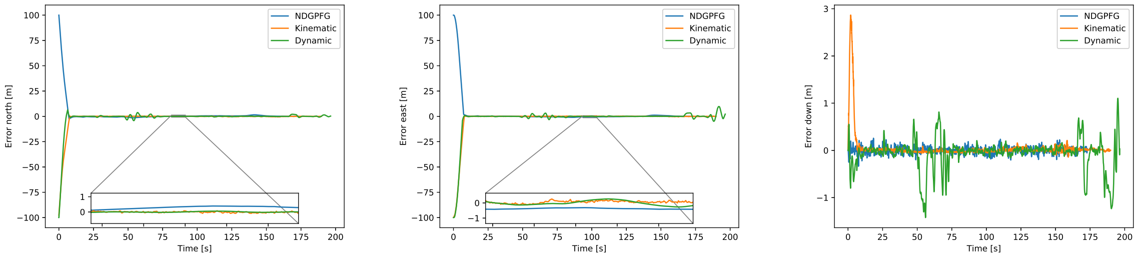


(a) NE-frame

(b) NED-frame

Figure 47: Simulation results for curved path

Looking at figure 48 it is easy to see the oscillation for the dynamic controler. This is seen at $50s$, $140s$, $170s$ and $190s$ in figure 48a and 48b. In down, compared to the NDGPFG and kinematic, the dynamic has some big errors. Where the value exceeds $1m$ at times.



(a) Error in north (b) Error in east (c) Error in down

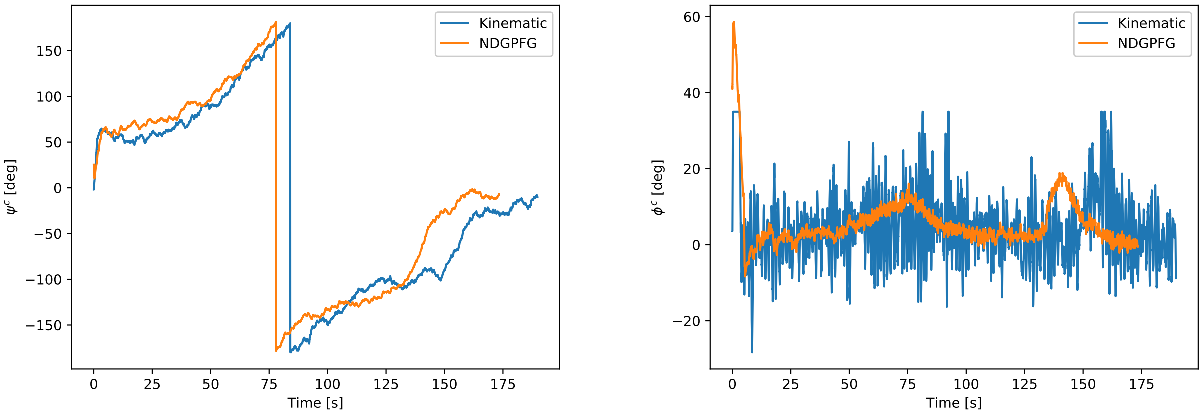
Figure 48: Compared error for the NDGPFG and the two NMPC for curved path

The performance error is seen in table 17. Although the oscillations, the dynamic NMPC had smaller errors than the NDGPFG for north and down. The kinematic had the smallest error for north and east, and NDGPFG for down.

Var	gc	NMPC Kinematic	NMPC Dynamic
J_{e_n}	2.275	1.790	1.615
J_{e_e}	2.995	2.475	2.741
J_{e_d}	0.057	0.0722	0.184

Table 17: Position error performance for the controllers at curved path in m

Also, the NDGPFG uses the same autopilot in the lateral direction, so comparing the commanded yaw angle against the NMPC is seen in figure 49. Also, here the commanded yaw angle is oscillation more than the NDGPFG, see figure 49a. Which again leads to more oscillation in the commanded roll, see figure 49b. Note also that the reason that the NDGPFG is finish before the NMPC, is because of the airspeed been different.



(a) Commanded yaw angle (b) Commanded roll angle

Figure 49: Comparing the commanded angles from the NDGPFG and the kinematic model NMPC for curved path

Looking at the angle errors in the autopilot, this is seen in table 18. Here it is easier to see that the errors for the kinematic are much higher than for the NDGPFG.

Var	VFB	NMPC Kinematic
J_{e_θ}	1.144	9.048
J_{e_ψ}	1.364	20.813
J_{e_ϕ}	1.2026	13.546

Table 18: Autopilot error performance for curved path in deg

This leads to the more use of actuators, which is seen in table 19. One can see that for aileron, the dynamic is using the less amount. The NDGPFG is using the least amount of elevator and throttle. The dynamic NMPC is the only controler where there are explicit constraints on the use of actuators. As it is seen in table 16 and 19 that the dynamic do not have drastic less use of these actuators. The goal in this thesis has not been to use the least amount of actuators. Hence, the gain matrix $\mathbf{R}$ seen in section 5.4.2 and 5.5.2 is not making the use of input expensive.

Var	gc	NMPC Kinematic	NMPC Dynamic
$J_{u_{\delta_a}}$	1.951	31.82	1.929
$J_{u_{\delta_e}}$	2.370	12.05	2.690
$J_{u_{\delta_t}}$	0.0338	0.5478	0.1099

Table 19: Use of actuators for curved path in deg for aileron and elevator and dimensionless throttle

5.6.5 Complexity of path and models

The difference between the straight and curved path is that the curved path continues, meaning there is no need for a switch between line segments. For a MPC the optimal input is found over a time horizontal. In this time horizontal, the MPC do not know if a switch is inside this time horizon. Meaning that the optimal path continues beyond this switch point. This is illustrated in figure 50. Here, the thick black line is the actual movement, and the dotted lines are the evolution of the path over time horizontal. It is hard to see, but the line pointing in the northeast direction at the corner are dotted lines. This is where the optimal route goes beyond the switch point and the waypoint itself.

