CRANFIELD UNIVERSITY

C BALAS

MODELLING AND LINEAR CONTROL OF A QUADROTOR

SCHOOL OF ENGINEERING

MSc THESIS

CRANFIELD UNIVERSITY

SCHOOL OF ENGINEERING

MSc THESIS

Academic year 2006-2007

C BALAS

Modelling and Linear Control of a Quadrotor

Supervisor: Dr J. F. Whidborne

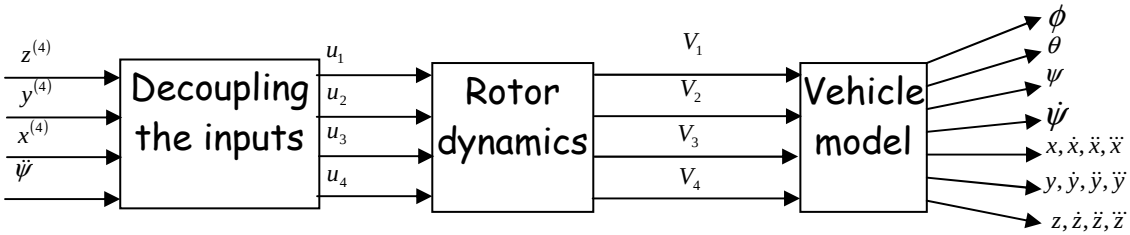
September 2007

This thesis is submitted in partial fulfilment of the requirements for the Degree of Master of Science

Abstract

This report gives details about the different methods used to control the position and the yaw angle of the Draganflyer Xpro quadrotor. This investigation has been carried out using a full non linear Simulink model.

The three different methods are not described chronologically but logically, starting with the most mathematical approach and moving towards the most physically feasible approach.
In order to understand the common features of each approach, it is important to consider the following structure:



The methods differ in the following ways:

- ✓ Modelling the rotor dynamics
- ✓ Decoupling the inputs
- ✓ Designing the control law

It can be foreseen that the mathematical approach will take into account all the different parameters and the following approaches will be simplifications of the first method making justified assumptions.

The first method uses a PID controller and feeds back the following variables: $x, \dot{x}, \ddot{x}, \dddot{x}, y, \dot{y}, \ddot{y}, \dddot{y}, z, \dot{z}, \ddot{z}, \dddot{z}, \psi, \dot{\psi}$.

The second method uses also a PID controller but feeds back $\phi, \dot{\phi}, \theta, \dot{\theta}$ instead of $\ddot{x}, \dddot{x}, \ddot{y}, \dddot{y}$.

The third and last method feeds back the same variables as the second method but uses a simpler model for the rotor dynamics. Both PID and LQR techniques have been investigated with this model.

The achieved performances were not always acceptable. In fact, only the third method gave rise to satisfactory results. Thus, the investigation of aggressive manoeuvres, trajectory tracking and robustness has been carried out only with the third model.

A study of aggressive manoeuvres was undertaken to maintain quadrotor stability for all applied inputs.
The success of the project was measured against the quadrotor's ability to track a given input trajectory.
Finally, the report concludes with suggestions for future work in order to enhance the trajectory tracking and limit the effects of actuator and sensor failures.

Acknowledgements

This thesis is the result of 6 months of work during which I have been accompanied and supported by many people.

First of all, I would like to thank my supervisor, Dr James Whidborne. Before being my thesis supervisor, he was one of my lecturers in Cranfield University. As a lecturer, he taught me all the required materials to achieve successfully this project. As a supervisor, although his time schedule was very busy, he has made the effort to be as available as possible to solve my problems. And thanks to his strong ability to listen to others, his answers were always consistent with my queries.

The other lecturers of the Aerospace Dynamics MSc have also contributed to the development of this work through their taught materials.

Then, I would like to thank several students from my department. Vicente Martinez, the author of the full non linear model of the Draganflyer X-pro, has been very cooperative by answering all my questions about his model. Ian Cowling, a PhD student working on the quadrotor, advised me of the useful literature to read. Tom Carr, one of my flatmates, has given to me all the required support to work efficiently and has also shown a lot of patience for one year to teach me proper English.

Finally, I feel very grateful to my parents, who have always supported me, mentally and financially, within my studies.

Table of contents

Notations

g	Gravitational acceleration	$(m.\sec^{-2})$
I_{xx}	Draganflyer X-pro's moment of inertia along x axis	$(kg.m^2)$
I_{yy}	Draganflyer X-pro's moment of inertia along y axis	$(kg.m^2)$
I_{zz}	Draganflyer X-pro's moment of inertia along z axis	$(kg.m^2)$
l	Arm length of the Draganflyer X-pro (from c.g. to tip)	(m)
m	Mass of the Draganflyer X-pro	(kg)
p	Rate of change of roll angle in body axes system	(*rad/sec*)
q	Rate of change of pitch angle in body axes system	(*rad/sec*)
Q_i	Torque generated by the ith rotor	(*N.m*)
r	Rate of change of yaw angle in body axes system	(*rad/sec*)
T_i	Thrust generated by the ith rotor	(*N*)
u	Airspeed along x axis in body axes system	(*m/sec*)
u_1, u1	Vertical thrust generated by the four rotors	(*N*)
u_2, u2	Rolling moment	(*N.m*)
u_3, u3	Pitching moment	(*N.m*)
u_4, u4	Yawing moment	(*N.m*)
v	Airspeed along y axis in body axes system	(*m/sec*)
V_i	Voltage applied on the ith rotor	(*Volts*)
w	Airspeed along z axis in body axes system	(*m/sec*)
x	x coordinate of the Draganflyer X-pro's c.g. (Earth axes)	(*m*)
y	y coordinate of the Draganflyer X-pro's c.g. (Earth axes)	(*m*)
z	z coordinate of the Draganflyer X-pro's c.g. (Earth axes)	(*m*)
ϕ	Roll angle of the Draganflyer X-pro (Euler angles)	(*rad*)
θ	Pitch angle of the Draganflyer X-pro (Euler angles)	(*rad*)
ψ	Yaw angle of the Draganflyer X-pro (Euler angles)	(*rad*)

Derivatives with respect to time are expressed with the dot sign above the variable names.

List of Figures

1. Introduction

Sensing and actuating technologies developments make, nowadays, the study of mini Unmanned Air Vehicles (UAVs) very interesting. Among the UAVs, the VTOL (Vertical Take Off and Landing) systems represent a valuable class of flying robots thanks to their small area monitoring and building exploration.

In this work, we are studying the behaviour of the quadrotor. This flying robot presents the main advantage of having quite simple dynamic features. Indeed, the quadrotor is a small vehicle with four propellers placed around a main body.

The main body includes power source, sensors and control hardware. The four rotors are used to controlling the vehicle. The rotational speeds of the four rotors are independent. Thanks to this independence, it's possible to control the pitch, roll and yaw attitude of the vehicle. Then, its displacement is produced by the total thrust of the four rotors whose direction varies according to the attitude of the quadrotor. The vehicle motion can thus be controlled.

However, a closed loop control system is required to achieve stability and autonomy.

The aim of this project is to control the position and the yaw angle of the Draganflyer X-pro quadrotor using PID (proportional-integral-derivative) and LQR (linear quadratic regulator) controllers. This vehicle is represented by a full non linear Simulink model developed with experimental data.

The closed loop system is designed to be robustly stable. The desired position has to be reached as fast as possible without any steady state error.

In order to measure these performances, the UAV's ability to track a given input trajectory will be assessed through Simulink simulations. Also, the effects of actuator

and sensor failures will be investigated in order to evaluate the robustness of the vehicle.

Project's deliverables:

- ✓ one Simulink model using PID control technique
- ✓ one Simulink model using LQR controller
- ✓ the associated Matlab files
- ✓ an interface in order to tune the parameters and run the simulations more easily
- ✓ Two videos showing the flight trajectory of the UAV in real time. These videos have been made with Matlab and use the simulated results

These deliverables are in the enclosed DVD. Details on the DVD contents are in the appendix 6. Print screens of the software interface are in the appendix 7.

2. Literature review

2.1. Modelling

2.1.1. Body axes system

In most of papers, the body axes orientation is along the arms of the vehicle as shown on the following figure.

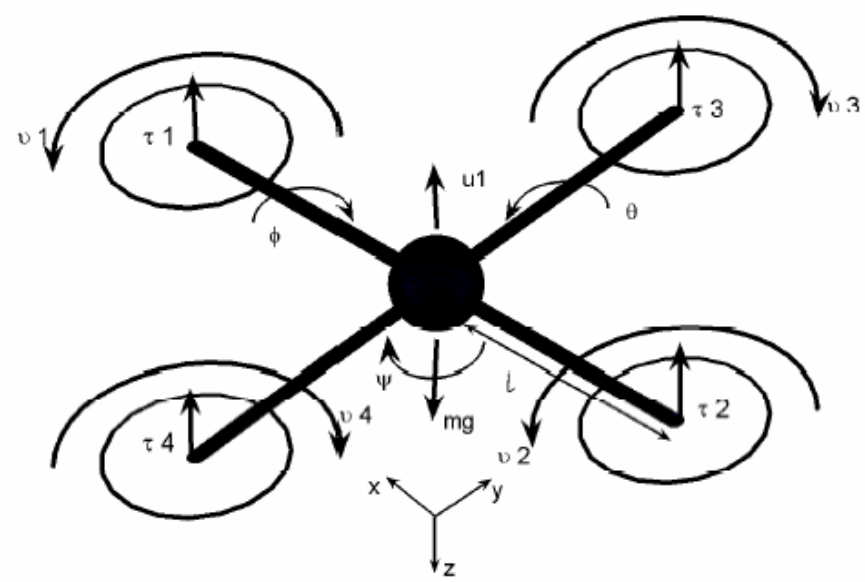


Figure 2.1: quadrotor schematic
(Taken from Cowling, et al. [8] without permission)

However, Mokhtari and Benallegue have tried to model the vehicle with a different axes orientation [13]:

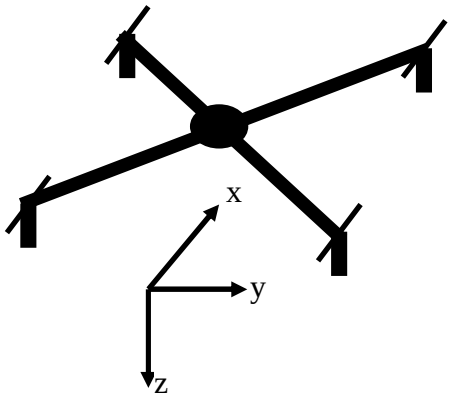


Figure 2.2: alternative orientation for the body axes system

As no comparison has been carried out between the two different axes orientation, we can't say which one is the better one. As this project uses the model realised in [8], we are going to work with the more widely used orientation, which means with x and y axes along the arms of the robot.
Also, the body axes centre is assumed to be at the same position as the centre of gravity.

2.1.2. The equations of motion

Two different methods have been investigated to achieve this task. We can either use the Lagrangian equation as in [4], [5] or the Newton's law as in the other papers. Let's explain the second method which is more comprehensible.

The quadrotor is controlled by independently varying the speed of the four rotors.
Hence, with the notation of the figure 2.1 (υ_i and τ_i are respectively the normalized torque and normalized thrust from the ith rotor), we have the following inputs:

- ✓ The total thrust: $u_1 = \tau_1 + \tau_2 + \tau_3 + \tau_4$
- ✓ The rolling moment: $u_2 = l(\tau_3 - \tau_4)$
- ✓ The pitching moment: $u_3 = l(\tau_1 - \tau_2)$
- ✓ The yawing moment: $u_4 = \upsilon_1 + \upsilon_2 - \upsilon_3 - \upsilon_4$

The way of modelling the quadrotor differs from the one used for fixed wing vehicle in the fact that we are not making the rotational transformations in the same order to go from the earth to body axes. Indeed, the most practical way is to carry out the final rotation of the earth to body transformation along the thrust direction [8]. Thus, we take for the body to earth transformation, the following direction cosine matrix:

$$R_{zxy} = \begin{bmatrix} s_\theta s_\phi s_\psi + c_\psi c_\theta & s_\phi s_\theta c_\psi - c_\theta s_\psi & s_\theta c_\phi \\ c_\phi s_\psi & c_\psi c_\phi & -s_\phi \\ c_\theta s_\psi s_\phi - s_\theta c_\psi & c_\theta s_\phi c_\psi + s_\theta s_\psi & c_\theta c_\phi \end{bmatrix}$$

where:

- ϕ, θ, ψ are the roll, pitch and yaw angle respectively
- $s_\psi = \sin(\psi), c_\psi = \cos(\psi), t_\phi = \tan(\phi)$, etc

Thus, $$\ddot{x} = -\frac{s_\theta c_\phi}{m} u_1,\ \ddot{y} = \frac{s_\phi}{m} u_1,\ \ddot{z} = -\frac{c_\theta c_\phi}{m} u_1 + g$$

(where x, y and z are the translational positions)

Also, to relate Euler angular rates to body angular rates, we have to use the same order of rotation. This gives rise to:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} c_\psi & -s_\psi & 0 \\ \frac{s_\psi}{c_\phi} & \frac{c_\psi}{c_\phi} & 0 \\ s_\psi t_\phi & c_\psi t_\phi & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

By differentiating,

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} -\dot{\psi} * s_\psi & -\dot{\psi} * c_\psi & 0 \\ \dot{\psi} * c_\psi c_\phi + \dot{\phi} * s_\psi s_\phi & -\dot{\psi} * s_\psi c_\phi + \dot{\phi} * c_\psi s_\phi & 0 \\ \hline c_\phi^{\,2} & c_\phi^{\,2} & \\ \dot{\psi} * c_\psi t_\phi + \frac{\dot{\phi} * s_\psi}{c_\phi^{\,2}} & -\dot{\psi} * s_\psi t_\phi + \frac{\dot{\phi} * c_\psi}{c_\phi^{\,2}} & 0 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + \begin{bmatrix} c_\psi & -s_\psi & 0 \\ \frac{s_\psi}{c_\phi} & \frac{c_\psi}{c_\phi} & 0 \\ s_\psi t_\phi & c_\psi t_\phi & 1 \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & -\dot{\psi} * c_\phi & 0 \\ \frac{\dot{\psi}}{c_\phi} & \dot{\phi} * t_\phi & 0 \\ \dot{\psi} * t_\phi & \frac{\dot{\phi}}{c_\phi} & 0 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} c_\psi & -s_\psi & 0 \\ \frac{s_\psi}{c_\phi} & \frac{c_\psi}{c_\phi} & 0 \\ s_\psi t_\phi & c_\psi t_\phi & 1 \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}$$

If I is the inertia matrix of the vehicle and $\vec{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$,

$$\frac{d(I\vec{\omega})}{dt} = \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} = I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \vec{\omega} \wedge (I\vec{\omega}) \qquad \Rightarrow \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} + I^{-1}(I\vec{\omega}) \wedge \vec{\omega}$$

Assuming that the structure is symmetrical ([1] and [13]),

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

In some papers, the second term of the right side of the above equation ($I^{-1}(I\vec{\omega}) \wedge \vec{\omega}$) is neglected [8], [19]. This approximation can be made by assuming that:

- ✓ the angular rate about the z axis, r, is small enough to be neglected
- ✓ $I_{xx} = I_{yy}$

Let's just assume, for the moment, that the moments of inertia along the x axis and y axis are equalled [8].

Hence,

$$\ddot{\phi} = -\dot{\psi}\dot{\theta} * c_\phi + \frac{c_\psi}{I_{xx}} u_2 - \frac{s_\psi}{I_{yy}} u_3 + \frac{I_{yy} - I_{zz}}{I_{xx}} (\dot{\psi} - \dot{\theta} * s_\phi) \dot{\theta} * c_\phi$$

$$\ddot{\theta} = \frac{\dot{\psi}\dot{\phi}}{c_\phi} + \dot{\phi}\dot{\theta} * t_\phi + \frac{s_\psi}{c_\phi I_{xx}} u_2 + \frac{c_\psi}{c_\phi I_{yy}} u_3 - \frac{I_{yy} - I_{zz}}{I_{xx}} (\dot{\psi} - \dot{\theta} * s_\phi) \frac{\dot{\phi}}{c_\phi}$$

$$\ddot{\psi} = \dot{\phi}\dot{\psi} * t_\phi + \frac{\dot{\phi}\dot{\theta}}{c_\phi} + \frac{s_\psi t_\phi}{I_{xx}} u_2 + \frac{c_\psi t_\phi}{I_{yy}} u_3 + \frac{1}{I_{zz}} u_4 - \frac{I_{yy} - I_{zz}}{I_{xx}} (\dot{\psi} - \dot{\theta} * s_\phi) \dot{\phi} * t_\phi$$

These equations have been established assuming that the structure is rigid.
The gyroscopic effect resulting from the propellers rotation has been neglected. The investigation of this effect has been done in [12], [17].
If we want to take into account these gyroscopic torques, due to the combination of the rotation of the airframe and the four rotors, we have to consider the following equation [17]:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I^{-1} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} + I^{-1}(I\vec{\omega}) \wedge \vec{\omega} + I^{-1} G_a$$

$$\text{with: } \begin{cases} G_a = I_r (\vec{\omega} \wedge \vec{e}_z) \omega_{m\_d} \\ \omega_{m\_d} = \omega_{m\_2} + \omega_{m\_4} - \omega_{m\_1} - \omega_{m\_3} \end{cases}$$

2.1.3. Dynamic of the rotor

In [8], Cowling assumes that the influence of the actuator dynamics can be neglected. Thus, he only considers a linear relationship between the voltage applied to each rotors and the associated rotor speeds.

In [4], Bouabdallah takes into consideration the rotor dynamics.

The motor time constant needs to be investigated to see if it's small enough to be neglected.

If it's not the case, we should take into consideration the following equation [4]:

$$\begin{cases} \dot{\omega}_m = -\frac{1}{\tau}\omega_m - \frac{d}{\eta r^3 J}{\omega_m}^2 + \frac{1}{k_m \tau}u \\ \frac{1}{\tau} = \frac{{k_m}^2}{RJ} \end{cases}$$

with:

- ✓ ω_m : motor angular speed
- ✓ u : motor input
- ✓ τ : motor time constant
- ✓ k_m : torque constant
- ✓ d : drag factor
- ✓ η : gear box efficiency
- ✓ r : gear box reduction ratio
- ✓ J : propeller inertia
- ✓ R : motor internal resistance

Then, we have to relate the rotor speed with the thrust and the torque as done in [8]:

- ✓ Thrust is proportional to the square of the rotational speed
- ✓ Torque: $\upsilon_i = \frac{\tau_i (V_C + v_i) + 0.125 \rho b c {R_p}^4 {\omega_{m\_i}}^3 C_d}{\omega_{m\_i}}$

  with:

  - τ_i : thrust acting on the ith rotor
  - V_C : vertical speed

- v_i : induced velocity
- ρ : air density
- b : number of blades
- c : chord of the blade
- R_p : radius of the propeller
- C_d : drag coefficient

2.1.4. Control perspective

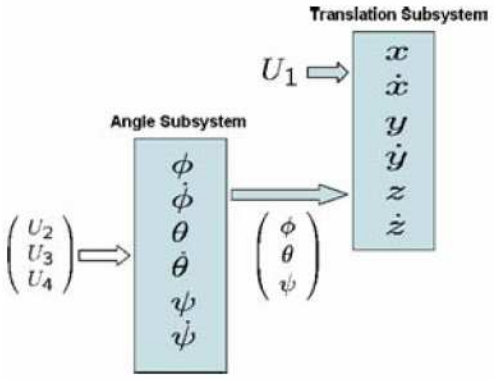


Figure 2.3: decomposition of the dynamical model into two subsystems
(taken from Bouabdallah, et al. [2] without permission)

As mentioned in [2], the angular attitude of the VTOL does not depend on translation components whereas the translational motion depends on angles.

Hence, it seems to be more judicious to control firstly the rotational aspect of the vehicle because of its independence and then to consider the control of the translational motion.
Indeed, after having designed and optimized the attitude controller, we can use these new dynamical features to enhance the translational motion. Thus, the attitude controller would form a part of an inner loop, and the translational controller would be placed in an outer loop.

2.2. Control of the quadrotor

A lot of different methods have already been studied to achieve autonomous flights. As this paper is about linear controller, we are not going to give too many details on what has been done about non linear controllers and visual feedback. But note that autonomous flight and trajectory following have been achieved using visual feedback and using some non linear control techniques as well.

2.2.1. PID controller

This control technique has already been investigated in [4] to stabilize the attitude of the quadrotor.

To design this controller, the model has been linearised around the hover situation. Hence, the gyroscopic effects haven't been taken into consideration in the controller design.

The closed loop model has been simulated on Simulink with the full non linear model. The controller parameters have been adjusted with this more complete model. The simulation has lead to satisfactory results. The quadrotor attitude stabilizes itself after 3 seconds.

This simulation has been validated on the real system. During the test, a closed loop speed control has been implemented on each rotor. This speed control enables a faster response. The experimental results are consistent with the theoretical ones. The control of the robot attitude remains efficient around the hover.

However, we have to bear in mind that this performance is valid only around the hover. If the VTOL undergoes a strong perturbation, it may not be able to recover on its own the hover situation. Also, the robustness of the obtained closed loop system has not been studied. The failure of an actuator, for example, is likely to deteriorate seriously the dynamic properties.

2.2.2. LQR controller

✓ Classic LQR

Hoffmann et al. used this technique in the attitude loop [11]. At low thrust levels, the control was satisfactory but at higher thrust levels, performance was degraded due to vibrations. A solution to this problem is to apply lower costs on attitude deviations by varying the matrix Q but this degrades tracking performance. A good compromise has to be found.

Castillo implemented iteratively from simulation results LQR controller to make the quadrotor hover correctly [5]. The feedback was applied to y and ϕ.

In [10], Cowling is using the same kind of controller on x, y, z and ψ to follow a reference trajectory. However, his LQR controller has been designed with a model linearized at the hover. His simulation shows a flight path quite consistent with the reference trajectory.

✓ State dependent LQR

Bouabdallah has already implemented such kind of controller in the closed loop system to stabilize the angular attitude of the UAV ([4]). His method was adapted through the robot trajectory. Indeed, in order to optimize the system for a larger flight envelope than the hover configuration, he has linearized the state space representation around each flight condition.

Then, he has applied the classical techniques to get the associated LQR control gains at any state. As he didn't taken into account the actuators dynamics, he obtained only average performance in his flight experiment.

This technique has been called state dependent Riccati equation control in [18].

2.2.3. H infinity controller

In [7], Chen has studied the influence of a H_∞ controller in the closed loop system for position control (feedback on w, θ, ϕ and r). The simulation based on a non linear model leads to satisfactory results. Indeed, he succeeded to obtain "robustness, good reference tracking and disturbance rejection" thanks to a two degree of freedom architecture (Chen, 2003). This kind of architecture allows decoupling of commands and tracking control with measurement control.

In [6], Chen investigates the effects of combining Model Based Predictive Control (MBPC) with two degree of freedom H_∞ controller.

The role of the H_∞ controller is to get a robust stability and a good control of the trajectory. The role of the MBPC controller is to enable longitudinal and lateral trajectory control for a large flight envelope.

The H_∞ controller has been divided into two different loops. The inner loop stabilizes the roll and pitch angles, the yaw rate and the vertical speed. The outer loop considers longitudinal and lateral speed, the height and the yaw angle. This outer loop is then closed with the MBPC controller.

Disturbances and various inputs and outputs constraints have been tested and have given rise to satisfactory performance.

In [14], Mokhtari examines the influence of robust feedback linearization and GH_∞ controller on the quadrotor. The loop is applied on x, y, z and ψ. He inferred that when the weighting functions are judiciously chosen, the tracking error of the desired trajectory is satisfactory. These convergent outputs are obtained even when uncertainties on system parameters and disturbances occur.

2.2.4. Alternative methods

✓ Feedback linearization controller

In [1], Benallegue et al. use an inner and an outer loop to control the robot. The role of the inner loop is to obtain a linear relationship between the inputs and the outputs so that they can apply linear control techniques to the system.

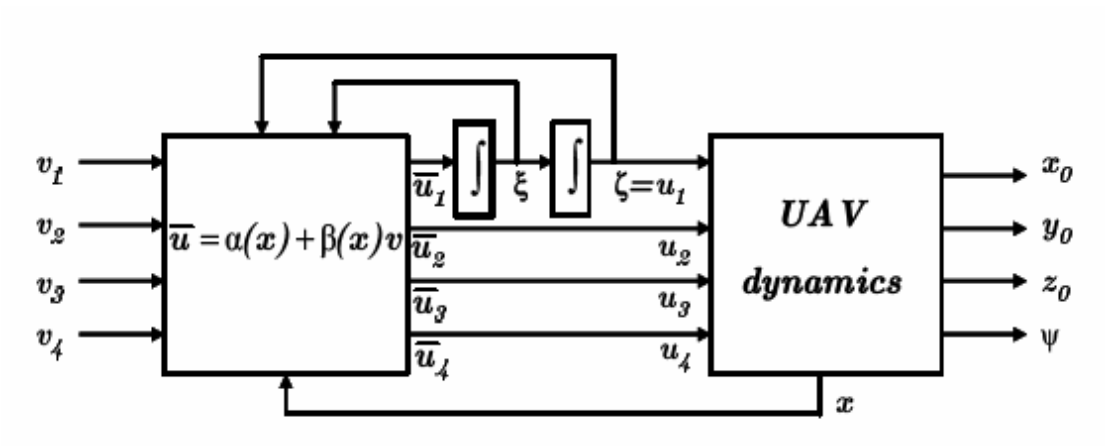


Figure 2.4: block diagram of the inner loop
(taken from Benallegue, et al. [1] without permission)

Then, the outer loop is the classical linear controller (polynomial control law).
This technique is called feedback linearization controller.
The advantage of this method is that linear controller is not designed around a specific state. The performances achieved by designing the closed loop system are now valid for all flight conditions.

✓ Pole placement

This well known technique has been used to control the height in [2] and [3] and to control the velocity in [18].

✓ Double lead compensator

In [15], Pounds et al. augment the attitude performance of the vehicle by placing a double lead compensator in an inner loop and a proportional controller in an outer loop.

- ✓ State estimator

Also, to avoid noise differentiation in the outer loop, it's possible to add an observer as in [1], [11] and [19]. This enables the outputs to be reconstructed and estimated without any sensor. Thus, we don't have any measurement noise and we can differentiate the outputs without increasing any parasite signals.

- ✓ Non linear methods

Respecting Lyapunov criterion enables simple stability to be ensured for equilibrium.
In [2] and [3], Bouabdallah et al. use Lyapunov criterion on the angular components as well as in [16] and [17]. Concerning the height control, they use the pole placement method. The augmented vehicle gives good results in both simulations and flight experiments. However, this paper does not investigate position control.
This criterion seems to be efficient as it has also been successfully used in [11] and [19] through integral sliding mode to stabilize the altitude.

2.3. Summary

Technique applied / Objective	PID	LQR		H_∞	Alternative methods
		Classic	State dependent		
Control of altitude				[6] (in an outer loop)	✓ Pole placement: [2], [3] ✓ Feedback linearization: [13], [14], [1] (with observer) ✓ Non linear: [11], [19] (Integral sliding mode with observer)
Control of attitude	[4], [15]	[11], [19] (with observer)	[18], [4] (with observer)	[6] (θ, ϕ in an inner loop + ψ in an outer loop)	✓ Feedback linearization: [13], [14], [1] (with observer) ✓ Double lead compensator: [15] ✓ Non linear: [2], [3], [16], [17] (Lyapunov)
Control of the horizontal components of the position		[19] (with observer)			✓ Feedback linearization: [14], [1] (with observer)
Control of the velocity				[6] (w in an inner loop + u, v in an outer loop)	✓ Pole placement: [18]
Others		✓ Control of x, y, z, ψ: [10] ✓ Control of y, ϕ: [5]		✓ Control of x, y, z, ψ: [14] ✓ Control of w, θ, ϕ, r: [7] ✓ Inner loop on r: [6]	

As we can notice, the modelling of the quadrotor differs from one paper to another one. According to the assumptions which have been considered, we don't have the same equations of motion. Thus, as a first step of the thesis, it can be interesting to investigate the effect of these assumptions and check if they are judicious enough to be taken into account in the full non linear model.

Concerning the control design, a lot of different techniques have already been applied to achieve autonomous flight. As described previously, there is not one linear controller which enables both good tracking performance and robust stability.

Thus, it should be interesting to study the influence of each technique and to find the combination which optimizes these performances.

Therefore, the purpose of this project would be to design this complex control system to get still better performances for both simulation and flight experiment. This control law will also intend to minimize the effects of actuator and sensor failures by enhancing the robustness of the system.

3. Using a mathematical approach: acceleration feedback

This approach aims to consider all the equations of the vehicle dynamics. The approximations tend to be minimized.

The feedback variables are: $\dddot{x}, \ddot{x}, \dot{x}, x, \dddot{y}, \ddot{y}, \dot{y}, y, \dddot{z}, \ddot{z}, \dot{z}, z, \dot{\psi}, \psi$.

3.1. Modelling the rotor dynamics

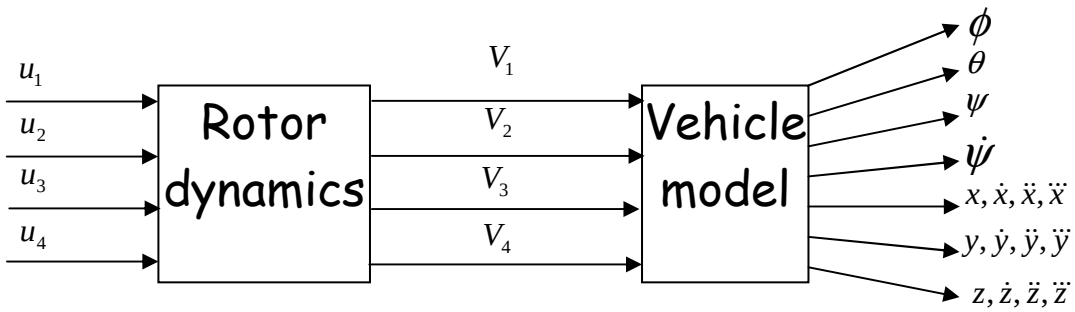


Figure 3.1: location and role of the rotor dynamics block in the mathematical approach

Modelling the rotor dynamics enables new inputs to be considered. These inputs are more meaningful than voltages:

- ✓ Vertical thrust: u1
- ✓ Rolling moment: u2
- ✓ Pitching moment: u3
- ✓ Yawing moment: u4

This step is carried out investigating the step responses of the Simulink vehicle model. Because of aerodynamics properties, the rotors are far from being linear. For example, a positive step applied on the four voltages doesn't give rise to the same response as a negative one.

Thus, it is firstly necessary to find out the four voltage combinations which are going to be used to controlling the vehicle's motion in order to model, then, the relations between these voltage combinations and the famous variables u1, u2, u3, u4.

3.1.1. The four voltage combinations

According to the full non linear model, the convention used is as followed:

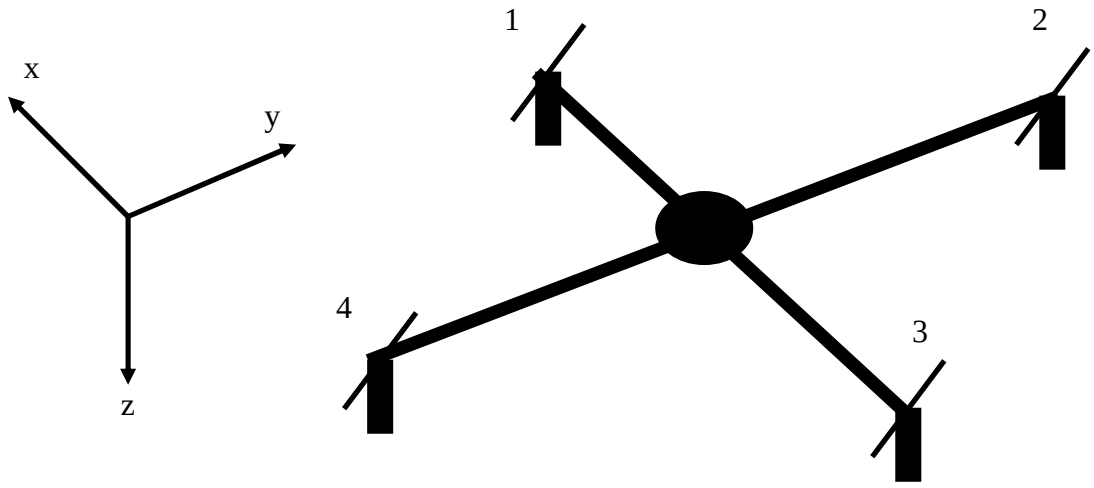


Figure 3.2: Simulink model's convention

Using the model's notation, the quadrotor is controlled by:

- ✓ Vertical thrust (sum of the four thrusts): $u_1 = T_1 + T_2 + T_3 + T_4$
- ✓ Rolling moment (thrust difference): $u_2 = l(T_4 - T_2)$
- ✓ Pitching moment (thrust difference): $u_3 = l(T_1 - T_3)$
- ✓ Yawing moment (algebraic sum of the four torques): $u_4 = Q_1 + Q_2 + Q_3 + Q_4$

Therefore, the four voltage combinations used are:

- ✓ Vertical thrust (motion along z axis): $V_1 + V_2 + V_3 + V_4$
- ✓ Rolling moment (motion along y axis): $V_4 - V_2$
- ✓ Pitching moment (motion along x axis): $V_1 - V_3$
- ✓ Yawing moment (control of psi): $V_1 - V_2 + V_3 - V_4$

Thus, to go from these combinations to the voltages, we use this transformation:

$$\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} 0.25 & 0 & 0.5 & 0.25 \\ 0.25 & -0.5 & 0 & -0.25 \\ 0.25 & 0 & -0.5 & 0.25 \\ 0.25 & 0.5 & 0 & -0.25 \end{bmatrix} \begin{bmatrix} V_1 + V_2 + V_3 + V_4 \\ V_4 - V_2 \\ V_1 - V_3 \\ V_1 - V_2 + V_3 - V_4 \end{bmatrix}$$

In Simulink, this gives rise to the following connections:

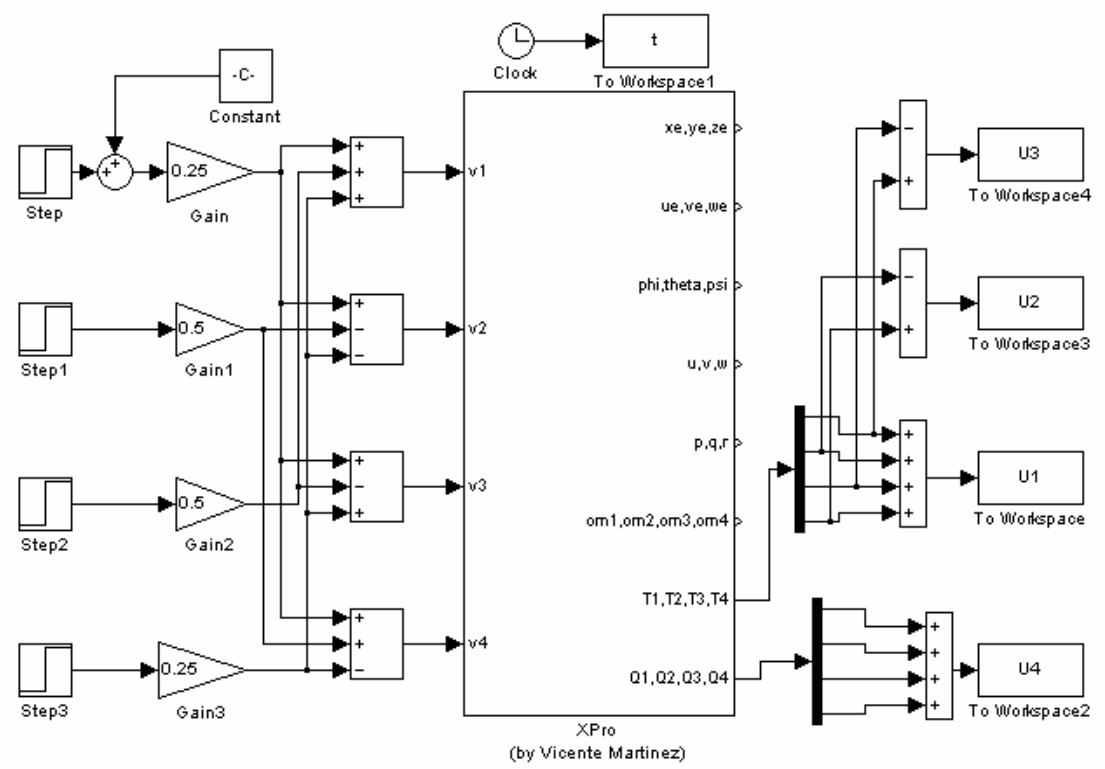


Figure 3.3: Simulink model for the voltages combination

From now on, thanks to this model, we can work out the relations between the voltage combinations and u1, u2, u3 and u4 by looking at the four different step responses.

