Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Guidance and Control of a Fixed Wing Aircraft

Autor: Luis F. Fernández González

Tutor: Begoña C. Arrue Ullés

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Guidance and Control of a Fixed Wing Aircraft

Autor:
Luis F. Fernández González

Tutor:
Begoña C. Arrue Ullés
Profesor titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Trabajo Fin de Grado: Guidance and Control of a Fixed Wing Aircraft

Autor: Luis F. Fernández González

Tutor: Begoña C. Arrue Ullés

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia y amigos

A mis maestros

Agradecimientos

A Begoña C. Arrue Ullés por avivar la curiosidad ingenieril, por su infinito apoyo y paciencia. Y por tan sabiamente desaconsejar la reinvención de la rueda.

Summary

The following document contains the development of a guidance and control system for small fixed-wing aircraft. The flight controller developed is based on the one mentioned in ([1] R. W. Beard and T. W. McLain, Small unmanned aircraft: theory and practice, Princeton University Press, 2012.) and, once finished, its functionality is tested in a simulation. A mathematical model of a well known and parameterized aircraft is developed for the simulation. The model includes equations for kinematic, dynamic, force and moment behaviour. The parameters of the autopilot controlling the aircraft are first tuned with linear models and then with the complete mathematical model. Finally, the entire system is tested in a virtual environment simulated in Matlab.

Abstract

Unmanned aerial systems (UAS) are playing increasingly prominent roles in defense programs and defense strategy around the world. And although civil and commercial applications are not as well developed, potential applications are extremely broad in scope, including environmental monitoring, border patrol, aerial surveillance and mapping, traffic monitoring, precision agriculture, disaster relief, ad hoc communications networks, and rural search and rescue. For many of these applications to develop to maturity, the reliability of UAS needs to increase, their capabilities extended further, their ease of use needs to be improved, and their cost must decrease [1].

For operation, UAS use a combination of hardware and software to control the aircraft during all phases of flight without the assistance of a pilot. This is called a flight controller. A flight controller for fixed-wing aircraft is developed in this document. Afterwards, a simulation is carried out to demonstrate its operation.

The developed flight controller must be able to control an aircraft so that it flights through a series of GPS positions. For the simulation, a photogrammetric session has been chosen because it includes takeoff and landing maneuvers and a flight with specific requirements. A photogrammetric session makes it easy to check that the autopilot is capable of fulfilling certain specifications, as it must be able to guide de aircraft so as to fly over the area to be photographed in a precise way.

This document follows a quick and practical approach to coordinate transformations, aerodynamics, autopilot design, state estimation and path planning without going into too much detail on these topics. The aim is, therefore, to slightly cover these wide range of topics, focusing in particular on their application to small fixed-wing vehicles.

To achieve greater cohesion, this document has been divided into two distinct parts: *Guidance and Control* and *Simulation*. In turn, each part is structured in: *Introduction*, *Development*, *Results* and *Conclusions and Improvements*.

Index

TABLE INDEX

FIGURE INDEX

Notation

ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle
α	Angle of attack
β	Sideslip angle

1. GUIDANCE AND CONTROL

1.1 Introduction to Guidance and Control

A guidance and control system must be able to determine the actuation on the control surfaces of an aircraft to navigate effectively through a series of waypoints. To achieve this, it must be first determined the path the aircraft must follow to get from one waypoint to another. Altitude, speed and course of the vehicle must then be determined for the next iteration of the autopilot. The result of all the above operations must be the set of actions to be applied to the actuators of the control surfaces.

1.2 System Architecture

To carry out the operations mentioned in the introduction, the system uses a series of modules that delimit the set of these operations. The system is therefore composed of the path manager, the path following and the autopilot blocks. As shown in Fig. 1, the path manager is responsible for defining the trajectories that the vehicle must follow, while the path following is in charge of determining the altitude, airspeed and heading. Finally, the autopilot calculates the required actions on the control surfaces.

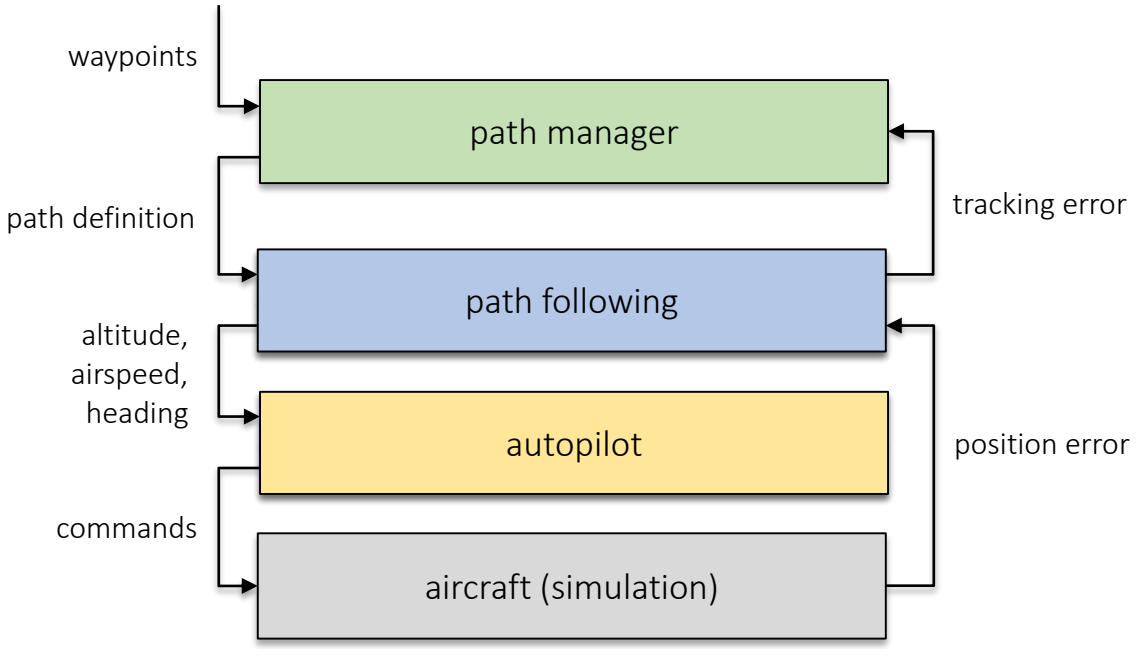


Fig. 1 System architecture and aircraft

Described below are the equations that govern the different blocks of the system. Later on, a common method is described to tune the parameters of the autopilot and the results obtained from isolated tests with it are provided.

1.3 Path Manager

The aim of this section is to describe a very simple strategy that combines straight-line and orbital paths to make transitions between consecutive waypoints. The result of the algorithm shown in this section can be described as the definition of the path the aircraft must follow from takeoff to landing through all waypoints.

1.3.1 Transitions between waypoints

Straight-line and orbit guidance strategies can be used to follow a series of waypoints. Define a waypoint path as an ordered sequence of waypoints

$$\mathcal{W} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_N\},$$

where $\boldsymbol{w}_i = (w_{n,i}, w_{e,i}, w_{d,i})^T \in \mathbb{R}^3$.

Consider the following scenario:

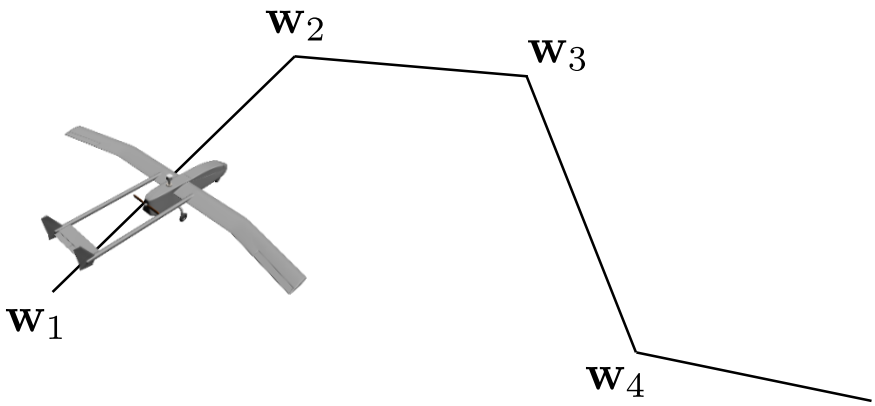


Fig. 2 Sequence of waypoints

When the vehicle reaches $\boldsymbol{w}_i$ it is desired to switch the guidance algorithm so that it will track the straight line segment $\overline{\boldsymbol{w}_i \boldsymbol{w}_{i+1}}$. Naturally, there are many ways in wich this behaviour can be accomplished. The first that come to our minds would probably be to design the algorithm so that it switches when the vehicle enters a ball around $\boldsymbol{w}_i$. The problem with small aerial vehicles is that wind can easily push the vehicle off course, thus forcing the vehicle to try and turn repeatedly until it reaches the ball.

A suitable alternative, one that is not sensitive to tracking error, is to use a half-plane switching criteria. Given a point $\boldsymbol{r} \in \mathbb{R}^3$ and a normal vector, define the half plane $\mathcal{H}(\boldsymbol{r}, \boldsymbol{n}) \triangleq \{\boldsymbol{p} \in \mathbb{R}^3 : (\boldsymbol{p} - \boldsymbol{r})^T \boldsymbol{n} \geq 0\}$.

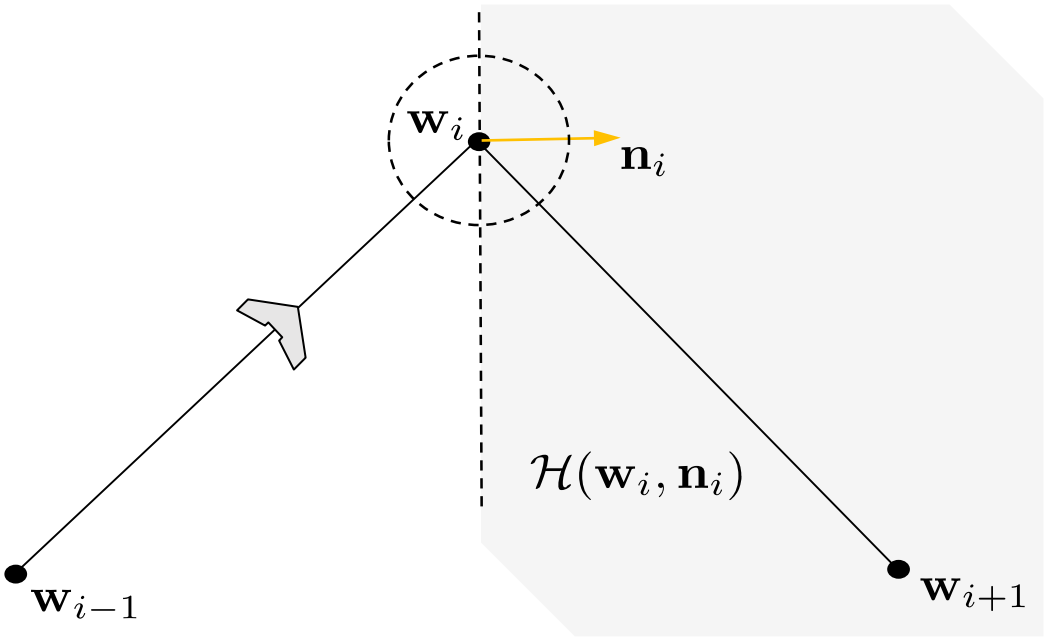


Fig. 3 Half-plane switching criteria

The aircraft tracks the straight-line path from $\boldsymbol{w_{i-1}}$ to $\boldsymbol{w_i}$ until it enters $\mathcal{H}(\boldsymbol{w_i}, \boldsymbol{n_i})$, at which point it will track the straight-line path from $\boldsymbol{w_i}$ to $\boldsymbol{w_{i+1}}$. The paths, however, provide neither a smooth nor balanced transition between the straight-line segments. An alternative is to smoothly transition between waypoints by inserting a fillet.

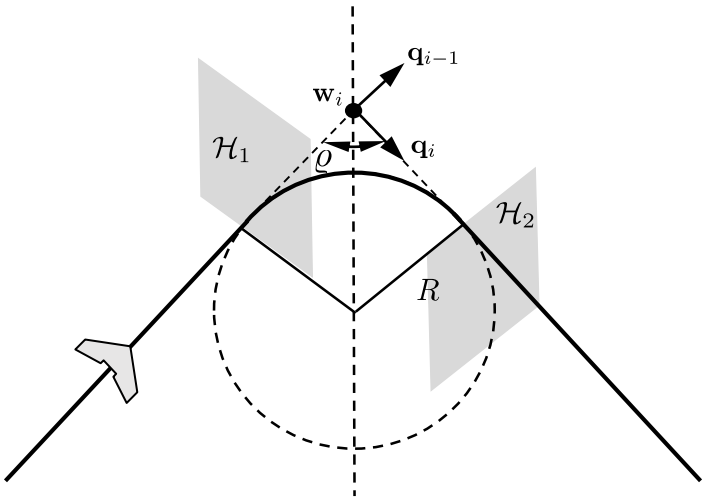


Fig. 4 Fillet smoothed transition

To put into practice the fillet maneuver, we will follow the straight-line segment $\overline{\boldsymbol{w_{i-1}w_i}}$ until entering the half plane $\mathcal{H}_1$. The right-handed orbit of radius R is then followed until entering the half plane $\mathcal{H}_2$, at which point the stright-line segment $\overline{\boldsymbol{w_i w_{i+1}}}$ is followed.

The angle between waypoints $\overline{\boldsymbol{w}_{i-1}\boldsymbol{w}_i}$ and $\overline{\boldsymbol{w}_i\boldsymbol{w}_{i+1}}$ is given by

$$\varrho = \cos^{-1}\left(-\boldsymbol{q}_{i-1}^T\boldsymbol{q}_i\right)$$

whereas the center of the fillet is

$$\boldsymbol{c} = \boldsymbol{w}_i - \frac{R}{\sin\frac{\varrho}{2}}\frac{\boldsymbol{q}_{i-1} - \boldsymbol{q}_i}{\|\boldsymbol{q}_{i-1} - \boldsymbol{q}_i\|}$$

The half plane $\mathcal{H}_1$ is defined by the location

$$\boldsymbol{r}_1 = \boldsymbol{w}_i - \frac{R}{\tan\frac{\varrho}{2}}\boldsymbol{q}_{i-1}$$

and the normal vector

$$\boldsymbol{q}_{i-1} = \frac{\boldsymbol{w}_i - \boldsymbol{w}_{i-1}}{\|\boldsymbol{w}_i - \boldsymbol{w}_{i-1}\|}$$

Similarly, he half plane $\mathcal{H}_2$ is defined by

$$\boldsymbol{r}_2 = \boldsymbol{w}_i + \frac{R}{\tan\frac{\varrho}{2}}\boldsymbol{q}_i$$

$$\boldsymbol{q}_i = \frac{\boldsymbol{w}_{i+1} - \boldsymbol{w}_i}{\|\boldsymbol{w}_{i+1} - \boldsymbol{w}_i\|}$$

1.4 Path Following

The aim of this section is to develop guidance laws for tracking straight-line segments and constant-altitude circular orbits. Therefore, algorithms for the path following block are described below.

The first approach to solve the following problem that comes to our minds could be to track a moving point. But this can result in significant problems for small aircrafts if disturbances, such as those due to wind, are not properly accounted for (wich is an unpractical task). If the vehicle is flying into a strong wind, the progression of the trajectory point must be slowed accordingly. Similarly, if it is flying downwind, the speed of the tracking point must be increased to keep it from overrunning the desired position.

Rather than using a trajectory tracking approach, the objective here is to be on the path rather than at a certain point at a particular time. With path following, the time dependence of the problem is removed.

The method described below is very simple and therefore has some limitations in terms of manoeuvres such as ascent or descent. As the method describes a straight line between two points to generate the trajectory, the attitude of the aircraft, as well as the speed on orbital trajectories, is very limited. The changes that have been added to this core algorithm are found in the corresponding part of the annex.

1.4.1 Straight-line Path Following

A straight-line path is described by two vectors in $\mathbb{R}^3$, namely

$$P_{line}(\boldsymbol{r}, \boldsymbol{q}) = \{\boldsymbol{x} \in \mathbb{R}^3 : \boldsymbol{x} = \boldsymbol{r} + \lambda\boldsymbol{q}, \lambda \in \mathbb{R}\},$$

where $\mathbb{R}^3$ is the origin of the path, and $q \in \mathbb{R}^3$ is a unit vector whose direction indicates the desired direction of travel. The course angle of $P_{line}(\boldsymbol{r}, \boldsymbol{q})$, as measured from north is given by

$$X_q \triangleq \tan^{-1}\frac{q_e}{q_n}$$

where $\boldsymbol{q} = (q_n\ q_e\ q_d)^T$ expresses the north, east, and down components of the unit direction vector. The path-following problem is most easily solved in a frame relative to the straight-line path. Selecting **r** as the center of the path frame, with the x-axis aligned with the projection of **q** onto the local north-east plane, the z-axis aligned with the inertial z-axis, and the y-axis selected to create a right-handed coordinate system, then

$$R_i^P \triangleq \begin{pmatrix} \cos X_q & \sin X_q & 0 \\ -\sin X_q & \cos X_q & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

is the transformation from the inertial frame to the path frame, and

$$\boldsymbol{e}_p = \begin{pmatrix} e_{p_x} \\ e_{p_y} \\ e_{p_z} \end{pmatrix} \triangleq R_i^P \left(\boldsymbol{p}^i - \boldsymbol{r}^i \right)$$

is the relative path error expressed in the path frame. The lateral straight-line path following problem is to select X^c so that $e_{p_y} \to 0$ asymptotically when X_q is known.

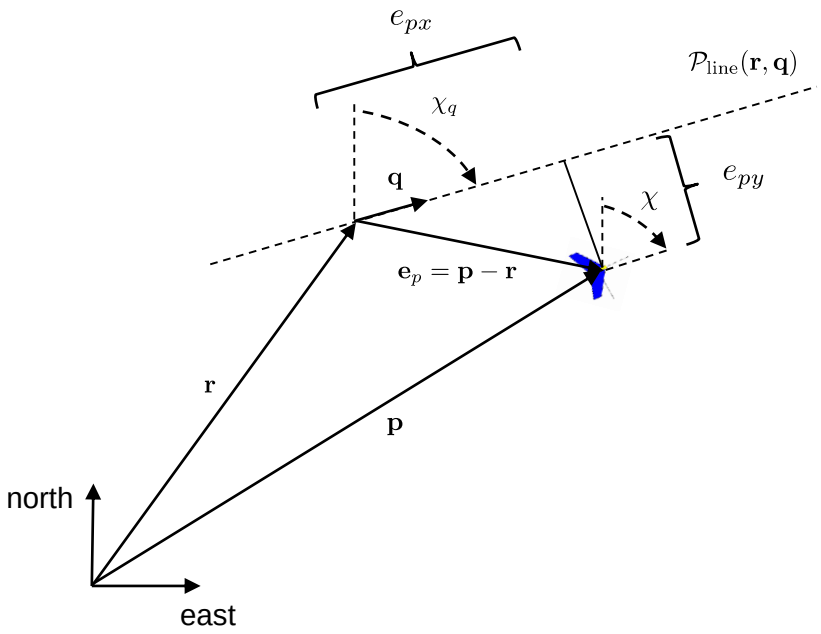


Fig. 5 North-east straigh-line path problem projection

The strategy will be to construct a desired course angle vector field that results in the vehicle moving toward the path so when e_{p_y} is large, the aircraft is directed to approach the path with course angle $X^\infty \in (0, \frac{\pi}{2}]$, and so that as e_{p_y} approaches zero, the desired course also approaches zero. The command for lateral path following is given by

$$X^c(t) = X_q - X^\infty \frac{2}{\pi} \tan^{-1}\left(k_{path} e_{p_y}(t) \right)$$

where k_{path} is a positive constant that influences the rate of the transition from X^∞ to zero. Large values of k_{path} yield abrupt transitions from X^∞ to zero, while small ones give smooth transitions.

To calculate the desired altitude, it is necessary to project the relative path error vector onto the vertical plane containing the path direction vector **q**. The projection **s** of the relative error vector is defined as

$$\boldsymbol{s}^i = \begin{pmatrix} s_n \\ s_e \\ s_d \end{pmatrix} = \boldsymbol{e}_p^i - \left(\boldsymbol{e}_p^i \cdot \boldsymbol{n} \right) \boldsymbol{n}$$

where

$$\boldsymbol{e}_p^i = \begin{pmatrix} e_{p_n} \\ e_{p_e} \\ e_{p_d} \end{pmatrix} \triangleq \boldsymbol{p}^i - \boldsymbol{r}^i = \begin{pmatrix} p_n - r_n \\ p_e - r_e \\ p_d - r_d \end{pmatrix}$$

and the unit vector normal to the $\boldsymbol{q} - \boldsymbol{k}^i$ plane is calculated as

$$\boldsymbol{n} = \frac{\boldsymbol{q} \times \boldsymbol{k}^i}{\|\boldsymbol{q} \times \boldsymbol{k}^i\|}$$

The desired altitude for an aircraft at **p** following the straight-line path is given by

$$h_d(\boldsymbol{r}, \boldsymbol{p}, \boldsymbol{q}) = -r_d + \sqrt{s_n^2 + s_e^2}\left(\frac{q_d}{\sqrt{q_n^2 + q_e^2}}\right)$$

The longitudinal straight-line path following problem is to select h^c so that $h \to h_d(\boldsymbol{r}, \boldsymbol{p}, \boldsymbol{q})$, which results in zero steady-state error in altitude for straight-line paths.

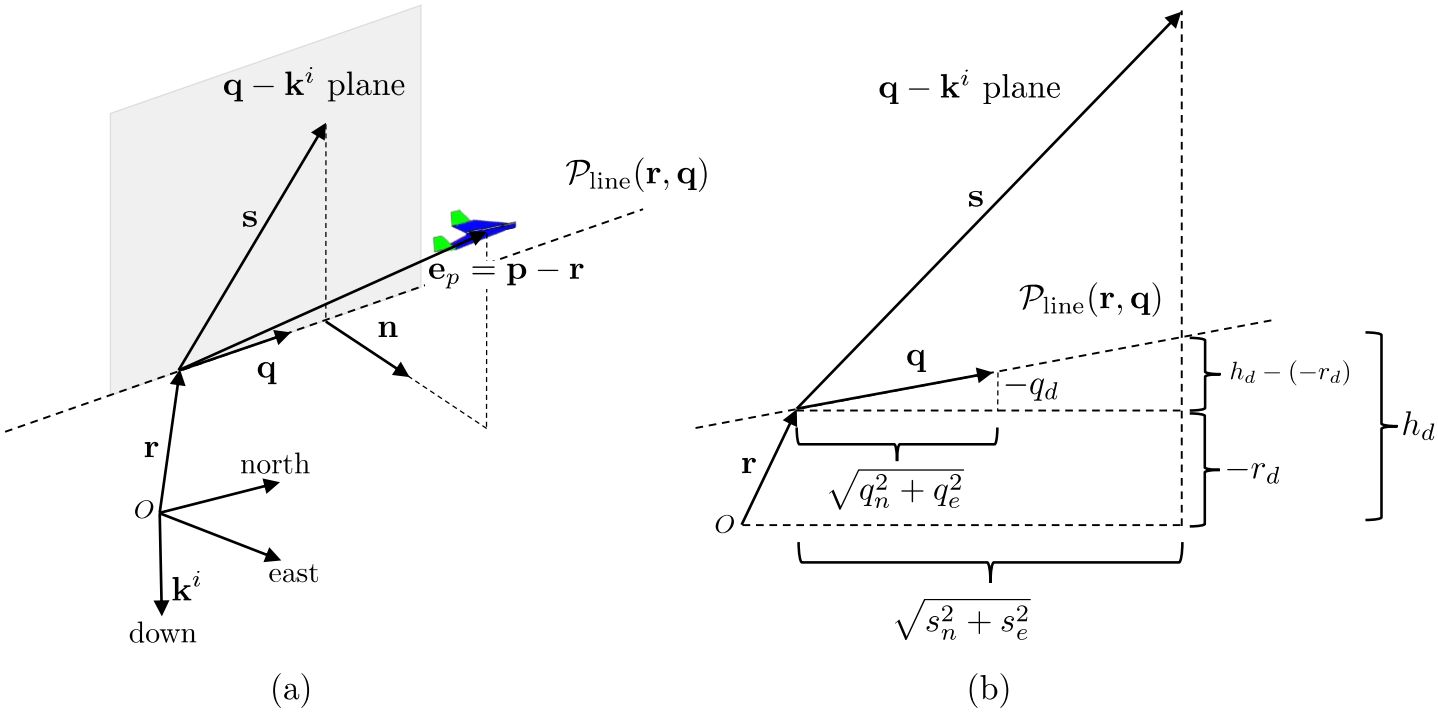


Fig. 6 q-k^i plane path problem projection

1.4.2 Orbit Following

An orbit path is described by a center $\boldsymbol{c} \in \mathbb{R}^3$, a radius $\rho \in \mathbb{R}$, and a direction $\lambda \in \{-1, 1\}$, as

$$P_{orbit}(\boldsymbol{c}, \rho, \lambda) = \{\boldsymbol{r} \in \mathbb{R}^3 : \boldsymbol{r} = \boldsymbol{c} + \lambda\rho(\cos\varphi, \sin\varphi, 0)^T, \varphi \in [0, 2\pi)\}$$

where $\lambda = 1$ signifies a clockwise orbit and $\lambda = -1$ a counterclockwise one. The center of the orbit is expressed in inertial coordinates so that $\boldsymbol{c} = (c_n, c_e, c_d)^T$, where $-c_d$ represents the desired altitude of the orbit and to maintain altitude we let $h^c = -c_d$.

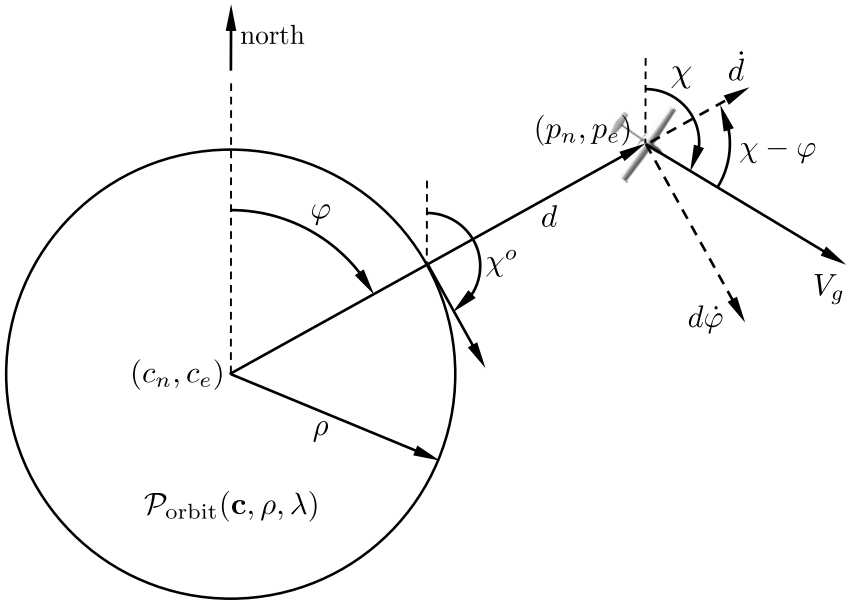


Fig. 7 Top-down view of the orbital path problem

Similarly to the straight-line path following problem, the strategy is to construct a desired course field that moves the aircraft onto the orbit $P_{orbit}(\boldsymbol{c}, \rho, \lambda)$. When the distance between the vehicle and the center of the orbit is large, it is desirable for the vehicle to fly toward the orbit center.

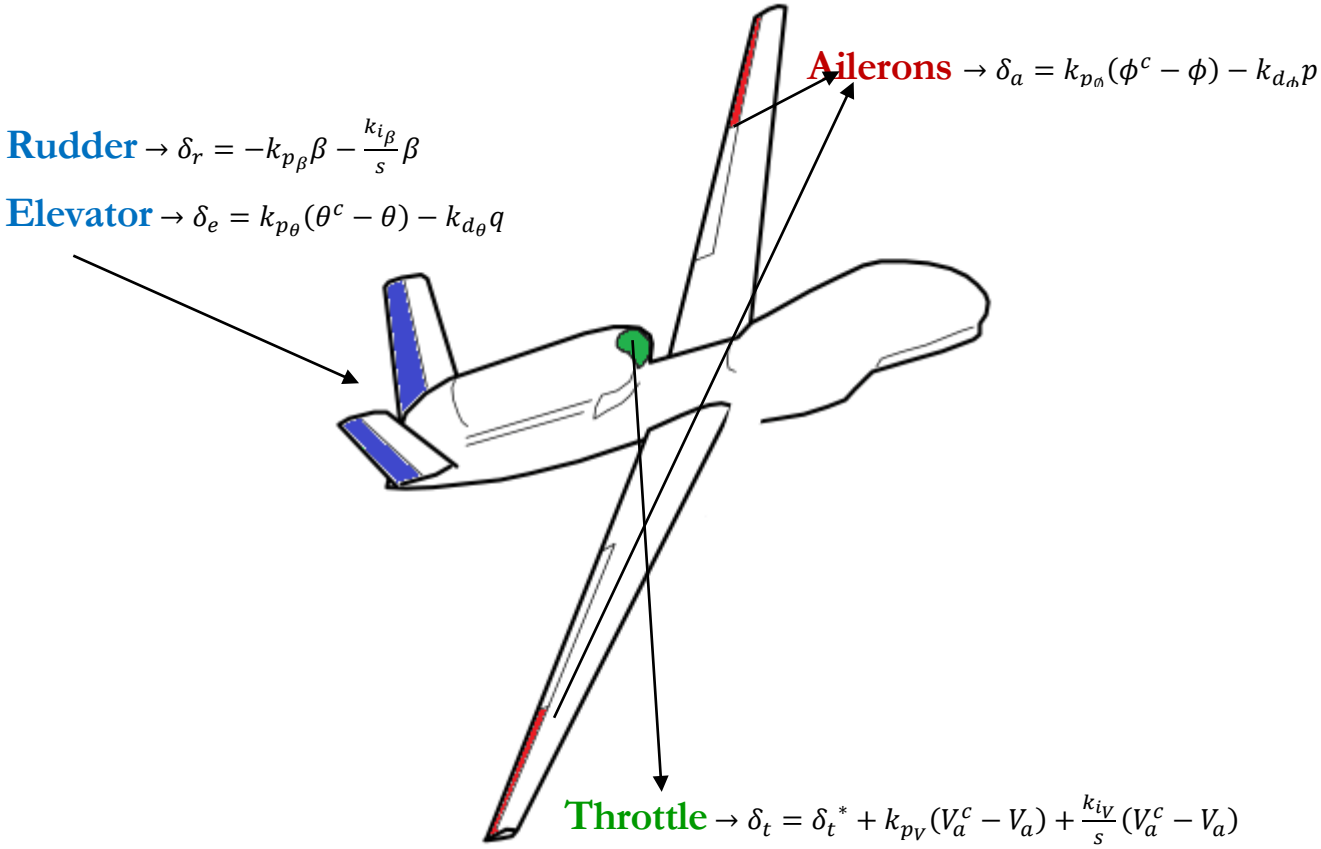
The course command for orbit following is given by

$$X^c(t) = \varphi + \lambda\left[\frac{\pi}{2} + \tan^{-1}\left(k_{orbit}\left(\frac{d-\rho}{\rho}\right)\right)\right]$$

where d is the radial distance from the desired center of the orbit to the vehicle, $\varphi = \tan^{-1}(p_e - c_e, p_n - c_n)$ is the phase angle of the relative position as shown in Fig. 7 and $k_{orbit} > 0$ is a constant that specifies the rate of transition from $\frac{\lambda\pi}{2}$ to zero.

1.5 Autopilot

An autopilot is a system used to guide an aircraft without the assistance of a pilot. For UAVs, the autopilot is in complete control of the aircraft during all phases of flight and, while some control functions may reside in the ground control station, the autopilot portion of the UAV control system resides on board the vehicle.



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

Pitch $(altitude\ hold)$ → $\theta^c = k_{p_h}(h^c - h) + \frac{k_{i_h}}{s}(h^c - h)$

Pitch $(airspeed\ hold)$ → $\theta^c = k_{p_{V2}}(V_a^c - V_a) + \frac{k_{i_{V2}}}{s}(V_a^c - V_a)$

Fig. 8 Autopilot PID equations for the control surfaces

1.6 Autopilot Design Using Succesive Loop Closure

The primary goal in autopilot design is to control the inertial position (pn, pe, h) and attitude (∅, θ, ψ) of the aircraft. It is assumed that the longitudinal dynamics are decoupled from the lateral dynamics, wich for most flight maneuvers of interest yields good performance. This simplifies the development of the autopilot significantly and allows to utilize a technique commonly used for autopilot design called successive loop closure.

The basic idea behind successive loop closure is to close several simple feedback loops in succession rather than designing a single (presumably more complicated) control system. A necessary condition in the design process is that the inner loop has the highest bandwidth (bandwidth been the band of frequencies to which the system responds satisfactorily) –without violating the saturation constraints-, so if the frequency of the reference signal is within bandwidth, then the ouput will be able to track it; else, tracking will be very slow.

The key assumption made is that for frequencies well below the bandwidth of the inner loop, the closed loop transfer function can be modeled as a gain of 1. With the inner-loop transfer function modeled as a gain of 1, design of the second loop is simplified because it includes only the plant transfer function and the compensator. The critical step in closing the loops successively is to design the bandwidth of the next loop so that it is a factor of 5 to 10 times smaller in frequency, thus ensuring that the unity gain assumption on the inner loop is not violated over the range of frequencies that the middle loop operates.

Since the longitudinal and lateral dynamics are decoupled, the autopilot design will be divided into lateral autopilot and longitudinal autopilot.

1.6.1 Lateral Autopilot

The lateral autopilot includes roll-attitude hold as an inner loop and course-angle hold as an outer loop.

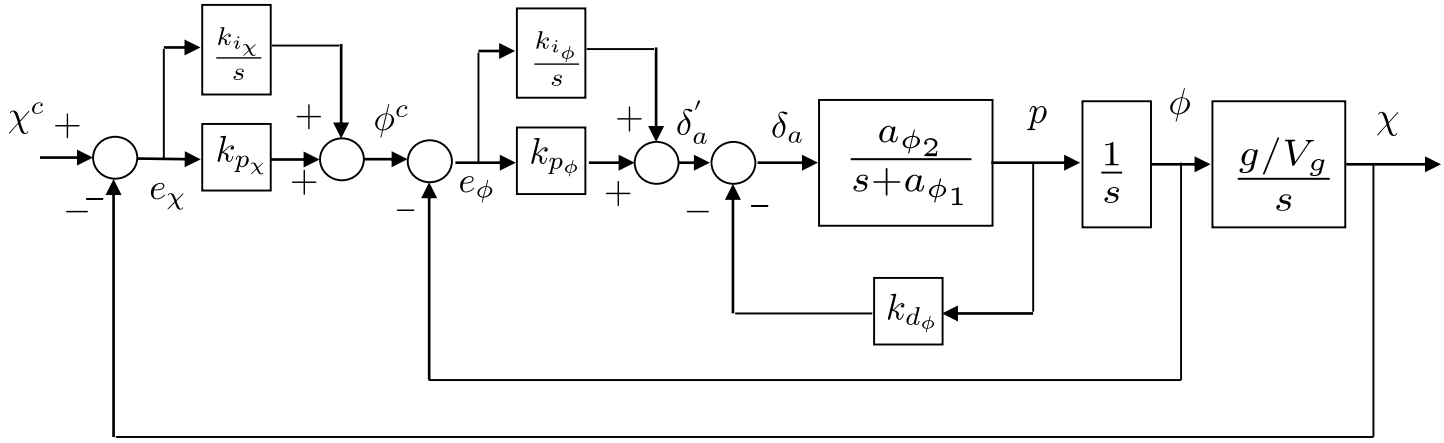


Fig. 9 Autopilot for lateral control using successive loop closure

1.6.1.1 Roll Loop Equations

The inner loop of the lateral autopilot is used to control roll angle and roll rate. Here is a systematic method for selecting the control gains based on the desired response of closed-loop dynamics:

Given the transfer function from ϕ^c to ϕ

$$H_{\phi/\phi^c} = \frac{a_{\phi 2} k_{p_\phi}}{s^2 + \left(a_{\phi 1} + a_{\phi 2} k_{d_\phi}\right) s + a_{\phi 2} k_{p_\phi}}$$

and since the DC gain of a system is the gain at the steady state (with t tending to infinity or s tending to zero), it can be noted that the DC gain is equal to 1.