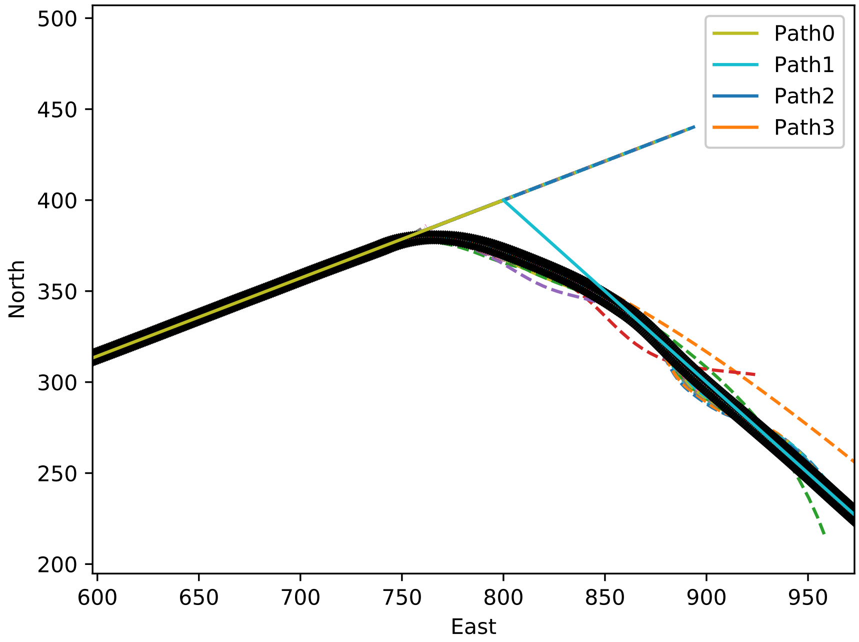
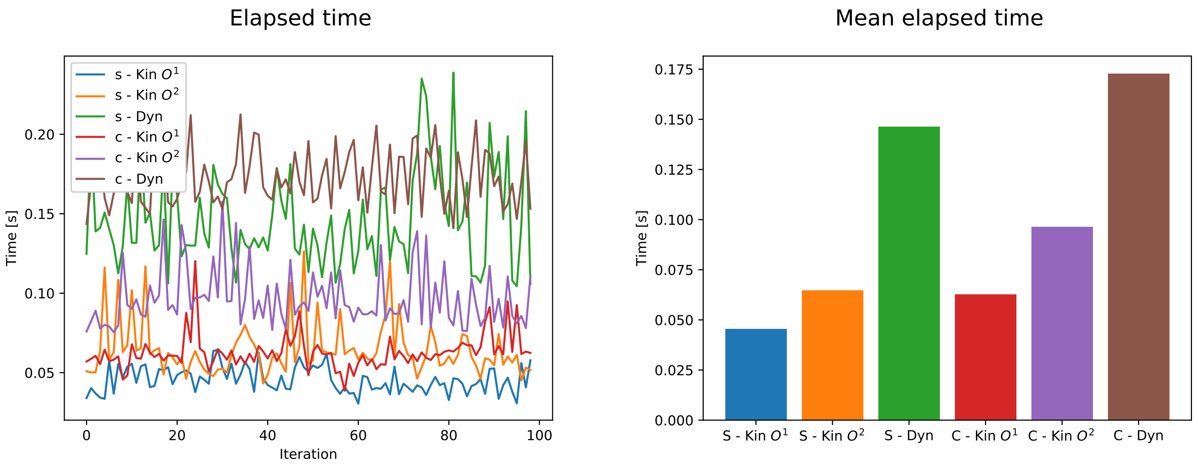


Figure 50: Optimal path straight ahead

From figure 42 the most significant deviation from path is at the switch. This is, of course, obviously when it is designed to cut corners, but in this corner-cutting, the controler is *free* to calculated the converge to the path. From table 14 and 17 can see that the error for the curved path is much lower, which is excepted.

There is also a difference in complexity between the straight line and the curved path. Calculating the jacobian of the straight-line path is easily done, even by hand. For the curved path, this needs to be evaluated at every iteration because the curves, meaning it is more complex. This is seen when looking at the elapsed time comparing the straight line and the curved, see figure 51. The mean time at each iteration is higher for the curved compared to the straight for the same model.

Within the NMPC the complexity of the dynamic model is higher than compared to the kinematic, which is simplified. This is also seen in the calculation time in figure 51 where the dynamic is using the most calculation time. Including in figure 51 is also a first-order kinematic model. So it is easy to see that the more complex, the more time. In this thesis, the NMPC is running at a frequency of $20Hz$, meaning every $0.05s$, which means that from figure 51b the only control that could have been used is the kinematic first-order model in NMPC. Note that this simulation time is very dependent on the system it is running on. So this does not mean that the controls using more time than $0.05s$ could not be used. This is just meant as an example.



(a) (b)

Figure 51: Elapsed time for the NMPC comparing the complexity of both the models and the straight vs curved path. S = straight, C = curved

The whole goal of the NMPC and the OCP is to minimize the cost function, equation 92. Can see that for all models, the cost function is minimized, which is expected.