Note that this step is carried out around the hover. We are not going to give any details about the influence of the vertical speed on the rotor dynamics but this work has been done while studying the control of vertical flight.

The rotors had been modelled around three different flight conditions:

- ✓ Climbing
- ✓ Hover

- ✓ Descent

It has led to the following conclusion: the control of altitude for vertical flight is not enhanced by taking into account the influence of vertical speed on the rotor dynamics. The associated gain schedule was worthless. The performances achieved weren't better than those obtained with the model around the hover. Consequently, only this model has been kept.

3.1.2. Vertical thrust

While we apply a unit step to the first new input ($V_1 + V_2 + V_3 + V_4$), we can observe the following response on u1:

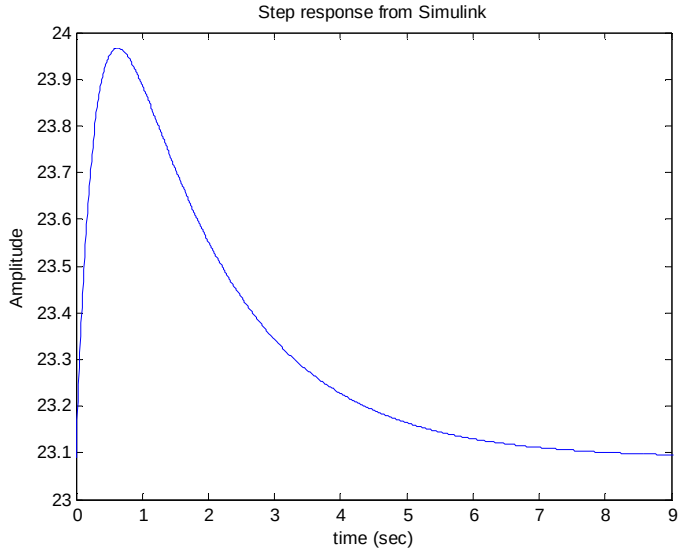


Figure 3.4: vertical thrust response to a step applied on $V_1 + V_2 + V_3 + V_4$

Note that the initial value of the applied voltage was different from zero in order to make the quadrotor hover. That's why the initial value of the vertical thrust is different from zero.

The quadrotor hovers when the sum of the four voltages equals 29.2238 V. This corresponds to a vertical thrust equalled to: $m * g = 2.354 * 9.81 = 23.0927$ N.

Now, we want to find the transfer function whose step response is as close as possible to the previous plot. Let's call this transfer function H and the associated step response y. We have:

$$Y(s) = H(s) * \frac{1}{s}$$

As the step response looks like a second order impulse response, it's easier to consider y as the impulse response of the second order system H1. We have:

$$H1(s) = Y(s) = H(s) * \frac{1}{s} = \frac{K}{(1+\tau_1 s)(1+\tau_2 s)}$$

Thanks to the inverse Laplace transform,

$$y(t) = \frac{K}{\tau_1 - \tau_2}\left(\exp\left(-\frac{t}{\tau_1}\right) - \exp\left(-\frac{t}{\tau_2}\right)\right)u(t)$$

Then, to determine the three parameters, we apply the three following conditions:

- ✓ $\dot{y}(0) = 4.3$
- ✓ $\dot{y}(0.63) = 0$
- ✓ $y(0.63) = 0.875$

Solving the set of equations gives:

$$H(s) = H1(s) * s = \frac{2.105s}{0.4895s^2 + 1.873s + 1}$$

As the final value is slightly different from zero, we need to add an other term.

$$H(s) = \frac{2.105s + a}{0.4895s^2 + 1.873s + 1}$$

Using final value theorem, we have $y(\infty) = a = -0.0425$. Hence, the modelling gives rise to the following non minimal phase system:

$$\frac{u_1}{V_1 + V_2 + V_3 + V_4} = \frac{2.105s - 0.0425}{0.4895s^2 + 1.873s + 1} \quad \text{N/V}$$

This study is validated by comparing the step response of the above transfer function with the scope from the simulation:

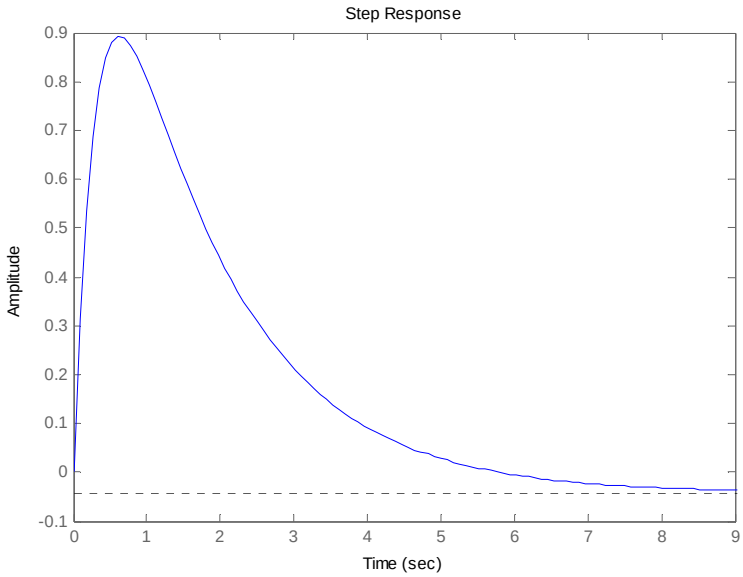


Figure 3.5: modelling of the relation between vertical thrust and $V_1 + V_2 + V_3 + V_4$

The two plots look similar enough.

The thrust response to a step applied on the voltages may look different from the one intuitively expected, but the influence of the vertical speed on the provided thrust must be born in mind.

The associated Simulink subsystem is:

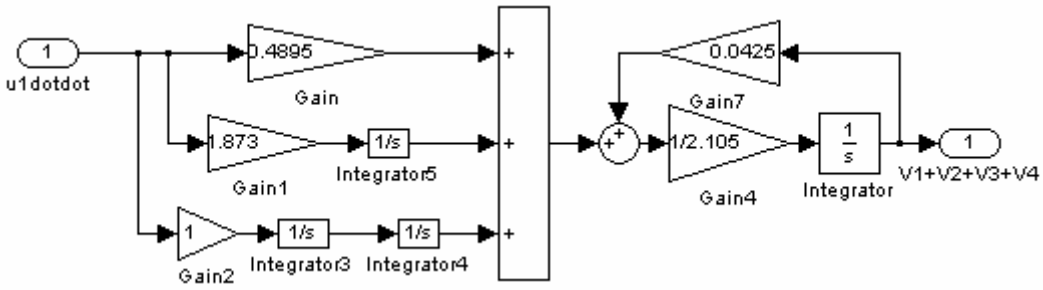


Figure 3.6: Simulink subsystem relating vertical thrust to $V_1 + V_2 + V_3 + V_4$

3.1.3. Pitching and rolling moments

For this part, we apply a step of 0.001 V to the second or third new input. The amplitude is small in order to get rid off the singularities. Indeed, the UAV must not be too banked.

We can observe the following response on u2/l or u3/l:

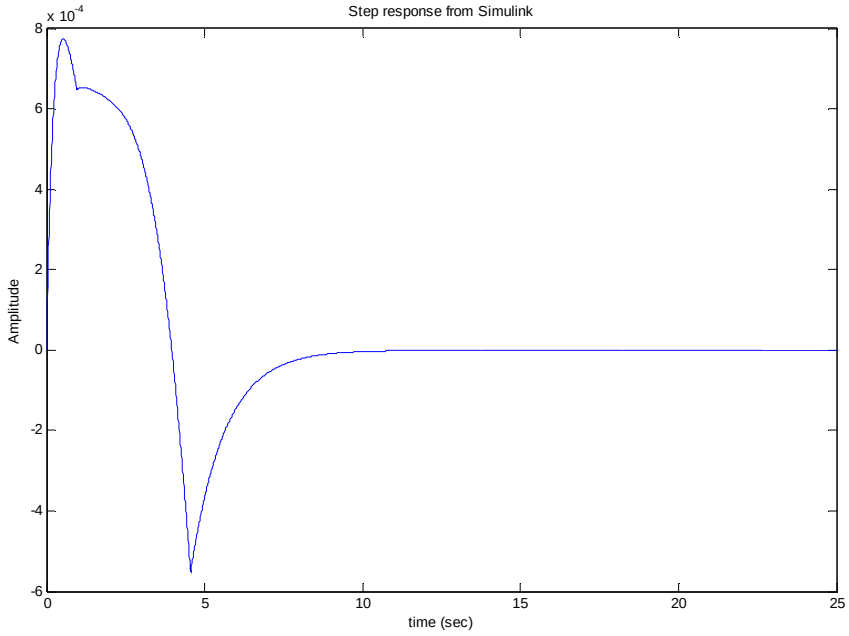


Figure 3.7: pitching moment response to a step applied on $V_1 - V_3$

Note that the response is completely different from the previous one whereas u2 and u3 are only linear combinations of thrusts. This is because of the non linearities of the rotors. A negative voltage step doesn't have the same influence as a positive one on the provided thrust.

Once again, this plot can be approximated by the impulse response of a second order system. This time, the damping ratio is less than 1.

Say t_1 is the time of the first overshoot and t_2 the time of the first undershoot.

Thus, we have: $t_2 - t_1 = 3\,\mathrm{s}$

We know that the impulse response for a second order system, whose damping ratio is inferior to 1, is:

$$y(t) = A\exp(-\zeta\omega_n t)\sin\left(\omega_n\sqrt{1-\zeta^2}\,t\right)$$

Now, $y(t_1)$ and $y(t_2)$ are respectively local maximum and local minimum. Hence,

$$\sin\left(\omega_n\sqrt{1-\zeta^2}\,t_1\right)=1 \qquad \sin\left(\omega_n\sqrt{1-\zeta^2}\,t_2\right)=-1 \qquad \text{and} \qquad \omega_n\sqrt{1-\zeta^2}\,t_2-\omega_n\sqrt{1-\zeta^2}\,t_1=\pi$$

$$\Rightarrow \quad \left|\frac{y(t_1)}{y(t_2)}\right| = 2 = \left|\frac{A\exp(-\zeta\omega_n t_1)\sin\left(\omega_n\sqrt{1-\zeta^2}\,t_1\right)}{A\exp(-\zeta\omega_n t_2)\sin\left(\omega_n\sqrt{1-\zeta^2}\,t_2\right)}\right| = \frac{A\exp(-\zeta\omega_n t_1)}{A\exp(-\zeta\omega_n t_2)} = \exp(\zeta\omega_n(t_2-t_1))$$

$$\zeta\omega_n = \frac{\ln(2)}{t_2-t_1} = \frac{\ln(2)}{3} \qquad (1)$$

$$\Rightarrow \quad \omega_n\sqrt{1-\zeta^2} = \frac{\pi}{t_2-t_1} = \frac{\pi}{3} \qquad (2)$$

$$(1)^2+(2)^2 \quad \Rightarrow \quad \omega_n = \sqrt{\left(\frac{\ln(2)}{3}\right)^2+\left(\frac{\pi}{3}\right)^2} = 1.07 \text{ rad/sec}$$

And, $\zeta = \dfrac{\ln(2)}{3\omega_n} = \dfrac{\ln(2)}{3*1.07} = 0.215$.

At last,

$$y(t_1) = A\exp(-\zeta\omega_n t_1)\sin\left(\omega_n\sqrt{1-\zeta^2}\,t_1\right) = 0.001*K*\exp(-\zeta\omega_n t_1) = 0.8*0.001$$

$$\Rightarrow \; K = 1.155 \text{ N.m/V}$$

Therefore, $\dfrac{u_2}{V_4-V_2} = \dfrac{u_3}{V_1-V_3} = l\dfrac{K*s}{\dfrac{s^2}{\omega_n^2}+2\zeta\dfrac{s}{\omega_n}+1} = l\dfrac{1.155s}{0.8696s^2+0.4018s+1}$.

The associated step response is not very similar to the previous scope, but this is the best approximation we can make using a second order system.

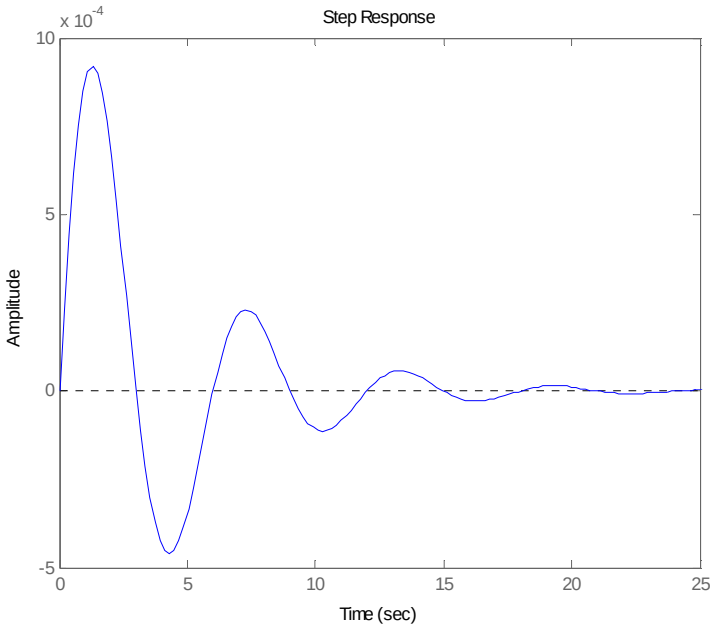


Figure 3.8: modelling the relation between pitching moment and $V_1 - V_3$

Nevertheless, we can notice that the time constants and amplitude are in the same order of magnitude as those obtained in the simulation.

The associated Simulink subsystems are exactly the same for pitching and rolling moments:

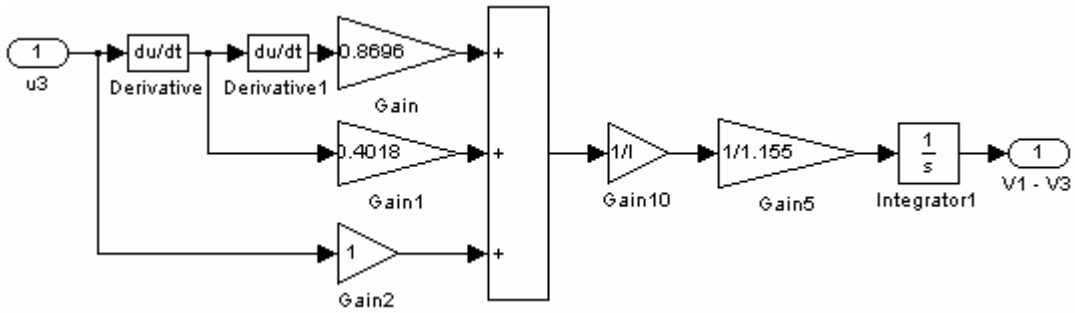


Figure 3.9: Simulink subsystem relating pitching moment to $V_1 - V_3$

3.1.4. Yawing moment

When we apply a unit step to the fourth new input, we can observe the following response on u4 (torque response):

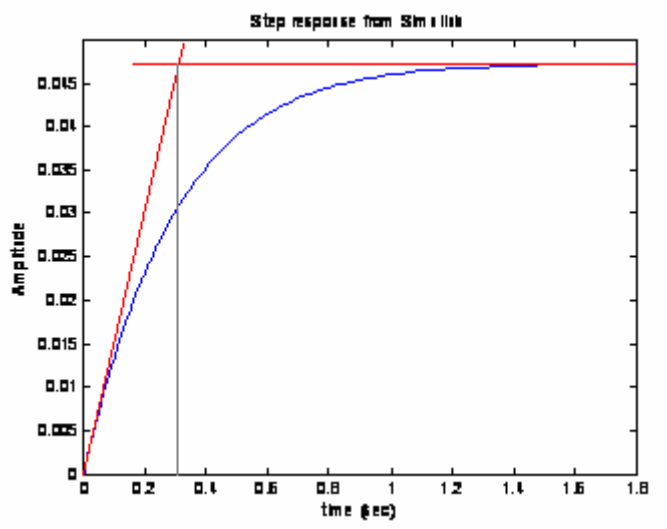


Figure 3.10 : yawing moment response to a step applied on $V_1 - V_2 + V_3 - V_4$

This time, the step response looks like the step response of a first order system. Thus, the relation between voltage and torque can be approximated by:

$$H(s) = \frac{K}{1 + \tau * s}$$

The time constant can be read on the above figure. We have: $\tau = 0.314\,\text{s}$.

The D.C. gain is also read on the figure thanks to the steady state value. We have: $K = 0.045\,\text{N.m/V}$.

Therefore,

$$\frac{u_4}{V_1 - V_2 + V_3 - V_4} = \frac{0.045}{0.314s + 1}$$

This approximation is very close to the reality. Indeed, the associated step response matches perfectly with the simulation result:

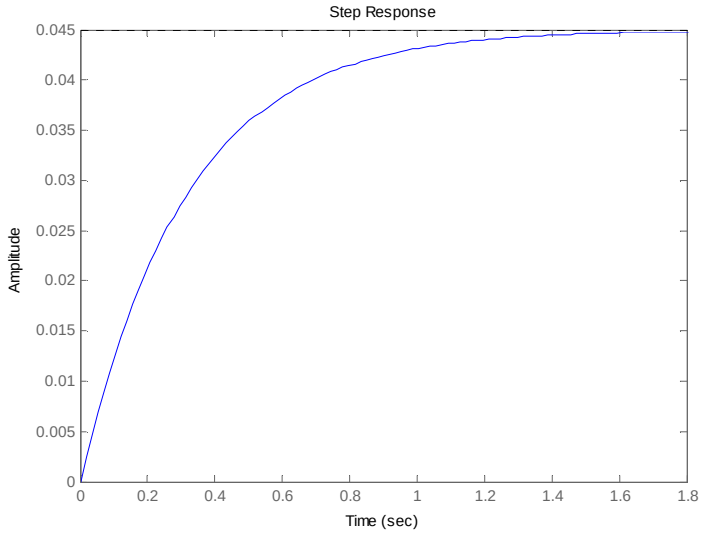


Figure 3.11: modelling the relation between yawing moment and $V_1 - V_2 + V_3 - V_4$

The associated Simulink subsystem is:

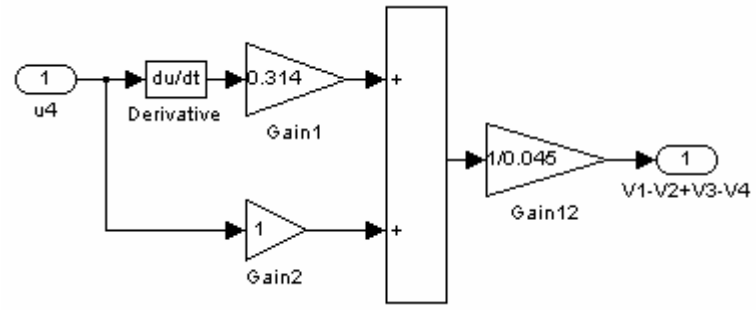


Figure 3.12: Simulink subsystem relating yawing moment to $V_1 - V_2 + V_3 - V_4$

Thanks to the modelling of the rotor dynamics, we can now work with the following input variables: u1, u2, u3 and u4.

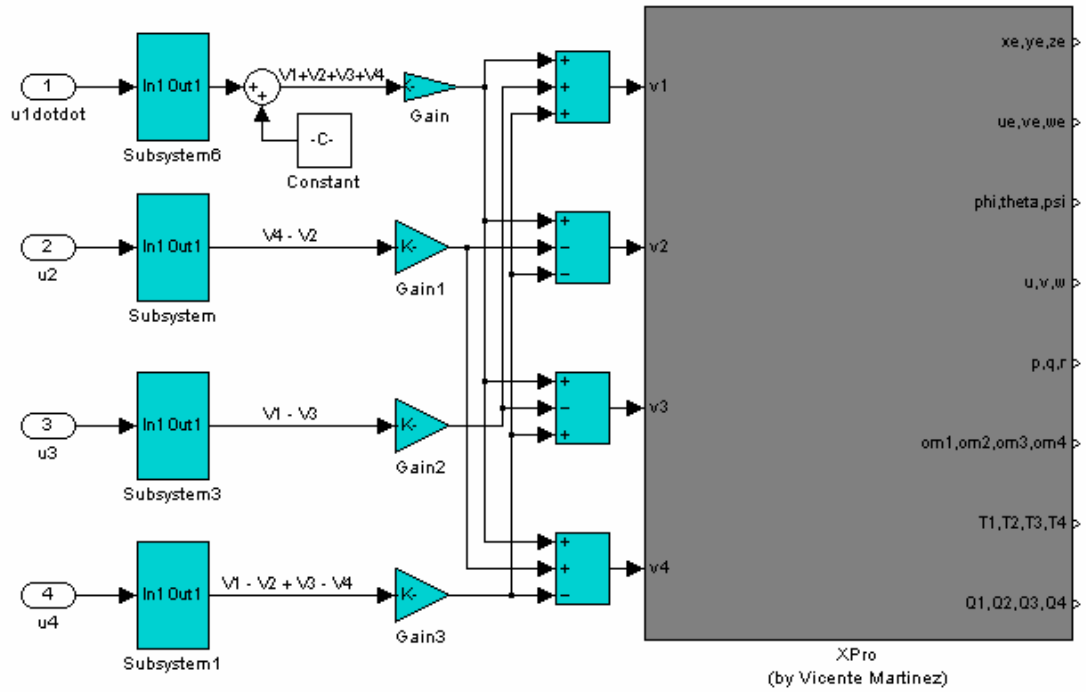


Figure 3.13: Simulink model including rotor and vehicle dynamics (mathematical approach)

3.2. Decoupling the inputs

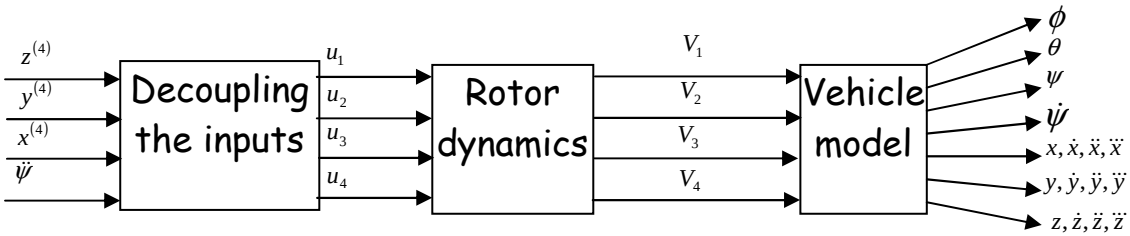


Figure 3.14: location and role of the decoupling block in the mathematical approach

The aim of this step is once again to change the input variables in order to make the control easier. As we want to control x, y, z and psi, we are going to work out the linearised relations between the derivatives of x, y, z, psi and u1, u2, u3, u4.

3.2.1. Coupling between x, y, z and phi, theta, u1

From the literature review,

$$\ddot{x} = -\frac{s_\theta c_\phi}{m} u_1 , \ \ddot{y} = \frac{s_\phi}{m} u_1 , \ \ddot{z} = -\frac{c_\theta c_\phi}{m} u_1 + g$$

(where x, y and z are the translational positions)

After having linearised the above equations around $\phi_0, \theta_0, u_{1\_0} = \frac{m(g - \ddot{z}_0)}{c_{\theta_0} c_{\phi_0}}$, we obtain:

$$\ddot{x} = \ddot{x}_0 - \frac{s_{\theta_0} c_{\phi_0}}{m}(u_1 - u_{1\_0}) - \frac{c_{\theta_0} c_{\phi_0}}{m} u_{1\_0}(\theta - \theta_0) + \frac{s_{\theta_0} s_{\phi_0}}{m} u_{1\_0}(\phi - \phi_0)$$

$$\ddot{y} = \ddot{y}_0 + \frac{s_{\phi_0}}{m}(u_1 - u_{1\_0}) + \frac{c_{\phi_0}}{m} u_{1\_0}(\phi - \phi_0)$$

$$\ddot{z} = \ddot{z}_0 - \frac{c_{\theta_0} c_{\phi_0}}{m}(u_1 - u_{1\_0}) + \frac{s_{\theta_0} c_{\phi_0}}{m} u_{1\_0}(\theta - \theta_0) + \frac{c_{\theta_0} s_{\phi_0}}{m} u_{1\_0}(\phi - \phi_0)$$

If we differentiate twice the previous set of equations, we get:

$$x^{(4)} = -\frac{s_{\theta_0} c_{\phi_0}}{m} \ddot{u}_1 - \frac{c_{\theta_0} c_{\phi_0}}{m} u_{1\_0} \ddot{\theta} + \frac{s_{\theta_0} s_{\phi_0}}{m} u_{1\_0} \ddot{\phi}$$

$$y^{(4)} = +\frac{s_{\phi_0}}{m} \ddot{u}_1 + \frac{c_{\phi_0}}{m} u_{1\_0} \ddot{\phi}$$

$$z^{(4)} = -\frac{c_{\theta_0} c_{\phi_0}}{m} \ddot{u}_1 + \frac{s_{\theta_0} c_{\phi_0}}{m} u_{1\_0} \ddot{\theta} + \frac{c_{\theta_0} s_{\phi_0}}{m} u_{1\_0} \ddot{\phi}$$

And this new set of equations enables the famous input variables (u1, u2, u3, u4) and the variables we want to control (x, y, z, psi) to be related.

3.2.2. Coupling between phi, theta, psi and u2, u3, u4

Indeed, from the literature review,

$$\ddot{\phi} = -\dot{\psi}\dot{\theta} * c_{\phi} + \frac{c_{\psi}}{I_{xx}} u_2 - \frac{s_{\psi}}{I_{yy}} u_3 + \frac{I_{yy} - I_{zz}}{I_{xx}} \left(\dot{\psi} - \dot{\theta} * s_{\phi}\right) \dot{\theta} * c_{\phi}$$

$$\ddot{\theta} = \frac{\dot{\psi}\dot{\phi}}{c_{\phi}} + \dot{\phi}\dot{\theta} * t_{\phi} + \frac{s_{\psi}}{c_{\phi} I_{xx}} u_2 + \frac{c_{\psi}}{c_{\phi} I_{yy}} u_3 - \frac{I_{yy} - I_{zz}}{I_{xx}} \left(\dot{\psi} - \dot{\theta} * s_{\phi}\right) \frac{\dot{\phi}}{c_{\phi}}$$

$$\ddot{\psi} = \dot{\phi}\dot{\psi} * t_{\phi} + \frac{\dot{\phi}\dot{\theta}}{c_{\phi}} + \frac{s_{\psi} t_{\phi}}{I_{xx}} u_2 + \frac{c_{\psi} t_{\phi}}{I_{yy}} u_3 + \frac{1}{I_{zz}} u_4 - \frac{I_{yy} - I_{zz}}{I_{xx}} \left(\dot{\psi} - \dot{\theta} * s_{\phi}\right) \dot{\phi} * t_{\phi}$$

The linearization of the above equations around $\phi_0, \psi_0, u_{2\_0} = u_{3\_0} = u_{4\_0} = 0\,\text{N.m}$ gives:

$$\ddot{\phi} = \frac{c_{\psi_0}}{I_{xx}} u_2 - \frac{s_{\psi_0}}{I_{yy}} u_3$$

$$\ddot{\theta} = \frac{s_{\psi_0}}{c_{\phi_0} I_{xx}} u_2 + \frac{c_{\psi_0}}{c_{\phi_0} I_{yy}} u_3$$

$$\ddot{\psi} = \frac{s_{\psi_0} t_{\phi_0}}{I_{xx}} u_2 + \frac{c_{\psi_0} t_{\phi_0}}{I_{yy}} u_3 + \frac{1}{I_{zz}} u_4$$

3.2.3. Combining the two couplings

Therefore, combining the two sets of equations, we obtain the key equations that relate u1, u2, u3, u4 and x, y, z, psi. The matrix form is:

$$\begin{bmatrix} z^{(4)} \\ y^{(4)} \\ x^{(4)} \\ \ddot{\psi} \end{bmatrix} = T * \begin{bmatrix} \ddot{u}_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

$$T = \begin{bmatrix} -\frac{c_{\theta_0} c_{\phi_0}}{m} & \frac{u_{1\_0}}{mI_{xx}}\left(\frac{s_{\theta_0} c_{\phi_0} s_{\psi_0}}{c_{\phi_0}} + c_{\theta_0} s_{\phi_0} c_{\psi_0}\right) & \frac{u_{1\_0}}{mI_{yy}}\left(\frac{s_{\theta_0} c_{\phi_0} c_{\psi_0}}{c_{\phi_0}} - c_{\theta_0} s_{\phi_0} s_{\psi_0}\right) & 0 \\ \frac{s_{\phi_0}}{m} & \frac{c_{\phi_0}}{m} u_{1\_0} \frac{c_{\psi_0}}{I_{xx}} & -\frac{c_{\phi_0}}{m} u_{1\_0} \frac{s_{\psi_0}}{I_{yy}} & 0 \\ -\frac{s_{\theta_0} c_{\phi_0}}{m} & \frac{u_{1\_0}}{mI_{xx}}\left(s_{\theta_0} s_{\phi_0} c_{\psi_0} - \frac{c_{\theta_0} c_{\phi_0} s_{\psi_0}}{c_{\phi_0}}\right) & -\frac{u_{1\_0}}{mI_{yy}}\left(\frac{c_{\theta_0} c_{\phi_0} c_{\psi_0}}{c_{\phi_0}} + s_{\theta_0} s_{\phi_0} s_{\psi_0}\right) & 0 \\ 0 & \frac{s_{\psi_0} t_{\phi_0}}{I_{xx}} & \frac{c_{\psi_0} t_{\phi_0}}{I_{yy}} & \frac{1}{I_{zz}} \end{bmatrix}$$

Note that these relations come from linearizations around the following flight conditions:

- ✓ ϕ_0, θ_0, ψ_0
- ✓ $\dot{\phi}_0 = \dot{\theta}_0 = \dot{\psi}_0 = 0$ rad/sec
- ✓ $u_{1\_0} = \frac{m(g - \ddot{z}_0)}{c_{\theta_0} c_{\phi_0}}$ N and $u_{2\_0} = u_{3\_0} = u_{4\_0} = 0$ N.m

The knowledge of $\phi_0, \theta_0, \psi_0, \ddot{z}_0$ at each step time is therefore enough to transform the inputs u1, u2, u3, u4 into derivatives of x, y, z and psi. These last variables are now considered as the inputs of the system.

For the implementation in Simulink, we need to consider the inverse of the transformation matrix:

$$\begin{bmatrix} \ddot{u}_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = T^{-1} * \begin{bmatrix} z^{(4)} \\ y^{(4)} \\ x^{(4)} \\ \ddot{\psi} \end{bmatrix}$$

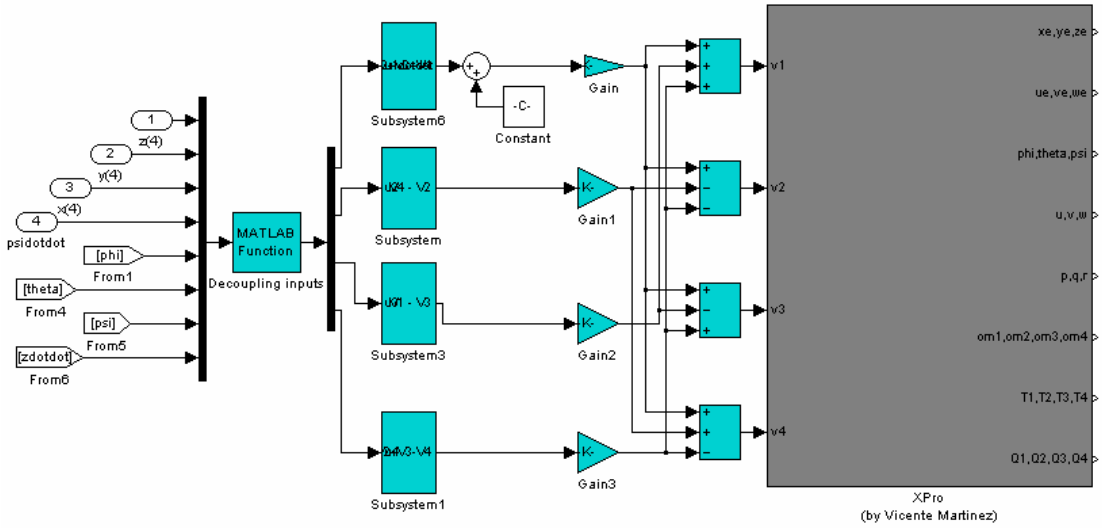


Figure 3.15: Simulink model of the ensemble {decoupling block, rotor dynamics, X pro} (mathematical approach)

The Matlab function used to decouple the inputs is given in the appendix 1.1.