If the desired response is given by a canonical second-order transfer function and the proportional gain is selected so that the ailerons saturate when the roll error is e_ϕ^{max} (being a design parameter), we get

$$k_{p_\phi} = \frac{\delta_a^{max}}{e_\phi^{max}} sign(a_{\phi 2})$$

The natural frequency of the roll loop is therefore given by

$$\omega_{n_\phi} = \sqrt{|a_{\phi 2}| \frac{\delta_a^{max}}{e_\phi^{max}}}$$

and finally

$$k_{d_\phi} = \frac{2\zeta_\phi \omega_{n_\phi} - a_{\phi 1}}{a_{\phi 2}}$$

where the damping ratio $\zeta_\emptyset$ is a design parameter.

It must be noted that the open-loop transfer function in the figure below, is a type one system which implies that zero steady-state tracking error in roll should be achievable without an integer.

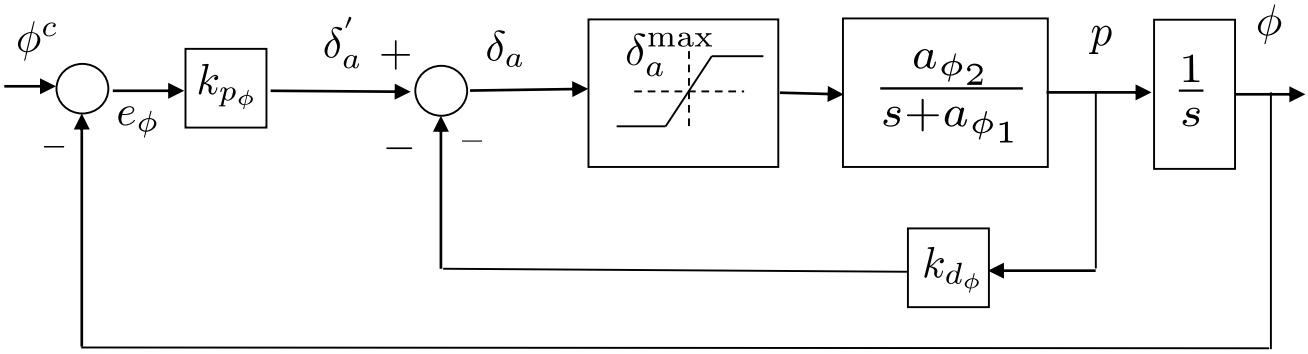


Fig. 10 Roll attitude hold control loops

Although in the process of creating the linear, reduced-order model of the roll dynamics, some terms were neglected –being seen now as disturbances-, I am not using any until the course loop and only to correct for steady state values. Integrators are known to add delays and instability and thus, are not very convenient on inner loops.

The output of the roll attitude hold loop is

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

1.6.1.2 Course Hold Loop Equations

If the the roll loop has been adequately tuned, then the DC gain is 1 over the range of frequencies from 0 to ω_{n_ϕ}. Under this condition, the resulting block diagram for the course hold can be simplified to the block below for the purposes of designing the outer loop.

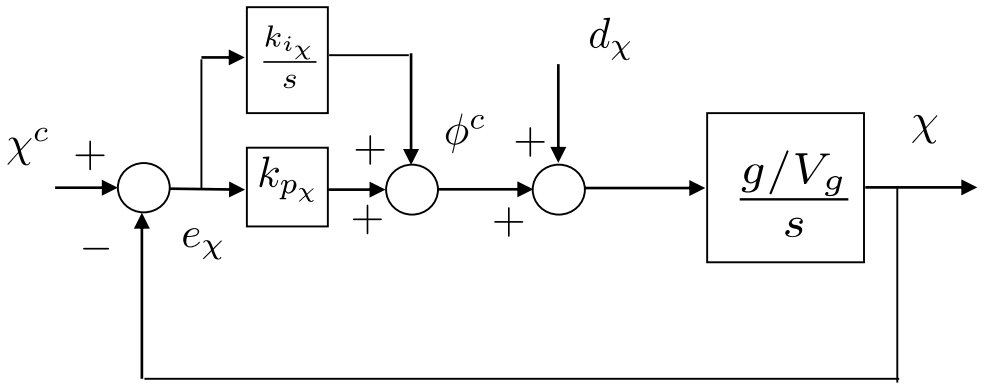


Fig. 11 Course hold outer feedback loop

The objective of the course hold design is to select k_{p_χ} and k_{i_χ} so that the course χ asymptotically tracks steps in the commanded course χ^c. From the simplified block diagram, the transfer functions from the inputs χ^c and d_χ to the output are given by

$$\chi = \left(\frac{g/V_g s}{s^2 + k_{p_\chi} g/V_g s + k_{i_\chi} g/V_g}\right) d_\chi + \left(\frac{k_{p_\chi} g/V_g s + k_{i_\chi} g/V_g}{s^2 + k_{p_\chi} g/V_g s + k_{i_\chi} g/V_g}\right) \chi^c$$

It must be noted that, if d_χ and χ^c are constants, then the final value theorem $\lim_{t\to\infty} x(t) = \lim_{s\to 0} sX(s)$ implies that $X \to X^c$. The transfer function from χ^c to χ has the form

$$H_{X/X^c} = \frac{2\zeta_x \omega_{n_x} s + \omega_{n_x}^2}{s^2 + 2\zeta_x \omega_{n_x} s + \omega_{n_x}^2}$$

As with the inner feedback loops, we can choose the natural frequency and damping of the outer loop and from those values calculate the feedback gains k_{p_x} and k_{i_x}. Note that because of the numerator zero, the standard intuition for the selection of ζ_x does not hold for this transfer function. Larger ζ_x results in larger bandwidth and smaller overshoot.

Comparing coefficients in the equations above, we find

$$k_{p_x} = \frac{2\zeta_x \omega_{n_x} V_g}{g}$$

$$k_{i_x} = \frac{\omega_{n_x}^2 V_g}{g}$$

To ensure proper function of this successive-loop-closure design, it is essential that there be sufficient bandwidth separation between the inner an outer feedback loops. Adequate separation can be achieved by letting

$$\omega_{n_x} = \frac{\omega_{n_\phi}}{W_x}$$

where W_x the separation is a design parameter, as said beforehand, usually chosen to be between 5 and 10. Generally, more bandwidth separation is better but requires either slower response in the χ loop (lower ω_{n_x}), or faster response in the ϕ loop (higher ω_{n_ϕ}). Faster response usually comes at the cost of requiring more actuator control authority, which may not be possible given the pysical constraints of the actuators.

The output of the course hold loop is

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

1.6.1.3 Sideslip Hold Loop Equations

If the aircraft is equipped with a rudder, the rudder can be used to maintain zero sideslip angle. The sideslip hold loop is shown in the figure below, and the transfer function β^c from to β is given by

$$H_{\beta/\beta^c} = \left(\frac{a_{\beta 2} k_{p_\beta} s + a_{\beta 2} k_{i_\beta}}{s^2 + \left(a_{\beta 1} + a_{\beta 2} k_{p_\beta}\right) s + a_{\beta 2} k_{i_\beta}}\right)$$

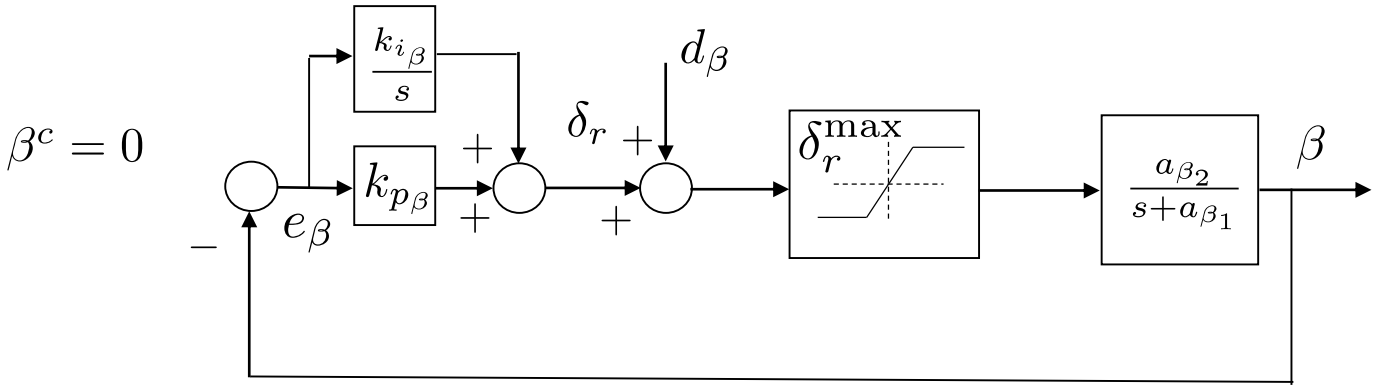


Fig. 12 Sideslip hold control loop

Supossing that the maximum error in sideslip is given by e_β^{max} and that the maximum allowable rudder deflection is given by δ_r^{max}, we obtain

$$k_{p_\beta} = \frac{\delta_r^{max}}{e_\beta^{max}} sign(a_{\beta 2})$$

And by choosing a value for ζ_β to give the desired damping, we get

$$k_{i_\beta} = \frac{1}{a_{\beta 2}} \left(\frac{a_{\beta 1} + a_{\beta 2} k_{p_\beta}}{2\zeta_\beta} \right)^2$$

The output of the sideslip hold loop is

$$\delta_r = -k_{p_\beta}\beta - \frac{k_{i_\beta}}{s}\beta$$

1.6.1.4 Control Gains Tuning using Matlab

For the design of the control loops it must be taken into account both the physical capabilities of the actuators and the inertial properties of the vehicle, including the structural integrity of the aircraft.

For the design of the experiments, it is also very important to take into account how the system will behave in the different situations to which we subject it. Therefore, we must take a pragmatical approach when deciding the inputs for the control loops, as well as when analyzing the responses of the magnitudes of which we have intuitive physical awareness, which I reckon can be very puzzling if we do not have an aeronautical based education but, nonetheless, can be overcomed with a reasonable research in the field.

I must say that analyzing results from existing experiments showing cinematic and dynamic behaviour on aircrafts provide with the sense of aeronautical intuition that is needed to be able to comprehend the physical capabilities of these stunning flying marvels.

Some magnitudes relative to the dynamic behaviour of most aircraft can be seen in the table below.

roll rate (p)	80 deg/s
yaw rate (r)	50 deg/s
heading	20 deg/s
aileron deflection (each)	±30 degrees
ruddervator deflection (as rudder)	±30 degrees

Table 1 Common dynamic behaviour of MAVS [2], [3] and [4]

There are seven gains associated with the lateral autopilot. The derivative gain k_{d_ϕ} provides roll rate damping for the innermost loop. The roll attitude is regulated with the proportional gain k_{p_ϕ}. The course angle is regulated with the proportional and integral gains k_{p_x} and k_{i_x}. The sideslip hold is governed with the proportional and integral gains k_{p_β} and k_{i_β}. The idea with successive loop closure is that the gains are successively chosen beginning with the inner loop and working outward. In particular, k_{d_ϕ} and k_{p_ϕ} are selected first, and finally k_{p_x} and k_{i_x}. The gains k_{p_β} and k_{i_β} are selected indepently.

To select k_{d_ϕ} and k_{p_ϕ} it has been taken into account the physical limitations of the ailerons and the cinematic and dynamic behaviour of the aircraft.

After a thorough study of the dynamics I've found out that the behaviour of some of the system loops differs greatly from the one outside the simulation if only the method of comparison with second-order systems is used for the design of the gains. Therefore, I've decided to select some of the gains by carefully analyzing the system with respect to the data available from existing aircrafts.

1.6.1.4.1 Roll Loop Design

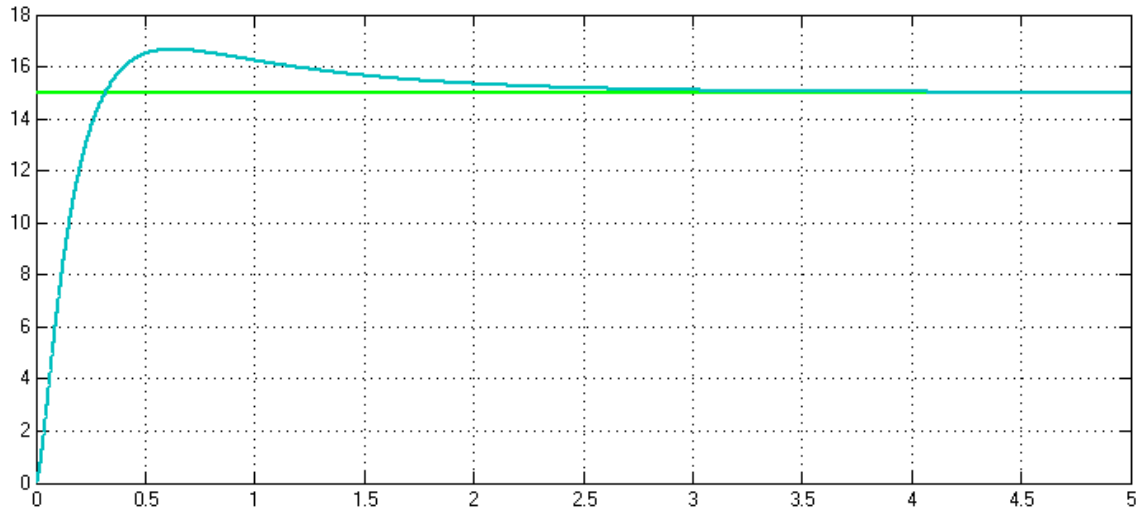


Fig. 13 Commanded roll angle and roll response [deg]

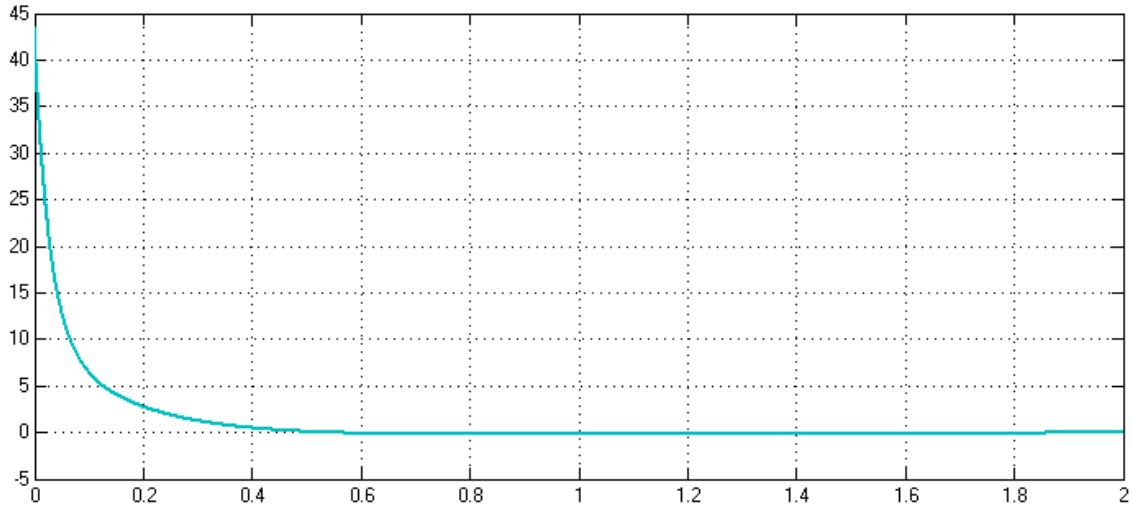


Fig. 15 Ailerons actuation [deg]

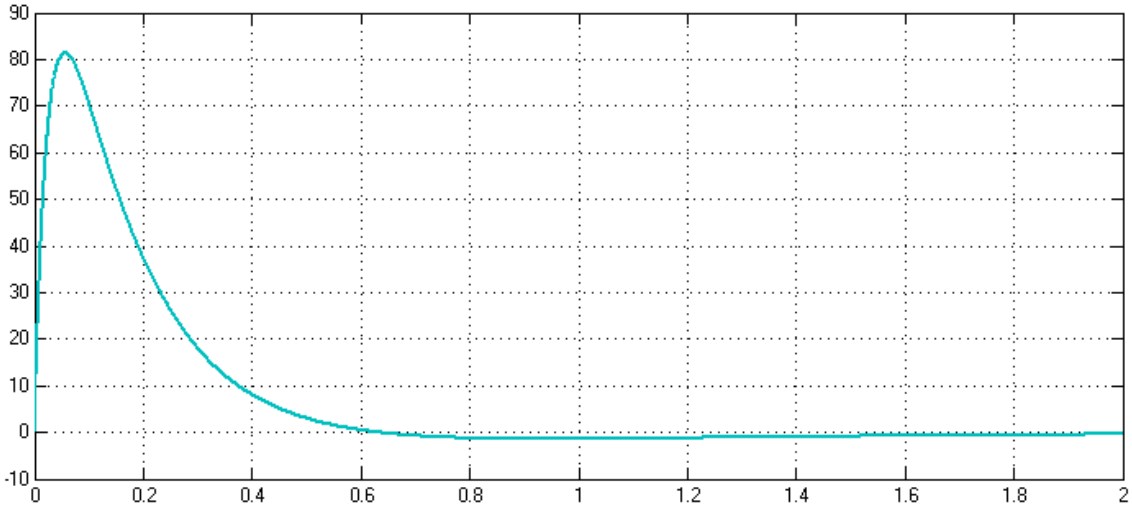


Fig. 15 Roll rate [deg/s]

The roll loop design has been conducted commanding a banking angle of 15º. It can be seen that the roll rate keeps between common values. Also, the ailerons command does not exceeds its physical limitations (60º). Since most of the aircraft's weight is located on the belly and the ailerons are so far off the wings, it is typical for the bank response to be the fastest actuation on the aircraft. Since the ailerons actuation refers to the angle difference between both the ailerons, the command for each of them results in a deflection half the value shown. For a faster behaviour, the roll action must be under-damped, which results in overshoot; one degree and a half is ridiculously small, though, so the behaviour is perfectly fine. As soon as the desired roll is achieved, the ailerons go back to zero deflection.

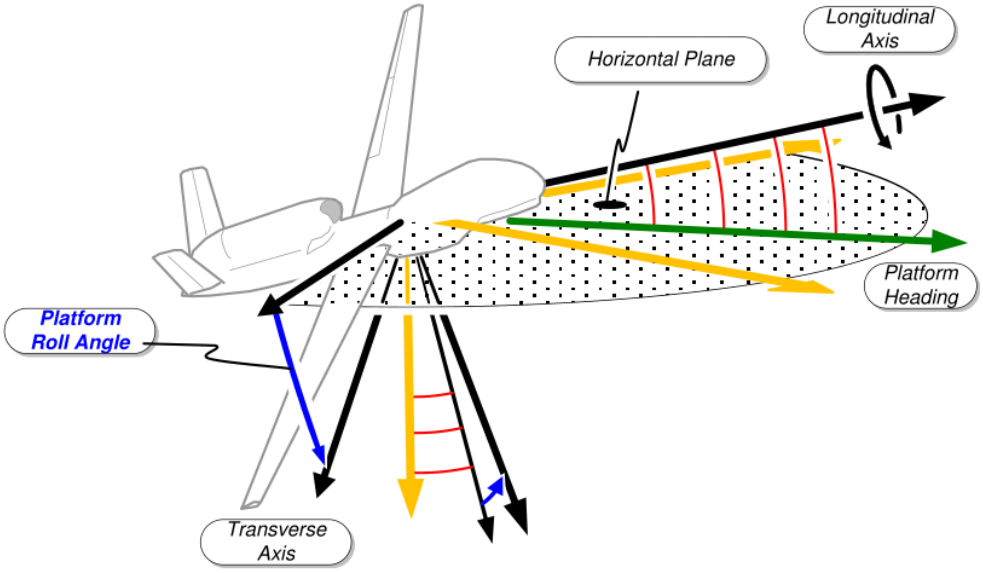


Fig. 16 Roll rotation example

1.6.1.4.2 Course Hold

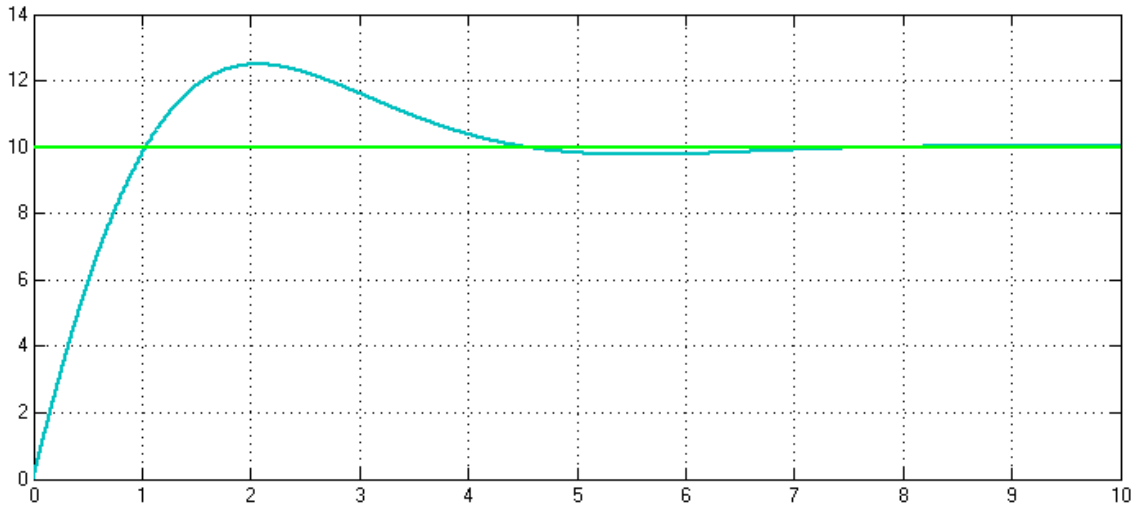


Fig. 17 Commanded course angle and course response [deg]

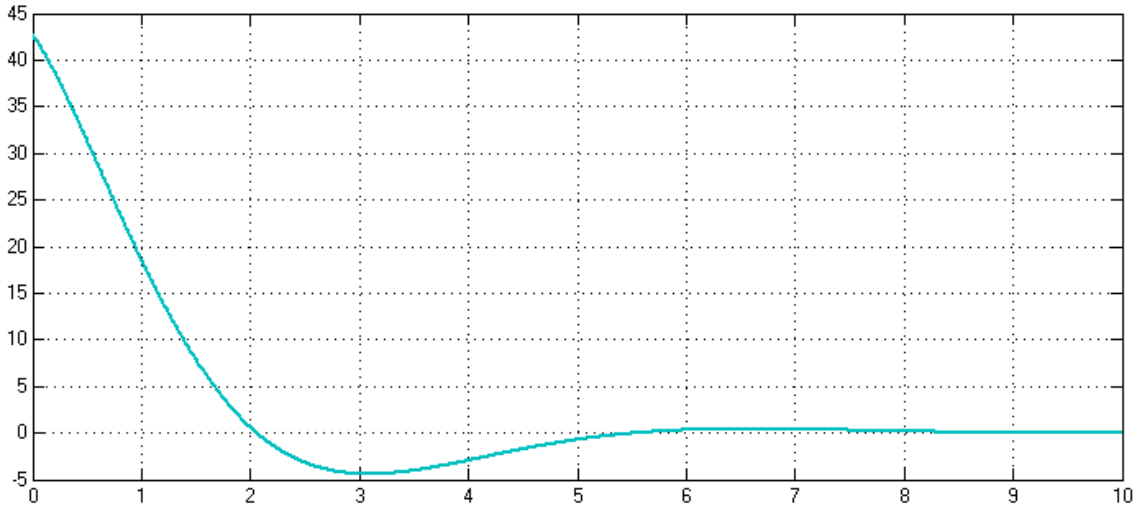


Fig. 18 Commanded roll [deg]

For the course hold design it has been comanded a 10 degrees clockwise turn. There's a big difference in speed with respect to the roll response because the course hold loop has to allow the inner one (the roll loop) to reach the desired command. Here, the rise time is also consistent with a typical heading rate. In the second figure it can be seen that the vehicle is commanded to place the belly so that the lateral component of the lift force pushes the aircraft to increase the heading angle and that, once the desired course is reached, the roll signal returns to 0.

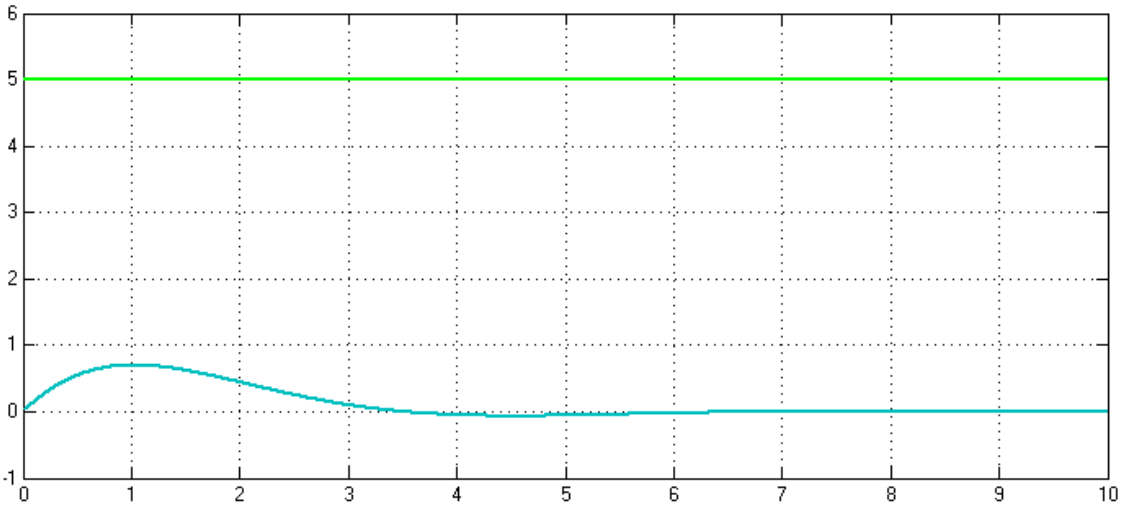


Fig. 19 Disturbance in course angle and course response [deg]

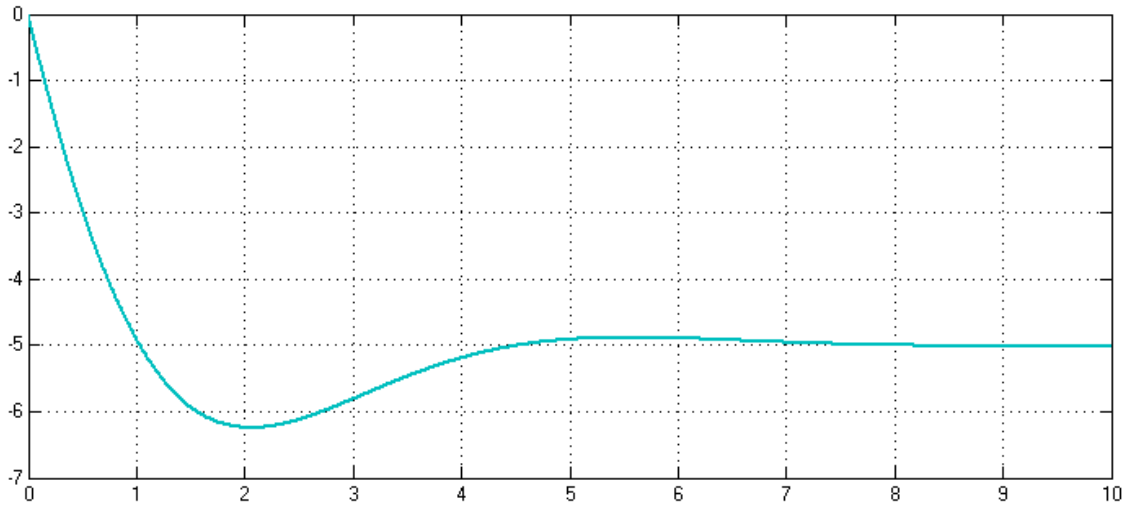


Fig. 20 Commanded roll [deg]

If a gust of wind pushes the aircraft in such a way so that it drifts from course, say 5º, it can be seen that it takes more or less 4 seconds to correct course. But this is good because we don't want the vehicle to sharp turn; as a consecuence, the aircraft will continue its path smoothly. From the seconde figure it is clear that the UAV shows its belly in the direction where the disturbance took place so that the lateral component of the lift force helps adjust the drifting. The aircraft will continue to counteract the disturbance by commanding a roll angle different than 0 until the disturbance dissappears.

1.6.1.4.3 Sideslip Hold

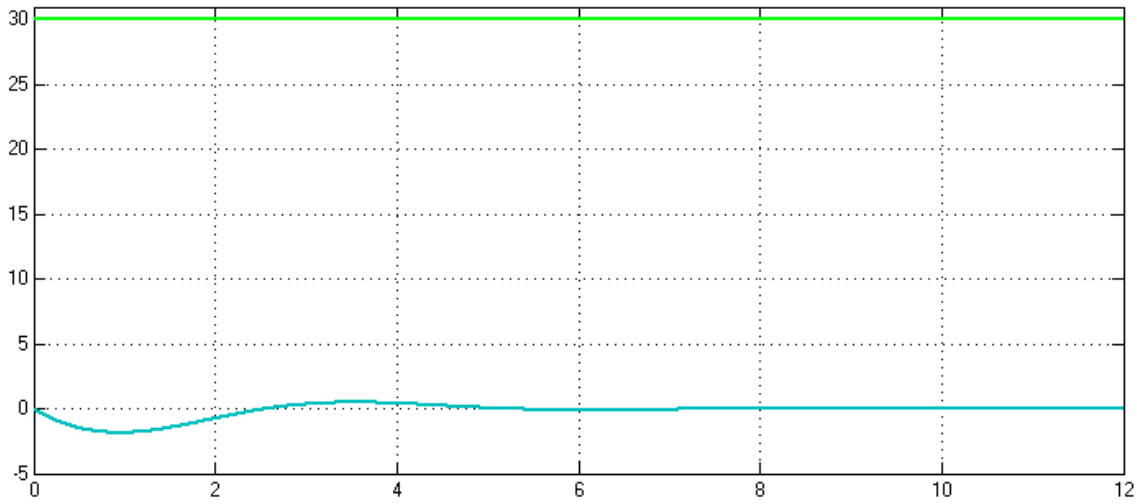


Fig. 21 Disturbance and yaw angle response [deg]

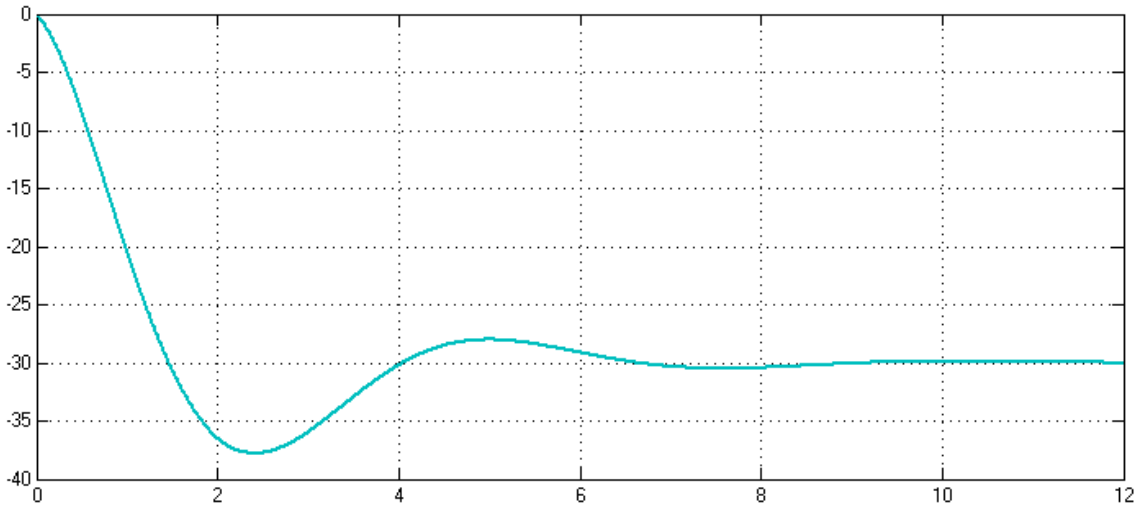


Fig. 22 Rudder actuation [deg]

One of the interesting advantages of the ruddervators is that the maximum total deflection exceeds the one of its t-tail counterpart which results in better maneuverability. If our vehicle ecounters a gust, it will likely change its attitude. If the yaw angle is affected 30º as a disturbance, the ruddervators will counteract the effect. Since the tail is located far off the center of gravity, its angular momentum will be quite strong and the aircraft will soon regain its attitude. Although it seems slow to react, it must be taken into account that the effect is very little, wich accounts for a faster damping of the disturbance. The rudder will continue acting upon the the effects of the disturbance until the last one disappears.

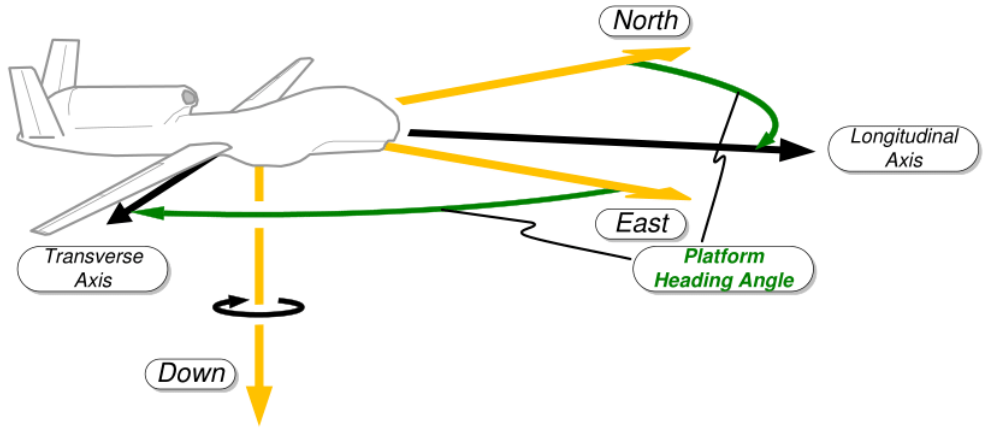


Fig. 23 Heading angle example

k_{d_ϕ}	0,3
k_{p_ϕ}	2,9
k_{i_ϕ}	3,1
k_{p_x}	4,29
k_{i_x}	4,005
k_{p_β}	-2
k_{i_β}	-12,88

Table 2 PID values obtained from the lateral controller design

1.6.2 Longitudinal Autopilot

The longitudinal autopilot is more complicated than his partner because airspeed plays a significant role in the longitudinal dynamics. The design objective will be to regulate airspeed and altitude using the throttle and the elevator as actuators. The method used depends on the altitude error and will be divided into the regimes shown below and implemented as a state machine.