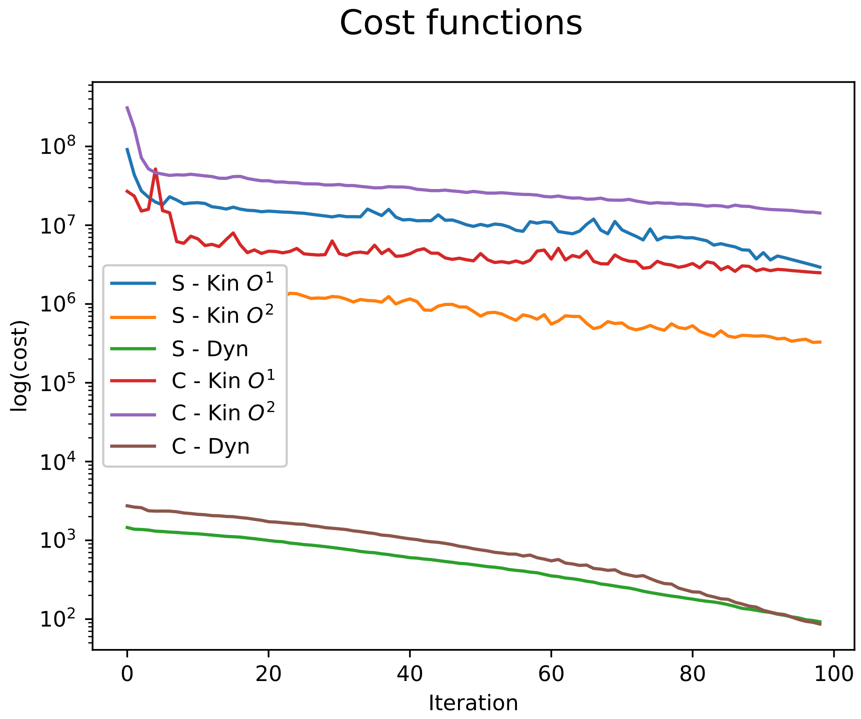


Figure 52: Cost

6 Conclusion

In this thesis, the main focus has been control algorithm design for path following for fixed-wing UAV influenced by wind. A solution based on a simplified kinematic model and a full dynamic model used in the NMPC to solve the path following problem and compared against a gc. The path has been straight and curved. Seen in section 5 that both NMPC outperformed the VFB controller on error from path in the straight line. This was also the case for the curved where the two NMPC outperformed the NDGPFG controller, except in height.

Also a mechanism to initialize the path variable and for switching between path segments has been has been successful made.

Kinematic model :

The kinematic is a more aggressive controller, where the use of actuators was much higher than in the dynamic and geometric controllers. This could lead to wear and tear on the actuators and is not wanted. With the high pitch and yaw angles oscillation, this could mean that the simplified model is not a good enough approximating for the UAV.

The error from path is good and outperforms the gc for both straight and curved path.

Dynamic model :

This controller did good in straight and had the lowest path error, and used acceptable actuators. The controller did have some unexpected oscillation, but they were small enough to be accepted. In the curved path, this controller oscillation several times, especially in height where the errors were greater than $2m$. This could be improved. If the overall goal is to example, save fuel or reduce the use of actuators, a higher penalizing on the $\mathbf{R}$ matrix could be done.

Future work

Kinematic model :

With the high use of actuators, a low-pass filter could be introduced to reduce this use. The simplified kinematic model could be further developed to minimize the high oscillation. In this thesis, the model identification was done on the closed-loop dynamic of UAV without wind. It could be tried to include wind in the model identification where the UAV gets commanded steps under wind conditions. Then the model could be fitted to the UAV closed-loop dynamic response. Could also include more validation sets to test the model, with more configuration of step inputs with different patterns and magnitude. An analysis of how the disturbance in the model affects the NMPC should also be conducted. This can describe how much impact a possible model noise makes the oscillation in states. Moreover, the simplified kinematic model structure could be changed. One could treat the airspeed as an input to the NMPC instead of a part of it. However, then the airspeed would have been constant over the time horizontal, and one of the beneficial properties of the optimal controller be gone. A different approach could be to include the dynamic airspeed controller with a kinematic model.

One could also use feedback from the UAV instead of the internal NMPC states.

Dynamic model :

In the curved path, this controller oscillation several times. This could be improved. This also happens, but at a smaller scale for the straight-line path. The gain matrix $\mathbf{Q}$ and $\mathbf{R}$ could have been tried to tune better. The best guess would be to try not to update the path parameter at every iteration and tune the path parameter z_1 in element in matrix $\mathbf{Q}$ and virtual input v in matrix $\mathbf{R}$ better. Then maybe a better behaviour on the path could be achieved.

Other things : For better comparisons, the controller could have been simulated with the same wind.

7 Attachments

7.1 Overview of controller in the repository UAVlab

All the controllers are found the repository UAVlab autofly in the branch *path_following_thomas*

Kinematic model NMPC

For straight:

main : Thomas_main_NMPC_kinematisk_straight.py

acados : acados_settings_kinematisk_straight.py

model : model_NMPC_kinematisk_straight.py

For curved:

main : Thomas_main_NMPC_kinematisk_bspline.py

acados : acados_settings_kinematisk_bspline.py

model : model_NMPC_kinematisk_bspline.py

Dynamic model NMPC

For straight:

main : Thomas_main_NMPC_dynamic_straight.py

acados : acados_settings_dynamikk_straight.py

model : model_NMPC_dynamisk_UAV.py

For curved:

main : Thomas_main_NMPC_dynamic_bspline.py

acados : acados_settings_dynamikk_Thomas_bspline.py

model : model_NMPC_dynamisk_UAV_bspline.py

VFB

main : Thomas_main_vector_field_path.py

NDGPFG

main :Thomas_main_curved_GUIDE.py

7.2 Skywalker X8 parameters

From [23] the values for the Skywalker X8 is collected, see table 20.