3.3. Designing the control law

With this mathematical approach, only a PID controller has been designed. Here are the state vector and the input vector:

$$X = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \ddot{x} & \ddot{y} & \ddot{z} & \dddot{x} & \dddot{y} & \dddot{z} & \psi & \dot{\psi} \end{bmatrix}^T$$
$$U = \begin{bmatrix} z^{(4)} & y^{(4)} & x^{(4)} & \ddot{\psi} \end{bmatrix}^T$$

All the state variables need to be fed back in order to get a stable system. Even though we don't have sensors to measure $\ddot{x}, \ddot{y}, \ddot{z}$, it's always possible to estimate these variables with a Kalman filter.

The control law is firstly designed using the root locus method. Then, we implement the control law in the linear model and check the simulation results. Finally, we apply the designed feedback loop to the full non linear model to check if the results are still satisfactory.

3.3.1. Design of the PID controller

The system we have to control is:

$$\dot{X} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} X + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} U$$

$$Y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} X$$

The root loci used to tune the different gains are in the appendix 1.2.

Here is a table summing up the chosen gains:

Variables controlled / Variables fed back	Control of x	Control of y	Control of z	Control of psi
$\dddot{x}$ feedback	10			
$\ddot{x}$ feedback	25			
$\dot{x}$ feedback	24.07			
x feedback	8.07			
$\dddot{y}$ feedback		10		
$\ddot{y}$ feedback		25		
$\dot{y}$ feedback		24.07		
y feedback		8.07		
$\dddot{z}$ feedback			10	
$\ddot{z}$ feedback			25	
$\dot{z}$ feedback			24.07	
z feedback			8.07	
$\dot{\psi}$ feedback				10
ψ feedback				25

The closed loop system designed with these feedback gains presents the following dynamic features:

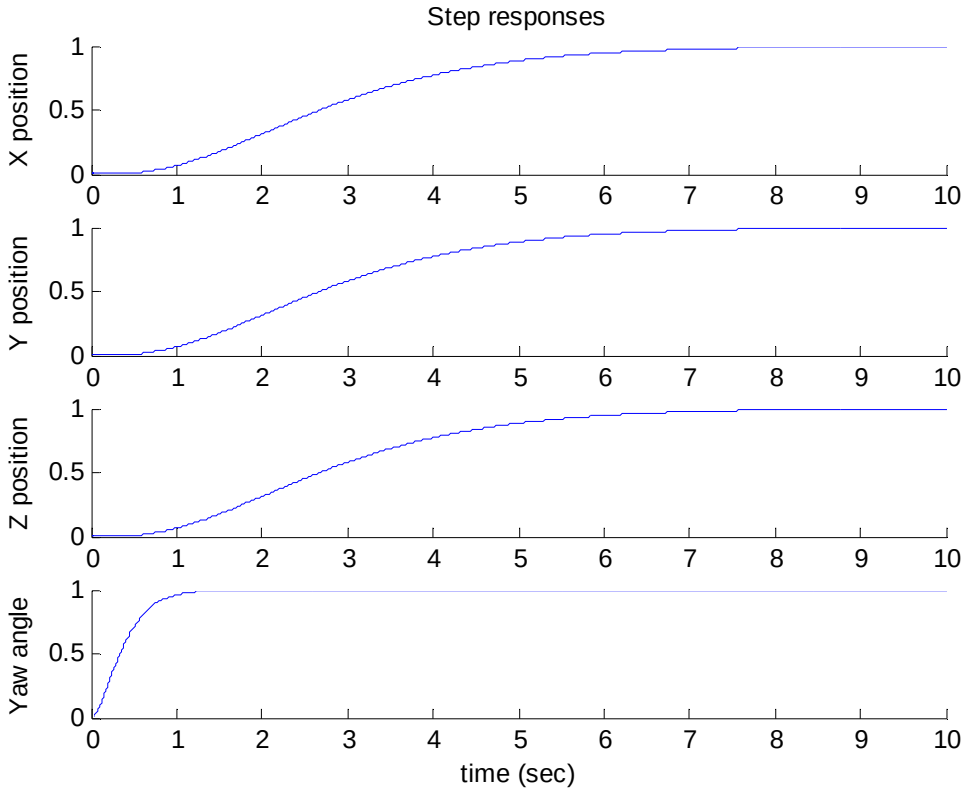


Figure 3.16 : dynamic features of the closed loop system designed in Matlab with the mathematical approach

The Matlab code used to create the closed loop system is in the appendix 1.3.

3.3.2. Simulation with the linear model

The first task in this part is to build the linear model. Bearing in mind that:

$$\frac{u_1}{V_1+V_2+V_3+V_4} = \frac{2.105s-0.0425}{0.4895s^2+1.873s+1} \text{ N/V}$$

$$\frac{u_2}{V_4-V_2} = \frac{u_3}{V_1-V_3} = l\frac{1.155s}{0.8696s^2+0.4018s+1} \text{ N.m/V}$$

$$\frac{u_4}{V_1-V_2+V_3-V_4} = \frac{0.045}{0.314s+1} \text{ N.m/V}$$

We choose to consider the following inputs:

$$U = \begin{bmatrix} 2.105\left(\dot{V}_1 + \dot{V}_2 + \dot{V}_3 + \dot{V}_4\right) - 0.0425\left(V_1 + V_2 + V_3 + V_4\right) \\ 1.155 * l\left(\dot{V}_4 - \dot{V}_2\right) \\ 1.155 * l\left(\dot{V}_1 - \dot{V}_3\right) \\ 0.045\left(V_1 - V_2 + V_3 - V_4\right) \end{bmatrix}$$

Hence the state vector is:

$$X = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \ddot{x} & \ddot{y} & \ddot{z} & \dddot{x} & \dddot{y} & \dddot{z} & \psi & \dot{\psi} \\ & & & & & & & & & & & & & \\ & & & & & & & & & & & & u_1 & u_2 & u_3 & u_4 & \dot{u}_1 & \dot{u}_2 & \dot{u}_3 & \phi & \theta & \dot{\phi} & \dot{\theta} \end{bmatrix}^T$$

Note that this state vector is not minimal but, as we need to know the values of phi and theta at each step time, it's better to include these variables in the state vector.

The output vector is: $Y = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \ddot{x} & \ddot{y} & \ddot{z} & \dddot{x} & \dddot{y} & \dddot{z} & \psi & \dot{\psi} & \phi & \theta \end{bmatrix}^T$

We have:

$$\begin{bmatrix} \ddot{u}_1 \\ \ddot{u}_2 \\ \ddot{u}_3 \\ \dot{u}_4 \end{bmatrix} = A_u * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \end{bmatrix} + B_u * U$$

with:

$$A_u = \begin{bmatrix} -\frac{1}{0.4895} & 0 & 0 & 0 & -\frac{1.873}{0.4895} & 0 & 0 \\ 0 & -\frac{1}{0.8696} & 0 & 0 & 0 & -\frac{0.4018}{0.8696} & 0 \\ 0 & 0 & -\frac{1}{0.8696} & 0 & 0 & 0 & -\frac{0.4018}{0.8696} \\ 0 & 0 & 0 & -\frac{1}{0.314} & 0 & 0 & 0 \end{bmatrix}$$

$$B_u = \begin{bmatrix} \frac{1}{0.4895} & 0 & 0 & 0 \\ 0 & \frac{1}{0.8696} & 0 & 0 \\ 0 & 0 & \frac{1}{0.8696} & 0 \\ 0 & 0 & 0 & \frac{1}{0.314} \end{bmatrix}$$

And,

$$\begin{bmatrix} z^{(4)} \\ y^{(4)} \\ x^{(4)} \\ \ddot{\psi} \end{bmatrix} = A_x * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \dot{u}_1 \end{bmatrix} + B_x * U$$

with:

$$A_x = \begin{bmatrix} \frac{c_{\theta_0} c_{\phi_0}}{0.4895m} & \frac{u_{1\_0}}{mI_{xx}} \left(\frac{s_{\theta_0} c_{\phi_0} s_{\psi_0}}{c_{\phi_0}} + c_{\theta_0} s_{\phi_0} c_{\psi_0} \right) & \frac{u_{1\_0}}{mI_{yy}} \left(\frac{s_{\theta_0} c_{\phi_0} c_{\psi_0}}{c_{\phi_0}} - c_{\theta_0} s_{\phi_0} s_{\psi_0} \right) & 0 & \frac{1.873 c_{\theta_0} c_{\phi_0}}{0.4895m} \\ -\frac{s_{\phi_0}}{0.4895m} & \frac{c_{\phi_0}}{m} u_{1\_0} \frac{c_{\psi_0}}{I_{xx}} & -\frac{c_{\phi_0}}{m} u_{1\_0} \frac{s_{\psi_0}}{I_{yy}} & 0 & -\frac{1.873 s_{\phi_0}}{0.4895m} \\ \frac{s_{\theta_0} c_{\phi_0}}{0.4895m} & \frac{u_{1\_0}}{mI_{xx}} \left(s_{\theta_0} s_{\phi_0} c_{\psi_0} - \frac{c_{\theta_0} c_{\phi_0} s_{\psi_0}}{c_{\phi_0}} \right) & -\frac{u_{1\_0}}{mI_{yy}} \left(\frac{c_{\theta_0} c_{\phi_0} c_{\psi_0}}{c_{\phi_0}} + s_{\theta_0} s_{\phi_0} s_{\psi_0} \right) & 0 & \frac{1.873 s_{\theta_0} c_{\phi_0}}{0.4895m} \\ 0 & \frac{s_{\psi_0} t_{\phi_0}}{I_{xx}} & \frac{c_{\psi_0} t_{\phi_0}}{I_{yy}} & \frac{1}{I_{zz}} & 0 \end{bmatrix}$$

$$B_x = \begin{bmatrix} -\frac{c_{\theta_0} c_{\phi_0}}{0.4895m} & 0 & 0 & 0 \\ \frac{s_{\phi_0}}{0.4895m} & 0 & 0 & 0 \\ -\frac{s_{\theta_0} c_{\phi_0}}{0.4895m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Thus, if we consider the full state vector, we just need to concatenate A_u with A_x and B_u with B_x.

As the system is not a linear time invariant system, we cannot use the LTI state space Simulink block. Indeed, we want to update the matrices A and B at each step time taking into account the new values of phi, theta, psi and vertical acceleration.

Thus, the only solution is to create the state space system “by hand”:

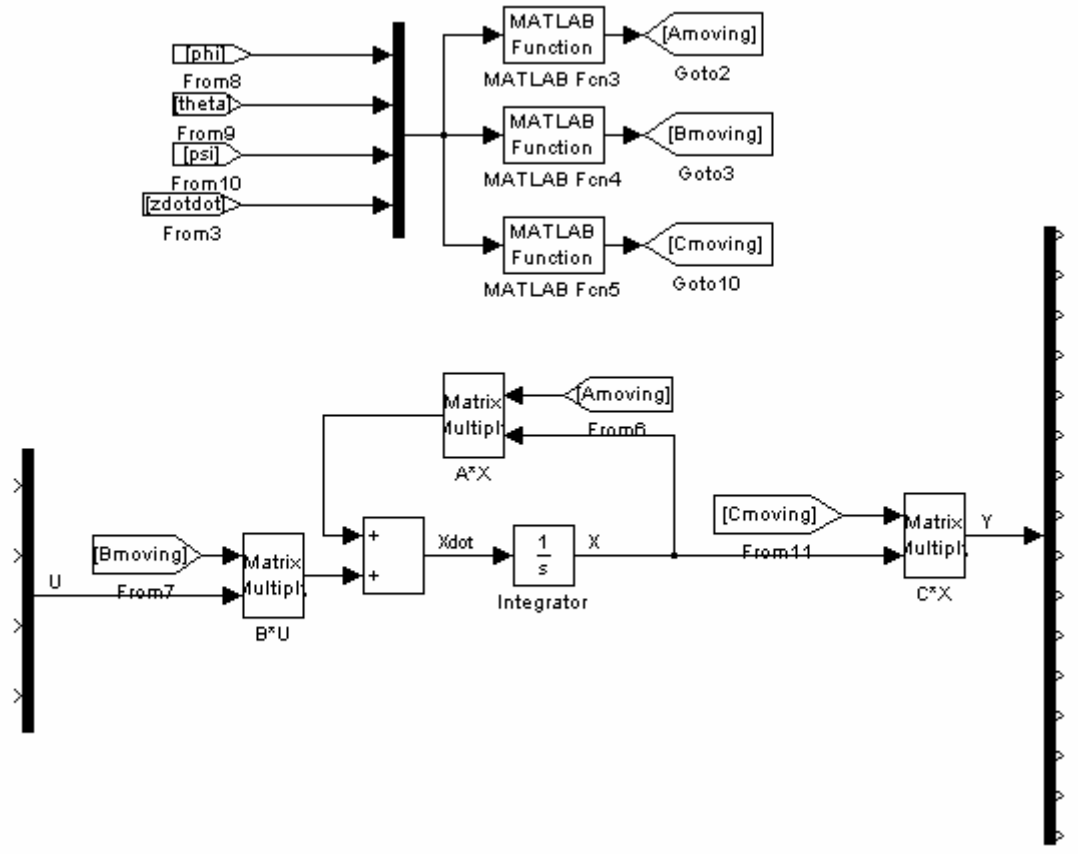


Figure 3.17 : linear Simulink model of the quadrotor

Details of the Matlab functions used to generate A, B and C are given in the appendix 1.4.

Then, we add the rotor dynamics subsystems we studied previously. As the inputs we consider in this model are not the same, the rotor dynamics blocks are simplified compare to those detailed in the rotor dynamics paragraph.
The function used to decouple the inputs remains exactly the same (see appendix 1.1).

Finally, we just need to apply the control law we designed with the root loci.

We obtain the following model:

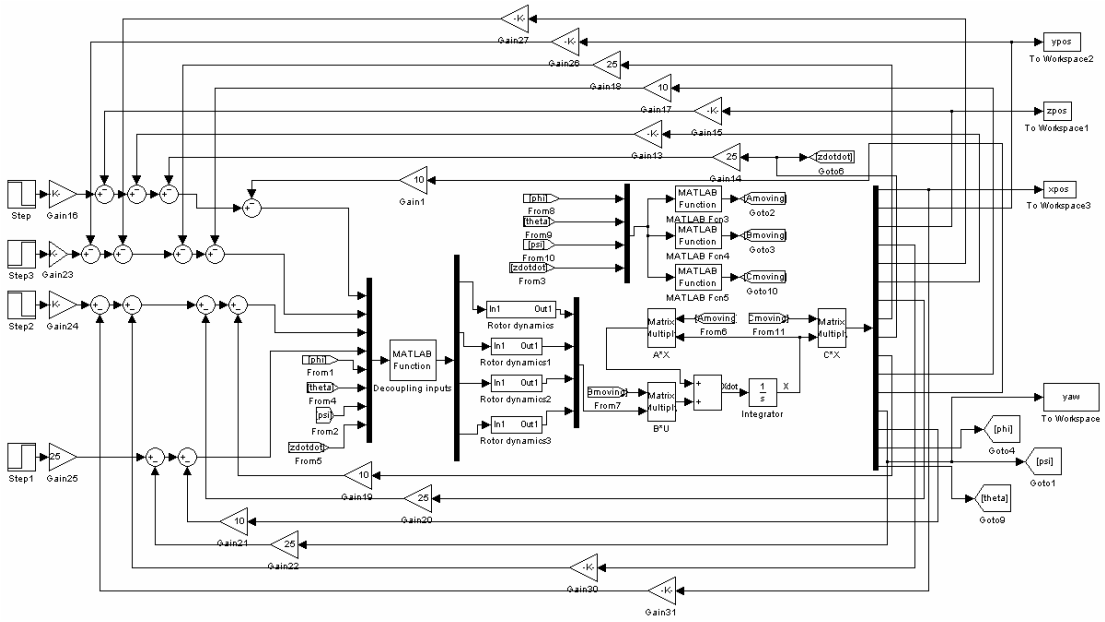


Figure 3.18 : Simulink closed loop system using the linear model of the quadrotor (mathematical approach)

3.3.3. Results of the simulation

As we can notice on the following figure, the simulation gives good results. The settling time has slightly increased due to some oscillations around the equilibrium. The closed loop system is stable. The autopilot function works well without any steady state error.

Indeed, if we apply the following step values to the inputs:

- ✓ x=-3 m
- ✓ y=4 m
- ✓ z=-5 m
- ✓ psi=0.2 rad

we get the following responses:

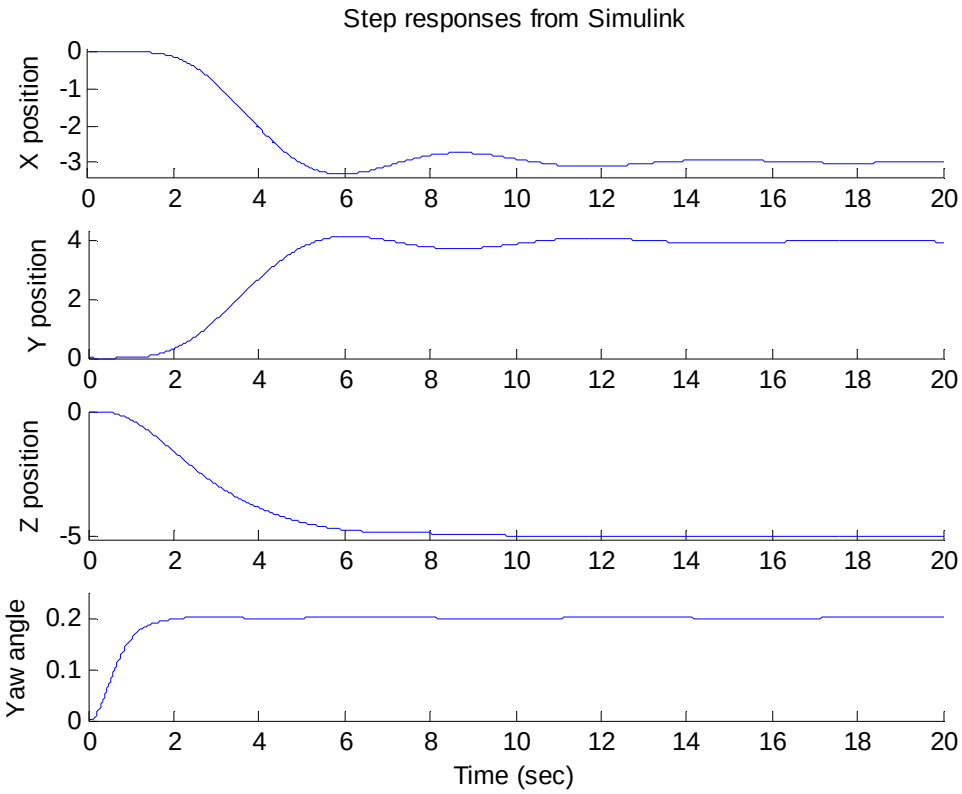


Figure 3.19 : dynamic features of the Simulink closed loop system using the linear model of the quadrotor (mathematical approach)

The small oscillations can be explained by the solver used for the simulation (ode1: Euler method) and the sample time (0.01 sec) which reduce the accuracy of the calculations. This solver cannot differentiate signals very precisely for example.
In the rotor dynamics subsystems, there are some derivative blocks and these blocks can affect significantly the accuracy.

However, with a settling time equalled to 15 seconds, the PID controller gives satisfactory results with the linear model.

3.4. Performances on the full non linear model

Now, the effects of the PID controller need to be investigated on the non linear model. If the results are still satisfactory, we will be able to validate the linearization process undertaken.

3.4.1. Applying the previous control law

The simulation is run with the ensemble {vehicle's model + rotor dynamics + decoupling inputs} on which we add the feedback loop with the previous gain values. Unfortunately, the previous gain values didn't give good results. The altitude and yaw angle control remained satisfactory but the lateral components of the vehicle's position diverged because of the size of signals.

This can be explained by:

- ✓ the rotor dynamics modelling which wasn't very precise for rolling and pitching moments
- ✓ the rotor dynamics modelling which has been carried out around the hover with a small step amplitude (0.001 V) for rolling and pitching moments

3.4.2. Re-designing the control law

Thus, the control law needed to be reviewed with smaller gains using root loci method. The associated root loci are not given because they are exactly the same as those given in the appendix 1.2 except that they are translated along the real axis towards the origin. The poles are still left half plane but closer to the origin.

The chosen gain values are gathered in the following table:

Variables controlled / Variables fed back	Control of x	Control of y	Control of z	Control of psi
$\dddot{x}$ feedback	0.2			
$\ddot{x}$ feedback	0.01			
$\dot{x}$ feedback	1.925*10^(-4)			
x feedback	1.29*10^(-6)			
$\dddot{y}$ feedback		0.2		
$\ddot{y}$ feedback		0.01		
$\dot{y}$ feedback		1.925*10^(-4)		
y feedback		1.29*10^(-6)		
$\dddot{z}$ feedback			5	
$\ddot{z}$ feedback			6.25	
$\dot{z}$ feedback			3.0071	
z feedback			0.504	
$\dot{\psi}$ feedback				0.2
ψ feedback				0.01

This design has been carried out by investigating firstly the control of the lateral components. Indeed, the lateral behaviour is the hardest to control and requires slow dynamic features. The lateral control is therefore the limiting factor.

Once these components are controlled, the aim is to get the same dynamic features for altitude tracking.

The problem encountered is that the altitude input requires large signal variations to be effective. The smallest gain values for which the altitude control was still good are those written in the above table.

As for the control of psi, its dynamic features have been slowed down in order not to have any bad influence on the position control.

3.4.3. Implementation in Simulink

Here is the Simulink model used for the simulation:

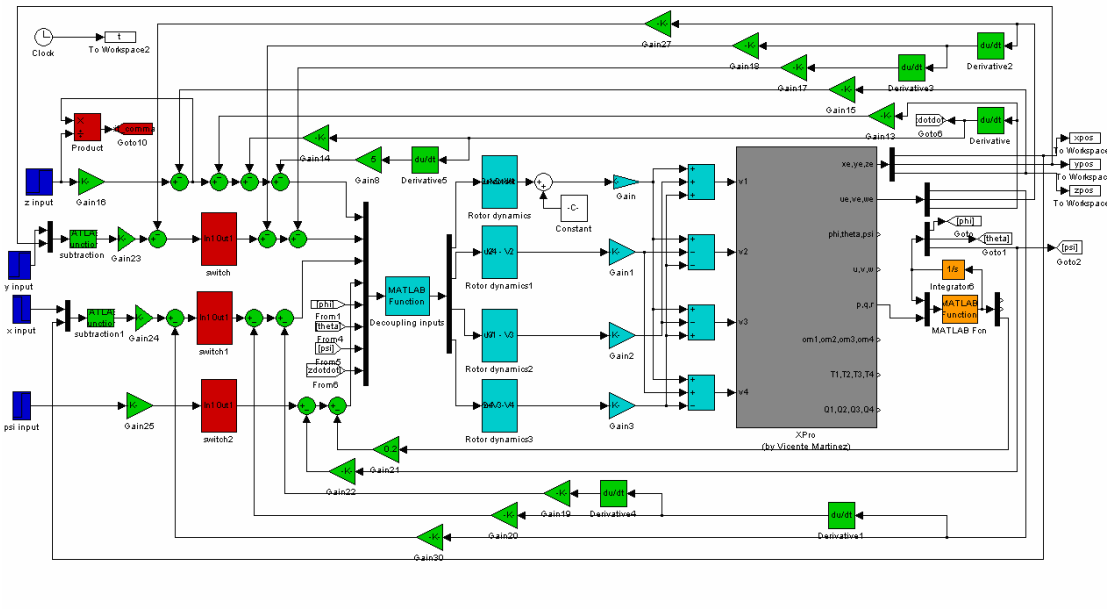


Figure 3.20 : Simulink closed loop system including the full non linear model (mathematical approach)

As the vertical dynamic behaviour is faster than the lateral one, switches are added on the lateral feed forward path (red blocks). Indeed, the signals on the vertical path are too large compare to those on the lateral path. Thus, the lateral control is badly influenced by these large variations of vertical thrust. We need to wait for the vertical motion to be finished to tackle the lateral motion.

For the same reasons, the control of the yaw angle can only begin after the desired altitude is reached.

This constraint is respected thanks to the three red blocks.

The green blocks correspond to the control law. Note that the first subtractions on both x and y paths have been carried out using a Matlab function instead of the classical sum block from Simulink because of precision issues.

The orange blocks enable the Euler angles to be determined taking into account that the order of rotation must be different from the usual one (final rotation along the thrust direction). The Euler angles suggested in the outputs of the vehicle's model use the common convention. Therefore, these angles need to be calculated again using the body rates (p, q, r) and the appropriate transformation matrix. The rates of change of

Euler angles are then integrated to give the desired Euler angles. Details of the Matlab function used to do this transformation are given in the appendix 1.5.

3.4.4. Results from simulations

As expected with the pole locations on the root loci, we obtain a very slow dynamic behaviour:

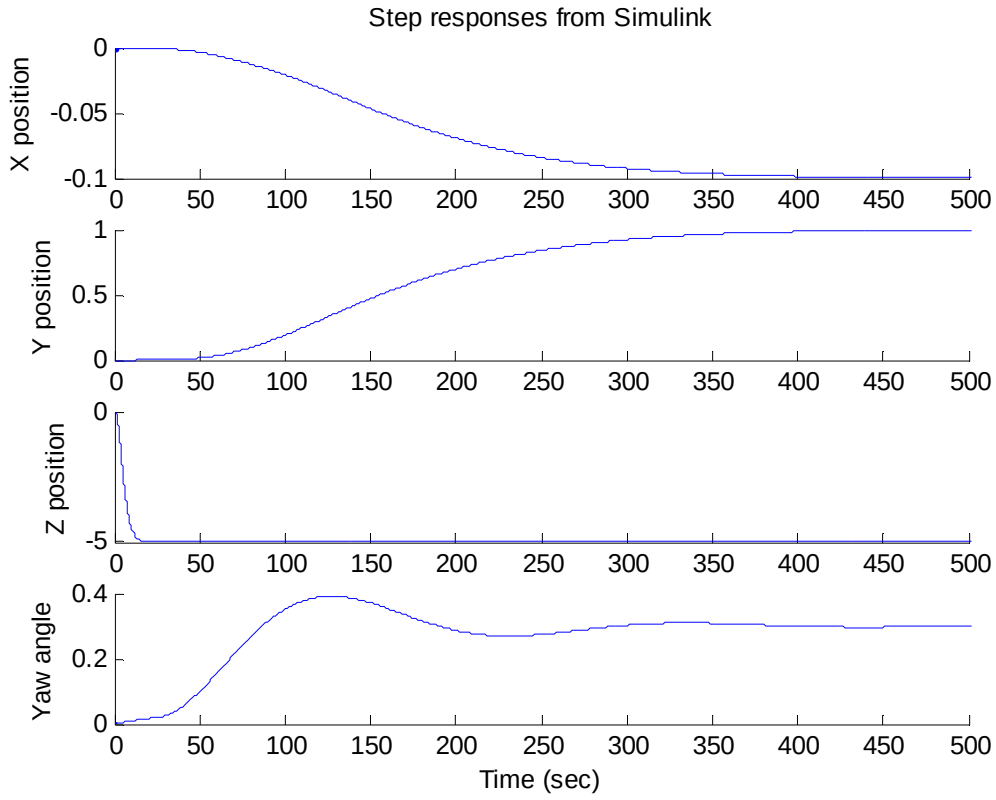


Figure 3.21 : dynamic features of the Simulink closed loop system using the full non linear model of the quadrotor (mathematical approach)

Beneficially there is no steady state error. Even if the UAV requires a long time to reach the desired position (400 sec ~ 7 min), it finally achieves its task and hovers at the final position.

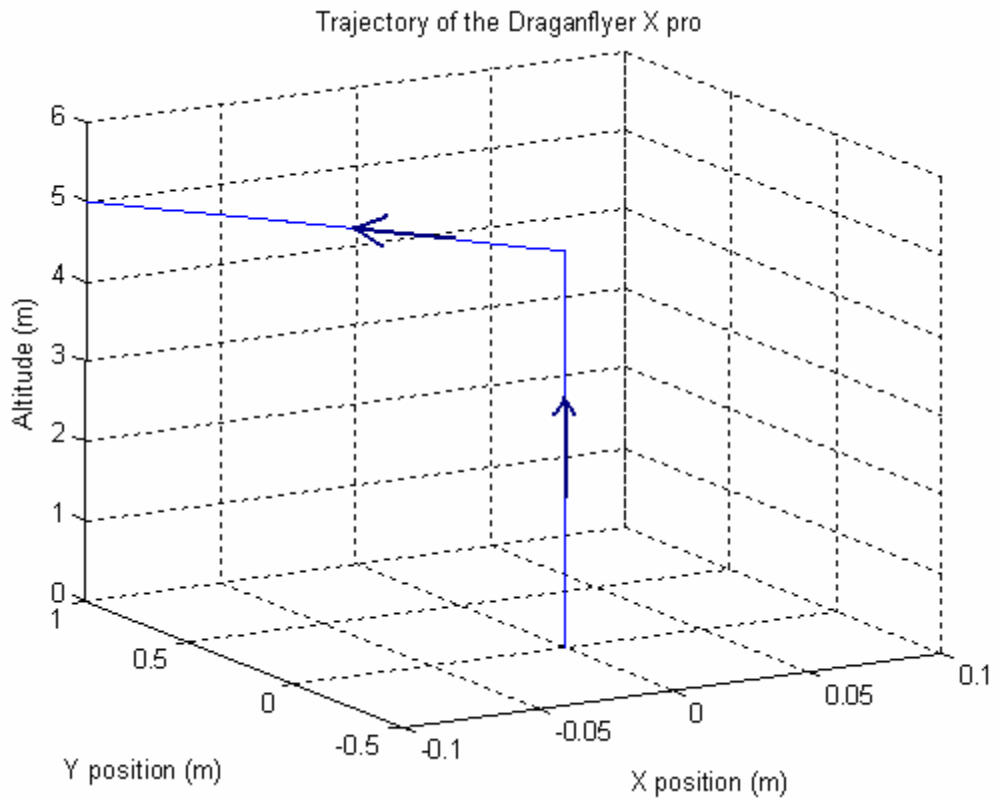


Figure 3.22 : trajectory of the quadrotor simulated with the full non linear model (mathematical approach)

This figure shows that the quadrotor begins moving upward and then sideward. This feature affects badly the trajectory following. We want the quadrotor to reach directly the desired coordinates.

3.5. The reasons why this approach is unacceptable

3.5.1. The flight dynamics

The fact that the vehicle reaches the desired point in two steps and needs 7 minutes to settle at its final position is not acceptable.

3.5.2. The derivative blocks

Because of noise amplification, the derivative blocks cannot be implemented in a model using data from sensors.
Even though a Kalman filter can be implemented to estimate $\ddot{x}, \ddot{y}, \ddot{z}$, feeding back variables which are not accessible through sensors makes the control too complicated. Besides, as some derivative blocks are also present in the rotor dynamics subsystem, an other Kalman filter would be required to estimate these signals.

3.5.3. The signals amplitude

The size of the signals is too small to be implemented on the Draganflyer X pro. Indeed, this model requires a very high level of precision which cannot be achieved with the real vehicle and its sensors. This model is too sensitive to the sensor bias for example.

Consequently, the control system structure needs to be reviewed.

4. Towards an engineering approach: phi and theta feedback

This approach aims to get rid of all derivatives block. Thus, instead of feeding back acceleration and derivative of acceleration, we now choose to feed back attitude and rate of change of attitude.

4.1. Modelling the rotor dynamics

The rotor dynamics model remain the same, but the derivative blocks are removed. In the previous approach, the rotor dynamics subsystem enabled the transformation from voltages to u1, u2, u3 and u4. From now on, this block carries out the following input transformation:

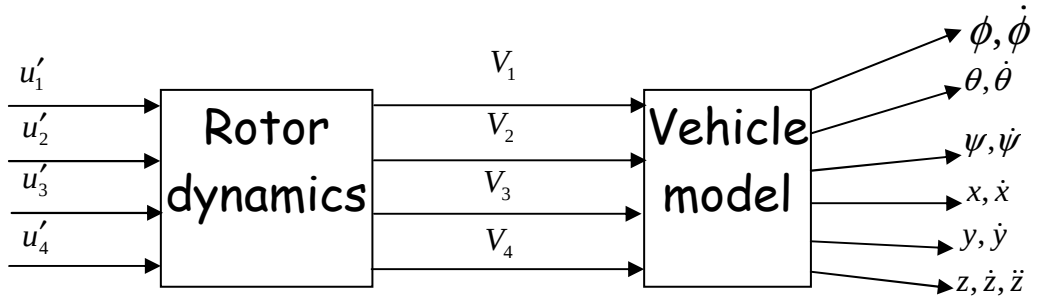


Figure 4.1 : location and role of the rotor dynamics block (Towards an engineering approach)

With:

$$u_1' = 2.105\left(\dot{V}_1 + \dot{V}_2 + \dot{V}_3 + \dot{V}_4\right) - 0.0425\left(V_1 + V_2 + V_3 + V_4\right)$$
$$u_2' = 1.155 * l\left(\dot{V}_4 - \dot{V}_2\right)$$
$$u_3' = 1.155 * l\left(\dot{V}_1 - \dot{V}_3\right)$$
$$u_4' = 0.045 * \left(V_1 - V_2 + V_3 - V_4\right)$$

The associated Simulink model is:

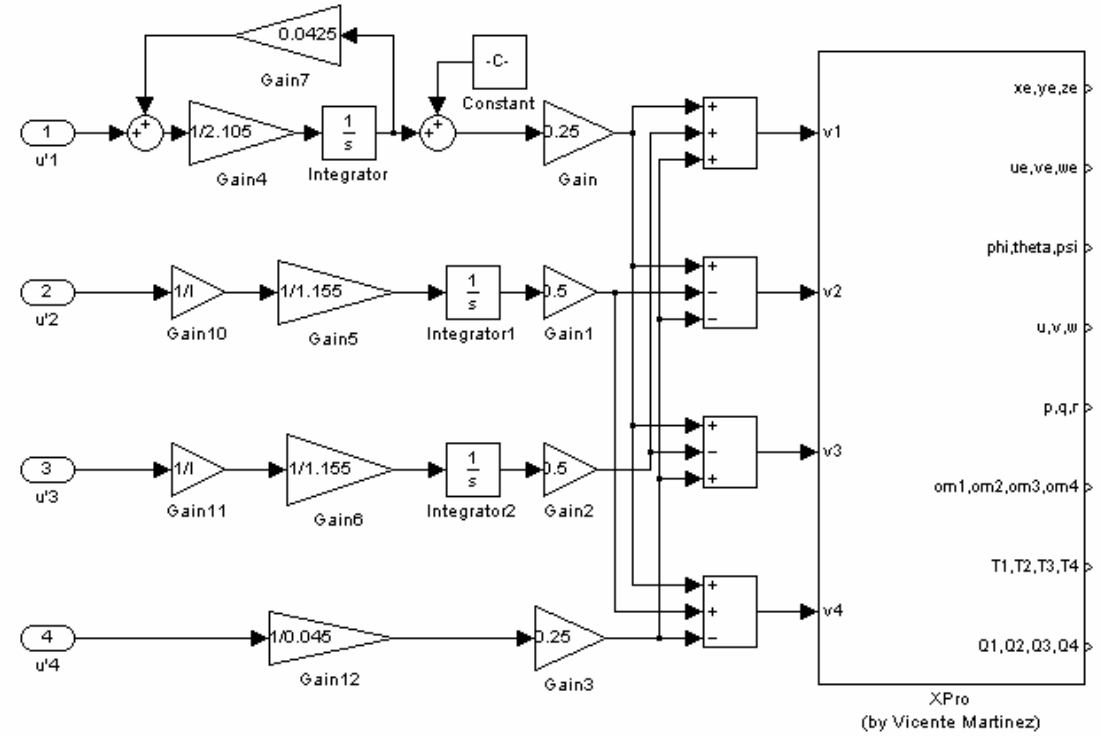


Figure 4.2 : Simulink model of the ensemble {rotor dynamics + vehicle} (Towards an engineering approach)

Thanks to the transfer functions we worked out in the previous chapter, we have:

$$u_1' = 0.4895 * \ddot{u}_1 + 1.873 * \dot{u}_1 + u_1$$
$$u_2' = 0.8696 * \ddot{u}_2 + 0.4018 * \dot{u}_2 + u_2$$
$$u_3' = 0.8696 * \ddot{u}_3 + 0.4018 * \dot{u}_3 + u_3$$
$$u_4' = 0.314 * \dot{u}_4 + u_4$$

4.2. Decoupling the inputs

After some experiments, it appears that the coupling between x, y, z and ϕ, θ, u_1 is not significant.