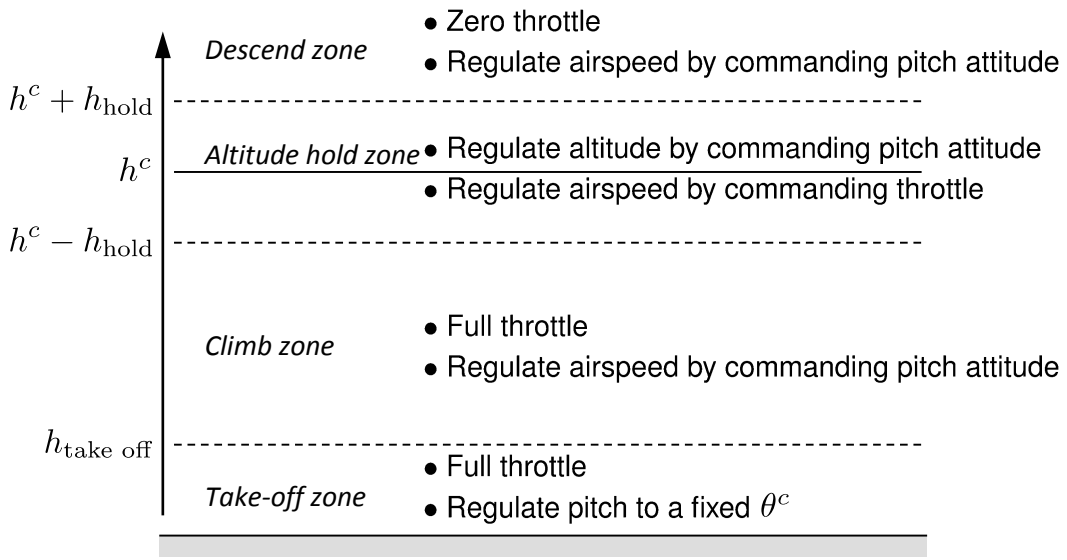


Fig. 24 Flight regimes for the longitudinal autopilot

In the *take-off zone*, full throttle is commanded and pitch is regulated to a fixed angle using the elevator. The objective in the *climb zone* is to maximize the climb rate, full throttle is commanded and the airspeed is regulated using the pitch angle. If the airspeed increases above its nominal value, then the aircraft is caused to pitch up, which results in an increase in climb rate and a decrease in airspeed. Similarly, if the airspeed drops below the nominal value, the aircraft is pitched down, thereby increasing the airspeed but also decreasing the climb rate. Regulating the airspeed using pitch attitude effectively avoids stall conditions. After take-off the

aircraft is attempting to increase its airspeend and doing so by pitching down would drive the aircraft into the ground, that's why we do not regulate airspeed with pitch attitude immediately after take-off.

In the *descend zone* the throttle is commanded to zero. Again, stall conditions are avoided by regulating airspeed using the pitch angle, thus maximizing the descent rate at a given airspeed. In the *altitude hold zone*, the airspeed is regulated by adjusting the throttle and the altitude is regulated by commanding pitch.

To implement the longitudinal autopilot, we need the following feedback loops:

1.6.2.1 Pitch Attitude Hold Loop Equations

The pitch attitude hold loop is similar to the roll attitude hold loop and a similar approach will be conducted.

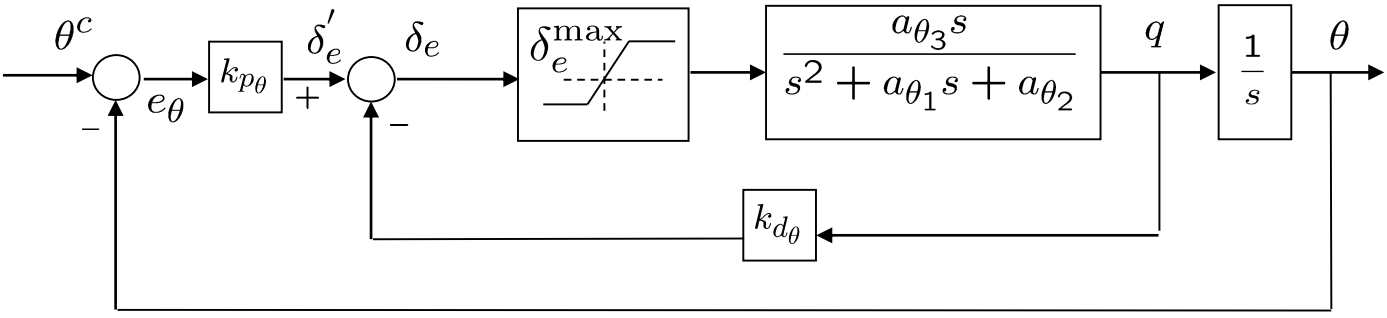


Fig. 25 Pitch attitude hold feedback loops

The transfer function from θ^c to θ is given by

$$H_{\theta/\theta^c} = \frac{a_{\theta 3}k_{p_\theta}}{s^2 + (a_{\theta 1} + a_{\theta 3}k_{d_\theta})s + (a_{\theta 2} + a_{\theta 3}k_{p_\theta})}$$

but contrary to the ones before, its DC gain is not equal to one.

If the desired response is given by the canonical second-order transfer function and the proportional gain is set so to avoid saturation when the maximum input error is experienced, we get

$$k_{p_\theta} = \frac{\delta_e^{max}}{e_\theta^{max}} sign(a_{\theta 3})$$

since k_{p_θ} and $a_{\theta 3}$ need to be of the same sign to ensure stability.

The bandwidth limit of the pitch loop can be calculated as

$$\omega_{n_\theta} = \sqrt{a_{\theta 2} + |a_{\theta 3}| \frac{\delta_e^{max}}{e_\theta^{max}}}$$

and finally

$$k_{d_\theta} = \frac{2\zeta_\theta \omega_{n_\theta} - a_{\theta 1}}{a_{\theta 3}}$$

where the damping ratio $\zeta_\emptyset$ is a design parameter.

The DC gain of this inner-loop transfer function is given by

$$K_{\theta_{DC}} = \frac{a_{\theta 3}k_{p_\theta}}{a_{\theta 2} + a_{\theta 3}k_{p_\theta}}$$

which for typical gain values is significantly less than one. The design of the outer loops will use this DC gain to represent the gain of the inner loop over itrs full badwidth. An integral feedback term could be employed to

ensure unity DC gain on the inner loop. The addition of an integral term, however, can severely limit the bandwidth of the inner loop. For this reason, I am not using the integral control on th epitch loop. The actual pitch, though, will not converge to the commanded pitch angle. This fact will be taken into account in the development of the outer loops.

The output of the pitch attitude-hold loop is

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

1.6.2.2 Altitude Hold Using Commanded Pitch Loop Equations

The altitude-hold autopilot utilizes a successive-loop-closure strategy with the pitch-attitude-hold autopilot as an inner loop, as shown in the figure below.

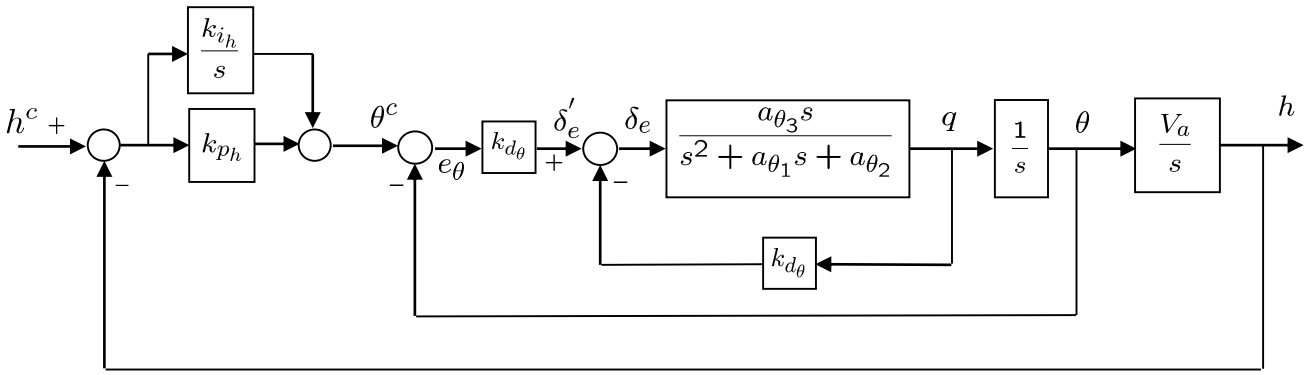


Fig. 26 Successive loop feedback structure for altitude-hold autopilot

Assuming that the pitch loop functions as designed and that $\theta \approx K_{\theta_{DC}} \theta^c$, the altitude-hold loop using the commanded pitch can be approximated by the block diagram below.

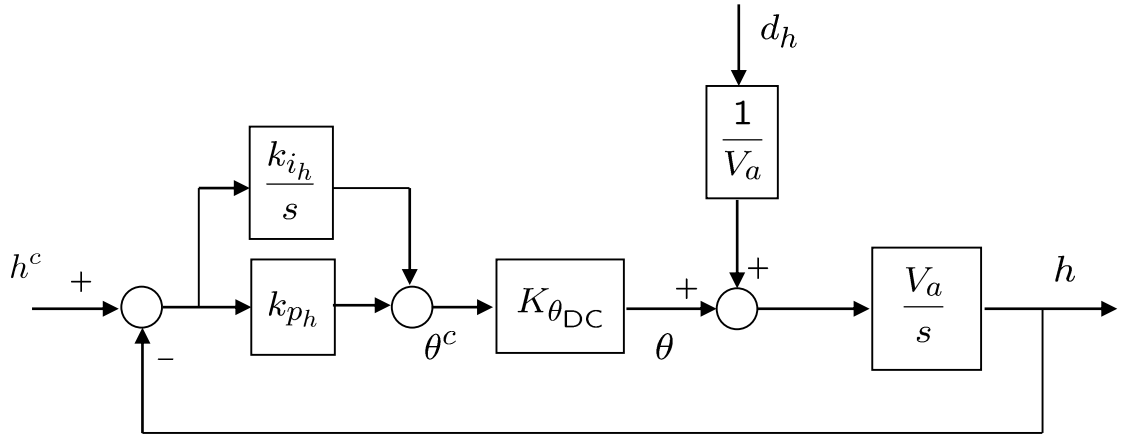


Fig. 27 Altitude-hold loop using the commanded pitch angle

In the Laplace domain:

$$h = \left(\frac{s}{s^2 + K_{\theta_{DC}} V_a k_{p_h} s + K_{\theta_{DC}} V_a k_{i_h}}\right) d_h + \left(\frac{K_{\theta_{DC}} V_a k_{p_h} \left(s + \frac{k_{i_h}}{k_{p_h}}\right)}{s^2 + K_{\theta_{DC}} V_a k_{p_h} s + K_{\theta_{DC}} V_a k_{i_h}}\right) h^c$$

where it can be seen that the DC gain is 1 and constant disturbances are rejected. The closed-loop transfer funtion is again independent of aircraft parameters and is dependent only on the known airspeed.

Similar to the course loop, let

$$\omega_{n_h} = \frac{1}{W_h} \omega_{n_\theta}$$

where the bandwidth separation W_h is a design parameter that is usually between 5 and 15. If the desired response of the altitude-hold loop is given by the canonical second-order transfer function and the gains k_{i_h} and k_{p_h} should be chosen such that the bandwidth of the altitude-from-pitch loop is less than the bandwidth of the pitch-attitude-hold loop, then

$$k_{i_h} = \frac{\omega_{n_h}^2}{K_{\theta_{DC}}}$$

$$k_{p_h} = \frac{2\zeta_h \omega_{n_h}}{K_{\theta_{DC}} V_a}$$

The output of the altitude-hold-with-pitch loop is

$$\theta^c = k_{p_h}(h^c - h) + \frac{k_{i_h}}{s}(h^c - h)$$

1.6.2.3 Airspeed Hold Using Commanded Pitch

The dynamic model for airspeed using pitch angle is the following:

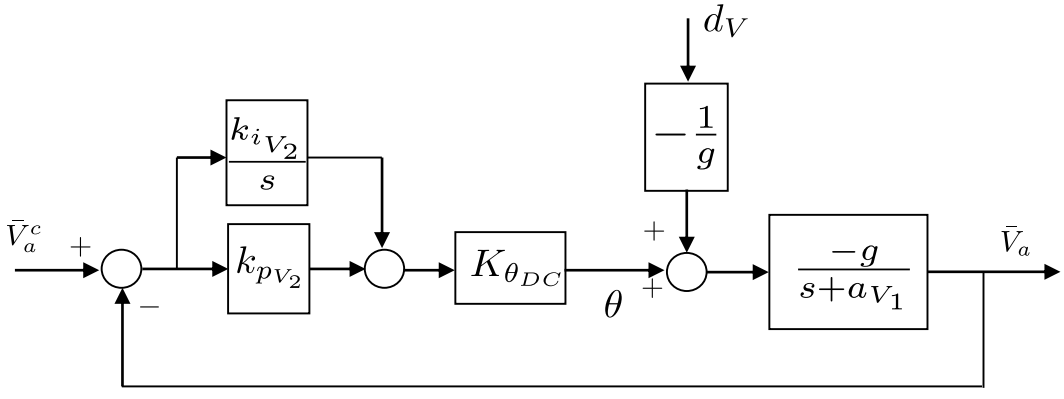


Fig. 28 Airspeed regulation using the pitch angle

Where it can be seen that disturbance rejection requires a PI controller. In the Laplace domain, we have

$$\bar{V}_a = \left(\frac{s}{s^2 + \left(a_{V1} - K_{\theta_{DC}} g k_{p_{V2}}\right)s - K_{\theta_{DC}} g k_{i_{V2}}}\right) d_V + \left(\frac{-K_{\theta_{DC}} g k_{p_{V2}} \left(s + \frac{k_{i_{V2}}}{k_{p_{V2}}}\right)}{s^2 + \left(a_{V1} - K_{\theta_{DC}} g k_{p_{V2}}\right)s - K_{\theta_{DC}} g k_{i_{V2}}}\right) \bar{V}_a{}^c$$

Note that the DC gain is 1 and the step disturbances are rejected.

To hold a constant airspeed, the pitch angle must approach a non-zero angle of attack. The integrator will wind up to command the appropriate angle of attack.

The gains $k_{p_{V2}}$ and $k_{i_{V2}}$ should be chosen so that the bandwidth of the airspeed-from-pitch loop is less than the bandwidth of the pitch-attitude-hold loop. Let

$$\omega_{n_{V2}} = \frac{1}{W_{V2}} \omega_{n_\theta}$$

where the bandwidth separation W_{V2} is a design parameter. If we match, as done before, the denominator coefficients in our Laplace domain equation with those of a canonical second-order transfer function, we get

$$k_{i_{V2}} = -\frac{\omega_{n_{V2}}^2}{K_{\theta_{DC}} g}$$

$$k_{p_{V2}} = \frac{a_{V1} - 2\zeta_{V2}\omega_{n_{V2}}}{K_{\theta_{DC}} g}$$

The output of the airspeed hold with pitch loop is

$$\theta^c = k_{p_{V2}}(V_a^c - V_a) + \frac{k_{i_{V2}}}{s}(V_a^c - V_a)$$

1.6.2.4 Airspeed Hold Using Throttle

The dynamic model for airspeed using the throttle as an input is

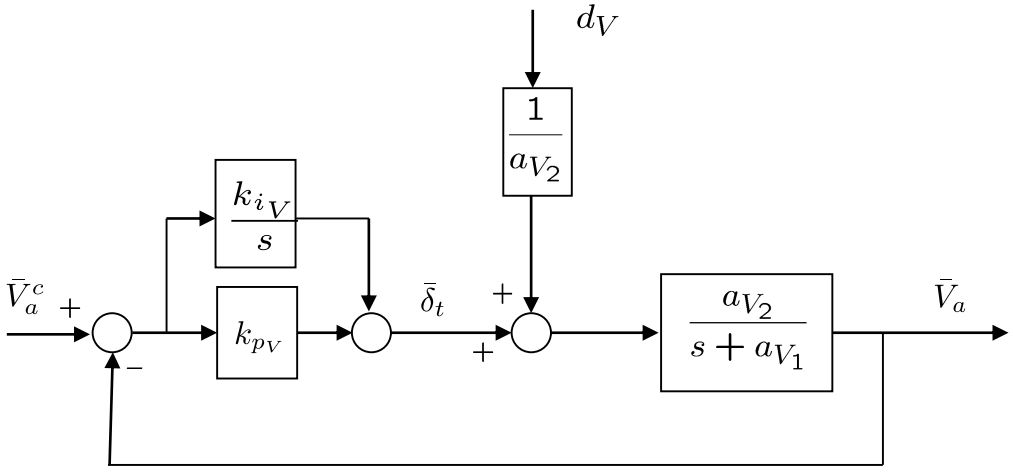


Fig. 29 Airspeed hold using throttle

and if proportional-integral control is used, then

$$\bar{V}_a = \left(\frac{1}{s^2 + (a_{V1} + a_{V2}k_{p_V})s + a_{V2}k_{i_V}}\right) d_V + \left(\frac{a_{V2}(k_{p_V}s + k_{i_V})}{s^2 + (a_{V1} + a_{V2}k_{p_V})s + a_{V2}k_{i_V}}\right) \bar{V}_a{}^c$$

which results in a DC gain of 1, with step disturbance rejection. k_{p_V} and k_{i_V} are calculated, as previously mentioned, as

$$k_{i_V} = \frac{\omega_{n_V}^2}{a_{V2}}$$

$$k_{p_V} = \frac{2\zeta_V \omega_{n_V} - a_{V1}}{a_{V2}}$$

The output of the airspeed hold with throttle loop is

$$\delta_t = \delta_t{}^* + k_{p_V}(V_a^c - V_a) + \frac{k_{i_V}}{s}(V_a^c - V_a)$$

where, if $\delta_t{}^*$ is not known, it can simply be thought of as a step disturbance, and the integrator will wind up to reject it.

1.6.2.5 Control Gains Tuning using Matlab

For the design of the longitudinal autopilot, we have to pay special attention to the saturation limits of the control surfaces and the lift force limitations with respect to the angle of attack. As the simulink model for the design of the control loops does not take into account all the ins and outs of the aerodynamic equations, I have incorporated saturations from ±60° pitch (which is very close to angle of attack in the absence of wind), which depicts the moment where the lift force falls. Once the pitch attitude hold loop is designed, the last three feedback loops can be designed somewhat independently.

pitch rate (q)	65 deg/s
elevator deflection	±30 degrees

Table 3 Common dynamic behaviour of small unmanned aerial vehicles [2] and [4]

There are eight gains associated with the lateral autopilot. The derivative gain k_{d_θ} provides pitch rate damping for the innermost loop. The pitch attitude is regulated with the proportional gain k_{p_θ}. The altitude is regulated with the proportional and integral gains k_{p_h} and k_{i_h}. The airspeed hold using pitch is governed with the proportional and integral gains $k_{p_{V_2}}$ and $k_{i_{V_2}}$. And finally, the airspeed hold using throttle is governed with the proportional and integral gains k_{p_V} and k_{i_V} The idea with successive loop closure is that the gains are successively chosen beginning with the inner loop and working outward. In particular, k_{d_θ} and k_{p_θ} are selected first, and the rest of the gains are selected taking into account the behaviour of the inner loop.

1.6.2.5.1 Pich Hold

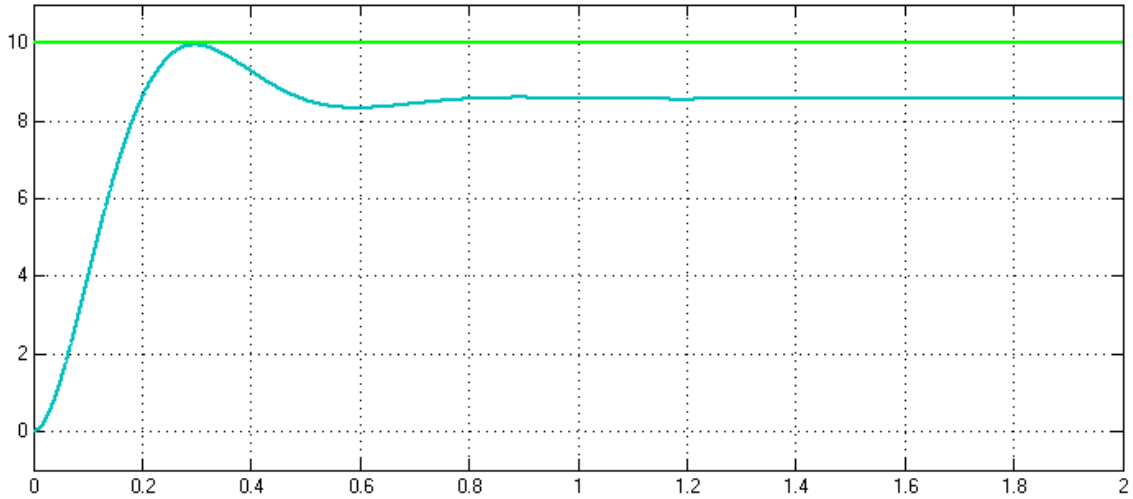


Fig. 31 Commanded pitch angle and pitch response [deg]

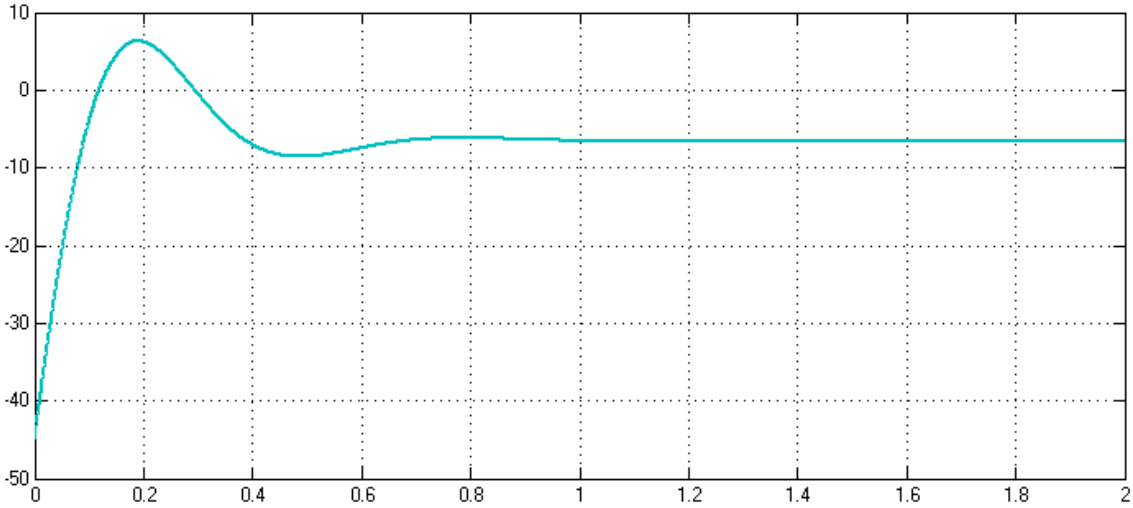


Fig. 31 Elevator actuation [deg]

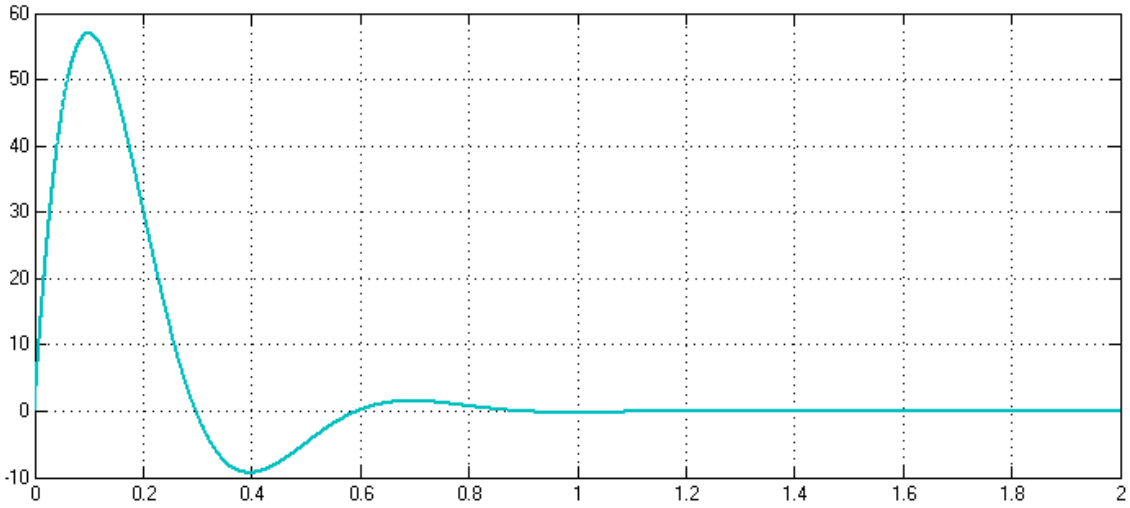


Fig. 32 Pitch rate [deg/s]

Given a pitch command of 10º, it can be seen in Fig. 31 that the response is very quick. There is a steady state error though. The pitch rate is also faithful to the physical capabilities of the aircraft since its values do not exceed the average 65º/s. The ruddervators can be seen as having a very fast operation, but is a speed commonly seen on servo actuated control surfaces. The negative value on the figure to the center means that the elevator is pointing upwards so that the surrounding airmass is making the aircraft to pitch up. Since the desired pitch is different than 0, the ruddervators will continue to be deflected until altitude level flight is commanded, this is because of the inherent stabilization of the aircraft.

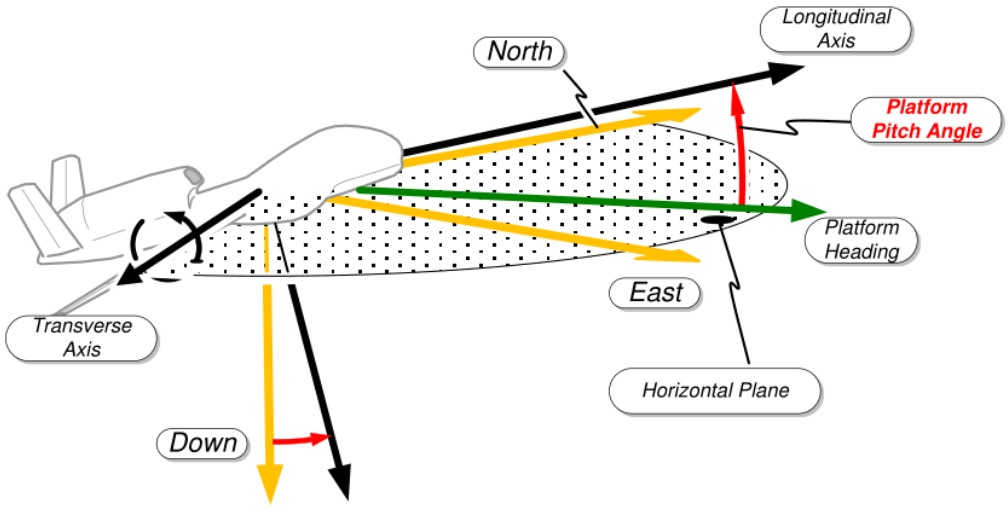


Fig. 33 Pitch angle example

1.6.2.5.2 Altitude Hold Using Pitch

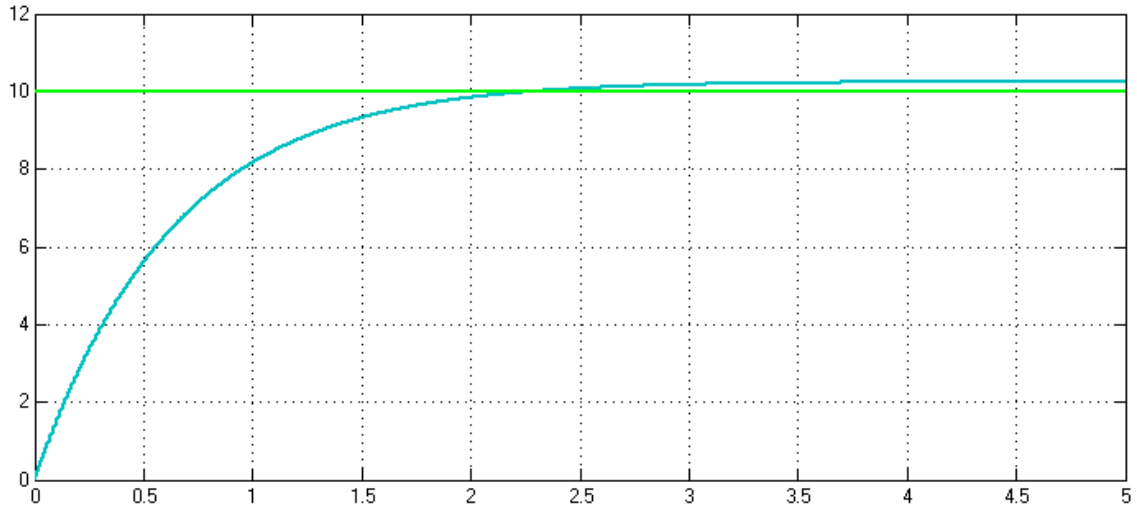


Fig. 35 Commanded altitude and response [m]

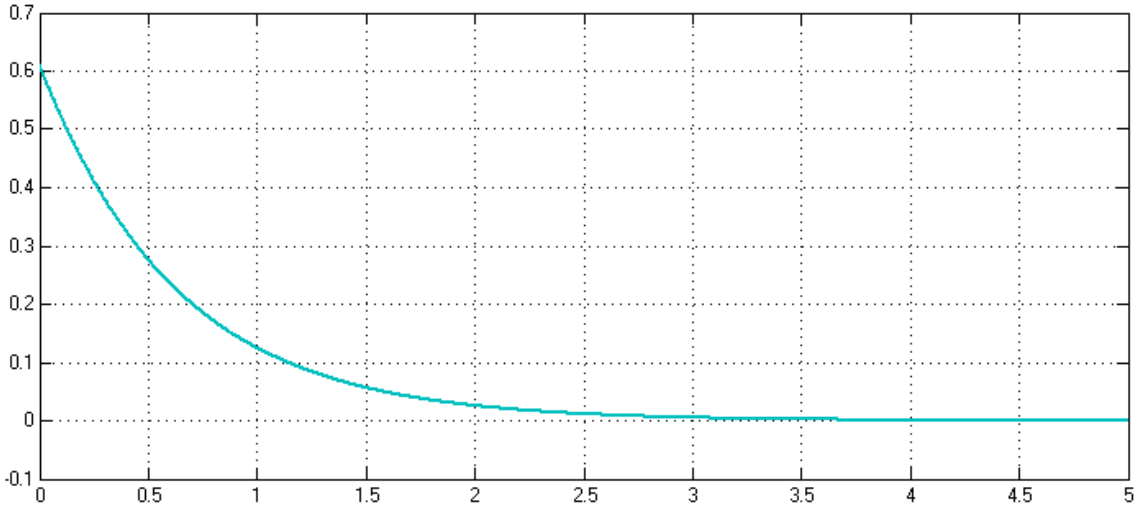


Fig. 35 Commanded pitch [rad]

For a difference of 10 meters in height, the aircraft is able to climb at approximately a maximum rate of at least 15 m/s, which is consistent with its operational speed range. In Fig. 35 we can see that the steepest climb is of 35º. The commanded pitch is slow enough to allow the elevators to pitch up or down accordingly. It can be seen in **¡Error! No se encuentra el origen de la referencia.** that the system gives the command to climb and then, when it reaches the desired altitude, it tells the vehicle stop pitching up/down, which translates to cease the climbing.

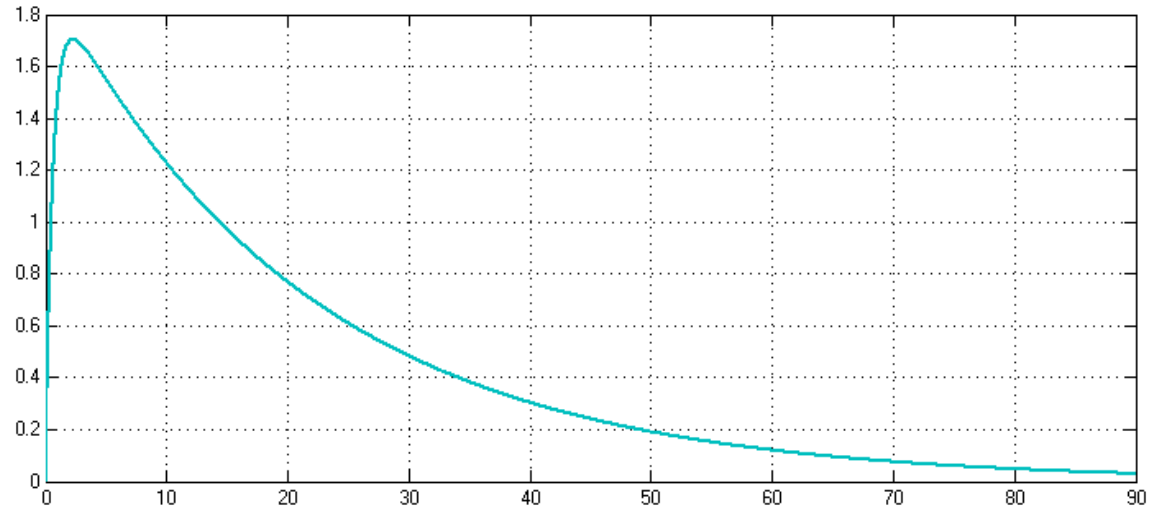


Fig. 37 Altitude during level flight pitch disturbance [m]

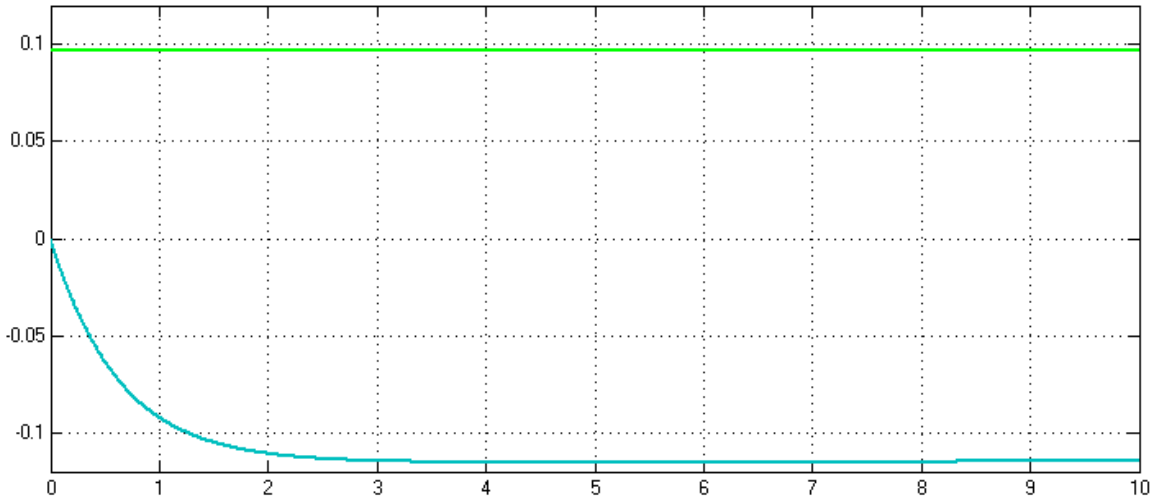


Fig. 37 Pitch disturbance during level flight and pitch signal response [rad]

If the aircraft is **climbing** and enters a region where high winds make its nose to pitch up 5.5º, the vehicle is going to climb around 1 meter in addition. And it will take about one minute for the vehicle to get near its desired altitude. But this additional climbing will only take place if the aircraft is already climbing; if the aircraft is in level flight, from the equation of the disturbance $d_h \triangleq (u \sin\theta - V_a\theta) - v \sin\phi \cos\theta - w \cos\phi \cos\theta$, we have that $d_h \approx 0$ The aircraft will be fighting the disturbance by commanding to pitch down until it desappears, but the transient response will take place for less than 10 seconds. It can be seen that the response against disturbances is slower than the response against a commanded altitude difference. This occurs because the integral gain is twenty times smaller than the proportional gain.