What	Value
ρ	1.2250
c	0.3571
S	0.7500
b	2.1000
C_{l_p}	-0.4042
C_{n_p}	0.0044
C_{m_q}	-1.3012
C_{m_α}	-0.4629
$C_{m_{\delta_e}}$	-0.2292
C_{D_0}	0.0197
C_{D_α}	0.0791
$C_{D_{\delta_e}}$	0.0633
S_{prop}	0.1018
C_{prop}	1
k_{motor}	40
J_x	0.335
J_y	0.140
J_{xz}	-0.029
m	3.3640

Table 20: Parameters Skywalker X8 [23]

7.3 Differentiate a rotation matrix

Example 7.1. *Differentiate a rotation matrix.*
To find the time differentiation of a rotation matrix, have that

$$\frac{d}{dt}R(t) = S(w(t))R(t) \tag{232}$$

Where, R *is a rotation matrix and* $S(\cdot)$ *is a skew symmetric matrix.*
Starting with rotation matrix R_v^b

$$\begin{aligned}\frac{d}{dt}R_v^b =& \frac{d}{dt}[R_{v2}^b(\phi)]R_{v1}^{v2}(\theta)R_v^{v1}(\psi) + R_{v2}^b(\phi)\frac{d}{dt}[R_{v1}^{v2}(\theta)]R_v^{v1}(\psi) + R_{v2}^b(\phi)R_{v1}^{v2}(\theta)\frac{d}{dt}[R_v^{v1}(\psi)]\\ =& S(\dot{\phi}\mathbf{i})R_v^b(\Theta) + S(R_{v2}^b(\phi)\dot{\theta}\mathbf{j})R_v^b(\Theta) + S(R_{v2}^b(\phi)R_{v1}^{v2}(\theta)\dot{\psi}\mathbf{k})R_v^b(\Theta)\\ =& [S(\dot{\phi}\mathbf{i}) + S(R_{v2}^b(\phi)\dot{\theta}\mathbf{j}) + S(R_{v2}^b(\phi)R_{v1}^{v2}(\theta)\dot{\psi}\mathbf{k})]R_v^b(\Theta)\end{aligned}$$

Here we have utilize that fact that $RS(a)R^T = S(Ra)$*. Using equation 232 we have that the rotation velocity is given by:*

$$\omega = \dot{\phi}\mathbf{i} + R_{v2}^b(\phi)\dot{\theta}\mathbf{j} + R_{v2}^b(\phi)R_{v1}^{v2}(\theta)\dot{\psi}\mathbf{k}$$

Where

$$\begin{aligned}\mathbf{i} =& [1 \quad 0 \quad 0]^T\\ \mathbf{j} =& [0 \quad 1 \quad 0]^T\\ \mathbf{k} =& [0 \quad 0 \quad 1]^T\end{aligned} \tag{233}$$

This represent the rotation rotate, so when say that i *is in the first part, this is because this rotation matrix is around x. So j is around y and k is around z.*
This gives:

$$\omega = \begin{bmatrix}\dot{\phi}\\0\\0\end{bmatrix} + R_{v2}^b(\phi)\begin{bmatrix}0\\\dot{\theta}\\0\end{bmatrix} + R_{v2}^b(\phi)R_{v1}^{v2}(\theta)\begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix}$$

7.4 Time differentiation a vector

Example 7.2. *Time differentiation a vector:*
Given a vector expressed in body, where i,j,k is defined in equation 233

$$\mathbf{p} = p_x\mathbf{i}^b + p_y\mathbf{j}^b + p_z\mathbf{k}^b \tag{234}$$

Then the time differentiation in the inertia frame, i, is

$$\frac{d}{dt_i}\mathbf{p} = \frac{d}{dt_b}\mathbf{p} + \omega_{b/i} \times \mathbf{p} \tag{235}$$

7.5 Relativ degree

Example 7.3. *Given the system*

$$\dot{x} = \begin{bmatrix} -x_1 + 2u \\ x_3 \\ -x_3 + x_2 x_3 + u \end{bmatrix} \tag{236}$$

$$y = x_2 \tag{237}$$

Rewrite the system as

$$\begin{aligned} \dot{\mathbf{x}} =& \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})u \\ y =& \mathbf{h}(\mathbf{x}) \end{aligned} \tag{238}$$

Then the matrix $\mathbf{f}(\mathbf{x})$ *and* $\mathbf{g}(\mathbf{x})$ *is defined as:*

$$\dot{x} = \underbrace{\begin{bmatrix} -x_1 \\ x_3 \\ -x_3 + x_2 x_3 \end{bmatrix}}_{\mathbf{f}(\mathbf{x})} + \underbrace{\begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}}_{\mathbf{g}(\mathbf{x})} \tag{239}$$

and $\mathbf{h}(\mathbf{x}) = x_2$.

$$\begin{aligned} L_f h =& \frac{\partial h}{\partial x}\dot{x} \\ =& \frac{\partial h}{\partial x} f + \frac{\partial h}{\partial x} g u \\ =& \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -x_1 \\ x_3 \\ -x_3 + x_2 x_3 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} u \\ =& x_3 \end{aligned} \tag{240}$$

$$\begin{aligned} L_f^1 h =& \frac{\partial L_f}{\partial x} f + \frac{\partial h}{\partial x} g u \\ =& \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -x_1 \\ x_3 \\ -x_3 + x_2 x_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} u \\ =& -x_3 + x_2 x_3 + u \end{aligned} \tag{241}$$

Here we see that input u *appears, and equation 77 holds, and hence the system has a relative degree* $\rho = 2$.