Hence, we consider the following set of equations:

$$\ddot{x} = -\frac{u_{1\_0}}{m}\theta = -g\theta$$

$$\ddot{y} = \frac{u_{1\_0}}{m}\phi = g\phi$$

$$\ddot{z} = -\frac{1}{m}\left(u_1 - u_{1\_0}\right)$$

Only the coupling between ϕ, θ, ψ and u_2, u_3, u_4 needs to be considered. From the previous chapter, the linearization around ϕ_0, ψ_0 had given:

$$\ddot{\phi} = \frac{c_{\psi_0}}{I_{xx}}u_2 - \frac{s_{\psi_0}}{I_{yy}}u_3$$

$$\ddot{\theta} = \frac{s_{\psi_0}}{c_{\phi_0} I_{xx}}u_2 + \frac{c_{\psi_0}}{c_{\phi_0} I_{yy}}u_3$$

$$\ddot{\psi} = \frac{s_{\psi_0} t_{\phi_0}}{I_{xx}}u_2 + \frac{c_{\psi_0} t_{\phi_0}}{I_{yy}}u_3 + \frac{1}{I_{zz}}u_4$$

These equations are approximated by:

$$u'_{2\_decoupled} = c_{\psi_0}u'_2 - \frac{I_{xx}s_{\psi_0}}{I_{yy}}u'_3 = I_{xx}\left(0.8696 * \phi^{(4)} + 0.4018 * \dddot{\phi} + \ddot{\phi}\right)$$

$$u'_{3\_decoupled} = \frac{s_{\psi_0} I_{yy}}{c_{\phi_0} I_{xx}}u'_2 + \frac{c_{\psi_0}}{c_{\phi_0}}u'_3 = I_{yy}\left(0.8696 * \theta^{(4)} + 0.4018 * \dddot{\theta} + \ddot{\theta}\right)$$

$$u'_{4\_decoupled} = \frac{I_{zz}s_{\psi_0}t_{\phi_0}}{I_{xx}}u'_2 + \frac{I_{zz}c_{\psi_0}t_{\phi_0}}{I_{yy}}u'_3 + u'_4 \approx I_{zz}\left(0.314 * \dddot{\psi} + \ddot{\psi}\right)$$

These equations give new inputs which are “in the same direction” as the Euler angles.

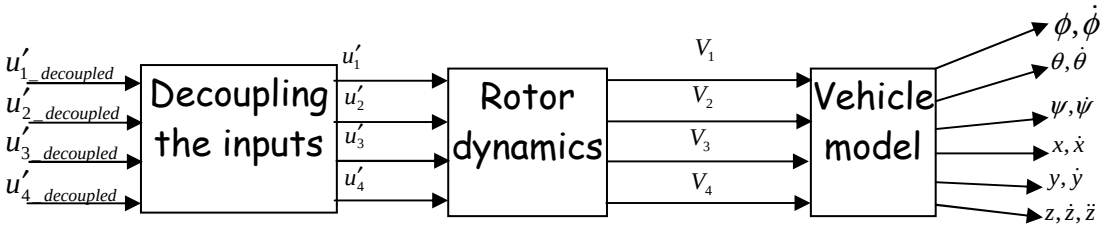


Figure 4.3 : location and role of the decoupling block (Towards an engineering approach)

In order to write the Matlab function used in the decoupling block, we need to calculate the inverse of the transformation matrix:

$$\begin{bmatrix} u'_2 \\ u'_3 \\ u'_4 \end{bmatrix} = \begin{bmatrix} c_{\psi_0} & s_{\psi_0} c_{\phi_0} \frac{I_{xx}}{I_{yy}} & 0 \\ -s_{\psi_0} \frac{I_{yy}}{I_{xx}} & c_{\psi_0} c_{\phi_0} & 0 \\ 0 & -s_{\phi_0} \frac{I_{zz}}{I_{yy}} & 1 \end{bmatrix} \begin{bmatrix} u'_{2\_decoupled} \\ u'_{3\_decoupled} \\ u'_{4\_decoupled} \end{bmatrix}$$

The Matlab function is in the appendix 2.1.

Here is the associated Simulink model:

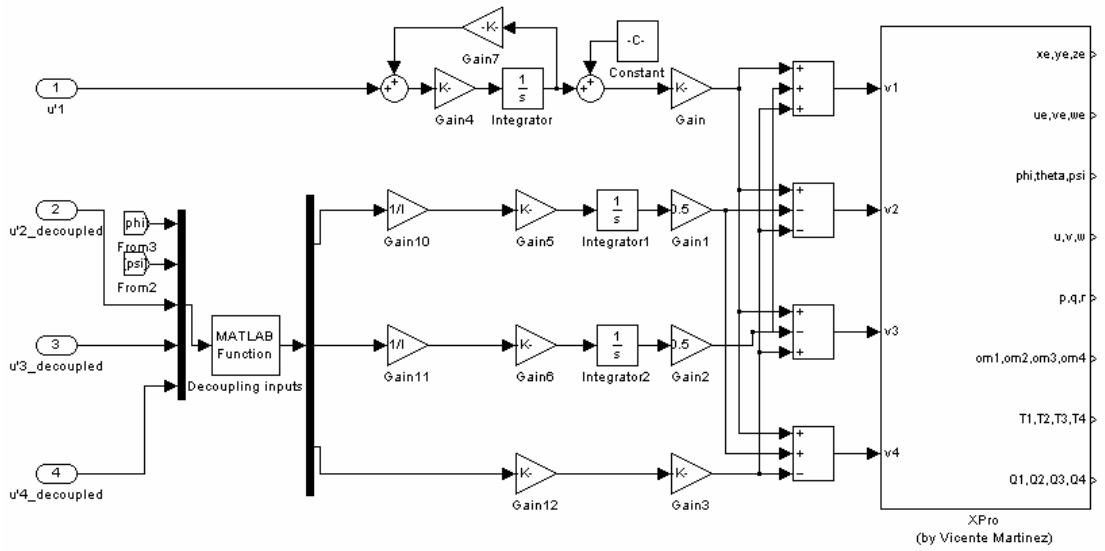


Figure 4.4 : Simulink model of the ensemble {decoupling block, rotor dynamics, Xpro} (Towards an engineering approach)

4.3. Designing the control law

Again with this approach, only a PID controller has been designed.

Inspired by the figure 2.3 (decomposition of the dynamical model into two subsystems), the design of the control law has been split into two different steps. The attitude and altitude of the quadrotor are firstly controlled in an inner loop. Then, an outer loop feeds back the lateral components of the position and the velocity to control x and y.

4.3.1. The inner loop

The state vector is:

$$X_{inner} = \begin{bmatrix} z & \dot{z} & \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi} & u_1 & u_2 & u_3 & u_4 & \dot{u}_1 & \dot{u}_2 & \dot{u}_3 \end{bmatrix}^T$$

We begin considering the following input vector:

$$U_{inner} = \begin{bmatrix} u_1' & u_2' & u_3' & u_4' \end{bmatrix}^T$$

We have

$$\dot{X}_{inner} = A_{inner} X_{inner} + B_{inner} U_{inner}$$

With

$$A_{INNER} = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{m} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\cos(\psi_0)}{I_{XX}} & -\frac{\sin(\psi_0)}{I_{YY}} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\sin(\psi_0)}{\cos(\phi_0) I_{XX}} & \frac{\cos(\psi_0)}{\cos(\phi_0) I_{YY}} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\sin(\psi_0)\tan(\phi_0)}{I_{XX}} & \frac{\cos(\psi_0)\tan(\phi_0)}{I_{YY}} & \frac{1}{I_{ZZ}} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{0.314} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{0.4895} & 0 & 0 & 0 & -\frac{1.873}{0.4895} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{0.8696} & 0 & 0 & 0 & -\frac{0.4018}{0.8696} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{0.8696} & 0 & 0 & 0 & -\frac{0.4018}{0.8696}
\end{bmatrix}$$

$$B_{INNER} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{0.314} \\ \frac{1}{0.4895} & 0 & 0 & 0 \\ 0 & \frac{1}{0.8696} & 0 & 0 \\ 0 & 0 & \frac{1}{0.8696} & 0 \end{bmatrix}$$

Now, we want to make the decoupled inputs interfere in the input vector:

$$\dot{X}_{inner} = A_{inner} X_{inner} + B_{inner} U_{inner} = A_{inner} X_{inner} + B_{inner} * T * U_{inner\_decoupled}$$

With

$$\begin{bmatrix} u'_1 \\ u'_2 \\ u'_3 \\ u'_4 \end{bmatrix} = T * \begin{bmatrix} u'_{1\_decoupled} \\ u'_{2\_decoupled} \\ u'_{3\_decoupled} \\ u'_{4\_decoupled} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\psi_0} & s_{\psi_0} c_{\phi_0} \frac{I_{xx}}{I_{yy}} & 0 \\ 0 & -s_{\psi_0} \frac{I_{yy}}{I_{xx}} & c_{\psi_0} c_{\phi_0} & 0 \\ 0 & 0 & -s_{\phi_0} \frac{I_{zz}}{I_{yy}} & 1 \end{bmatrix} \begin{bmatrix} u'_{1\_decoupled} \\ u'_{2\_decoupled} \\ u'_{3\_decoupled} \\ u'_{4\_decoupled} \end{bmatrix}$$

Thus, the new B matrix to consider for the control design is:

$$B'_{inner} = B_{inner} * T$$

The design of the PID controller is carried out around the hover using root locus method. The associated root loci are in the appendix 2.2.

Here is a table gathering the different gain values:

Variables controlled / Variables fed back	Control of ϕ	Control of θ	Control of ψ	Control of z
$\dot{\phi}$ feedback	0.0208			
ϕ feedback	0.00239			
$\dot{\theta}$ feedback		0.0208		
θ feedback		0.00238		
$\dot{\psi}$ feedback			0.3155	
ψ feedback			0.111	
$\dot{z}$ feedback				-7
$\ddot{z}$ feedback				-5.45
z feedback				-2.45

4.3.2. The outer loop

The state and input vectors are:

$$X_{outer} = [x \quad y \quad \dot{x} \quad \dot{y}]^T \qquad U_{outer} = \begin{bmatrix} \phi \\ \theta \end{bmatrix}$$

Thus, the linearization around the hover gives:

$$\dot{X}_{outer} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} X_{outer} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & -g \\ g & 0 \end{bmatrix} U_{outer}$$

The control design must take into account the dynamics of phi and theta. Thus, before investigating the root loci, this system is connected to the previous closed loop system. The associated Matlab code is in the appendix 2.3.

The control of x and y is once again carried out by investigating the different root loci. These plots are in the appendix 2.4.

Here is a table with the different gain values:

Variables controlled / Variables fed back	Control of x	Control of y
$\dot{x}$ feedback	-1.4*10^(-5)	
x feedback	-4.12*10^(-7)	
$\dot{y}$ feedback		1.412*10^(-5)
y feedback		4.17*10^(-7)

4.3.3. Expected results

The established model with both inner and outer loops closed gives rise to the following dynamic features:

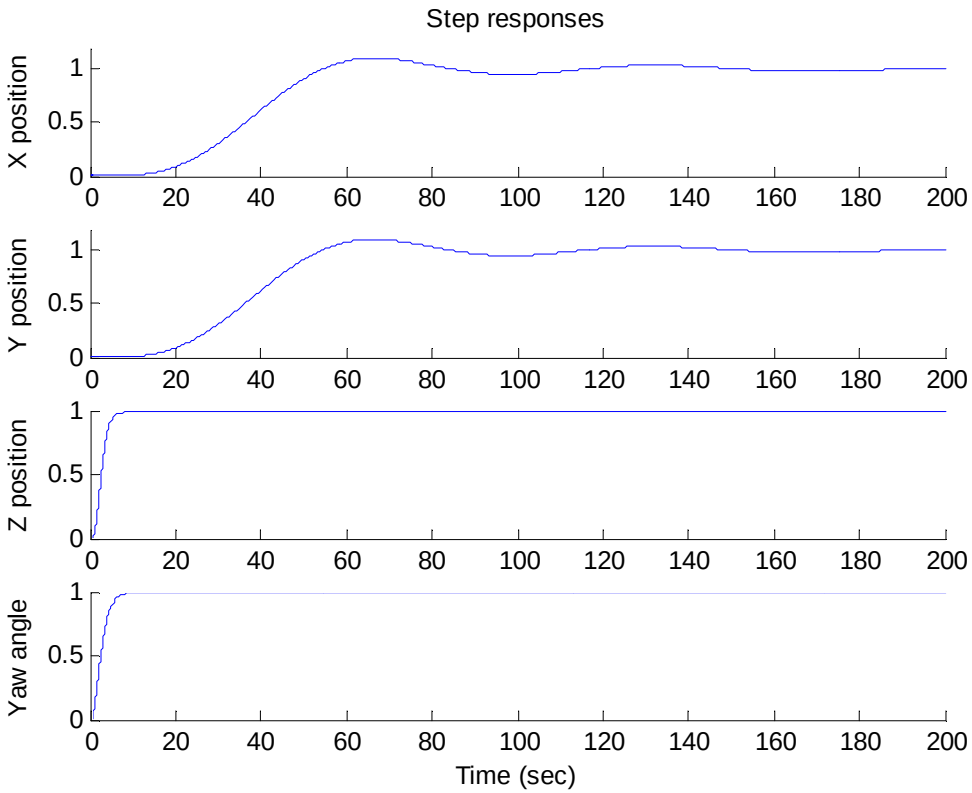


Figure 4.5 : dynamic features of the closed loop system designed in Matlab (Towards an engineering approach)

4.4. Achieved performances

4.4.1. Applying the previous control law

When we apply the previous control law to the full non linear model, the simulations don't give the expected results. In fact, the UAV is unstable and its lateral position diverges. This is certainly due to an underestimated rotor lag.

4.4.2. Review of the control structure

In order to figure this problem out, a stabiliser block is added on the first input path. The role of this block is to make the vertical thrust vary with the attitude of the quadrotor.

When the quadrotor is banked and the rate of change of banking angle make it go away from the hover attitude, that means that the control law wants the quadrotor to go in the specified direction. Thus, in order to make this command more effective, we increase the voltages sum by 0.1V.

When the quadrotor is banked and the rate of change of this banking angle make it recover the hover attitude, that means that the control law wants the quadrotor to stop moving in that direction. Again, to get a more effective command, we decrease the voltages sum by 0.1V.

Details of the associated Matlab function are in the appendix 2.5.

4.4.3. Implementation in Simulink and associated results

The Simulink model including the classical control law and the stabiliser is:

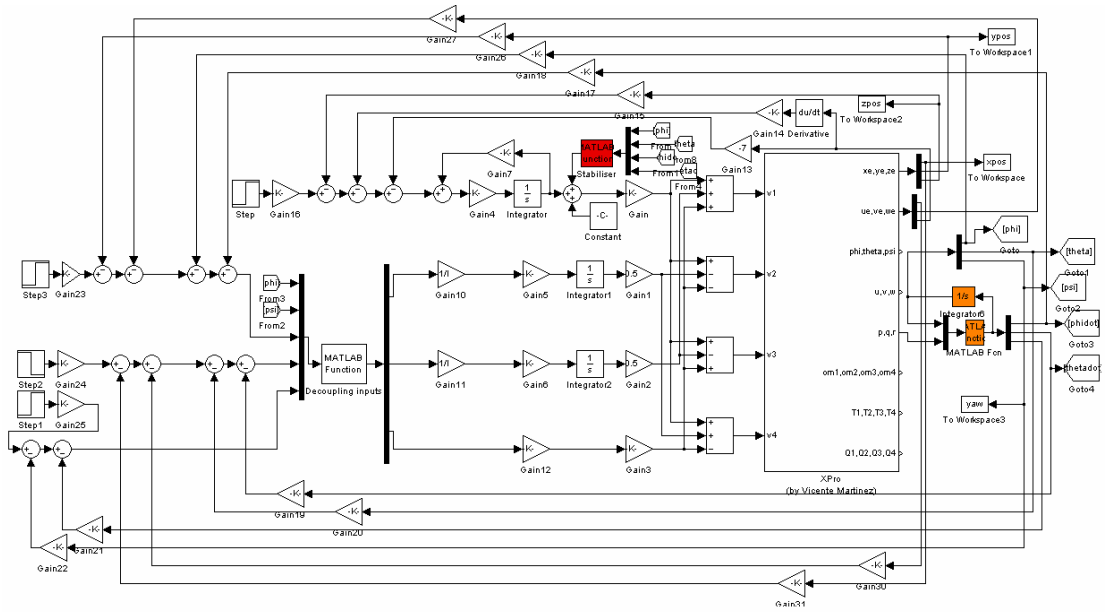


Figure 4.6 : Simulink model of the closed loop system using the full non linear model (Towards an engineering approach)

As we can see on the following plot, a phenomenon called “hunting” is happening. The quadrotor doesn’t hover at the desired position but fluctuates around the final values.

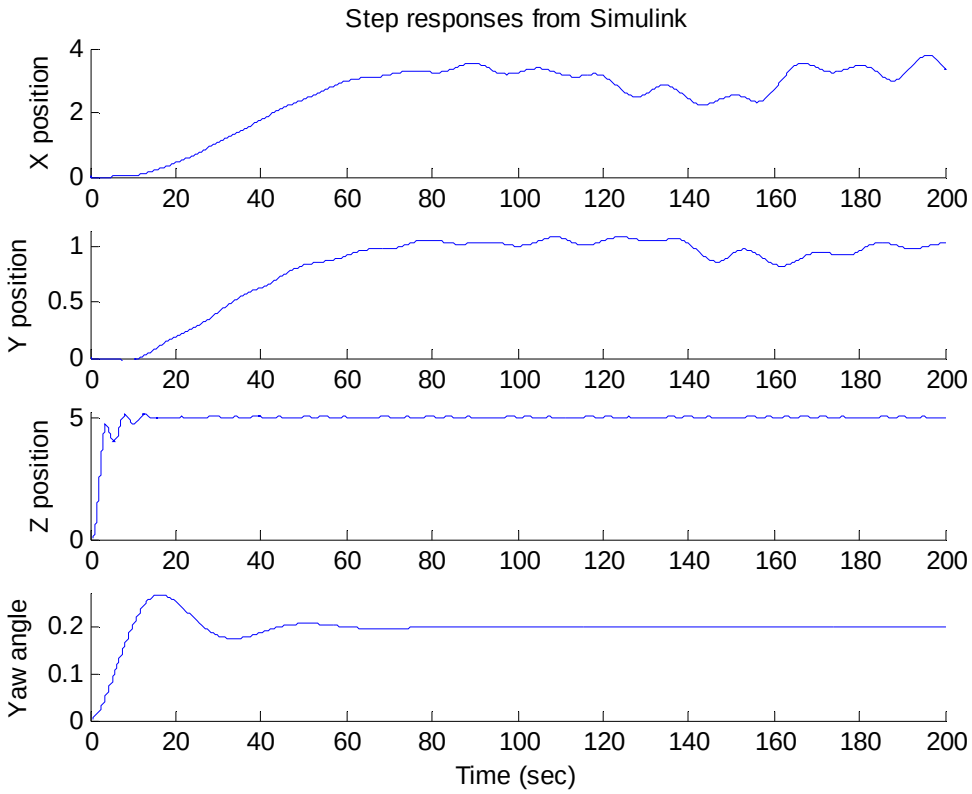


Figure 4.7 : dynamic features of the Simulink closed loop system using the full non linear model (Towards an engineering approach)

This can be explained by a very small phase margin. Indeed, the closed loop system is just stable enough to fly around the equilibrium. This is due to the rotor lag which may have been underestimated. If we go back to the rotor dynamics modelling in the previous chapter, we can notice that the pitching and rolling moments have not been precisely modelled. And this hunting phenomenon happens only with the lateral components.

The small oscillations along Z axis come from the stabiliser which makes the vertical thrust vary with the vehicle's attitude (+/- 0.1V).

4.5. The reasons why this approach is unacceptable

4.5.1. The hunting phenomenon

The fluctuations around the equilibrium are too large to be acceptable. Their amplitude can reach 30% of the input value.

4.5.2. The signals amplitude

Once again, the feedback gains are too low and provide a weak signal which makes the Draganflyer X-pro require too high precision.

4.5.3. The stability margin

The hunting phenomenon is a consequence of a very small stability margin. If the UAV undergoes some external disturbances such as wind, it will fly away from its input. Some phase lead is required to enhance the stability margin.

These disadvantages make this approach unacceptable. The vehicle's modelling needs to be reviewed, especially the rotor dynamics which aimed to be precisely modelled but which are in fact modelled around a too specific flight condition. We need a model more tolerant which remains valid for various flight conditions.

5. Using an engineering approach

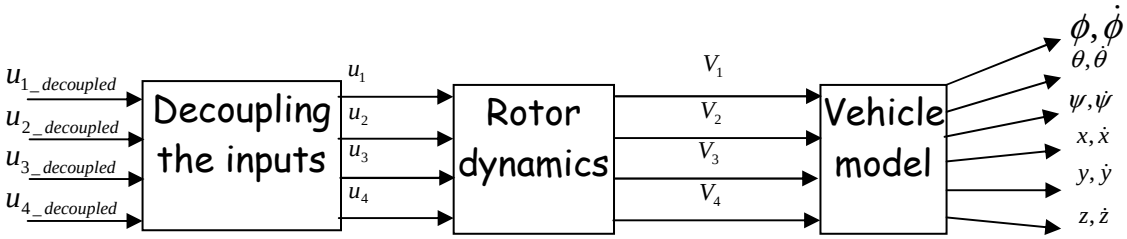


Figure 5.1 : structure of the engineering approach

As we can see on the above figure, this last approach feeds back the position, the velocity, the attitude and rate of change of attitude. The control structure is the same as in the previous approach except that the vertical acceleration is not fed back. Besides, the rotor dynamics block relates the voltages with u1, u2, u3, u4 and not u1', u2', u3', u4' as it used to do.

5.1. Modelling the rotor dynamics

The rotor dynamics need to be investigated more deeply. The unsatisfactory results we got so far come from a bad modelling.

As stated at the end of the previous chapter, we need a model more tolerant which remains valid for many flight conditions.

5.1.1. Criticising the previous approach

When we studied the step responses of the Draganflyer X-pro Simulink model to linearize the rotors, we took into account the influence of the quadrotor's airspeed on the provided thrust. Considering the air flow generated by the quadrotor's motion in the rotors modelling would have been interesting if this influence didn't depend so much on the flight condition. This method could have led to good performances if it had been investigated around many flight conditions.

Note that the performances achieved were satisfactory for the yaw angle control. That means that the relation between torque and applied voltage remains valid for a large

flight envelope. This makes sense with what has just been said. Indeed, a yawing moment doesn't cause any extra air flow through the rotors. Thus, the results from Simulink are sufficient to give a correct relation.

However, as derivative blocks are not allowed anymore, both relations between thrust and voltage and between torque and voltage need to be reviewed.

The relation between torque and applied voltage

$$\frac{u_4}{V_1 - V_2 + V_3 - V_4} = \frac{0.045}{0.314s + 1}$$

will just be used to approximate the rotor lag.

5.1.2. The new approach

In order to get a simple model valid for a large flight envelope, it's better to look at the relation between thrust and supplied voltage without any free stream (hover). Wind tunnel tests have been carried out by the author of the full non linear model. This person has measured the thrust provided by one of the four propellers for ten different values of voltage. During the experiment, the quadrotor was fixed at the hover (no free stream). A linear relation has then been worked out in Excel to find the Direct Current gain (D.C. gain). The associated Excel file is in the appendix 3.1.

The D.C. gain is:

$$\left.\frac{u_1}{V_1 + V_2 + V_3 + V_4}\right|_{steady\_state} = 0.5205 \text{ N/V}$$

From now on, we just need to consider the rotor lag. In order to model the rotor lag, we are going to add a phase lead block in front of the vehicle's model. Assuming that the lag mainly comes from the rotational speed to settle, the rotor lag can be measured on the figure 3.10 (yawing moment response to a step applied on $V_1 - V_2 + V_3 - V_4$). The time constant was 0.314 second.

Thus, the exact transfer function we should use to compensate the rotor lag is:

$$H(s) = 0.314s + 1$$

As this transfer function is not proper, it has to be approximated by a classical phase lead transfer function:

$$H(s) = \frac{Ts+1}{\frac{T}{10}s+1} \quad \text{with T=0.314 second}$$

Here is the associated Bode diagram:

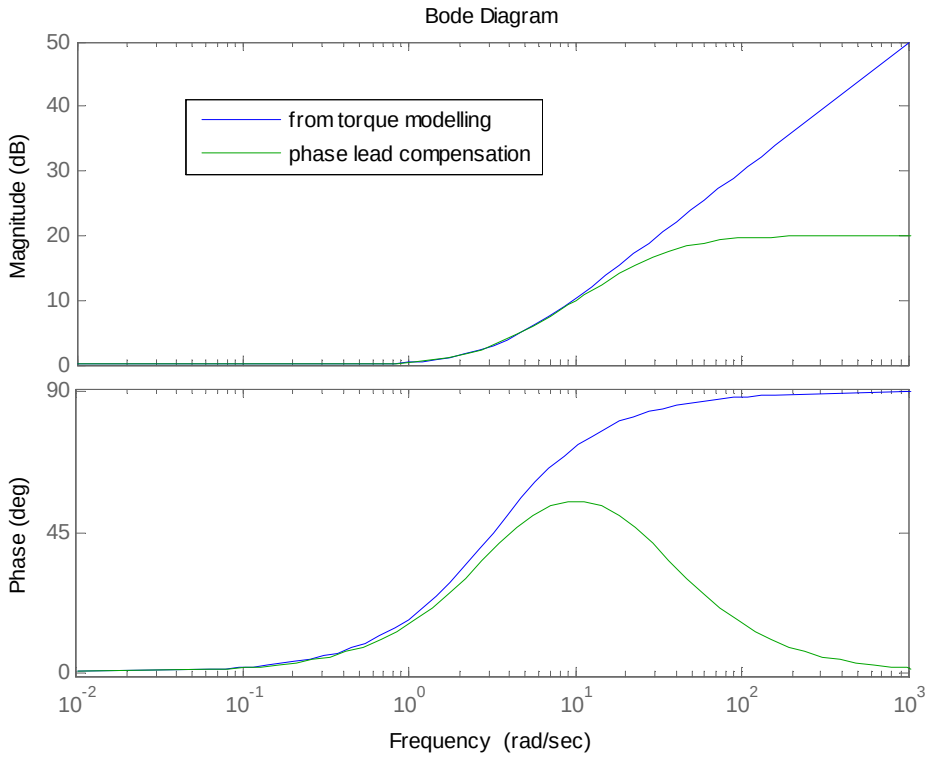


Figure 5.2 : Bode diagram of the phase lead transfer function used for rotor lag

As we can see on the above figure, the approximation remains valid for a limited range of frequencies. When the thrust or torque input varies too rapidly ($\omega > 10$ rad/sec), the rotor dynamics modelling is not valid. This is going to limit the performances of the control law. Gain values cannot be too large; otherwise high frequency signals are generated. However, 10 rad/sec is large enough to provide satisfactory dynamic behaviour.

Note that the magnitude of the phase lead transfer function is 0 dB for low frequencies and 20 dB for high frequencies. That means that at low frequencies the

voltage is entirely converted into thrust, so no need to amplify the signal, and at high frequency, as the provided thrust is attenuated, the phase lead block multiply the input by 10.

Here is the Simulink model of this modelling:

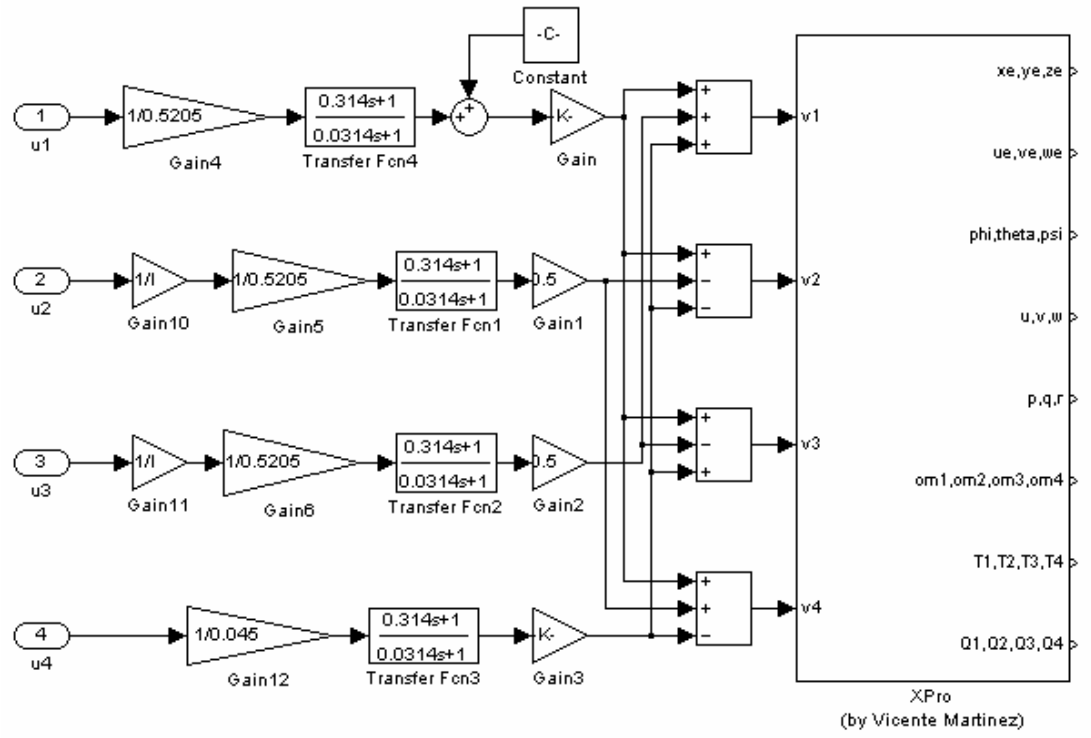


Figure 5.3 : Simulink model of the ensemble {rotor dynamics + X pro} (Using an engineering approach)

The D.C. gain for the torque is different from the one for the thrust. As the relation between torque and voltage has been properly modelled by a first order transfer function in the chapter 3 (see figure 3.11), we are using the same value for the D.C. gain (0.045 N.m/V).

As u2 and u3 are thrust multiplied by length, we just need to multiply these two inputs by 1/l.

5.2. Decoupling the inputs

The associated Matlab function is the same as in the previous chapter (see appendix 2.1). We keep assuming that the coupling between x, y, z and ϕ, θ, u_1 is not significant. Therefore, we consider only the coupling between ϕ, θ, ψ and u_2, u_3, u_4:

$$u_{2\_decoupled} = c_{\psi_0} u_2 - \frac{I_{xx} s_{\psi_0}}{I_{yy}} u_3 = I_{xx} \ddot{\phi}$$

$$u_{3\_decoupled} = \frac{s_{\psi_0} I_{yy}}{c_{\phi_0} I_{xx}} u_2 + \frac{c_{\psi_0}}{c_{\phi_0}} u_3 = I_{yy} \ddot{\theta}$$

$$u_{4\_decoupled} = \frac{I_{zz} s_{\psi_0} t_{\phi_0}}{I_{xx}} u_2 + \frac{I_{zz} c_{\psi_0} t_{\phi_0}}{I_{yy}} u_3 + u_4 = I_{zz} \ddot{\psi}$$

The transformation carried out in Simulink is:

$$\begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} c_{\psi_0} & s_{\psi_0} c_{\phi_0} \frac{I_{xx}}{I_{yy}} & 0 \\ -s_{\psi_0} \frac{I_{yy}}{I_{xx}} & c_{\psi_0} c_{\phi_0} & 0 \\ 0 & -s_{\phi_0} \frac{I_{zz}}{I_{yy}} & 1 \end{bmatrix} \begin{bmatrix} u_{2\_decoupled} \\ u_{3\_decoupled} \\ u_{4\_decoupled} \end{bmatrix}$$

5.3. Designing the control law

5.3.1. The state space system

The state vector is:

$$X = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$$

The input vector is:

$$U = \begin{bmatrix} u_1 & u_{2\_decoupled} & u_{3\_decoupled} & u_{4\_decoupled} \end{bmatrix}^T$$

Hence, we have:

$$\dot{X} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} X + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1/m & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1/I_{xx} & 0 & 0 \\ 0 & 0 & 1/I_{yy} & 0 \\ 0 & 0 & 0 & 1/I_{zz} \end{bmatrix} U$$

Details of the associated Matlab file are in the appendix 3.2. This Matlab file also includes the design of both PID and LQR controllers.