1.6.2.5.3 Airspeed Hold Using Commanded Pitch

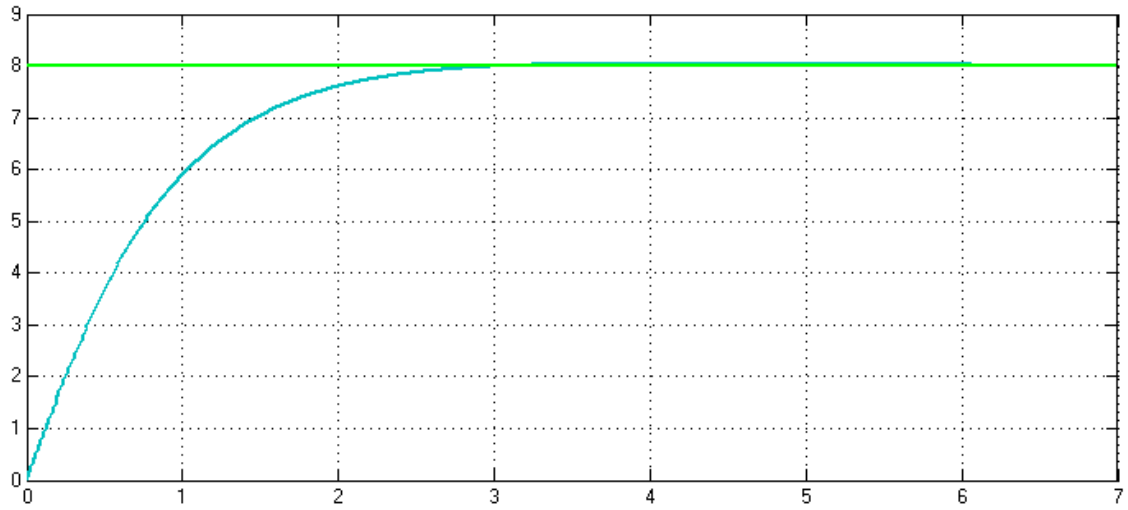


Fig. 39 Commanded airspeed increment using pitch [m/s]

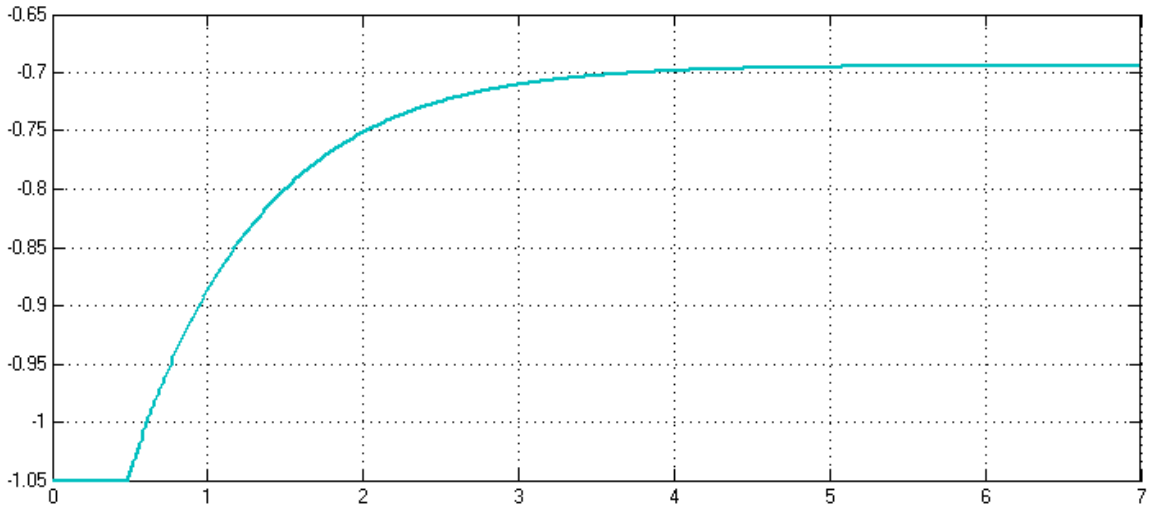


Fig. 39 Commanded pitch [rad]

For the aircraft to increase its velocity over the cruise speed 8 m/s, the vehicle has to pitch down and, since the desired velocity exceeds the cruise speed, the nose needs to continue pointing down to the earth; this can be seen in Fig. 39. Although the aircraft is told to command a pitch angle between ±60º (it can be seen that it has saturated in Fig. 39), the gravity is helping the aircraft increase its speed. As a result, an increment of 28.8 km/h on the speed is achieved in a relatively short period of time, as seen in Fig. 39.

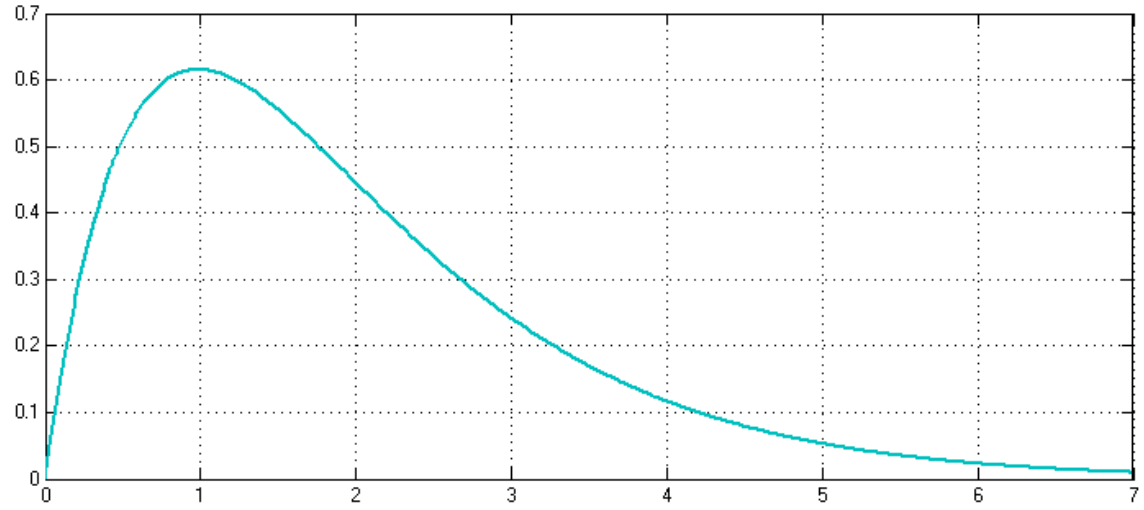


Fig. 41 Airspeed variation by disturbance in pitch [m/s]

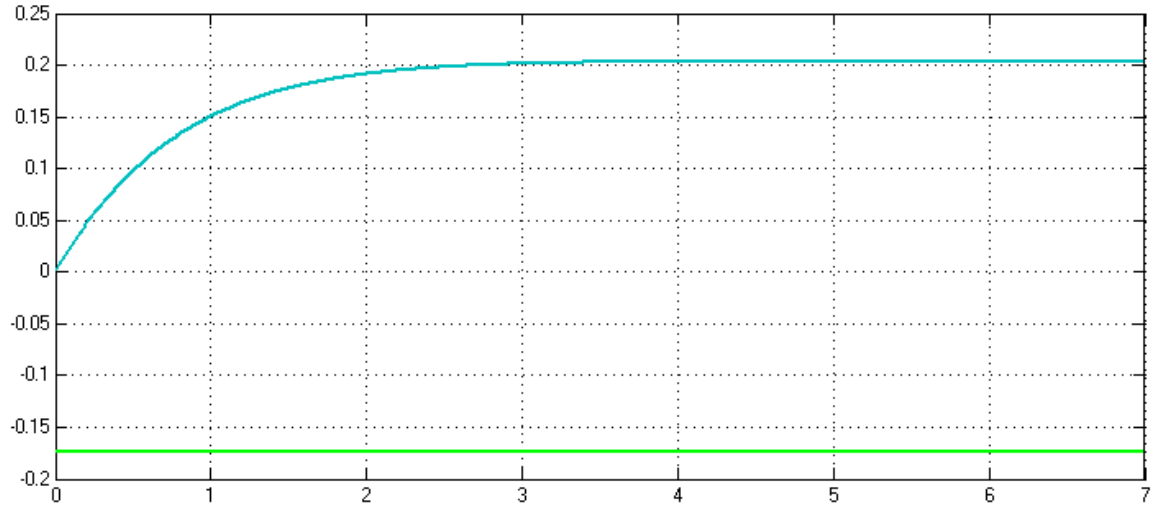


Fig. 40 Disturbance in pitch and commanded pitch [rad]

Although the equation of the disturbance (it includes linearized dynamics and wind) is a bit complex, it can always be translated into a pitch variation. For the aircraft to mitigate the effects of a disturbance resulting on a 10º nose down motion, the commanded pitch must make the vehicle to pitch up. It can be seen that the system is quite slow if compared with the dynamics beforehand calculated, but it also should be taken into account that a wind forcing the nose to pitch down and thus increasing the speed all the time is not an easy one to counteract since the elevator is only slightly deflected.

1.6.2.5.4 Airspeed Hold Using Throttle

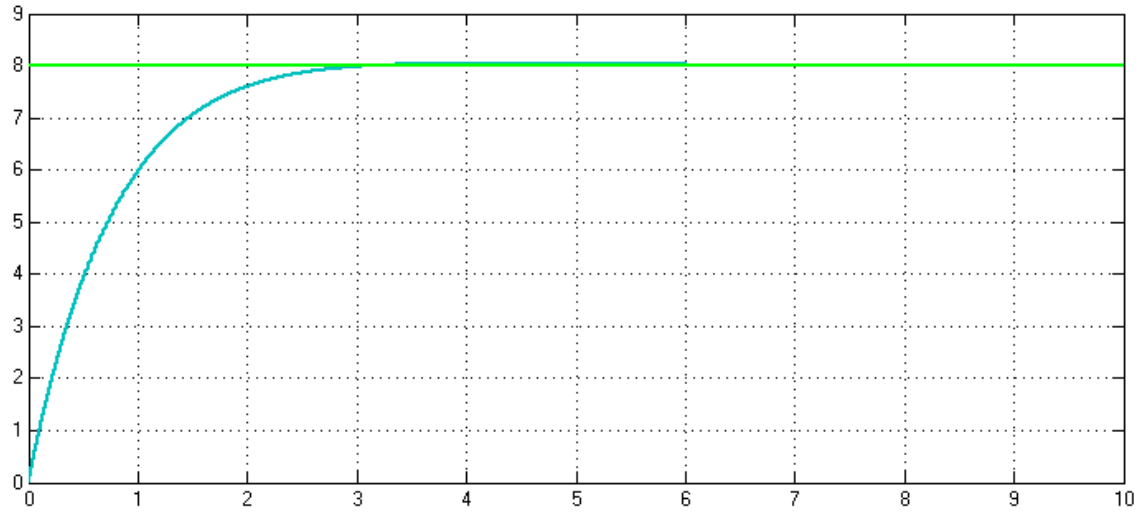


Fig. 43 Commanded airspeed increment using throttle [m/s]

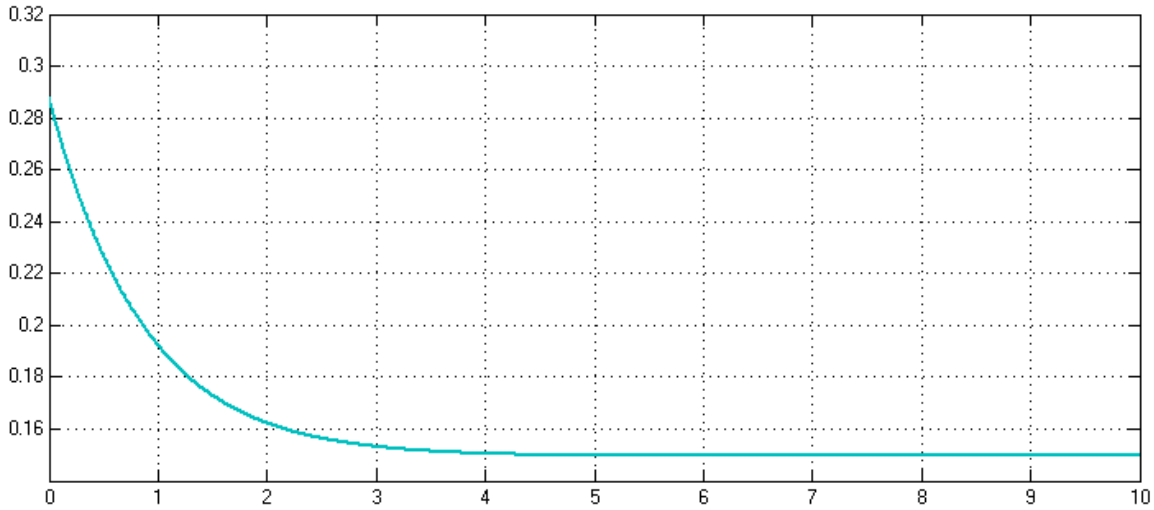


Fig. 43 Throttle setting increment

The cruise speed is achieved by a throttle signal of 0.3, whereas the maximum power the engine can provide corresponds to a signal of 0.6 (although is not convenient to force it continuously). A throttle signal of 0.5 provides the aircraft with enough power to achieve its maximum speed (without risking the integrity of its components). A 30 km/h speed increase command results in the engine working above its nominal value (0 for cruise speed on the second figure). It can be seen that an increment of 30 km/h can be achieved in 3 seconds with pitch control or with throttle control.

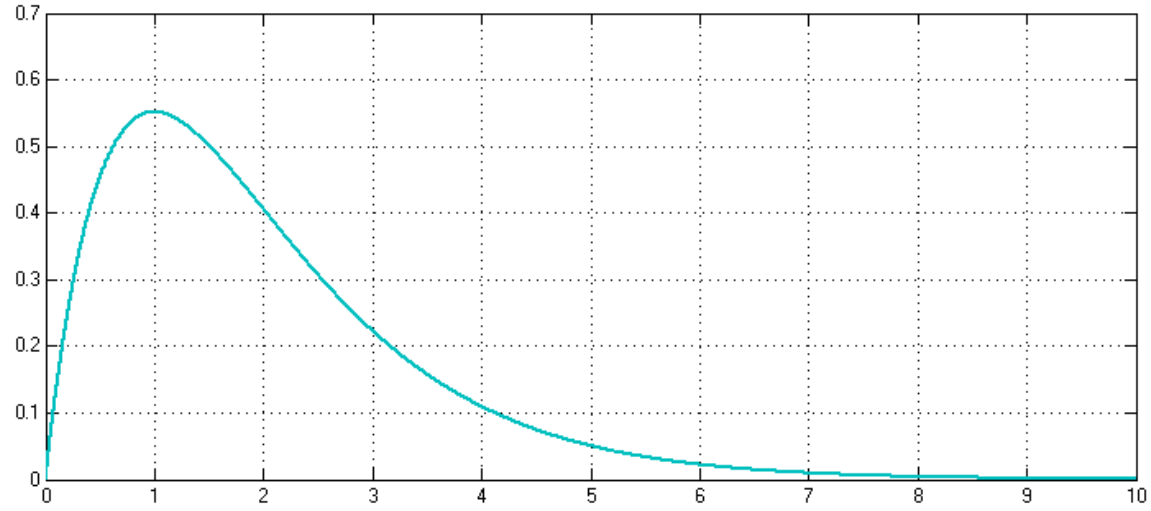


Fig. 45 Response in airspeed for disturbance in engine speed [m/s]

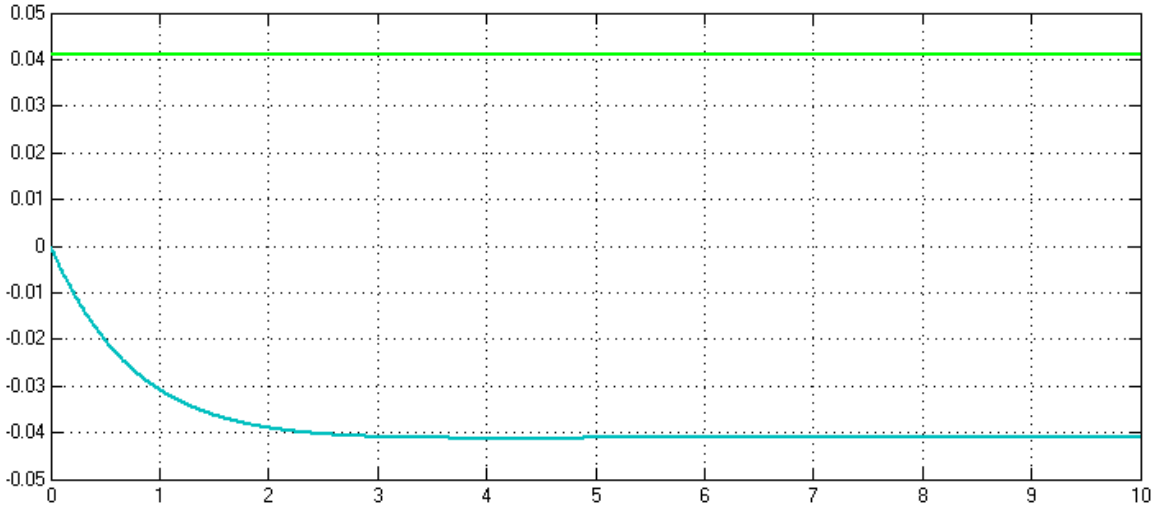


Fig. 44 Disturbance in engine speed and the throttle setting

For the aircraft to mitigate the effects of a disturbance resulting on a increase of the engine rotation, the engine must slow down. And it will continue to do so until the disturbance dissappears.

$\mathbf{k_{d_\theta}}$	-0.4092
$\mathbf{k_{p_\theta}}$	-4.5
$\mathbf{k_{p_h}}$	0.0611
$\mathbf{k_{i_h}}$	0.0028
$\mathbf{k_{pv2}}$	-0.1556

$k_{i_{V2}}$	-0.1232
k_{p_V}	0.036
k_{i_V}	0.0274

Table 4 PID values obtained from the longitudinal controller design

1.7 Flight Controller Pseudocode

Using the blocks developed in the previous sections (path manager, path following and autopilot), the flight controller allows an aircraft to navigate between waypoints in a completely autonomous way.

Following a cascade structure, the path manager block receives the waypoints and defines a path, which will be the input to the path following block. The path following block will calculate the altitude, speed and course that the autopilot will use to finally command the control surfaces. Lastly, to close the loop, the position error returns to the path following block, while the tracking error returns to the path manager.

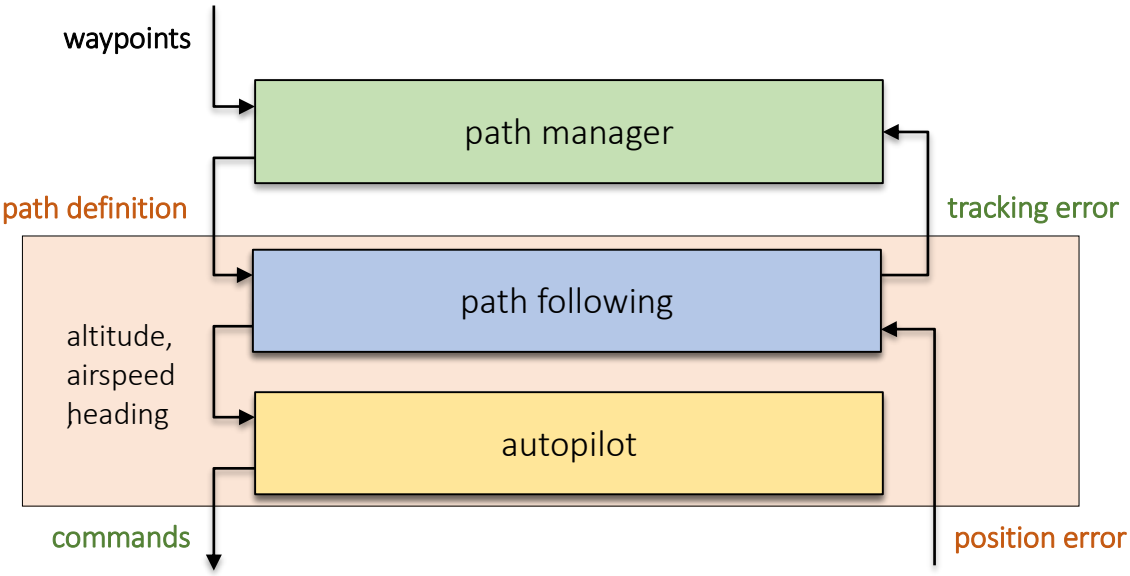


Fig. 46 Flight controller architecture

The algorithm used by the flight controller to pilot the aircraft is reflected below in pseudocode.

path manager:

input (list of waypoints, vehicle position)

1: calculate the tracking error to determine whether to go to the next waypoint

2: **if** afirmative **then** define a path between the reached wp and the next one

3: invoke the flight controller

output (path definition)

flight controller:

input (path definition, vehicle position)

1: calculate the desired height and course using the path following block

2: calculate control surface actuations

3: estimate the vehicle’s position and attitude with sensors or models

output (vehicle position)

<The complete Matlab code can be read in Annex A>

1.8 Autopilot Fine-tuning Using a Model of the Aircraft

Normally, the linear models of the aircraft used to design the autopilot controller do not behave so similarly to the actual vehicle as to use the parameters obtained from the design in reality. In practice, it is common to invest a few hours of flight time in known maneuvers through which, due to the behavior that is being seen, the parameters of the controller can be tuned. The pilot usually adjusts the parameters of the controller until the aircraft's performance is as close as possible to that expected.

Since I don't have a real flying vehicle for this paper, I'm going to reproduce the PID tuning using known flight maneuvers through a simulation, for which I will need a mathematical model of the aircraft.

1.8.1 Mathematical Model of the Aircraft

The mathematical model of the aircraft represents its behaviour in the face of different stimuli. For the generation of the model, therefore, it will be necessary to incorporate equations that combine kinematics, dynamics, forces and moments. The coordinate frames to which the components of these equations refer will also have to be considered. In order to relate them, matrix transformations will be used.

The coordinate frames and equations that comprise the mathematical model of an aircraft are presented below.

1.8.1.1 coordinate frames

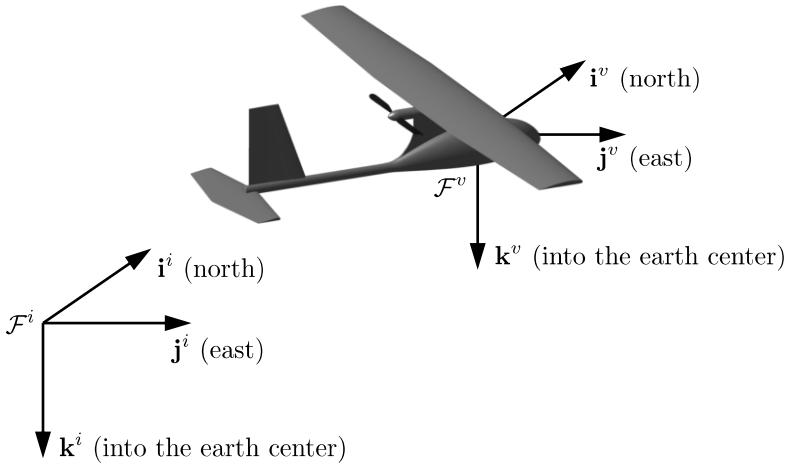


Fig. 50 Inertial frame and vehicle frame

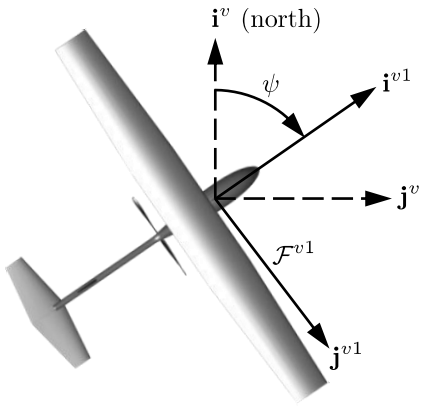


Fig. 50 Vehicle-1 frame

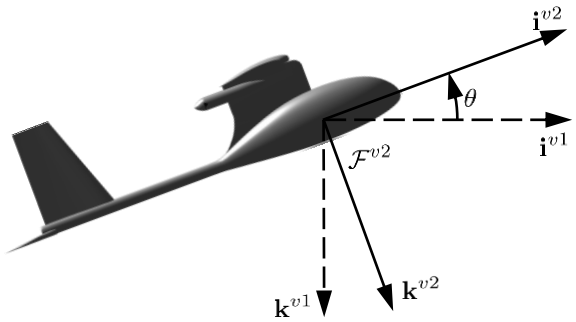


Fig. 50 Vehicle-2 frame

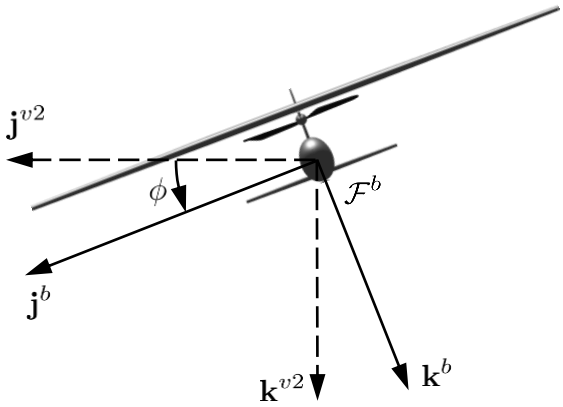


Fig. 50 Body frame

1.8.1.2 Kinematics and Dynamics

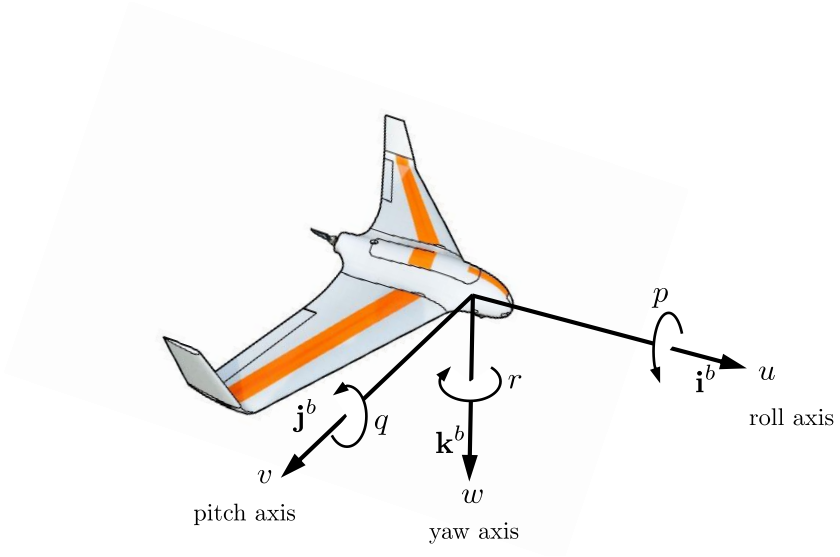


Fig. 51 Definition of the axes of motion

Name	Description
p_n	Inertial North position of MAV expressed along $\mathbf{i}^i$ in $\mathcal{F}^i$.
p_e	Inertial East position of MAV expressed along $\mathbf{j}^i$ in $\mathcal{F}^i$.
p_d	Inertial Down position of MAV expressed along $\mathbf{k}^i$ in $\mathcal{F}^i$.
u	Ground velocity expressed along $\mathbf{i}^b$ in $\mathcal{F}^b$.
v	Ground velocity expressed along $\mathbf{j}^b$ in $\mathcal{F}^b$.
w	Ground velocity expressed along $\mathbf{k}^b$ 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	Body angular (roll) rate expressed along $\mathbf{i}^b$ in $\mathcal{F}^b$.
q	Body angular (pitch) rate expressed along $\mathbf{j}^b$ in $\mathcal{F}^b$.
r	Body angular (yaw) rate expressed along $\mathbf{k}^b$ in $\mathcal{F}^b$.

Table 5 State variables for the equations of motion

$$\begin{pmatrix}\dot{p}_n\\ \dot{p}_e\\ \dot{p}_d\end{pmatrix}=\begin{pmatrix}c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi\\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi\\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta\end{pmatrix}\begin{pmatrix}u\\ v\\ w\end{pmatrix}$$

$$\begin{pmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{pmatrix}=\begin{pmatrix}rv-qw\\ pw-ru\\ qu-pv\end{pmatrix}+\frac{1}{m}\begin{pmatrix}f_x\\ f_y\\ f_z\end{pmatrix}$$

$$\begin{pmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{pmatrix}=\begin{pmatrix}1 & \sin\phi\tan\theta & \cos\phi\tan\theta\\ 0 & \cos\phi & -\sin\phi\\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta}\end{pmatrix}\begin{pmatrix}p\\ q\\ r\end{pmatrix}$$

$$\begin{pmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{pmatrix}=\begin{pmatrix}\Gamma_1 pq-\Gamma_2 qr\\ \Gamma_5 pr-\Gamma_6(p^2-r^2)\\ \Gamma_7 pq-\Gamma_1 qr\end{pmatrix}\begin{pmatrix}\Gamma_3 l+\Gamma_4 n\\ \frac{1}{j_y}m\\ \Gamma_4 l+\Gamma_8 n\end{pmatrix}$$

where

$$\Gamma_1=\frac{J_{xz}\left(J_x-J_y+J_z\right)}{\Gamma}$$

$$\Gamma_2=\frac{J_z\left(J_z-J_y\right)+J_{xz}^2}{\Gamma}$$

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

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

$$\Gamma_5=\frac{J_z-J_x}{J_y}$$

$$\Gamma_6=\frac{J_{xz}}{J_y}$$

$$\Gamma_7=\frac{\left(J_x-J_y\right)J_x+J_{xz}^2}{\Gamma}$$

$$\Gamma_8=\frac{J_x}{\Gamma}$$

and J are the moments of inertia of the airplane.

1.8.1.3 forces

$$\begin{pmatrix}f_x\\ f_y\\ f_z\end{pmatrix}=\begin{pmatrix}-mg\sin\theta\\ mg\cos\theta\sin\phi\\ mg\cos\theta\cos\phi\end{pmatrix}+\frac{1}{2}\rho V_a^2 S\begin{pmatrix}C_X(\alpha)+C_{X_q}(\alpha)\frac{c}{2V_a}q+C_{X_{\delta_e}}(\alpha)\delta_e\\ C_{Y_0}+C_{Y_\beta}\beta+C_{Y_p}\frac{b}{2V_a}p+C_{Y_r}\frac{b}{2V_a}r+C_{Y_{\delta_a}}\delta_a+C_{Y_{\delta_r}}\delta_r\\ C_Z(\alpha)+C_{Z_q}(\alpha)\frac{c}{2V_a}q+C_{Z_{\delta_e}}(\alpha)\delta_e\end{pmatrix}+\frac{1}{2}\rho S_{prop}C_{prop}\begin{pmatrix}(k_{motor}\delta_t)^2-V_a^2\\ 0\\ 0\end{pmatrix}$$

where

$$C_X(\alpha)\triangleq-C_D(\alpha)\cos\alpha+C_L(\alpha)\sin\alpha$$

$$C_{X_q}(\alpha)\triangleq-C_{D_q}\cos\alpha+C_{L_q}\sin\alpha$$

$$C_{X_{\delta_e}}(\alpha)\triangleq-C_{D_{\delta_e}}\cos\alpha+C_{L_{\delta_e}}\sin\alpha$$

$$C_Z(\alpha)\triangleq-C_D(\alpha)\sin\alpha-C_L(\alpha)\cos\alpha$$

$$C_{Z_q}(\alpha)\triangleq-C_{D_q}\sin\alpha-C_{L_q}\cos\alpha$$

$$C_{Z_{\delta_e}}(\alpha)\triangleq-C_{D_{\delta_e}}\sin\alpha-C_{L_{\delta_e}}\cos\alpha$$

and

$$C_L(\alpha)=\left(1-\sigma(\alpha)\right)\left(C_{L_0}+C_{L_\alpha}\alpha\right)+\sigma(\alpha)(2sign(\alpha)\sin^2\alpha\cos\alpha)$$

$$C_D(\alpha) = C_{D_p} + \frac{\left(C_{L_0} + C_{L_\alpha}\alpha\right)^2}{\pi e AR}$$

where e is the Oswald efficiency factor, which ranges between 0,8 and 1,0.

and

$$\sigma(\alpha) = \frac{1 + e^{-M(\alpha-\alpha_0)} + e^{M(\alpha+\alpha_0)}}{(1 + e^{-M(\alpha-\alpha_0)})(1 + e^{M(\alpha+\alpha_0)})}$$

1.8.1.4 moments

$$\begin{pmatrix} l \\ m \\ n \end{pmatrix} = \frac{1}{2}\rho V_a^2 S \begin{pmatrix} b\left(C_{l_0} + C_{l_\beta}\beta + C_{l_p}\frac{b}{2V_a}p + C_{l_r}\frac{b}{2V_a}r + C_{l_{\delta_a}}\delta_a + C_{l_{\delta_r}}\delta_r\right) \\ c\left(C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{c}{2V_a}q + C_{m_{\delta_e}}\delta_e\right) \\ b\left(C_{n_0} + C_{n_\beta}\beta + C_{n_p}\frac{b}{2V_a}p + C_{n_r}\frac{b}{2V_a}r + C_{n_{\delta_a}}\delta_a + C_{n_{\delta_r}}\delta_r\right) \end{pmatrix} + \begin{pmatrix} -k_{T_p}(k_\Omega\delta_t)^2 \\ 0 \\ 0 \end{pmatrix}$$

1.8.2 PID Design Results

1.8.2.1 Roll Loop Design

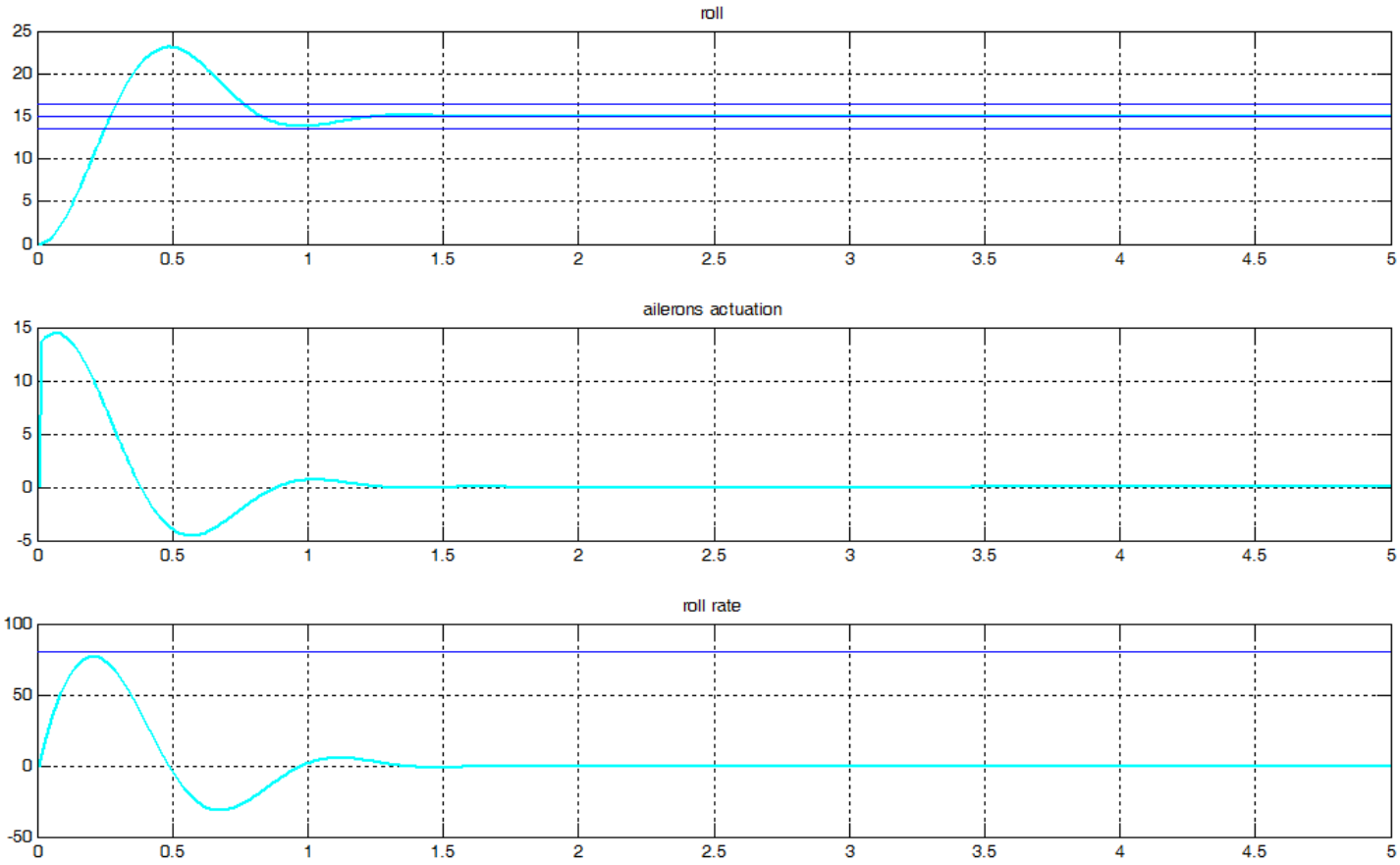


Fig. 52 Roll [deg], ailerons actuation [deg] and roll rate [deg/s]

The new controller parameters make the aircraft react more quickly to roll changes than the ones when designing with the linear model. And, although there is an overshoot, it is small considering that it is about 8 degrees. The actuation on the ailerons is good and the roll rate is no different from that of similar aircrafts. Naturally, when the desired roll angle is reached, the flaps return to their original position.