7.6 Kinematic model path followability

Rewrite the system as

$$\begin{aligned}\dot{\mathbf{x}} =& \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})u \\ y =& \mathbf{h}(\mathbf{x})\end{aligned}$$

Then the matrix $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ is defined as:

$$\dot{x} = \underbrace{\begin{bmatrix} V_a cos(\theta) cos(\psi) + w_n \\ V_a cos(\theta) sin(\psi) w_e \\ -V_a sin(\theta) + w_d \\ b_{Va}(-V_a) \\ b_\theta(-\theta) \\ b_\psi(-\psi) \\ z2 \\ 0 \end{bmatrix}}_{\mathbf{f}(\mathbf{x})} + \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ b_{Va} & 0 & 0 & 0 \\ 0 & b_\theta & 0 & 0 \\ 0 & 0 & b_\psi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{g}(\mathbf{x})} \underbrace{\begin{bmatrix} V_a^c \\ \theta^c \\ \psi^c \\ v \end{bmatrix}}_{\mathbf{u}}$$

with

$$y = h = \begin{bmatrix} p_n - p(z_1) & p_e - p(z_1) & p_d - p(z_1) & V_a & z_1 \end{bmatrix}^T$$

Where $p(z_1)$ is the path parameterization. Using Matlab the following Lie derivatives is calculated:

$$L_f h(x) = \begin{bmatrix} w_n - \mathbf{p}(z_1)_0 z_2 + V_a cos(\psi) cos(\theta) \\ w_e - \mathbf{p}(z_1)_1 z_2 + V_a cos(\theta) sin(\psi) \\ w_d - \mathbf{p}(z_1)_2 z_2 - V_a sin(\theta) \\ z_2 \\ V_a^c b_{V_a} - V_a b_{V_a} \end{bmatrix}$$

Here the input V_a^c appears in the last line. Note also that the the path $\mathbf{p}(z_1)_i$ the subscript is the index.

$$L_f^1 h(x) = \begin{bmatrix} V_a^c b_{V_a} A - V_a b_{V_a} A - p(z_1)_0 v - V_a b_\psi \psi^c B + V_a b_\psi \psi B - V_a b_\theta \theta^c C + V_a b_\theta \theta C \\ V_a^c b_{V_a} B - V_a b_{V_a} B - p(z_1)_1 v + V_a b_\psi \psi^c A - V_a b_\psi \psi A - V_a b_\theta \theta^c D + V_a b_\theta \theta D \\ V_a b_{V_a} s(\theta) - V_a^c b_{V_a} s(\theta) - p(z_1)_2 v - V_a b_\theta \theta^c c(\theta) + V_a b_\theta \theta c(\theta) \\ v \\ b_{V_a}^2 (V_a - V_a^c) \end{bmatrix}$$

where:

$$\begin{aligned} A =& c(\psi) c(\theta) \\ B =& c(\theta) s(\psi) \\ C =& c(\psi) s(\theta) \\ D =& s(\psi) s(\theta) \end{aligned}$$

Where $s()$ and $c()$ denote $sin()$ and $cos()$. Here input appears in the four other lines. This gives

$$\mathbf{r} = \begin{bmatrix} 2 & 2 & 2 & 2 & 1 \end{bmatrix}^T \tag{242}$$

Finding the matrix A given in equation 89

$$\mathbf{A}(x) = \begin{bmatrix} \frac{\partial}{\partial u_1} L_f^2 h_1(x) & \frac{\partial}{\partial u_2} L_f^2 h_1(x) & \frac{\partial}{\partial u_3} L_f^2 h_1(x) & \frac{\partial}{\partial u_4} L_f^2 h_1(x) \\ \frac{\partial}{\partial u_1} L_f^2 h_2(x) & \frac{\partial}{\partial u_2} L_f^2 h_2(x) & \frac{\partial}{\partial u_3} L_f^2 h_2(x) & \frac{\partial}{\partial u_4} L_f^2 h_2(x) \\ \frac{\partial}{\partial u_1} L_f^2 h_3(x) & \frac{\partial}{\partial u_2} L_f^2 h_3(x) & \frac{\partial}{\partial u_3} L_f^2 h_3(x) & \frac{\partial}{\partial u_4} L_f^2 h_3(x) \\ \frac{\partial}{\partial u_1} L_f^2 h_4(x) & \frac{\partial}{\partial u_2} L_f^2 h_4(x) & \frac{\partial}{\partial u_3} L_f^2 h_4(x) & \frac{\partial}{\partial u_4} L_f^2 h_4(x) \\ \frac{\partial}{\partial u_1} L_f^1 h_5(x) & \frac{\partial}{\partial u_2} L_f^1 h_5(x) & \frac{\partial}{\partial u_3} L_f^1 h_5(x) & \frac{\partial}{\partial u_4} L_f^1 h_5(x) \end{bmatrix}$$

Using Matlab the matrix is

$$\mathbf{A}(x) = \begin{bmatrix} b_{V_a} cos(\psi)cos(\theta) & -V_a b_\theta cos(\psi)sin(\theta) & -V_a b_\psi cos(\theta)sin(\psi) & -\frac{\partial p(z_1)}{\partial z_1} \\ b_{V_a} cos(\theta)sin(\psi) & -V_a b_\theta sin(\psi)sin(\theta) & V_a b_\psi cos(\psi)cos(\theta) & -\frac{\partial p(z_1)}{\partial z_1} \\ -b_{V_a} sin(\theta) & -V_a b_\theta cos(\theta) & 0 & -\frac{\partial p(z_1)}{\partial z_1} \\ 0 & 0 & 0 & 1 \\ b_{V_a} & 0 & 0 & 0 \end{bmatrix}$$

By inspection, the airspeed can not be 0, b_{V_a} is a positive constant, and the pitch angle θ is constrained to not be $\frac{\pi}{2}$. The jacobian of the path parameterization can not be zero, because it must be differenctable. The heading angle ψ is constrant to be $(-\pi, \pi)$. The only thing making matrix **A** rank deficient is if $\theta = \frac{pi}{2}$, which is infeasible.
Meaning that the system 112 is followable.