5.3.2. PID controller

Once again, the root locus method is used to design this controller. The associated root loci are enclosed in the appendix 3.3.

Here is a table gathering the chosen gain values:

Variables controlled / Variables fed back	Control of x	Control of y	Control of z	Control of psi
$\dot{\theta}$ feedback	5			
θ feedback	37.3			
$\dot{x}$ feedback	-10.92			
x feedback	-10.9			
$\dot{\phi}$ feedback		5		
ϕ feedback		37.3		
$\dot{y}$ feedback		10.92		
y feedback		10.9		
$\dot{z}$ feedback			-10	
z feedback			-10.6	
$\dot{\psi}$ feedback				3
ψ feedback				7.56

In order to optimize the vehicle's performances, these values have been chosen so that for a specific controlled variable (x, y, z or psi), the last feedback brings the slowest pole as far as possible from the imaginary axis to get fast dynamic features but not too far in order to attenuate the high frequency signals.

Indeed, the rotor modelling remains correct for frequencies inferior to 10 rad/sec. Thus, higher frequencies need to be attenuated so that the input signal of the phase lead blocks has mainly low frequency components. The requirement to achieve this constraint is: the slowest poles of the closed loop system must have a real part superior to -10 rad/sec. Meanwhile, the dynamic of the closed loop system has to be rapid. Thus, the slowest closed loop poles are designed to be located at the left of x=-1rad/sec.

This criterion is illustrated on the following figure:

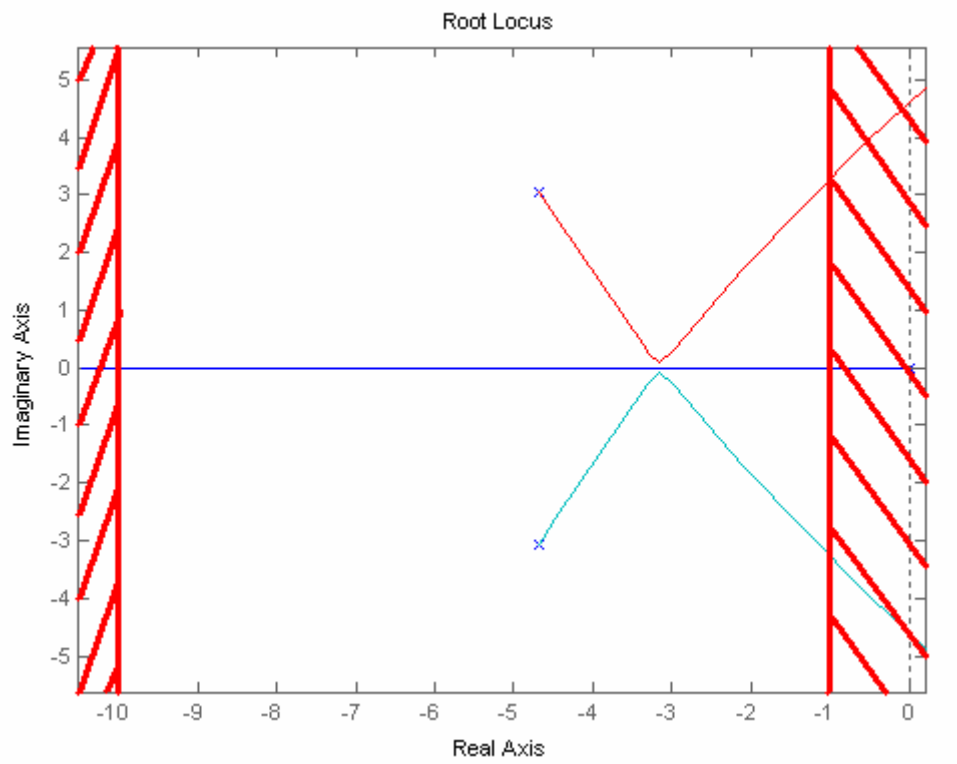


Figure 5.4 : criterion on the location of the slowest closed loop poles

As we can see on the root loci in the appendix 3.3, the above criterion is respected for the four controlled variables.

Here are the step responses we should get in Simulink with the non linear model:

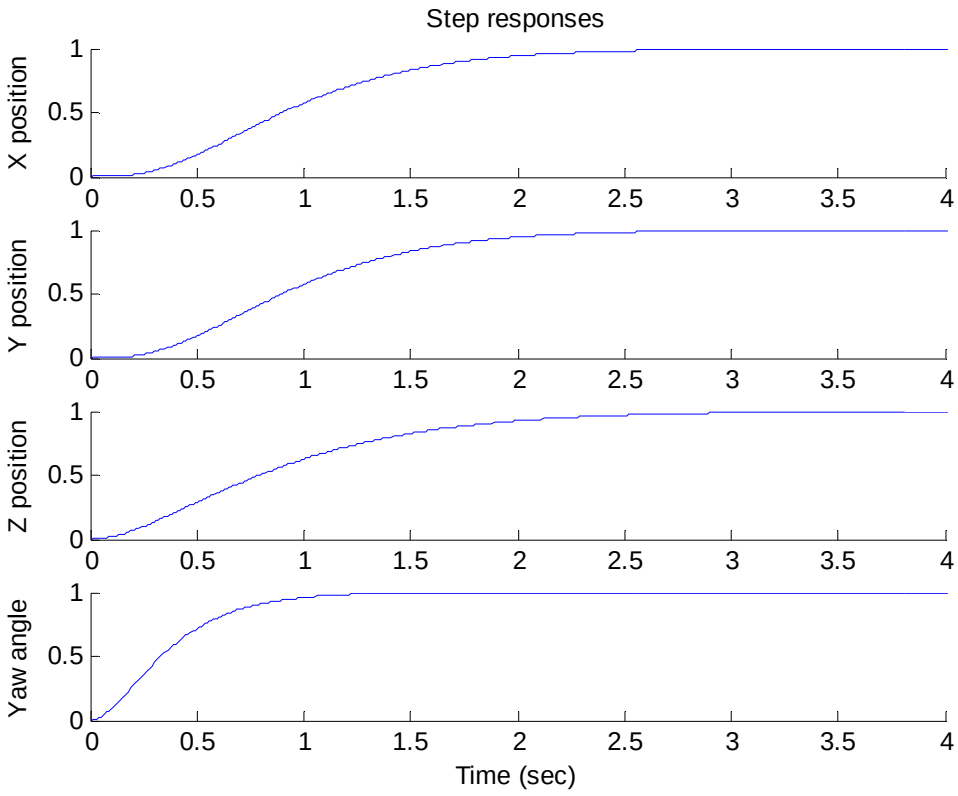


Figure 5.5 : influence of PID controller on the closed loop system designed in Matlab (Using an engineering approach)

5.3.3. LQR controller

$$X = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$$
$$U = \begin{bmatrix} u_1 & u_{2\_decoupled} & u_{3\_decoupled} & u_{4\_decoupled} \end{bmatrix}^T$$

This kind of controller aims to minimize the following quadratic cost function:

$$J = \int_0^\infty \left(X^T(t) Q X(t) + U^T(t) R U(t) \right) dt$$

using a feedback controller K such that $U(t) = -KX(t)$. Q and R are weighting matrices. In Matlab, thanks to the command "are()", it's relatively easy to work out the optimal controller K such that J is minimized. Indeed, if the state space representation of the system is $\dot{X} = AX + BU$, we have:

$$K = R^{-1}B^T * are(...)$$

(see Matlab code in the appendix 3.2 for further details)

Consequently, the task in the LQR design is to choose appropriate weighting matrices.

Q and R are usually diagonal matrices. Coefficients of Q limit the amplitude of the state variables, coefficients of R limit the amplitude of the inputs. The larger these coefficients, the smaller the state/input variables. The chosen weighting matrices are:

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$$

The Q matrix has unitary coefficients except for the controlled variables (x, y, z and psi). Indeed, the priority is to make the UAV reach its desired position as fast as possible. In other terms, the main goal is to minimize the $L_2 - norm$ of x, y, z and psi. Multiplying the associated coefficients by 10 highlights these variables and makes the cost function depend more significantly on them.

After different attempts, it appears that the dynamic performances are optimized with the above R matrix. This means that the input amplitudes don't really need to be minimized.

With these weighting matrices, the controller is in fact designed to minimize the $L_2 - norm$ of x, y, z and psi without considering the other variables too much.

The controller generated by Matlab with the above weighting matrices has the following characteristics:

Variables controlled / Variables fed back	Control of x	Control of y	Control of z	Control of psi
$\dot{\theta}$ feedback	4.3043			
θ feedback	25.2884			
$\dot{x}$ feedback	-7.8458			
x feedback	-10			
$\dot{\phi}$ feedback		4.2976		
ϕ feedback		25.2673		
$\dot{y}$ feedback		7.8431		
y feedback		10		
$\dot{z}$ feedback			-15.7751	
z feedback			-31.6228	
$\dot{\psi}$ feedback				3.9936
ψ feedback				10

Here are the dynamic features we can expect to get with the full non linear model:

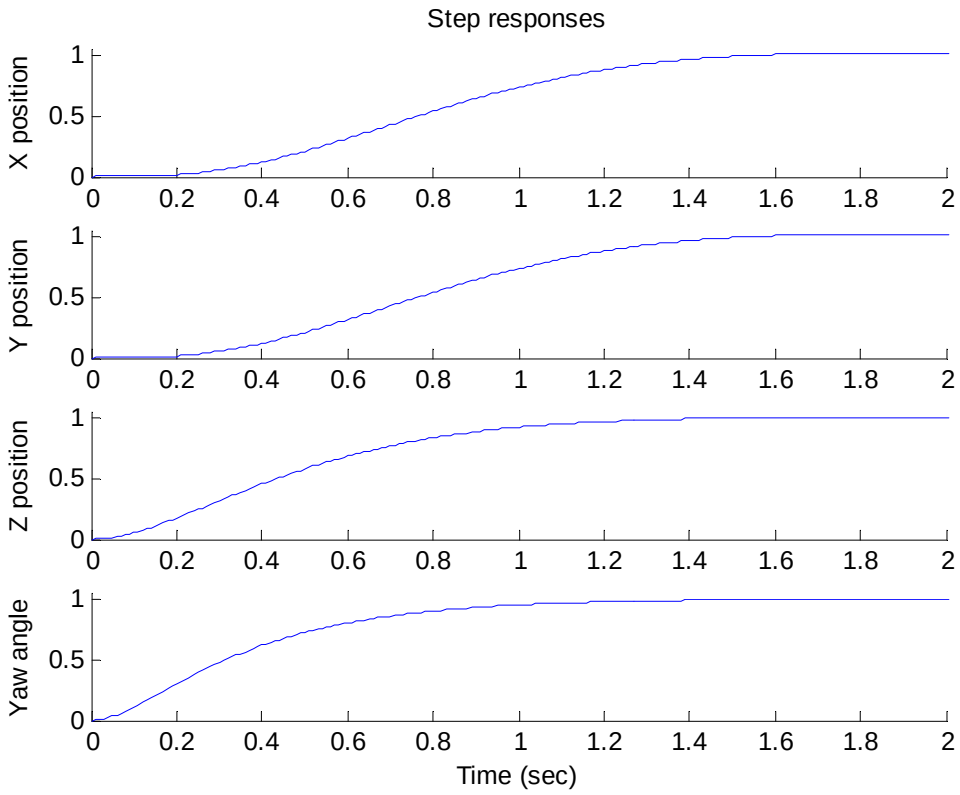


Figure 5.6 : influence of LQR controller on the closed loop system designed in Matlab (Using an engineering approach)

5.4. Achieved performances

Here is the Simulink model used for PID simulations. The model run for LQR simulations is the same except that the feedback gains are different.

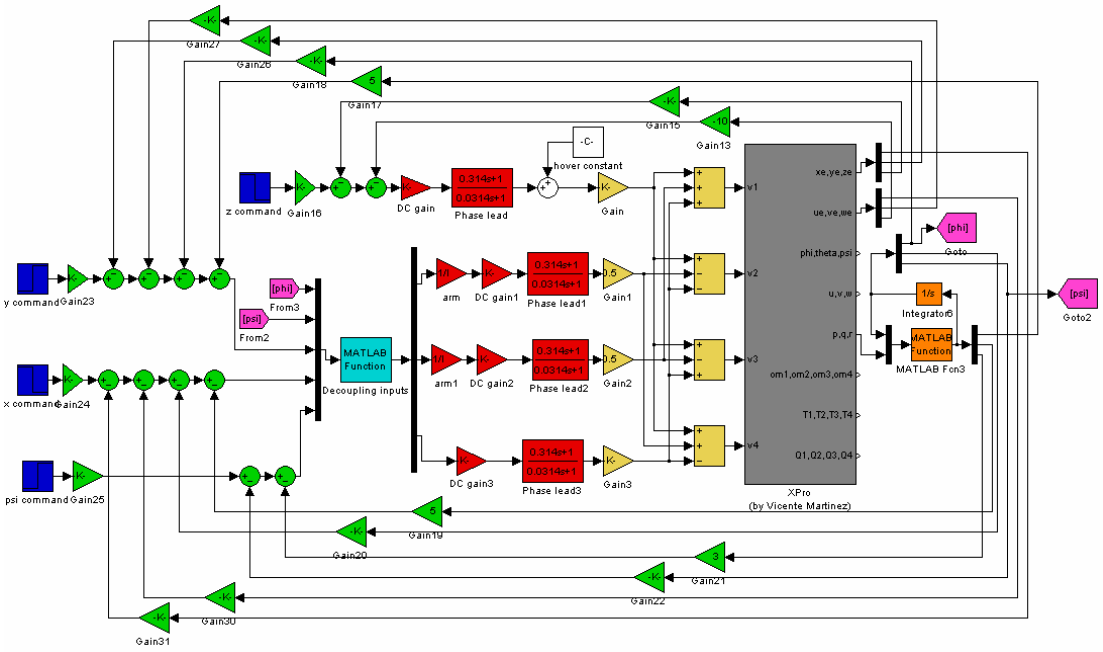


Figure 5.7 : Simulink model of the closed loop system using the full non linear model (Using an engineering approach)

5.4.1. PID controller

When the previous PID control law is applied on the above model, the step responses obtained are very similar to those expected with the theoretical model:

- ✓ the settling time is around 3 seconds
- ✓ no overshoot for x, y and z
- ✓ small overshoot for the yaw angle
- ✓ no steady state error

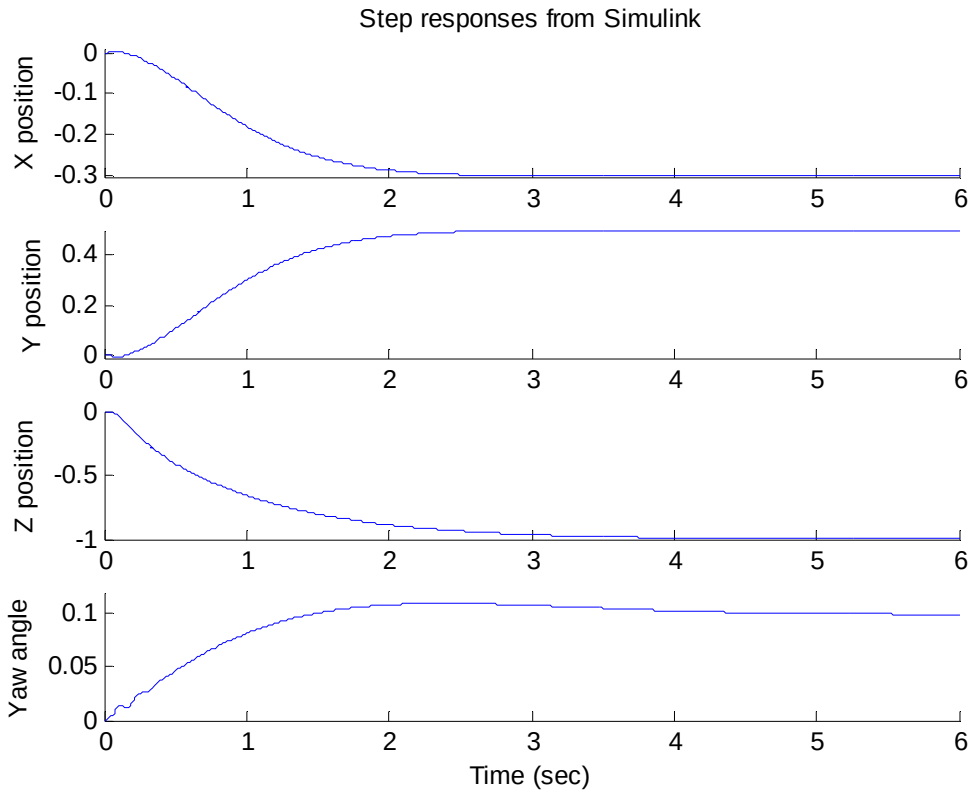


Figure 5.8 : influence of PID controller on the Simulink closed loop system using the full non linear model (Using an engineering approach)

Considering the inertia of the Draganflyer X-pro, this dynamic behaviour is completely satisfactory.

However, if the input command is too large, the UAV doesn't behave like this anymore. This is due to the pitch and roll angles which become too large. Indeed, the quadrotor bank angles must remain in a certain range of values in order to maintain the altitude. The investigation of aggressive manoeuvres is carried out in the next chapter.

5.4.2. LQR controller

The implementation of the previous LQR controller on the full non linear model gives rise to the expected results.

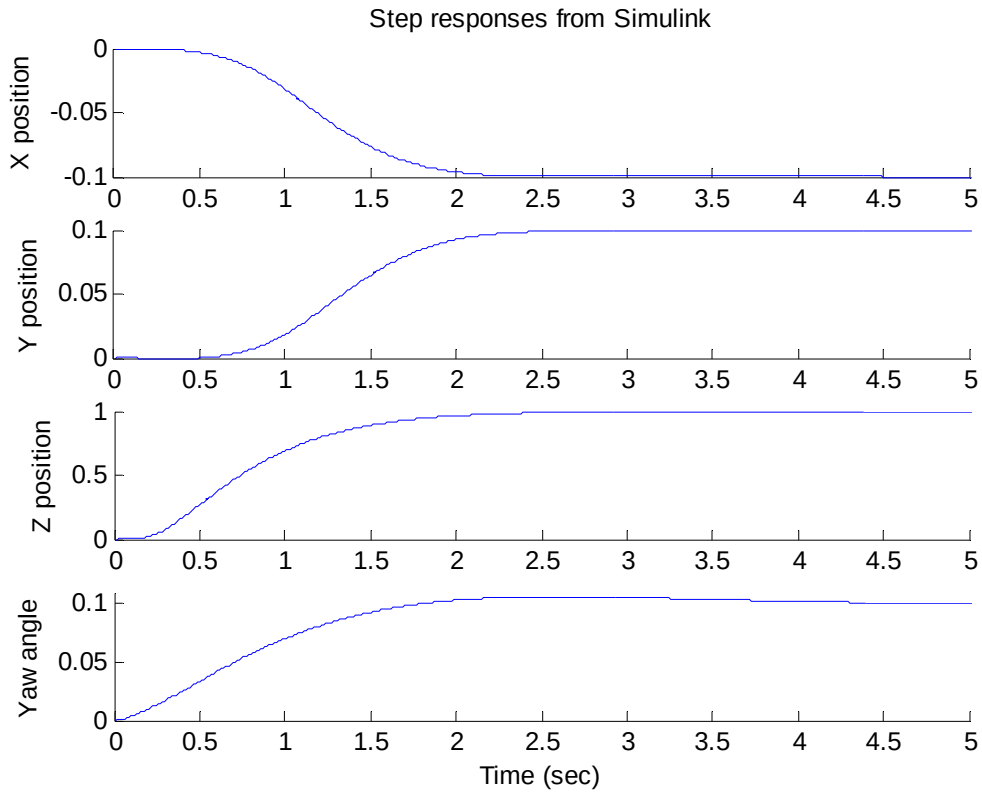


Figure 5.9 : influence of LQR controller on the Simulink closed loop system using the full non linear model (Using an engineering approach)

The achieved performances are slightly faster than those obtained with the PID controller. But once again, this nice dynamic behaviour is valid only for small input amplitudes.

If we want the quadrotor to fly correctly for larger input amplitudes, we have to firstly investigate the vehicle's limits and then, rearrange accordingly the control structure.

6. The quadrotor's limits: aggressive manœuvres

This chapter is designed to make the quadrotor stable for any applied inputs. This study is carried out by investigating all the different kinds of limits the UAV can reach. This work is meant to provide a new model robust against aggressive commands.

6.1. Voltage limits of the Draganflyer X-pro

The voltage input for each propeller must vary between 0 and 12V. Note that the voltage required to make the UAV hover is 7.3060 V on each rotor.

6.1.1. Implementation in Simulink

A saturation block is added in front of each voltage input:

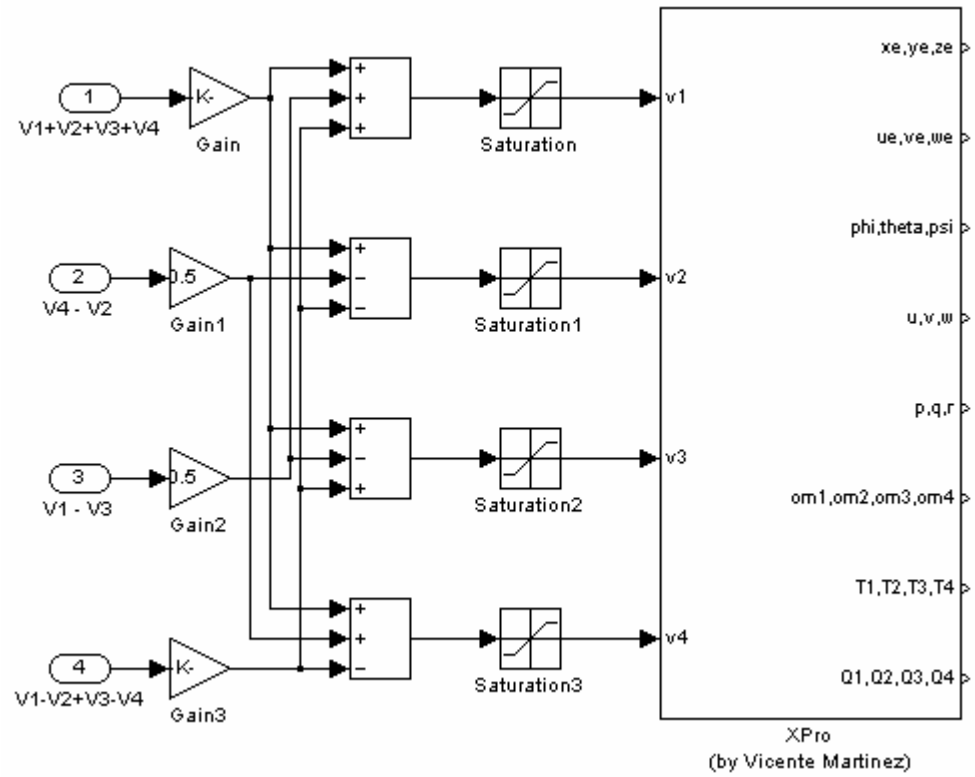


Figure 6.1 : modelling the voltage limits

6.1.2. Consequences and solution

Both PID and LQR control laws provide large voltages because of their gains. As soon as the commands for x, y and z approach 0.01 meter, the voltages saturate at the upper limit. This saturation gives rise to unexpected undershoots and overshoots.
As the voltage limit comes largely in first position before the other limits such as roll or pitch angle, the feedback gains need to be reduced.

- ✓ PID controller

Variables controlled / Variables fed back	Control of x	Control of y	Control of z	Control of psi
$\dot{\theta}$ feedback	1			
θ feedback	1.49			
$\dot{x}$ feedback	-0.087			
x feedback	-0.0173			
$\dot{\phi}$ feedback		1		
ϕ feedback		1.49		
$\dot{y}$ feedback		0.087		
y feedback		0.0173		
$\dot{z}$ feedback			-5	
z feedback			-2.66	
$\dot{\psi}$ feedback				1
ψ feedback				0.841

With this new control law, the rotor voltages begin saturating for the following flight path:

- ✓ 5 meters along X axis,
- ✓ 5 meters along Y axis,
- ✓ 5 meters climbing along Z axis
- ✓ Rotation of 0.1 rad

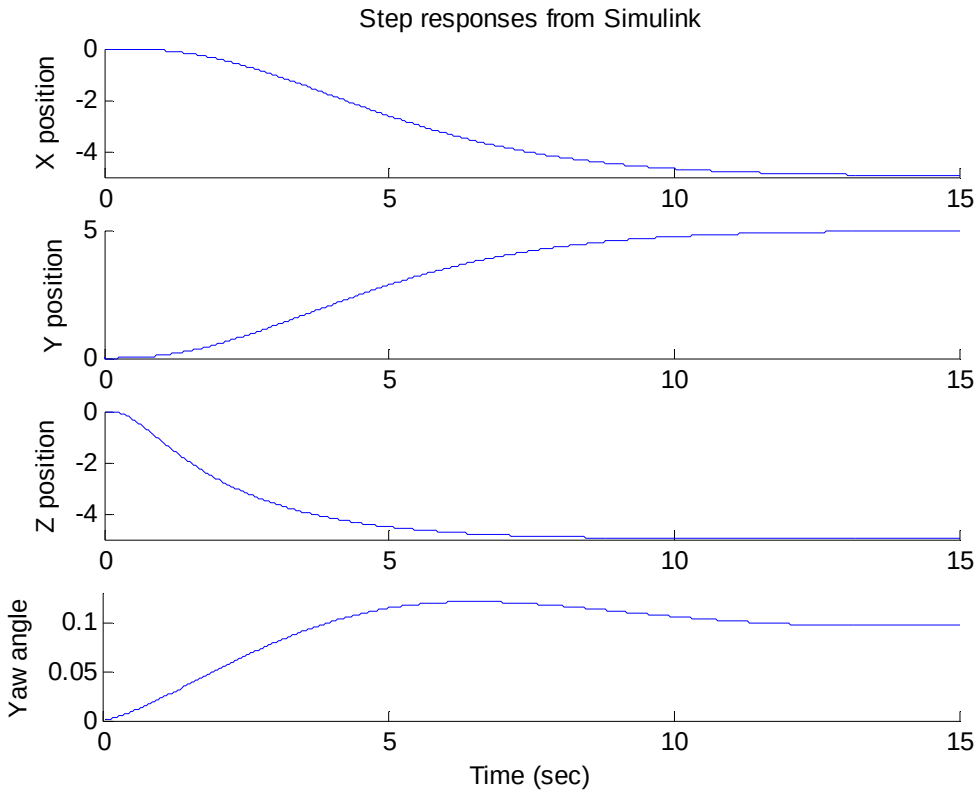


Figure 6.2 : performances of new PID controller designed to avoid voltage saturation

Of course, with these new gains, the settling time has significantly increased, but 15 seconds can still be considered as satisfactory performances.

- ✓ LQR controller

We keep the same Q weighting matrix but we increase the coefficients of R in order to limit the voltage amplitude:

$$R = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100000 & 0 & 0 \\ 0 & 0 & 100000 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

Variables controlled / Variables fed back	Control of x	Control of y	Control of z	Control of psi
$\dot{\theta}$ feedback	0.3863			
θ feedback	0.4426			
$\dot{x}$ feedback	-0.0302			
x feedback	-0.01			
$\dot{\phi}$ feedback		0.3846		
ϕ feedback		0.4413		
$\dot{y}$ feedback		0.0302		
y feedback		0.01		
$\dot{z}$ feedback			-1.2242	
z feedback			-0.3162	
$\dot{\psi}$ feedback				0.8336
ψ feedback				1

As the new PID controller, the new LQR controller makes the rotor voltages saturate from the following commands:

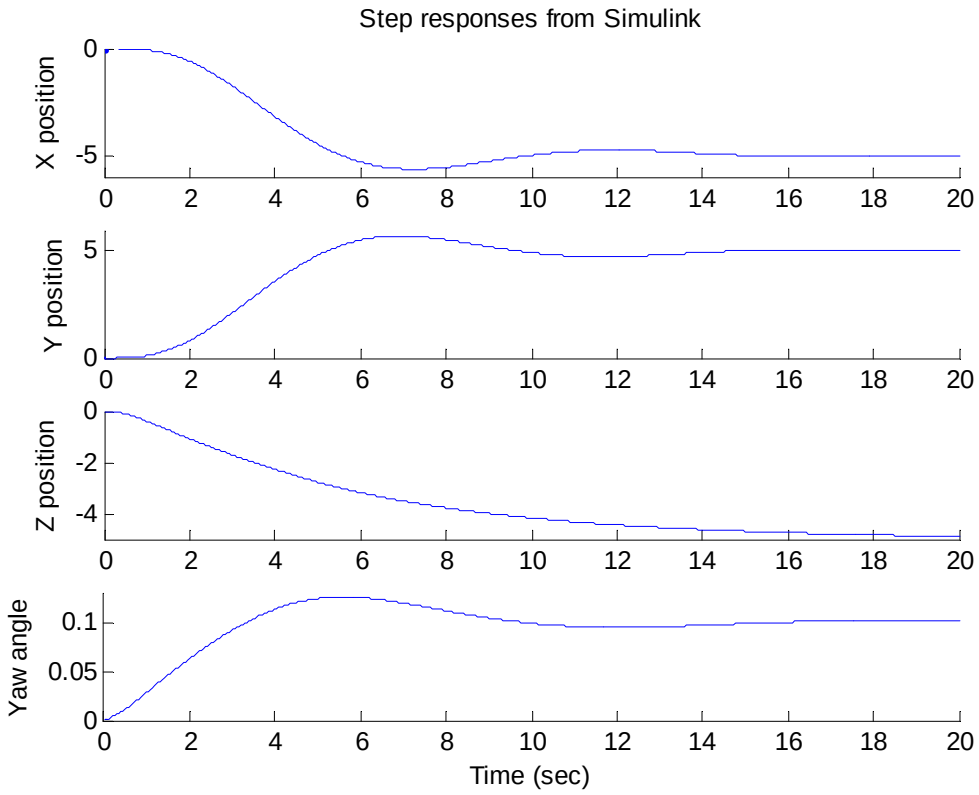


Figure 6.3 : performance of new LQR controller designed to avoid voltage saturation

6.2. Effects of aggressive altitude command

6.2.1. Climbing

When the quadrotor's altitude command is too high, the demand on $V_1+V_2+V_3+V_4$ becomes large. Because of the saturation blocks, the lateral and yaw angle demands cannot influence the voltage inputs if $V_1+V_2+V_3+V_4$ is too large.

Let's consider the following demands:

- ✓ $V_1+V_2+V_3+V_4=52\,\text{V}$
- ✓ $V_4-V_2=1\,\text{V}$
- ✓ $V_1-V_3=1\,\text{V}$
- ✓ $V_1-V_2+V_3-V_4=2\,\text{V}$

We have:

$$\begin{bmatrix}V_1\\V_2\\V_3\\V_4\end{bmatrix}=\begin{bmatrix}0.25 & 0 & 0.5 & 0.25\\0.25 & -0.5 & 0 & -0.25\\0.25 & 0 & -0.5 & 0.25\\0.25 & 0.5 & 0 & -0.25\end{bmatrix}\begin{bmatrix}V_1+V_2+V_3+V_4\\V_4-V_2\\V_1-V_3\\V_1-V_2+V_3-V_4\end{bmatrix}$$

$$\Rightarrow \quad \begin{bmatrix}V_1\\V_2\\V_3\\V_4\end{bmatrix}=\begin{bmatrix}0.25 & 0 & 0.5 & 0.25\\0.25 & -0.5 & 0 & -0.25\\0.25 & 0 & -0.5 & 0.25\\0.25 & 0.5 & 0 & -0.25\end{bmatrix}\begin{bmatrix}52\\1\\1\\2\end{bmatrix}=\begin{bmatrix}14\\12\\13\\13\end{bmatrix}$$

After the saturation blocks, the four voltages are all equalled to 12 volts. Thus, only the vertical command will be considered. No moment can be generated. If the UAV wants to go from point A to point B, it won't fly a straight line as we would like. It will firstly reach a specific altitude through a vertical motion, and then it will move laterally.

To prevent this situation, an other saturation block is added:

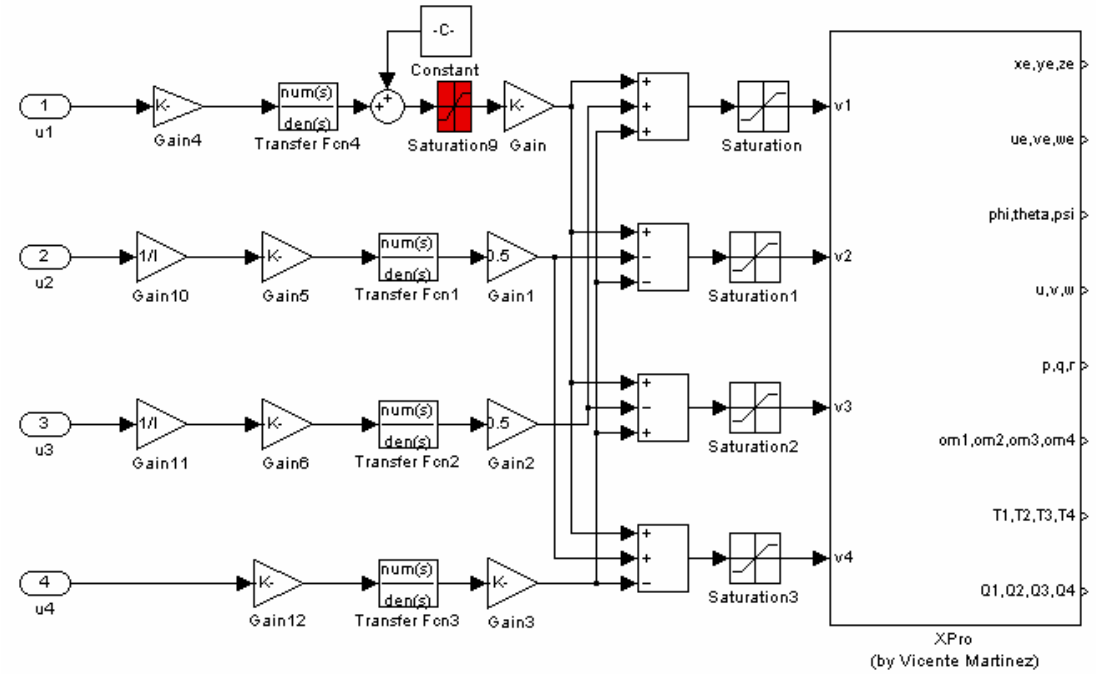


Figure 6.4 : saturation block to enable simultaneous lateral and vertical motions

The upper limit is 44 volts. Consequently, there are always 4 extra volts available to generate a moment.