1.8.2.2 Course Hold Loop

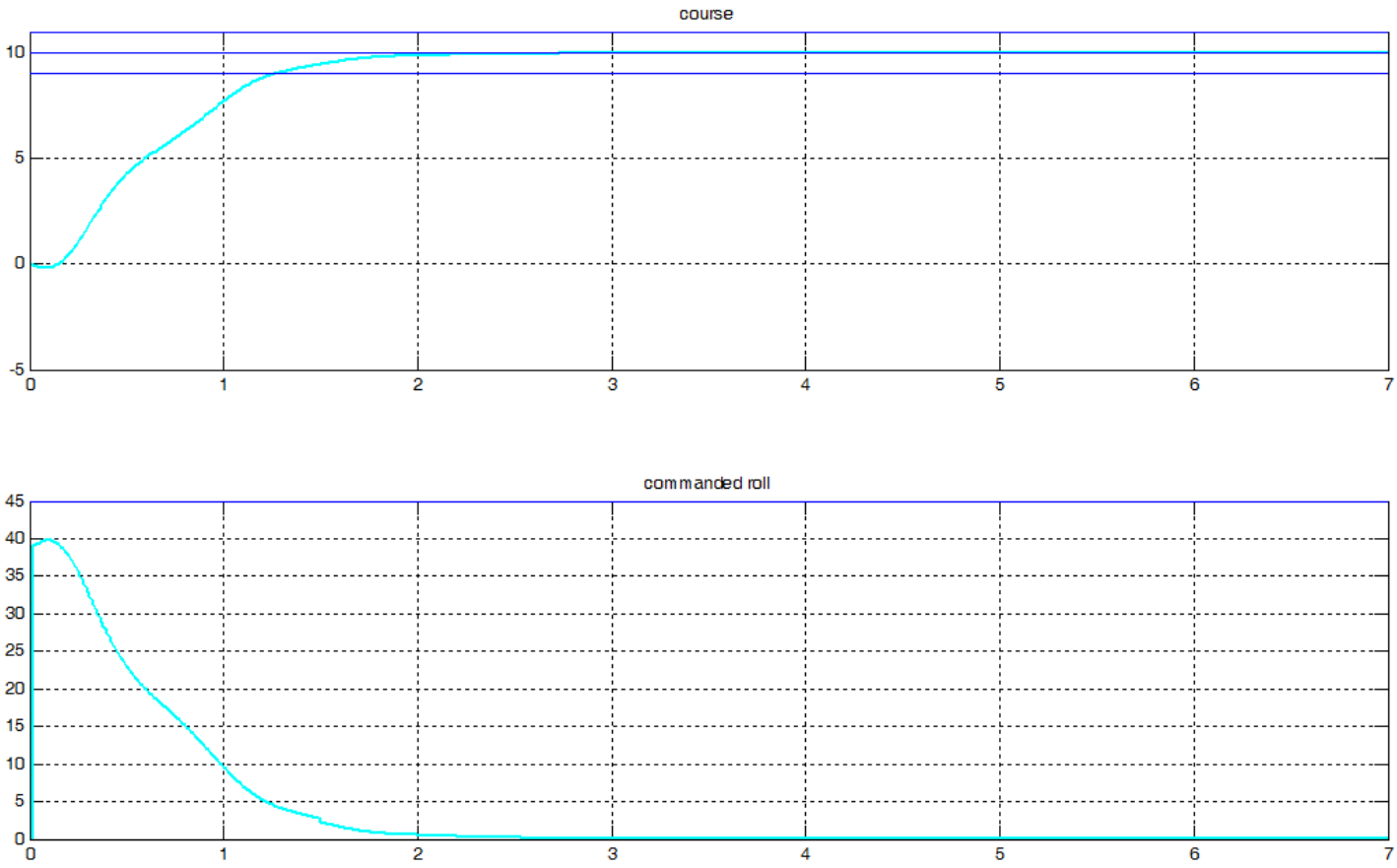


Fig. 53 Course in degrees and commanded roll in degrees

Logically, it takes longer for the aircraft to reach the desired course than for it to reach the desired roll. As the speed of roll change is not limited, it can be seen that, although a large roll is commanded in a very short time, the aircraft does not have time to turn as fast as commanded to and the course response is, therefore, uneven. It can also be seen that, once the desired course is reached, the autopilot commands a null roll.

1.8.2.3 Pitch Attitude Hold

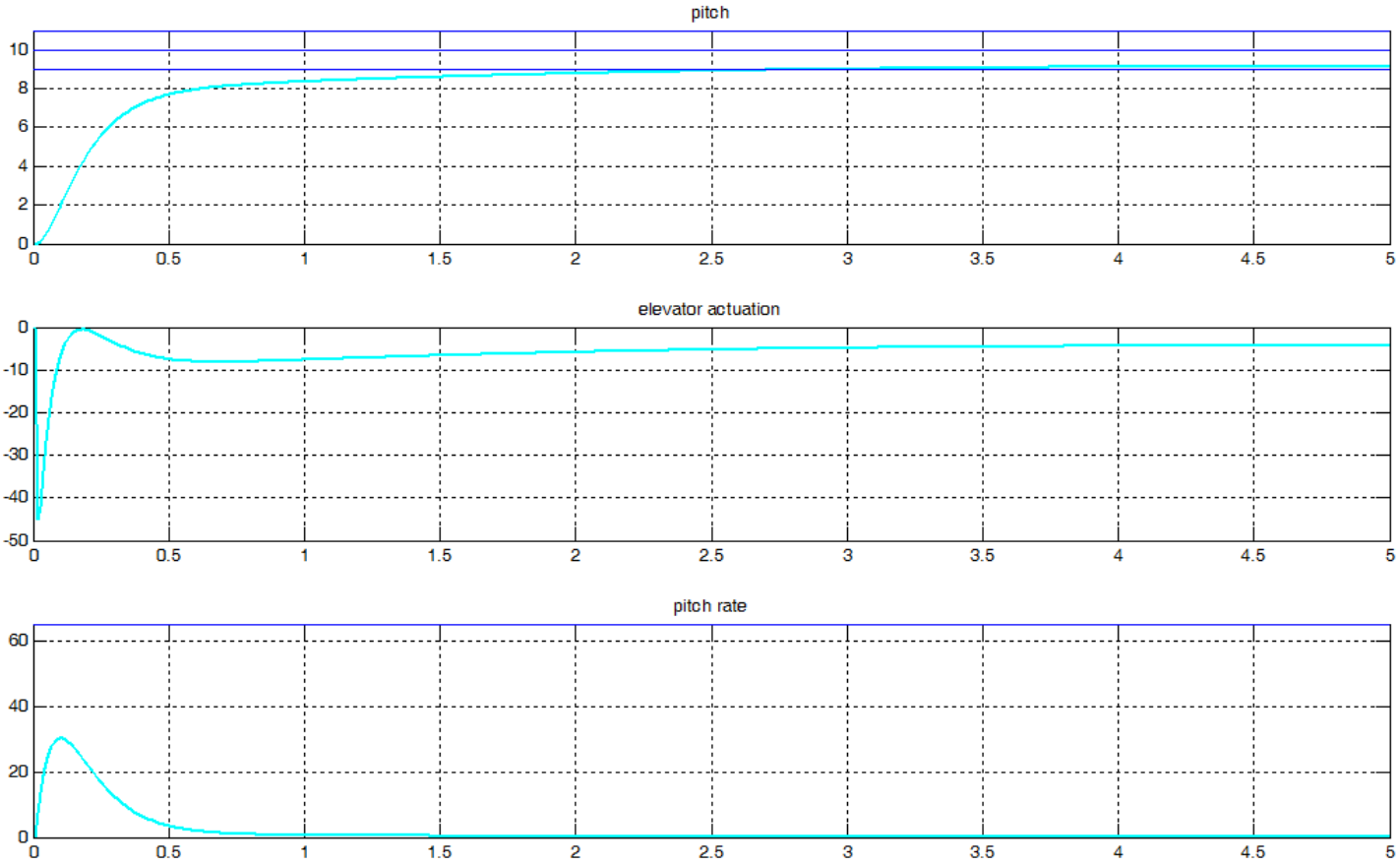


Fig. 54 Pitch [deg], elevator actuation [deg] and pitch rate [deg/s]

As mentioned earlier, the autopilot will never be able to bring the aircraft up to the desired pitch value in steady state. Such a response can be seen in the figure above. The elevator is required to deflect quite a great deal in a very short period of time, but it is feasible for the type of servos built in the type of aircraft being studied. The pitch change speed is, like the roll speed, within the common parameters.

1.8.2.4 Altitude Hold using Commanded Pitch

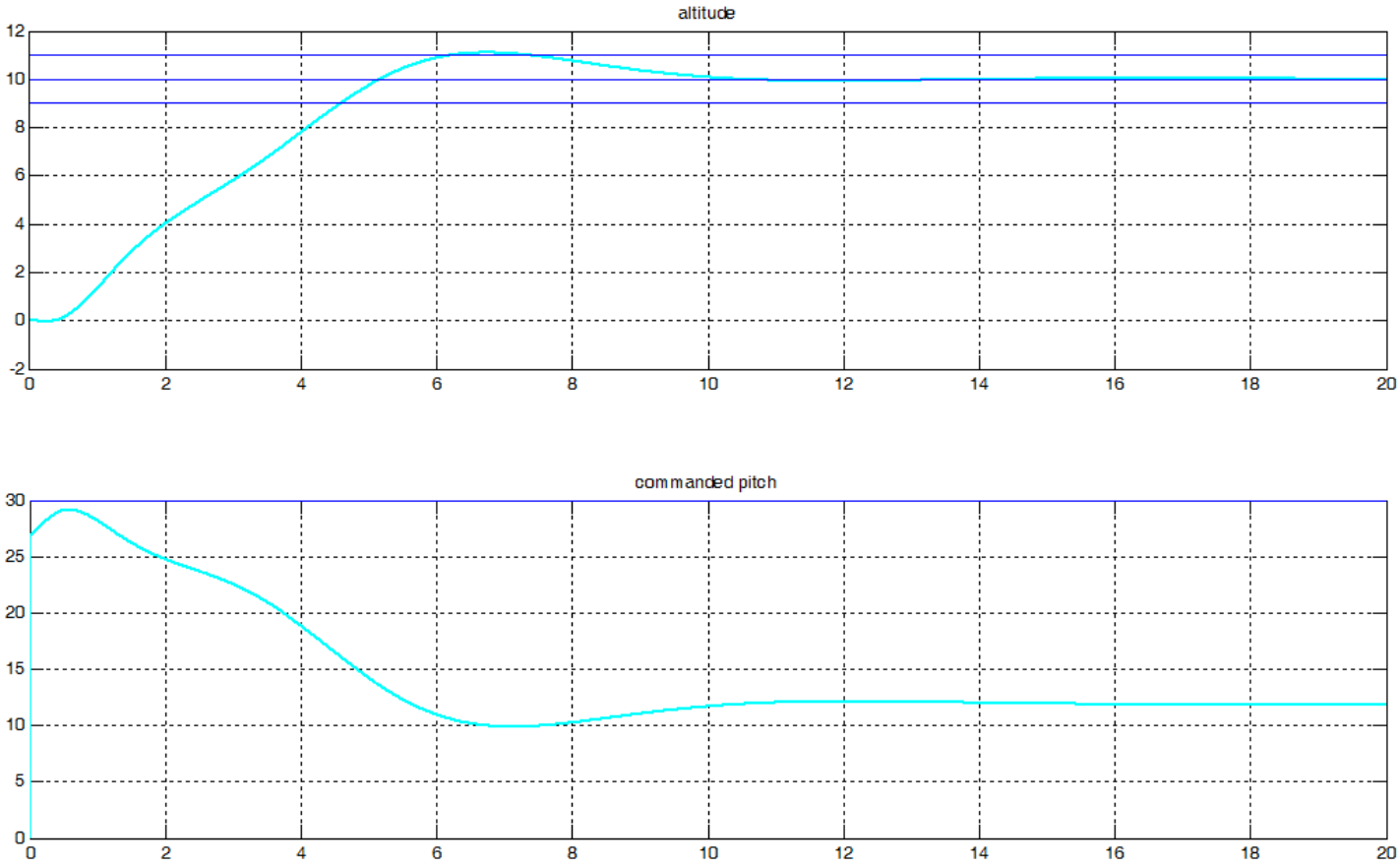


Fig. 55 Altitude [m] and commanded pitch [degrees]

The aircraft response to a request for an altitude change is by far the slowest. This is because the configuration of the spacecraft makes the mass of the fuselage be distributed in such a way that it is harder for it to rotate around the axis that passes through its wings than it is for it to rotate around the axis that passes through its nose. Unlike the previous cases, once the desired altitude has been reached, a zero pitch is not commanded. This is because it is very complicated to design the altitude hold block without linking the airspeed one. As these blocks are interconnected, the response of one will always include the response of the other.

1.8.2.5 Airspeed Hold using Commanded Pitch

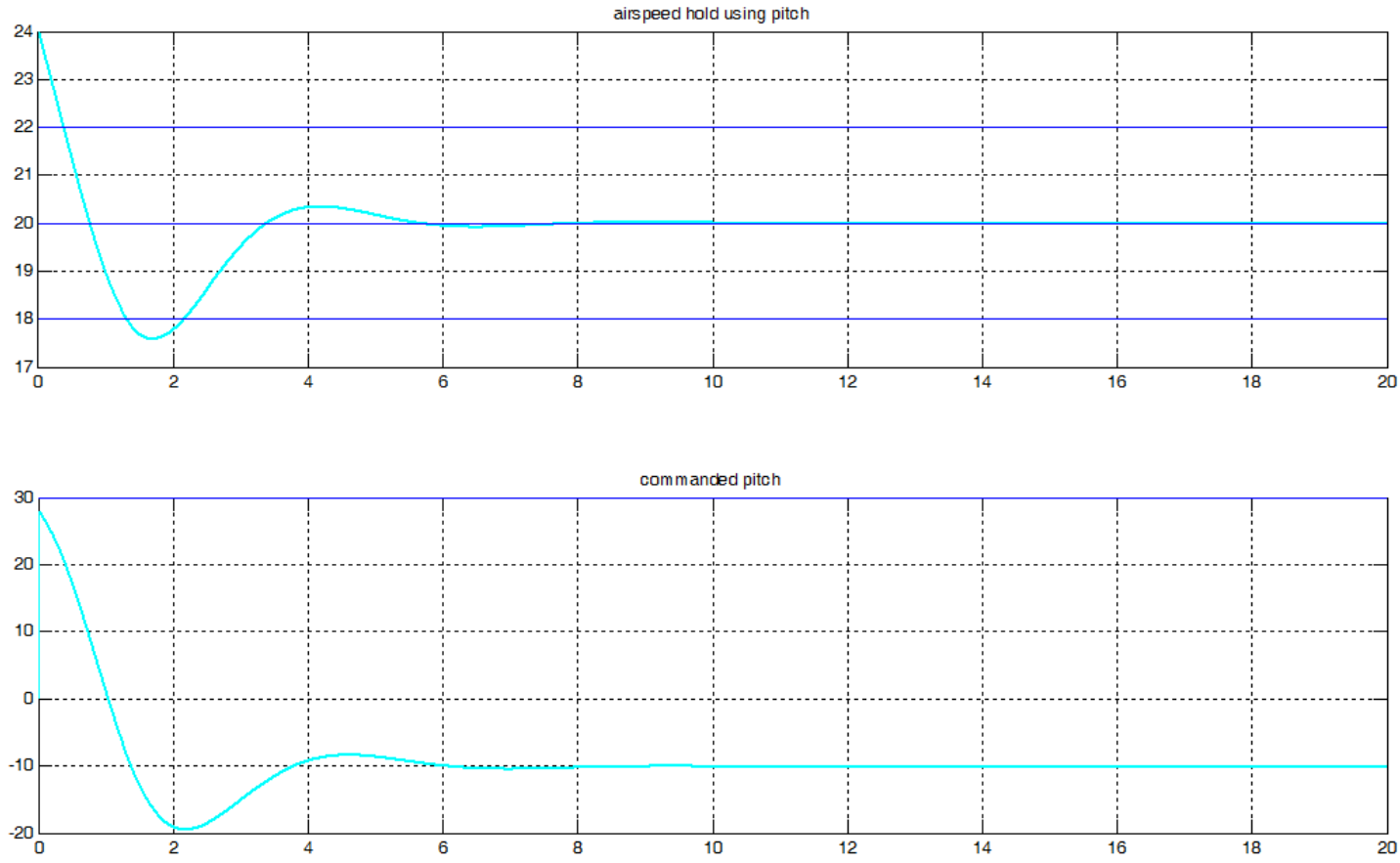


Fig. 56 Airspeed hold using pitch [m/s] and commanded pitch [degrees]

To decrease the airspeed using the pitch angle, the vehicle is forced to pitch up. As a result, the aircraft increases its potential energy and decreases its kinetic energy, thus reducing its speed. As with the altitude hold using the pitch block, not unlinking these blocks causes the steady state response not to return to zero.

1.8.2.6 Airspeed Hold using Throttle

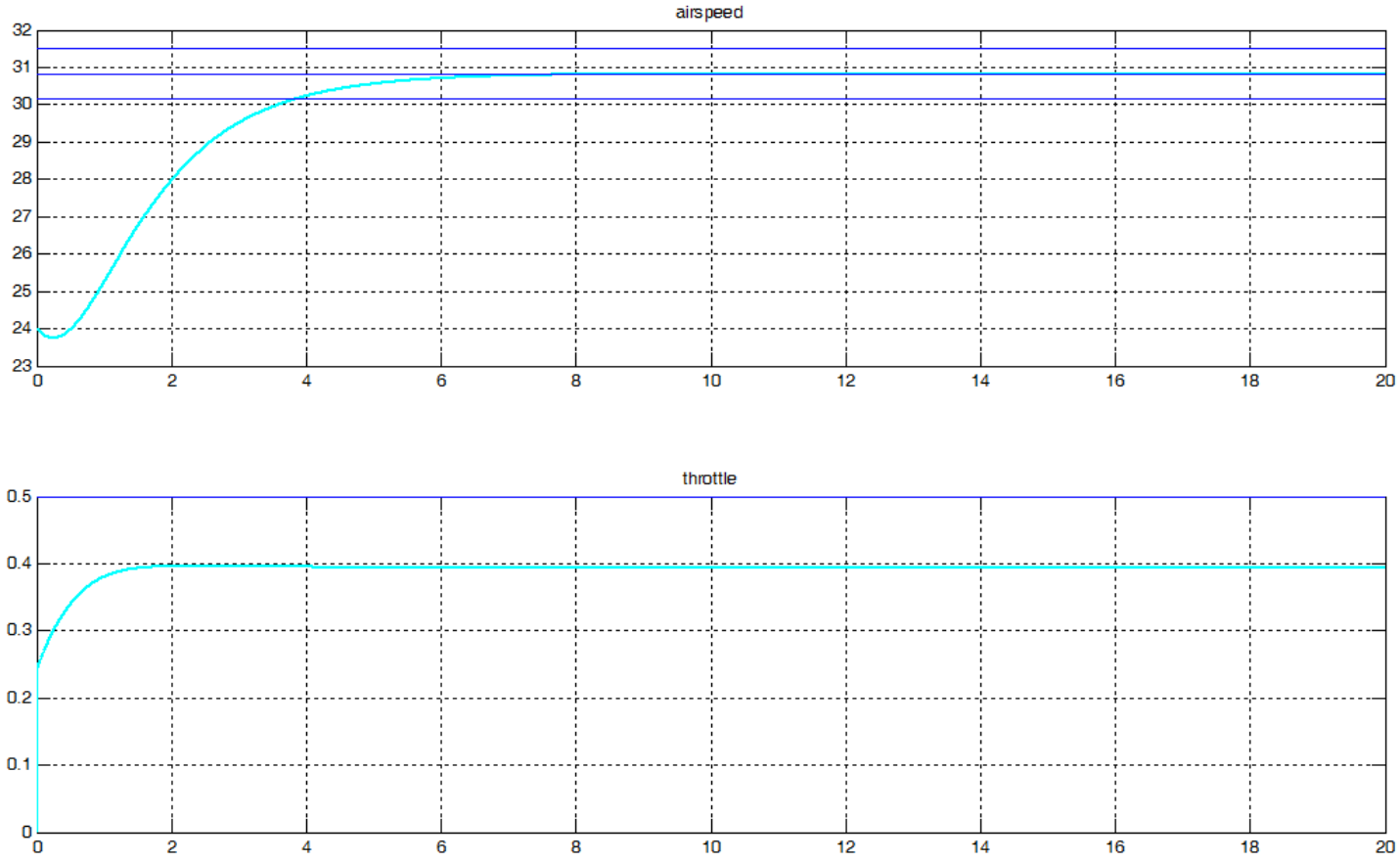


Fig. 57 Airspeed [m/s] and commanded throttle

The acceleration achieved by using the engine alone may vary depending on the power of the engine being used, so the rate of change is not solely related to the controller parameter. Naturally, after the desired speed value is reached, the motor must run at the same power to maintain this speed.

1.8.2.7 PID Values

$\mathbf{k_{p_\phi}}$	0,9
$\mathbf{k_{i_\phi}}$	2,7
$\mathbf{k_{d_\phi}}$	0,01
$\mathbf{k_{p_\chi}}$	3,9

$\mathbf{k_{i_\chi}}$	0,07
$\mathbf{k_{p_\theta}}$	-5,1
$\mathbf{k_{i_\theta}}$	-1,2
$\mathbf{k_{p_h}}$	0,047
$\mathbf{k_{i_h}}$	0,009
$\mathbf{k_{p_\beta}}$*	-0,01
$\mathbf{k_{p_{V_2}}}$	-0,12
$\mathbf{k_{i_{V_2}}}$	-0,1
$\mathbf{k_{p_V}}$	0,036
$\mathbf{k_{i_V}}$	0,0274

Table 6 PID values obtained from the fine-tuning design

* The integral gain of the sideslip hold loop does not work well with the algorithm used in the autopilot. This is because the sideslip angle is strongly influenced by wind and vehicle inertia. If the rudder fails to reach a zero sideslip angle in a short time, the integrator will grow and a wind-up situation will occur. This situation can cause a large overshoot and may potentially destabilize the aircraft. One way to avoid all this would be to provide the autopilot controller with a timeout to reset the integrator or use a simple yet effective P type controller.

2. SIMULATION

2.1 Introduction to Simulation

To test the flight controller, a simple simulation will be carried out in Matlab. The simulation will consist of a photogrammetric survey mission and will include: assisted takeoff, flying over an area following a series of waypoints and net landing. A photogrammetric mission is especially interesting for testing a flight controller because the area to be photographed has to be covered in straight lines and with the appropriate overlapping between them.

The place chosen for the simulation is the Sabangau River in Borneo, which is currently suffering from a major environmental problem.

2.2 Four decades of forest persistence, clearance and logging on Borneo

The native forests of Borneo have been impacted by selective logging, fire, and conversion to plantations at unprecedented scales since industrial-scale extractive industries began in the early 1970s. It is estimated that 75.7% (558.060 km^2) of Borneo's area (737.188 km^2) was forested around 1973 and that the 1973 forest area had declined by 30.2% (168.493 km^2) in 2010. The highest losses were recorded in Sabah and Kalimantan with 39.5% and 30.7% of their total forest area in 1973 becoming non-forest in 2010. There is still hope for biodiversity conservation though. Protecting logged forests from fire and conversion to plantations is an urgent priority for reducing rates of deforestation in Borneo [5].

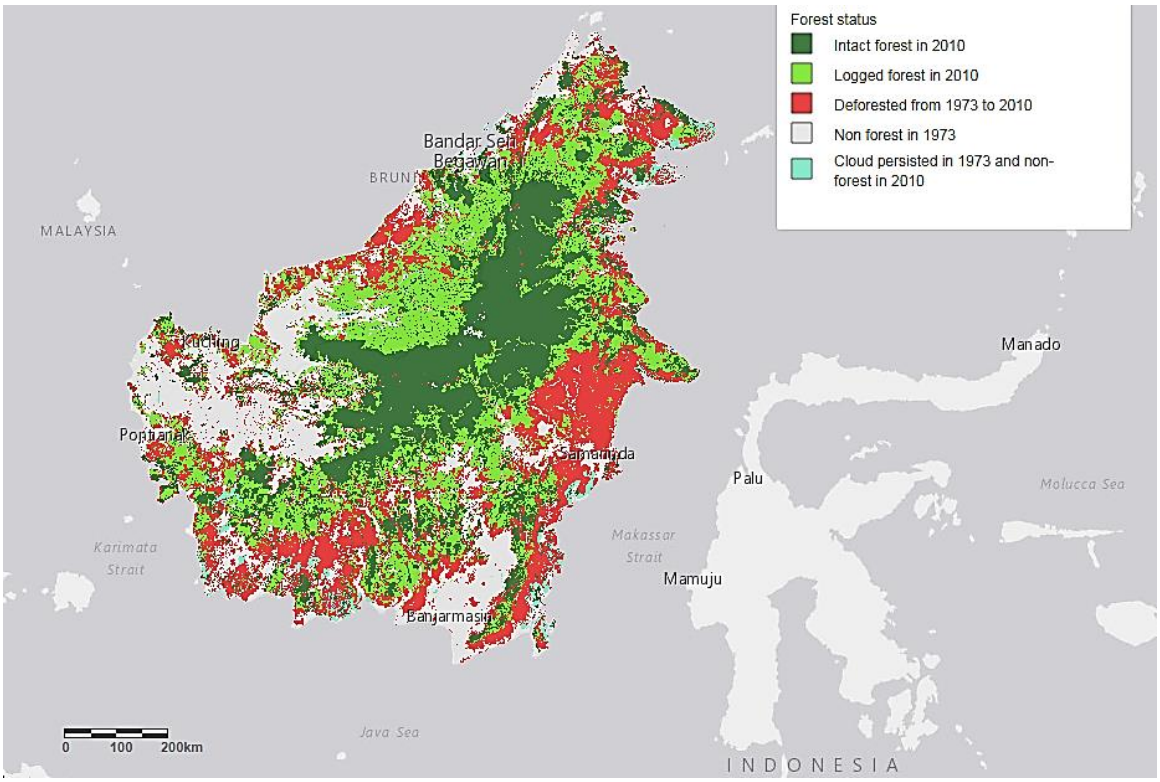


Fig. 58 Borneo Forests status

2.2.1 Sabangau National Park

Sabangau National Park is a national park in Central Kalimantan –the Indonesian part of the island of Borneo-. The park is centered on Sabangau River. It flows through the Kelompok Hutan Kahayan –a 5.300 km^2 peat swamp forest-, between the Katingan and Kahayan rivers. The forest is a dual ecosystem, with diverse tropical trees standing on a 10m - 12m layer of peat (partly decayed and waterlogged plant material), which in turn covers relatively infertile soil.

The forest is home to the world's largest orangutan population, estimated at 6.910 individuals in 2003, and other rare or unique species. Since the forest has been damaged by legal and illegal forestry, there is no longer any continuous forest cover where orangutans may cross the river. The western part, however, is now protected as either National Park or National Laboratory Research Area.

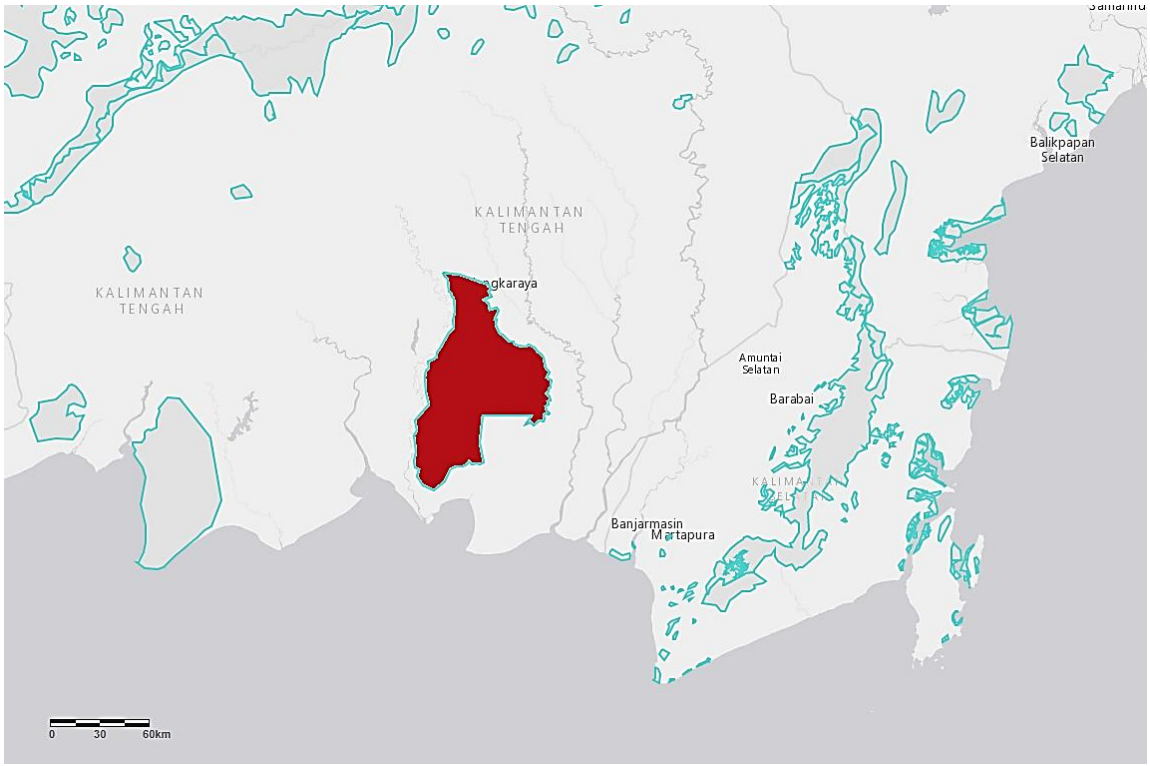


Fig. 59 Location of Sabangau National Park

2.2.2 Biodiversity Conservation in Borneo

In order to protect the region, further poaching, logging, or fire damage must be prevented and natural hydrological conditions must be restored. The use of unmanned aerial vehicles can help achieve this goal to a considerable extent.

To simulate a contribution to the cause, the mission will take place over the river Sebangau. The river will be flown over to carry out a photogrammetric study of the area. The study could serve as a basis for aerial seed planting, forest sizing, plant monitoring and reforestation missions.

2.2.2.1 Aerial Seed Planting

Large-scale reforestation significantly reduces the carbon dioxide in the atmosphere, thus counteracting global warming. In addition, new trees fight erosion, promote biodiversity, and protect the habitat of local wildlife.

Natural regeneration may develop a more reasonable and stable structure and is an easy way to restore vegetation of felled lands. However, in large and remote areas where seeds or coppices are rather limited, natural regeneration and subsequent forest succession may take much longer. To overcome this issue, aerial seeding is commonly applied to promote vegetation recovery and thus shorten the time required for ecosystem restoration. Aerial seeding has a long history –more than 50 years- in China, and has been accomplished for 1.364 km^2 in 2012 alone. Aerial seeding has also been widely used around the world where had significantly accelerated forest vegetation recovery.

2.2.2.2 Soil Analysis, Forest Sizing and Post Planting Monitoring

Aerial photogrammetry provide detailed information on the soil surface which can be used for monitoring surface water flows which is very important for planting planning. It also has been used to monitor land use and to generate contour maps of forest areas.

A series of overlapping photographs are taken over the study area so that any single point on the ground surface can be identified in 3 separate images. The position of that point is then triangulated. These are then brought together to create a single visual image of the area and a 3-dimensional representation of the ground surface.

Once the ground surface is calculated: water flow models can be applied to the calculated soil surface model to show where water is likely to flow, or forest areas can be calculated.

Multispectrum images are taken to analyze soil, water stress and tree species.

2.2.2.3 Fire Spotting

The biggest long-term problem inside Borneo's peat-swamp forests is drainage, the artificial construction of canals and channels inside the forest that causes the peat to dry out and sends the fire-risk skyrocketing.

Even though most peat-swamp forests are now protected in Kalimantan, under the Indonesian government's moratorium on conversion of peatlands, they remain threatened by the long-term consequences of rampant illegal logging and failed development programmes in this habitat during the late 1990's and early 2000's.

During this time, a network of large canals were dug to drain the peatland for planned agricultural. Inside the forest, illegal loggers cut narrow channels into the peat to float out felled timber, hundreds of them inside Sabangau alone, several stretching over 10 km into the forest. These channels remain long after the immediate problem has been solved, and drain the once water-logged peat every dry-season.

In their natural state these forests are permanently flooded and new peat accumulates, but when the peat is drained it dries out, oxidises and degrades, putting the whole ecosystem at risk. The crumbling peat surface undermines shallow-rooted trees, and little water is retained in the peat by the end of the dry season. Dried peat is highly flammable and fires frequently break out in the forest margins and the surrounding sedge swamp, sometimes burning large areas.

UAVs can be equiped with microwave radiometers, smoke sensors and thermal cameras for fire-spotting; effectively reducing fires and geolocating hot-spots.

2.2.2.4 Reforestation

Reforesting burnt and cleared areas of forest is an essential part of conservation work. Replanting trees prevents soil erosion, expands wildlife habitats and reduces fire-risks; it also contributes to the global fight to stop climate-change.

The degraded sedge swamp on the edge of the Sabangau River, a barren grass and fern covered landscape is a very interesting site to use aerial seeding by UAVs.

The first step in the process would be to use mapping and surveying to locate the exact micro-site in which a seed will thrive.

After the mapping, it is time for seeding. Armed with a payload of seed pellet -each a mixture of fertilizer, natural pest deterrents, and the seed- and a compressed air gun, the aircraft fires hundreds of pellets into the ground.

Because the machine can plant seeds so precisely, obstacles like rocks and downed trees aren't an issue. Resolutions of the order of 3 cm are needed though.

2.3 Mission Planner

To get quality and useful information out of a photogrammetric survey, a thorough planification must be carried through. The information to be gathered includes: purpose and use of the images, scale in which the final product will be shown, accuracy and precision. At the same time, maps of the area of interest, aerial images and any relevant information of the kind are gathered to carefully plan the flight. The camera specifications, cruise speed of the aircraft, endurance and ceiling (among others) are also very important.