7.7 Path parameter initialize

```
def z1_init(p,w1,w2):
    p = np.array(p)
    r = np.array(w1)
    q_vec = np.array(w2)-np.array(w1)

    r = r.flatten()
    q_vec = q_vec.flatten()
    q_norm = LA.norm(q_vec)

    q = q_vec/q_norm

    chi_q = math.atan2(q[1],q[0])
    chi_q = SSA(chi_q)

    e_px = np.cos(chi_q)*(p[0]-r[0])+np.sin(chi_q)*(p[1]-r[1])

    return max(-1, -1 + e_px/q_norm)
```

7.8 Switch

```
def Switch(w1,w2,w3):
  R = 100

  diff_1 = np.array(w2-w1)
  diff_2 = np.array(w3-w2)

  norm_1 = LA.norm(diff_1.flatten())
  norm_2 = LA.norm(diff_2.flatten())

  q_mi = diff_1/norm_1
  q_i = diff_2/norm_2

  angle = np.arccos(-np.transpose(q_mi) @ q_i)

  pos_slutt = w2 - (R/np.tan(angle/2))*q_mi
  diff_pos_slutt = np.array(w1-pos_slutt)
  len_pos_slutt = LA.norm(diff_pos_slutt.flatten())

  return -1+len_pos_slutt/norm_1
```

7.9 B-spline

```
'''
    Retrieved from internet:
      https://github.com/casadi/casadi/issues/1484
'''

from casadi import if_else, logic_and
from abc import ABCMeta

class BSpline(object):
  """
  B-Spline base class.
  """

  __metaclass__ = ABCMeta

  def basis(self, t, x, k, i):
    """
    Evaluate the B-Spline basis function using Cox-de Boor recursion.

    Arguments:
    x -- The point at which to evaluate
    k -- The order of the basis function
    i -- The knot number
    """
    if k == 0:
      return if_else(logic_and(t[i] <= x, x < t[i + 1]), 1.0, 0.0)
    else:
      if t[i] < t[i + k]:
        a = (x - t[i]) / (t[i + k] - t[i]) * self.basis(t, x, k - 1, i)
      else:
        a = 0.0
      if t[i + 1] < t[i + k + 1]:
        b = (t[i + k + 1] - x) / (t[i + k + 1] - t[i + 1]) * self.basis(t, x, k - 1, i + 1)
      else:
        b = 0.0
      return a + b


class BSpline1D(BSpline):
  """
  Arbitrary order, one-dimensional, non-uniform B-Spline implementation using Cox-de Boor recursion.
  """
  def __init__(self, t, w, k=3):
    """
    Constructor.

    Arguments:
    t -- Knot vector
    w -- Weight vector
    k -- Spline order
    """

    # Store arguments
    self.t = t
    self.w = w
    self.k = k

  def __call__(self, x):
    """
    Evaluate the B-Spline at point x.

    The support of this function is the half-open interval [t[0], t[-1]).
    """
    y = 0.0
    for i in range(len(self.t) - self.k - 1):
      y += if_else(logic_and(x >= self.t[i], x <= self.t[i + self.k + 1]), self.w[i] * self.basis(self.t, x, self.k, i), 0.0)
    return y
```

```
import numpy as np
from numpy import linalg as LA
import math
import casadi as ca
from casadi import *

from Thomas_ctrl_bspline.BsplineDefs import *

def FindMellomPoints(P1,P2,T1,T2):

  P1 = P1.flatten()
  P2 = P2.flatten()

  T1 = T1.flatten()
  T2 = T2.flatten()
  a = 16 - LA.norm(T1 + T2)**2
  b = 12 * (P2-P1) @ (T1+T2)
  c = -36 * LA.norm(P2-P1)**2

  alpha = (-b+math.sqrt(b**2-4*a*c)) /(2*a)

  mel_P1 = P1 + 1/3*alpha * T1
  mel_P2 = P2 - 1/3*alpha * T2

  return [mel_P1,mel_P2]


def alpha(q00,qkk,q11,q22):
  '''
    q0 = q_k-1
    qk = q_k
    q1 = q_k+1
    q2 = q_k+1

  '''
  return LA.norm( np.cross(q00,qkk) ) / ( LA.norm( np.cross(q00,qkk) ) + LA.norm( np.cross(q11,q22) )  )



def T_u_v(alpha,qkk,q11):
  '''
    unit tangent vector of V_n
    qk = q_k
    q1 = q_k+1
  '''
  V = (1-alpha)*qkk + alpha*q11
  return V/LA.norm(V)


def PathGenerator(Q):
  '''
  Input:
    Q --> ndarray
  Output:
    3x SX.Function
  '''

  nQ = len(Q)

  q = dict()

  for k in range(1,nQ):
    q['{0}'.format(k)] = Q[k]-Q[k-1]

  q['0'] = 2*q['1'] - q['2']
  q['-1'] = 2*q['0'] - q['1']


  q[str(nQ)] = 2*q[str(nQ-1)] - q[str(nQ-2)]
  q[str(nQ+1)] = 2*q[str(nQ)] - q[str(nQ-1)]

  new_points = []

  xPoints,yPoints,zPoints  = [],[],[]

  xPoints.append(Q[0][0])
  yPoints.append(Q[0][1])
  zPoints.append(Q[0][2])

  u_hat = dict()
  u_hat[str(0)] = np.array([0,0,0])

  for k in range(0,nQ-1):
    a1 = alpha(q[str(k-1)],q[str(k)],q[str(k+1)],q[str(k+2)])
    T1 = T_u_v(a1,q[str(k)],q[str(k+1)])

    a2 = alpha(q[str(k)],q[str(k+1)],q[str(k+2)],q[str(k+3)])
    T2 = T_u_v(a2,q[str(k+1)],q[str(k+2)])

    [n1,n2] = FindMellomPoints(Q[k],Q[k+1],T1,T2)

    new_points.append([n1,n2])

    u_hat[str(k+1)] = u_hat[str(k)] + 3 * LA.norm(np.subtract(n1,Q[k]))

    for corr in zip([n1,n2]):
      xPoints.append(corr[0][0])
      yPoints.append(corr[0][1])
      zPoints.append(corr[0][2])

  xPoints.append(Q[nQ-1][0])
  yPoints.append(Q[nQ-1][1])
  zPoints.append(Q[nQ-1][2])

  U_x = []
  U_y = []
  U_z = []
  for j in range(2):
    U_x.append(0)
    U_y.append(0)
    U_z.append(0)

  for k in range(nQ):
    for j in range(2):
      U_x.append(u_hat[str(k)][0]/u_hat[str(nQ-1)][0])
      U_y.append(u_hat[str(k)][1]/u_hat[str(nQ-1)][1])
      U_z.append(u_hat[str(k)][2]/u_hat[str(nQ-1)][2])

  for j in range(2):
    U_x.append(1)
    U_y.append(1)
    U_z.append(1)

  x_arr = np.array(xPoints)
  y_arr = np.array(yPoints)
  z_arr = np.array(zPoints)

  x_knot = np.array(U_x)
  y_knot = np.array(U_y)
  z_knot = np.array(U_z)

  k = 3

  bsplineX = BSpline1D(x_knot,x_arr)
  bsplineY = BSpline1D(y_knot,y_arr)
  bsplineZ = BSpline1D(z_knot,z_arr)


  uc = ca.SX.sym('uc',1)

  bX = ca.Function('bX',[uc],\
       [bsplineX(uc)])

  bY = ca.Function('bY',[uc],\
       [bsplineY(uc)])

  bZ = ca.Function('bZ',[uc],\
       [bsplineZ(uc)])

  return [bX,bY,bZ]
```