The lower limit is 0 volt. As the problem is symmetric, we could have put 4 volts as the lower limit. But the reason for that choice is given in the next paragraph.

6.2.2. Descent

If the quadrotor's rate of descent is too large, it cannot recover the hover flight condition because of the voltage upper limit. Thus, in order to keep the altitude control effective, the descent airspeed has to remain inferior to a certain limit.

After some simulations, it appears that this limit is 7 m/sec (25.2 km/h). Therefore, a saturation block is placed in the model as followed:

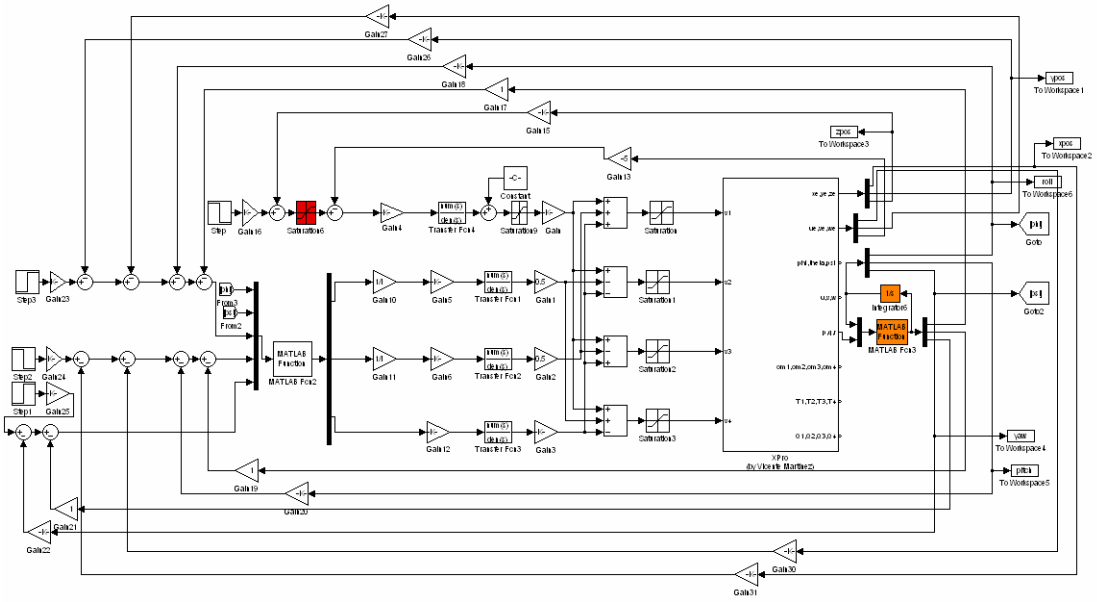


Figure 6.5 : limiting the Draganflyer X-pro's rate of descent

As we don't want to limit the rate of climb, the upper limit is 10^{10} m/sec. As this limit is expressed in terms of vertical airspeed demand, the lower limit depends on the $\dot{z}$ feedback gain. For the PID controller, the lower limit is $-5*7=-35$ m/sec, and for the LQR controller, it's $-1.2242*7=-8.5694$ m/sec.

Note that if the vehicle is at hover and the demand in $V_1+V_2+V_3+V_4$ is 0 volt, the rate of descent reaches 7m/sec in less than 1 second ($g=9.81m.\sec^{-2}$). As the new saturation block tends to stabilize the descent airspeed once it has reached 7m/sec, the demand $V_1+V_2+V_3+V_4=0\,\mathrm{V}$ cannot last more than one second. That's why, the lower limit of the previous saturation block is 0 V. Adding a constraint to prevent an event which can only last one second is worthless.

6.3. Effects of aggressive lateral command

6.3.1. Limit on roll and pitch angles

This limit is the most obvious. The quadrotor must not bank too much in order to remain vertically stable. Indeed, the more banked the vehicle, the less effective u1. After some simulations, one can find out that when $|\phi| > 0.6$ rad or $|\theta| > 0.6$ rad, the altitude control is badly influenced (0.6 rad=34 deg). Thus the demands on roll and pitch angles need to be bound with saturation blocks.

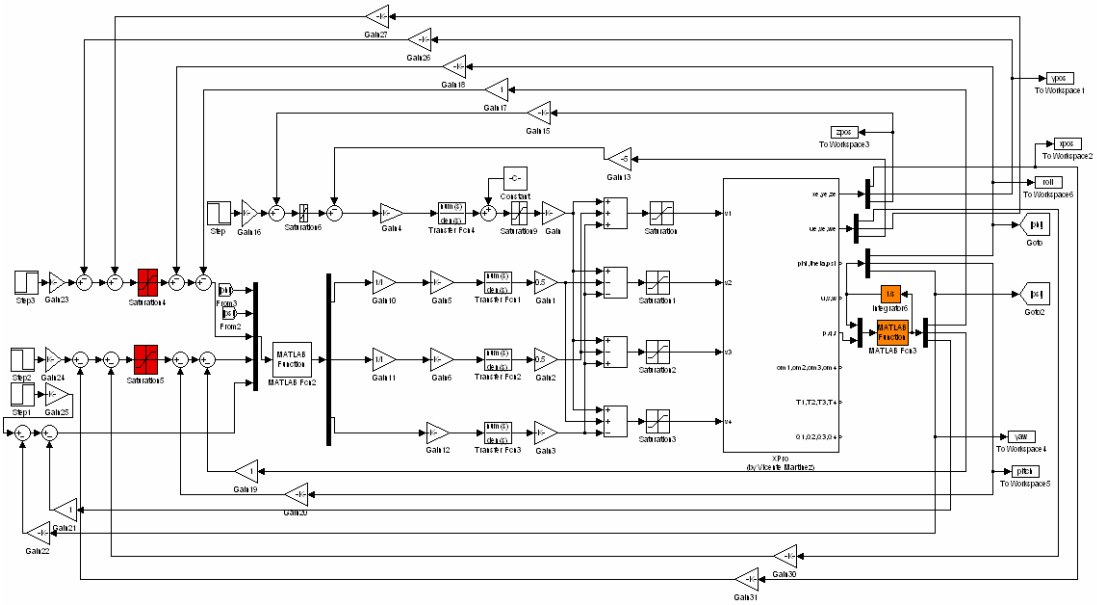


Figure 6.6 : limiting the Draganflyer X-pro's roll and pitch angles

As the control of roll and pitch angles features a significant overshoot, the limit on the demand has to be smaller to anticipate this overshoot. The chosen limit is 0.3 rad.

Therefore,

- ✓ For PID controller, the upper/lower limit is $\pm 0.3 * 1.49 = \pm 0.447$ rad
- ✓ For LQR controller, the upper/lower limit is $\pm 0.3 * 0.442 = \pm 0.1326$ rad

6.3.2. Limit on forward speed

Because of aerodynamic effects, lateral airspeed influences the thrust provided by the propellers. If this airspeed becomes too important, the vertical thrust begins decreasing significantly and the altitude control becomes impossible.

After several simulations, one finds that the forward speed must remain inferior to 16.1 m/sec (58 km/h). The solution to respect this criterion is to add a saturation block on the forward speed demands:

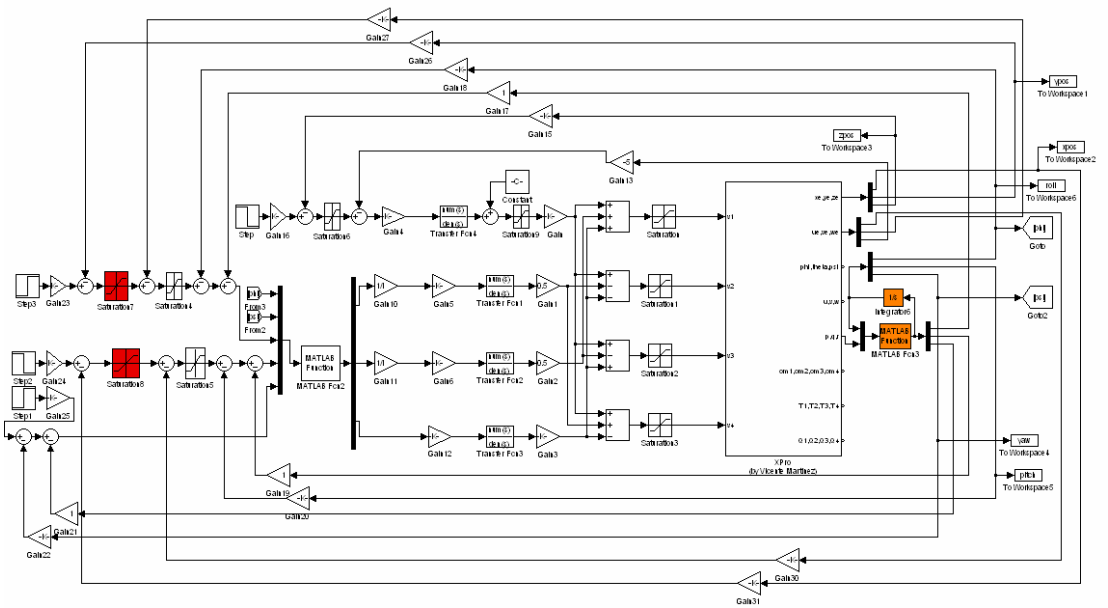


Figure 6.7 : limiting the Draganflyer X-pro's forward speed

The maximum forward speed is split into two maxima for x and y axes by dividing 16.1 m/sec by $\sqrt{2}$. This gives 11.4 m/sec.

Therefore,

- ✓ For PID controller, the upper/lower limit is $\pm 0.087 * 11.4 = 0.9918$ m/sec
- ✓ For LQR controller, the upper/lower limit is $\pm 0.0302 * 11.4 = 0.3443$ m/sec

6.4. Performances of the new Simulink model

These saturation blocks enable the UAV to be stable for any applied inputs, but on the other hand, they increase the settling time for long distance flights.
Indeed, with the PID controller, saturation blocks have the following influences:

6.4.1. Large descent

The settling time is affected as soon as the z command is superior to 60 meters (if the other commands are set to zero).

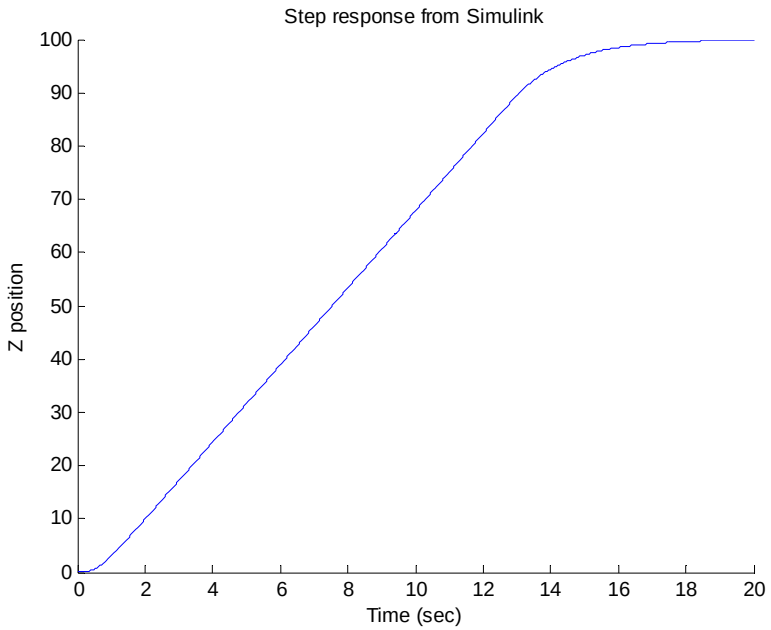


Figure 6.8 : influence of saturation blocks on large descent

The settling time was 10 seconds for small inputs. To fly down for 100 meters, the UAV needs 20 seconds.

6.4.2. Large climbing

When there's no other motion, the settling time of the altitude control begins increasing for $z_{command} < -60\,\text{m}$.

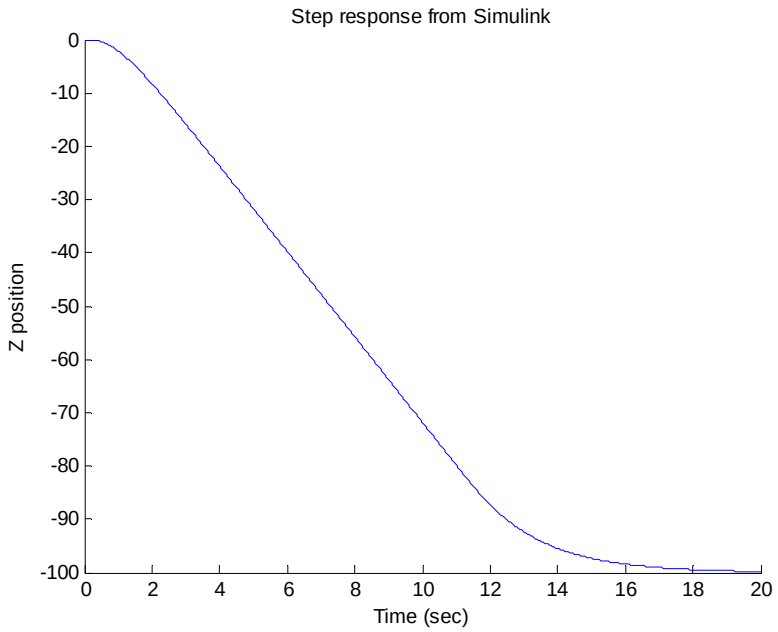


Figure 6.9 : influence of saturation blocks on large climbing

The settling time which was equalled to 10 seconds for small altitude commands is now equalled to 20 seconds to climb up 100 meters.

6.4.3. Large lateral commands

When the other inputs are negligible, the lateral settling time begins increasing for inputs superior to 100m. Here is the dynamic behaviour of y component for a large input:

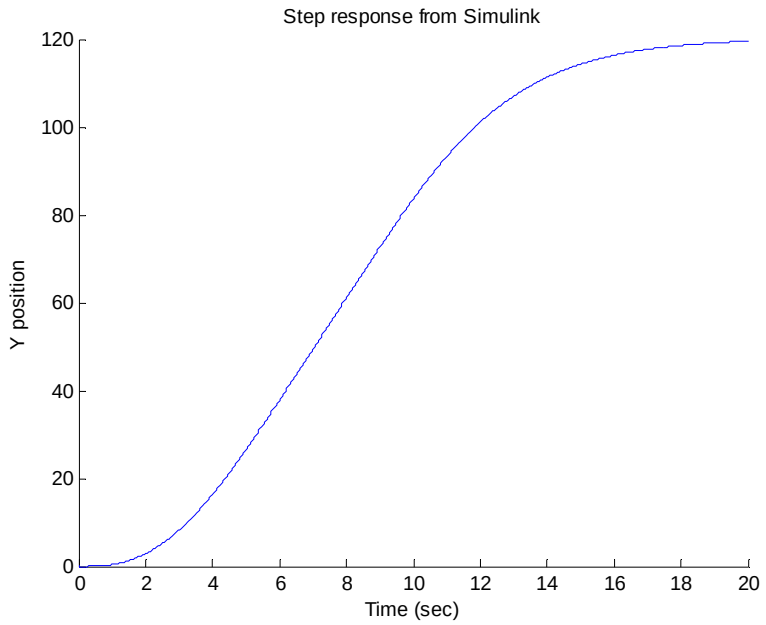


Figure 6.10 : influence of saturation blocks on lateral control

The settling time, which was 15 seconds, is now equalled to 20 seconds to move for 120 meters.

Note that large inputs on yaw angle have not been investigated on purpose. Indeed, it is assumed that step inputs larger than 3.14 rad are useless and don't need to be considered.

7. Trajectory following

This section aims to get a UAV which tracks as precisely as possible an input trajectory. To achieve this task, the control law needs to be readjusted.
PID and LQR controllers feed back the same variables on the same inputs. Only the gain values are different. As it is more intuitive to tune a PID controller than a LQR one, this chapter only considers the PID method.

7.1. Adapting PID controller to trajectory tracking

7.1.1. Reason why this step is necessary

In order to achieve good trajectory tracking, three main qualities are required for the closed loop system:

- ✓ Time constants as small as possible
- ✓ Damping ratios superior to 0.7 (otherwise, there is a resonance)
- ✓ The three components x, y and z must be synchronized in time domain

The first two requirements have already been studied. Only the third one needs to be optimized. This last requirement is very important because if the delay between x position and x command is different from the delay between z position and z command for example, the quadrotor won't be able to fly on the input flight path. Even if the quadrotor is flying behind the input position, the UAV's ability to fly along past input positions can prevent the vehicle from crashing in an obstacle.

Therefore, it is necessary to synchronize as well as possible the dynamic of the three components. The larger the difference, the further the vehicle will be from the input trajectory.

7.1.2. Re-design of the controller

In order to respect this requirement as well as possible, it's easier to work in the frequency domain.

The lateral components are already perfectly synchronized. Thus, the aim of this new controller is to synchronize the altitude control with the lateral control.

Let's have a look at the Bode diagrams of $\frac{X_{quad}(s)}{X_{command}(s)}$ and $\frac{Z_{quad}(s)}{Z_{command}(s)}$:

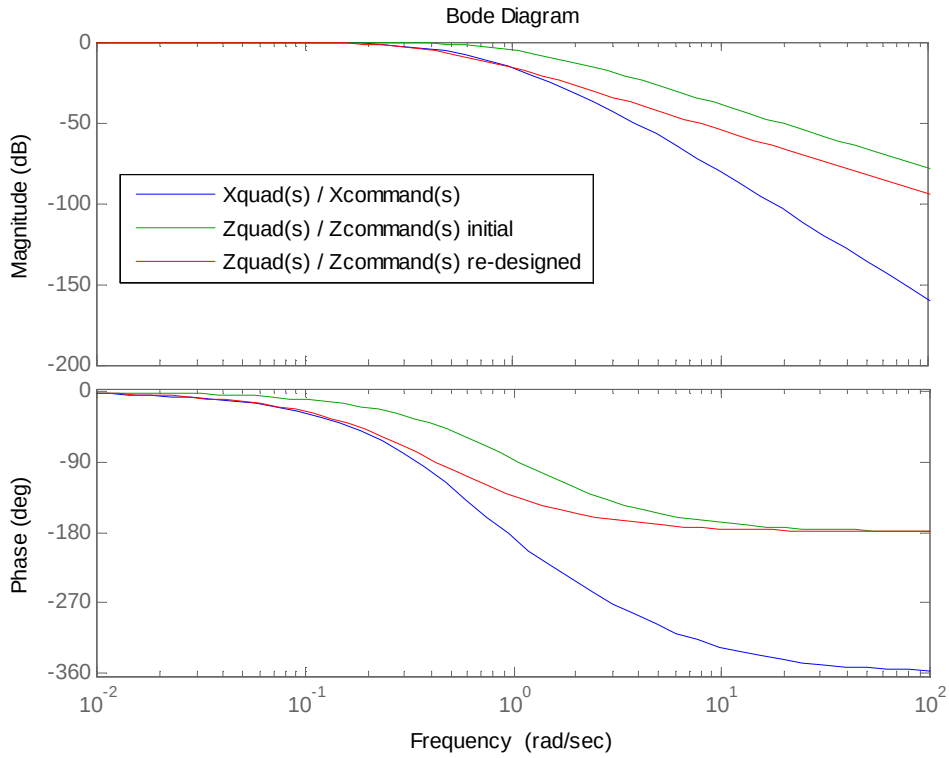


Figure 7.1 : Bode diagram of the closed loop transfer functions designed in Matlab

On the above figure, we can notice that high frequency inputs don't give the same responses for x and z.

However, if the control law of z is re-designed, it's possible to translate the green line towards the blue line and get dynamic behaviours more similar.

The new altitude control law approximates the Bode diagram of x with the red line. This new design seems to be optimal.

Here are the new feedback gains:

Variables controlled / Variables fed back	Control of z
$\dot{z}$ feedback	**-2**
z feedback	**-0.425**

7.1.3. New dynamic performances

The simulation with the non linear model gives rise to the following step responses:

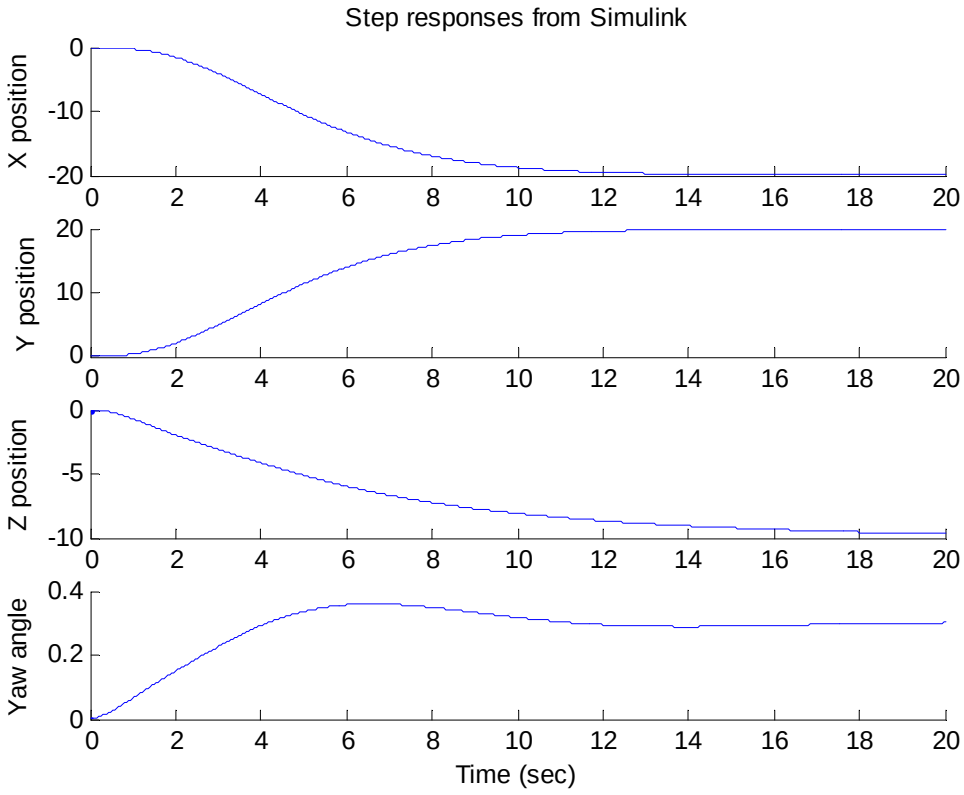


Figure 7.2 : dynamic features of PID controller re-designed for trajectory tracking (using the full non linear model)

The altitude step response is now slower. Note that the time it takes for the step responses to rise to 50% of their final value is approximately the same for x, y and z (around 5 seconds).

This last version of the Simulink model is enclosed in the appendix 4.1.

7.2. Generating specific trajectories

In order to create time dependent signals in Simulink, we use the following blocks:

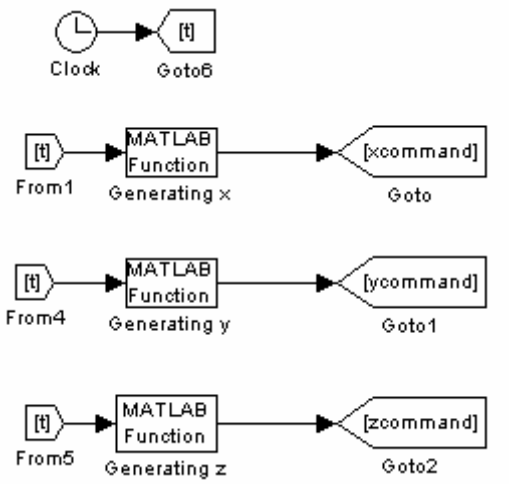


Figure 7.3 : Simulink model of the input signals generation

Then, according to the Matlab functions used to generate x(t), y(t) and z(t), every trajectory is possible.

7.2.1. Closed trajectory: circle

The Matlab functions used to create a circle are in the appendix 4.2.

7.2.2. Open trajectory: sinusoidal path

The Matlab functions used to create a sinusoidal trajectory are in the appendix 4.2.

7.3. Quadrotor's ability to track a given trajectory

7.3.1. Tracking a circle

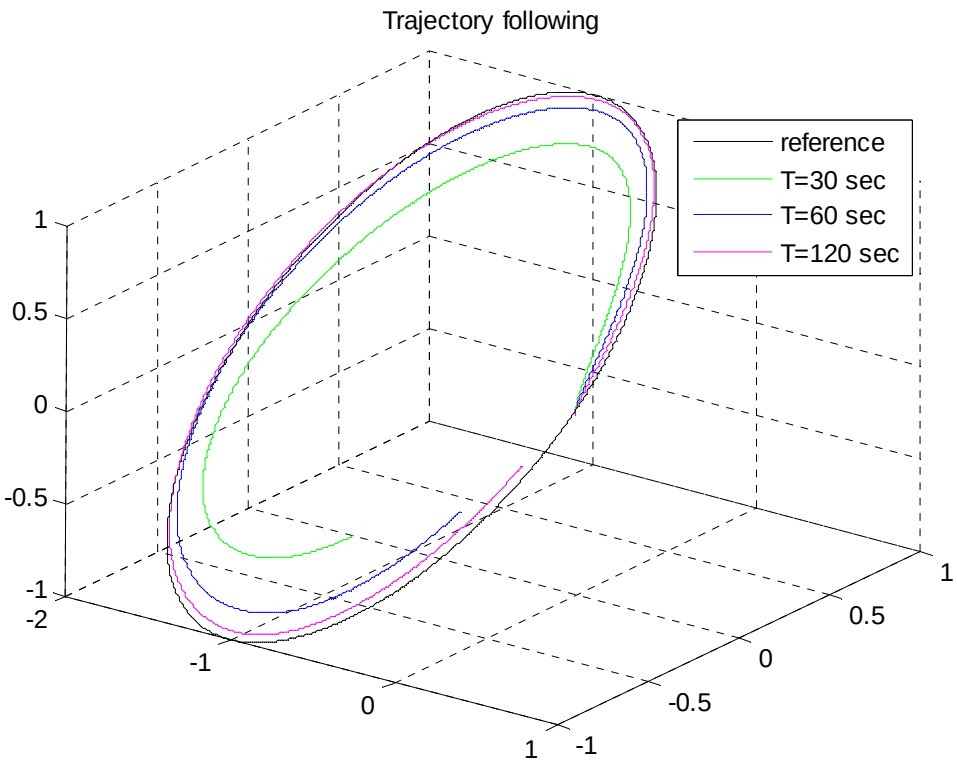


Figure 7.4 : ability of the quadrotor to follow a circle

As expected with the Bode diagram on figure 7.1, the magnitude begins being significantly attenuated for frequencies superior to 0.1 rad/sec (periods inferior to 60 seconds). For a period equalled to 120 seconds, the Draganflyer X-pro follows almost perfectly the circle.

Therefore, the UAV needs at least 2 minutes to complete accurately a circle.

7.3.2. Tracking a sinus

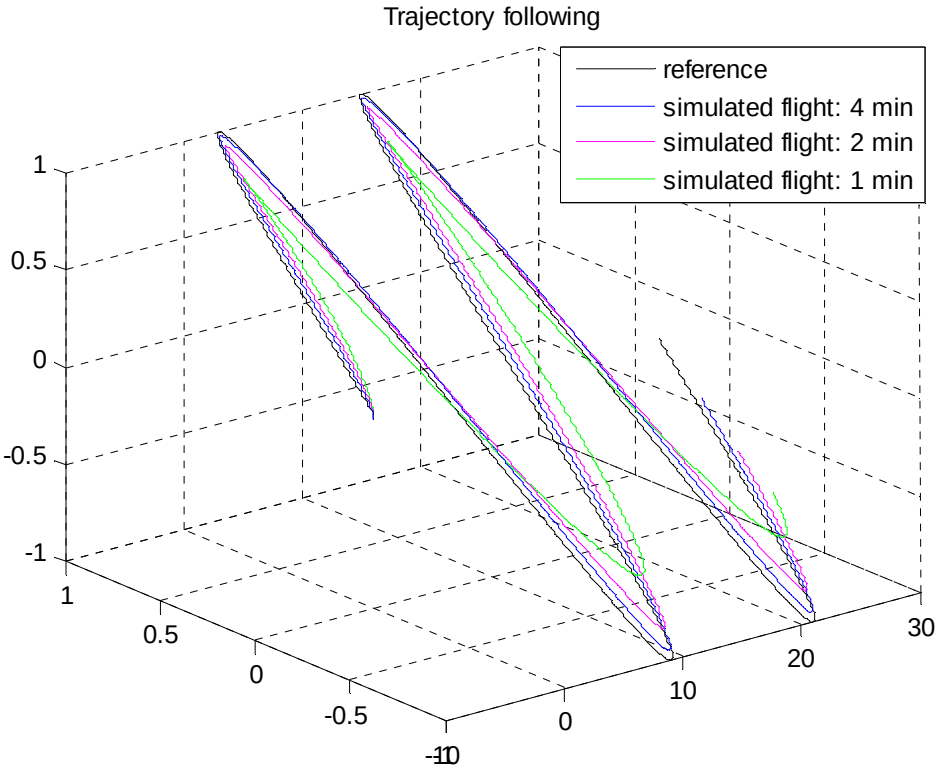


Figure 7.5 : ability of the quadrotor to follow an open trajectory

Here again, the quadrotor needs at least 2 minutes to track correctly one period.

If the trajectory input doesn't vary too rapidly (no components with frequency higher than 0.1 rad/sec), the ability of the Draganflyer X-pro to track the given flight path is really satisfactory.

8. Observing the quadrotor's flight attitude

This step has been carried out to check if the full non linear model is close enough to the reality. Indeed, real time videos have been made by gathering and processing the following simulated data:

- ✓ The three coordinates of the UAV position: x, y, z
- ✓ The three Euler angles of the UAV: ϕ, θ, ψ

With these data, it's possible to simulate entirely the flight of the vehicle. Thus, real time videos enable the consistence of the model to be checked.

These videos are included in the enclosed DVD. They use variables simulated with the latest Simulink model (see appendix 4.1).

8.1. Video of a step response

The Matlab code used to create the video file is in the appendix 5.1. Here is a snapshot of the video:

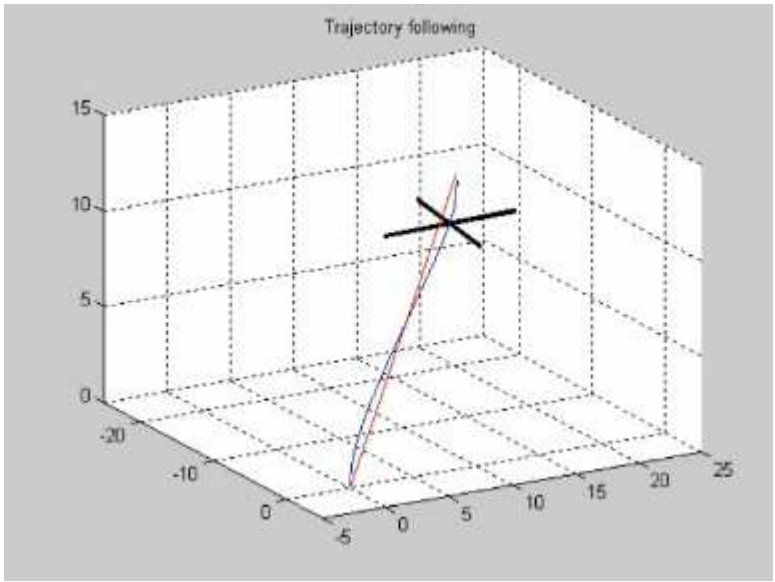


Figure 8.1 : snapshot of a video showing the quadrotor response to a step input

The video lasts 20 seconds like the simulation. The quadrotor has been widened (x10) so that it's easier to see the consistence of its bank angle. The yaw angle command is

a ramp whose slope equals 0.6 rad/sec. This command has been chosen to show that the decoupling is successfully done.

It can also be imagined that the quadrotor's mission is to inspect the area around itself. In that case, the vehicle needs to rotate to show the area in 360° through its video camera.

Even though it is not convenient to watch a 3D motion on a screen, we can nevertheless notice that the flight attitude is consistent during the 20 seconds. The first two seconds show nicely how the UAV tilts to move towards the desired position.

8.2. Video of trajectory tracking

The Matlab code used to create the video file is in the appendix 5.2. Here is a snapshot of the video:

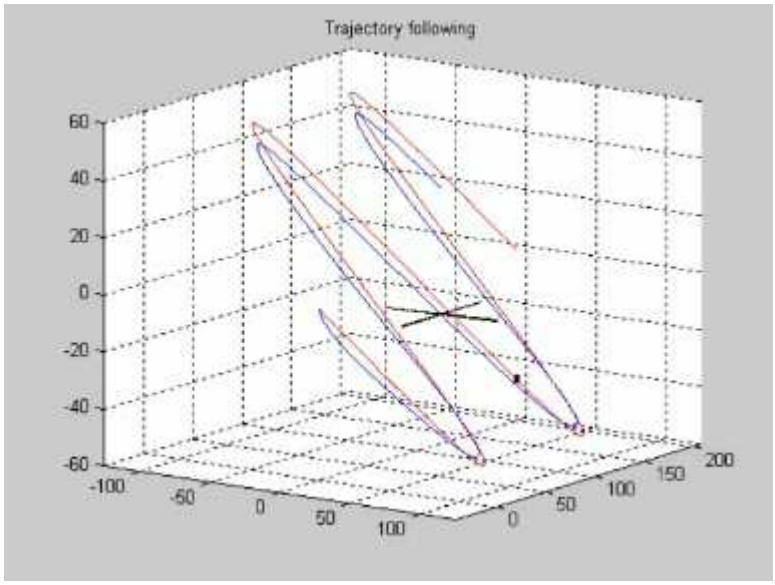


Figure 8.2 : snapshot of a video showing the quadrotor response to a specific trajectory

On this video, which lasts 2 minutes like the simulation, the vehicle has been widened by a factor 80. The commands for x, y and z are generated with the Matlab functions used in the previous chapter. The period of x and z is 60 seconds. That's why the

input trajectory is not very precisely followed. The command for the yaw angle is constant and equalled to zero.

As the inputs for x, y and z vary with time, it has been decided to include in the video the location of the command at each time step. The position of the command is represented by a small black mark. Thus, we can watch and understand better the effects of the delay between the control command and the UAV position.