2.3.1 Scale Determination

The most convenient scale for the image depends on the purpose for which it is intended. In general, it should be borne in mind that the smallest detail of the terrain appreciable on the photograph is that which has a size of 0.1 mm or his equivalent in pixels. Since I am planning a photogrametry mission to study where to plant tree seeds, I need to get resolutions of the order of the ones used to count trees (although I expect seed planting to be a bit more demanding, but that is ok for this project). For that reason, a ground sample distance (GSD) of, say, 3 cm is desired. This means that the distance beween pixel centers needs to be 3 centimeters or 30 000 microns. The camera I have chosen for this task (a widely used camera in photogrammetric surveys), the PhaseOne's iXU-RS 1000 has a pixel size of 4.6 microns, thus, the scale would be: 30 000/4.6 or 1:6 500.

2.3.2 Flight Height Determination

The flight altitude $\boldsymbol{Z}$ will depend on the intended scale of the photographs $\boldsymbol{E}$, taking into account the focal length $\boldsymbol{f}$ of the camera.

$$Z = E \times f$$

Due to the chosen lens, PhaseOne's Rodenstock 90 mm *f*, the desired flight height is $Z = 6\,500 \times 0.09 = 585\ m$.

2.3.3 Ground Surface Covered for each Photo without Overlapping

If the sensor is a $\boldsymbol{l_1 x l_2}$ matrix, it will contain the image of a terrain of dimensions $\boldsymbol{L_1 x L_2}$ and area **S** in accordance to:

$$L = l \times E$$

$$S = L_1 \times L_2$$

According to the camera specifications, the sensor size is 53.4 mm x 40.0 mm, thus the size of the ground surface covered given our scale is: 347.1 m x 260 m, covering a total surface area of 90 246 m^2 or 9.02 Ha.

2.3.4 Overlapping

In order to ensure the total coverage of the area of interest and that the photographs allow stereoscopic examination, it shall be considered a forward overlap $\boldsymbol{u}$ of 60% and a side overlap $\boldsymbol{v}$ of 40% in accordance with similar studies.

In favor of compensating a possible lateral error, a new flight path will be added to the total number of flight paths calculated to cover the area. Likewise, in order to counteract a longitudinal error, four photographs will be added to the total of those calculated as necessary for each path (two at the beginning and two at the end).

2.3.5 Useful Advance

The useful advance ***B*** is the distance between the shooting points of two consecutive pictures.

$$B = L_2 \times \left(1 - \frac{u}{100}\right)$$

In this case $B = 260\left(1 - \frac{60}{100}\right) = 104\ m.$

2.3.6 Flight Line Spacing

Assuming $\boldsymbol{l_1}$ as the longest side of a sensor, ***E*** the chosen scale and ***v*** the side overlap; the lateral separation ***A*** between flight lines shall be

$$A = L_1 \times \left(1 - \frac{v}{100}\right)$$

In our case $A = 347.1\left(1 - \frac{30}{100}\right) = 243\ m.$

2.3.7 Shooting Interval

The shooting interval is the elapsed time between shots for the required longitudinal overlap to occur. Assuming ***B*** as the distance between two shooting points, ***V*** being the resulting speed of the plane and ***I*** being the shooting interval:

$$I = \frac{B}{V}$$

Since the simulation will be carried on by Textron's Aerosonde UAV -with a cruise speed of 60 knots or 30.87 m/s-, the shooting interval will be $I = \frac{104}{30.87} = 3.369\ s$. This is for guidance only.

2.3.8 Number of Flight Paths

This is the number of flying lines needed to cover the entire area to be photographed, taking into account the lateral overlap. The area

$$NFP = \frac{Area\ width}{A} = \frac{1196}{243} = 5$$

2.3.9 Number of Shots per Flight Line, Total Number of Shots and Flight Time Above Target

$$NSFL = \frac{path\ lenght}{B} + 4 = \frac{2400}{104} + 4 = 27$$

$$TNS = NSFL \times NFP = 27 \times 5 = 135$$

$$FTAT = I \times TNS = 3.369 \times 135 = 455\ s = 7.6\ min$$

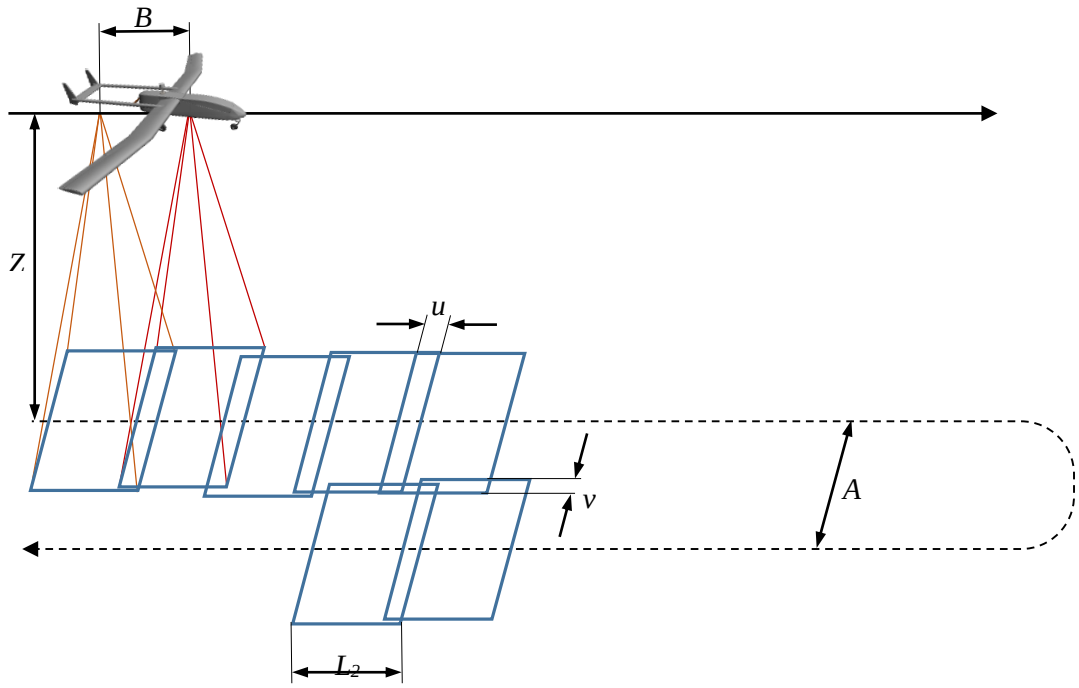


Fig. 60 Photogrametry nomenclature

2.3.10 Mission Plan

Fig. 61 Flight plan over Sabangau River (Borneo) 2494.3 m x 1196 m (298.3 Ha) [ESA's Sentinel-2] Scale 1:9000

2.3.11 Geolocation

Location	Latitude	Longitude	Elevation
Base camp	-2.316570º	113.908020º	14.7 m
Sabangau river A	-2.396338º	113.935791º	17.5 m
Sabangau river B	-2.407315º	113.957935º	19.0 m

Table 7 Latitude, longitude and elevation of the mission's points of interest

Although there is an elevation difference between locations, the area we are interested in -the vicinity of the river- does not have a substancial difference in height for the camera, so the aircraft will not change its flight altitude once over the river.

Fig. 62 Aerial view of Sabangau river and the Base camp [ESA's Sentinel-2]

2.3.12 Longitude-Latitude-Altitude to Earth-Centered-Earth-Fixed Conversion

The Earth-centered-Earth-fixe (ECEF) frame rotates along with the Earth. It is depicted in the picture below, and can be defined as having the origin at the center of mass of the Earth, the z-axis passing through the conventional terrestrial pole (mean spin axis of the planet), the x-axis passing through the intersection of the equatorial plane and the reference meridian (Greenwich) and finally, the y-axis completing the right-hand orthogonal coordinate system in the equatorial plane.

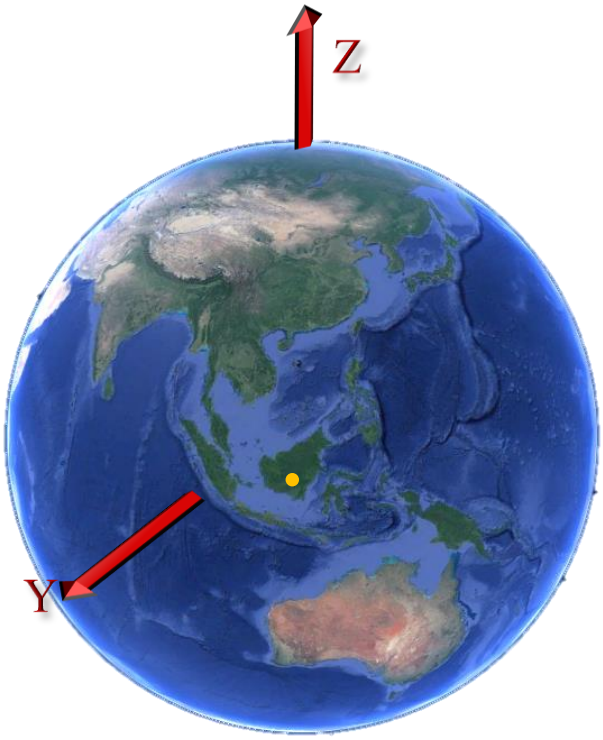


Fig. 63 ECEF coordinate system

The basic formulas for converting from latitude, longitude and altitude (above reference ellipsoid) to Cartesian ECEF are given in the Astronomical Almanac in Appendix K. They depend upon the equatorial earth radius ***a*** and the "flattening" parameter (the ratio of the difference between the equatorial and polar radii to a) ***F***.

For WGS84 (an earth centered datum the Global Positioning System uses), a = 6378137 m and (1/F) = 298.257224.

Given geodetic (not geocentric) latitude

$$C = \frac{1}{\sqrt{\cos^2(latitude) + (1-F)^2 \sin^2(latitude)}}$$

$$S = (1-F)^2 C$$

then a point in LLA has ECEF coordinates:

$$x = (aC + h)\cos(latitude)\cos(longitude)$$

$$y = (aC + h)\cos(latitude)\sin(longitude)$$

$$z = (aS + h)\sin(latitude)$$

Location (ground)	x	y	z
Base camp	-2 582 772.4 m	5 826 155.3 m	-256 085.3 m
Sabangau river A	-2 585 450.0 m	5 824 573.7 m	-264 898.4 m
Sabangau river B	-2 587 680.8 m	5 823 529.0 m	-266 111.2 m

Table 8 ECEF coordinates of the mission's points of interest

2.3.13 ECEF to Local-Level Frame Conversion

A local-level frame (LLF) serves to represent a vehicle's attitude and velocity when near the surface of the Earth. This frame is defined as having its origin at the center of the sensor frame, the x-axis pointing to true north (different from magnetic north), the y-axis pointing to the east and the z-axis completing the right-handed orthogonal coordinate system by pointing down to the center of the Earth and perpendicular to the reference ellipsoid. This frame is referred to as NED, since its axes are aligned with the north, east, and down directions.

The transformation from the ECEF to the NED is:

$$R_e^n = \begin{bmatrix} -\sin\varphi\cos\lambda & -\sin\varphi\sin\lambda & \cos\varphi \\ -\sin\lambda & \cos\lambda & 0 \\ -\cos\varphi\cos\lambda & -\cos\varphi\sin\lambda & -\sin\varphi \end{bmatrix}$$

Where **λ** (longitude) is a positive right-hand rotation around the ECEF's z-axis and **φ** (latitude) is a positive right-hand rotation around the ECEF's y-axis after the first rotation.

If $\mathbf{P_{e,ref}}$ is the position of the origin of the local NED (the take-off location) and $\mathbf{P_e}$ is the point we want to lacate in our local NED frame:

$$P_n = R_e^n(P_e - P_{e,ref})$$

where **λ** and **φ** correspond to $\mathbf{P_{e,ref}}$.

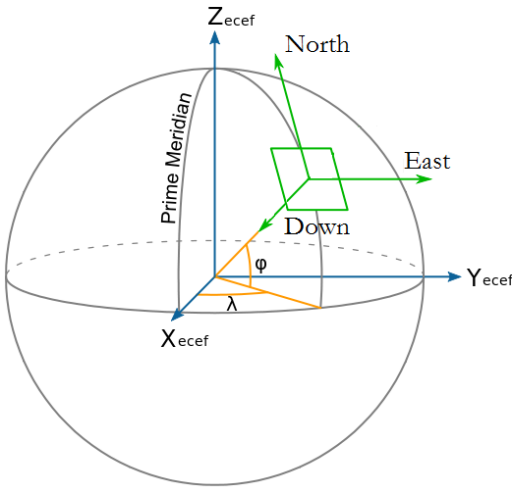


Fig. 64 The NED frame in relation to the ECEF frame

Location (ground)	north	east	down
Base camp	0	0	0
Sabangau river A	-8 820.5 m	3 088.8 m	4.1 m
Sabangau river B	-10 034.4 m	5 551.7 m	6.1 m

Table 9 NED coordinates of the mission's points of interest

2.3.14 Speed and Minimum Turn Radius

Almost always, to go from waypoint A to waypoint B involves turning. Usually, these waypoint transitions are accomplished by following orbits of different radii. Since the aircraft is flying at a certain speed, the minimum orbit radius the vehicle is able to follow depends upon the latter. The relationship between speed $\boldsymbol{v}$ and orbit radius $\boldsymbol{R}$ is given by:

$$R = \frac{v^2}{g \tan \phi}$$

According to the flight line spacing calculated the maximum radius orbit between waypoints should be around 97 m (at least less than half the distance between the waypoints involved) and, due to the fact that the average banking angle in these maneuvers is 30º, the speed at which the vehicle should fly along the area of interest on the photogrametry mission should be 111 km/h.

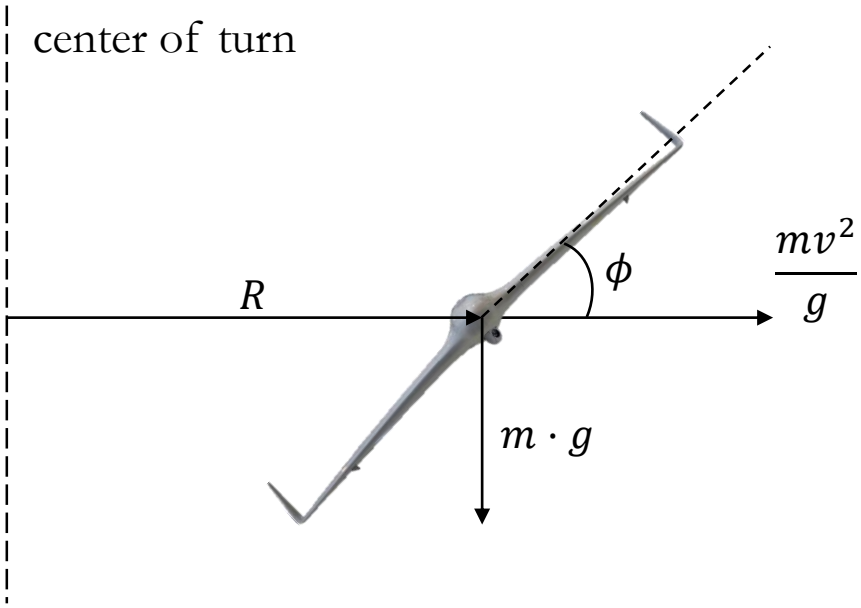


Fig. 65 Turn radius example

2.4 Simulation Results and Telemetry

The graphs obtained from the simulation are shown below. These include the route followed by the aircraft and telemetry throughout the mission.

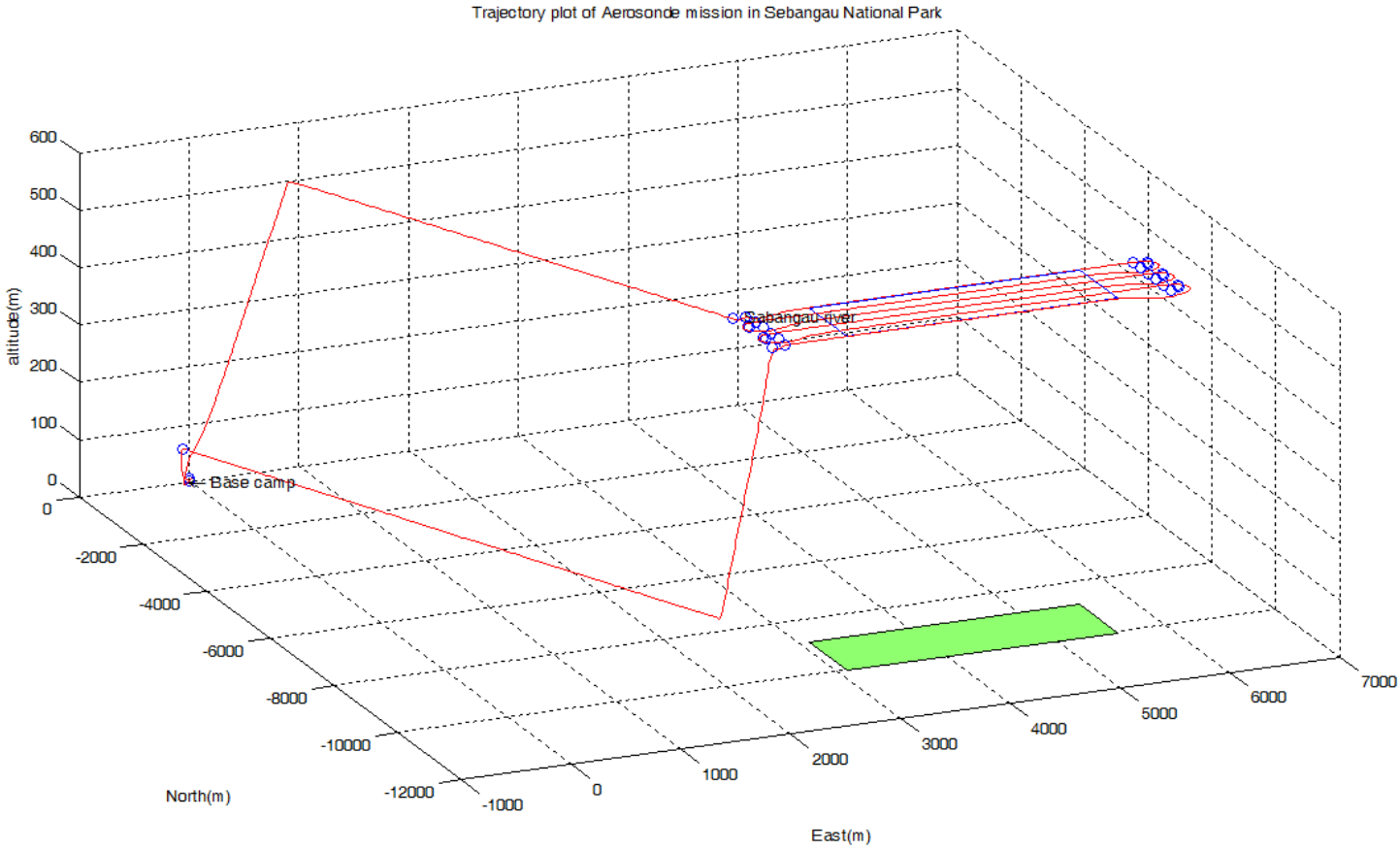


Fig. 66 Trajectory plot

Above it can be seen the trajectory the aircraft follows from the time it takes off at the base camp to the time it lands, flying above the area of interest (marked in green on the ground). Below you can see a zenithal view of the previous trajectory, where you can also see how the aircraft flies over the area of interest in straight and equally separated lines. You can also see how the vehicle initially takes off at 210 degrees to the north but quickly corrects its course to go to the first waypoint. The same thing happens after landing, since the clearing in the forest where the base camp is located only allows a runway with this orientation.

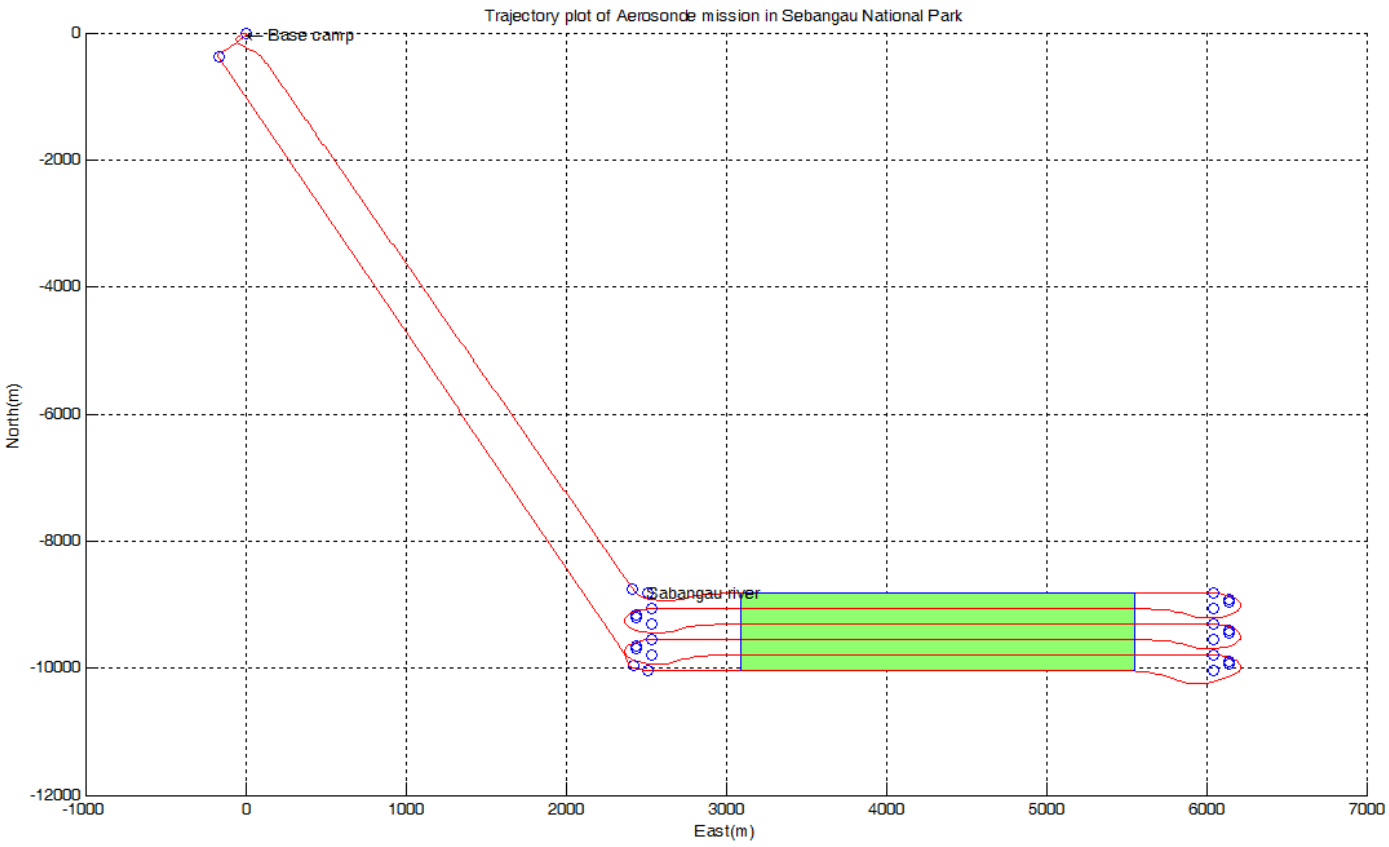


Fig. 67 Zenith view of the trajectory plot

2.4.1 Altitude

Telemetry allows for a more detailed understanding of how the aircraft behaved during the mission.

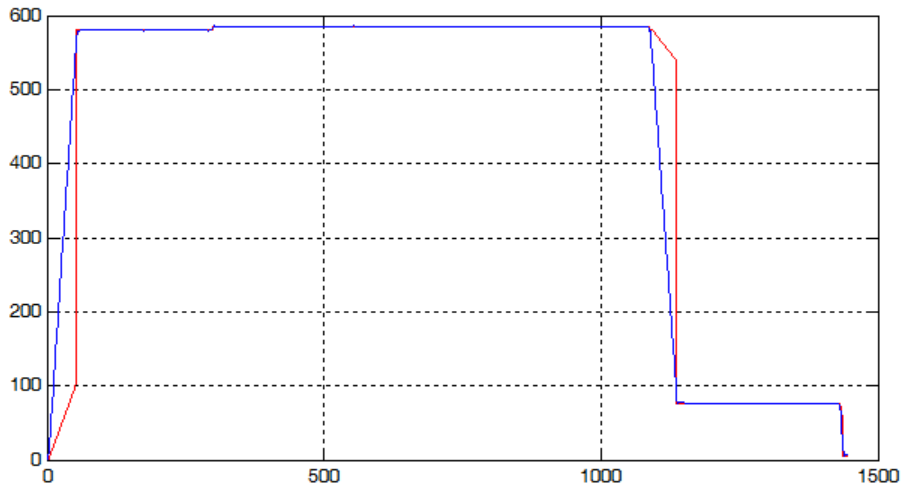


Fig. 68 Commanded altitude (red) and altitude (blue) during the mission in meters

Although the path manager establishes straight lines as pathways between waypoints, the flight controller gives priority to taking off at a certain angle of attack, hence the discrepancy between the commanded altitude and that actually accepted by the aircraft. Finally, in order to be able to use the vision-guided navigation system for net landing, the aircraft performs an approach manoeuvre until it reaches a flight below 100 metres, which will allow it to distinguish the net on which it will land.

2.4.2 Pitch

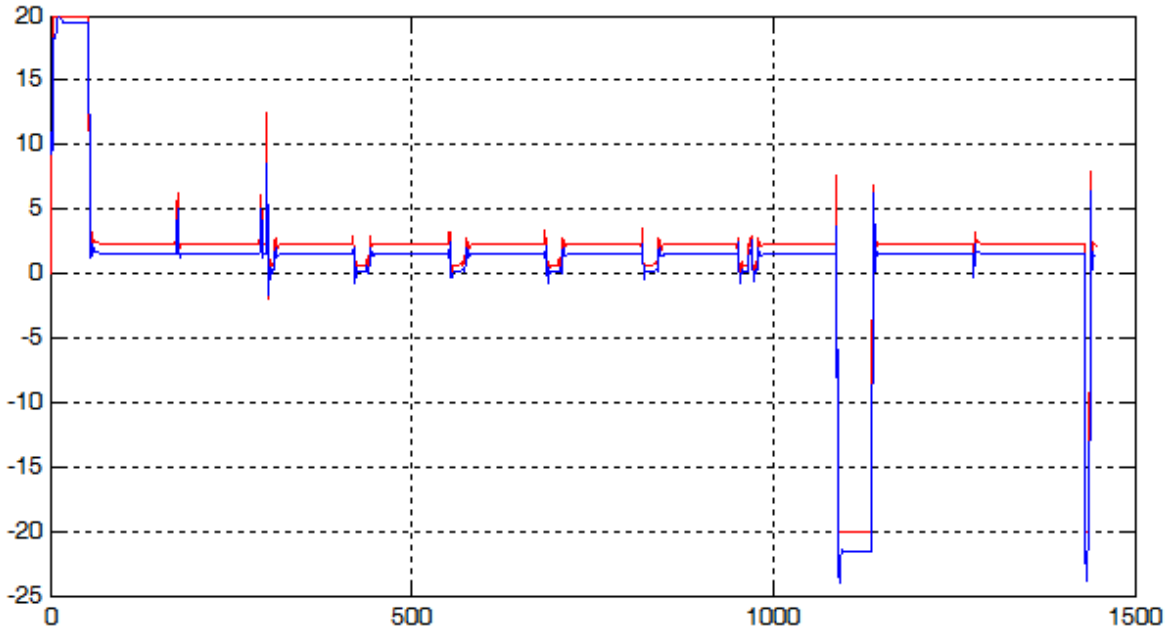


Fig. 69 Commanded pitch (red) and pitch (blue) in degrees

As mentioned in the development of the autopilot, the pitch will never reach the required value, hence the discrepancy between them commanded pitch and the actual pitch. The large pitch changes seen occur during landing, takeoff or approach manoeuvres while the small ones are meant to maintain altitude between waypoints.

2.4.3 Angle of Attack, Sideslip Angle and Airspeed Nomenclature

In order to better understand the following telemetry, the nomenclature used to relate the trajectory of the aircraft to the wind is shown below.

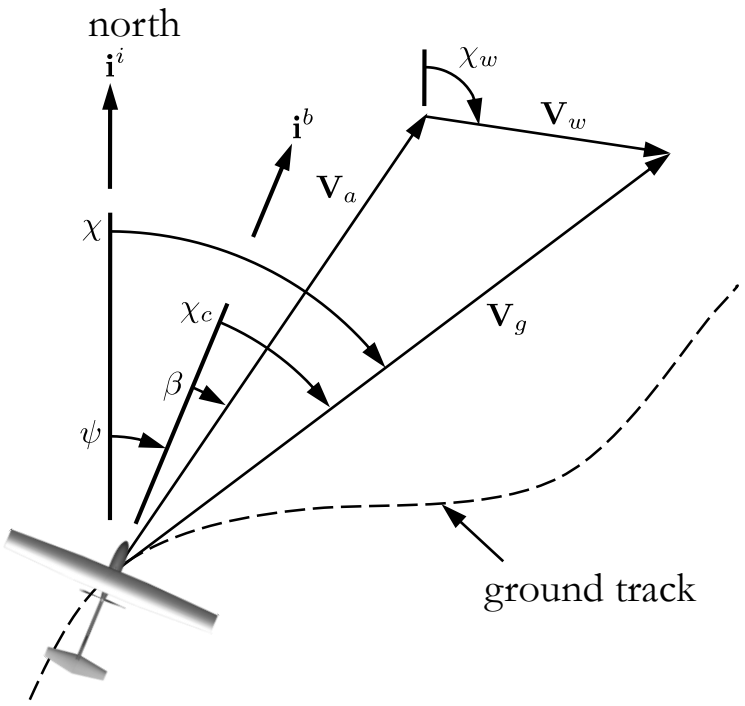


Fig. 70 North-East plane projected wind triangle nomenclature

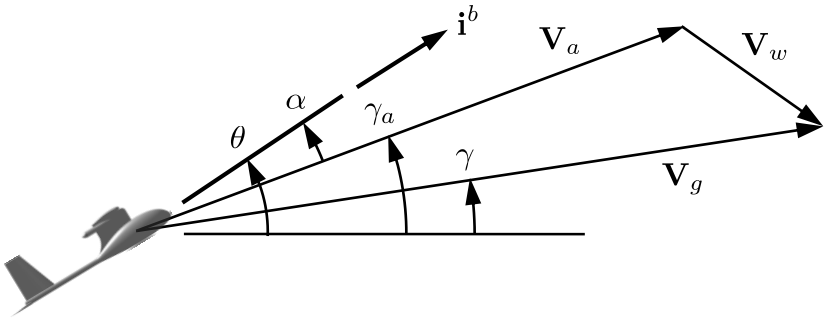


Fig. 71 Side projected wind triangle nomenclature

The direction of the ground speed vector relative to an inertial frame is specified using two angles. The flight path angle γ is defined as the angle between the horizontal plane and the ground velocity vector V_g, which is the direction the vehicle is traveling with respect to the ground. The course χ is the angle between the projection of the ground velocity vector onto the horizontal plane an true north. For level flight, the heading angle ψ is the angle between the north direction and th direction where the vehicle is pointed. The direction the vehicle is traveling with respect to the surrounding air mass is given by the airspeed vector V_a. If there is a constant ambient wind, the aircraft will need to crab into the wind in order to follow a ground track that is not aligned with the wind. The crab angle X_c is defined as the difference between the course and the heading angles. V_w is the wind speed. The relationship between the air-mass-referenced flight-path angle, the angle of attack, and the pitch angle is given by $\gamma_a = \theta - \alpha$.

2.4.4 Angle of attack

The angle of attack α is the angle between the line of the chord of an aerofoil and the relative airflow.

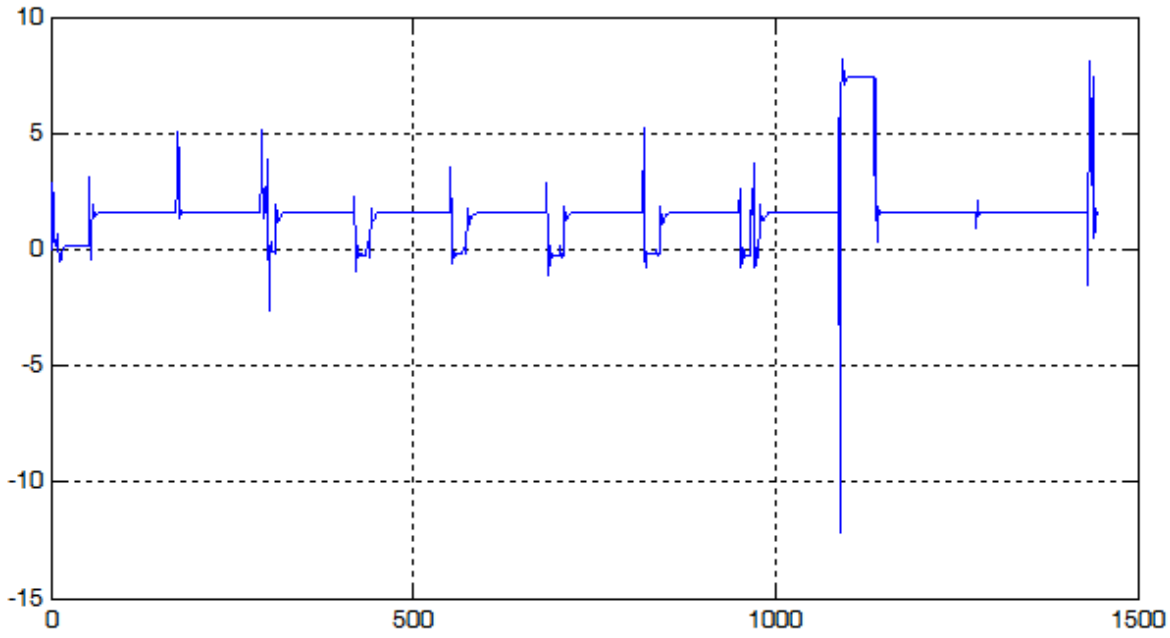


Fig. 72 Angle of attack in degrees

The angle of attack plays a very important role in lift and not paying attention to it can lead to stall conditions. Although not equal, the angle of attack will generally follow the same trend as the pitch angle. This will not be the case, however, during an abrupt and continuous descent, as the fuselage will begin to slide over the airflow.

2.4.5 Sideslip Angle

The sideslip angle is the angle between the actual direction of travel and the direction towards the vehicle is pointing.

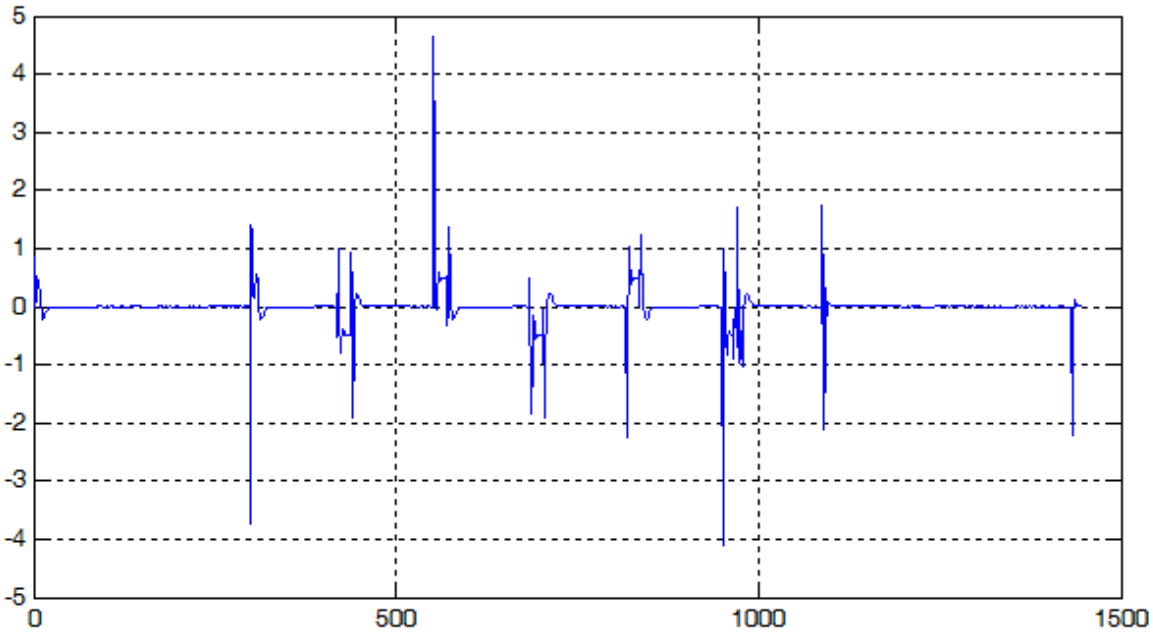


Fig. 73 Sideslip angle in degrees

As seen above, the fact that the automatic controller related to this angle only uses a proportional gain does not mean that it will not work properly. Since there is no wind in the simulation, the sideslip angle is mostly influenced by inertia. The sideslip angle will change whenever a sudden change in course or roll happens. The idea behind the sideslip controller is to assist during turning maneuvers to avoid the aircraft from skidding or slipping.