7.10 PID-gains

```
import uav.models.X8 as model
import numpy as np
import math
P = model.P


def PitchGains(P,Va,integral=False):
    rho = P['rho']
    c = P['c']
    S = P['S_wing']
    Jy = P['Jyy']
    Cmq = P['C_m_q']
    Cma = P['C_m_alpha']
    Cmde = P['C_m_delta_e']
    delta_e_max = P['elevator_max']

    if integral:
```

```
    e=35
    zeta =0.707
  else:
    e = 15
    zeta = 1

  e_max = e * np.pi/180

  a_theta_1 = - (rho*Va**2*c*S)/(2*Jy) * Cmq * c/(2*Va)
  a_theta_2 = - (rho*Va**2*c*S)/(2*Jy) * Cma
  a_theta_3 = (rho*Va**2*c*S)/(2*Jy) * Cmde

  K_p_theta = delta_e_max/e_max * np.sign(a_theta_3)

  omega_theta = math.sqrt(a_theta_2 + K_p_theta*a_theta_3)
  K_d_theta = (2*zeta * omega_theta - a_theta_1) / a_theta_3
  K_DC = K_p_theta * a_theta_3 / (a_theta_2+ K_p_theta*a_theta_3)

  if integral:
    K_i_theta = -0.8
    K_d_theta = 0
  else: K_i_theta = 0

  return [K_p_theta,K_i_theta,K_d_theta,omega_theta,K_DC]


def HeightGaings(Va,omega_p,K_DC,bwsep = 10, zeta = 0.707):
  omega_h = 1/bwsep * omega_p

  K_p_h = 2*zeta*omega_h/(Va*K_DC)
  K_i_h = omega_h**2/(Va*K_DC)
  K_d_h = 0

  return [-K_p_h,-K_i_h,-K_d_h]


def RollGains(P,Va,zeta= 1.8, e = 15):
  rho = P[’rho’]
  S = P[’S_wing’]
  b = P[’b’]
  Clp = P[’C_l_p’]
  Cnp = P[’C_n_p’]
  Clda = P[’C_l_delta_a’]
  Cnda = P[’C_n_delta_a’]
  Jz = P[’Jzz’]
  Jxz = P[’Jxz’]
  Jx = P[’Jxx’]
  T = Jx*Jz - Jxz**2
  T3 = Jz/T
  T4 = Jxz/T
  Cpp = T3 * Clp + T4*Cnp
  Cpda = T3 * Clda + T4*Cnda
  delta_a_max = P[’aileron_max’]

  e_max = e * np.pi/180
```

```

    a_roll_1 = -1/2 * rho * Va**2 * S * b * Cpp * b/(2*Va)
    a_roll_2 = 1/2 * rho * Va**2 * S * b * Cpda

    K_p_r = delta_a_max/e_max * np.sign(a_roll_2)
    omega_r = math.sqrt(a_roll_2 * K_p_r)
    K_d_r = (2*zeta*omega_r - a_roll_1)/a_roll_2

    return [K_p_r,K_d_r,omega_r]


def YawGains(Vg,omega_r,bwsep= 20,zeta= 0.5):
    omega_y = 1/bwsep * omega_r
    g = 9.81

    K_p_yaw = 2* zeta * omega_y * Vg/g  #5
    K_i_yaw = omega_y**2 * Vg/g #1
    K_d_yaw = 0

    return [K_p_yaw,K_i_yaw,K_d_yaw]

def VaGains(P,Va_star,alpha_star,elevator_star,throttel_star,freq=0.5,zeta = 1):
    rho = P['rho']
    S_prop = P['S_prop']
    S = P['S_wing']
    m = P['mass']
    C_D_0 = P['C_D_0']
    C_D_alpha = P['C_D_alpha1']
    C_d_elevator = P['C_D_delta_e']
    C_prop = P['C_prop']
    k = P['k_motor']

    av1 = rho * Va_star * S / m * (C_D_0 + C_D_alpha * alpha_star + C_d_elevator * elevator_star) + rho * S_prop/m * C_prop * Va_star
    av2 = rho*S_prop/m * C_prop * k**2 * throttel_star
    omega = 2*np.pi * freq

    K_i_Va = omega**2/av2
    K_p_Va = (2*zeta*omega-av1)/av2
    K_d_Va = 0

    return [K_p_Va,K_i_Va,K_d_Va]
```

8 Bibliography

[1] Thomas Leirfall. Specialization project (ttk455) - model predictive controller for path-following, 2020.