Once again, this video shows a consistent flight attitude.

8.3. Comments on the full non linear Simulink model

Consequently, the full non linear model used in this project can be partially validated. Indeed, the vehicle dynamics seem to be very close to the reality. However, a real experiment on the Draganflyer X-pro is still required to validate the rotor dynamics.

9. Conclusion and Future Work

This report showed that a mathematical approach can only give satisfactory results with the theoretical model but not with the full non linear model. In order to achieve good performances on the control of the non linear model, an exhaustive comprehension of the UAV dynamics was required. This understanding enabled justified assumptions to be made. The aim of stepping back from the equations was to build a physically feasible model valid for various initial conditions. This way of proceeding corresponds to the engineering approach and it has led to optimal performances.

However, because of voltage limitations, the control law used for trajectory tracking has not been completely optimized. Indeed, as it can be noticed on figures 5.8 and 5.9, small displacements can be flown more rapidly. A settling time of 2 seconds is achievable (with the control law established in 5.3.2) as long as the command inputs don't make the actuators saturate. With these dynamic features, a trajectory input varying at 1 rad/sec would still be correctly tracked.

Thus, it can be considered as future work to optimize the control law by making feedback gain values depend on the command inputs. This would enable the Draganflyer X-pro to take advantage of its available power as often as possible.

Because too much time has been spent designing control laws with a bad rotor model, two different objectives have not been completed. These objectives are:

- ✓ Designing H infinity controller
- ✓ Validating the full non linear model through experiments on the real Draganflyer X-pro

These two tasks can be studied in a future project.

Besides, the robustness of the designed control system can still be enhanced. Indeed, the effects of sensor and actuator failures have been investigated and this study gave rise to the following comments:

- ✓ Actuator failure: as the four propellers of the Draganflyer X-pro have only two blades, if one blade breaks down, the provided thrust is halved. Thus, the way of modelling this failure in Simulink is to halve one of the four voltage inputs. The upper limit for this voltage goes from 12V to 6V. The problem is that 4 x 7.3 V is required to hover. Thus, we cannot figure out the problem of an actuator failure because of gravity requirements.
- ✓ Sensor failure: all the sensors are necessary to control the quadrotor's position. If a sensor failure occurs, the only solution is to implement a Kalman filter (Linear Quadratic Estimator) to estimate the signal and get a stable quadrotor. The implementation of Kalman filters to estimate data from sensors can be suggested in a future thesis as well.

To conclude, the dynamic performances and robustness of the designed closed loop system are limited by the Draganflyer X-pro's power supply. As it could have been intuitively foreseen, stability and robustness mainly depend on the upper limit of the rotor voltages.

References

[1] Benallegue, A., Mokhtari, A. and Fridman, L. (2006), "Feedback linearization and high order sliding mode observer for a quadrotor UAV", *Proceedings of the 2006 International Workshop on Variable Structure Systems*, June 2006, Alghero, Italy, pp. 365.

[2] Bouabdallah, S., Murrieri, P. and Siegwart, R. (2005), "Towards autonomous indoor micro VTOL", *Autonomous Robots*, [Online], vol. 18, no. March, 2005, pp. 14/03/2007 available at: http://www.springerlink.com/content/llug3r141ll85726/fulltext.pdf.

[3] Bouabdallah, S., Murrieri, P. and Siegwart, R. (2004), "Design and control of an indoor micro quadrotor", *2004 IEEE International Conference on Robotics and Automation*, April 2004, New Orleans, pp. 4393.

[4] Bouabdallah, S., Noth, A. and Siegwart, R. (2004), *PID vs LQ control techniques applied to an indoor micro quadrotor*, Swiss Federal Institute of Technology.

[5] Castillo, P., Lozano, R. and Dzul, A., (2005), *Stabilization of a Mini Rotorcraft with Four Rotors*, IEEE Control Systems Magazine.

[6] Chen, M. and Huzmezan, M. (2003), "A combined MBPC/2 dof $H\infty$ controller for a quad rotor UAV", *AIAA Atmospheric Flight Mechanics Conference*, 2003, Austin, Texas.

[7] Chen, M. and Huzmezan, M. (2003), "A Simulation Model and $H\infty$ Loop shaping Control of a Quad Rotor Unmanned Air Vehicle", *Proceedings of MS03 Conference*, 2003, Palm Springs, California.

[8] Cooke, A.K., Cowling, I.D., Erbsloeh, S.D. and Whidborne, J.F., "Low cost system design and development towards an autonomous rotor vehicle", *22nd International Conference on Unmanned Air Vehicle Systems*, April 2007, Bristol, UK. To be presented.

[9] Cowling, I.D., Whidborne, J.F. and Cooke, A.K., "MBPC for autonomous operation of a quadrotor air vehicle", *21st International Conference on Unmanned Air Vehicle Systems*, April 2006, Bristol, UK, pp 35.1-35.9.

[10] Cowling, I.D., Whidborne, J.F. and Cooke, A.K., "Optimal trajectory planning and LQR", *Proc. UKACC Int. Conf. Control 2006 (ICC2006)*, September 2006, Glasgow, UK.

[11] Hoffmann, G. M., Rajnarayan, D. G., Waslander, S. L., Dostal, D., Jang, J. S. and Tomlin, C. J. (2004), "The stanford testbed of autonomous rotorcraft for multi agent control", *Digital Avionics Systems Conference*, Vol. 2, 2004, pp. 12.E.4- 121-10.

[12] McKerrow, P. (2004), "Modelling the Draganflyer four rotor helicopter", *2004 IEEE International Conference on Robotics and Automation*, April 2004, New Orleans, pp. 3596.

[13] Mokhtari, A. and Benallegue, A. (2004), "Dynamic feedback controller of Euler angles and wind parameters estimation for a quadrotor unmanned aerial vehicle", *2004 IEEE international conference on robotics and automation*, April 2004, New Orleans, pp. 2359.

[14] Mokhtari, A., Benallegue, A. and Daachi, B. (2005), "Robust feedback linearization and $GH_{infinity}$ controller for a quadrotor unmanned aerial vehicle", *2005 IEEE International Conference on Intelligent Robots and Systems*, August 2005, Canada, pp. 1009.

[15] Pounds, P., Mahony, R., Hynes, P. and Roberts, J. (2002), "Design of a four rotor aerial robot", *Proc. 2002 Australasian Conference on Robotics and Automation*, November 2002, Auckland, pp. 145.

[16] Tayebi, A. and McGilvray, S. (May 2006), "Attitude stabilization of a VTOL quadrotor aircraft", *ieee transactions on control systems technology*, [Online], vol. 14, no. May 2006, pp. 14/03/2007 available at: http://ieeexplore.ieee.org/iel5/87/34103/01624481.pdf?arnumber=1624481.

[17] Tayebi, A. and McGilvray, S. (2004), "Attitude stabilization of a four rotor aerial robot", *43rd IEEE Conference on Decision and Control*, December 2004, Bahamas, pp. 1216.

[18] Voos, H. (2006), "Nonlinear state dependent Riccati equation Control of a quadrotor UAV", *2006 IEEE International Conference on Control Applications*, October 2006, Munich, pp. 2547.

[19] Waslander, S. L., Hoffmann, G. M., Jang, J. S. and Tomlin, C. J. (2005), "Multi agent quadrotor testbed control design integral sliding mode vs reinforcement learning", *2005 IEEE International Conference on Intelligent Robots and Systems*, August 2005, Canada, pp. 468.

Appendices

Appendix 1.1: Decoupling inputs

Mathematical approach:

```
function
Uinit=decoupling_inputs_global(z_fourth_deriv,y_fourth_deriv,x_fourt
h_deriv,psi_dotdot,phi,theta,psi,zdotdot)

m=2.353598;
g=9.81;
Ix=0.1676;
Iy=0.1686;
Iz=0.29743;
l=0.5;

u1=m*(g-zdotdot)/(cos(theta)*cos(phi));

E=[-cos(theta)*cos(phi)/m
cos(psi)*cos(theta)*sin(phi)*u1/(Ix*m)+sin(psi)*sin(theta)*u1/(Ix*m)
-
sin(psi)*cos(theta)*sin(phi)*u1/(Iy*m)+cos(psi)*sin(theta)*u1/(Iy*m)
0 ;
   sin(phi)/m cos(psi)*cos(phi)*u1/(Ix*m) -
sin(psi)*cos(phi)*u1/(Iy*m) 0 ;
   -sin(theta)*cos(phi)/m cos(psi)*sin(theta)*sin(phi)*u1/(Ix*m)-
sin(psi)*cos(theta)*u1/(Ix*m) -
sin(psi)*sin(theta)*sin(phi)*u1/(Iy*m)-cos(psi)*cos(theta)*u1/(Iy*m)
0 ;
   0 sin(psi)*tan(phi)/Ix cos(psi)*tan(phi)/Iy 1/Iz ];
Ei=inv(E);
Uinit=Ei*[z_fourth_deriv ; y_fourth_deriv ; x_fourth_deriv ;
psi_dotdot];
```

Appendix 1.2: Root loci for mathematical approach

They are the same for x, y and z control. Thus, we give in this appendix only the root loci used for z control and psi control.

Control of z component:

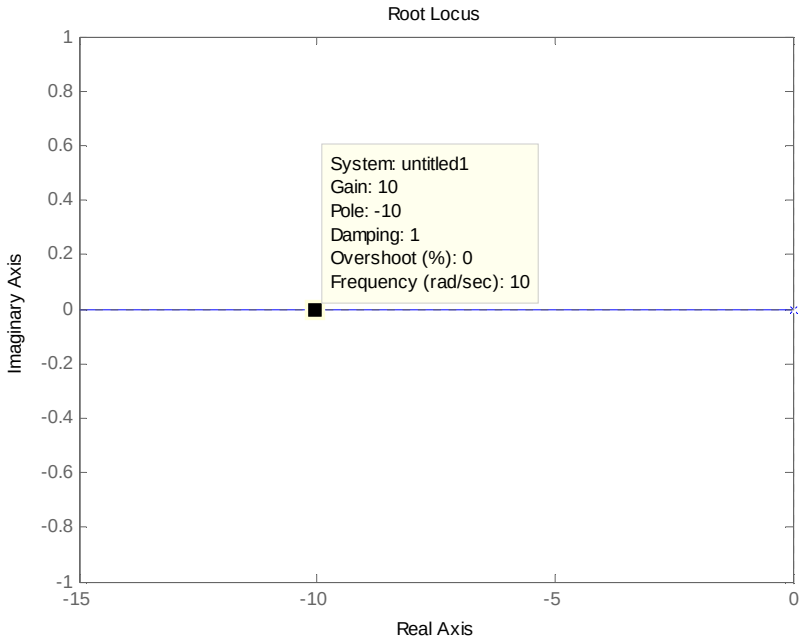


Figure 0.1 : root locus of z̈ feedback on the first input (mathematical approach)

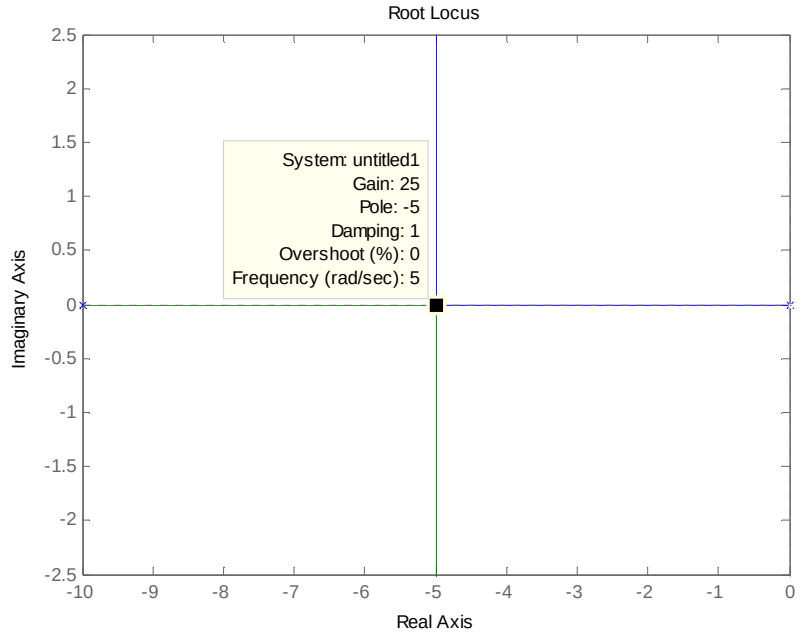


Figure 0.2 : root locus of z̈ feedback on the first input (mathematical approach)

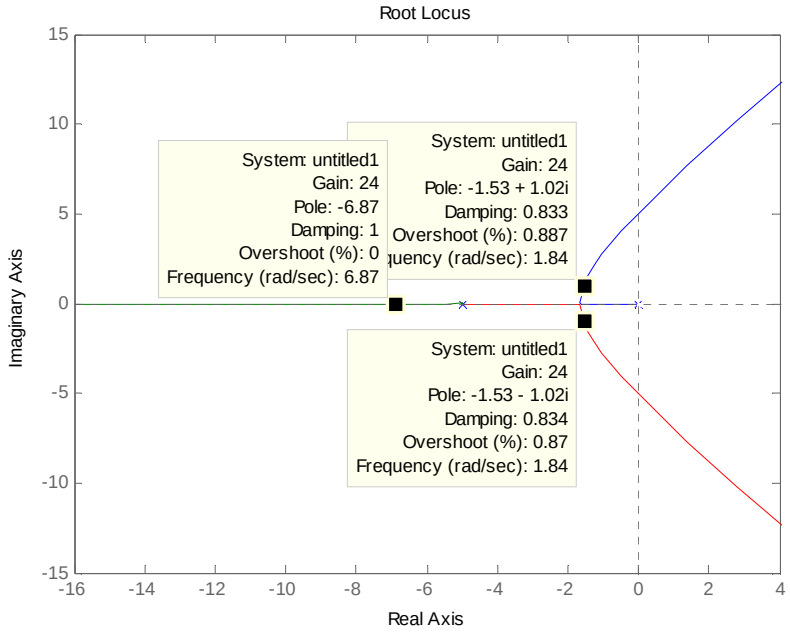


Figure 0.3 : root locus of ż feedback on the first input (mathematical approach)

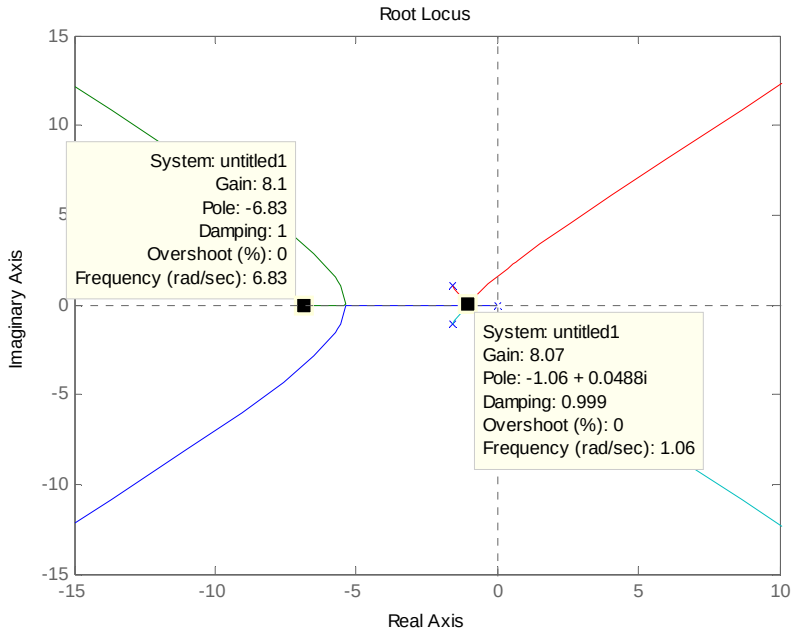


Figure 0.4 : root locus of z feedback on the first input (mathematical approach)

Control of yaw angle:

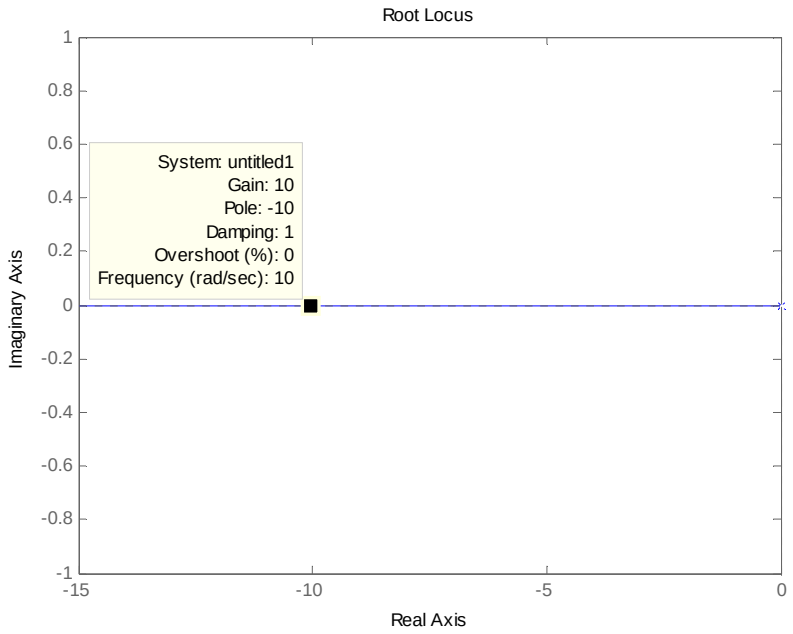


Figure 0.5 : root locus of $\dot{\psi}$ feedback on the fourth input (mathematical approach)

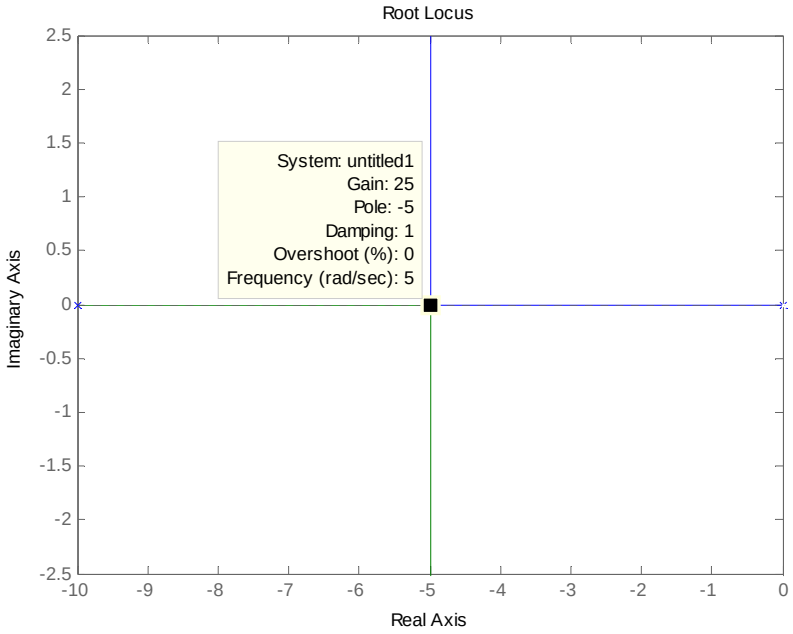


Figure 0.6 : root locus of ψ feedback on the fourth input (mathematical approach)

Appendix 1.3: Mathematical design of PID

Matlab code used to create the mathematical closed loop system:

```
m=2.353598;
g=9.81;
Ix=0.1676;
Iy=0.1686;
Iz=0.29743;
l=0.5;

phi=0*pi/180;
theta=0*pi/180;
psi=0*pi/180;
u1=m*g/(cos(theta)*cos(phi));

Amoving=[0 0 0 1 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 1 0 0 0 0 0 0 0 0 0 ;
0 0 0 0 0 1 0 0 0 0 0 0 0 0 ;
         0 0 0 0 0 0 1 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ;
0 0 0 0 0 0 0 0 1 0 0 0 0 0 ;
         0 0 0 0 0 0 0 0 0 1 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 1 0 0 0 ;
0 0 0 0 0 0 0 0 0 0 0 1 0 0 ;
         0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;
         0 0 0 0 0 0 0 0 0 0 0 0 0 1 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0
];
Bmoving=[0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ;
0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; -sin(theta)*cos(phi)/m
cos(psi)*sin(theta)*sin(phi)*u1/(Ix*m)-sin(psi)*cos(theta)*u1/(Ix*m)
-sin(psi)*sin(theta)*sin(phi)*u1/(Iy*m)-
cos(psi)*cos(theta)*u1/(Iy*m) 0 ; sin(phi)/m
cos(psi)*cos(phi)*u1/(Ix*m) -sin(psi)*cos(phi)*u1/(Iy*m) 0 ; -
cos(theta)*cos(phi)/m
cos(psi)*cos(theta)*sin(phi)*u1/(Ix*m)+sin(psi)*sin(theta)*u1/(Ix*m)
-
sin(psi)*cos(theta)*sin(phi)*u1/(Iy*m)+cos(psi)*sin(theta)*u1/(Iy*m)
0 ; 0 0 0 0 ; 0 sin(psi)*tan(phi)/Ix cos(psi)*tan(phi)/Iy 1/Iz ];
Cmoving=[1 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ;
0 0 1 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 1 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0
1 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 1 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 1 0
0 0 0 0 0 0 ; 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 1 0 0 0
0 0 ; 0 0 0 0 0 0 0 0 0 1 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 1 0 0 0 ; 0
0 0 0 0 0 0 0 0 0 1 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 1 0 ; 0 0 0 0 0
0 0 0 0 0 0 0 0 1 ];
Dmoving=0;
Smoving=ss(Amoving,Bmoving,Cmoving,Dmoving);
Smoving=tf(Smoving);

Am=Amoving;
E=[-cos(theta)*cos(phi)/m
cos(psi)*cos(theta)*sin(phi)*u1/(Ix*m)+sin(psi)*sin(theta)*u1/(Ix*m)
-
sin(psi)*cos(theta)*sin(phi)*u1/(Iy*m)+cos(psi)*sin(theta)*u1/(Iy*m)
0 ;
```

```
   sin(phi)/m cos(psi)*cos(phi)*u1/(Ix*m) -
sin(psi)*cos(phi)*u1/(Iy*m) 0 ;
   -sin(theta)*cos(phi)/m cos(psi)*sin(theta)*sin(phi)*u1/(Ix*m)-
sin(psi)*cos(theta)*u1/(Ix*m) -
sin(psi)*sin(theta)*sin(phi)*u1/(Iy*m)-cos(psi)*cos(theta)*u1/(Iy*m)
0 ;
   0 sin(psi)*tan(phi)/Ix cos(psi)*tan(phi)/Iy 1/Iz ];
Ei=inv(E);
Bm=Bmoving*Ei;
Cm=Cmoving;
Dm=Dmoving;
Sm=tf(ss(Am,Bm,Cm,Dm));

Sm1=minreal(feedback(Sm,10,2,11));
Sm2=minreal(feedback(Sm1,25,2,8));
Sm3=minreal(feedback(Sm2,24.07,2,5));
Sm4=minreal(feedback(Sm3,8.07,2,2));

Sm5=minreal(feedback(Sm4,10,3,10));
Sm6=minreal(feedback(Sm5,25,3,7));
Sm7=minreal(feedback(Sm6,24.07,3,4));
Sm8=minreal(feedback(Sm7,8.07,3,1));

Sm9=minreal(feedback(Sm8,10,1,12));
Sm10=minreal(feedback(Sm9,25,1,9));
Sm11=minreal(feedback(Sm10,24.07,1,6));
Sm12=minreal(feedback(Sm11,8.07,1,3));

Sm13=minreal(feedback(Sm12,10,4,14));
Sm14=minreal(feedback(Sm13,25,4,13));
```

Appendix 1.4: Generating the state space system

Matlab code used to generate A, B and C matrices for linear simulation:

```
function Matrix=A_generator4(phi,theta,psi,zdotdot)
m=2.353598;
g=9.81;
Ix=0.1676;
Iy=0.1686;
Iz=0.29743;
l=0.5;
u1=m*(g-zdotdot)/(cos(theta)*cos(phi));

Amoving=[0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 ;
    0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 ;
    0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 ;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 (1/0.4895)*sin(theta)*cos(phi)/m
cos(psi)*sin(theta)*sin(phi)*u1/(Ix*m)-sin(psi)*cos(theta)*u1/(Ix*m)
-sin(psi)*sin(theta)*sin(phi)*u1/(Iy*m)-
cos(psi)*cos(theta)*u1/(Iy*m) 0 (1.873/0.4895)*sin(theta)*cos(phi)/m
0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -(1/0.4895)*sin(phi)/m
cos(psi)*cos(phi)*u1/(Ix*m) -sin(psi)*cos(phi)*u1/(Iy*m) 0 -
(1.873/0.4895)*sin(phi)/m 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(1/0.4895)*cos(theta)*cos(phi)/m
cos(psi)*cos(theta)*sin(phi)*u1/(Ix*m)+sin(psi)*sin(theta)*u1/(Ix*m)
-
sin(psi)*cos(theta)*sin(phi)*u1/(Iy*m)+cos(psi)*sin(theta)*u1/(Iy*m)
0 (1.873/0.4895)*cos(theta)*cos(phi)/m 0 0 0 0 0 0 ;
    0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 sin(psi)*tan(phi)/Ix cos(psi)*tan(phi)/Iy 1/Iz 0 0
0 0 0 0 0 0 ;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ; 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1/0.314 0
0 0 0 0 0 0 0 ;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1/0.4895 0 0 0 -1.873/0.4895 0 0 0
0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1/0.8696 0 0 0 -0.4018/0.8696
0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1/0.8696 0 0 0 -
0.4018/0.8696 0 0 0 0 ;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 ; 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 ;
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 cos(psi)/Ix -sin(psi)/Iy 0 0 0 0 0
0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 sin(psi)/(cos(phi)*Ix)
cos(psi)/(cos(phi)*Iy) 0 0 0 0 0 0 0 0 ];

Matrix=Amoving;
```

```
function Matrix=B_generator4(phi,theta,psi,zdotdot)
m=2.353598;
g=9.81;
Ix=0.1676;
Iy=0.1686;
Iz=0.29743;
l=0.5;
u1=m*(g-zdotdot)/(cos(theta)*cos(phi));

Bmoving=[0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ;
0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; -(1/0.4895)*sin(theta)*cos(phi)/m 0 0
0 ; (1/0.4895)*sin(phi)/m 0 0 0 ; -(1/0.4895)*cos(theta)*cos(phi)/m
0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0
1/0.314 ; 1/0.4895 0 0 0 ; 0 1/0.8696 0 0 ; 0 0 1/0.8696 0 ; 0 0 0 0
; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ];

Matrix=Bmoving;



function Matrix=C_generator4(phi,theta,psi,zdotdot)
m=2.353598;
g=9.81;
Ix=0.1676;
Iy=0.1686;
Iz=0.29743;
l=0.5;
u1=m*(g-zdotdot)/(cos(theta)*cos(phi));

Cmoving=[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 ; 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 ];

Matrix=Cmoving;
```

Initialising the linear simulation:

```
xInitial = [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];
```

Appendix 1.5: From body to Euler angular rates

Matlab function used to transform body angular rates (p, q, r) into Euler angular rates (phi, theta, psi):

```
function Euler_rates=body_rates_to_euler(phi,theta,psi,p,q,r)
phidot=p*cos(psi)-q*sin(psi);
thetadot=p*sin(psi)/cos(phi)+q*cos(psi)/cos(phi);
psidot=p*sin(psi)*tan(phi)+q*cos(psi)*tan(phi)+r;

Euler_rates=[phidot,thetadot,psidot];
```

Appendix 2.1: Decoupling inputs (simplified)

Matlab function used to decouple the inputs between phi, theta, psi and u2, u3, u4:

```
function Uinit=decoupling_inputs(phi,psi,u2_new,u3_new,u4_new)
    Ix=0.1676;
    Iy=0.1686;
    Iz=0.29743;
    u2_init=cos(psi)*u2_new+sin(psi)*cos(phi)*Ix*u3_new/Iy;
    u3_init=-sin(psi)*Iy*u2_new/Ix+cos(psi)*cos(phi)*u3_new;
    u4_init=-sin(phi)*Iz*u3_new/Iy+u4_new;
    Uinit=[u2_init u3_init u4_init];
```

Appendix 2.2: Root loci of the inner loop

They are the same for phi and theta control. We choose to give the root loci of phi control, psi control and altitude control.

Control of roll angle:

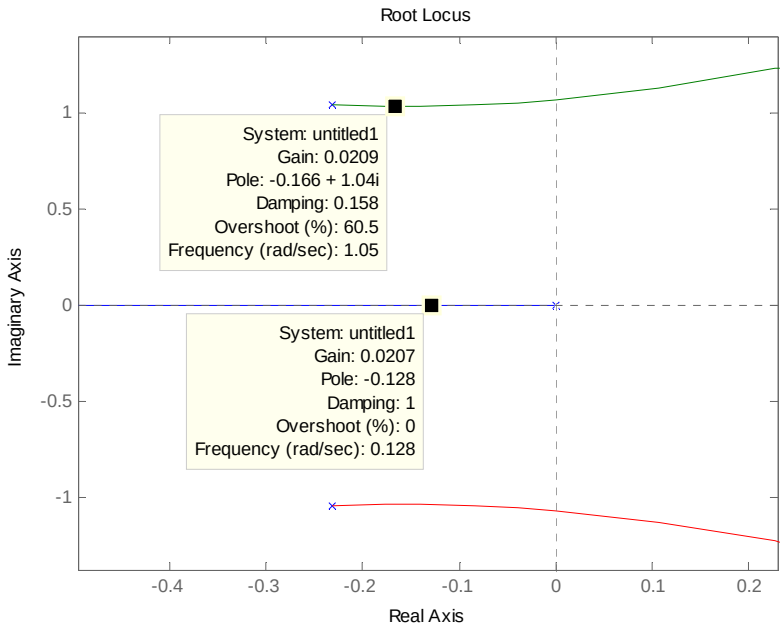


Figure 0.7 : root locus of $\dot{\phi}$ feedback on the second input (Towards an engineering approach)

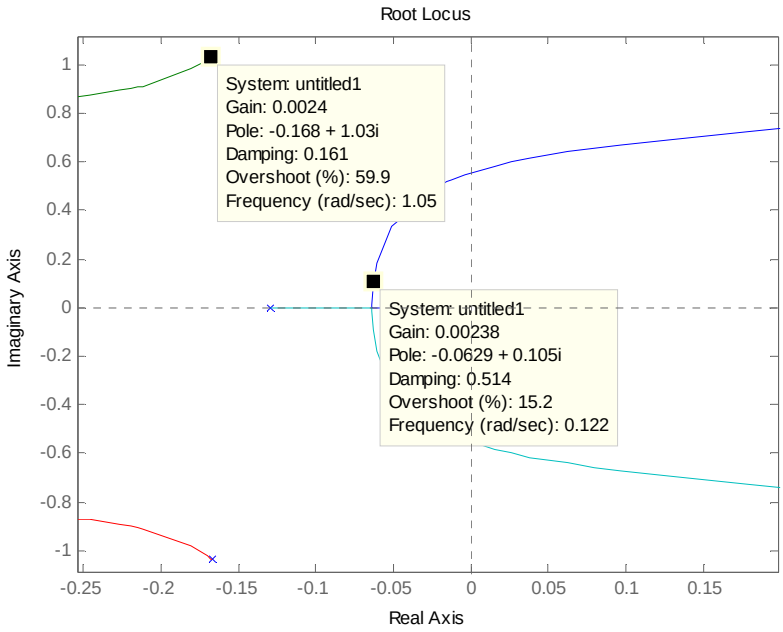


Figure 0.8 : root locus of ϕ feedback on the second input (Towards an engineering approach)

Control of yaw angle:

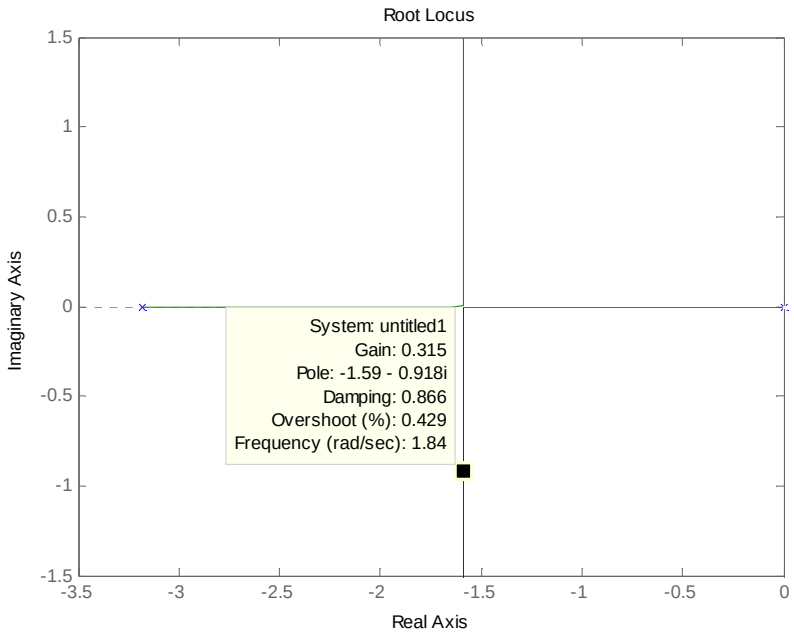


Figure 0.9 : root locus of $\dot{\psi}$ feedback on the fourth input (Towards an engineering approach)

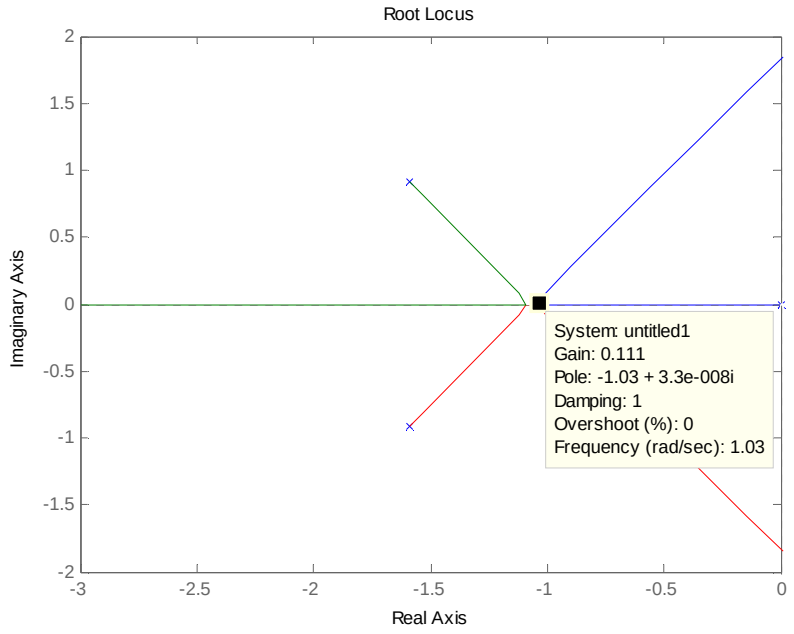


Figure 0.10 : root locus of ψ feedback on the fourth input (Towards an engineering approach)