2.4.6 Airspeed

The idea of speed control is to optimise fuel consumption during flight and increase engine life, as the aircraft is subjected to lower drag forces.

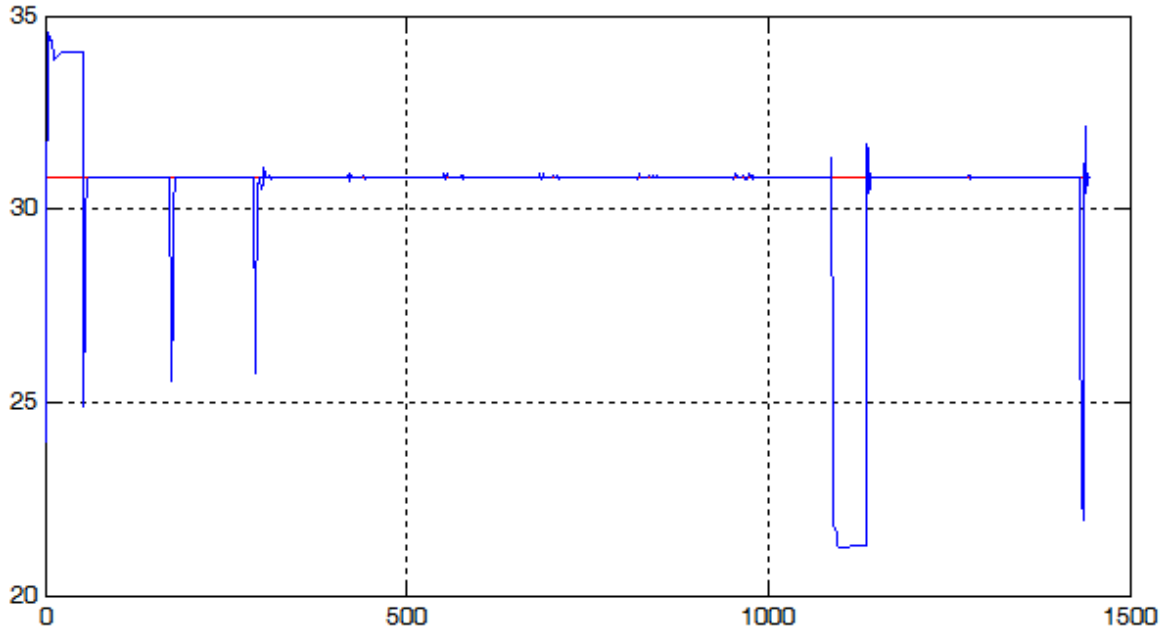


Fig. 74 Commanded airspeed (red) in m/s and actual airspeed (blue) in m/s

During takeoff, the autopilot commands the engine to run at maximum power. As the vehicle ascends, the speed decreases due to the fact that it is being used to gain altitude. During descent the engine is switched off so that the plane does not pick up too much speed and get difficult to control.

2.4.7 Roll

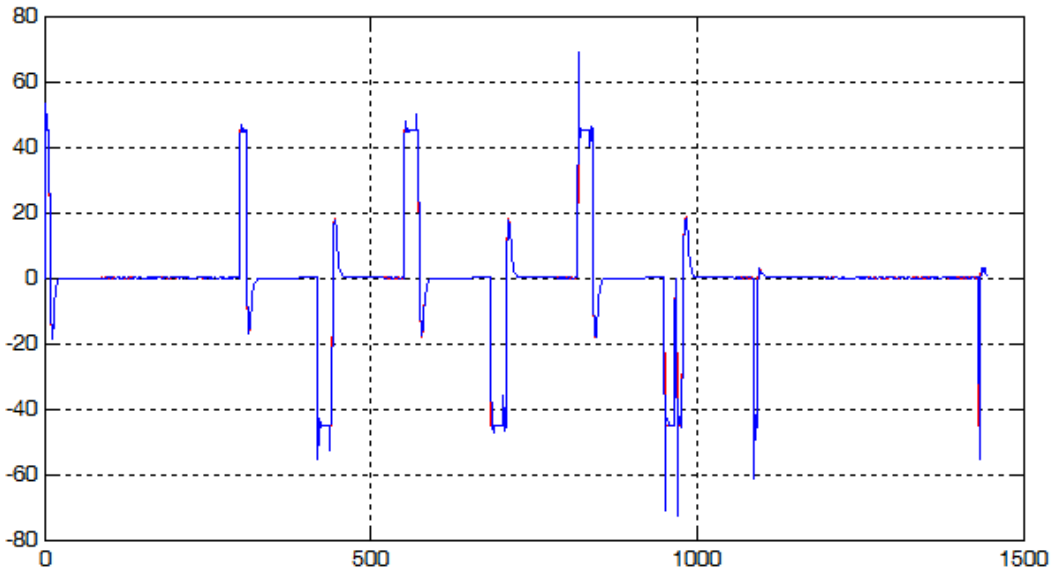


Fig. 75 Commanded roll (red) in degrees and actual roll (blue) in degrees

The reason the commanded roll looks so much like the actual one is because its controller loop has a very fast response. The first thing the flight controller requires of the aircraft is to redirect the course, so as soon as the vehicle takes off, the autopilot commands a large roll angle. Each time the aircraft passes through the transition plane of a waypoint, the autopilot changes course by commanding a change in the roll. It can be seen that, as the waypoints are located more or less in the same way, the changes in the roll angle are practically the same every time the vehicle surpasses one of these waypoints.

2.4.8 Body Frame Velocity: u, v and w

Although it may appear that the aircraft moves very little sideways or vertically in relation to its trajectory, this is not the case. Because, although the maximum and minimum values are small in relation to its speed along the i axis, they are large if we think it means that it moves sideways or vertically at that speed. Naturally, these movements take place when the vehicle changes the angle of the roll or pitch.

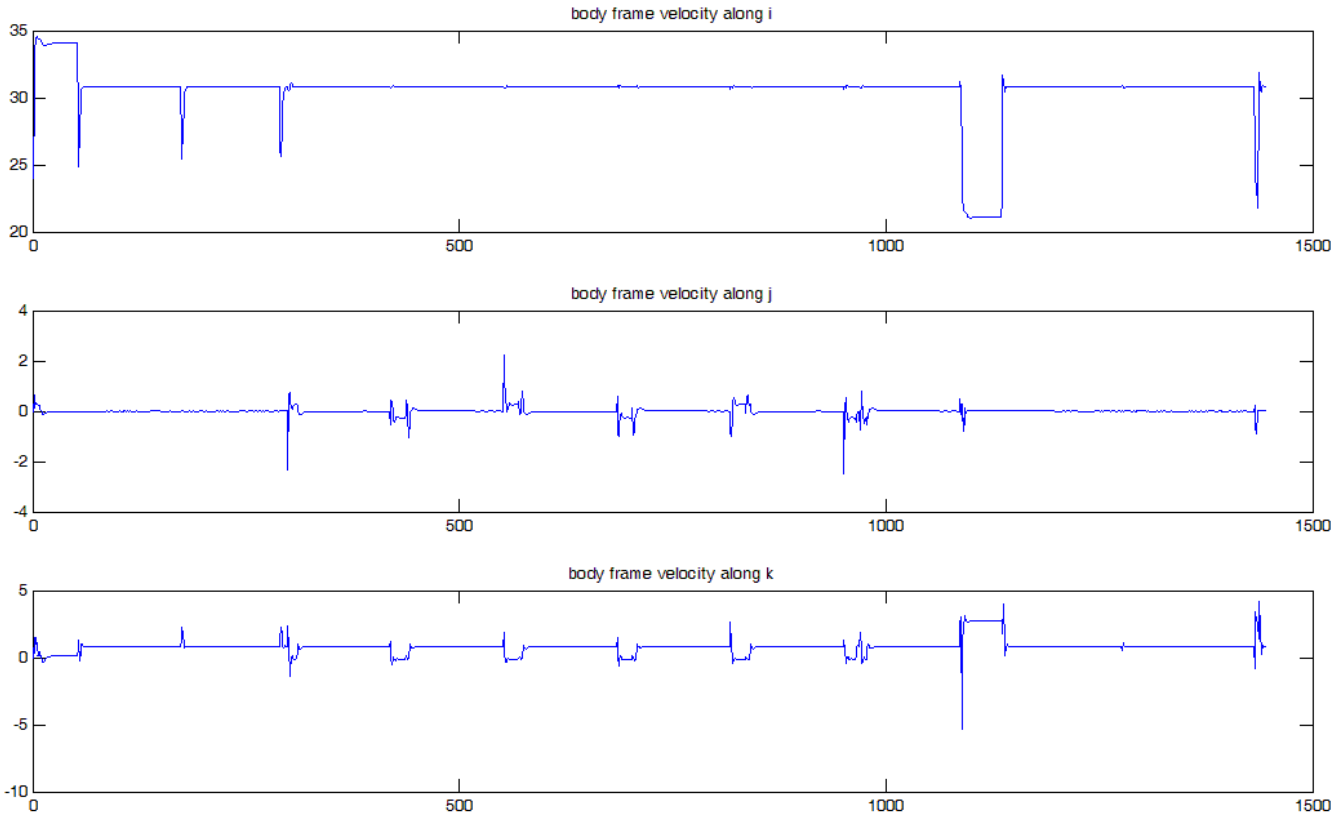


Fig. 76 From top to bottom: u, v and w in m/s

It can be seen in the first graph how the aircraft reaches its maximum speed during takeoff (when the autopilot commands full throttle) and when it reaches its minimum during descents (when it commands zero throttle). The autopilot seeks to maintain cruise speed for as long as possible in order to optimise the flight.

2.5 Conclusion

A flight controller has been developed in this paper. The flight controller is composed of a path manager, a path follower and an autopilot that uses the successive loop closure technique. Afterwards, the flight controller has been tested in a simple simulation in Matlab. In light of the results after the simulation, a future improvement could be the incorporation of an angular rate limitation. The system could also benefit greatly from the addition of dubins path features built into the path manager block.

3. ANNEXES

3.1 Annex A1. Autopilot Demonstration

Mission planning

Departure, landing and photogrammetric session starting points.

```
homePos = [0 0 0];
missionPos = [3088.8 -8820.5 585];
netHeight = 6;  netPos = homePos + [0 0 netHeight];

% The mission planner receives as input the locations from which the
% take-off and landing take place, as well as the position from which the
% photogrammetric session begins.
[ interNormal, interRoute ] = MISSION_PLANNING( homePos, missionPos, netPos );
```

Simulator

Autopilot sampling time in seconds and desired speed for the simulation in km/h.

```
T = 0.01;
speed = 111;    speed = speed/3.6;

% In turn, the simulator receives as input the positions of the transition
% planes and their normals calculated in the mission planner. The autopilot
% sampling time and the speed at which the UAV is desired to fly are also
% part of the input parameters of the simulator.
route = SIMULATOR( interNormal, interRoute, T, speed );
```

Route visualization

Finally, the path followed by the aircraft and the transition planes are displayed.

```
figure;
scatter3(interRoute(:,1),interRoute(:,2),interRoute(:,3));
hold on
plot3(route(:,1),route(:,2),-route(:,3),'Color','r');
title('Trajectory plot of Aerosonde mission in Sebangau National Park')
xlabel('East(m)')
ylabel('North(m)')
zlabel('altitude(m)')
txt_home = '\leftarrow Base camp';
txt_riverA = 'Sabangau river';
text( interRoute(1,1), interRoute(1,2), interRoute(1,3), txt_home )
text( interRoute(3,1), interRoute(3,2), interRoute(3,3), txt_riverA )
set(gcf,'color','w')
```

```
% Surveillance area visualization
x = [3088.8 5551.7 5551.7 3088.8 3088.8];
y = [-8820.5 -8820.5 -10034.4 -10034.4 -8820.5];
z = [585 585 585 585 585];
hold on
plot3( x, y, z)
patch( x, y, z)
```

3.2 Annex A1.1. Mission planning

```
function [ interNormal, interRoute ] = MISSION_PLANNING( homePos, photo_iniPos, netPos )
```

The mission planner takes into account aircraft capabilities and mission requirements and generates the waypoints that the uav must pass through to complete the photogrammetric survey.

INPUT:

- **homePos** is the location where the vehicle departures
- **photo_iniPos** is the location where the vehicle starts the photogrametric survey
- **netPos** is the location where the vehicle is recovered

OUTPUT:

- **interNormal** is an array containing the path's directions
- **interRoute** is every intermediate place on the route used to switch between straight line following mode and orbit following mode

Path planner

Spacing between flight lines, number of flight paths, path length and path's main direction

```
P_FLS = 243;
P_NFP = 5;
P_PL = 2400;
P_dir = [0 -1 0];

% The road planner generates an orderly list of waypoints taking into
% account only the mission requirements.
waypoints = path_planner( homePos, photo_iniPos, P_dir, P_FLS, P_NFP, P_PL, 1 );
```

Path manager

Minimum turning radius of the vehicle for the desired speed

```
minRadius = 97;

% Taking into account the aircraft's capabilities, the path manager
% generates a list of transition planes out of the waypoint list generated
% by the path planner.
[interNormal, interRoute] = path_manager( waypoints, minRadius, netPos );

end
```

3.3 Annex A1.1.1. Path planner

```
function waypoints = path_planner( home, W2, dir, FLS, NFP, PL, widen )
```

Taking into account the mission objectives, the path planner generates a list of waypoints.

INPUT:

- **home** is the departure and landing point
- **W2** is the initial waypoint (not departure point)
- **dir** is the direction over wich we want to construct the path
- **FLS** is the flight line spacing
- **NFP** is the number of flight paths
- **PL** is the path length
- **widen** is used to extend the surveyed area

OUTPUT:

- **waypoints** is the generated list of waypoints

Variable allocation and parameters

```
waypoints = zeros(NFP*2,3);
dirP = [-dir(2) dir(1) dir(3)];
L = 650;    landing_dist = 380; landing_height = 75;
```

Waypoint list generation

```
for i = 0:NFP
```

Normal to direction of flight

```
    waypoints(2*i+2,:) = W2 + FLS*dir*i - widen*L*dirP;
```

Direction of flight

```
    waypoints(2*i+3,:) = (W2 + PL*dirP) + FLS*dir*i + widen*L*dirP;
```

Changing waypoint order to achieve zig-zag motion

```
    if mod(2*i+2,4) == 0
        aux = waypoints(2*i+2,:);
        waypoints(2*i+2,:) = waypoints(2*i+3,:);
        waypoints(2*i+3,:) = aux;
    end

end
```

Including departure and landing approximation waypoints

```
waypoints(1,:) = home;
waypoints(end+1,:) = [-landing_dist*sin(210) landing_dist*cos(210) landing_height];
waypoints(end+2,:) = home;
```

3.4 Annex A1.1.2. Path manager

```
function [interNormal, interRoute] = path_manager( wp, minRadius, netPos )
```

The path manager generates a list of transition planes out of a waypoints list.

INPUT:

- **wp** contains each of the locations the vehicle is intended to visit.
- **minRadius** is the minimum orbit radius the vehicle can follow.
- **netPos** is the location where the vehicle is recovered.

OUTPUT:

- **interNormal** is an array containing the path's directions.
- **interRoute** is every intermediate place on the route used to switch between straight line following mode and orbit following mode.

Variable allocation

- **nInter** Number of intersections between straight-line paths. Array with the
- **cuspAngle** vector with the angles between straight-line paths
- **nTrans** number of transitions (straight-line to/from orbit)
- **interNormal** vector with the transition plane's normals
- **interRoute** vector with the transition's positions

```
nInter = size(wp,1)-2;
cuspAngle = zeros(nInter,1);
nTrans = nInter*2;
interNormal = zeros(nTrans,3);
interRoute = zeros(nTrans+2,3);
```

Transition planes calculation.

```
for i = 1:nInter-1
    interNormal((i-1)*2 +1,:) = (wp(i+1,:) - wp(i,:))/sqrt((wp(i+1,1) - wp(i,1))^2 + (wp(i+1,2)
- wp(i,2))^2 + (wp(i+1,3) - wp(i,3))^2);
    interNormal((i-1)*2 +2,:) = (wp(i+2,:) - wp(i+1,:))/sqrt((wp(i+2,1) - wp(i+1,1))^2 +
(wp(i+2,2) - wp(i+1,2))^2 + (wp(i+2,3) - wp(i+1,3))^2);
    cuspAngle(i) = acos(dot(-interNormal((i-1)*2 +1,:),interNormal((i-1)*2 +2,:)));
    interRoute((i-1)*2 +2,:) = wp(i+1,:) - (minRadius/tan(cuspAngle(i)/2))*interNormal((i-1)*2
+1,:);
    interRoute((i-1)*2 +3,:) = wp(i+1,:) + (minRadius/tan(cuspAngle(i)/2))*interNormal((i-1)*2
+2,:);
end
```

Including departure and landing positions

```
interRoute(1,:) = wp(1,:);    interRoute(end-3,:) = [];       interRoute(end-2,:) = [];
interRoute(end-1,:) = netPos;   interRoute(end,:) = netPos;

end
```

3.5 Annex A1.2. Simulator

```
function route = SIMULATOR( interNormal, interRoute, T, Va_c )
```

The simulator reproduces the behaviour of the aircraft from which the model is provided and returns the path followed by it.

INPUT:

- **interNormal** is an array containing the path's directions.
- **interRoute** is every intermediate place on the route used to switch between straight line following mode and orbit following mode.
- **T** is the base sampling time.
- **Va_c** is the desired speed for the aircraft.

OUTPUT:

- **route** is each of the points the vehicle has flown through at a sampling rate of 1 second.

Load aircraft

Loading mass, control surfaces, attitude and kinematics.

```
m = 13.5;
vehicleIni;
```

Simulation loop

The uav starts flying in a straight line to the first waypoint.

```
WPi = 2;   samplingIndex = 1;
routeType = 1;
route = interRoute(1,:);

while WPi < size(interRoute,1)
```

Path following

The path following decides the height, direction and flight mode of the vehicle. When the mission is over a landing signal is activated. When ascending or descending, if the aircraft deviates from the command provided by the path following, it maintains altitude to adjust itself to it.

```
    if WPi == size(interRoute,1)-1
        land = 1;
    else
        land = 0;
    end
    [ altitude_c, course_c, routeType, next ] = path_following( interNormal, interRoute, WPi,
pos, course, routeType, land );
    if flightPhase == 3 && -pos(3) > altitude_c
        altitude_c = interRoute(WPi,3);
    elseif flightPhase == 3 && -pos(3) < altitude_c
```

```
        altitude_c = interRoute(WPi,3);
    end
```

Autopilot

The autopilot provides the actuations for the control surfaces to meet the demands of the path planner.

```
    flightPhase = get_flightPhase( flightPhase, -pos(3), interRoute(WPi,3) );
    ctrl = autopilot( course_c, course, altitude_c, -pos(3), vehicAtt(1), vehicAtt(2), Va_c,
relwind(1), relwind(3), flightPhase, bodyRate(1), bodyRate(2), T, startFlags );    startFlags =
0;
```

State observer

The state observer provides an estimate of the aircraft's state, from kinematics and dynamics equations.

```
    [ pos, vehicAtt, vehicRate, bodyVel, bodyRate, inerVel, course, relwind ] = get_state( pos,
inerVel, vehicAtt, vehicRate, bodyVel, bodyRate, ctrl, wind, T, m );
```

Vehicle position sampling every second

```
    if mod( samplingIndex, 1/T ) == 0
        route(end+1,:) = pos;
    end
    samplingIndex = samplingIndex+1;
    WPi = WPi + next;

end

end
```

3.6 Annex A1.2.1. Path following

```
function [ altitude_c, course_c, routeType, next ] = path_following( interNormal, interRoute,
wp_i, pos, course, routeType, land )
```

Calculates the altitude and course to follow knowing the control parameters and the type of path being currently followed.

INPUT:

- **interNormal** contains the path's directions
- **interRoute** contains every intermediate place on the route used to change flight mode
- **wp_i** is the next waypoint index
- **pos** is the current position
- **course** is the current course
- **routeType** is the current route type being followed
- **land** is a signal activated when landing is commanded

OUTPUT:

- **altitude_c** is the desired altitude for the vehicle
- **course_c** is the desired course for the vehicle
- **routeType** is the route type to be followed (straight & orbit)
- **next** is a signal to increase waypoint index

Control parameters

```
k_path = 0.01;
X_inf = 75;     X_inf = X_inf*pi/180;
k_orbit = 1;
minRadius = 97;
```

Definitions

```
straight = 1;
orbit = 2;
```

State machine

Follow a straight line between the past and the current waypoint

```
if routeType == straight || land

    [ altitude_c, course_c ] = followStraightLine(interRoute(wp_i-1,:), interNormal(wp_i-1,:),
pos, course, X_inf, k_path);
    if dot(pos-interRoute(wp_i,:),interNormal(wp_i-1,:)) >= 0
        next = 1;
        routeType = orbit;
    else
        next = 0;
    end
```

Follow an orbit to transition between waypoints

```
elseif routeType == orbit  && ~land
    lambda = get_lambda( interRoute(wp_i-1,:), interRoute(wp_i,:), interRoute(wp_i+1,:) );
    orbitCenter = get_orbitCenter( interRoute(wp_i-1,:), interRoute(wp_i,:),
interRoute(wp_i+1,:), interRoute(wp_i+2,:));
    [ altitude_c, course_c ] = followOrbit(orbitCenter, minRadius, lambda, pos, course,
k_orbit);
    if dot(pos-interRoute(wp_i,:),interNormal(wp_i-1,:)) >= 0
        next = 1;
        routeType = straight;
    else
        next = 0;
    end
end

end
```

3.7 Annex A1.2.1.1. Follow straight line

```
function [ h_c, X_c ] = followStraightLine(r, q, p, X, X_inf, k_path)
```

followStraightLine calculates the course and altitude needed to follow the specified straight path taking into account the associated coefficients.

INPUT:

- **r** (NED) is the straight-line-path's starting point.
- **q** is the straight-line-path's direction.
- **p** (NED) is the vehicle's current position.
- **X** is the vehicle's current course.
- **X_inf** is a control gain. A big angle gives smooth transitions.
- **k_path** is a contorl gain. A small value gives smooth transitions.

OUTPUT:

- **h_c** altitude command.
- **X_c** course command.

Desired altitude computation

```
qk = cross(q,[0 0 1]);
n = qk/norm(qk);
ep = p - r;
s = ep - dot(ep,n)*n;
h_c = r(3) + sqrt(s(1)^2+s(2)^2) * (q(3)/(q(1)^2+q(2)^2));
```

Desired course computation

```
Xq = atan2(q(2),q(1));
while (Xq-X) < -pi
    Xq = Xq + 2*pi;
end
while (Xq-X) > pi
    Xq = Xq -2*pi;
end
epy = -sin(Xq)*(p(1)-r(1)) + cos(Xq)*(p(2)-r(2));

X_c = Xq - X_inf*2*atan(k_path*epy)/pi;

end
```

3.8 Annex A1.2.1.2. Get lambda

```
function lambda = get_lambda( w1, w2, w3 )
```

Determines the direction of rotation the vehicle must follow.

INPUT:

- **w1** is the previous waypoint
- **w2** is the current waypoint
- **w3** is the next waypoints

OUTPUT:

- **lambda** 1: clockwise orbit -1: anticlockwise orbit

```
X = cross( w2-w1,w3-w1 );

if X(3) > 0
    lambda = 1;
elseif X(3) < 0
    lambda = -1;
else
    lambda = 0;
end
```

3.9 Annex A1.2.1.3. Get orbit centre

```
function orbitCentre = get_orbitCenter( w1, w2, w3, w4 )
```

Calculates the position of the centre of an orbit around a waypoint given its four intersection planes.

INPUT:

- **w1** = intersection plane(i-1)
- **w2** = intersection plane(i)
- **w3** = intersection plane(i+1)
- **w4** = intersection plane(i+2)

OUTPUT:

- **orbitCenter** is the position of the orbit's centre

```
r1 = [w2(2)-w1(2), -w2(1)+w1(1), 0];
r2 = [w3(2)-w4(2), -w3(1)+w4(1), 0];

if ~norm(cross(r1,r2))
    orbitCentre = (w2+w3)/2;
else
    lambda = [r1(1) -r2(1); r1(2) -r2(2)]'*[w3(1)-w2(1);w3(2)-w2(2)];
    orbitCentre(1) = r1(1)*lambda(1) + w2(1);
    orbitCentre(2) = r1(2)*lambda(2) + w2(2);
    orbitCentre(3) = w1(3);
end

end
```

3.10 Annex A1.2.1.4. Follow orbit

```
function [ h_c, X_c ] = followOrbit(c, minRadius, lambda, p, X, k_orbit)
```

followOrbit calculates the course and altitude needed to follow the specified orbit taking into account the associated coefficients.

INPUT:

- **c** (NED) is the orbit's center.
- **minRadius** is the minimum radius the vehicle can follow at the current speed.
- **lambda**: 1.Clockwise orbit -1.Counterclockwise orbit
- **p** (NED) is the vehicle's current position.
- **X** is the vehicle's current course.
- **k_orbit** is a control gain. Small value gives smooth transitions.

OUTPUT:

- **h_c** altitude command.
- **X_c** course command.

Desired altitude computation

```
h_c = c(3);
```

Desired course computation

```
d = sqrt((p(1)-c(1))^2+(p(2)-c(2))^2);
phi = atan2(p(2)-c(2),p(1)-c(1));
while (phi-X) < -pi
    phi = phi + 2*pi;
end
while (phi-X) > pi
    phi = phi - 2*pi;
end

X_c = phi + lambda*(pi/2 + atan2(k_orbit*(d-minRadius),minRadius));

end
```

3.11 Annex A1.2.2. Get flight phase

```
function newPhase = get_flightPhase( phase, h, h_d )
```

INPUT:

- **phase** is the current flight phase.
- **h** is the aircraft's altitude.
- **h_d** is the desired altitude.

OUTPUT:

- **newPhase** 1.Take-off 2.Climb 3.Altitude hold 4.Descend

```
takeoff = 1;    climb = 2;  altHold = 3;    descend = 4;
```

Control parameters

```
h_hold = 20;    h_takeoff = 10;
```

State machine

```
switch phase
    case takeoff
        if h >= h_takeoff
            newPhase = climb;
        else
            newPhase = takeoff;
        end
    case climb
        if h >= h_d - h_hold
            newPhase = altHold;
        elseif h < h_takeoff*0.9
            newPhase = takeoff;
        else
            newPhase = climb;
        end
    case altHold
        if h >= h_d + h_hold
            newPhase = descend;
        elseif h < h_d - h_hold
            newPhase = climb;
        else
            newPhase = altHold;
        end
    case descend
        if h < h_d + h_hold
            newPhase = altHold;
        else
            newPhase = descend;
        end
end

end
```

3.12 Annex A1.2.3. Autopilot

```
function ctrl = autopilot( X_c, X, h_c, h, roll, pitch, Va_c, Va, beta, flightPhase, p, q, Ts,
start_flags )
```

The autopilot, which is divided into a longitudinal and a lateral part, determines the positions of the control surfaces and the speed of the engine to reach the desired speed, height and course. The autopilot is implemented via PIDs ordered in a cascade fashion. A state machine is used to implement the longitudinal control.

INPUT:

- **X_c** is the desired course
- **X** is the current course
- **h_c** is the desired altitude
- **h** is the current altitude
- **roll** is the current roll angle
- **pitch** is the current pitch angle
- **Va_c** is the desired speed with respect to the surrounding air mass
- **Va** is the current speed with respect to the surrounding air mass
- **beta** is the sideslip angle
- **flightPhase** is the phase of flight: [take-off, climb, altitude hold, descend]
- **p** is the roll rate
- **q** is the pitch rate
- **Ts** is the autopilot sampling time
- **start_flags** is a flag to reset the controller for the first time

OUTPUT:

- **ctrl** are de actuations: ailerons, elevator, rudder, throttle

Definitions

```
takeoff = 1;
climb = 2;
altHold = 3;
descend = 4;
```

Control parameters and physical limitations

The pitch hold loop has no integrator because having one would severely limit the bandwidth of the inner loop. The sideslip hold loop does not have an integrator because, if it does not reach the desired value in a short time (e.g. in a straight line with cross wind), the integrator reaches a very high value and becomes unstable. In order to incorporate an integrator, a timeout condition should be included in the controller reset.

```
limit_ailerons = 45;    limit_ailerons = limit_ailerons*pi/180;
limit_elevator = 45;    limit_elevator = limit_elevator*pi/180;
limit_rudder = 30;      limit_rudder = limit_rudder*pi/180;
limit_pitch = 20;       limit_pitch = limit_pitch*pi/180;
limit_roll = 45;        limit_roll = limit_roll*pi/180;
```

```
dt_max = 0.5;
pitch_takeoff = 11;     pitch_takeoff = pitch_takeoff*pi/180;

kp_roll = 0.9;          ki_roll = 2.7;      kd_roll = 0.01;
kp_x = 3.9;             ki_x = 0.07;
kp_pitch = -5.1;        kd_pitch = -1.2;
kp_h = 0.047;           ki_h = 0.009;
kp_B = 2;
kp_v2 = -0.12;          ki_v2 = -0.1;
kp_v = 0.036;           ki_v = 0.0274;

ctrl = [0 0 0 0];
```

Flag management

Flags are used to reset each of the PID structures that govern the actuators.

```
persistent cPrev_X;
persistent cPrev_h;
persistent cPrev_Va;
persistent cPrev_roll;

if start_flags
    flag_roll = 1;  flag_pitchV = 1;    flag_pitchH = 1;    flag_ailerons = 1;  flag_throttle =
1;
    cPrev_X = X_c;  cPrev_h = h_c;      cPrev_Va = Va_c;
else
    if cPrev_X ~= X_c || X == X_c
        flag_roll = 1;
    else
        flag_roll = 0;
    end
    if cPrev_h ~= h_c || h == h_c
        flag_pitchH = 1;
    else
        flag_pitchH = 0;
    end
    if cPrev_Va ~= Va_c || Va == Va_c
        flag_pitchV = 1;
        flag_throttle = 1;
    else
        flag_pitchV = 0;
        flag_throttle = 0;
    end
    cPrev_X = X_c;
    cPrev_h = h_c;
    cPrev_Va = Va_c;
end
```

Lateral autopilot

Includes ailerons and rudder control

```
roll_c = ctrl_roll( X_c, X, flag_roll, kp_x, ki_x, limit_roll, Ts );
if start_flags
```

```
    cPrev_roll = roll_c;
else
    if cPrev_roll ~= roll_c || roll == roll_c
        flag_ailerons = 1;
    else
        flag_ailerons = 0;
    end
    cPrev_roll = roll_c;
end
ctrl(1) = ctrl_ailerons( roll_c, roll, flag_ailerons, kp_roll, ki_roll, kd_roll, limit_ailerons,
Ts, p );
ctrl(3) = ctrl_rudder( 0, beta, kp_B, limit_rudder );
```

Longitudinal

Includes elevator and throttle control. The first three calls are used to reset persistent variables.

```
switch flightPhase
    case takeoff
        pitch_c = ctrl_pitch( Va_c, Va, flag_pitchV, kp_v2, ki_v2, limit_pitch, Ts );
        pitch_c = ctrl_pitch( h_c, h, flag_pitchH, kp_h, ki_h, limit_pitch, Ts );
        ctrl(4) = ctrl_throttle(Va_c, Va, flag_throttle, kp_v, ki_v, dt_max, Ts);

        pitch_c = pitch_takeoff;
        ctrl(4) = dt_max;
        ctrl(2) = ctrl_elevator( pitch_c, pitch, kp_pitch, kd_pitch, limit_elevator, q );
    case climb
        pitch_c = ctrl_pitch( Va_c, Va, flag_pitchV, kp_v2, ki_v2, limit_pitch, Ts );
        ctrl(4) = dt_max;
        ctrl(2) = ctrl_elevator( pitch_c, pitch, kp_pitch, kd_pitch, limit_elevator, q );
    case altHold
        pitch_c = ctrl_pitch( h_c, h, flag_pitchH, kp_h, ki_h, limit_pitch, Ts );
        ctrl(4) = ctrl_throttle(Va_c, Va, flag_throttle, kp_v, ki_v, dt_max, Ts);
        ctrl(2) = ctrl_elevator( pitch_c, pitch, kp_pitch, kd_pitch, limit_elevator, q );
    case descend
        pitch_c = ctrl_pitch( Va_c, Va, flag_pitchV, kp_v2, ki_v2, limit_pitch, Ts );
        ctrl(4) = 0;
        ctrl(2) = ctrl_elevator( pitch_c, pitch, kp_pitch, kd_pitch, limit_elevator, q );
end

end
```

3.13Annex A1.2.3.1. Control roll

```
function rollC = ctrl_roll( y_c, y, flag, kp, ki, limit, Ts )
```

The roll control function calculates the needed roll attitude using a PID.

INPUT:

- **y_c** desired output
- **y** current output
- **flag** is used to reset the ingegrator
- **kp** is the proportional gain
- **ki** is the integral gain
- **limit** is the saturation of the signal
- **Ts** is the controller sampling time

OUTPUT:

- **rollC** is the roll command

Set integrator

```
persistent integrator;
persistent error_d1;

if flag
    integrator = 0;
    error_d1 = 0;
end

error = y_c - y;
integrator = integrator + (Ts/2)*(error + error_d1);
error_d1 = error;
```

Set roll command

```
rollC = kp*error + ki*integrator;

if rollC > limit
    rollC = limit;
elseif rollC < -limit;
    rollC = -limit;
else
end
```

Integrator anti-wind-up scheme

```
if ki ~=0
    u_unsat = kp*error + ki*integrator;
    integrator = integrator + Ts/ki*(rollC-u_unsat);
end
```

3.14 Annex A1.2.3.2. Control ailerons

```
function da = ctrl_ailerons( y_c, y, flag, kp, ki, kd, limit, Ts, p )
```

The ailerons control function calculates the needed ailerons position using a PID.

INPUT:

- **y_c** desired output
- **y** current output
- **flag** is used to reset the ingegrator
- **kp** is the proportional gain
- **ki** is the integral gain
- **kd** is the derivative gain
- **limit** is the saturation of the signal
- **Ts** is the controller sampling time
- **p** is the roll rate

OUTPUT:

- **da** is the ailerons command

Set integrator

```
persistent integrator;
persistent error_d1;

if flag
    integrator = 0;
    error_d1 = 0;
end

error = y_c - y;
integrator = integrator + (Ts/2)*(error + error_d1);
error_d1 = error;
```

Set aileron position command

```
da = kp*error + ki*integrator - kd*p;
if da > limit
    da = limit;
elseif da < -limit;
    da = -limit;
end
```

Integrator anti-wind-up scheme

```
if ki ~=0
    u_unsat = kp*error + ki*integrator - kd*p;
    integrator = integrator + Ts/ki*(da-u_unsat);
end
```

3.15Annex A1.2.3.3. Control rudder

```
function dr = ctrl_rudder( y_c, y, kp, limit )
```

The rudder control function calculates the needed rudder position using a PID.