[2] Equinor completes world's first logistics operation with a drone to an offshore installation - equinor.com. `https://www.equinor.com/en/news/20200828-drone-transport-troll.html`. (Accessed on 05/03/2021).

[3] Venanzio Cichella, Isaac Kaminer, Vladimir Dobrokhodov, Enric Xargay, Naira Hovakimyan, and Antonio Pascoal. Geometric 3d path-following control for a fixed-wing UAV on SO(3). In *AIAA Guidance, Navigation, and Control Conference*. American Institute of Aeronautics and Astronautics, jun 2011.

[4] Hassan K. Khalil. Nonlinear control, 2015.

[5] Wei Ren and R.W. Beard. Trajectory tracking for unmanned air vehicles with velocity and heading rate constraints. *IEEE Transactions on Control Systems Technology*, 12(5):706–716, 2004.

[6] Roger Skjetne, Thor I Fossen, and Petar V Kokotović. Robust output maneuvering for a class of nonlinear systems. *Automatica (Oxford)*, 40(3):373–383, 2004.

[7] Cunjia Liu, Owen McAree, and Wen-Hua Chen. Path-following control for small fixed-wing unmanned aerial vehicles under wind disturbances. *International Journal of Robust and Nonlinear Control*, pages n/a–n/a, dec 2012.

[8] Hongda Chen, Kuochu Chang, and Craig S. Agate. Uav path planning with tangent-plus-lyapunov vector field guidance and obstacle avoidance. *IEEE Transactions on Aerospace and Electronic Systems*, 49(2):840–856, 2013.

[9] R.W Beard and McLain T.W. *Small Unmanned Aircraft : Theory and Practice*. Princeton University Press, 2012.

[10] P.B. Sujit, Srikanth Saripalli, and Joao Borges Sousa. Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicless. *IEEE Control Systems Magazine*, 34(1):42–59, 2014.

[11] Thomas Stastny, Adyasha Dash, and Roland Siegwart. Nonlinear mpc for fixed-wing uav trajectory tracking: Implementation and flight experiments. 2017.

[12] Thomas Stastny and Roland Siegwart. Nonlinear model predictive guidance for fixed-wing uavs using identified control augmented dynamics. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, jun 2018.

[13] Timm Faulwasser. *Optimization-based solutions to constrained trajectory-tracking and path-following problems*. 01 2013.

[14] Jun Yang, Cunjia Liu, Matthew Coombes, Yunda Yan, and Wen-Hua Chen. Optimal path following for small fixed-wing UAVs under wind disturbances. *IEEE Transactions on Control Systems Technology*, 29(3):996–1008, may 2021.

[15] Francisco Gavilan, Rafael Vazquez, and Sergio Esteban. Trajectory tracking for fixed-wing UAV using model predictive control and adaptive backstepping. *IFAC-PapersOnLine*, 48(9):132–137, 2015.

[16] Andrea Alessandretti, A. Pedro Aguiar, and Colin N. Jones. Trajectory-tracking and path-following controllers for constrained underactuated vehicles using model predictive control. In *2013 European Control Conference (ECC)*. IEEE, jul 2013.

[17] Shulong Zhao, Xiangke Wang, Daibing Zhang, and Lincheng Shen. Model predictive control based integral line-of-sight curved path following for unmanned aerial vehicle. In *AIAA Guidance, Navigation, and Control Conference*. American Institute of Aeronautics and Astronautics, jan 2017.

[18] Alessandro Rucco, A. Pedro Aguiar, Fernando Lobo Pereira, and João Borges de Sousa. A predictive path-following approach for fixed-wing unmanned aerial vehicles in presence of wind disturbances. In *Advances in Intelligent Systems and Computing*, pages 623–634. Springer International Publishing, dec 2015.

[19] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Jonathan Frey, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados: a modular open-source framework for fast embedded optimal control. *arXiv preprint*, 2019.

[20] Mark Owen, Randal W. Beard, and Timothy W. McLain. Implementing dubins airplane paths on fixed-wing. In *Handbook of Unmanned Aerial Vehicles*, pages 1677–1701. Springer Netherlands, aug 2014.

[21] Thomas J. Stastny, Gonzalo A. Garcia, and Shawn S. Keshmiri. Collision and obstacle avoidance in unmanned aerial systems using morphing potential field navigation and nonlinear model predictive control. *Journal of Dynamic Systems, Measurement, and Control*, 137(1), aug 2014.

[22] Timothy Arney. Dynamic path planning and execution using b-splines. In *2007 Third International Conference on Information and Automation for Sustainability*. IEEE, dec 2007.

[23] Kristoffer Gryte, Richard Hann, Mushfiqul Alam, Jan Rohac, Tor Arne Johansen, and Thor I. Fossen. Aerodynamic modeling of the skywalker x8 fixed-wing unmanned aerial vehicle, 2018.

[24] Bjarne Foss and Tor Aksel N. Heirung. *Merging Optimization and Control*. 03 2016.

[25] Namhoon Cho, Youdan Kim, and Sanghyuk Park. Three-dimensional nonlinear differential geometric path-following guidance law. *Journal of Guidance, Control, and Dynamics*, 38(12):2366–2385, dec 2015.

[26] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, In Press, 2018.

[27] B.L. Stevens, F.L. Lewis, and E.N. Johnson. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. Wiley, 2015.

[28] Thor I Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley, Hoboken, 1. aufl. edition, 2011.