INPUT:

- **y_c** desired output
- **y** current output
- **kp** is the proportional gain
- **limit** is the saturation of the signal

OUTPUT:

- **dr** is the rudder command

Set rudder position command

```
error = y_c - y;

dr = kp*error;

if dr > limit
    dr = limit;
elseif dr < -limit;
    dr = -limit;
else
end

end
```

3.16Annex A1.2.3.4. Control pitch

```
function pitchC = ctrl_pitch( y_c, y, flag, kp, ki, limit, Ts )
```

The pitch control function calculates the needed pitch attitude using a PID.

INPUT:

- **y_c** desired output
- **y** current output
- **flag** is used to reset the ingegrator
- **kp** is the proportional gain
- **ki** is the integral gain
- **limit** is the saturation of the signal
- **Ts** is the controller sampling time

OUTPUT:

- **pitchC** is the pitch command

Set integrator

```
persistent integrator;
persistent error_d1;

if flag
    integrator = 0;
    error_d1 = 0;
end

error = y_c - y;
integrator = integrator + (Ts/2)*(error + error_d1);
error_d1 = error;
```

Set pitch command

```
pitchC = kp*error + ki*integrator;

if pitchC > limit
    pitchC = limit;
elseif pitchC < -limit;
    pitchC = -limit;
else
end
```

Integrator anti-wind-up scheme

```
if ki ~=0
    u_unsat = kp*error + ki*integrator;
    integrator = integrator + Ts/ki*(pitchC-u_unsat);
end
```

3.17Annex A1.2.3.5. Control Throttle

```
function dt = ctrl_throttle( y_c, y, flag, kp, ki, limit, Ts )
```

The throttle control function calculates the needed throttle velocity using a PID.

INPUT:

- **y_c** desired output
- **y** current output
- **flag** is used to reset the ingegrator
- **kp** is the proportional gain
- **ki** is the integral gain
- **limit** is the saturation of the signal
- **Ts** is the controller sampling time

OUTPUT:

- **dt** is the throttle command

Set integrator

```
persistent integrator;
persistent error_d1;

if flag
    integrator = 0;
    error_d1 = 0;
end

error = y_c - y;
integrator = integrator + (Ts/2)*(error + error_d1);
error_d1 = error;
```

Set throttle speed command

```
dt = kp*error + ki*integrator;

if dt > limit
    dt = limit;
elseif dt < 0;
    dt = 0;
else
end
```

Integrator anti-wind-up scheme

```
if ki ~=0
    u_unsat = kp*error + ki*integrator;
    integrator = integrator + Ts/ki*(dt-u_unsat);
end
```

3.18Annex A1.2.3.6. Control elevator

```
function de = ctrl_elevator( y_c, y, kp, kd, limit, q )
```

The elevator control function calculates the needed elevator position using a PID.

INPUT:

- **y_c** desired output
- **y** current output
- **kp** is the proportional gain
- **kd** is the derivative gain
- **limit** is the saturation of the signal
- **q** is the pitch rate

OUTPUT:

- **de** is the elevator command

Set elevator position command

```
de = kp*(y_c-y) - kd*q;

if de > limit
    de = limit;
elseif de < -limit;
    de = -limit;
end

end
```

3.19Annex A1.2.4. Get state

```
function [ pos, vehicAtt, vehicRate, bodyVel, bodyRate, inerVel, course, relwind ] = get_state(
pos, inerVel, vehicAtt, vehicRate, bodyVel, bodyRate, ctrl, wind, T, m )
```

Estimates the state of the vehicle using kinematic and dynamics equations and calculating the forces and moments acting on the vehicle.

INPUT:

- **pos** = [posN posE posD]
- **inerVel** = [posN_dot posE_dot posD_dot]
- **vehicAtt** = [roll pitch yaw]
- **vehicRate** = [pitch_dot roll_dot yaw_dot]
- **bodyVel** = [u v w]
- **bodyRate** = [p q r]
- **ctrl** = [da de dr dt]
- **wind** = [Uw Vw Ww]
- **T** is the sampling time
- **m** is the mass of the vehicle

OUTPUT:

- **pos** = [posN posE posD]
- **vehicAtt** = [roll pitch yaw]
- **vehicRate** = [pitch_dot roll_dot yaw_dot]
- **bodyVel** = [u v w]
- **bodyRate** = [p q r]
- **inerVel** = [posN_dot posE_dot posD_dot]
- **course** is the current course
- **relwind** = [Va alfa beta]

Forces and moments

```
relwind = get_relwind( bodyVel, wind );
forces = get_forces( vehicAtt, bodyRate, ctrl, relwind );
moments = get_moments( bodyRate, ctrl, relwind );
```

Kinematics and dynamics

```
bodyRate2 = get_bodyRate2( bodyRate, moments );
bodyRate = get_bodyRate( bodyRate, bodyRate2, T );
bodyAcc = get_bodyAcc( bodyRate, bodyVel, forces, m );
bodyVel = get_bodyVel( bodyVel, bodyAcc, T );
vehicRateo = vehicRate;
vehicRate = get_vehicRate( vehicAtt, bodyRate );
vehicAtt = get_vehicAtt( vehicAtt, vehicRateo, vehicRate, T );
inerVelo = inerVel;
inerVel = get_inerVel( vehicAtt, bodyVel );
course = atan2(inerVel(2),inerVel(1));
pos = get_pos( pos, inerVelo, inerVel, T );
```

3.20 Annex A1.2.4.1. Get relative wind

```
function relwind = get_relwind( bodyVel, wind )
```

Calculates the speed with respect to the surrouding air mass, the angle of attack and the sideslip angle.

INPUT:

- **bodyVel** = [u v w]
- **wind** = [uw vw ww]

OUTPUT:

- **relwind** = [Va alfa beta]

```
relwind(1) = sqrt((bodyVel(1)-wind(1))^2+(bodyVel(2)-wind(2))^2+(bodyVel(3)-wind(3))^2);
if (bodyVel(3)-wind(3)) == 0 && (bodyVel(1)-wind(1)) == 0
    relwind(2) = 0;
else
    relwind(2) = atan((bodyVel(3)-wind(3))/(bodyVel(1)-wind(1)));
end
if (bodyVel(2)-wind(2)) == 0 && relwind(1) == 0
    relwind(3) = 0;
else
    relwind(3) = asin((bodyVel(2)-wind(2))/(relwind(1)));
end

relwind = [relwind(1) relwind(2) relwind(3)];

end
```

3.21 Annex A1.2.4.2. Get forces

```
function forces = get_forces( vehicAtt, bodyRate, ctrl, relwind )
```

Calculates the forces acting on the aircraft.

INPUT:

- **vehicAtt** = [roll pitch yaw]
- **bodyRate** = [p q r]
- **ctrl** = [da de dr dt]
- **relwind** = [Va alfa beta]

OUTPUT:

- **forces** = [fx fy fz]

Definitions

```
m = 13.5;
S = 0.55;
b = 2.8956;
c = 0.18994;
Sprop = 0.2027;
kmotor = 80;
e = 0.9;
CL0 = 0.28;
CD0 = 0.03;
CLa = 3.45;
CLq = 0;
CDq = 0;
CLde = -0.36;
CDde = 0;
Cprop = 1;
M = 50;
a0 = 0.4712;
CY0 = 0;
CYB = -0.98;
CYp = 0;
CYr = 0;
CYda = 0;
CYdr = -0.17;
AR = b^2/S;
ro = 1.2682;
g = 9.807;
```

Forces calculation

```
sigma = (1+exp(-M*(relwind(2)-a0))+exp(M*(relwind(2)+a0)))/((1+exp(-M*(relwind(2)-
a0)))*(1+exp(M*(relwind(2)+a0))));

CD_a = CD0 + (CL0 + CLa*relwind(2))^2/(pi*e*AR);
CL_a = (1-
sigma)*(CL0+CLa*relwind(2))+sigma*(2*sign(relwind(2))*cos(relwind(2))*(sin(relwind(2)))^2);
```

```
CX_a = -CD_a*cos(relwind(2))+CL_a*sin(relwind(2));
CXq_a = -CDq*cos(relwind(2))+CLq*sin(relwind(2));
CXde_a = -CDde*cos(relwind(2))+CLde*sin(relwind(2));
CZ_a = -CD_a*sin(relwind(2))-CL_a*cos(relwind(2));
CZq_a = -CDq*sin(relwind(2))-CLq*cos(relwind(2));
CZde_a = -CDde*sin(relwind(2))-CLde*cos(relwind(2));

forces = [-m*g*sin(vehicAtt(2)) m*g*cos(vehicAtt(2))*sin(vehicAtt(1))
m*g*cos(vehicAtt(2))*cos(vehicAtt(1))]...
    + 0.5*ro*relwind(1)^2*S*[CX_a+CXq_a*c*bodyRate(2)/(2*relwind(1)+0.000001)+CXde_a*ctrl(2)
CY0+CYB*relwind(3)+CYp*b*bodyRate(1)/(2*relwind(1)+0.000001)+CYr*b*bodyRate(3)/(2*relwind(1)+0.0
00001)+CYda*ctrl(1)+CYdr*ctrl(3)
CZ_a+CZq_a*c*bodyRate(2)/(2*relwind(1)+0.000001)+CZde_a*ctrl(2)]...
    + 0.5*ro*Sprop*Cprop*[(kmotor*ctrl(4))^2-relwind(1)^2 0 0];

end
```

3.22 Annex A1.2.4.3. Get moments

```
function moments = get_moments( bodyRate, ctrl, relwind )
```

Calculates the moments acting on the aircraft.

INPUT:

- **bodyRate** = [p q r]
- **ctrl** = [da de dr dt]
- **relwind** = [Va alfa beta]

OUTPUT:

- **moments** = [l m n]

Definitions

```
S = 0.55;
b = 2.8956;
c = 0.18994;
kTp = 0;
komega = 0;
Cm0 = -0.02338;
Cma = -0.38;
Cmq = -3.6;
Cmde = -0.5;
Cndr = -0.032;
Cl0 = 0;
Cn0 = 0;
ClB = -0.12;
CnB = 0.25;
Clp = -0.26;
Cnp = 0.022;
Clr = 0.14;
Cnr = -0.35;
Clda = 0.08;
Cnda = 0.06;
Cldr = 0.105;
ro = 1.2682;
```

Moments calculation

```
moments =
0.5*ro*relwind(1)^2*S*[b*(Cl0+ClB*relwind(3)+Clp*b*bodyRate(1)/(2*relwind(1)+0.000001)+Clr*b*bod
yRate(3)/(2*relwind(1)+0.000001)+Clda*ctrl(1)+Cldr*ctrl(3))
c*(Cm0+Cma*relwind(2)+Cmq*c*bodyRate(2)/(2*relwind(1)+0.000001)+Cmde*ctrl(2))
b*(Cn0+CnB*relwind(3)+Cnp*b*bodyRate(1)/(2*relwind(1)+0.000001)+Cnr*b*bodyRate(3)/(2*relwind(1)+
0.000001)+Cnda*ctrl(1)+Cndr*ctrl(3))]...
    + [-kTp*(komega*ctrl(4))^2 0 0];

end
```

3.23 Annex A1.2.4.4. Get body rate 2

```
function bodyRate2 = get_bodyRate2( bodyRate, moments )
```

Calculates the angular accelerations.

INPUT:

- **bodyRate** = [p q r]
- **moments** = [l m n]

OUTPUT:

- **bodyRate2** = [p_dot q_dot r_dot]

```
Jx = 0.8244;
Jy = 1.135;
Jz = 1.759;
Jxz = 0.1204;
gamma0 = Jx*Jz - Jxz*Jxz;
gamma = zeros(8,1);
gamma(1) = Jxz*(Jx - Jy + Jz)/gamma0;
gamma(2) = (Jz*(Jz - Jy) + Jxz*Jxz)/gamma0;
gamma(3) = Jz/gamma0;
gamma(4) = Jxz/gamma0;
gamma(5) = (Jz - Jx)/Jy;
gamma(6) = Jxz/Jy;
gamma(7) = (Jx*(Jx - Jy) + Jxz*Jxz)/gamma0;
gamma(8) = Jx/gamma0;


bodyRate2 = [gamma(1)*bodyRate(1)*bodyRate(2)-gamma(2)*bodyRate(2)*bodyRate(3),...
    gamma(5)*bodyRate(1)*bodyRate(3)-gamma(6)*(bodyRate(1)^2-bodyRate(3)^2),...
    gamma(7)*bodyRate(1)*bodyRate(2)-gamma(1)*bodyRate(2)*bodyRate(3)]...
    +[gamma(3)*moments(1)+gamma(4)*moments(3) moments(2)/Jy
gamma(4)*moments(1)+gamma(8)*moments(3)];

end
```

3.24 Annex A1.2.4.5 Get body rate

```
function bodyRate = get_bodyRate( bodyRate, bodyRate2, T )
```

Calculates the angular rates defined with respect to a reference system linked to the vehicle.

INPUT:

- **bodyRate** = [p q r]
- **bodyRate2** = [p_dot q_dot r_dot]s
- **T** is the sampling time

OUTPUT:

- **bodyRate** = [p q r]

```
bodyRate = bodyRate + bodyRate2*T;

end
```

3.25 Annex A1.2.4.6. Get body accelerations

```
function bodyAcc = get_bodyAcc( bodyRate, bodyVel, forces, m )
```

Calculates the accelerations defined with respect to a reference system linked to the vehicle.

INPUT:

- **bodyRate** = [p q r]
- **bodyVel** = [u v w]
- **forces** = [fx fy fz]
- **m** is the mass of the vehicle

OUTPUT:

- **bodyAcc** = [u_dot v_dow w_dot]

```
bodyAcc = [bodyRate(3)*bodyVel(2)-bodyRate(2)*bodyVel(3) bodyRate(1)*bodyVel(3)-
bodyRate(3)*bodyVel(1) bodyRate(2)*bodyVel(1)-bodyRate(1)*bodyVel(2)]...
    + [forces(1) forces(2) forces(3)]/m;

end
```

3.26 Annex A1.2.4.7. Get body velocities

```
function bodyVel = get_bodyVel( bodyVel, bodyAcc, T )
```

Calculates the velocity defined with respect to a reference system linked to the vehicle.

INPUT:

- **bodyVel** = [u v w]
- **bodyAcc** = [u_dot v_dow w_dot]
- **T** (sampling period)

OUTPUT:

- **bodyVel** = [u v w]

```
bodyVel = bodyVel + bodyAcc*T;

end
```

3.27 Annex A1.2.4.8. Get vehicle rates

```
function vehicRate = get_vehicRate( vehicAtt, bodyRate )
```

Calculates the angular rate of the vehicle using the kinematics equation.

INPUT:

- **vehicAtt** = [roll pitch yaw]
- **bodyRate** = [p q r]

OUTPUT:

- **vehicRate** = [pitch_dot roll_dot yaw_dot]

```
body2vehicle = [1 sin(vehicAtt(1))*tan(vehicAtt(2)) cos(vehicAtt(1))*tan(vehicAtt(2));...
    0 cos(vehicAtt(1)) -sin(vehicAtt(2));...
    0 sin(vehicAtt(1))/cos(vehicAtt(2)) cos(vehicAtt(1))/cos(vehicAtt(2))];

vehicRate = body2vehicle*bodyRate';
vehicRate = vehicRate';

end
```

3.28 Annex A1.2.4.9. Get vehicle attitude

```
function vehicAtt = get_vehicAtt( vehicAtto, vehicRateo, vehicRatef, T )
```

Calculates the attitude (-π,π) of the vehicle using the kinematics equation.

INPUT:

- **vehicAtto** is the previous attitude.
- **vehicRateo** = [pitch_dot roll_dot yaw_dot] (previous).
- **vehicRatef** = [pitch_dot roll_dot yaw_dot] (last).
- **T** is the sampling time.

OUTPUT:

- **vehicAtt** = [pitch roll yaw]

```
vehicAtt = vehicAtto + (vehicRateo + vehicRatef)*T/2;

while vehicAtt(1) > pi || vehicAtt(1) < -pi
    if vehicAtt(1) > pi
        vehicAtt(1) = vehicAtt(1) - 2*pi;
    elseif vehicAtt(1) < -pi
        vehicAtt(1) = vehicAtt(1) + 2*pi;
    end
end

while vehicAtt(2) > pi || vehicAtt(2) < -pi
    if vehicAtt(2) > pi
        vehicAtt(2) = vehicAtt(2) - 2*pi;
    elseif vehicAtt(2) < -pi
        vehicAtt(2) = vehicAtt(2) + 2*pi;
    end
end

while vehicAtt(3) > pi || vehicAtt(3) < -pi
    if vehicAtt(3) > pi
        vehicAtt(3) = vehicAtt(3) - 2*pi;
    elseif vehicAtt(3) < -pi
        vehicAtt(3) = vehicAtt(3) + 2*pi;
    end
end

end
```

3.29 Annex A1.2.4.10. Get inertial velocity

```
function inerVel = get_inerVel( vehicAtt, bodyVel )
```

Calculates the inertial speed give the attitude and speed in relation to a reference system attached to the vehicle.

INPUT:

- **vehicAtt** = [roll pitch yaw]
- **bodyVel** = [u v w]

OUTPUT:

- **inerVel** = [posN_dot posE_dot posD_dot]

```
body2inertial = [cos(vehicAtt(2))*cos(vehicAtt(3))
sin(vehicAtt(1))*sin(vehicAtt(2))*cos(vehicAtt(3))-cos(vehicAtt(1))*sin(vehicAtt(3))
cos(vehicAtt(1))*sin(vehicAtt(2))*cos(vehicAtt(3))+sin(vehicAtt(1))*sin(vehicAtt(3));...
    cos(vehicAtt(2))*sin(vehicAtt(3))
sin(vehicAtt(1))*sin(vehicAtt(2))*sin(vehicAtt(3))+cos(vehicAtt(1))*cos(vehicAtt(3))
cos(vehicAtt(1))*sin(vehicAtt(2))*sin(vehicAtt(3))-sin(vehicAtt(1))*cos(vehicAtt(3));...
    -sin(vehicAtt(2)) sin(vehicAtt(1))*cos(vehicAtt(2)) cos(vehicAtt(1))*cos(vehicAtt(2))];

inerVel = body2inertial*bodyVel';
inerVel = inerVel';

end
```

3.30 Annex A1.2.4.11. Get position

```
function pos = get_pos( poso, inerVelo, inerVelf, T )
```

Calculates the position of the vehicle using the kinematics equation.

INPUT:

- **poso** is the previous position of the vehicle.
- **inerVelo** = [posN_dot posE_dot posD_dot] (previous).
- **inerVelf** = [posN_dot posE_dot posD_dot] (last).
- **T** is the sampling time.

OUTPUT:

- pos = [posN posE posD]

```
pos = poso + (inerVelo + inerVelf)*T/2;

end
```

4. APPENDIX

4.1 Phase One's iXU-RS 1000 Aerial Camera Specifications

Fig. 77 iXU-RS 1000 with different lenses

The main reason I chose Phase One Aerial Camera Systems is because their cameras are designed exclusively for aerial photography and are widely used all over the world for studies such as this. The cameras are very suitable for use on UAVs given their light weight frame, reliability at high shutter speed, and excellent results even with vibrations and different light conditions. They also include features such us: scability to form multi-camera arrays as well as easy integration with flight management systems and GPS/IMU receivers.

As weather conditions deteriorate, the CMOS technology of the iXU-RS 1000 enables to move from ISO 50 all the way up to 6400, providing quality images all across the ISO range, so they are a great choice for projects where light conditions can't be predicted (as in forests). The 100 MP sensor, offers a small-bodied (the body is barely wider than their lens barrels), light weight (starting from 1.25 kg) medium format camera with high performance optics. The camera delivers 11 608 pixels cross-track coverage, wich is 12 percent greater than previously available medium format aerial camera systems.

Phase One cameras are easily integrated with UAVs given the comprehensive documentation, including 2D and 3D drawings that is available in the manufacturer's web. The cameras also have predefined settings for the most popular GNSS receivers and have the ability to write the IMU/GNSS data directly to the EXIF (a standard that specifies the formats for images tags used by digital cameras) of each file, creating geo-tagged images. The reliance of the RS shutter's capacity of 500 000 cycles along with the exposure time of 1/2500 s enables faster flying, and allows to execute and manage the most demanding aerial photography missions with higher operational efficiency, reliability, and in a cost effective way.

4.1.1 4-Band Solution. Four-Channel Imaging

Fig. 78 Phase One 4-band base plate and cameras

Phase One offers a fully automatic solution for capturing and processing 4-Band imagery, specifically designed for photogrammetric airborne tasks. It uses two synchronized Phase One metric calibrated cameras (RGB and NIR) mounted side by side on a specially designed base plate, an iX Controller computer and the iX Capture software. The software automatically generates distortion-free images and performs fine co-registration of pixels from the NIR to the RGB images, including processing different image sizes.

NIR wave length is very important for analyzing and detecting vegetation. In the past, capturing and analyzing images of vegetation required analog cameras using color or black and white infrared film. Using film presented challenges as film reacted differently to the infrared and visual wavelenghts and a lot of calibration was needed. Today's digital sensors offer a much easier workflow and have a real benefit, as they are sensitive to near infrared from 720 nm up to a wave lenght of approximately 1100 nm.

In digital cameras, users don't want their images recording NIR, which destroys the image, because the focus position is different for visible light and NIR and a combined visual and NIR image would appear unusable. To isolate the specific region of light that is needed to capture, an IR pass filter, which blocks visual light and only allows NIR light to pass from a specific wavelength, is used. Since NIR and visual light have different wavelengths, the focus positions are different, which requires the camera to have a different infinity focus.

The filter used to block specific wavelengths can be screwed in front of the lens or even placed in front of the sensor instead of the optical glass. When a filter is placed in front of the sensor, the camera can only be used for one purpose only. To mount the cameras side by side, a plate is needed on which both cameras can be attached in the nadir (the point directly below) position so that both footprints are almost identical. A Phase One multi-sync cable daisy chains the cameras together, enabling a synchronization speed of 100 microseconds. When used with Phase One's iX Capture software, an operator can control each camera (e.g. ISO, shutter speed and aperture), as well as view each image after it is captured.

If two cameras are mounted together, both capturing the same footprint by pointing nadir, their images can be combined in post processing to create four-channel images. One camera would be capturing visual light and the second one would have a filter attached to its lens to block visual light.

The images captured from RGB and NIR cameras can be merged in post processing software. Since our eyes can only see visual light and monitors can only display RGB, the merged image is normally displayed as a color infrared (CIR) in the following manner: red channel is replaced by the infrared, green channel is replaced by the red and blue channel is replaced by the green. These images typically display vegetation as red.

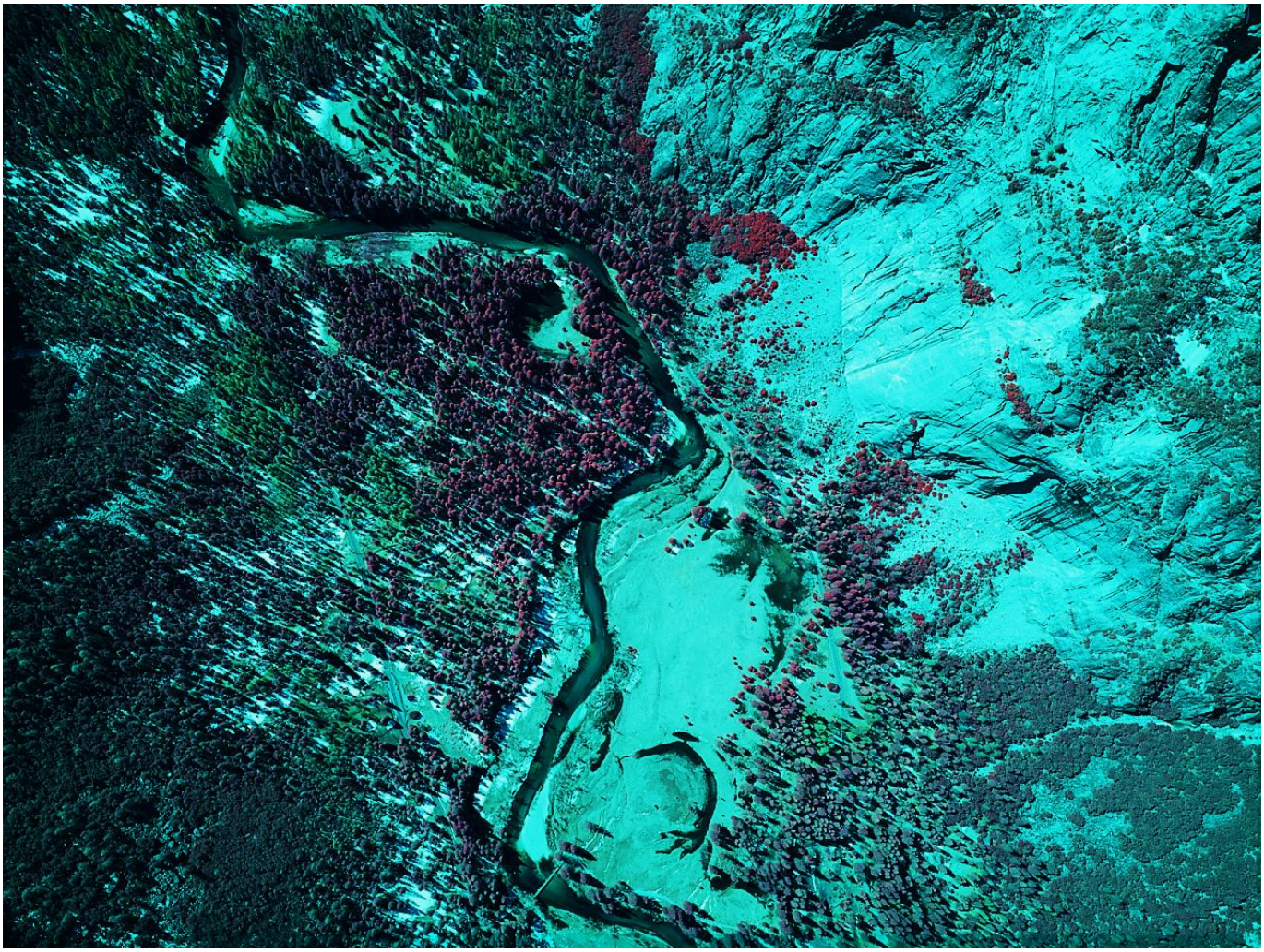


Fig. 79 CIR Aerial Photo

Four-channel imagery can be used to check when crops are healthy. Geo-referenced 4-channel images indicate where to actuate (if necessary) by flying over the field with a narrow band filter to match the bandwith with the tabulated "healthy" values.

4.1.2 iXU-RS 1000 Aerial Camera Technical Specifications

Resolution	100 MP (11608 x 8708)
Pixel Size	4.6 micron
Sensor size effective	53.4 x 40.0 mm
Light sensitivity (ISO)	50-6400
Shutter speed	Leaf shutter: up to 1/2500 second
Data storage	1 TB SSD storage

Capture rate – full resolution frame	0.6 s
Output formats	Phase One RAW, TIF, JPG, CIR and NDVI
Weight (excluding lens)	0.930 kg
Temperature	-10 ºC to 40 ºC
Humidity	15 to 80% (non-condensing)

Table 10 iXU-RS 1000 aerial camera specifications

4.1.3 Rodenstock 90 mm f/5.6 Lens Specifications

The choice of this lens comes from the fact that it enables the user to take images with very little distortion, without having to fly too close to the ground and, therefore allows to have more than enough time between shots, which in turn improves the accuracy of the system.

Aperture range	f/5.6 – f/22
LS shutter speed max.	Up to 1/2000 sec
Dimensions	97.4 x 93 x 218 mm
Weight	1150 g
Angle of view	100 MP 33.0º (Long side) 25.1º (Short side)

Table 11 Rodenstock 90 mm f/5.6 lens specifications

4.2 Textron System's Aerosonde UAV

Aerosonde is a small unmanned aerial vehicle (SUAV) designed to collect weather data, including temperature, atmospheric pressure, humidity, and wind measurements over oceans and remote areas. The aircraft was also the first of his kind to cross the Atlantic Ocean and the first unmanned vehicle to penetrate tropical cyclones.

Fig. 80 Textron System's Aerosonde UAV

Aerosonde is a widely used unmanned aircraft and hence the umpteen information (mass, geometry, propulsion and aerodynamic coefficients) available to the public. The table below is a recollection of aerodynamic coefficients and mass properties of the vehicle.

Parameter	Value	Long. coef.	Value	Lat. coef.	Value
m	13.5 kg	C_{L0}	0.28	C_{Y0}	0
J_x	0.8244 kg m^2	C_{D0}	0.03	C_{l0}	0
J_y	1.135 kg m^2	C_{m0}	-0.02338	C_{n0}	0
J_z	1.759 kg m^2	$C_{L\alpha}$	3.45	$C_{Y\beta}$	-0.98
J_{xz}	0.1204 kg m^2	$C_{D\alpha}$	0.3	$C_{l\beta}$	-0.12
S	0.55 m^2	$C_{m\alpha}$	-0.38	$C_{n\beta}$	0.25
b	2.8956 m	C_{Lq}	0	C_{Yp}	0
c	0.18994 m	C_{Dq}	0	C_{lp}	-0.26
S_{prop}	0.2027 m^2	C_{mq}	-3.6	C_{np}	0.022
k_{motor}	80	$C_{L\delta e}$	-0.36	C_{Yr}	0
k_{Tp}	0	$C_{D\delta e}$	0	C_{lr}	0.14

k_Ω	0	$C_{m\delta e}$	-0.5	C_{nr}	-0.35
e	0.9	C_{prop}	1.0	$C_{Y\delta a}$	0
		M	50	$C_{l\delta a}$	0.08
		α_0	0.4712	$C_{n\delta a}$	0.06
		ϵ	0.1592	$C_{Y\delta r}$	-0.17
		C_{Dp}	0.0437	$C_{l\delta r}$	0.105
		$C_{n\delta r}$	-0.032		

Table 12 Aerodynamic coefficients for the Aerosonde UAV

4.2.1 Aerosonde Specifications

Wing span	2.9 m
Engine	20 or 26 cc, 750 up to 1300 watt
MTOW	17.7 or 24.9 kg depending on engine type
Navigation	GPS
Take-off	Catapult or car roof rack
Landing	On net or belly
Communications	Radio or satellite
Range	>3000 km
Endurance	>30 h
Service altitude range	100 – 4000 m
Maximum altitude range	20 – 4572 m
Cruise speed	111 km/h
Max. speed	148 km/h
Min. speed (stall)	57 km/h

Table 13 Aerosonde UAV specifications

4.2.2 Assisted Take-off

For an aircraft to leave the ground, the following condition must be met:

$$F_{lift} > mg$$

To get an idea of the required take-off speed, we can use:

$$F_{lift} = \frac{1}{2}\rho V_a^2 S\left[C_L(\alpha) + C_{L_q}\frac{c}{2V_a}q + C_{L_{\delta e}}\delta_e\right]$$

where $C_L(\alpha)$ can be approximated by

$$C_L(\alpha) = C_{L_0} + C_{L_\alpha}\alpha$$

For the aerosonde, C_{L_q} is 0 so the equation gets simpler.

As the Aerosonde's belly is very short in the longitudinal direction, we cannot add to it a triangle of landing wheels that increase its inclination too much (in stationary position). Assuming therefore an angle of attack of 2° during take-off, the necessary speed (without wind) to raise from the ground will be 111 km/h which is almost the cruise speed, and thus a very far-fetched velocity. If a catapult is used however, with an inclination of 11º, the speed needed decreases to 72 km/h wich is easily realizable.

Fig. 81 UAV Factory 6 kJ portable pneumatic catapult

The catapult has the following specifications:

Rail length	4 m
Maximum UAV weight	35 kg
Launching height	1.1 m
Launching speed for 13.5 kg	24 m/s
Inclination	11º

Table 14 Factory 6 KJ catapult specs.

REFERENCES

[1] R. W. Beard and T. W. McLain, Small unmanned aircraft: theory and practice, Princeton University Press, 2012.

[2] T. M. Adami and J. Jim Zhu, "6DOF Flight Control of Fixed-Wing Aircraft by Trajectory Linearization," *2011 American Conference,* 2011.

[3] F. Michal Sobolic, Agile Flight Control Techniques for a Fixed-Wing Aircraft, Michigan: Massachusetts Institute of Technology, 2006.

[4] T. Espinoza, A. Dzul and M. Llama, "Linear and Nonlinear Controllers Applied to Fixed-Wing UAV," *International Journal of Advanced Robotic Systems,* 2012.

[5] D. L. A. Gaveau, S. Sloan, E. Molidena, H. Yaen, D. Sheil, N. K. Abram, M. Ancrenaz, R. Nasi, M. Quinones, N. Wielaard and E. Meijaard, "Four Decades of Forest Persistence, Clearance and Logging on Borneo," *PLOS One,* 2014.

5. BIBLIOGRAPHY

Abinaya, R., & R., A. (2017). Selection of Low-cost Recovery System for Unmanned Aerial Vehicle. *International Research Journal of Engineering and Technology (IRJET)*.

Adami, T. M., & Jim Zhu, J. (2011). 6DOF Flight Control of Fixed-Wing Aircraft by Trajectory Linearization. *2011 American Conference*.

Basic Navigational Mathematics, Reference Frames and the Earth's Geometry. (2013). En A. Noureldin, T. Karamat, & J. Georgy, *Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration.* Springer.

Budiharta, S. (2014). Restoring degraded tropical forests for carbon and biodiversity. *Environmental Research Letters*.

Cai, G., Chen, B., & Lee, T. (2011). Coordinate Systems and Transformations. En *Unmanned Rotorcraft Systems.* Springer.

DMA WGS 84 Development Committee. (1984). *Department of Defense World Geodetic System 1984 its Definition and Relationships with Local Geodetic Systems.* The Defense Mapping Agency.

Doucet, R. (1981). Resultats de l'ensemencement en Ligne a Trois Intensites en Terrain Sableux Laboure. *Service de la recherche forestière*.

Espinoza, T., Dzul, A., & Llama, M. (2012). Linear and Nonlinear Controllers Applied to Fixed-Wing UAV. *International Journal of Advanced Robotic Systems*.

L. A. Gaveau, D., Sloan, S., Molidena, E., Yaen, H., Sheil, D., Abram, N. K., . . . Meijaard, E. (2014). Four Decades of Forest Persistence, Clearance and Logging on Borneo. *PLOS One*.

Michal Sobolic, F. (2006). *Agile Flight Control Techniques for a Fixed-Wing Aircraft.* Michigan: Massachusetts Institute of Technology.

Phase One Industrial. (s.f.). *Monitoring Dead Trees and Preserving the Green in Yosemite Park.*

PhaseOne Industrial. (s.f.). *Four-Channel Imaging.*

Platanitis, G., & Shkarayev, S. (2005). Integration of an Autopilot for Micro Air Vehicle. *AIAA 2005-7066*.

Sanz Subirana, J., Juan Zornoza, J., & Hernández Pajares, M. (2011). *ESA Navipedia*. Obtenido de https://gssc.esa.int/navipedia/index.php/Transformations_between_ECEF_and_ENU_coordinates

Vassov, R., & Wolters, K. (2005). Aerial Seeding. *Canadian Silviculture*.

W. Beard, R., & W. McLain, T. (2012). *Small unmanned aircraft: theory and practice.* Princeton University Press.

Wu, X. (2004). *A Nonlinear Flight Controller Design for an Advanced Flight Control Test Bed by Trajectory Linearization Method.* Ohio: Ohio University.

Zubrinic, F., & Reuter, F. (s.f.). *Planeamiento de Vuelo Fotográfico para Tomas de Fotografías Verticales.* Faculatad de Ciencias Forestales - UNSE.