Control of altitude:

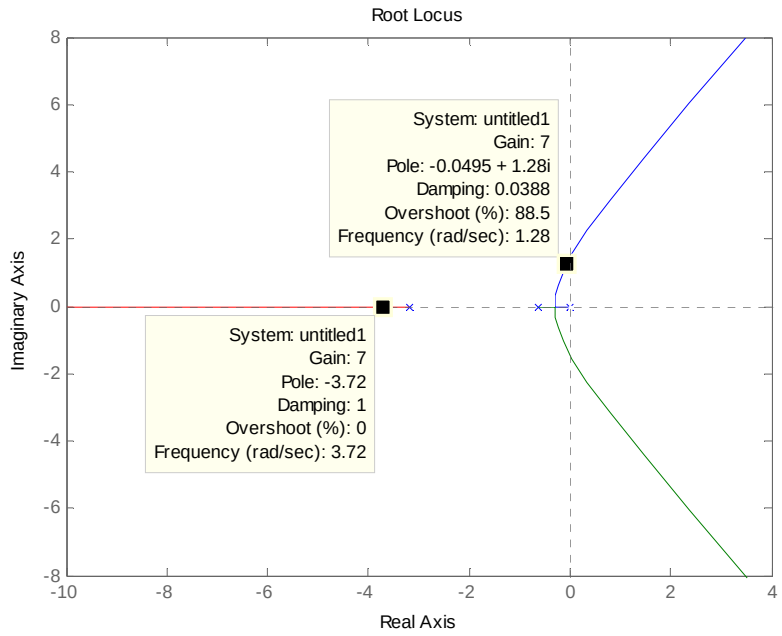


Figure 0.11 : root locus of ż feedback on the first input (Towards an engineering approach)

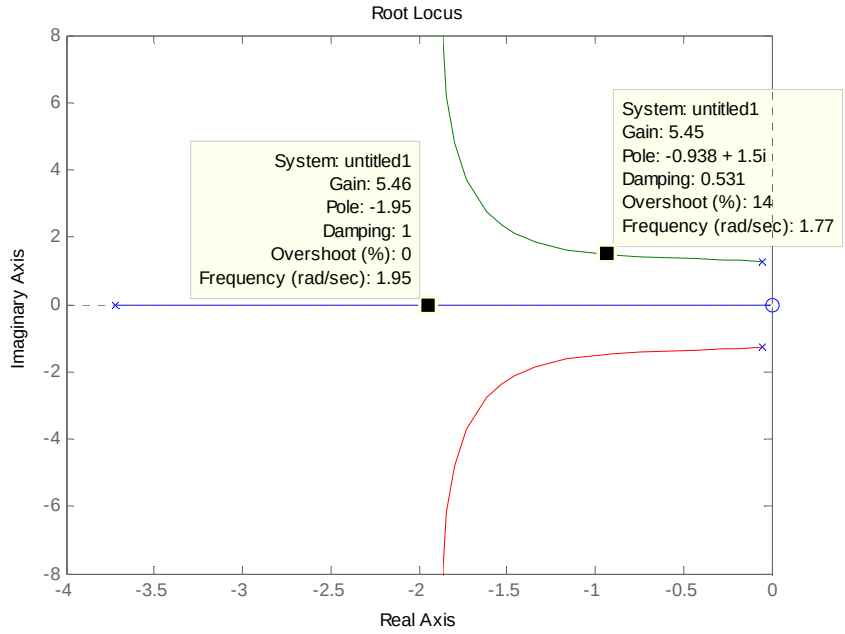


Figure 0.12 : root locus of z̈ feedback on the first input (Towards an engineering approach)

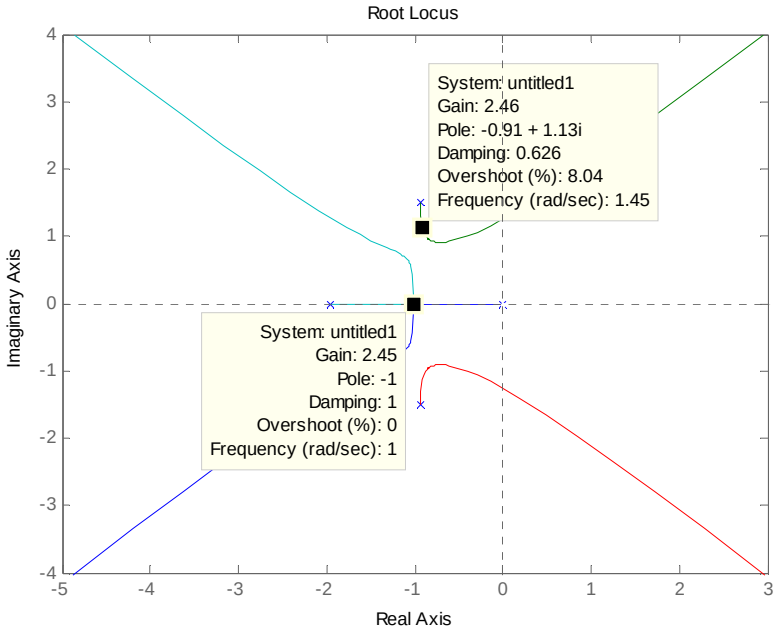


Figure 0.13 : root locus of z feedback on the first input (Towards an engineering approach)

Appendix 2.3: Design of PID (inner and outer loops)

Matlab code of the control law design:

- ✓ Translational components (outer loop)

```
m=2.353598;
g=9.81;
u1=m*g;

A=[0 0 1 0 ; 0 0 0 1 ; 0 0 0 0 ; 0 0 0 0 ];
B=[0 0 ; 0 0 ; 0 -u1/m ; u1/m 0];
C=[1 0 0 0 ; 0 1 0 0 ; 0 0 1 0 ; 0 0 0 1 ; 0 0 0 0 ; 0 0 0 0 ];
D=[0 0 ; 0 0 ; 0 0 ; 0 0 ; 0 -u1/m ; u1/m 0];
S=tf(ss(A,B,C,D));
```

- ✓ Angular subsystem (inner loop) and connection between the two subsystems

```
m=2.353598;
g=9.81;
Ix=0.1676;
Iy=0.1686;
Iz=0.29743;
l=0.5;

phi=0*pi/180;
theta=0*pi/180;
psi=0*pi/180;
u1=m*g/(cos(theta)*cos(phi));

Amoving=[0 1 0 0 0 0 0 0 0 0 0 0 0 0 ;
    0 0 0 0 0 0 0 0 -1/m 0 0 0 0 0 0 ;
    0 0 0 0 0 1 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 1 0 0 0 0 0 0 0 ;
0 0 0 0 0 0 0 1 0 0 0 0 0 0 ;
    0 0 0 0 0 0 0 0 0 cos(psi)/Ix -sin(psi)/Iy 0 0 0 0 ; 0 0 0 0 0 0
0 0 0 sin(psi)/(cos(phi)*Ix) cos(psi)/(cos(phi)*Iy) 0 0 0 0 ; 0 0 0
0 0 0 0 0 0 0 sin(psi)*tan(phi)/Ix cos(psi)*tan(phi)/Iy 1/Iz 0 0 0 ;
    0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 ;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 ; 0 0 0 0 0 0 0 0 0 0 0 0 -1/0.314 0 0 0
;
    0 0 0 0 0 0 0 0 -1/0.4895 0 0 0 -1.873/0.4895 0 0 ; 0 0 0 0 0 0
0 0 0 -1/0.8696 0 0 0 -0.4018/0.8696 0 ; 0 0 0 0 0 0 0 0 0 0 -
1/0.8696 0 0 0 -0.4018/0.8696 ];
Bmoving=[0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ;
0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 1/0.314 ;
1/0.4895 0 0 0 ; 0 1/0.8696 0 0 ; 0 0 1/0.8696 0 ];
Cmoving=[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 ; 0 0 cos(theta)*sin(phi)*u1/m sin(theta)*cos(phi)*u1/m 0 0 0 0 -
cos(theta)*cos(phi)/m 0 0 0 0 0 0 ; 0 0 1 0 0 0 0 0 0 0 0 0 0 0 ;
0 0 0 1 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 1 0 0 0 0 0 0 0 0 0 ; 0 0
0 0 0 1 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 1 0 0 0 0 0 0 0 ; 0 0 0 0
0 0 0 1 0 0 0 0 0 0 ];
```

```
Dmoving=0;
Smoving=ss(Amoving,Bmoving,Cmoving,Dmoving);

Am=Amoving;
Bm=Bmoving*[1 0 0 0 ; 0 cos(psi) sin(psi)*cos(phi)*Ix/Iy 0 ; 0 -
sin(psi)*Iy/Ix cos(psi)*cos(phi) 0 ; 0 0 -sin(phi)*Iz/Iy 1];
Cm=Cmoving;
Dm=Dmoving;
Sm=tf(ss(Am,Bm,Cm,Dm));

Sm1=minreal(feedback(Sm,0.0208,2,7));
Sm2=minreal(feedback(Sm1,0.00239,2,4));
Sm3=minreal(feedback(Sm2,0.0208,3,8));
Sm4=minreal(feedback(Sm3,0.00238,3,5));
Sm5=minreal(feedback(Sm4,0.3155,4,9));
Sm6=minreal(feedback(Sm5,0.111,4,6));
Sm7=minreal(feedback(Sm6,-7,1,2));
Sm8=minreal(feedback(Sm7,-5.45,1,3));
Sm9=minreal(feedback(Sm8,-2.45,1,1));

translational;

St=S*Sm9(4:5,:);
St1=minreal(feedback(St,1.412*10^(-5),2,4));
St2=minreal(feedback(St1,4.17*10^(-7),2,2));
St3=minreal(feedback(St2,-1.4*10^(-5),3,3));
St4=minreal(feedback(St3,-4.12*10^(-7),3,1));
```

Appendix 2.4: Root loci of the outer loop

The root loci for x and y control are the same.
Here are the root loci for the control of x component only:

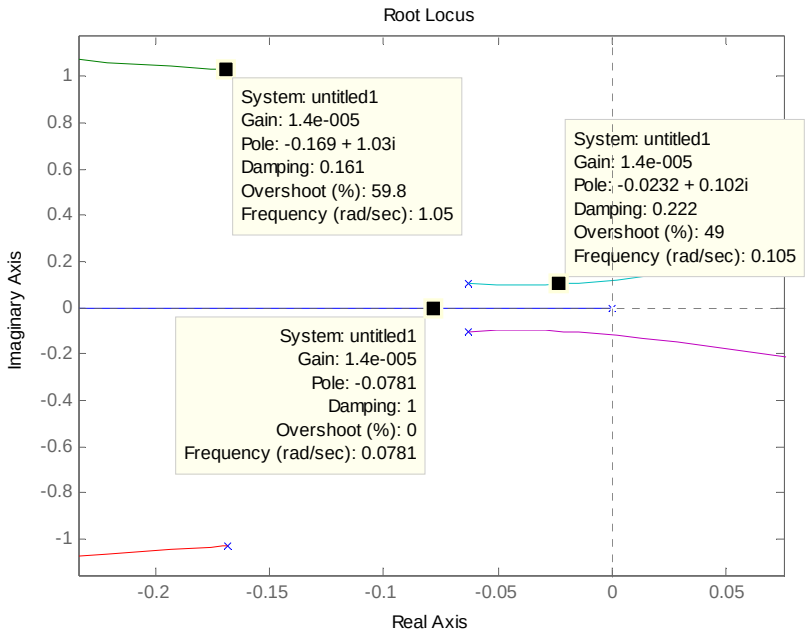


Figure 0.14 : root locus of $\dot{x}$ feedback on the third input (Towards an engineering approach)

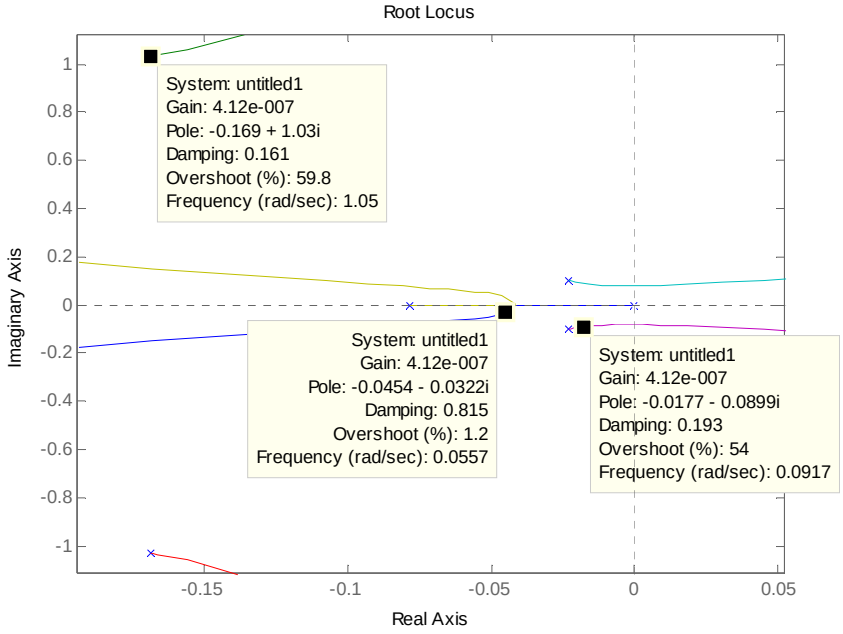


Figure 0.15 : root locus of x feedback on the third input (Towards an engineering approach)

Appendix 2.5: Stabilizer

Matlab function used to stabilize the quadrotor:

```
function V1=interpol_advanced(phi,theta,phidot,thetadot)
    if (phi<0 & phidot<-10^(-6))|(phi>0 & phidot>10^(-6))|(theta<0 &
thetadot<-10^(-6))|(theta>0 & thetadot>10^(-6)),
        V1=0.1;
    elseif (phi>0 & phidot<-10^(-6))|(phi<0 & phidot>10^(-
6))|(theta>0 & thetadot<-10^(-6))|(theta<0 & thetadot>10^(-6)),
        V1=-0.1;
    else
        V1=0
    end
```

Appendix 3.1: Thrust D.C. gain

Wind tunnel test results (no free stream => hover):

Gravity acceleration (m/s^2)	9,81

Supply voltage (volts)	Thrust (kg)	Thrust (N)
1	0,0064	0,062784
2	0,0248	0,243288
3	0,0584	0,572904
4	0,1093	1,072233
5	0,1712	1,679472
6	0,2544	2,495664
7	0,3378	3,313818
8	0,4407	4,323267
9	0,5475	5,370975
10	0,6559	6,434379

Linearization of the relation between the steady state value of the thrust and the applied voltage:

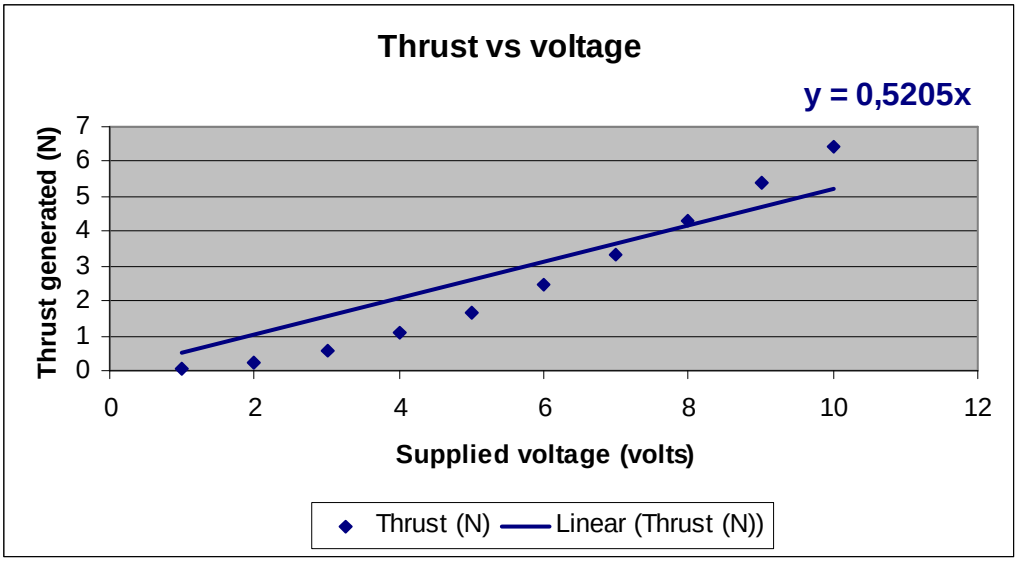


Figure 0.16 : modelling the D.C. gain between thrust and voltage

Appendix 3.2: Design of PID and LQR

Matlab code used to design PID and LQR controllers using the engineering approach:

```
m=2.353598;
g=9.81;
Ix=0.1676;
Iy=0.1686;
Iz=0.29743;
l=0.5;

phi=0;
theta=0;
psi=0;
u1=m*g;

Amoving=[0 0 0 1 0 0 0 0 0 0 0 0 ; 0 0 0 0 1 0 0 0 0 0 0 0 ; 0 0 0 0
0 1 0 0 0 0 0 0 ;
    0 0 0 0 0 0 0 -u1/m 0 0 0 0 ; 0 0 0 0 0 0 u1/m 0 0 0 0 0 ; 0 0 0
0 0 0 0 0 0 0 0 0 ;
    0 0 0 0 0 0 0 0 0 1 0 0 ; 0 0 0 0 0 0 0 0 0 0 1 0 ; 0 0 0 0 0 0
0 0 0 0 0 1 ;
    0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0
0 0 0 0 0 0 ];

Bmoving=[0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; -1/m 0 0
0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 cos(psi)/Ix -sin(psi)/Iy 0 ; 0
sin(psi)/(cos(phi)*Ix) cos(psi)/(cos(phi)*Iy) 0 ; 0
sin(psi)*tan(phi)/Ix cos(psi)*tan(phi)/Iy 1/Iz ];

Cmoving=[1 0 0 0 0 0 0 0 0 0 0 0 ; 0 1 0 0 0 0 0 0 0 0 0 0 ; 0 0 1 0
0 0 0 0 0 0 0 0 ; 0 0 0 1 0 0 0 0 0 0 0 0 ; 0 0 0 0 1 0 0 0 0 0 0 0
; 0 0 0 0 0 1 0 0 0 0 0 0 ; 0 0 0 0 0 0 1 0 0 0 0 0 ; 0 0 0 0 0 0 0 0
1 0 0 0 0 ; 0 0 0 0 0 0 0 0 1 0 0 0 ; 0 0 0 0 0 0 0 0 0 1 0 0 ; 0 0
0 0 0 0 0 0 0 1 0 ; 0 0 0 0 0 0 0 0 0 0 0 1 ];

Dmoving=[0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ;
0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ];

Bmoving=Bmoving*[1 0 0 0 ; 0 cos(psi) sin(psi)*cos(phi)*Ix/Iy 0 ; 0
-sin(psi)*Iy/Ix cos(psi)*cos(phi) 0 ; 0 0 -sin(phi)*Iz/Iy 1];

Smoving=tf(ss(Amoving,Bmoving,Cmoving,Dmoving));


%PID controller

S1=minreal(feedback(Smoving,5,2,10));
S2=minreal(feedback(S1,37.3,2,7));
S3=minreal(feedback(S2,10.92,2,5));
S4=minreal(feedback(S3,10.9,2,2));

S5=minreal(feedback(S4,5,3,11));
S6=minreal(feedback(S5,37.3,3,8));
```

```
S7=minreal(feedback(S6,-10.92,3,4));
S8=minreal(feedback(S7,-10.9,3,1));

S9=minreal(feedback(S8,-10,1,6));
S10=minreal(feedback(S9,-10.6,1,3));

S11=minreal(feedback(S10,3,4,12));
S12=minreal(feedback(S11,7.56,4,9));


%LQR controller

A=Amoving;
R=[0.01 0 0 0 ; 0 0.1 0 0 ; 0 0 0.1 0 ; 0 0 0 0.1];
B=Bmoving*inv(R)*Bmoving';

C=eye(12);
C(1,1)=10;
C(2,2)=10;
C(3,3)=10;
C(9,9)=10;

X=are(A,B,C);
K=inv(R)*Bmoving'*X;

Am_closed=Amoving-Bmoving*K;
Sm_closed=tf(ss(Am_closed, Bmoving,Cmoving,Dmoving));
```

Appendix 3.3: Root loci (engineering approach)

Root loci are the same for x and y. We give only the root loci of y, z and psi control.
Control of y component:

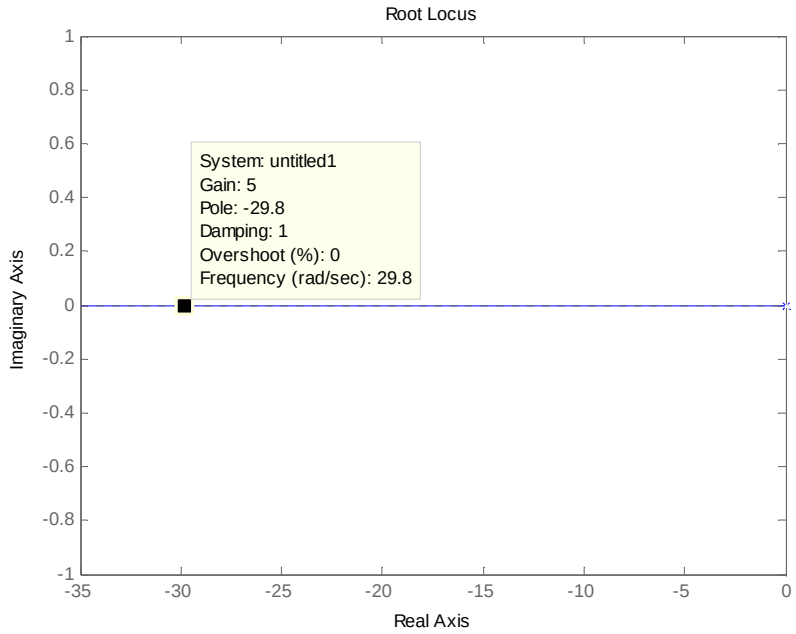


Figure 0.17 : root locus of $\dot{\phi}$ feedback on the second input (Using an engineering approach)

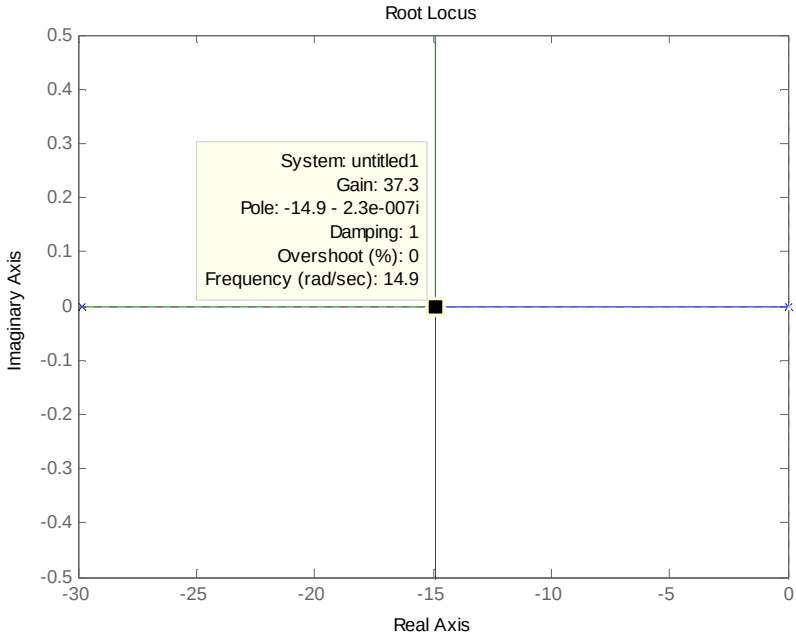


Figure 0.18 : root locus of ϕ feedback on the second input (Using an engineering approach)

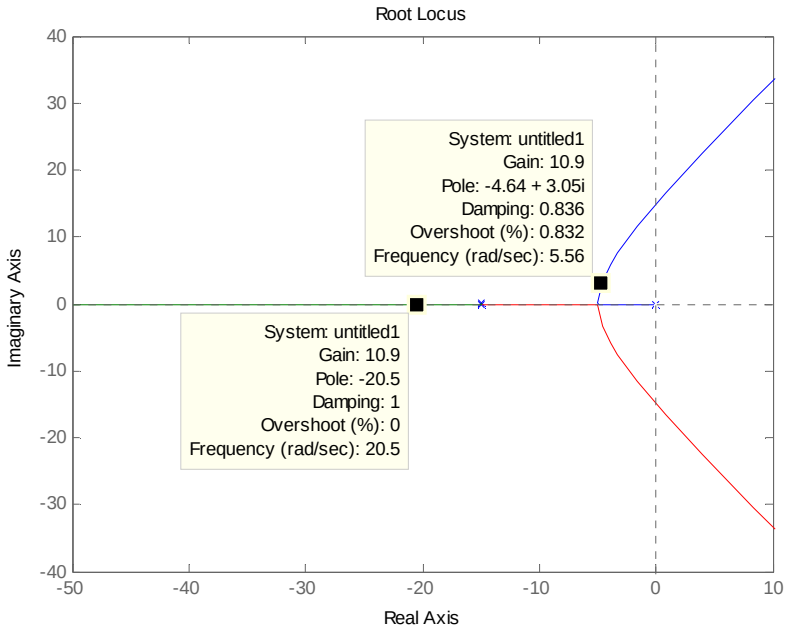


Figure 0.19 : root locus of $\dot{y}$ feedback on the second input (Using an engineering approach)

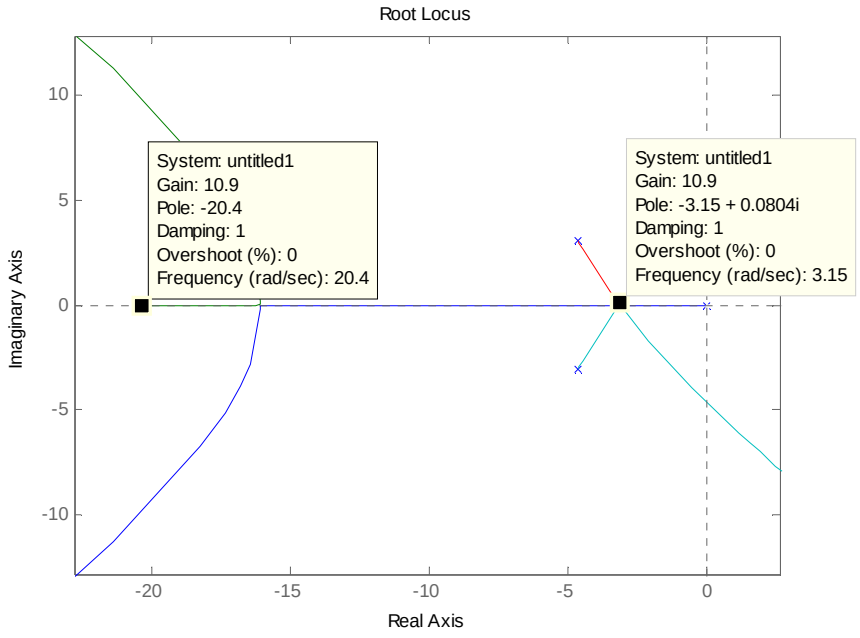


Figure 0.20 : root locus of y feedback on the second input (Using an engineering approach)

Control of z component:

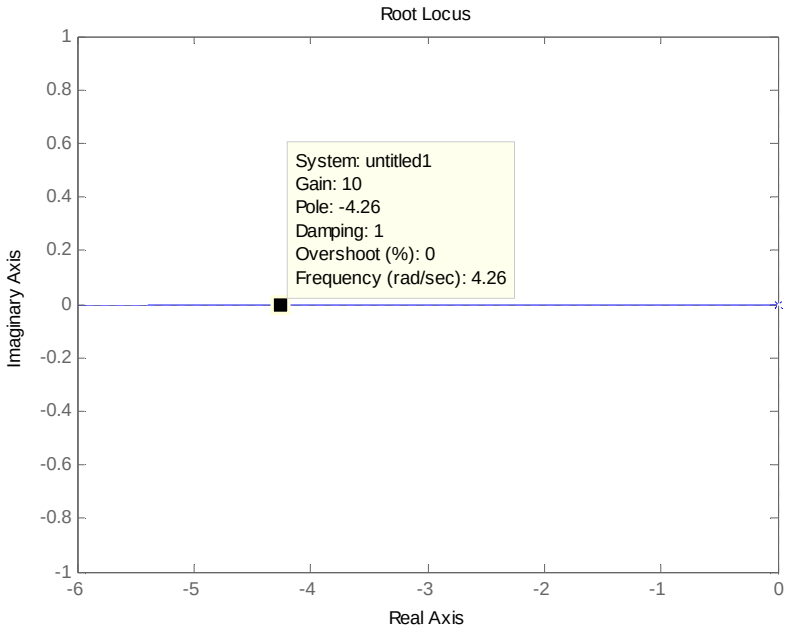


Figure 0.21 : root locus of ż feedback on the first input (Using an engineering approach)

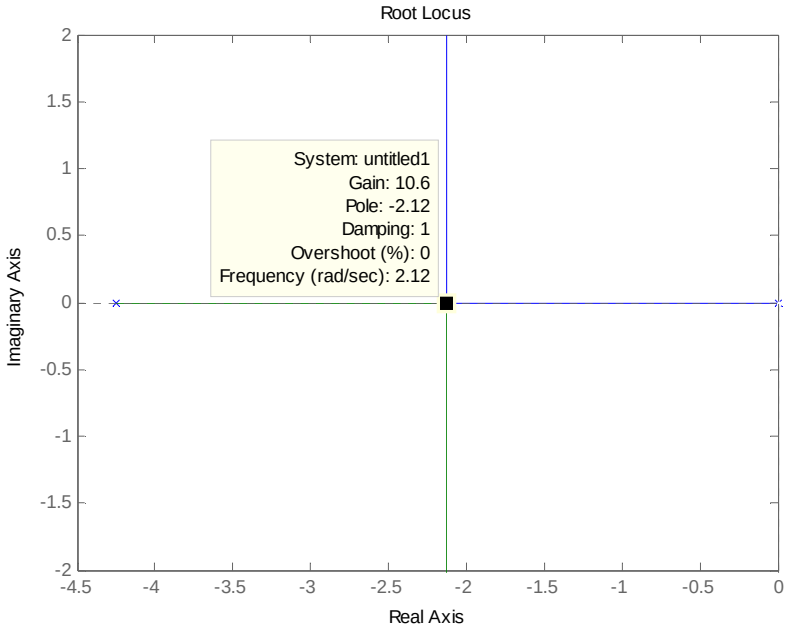


Figure 0.22: root locus of z feedback on the first input (Using an engineering approach)

Control of yaw angle:

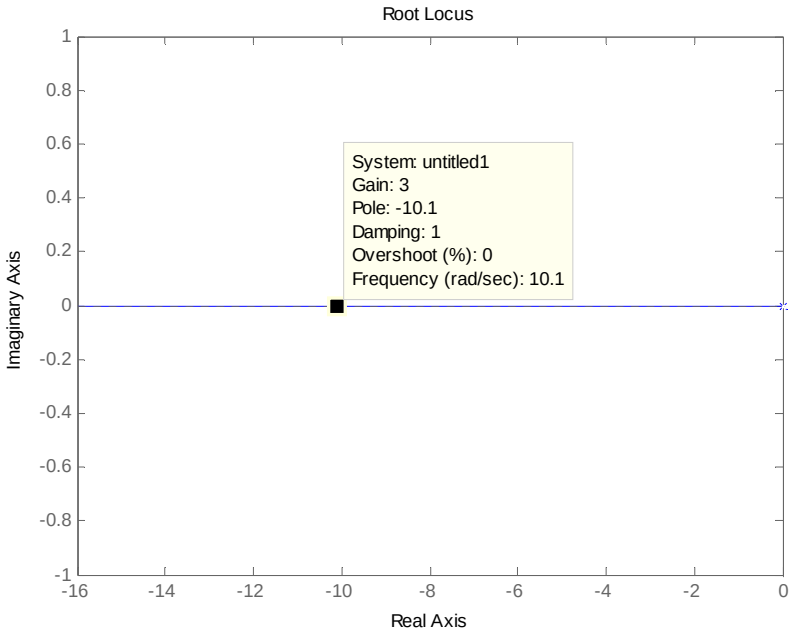


Figure 0.23: root locus of $\dot{\psi}$ feedback on the fourth input (Using an engineering approach)

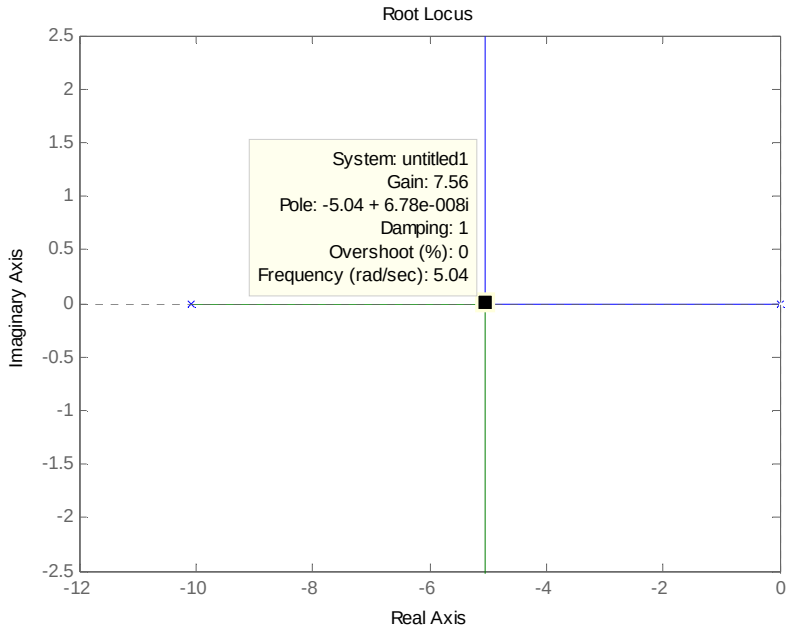


Figure 0.24: root locus of ψ feedback on the fourth input (Using an engineering approach)

Appendix 4.1: Latest version of the Simulink model

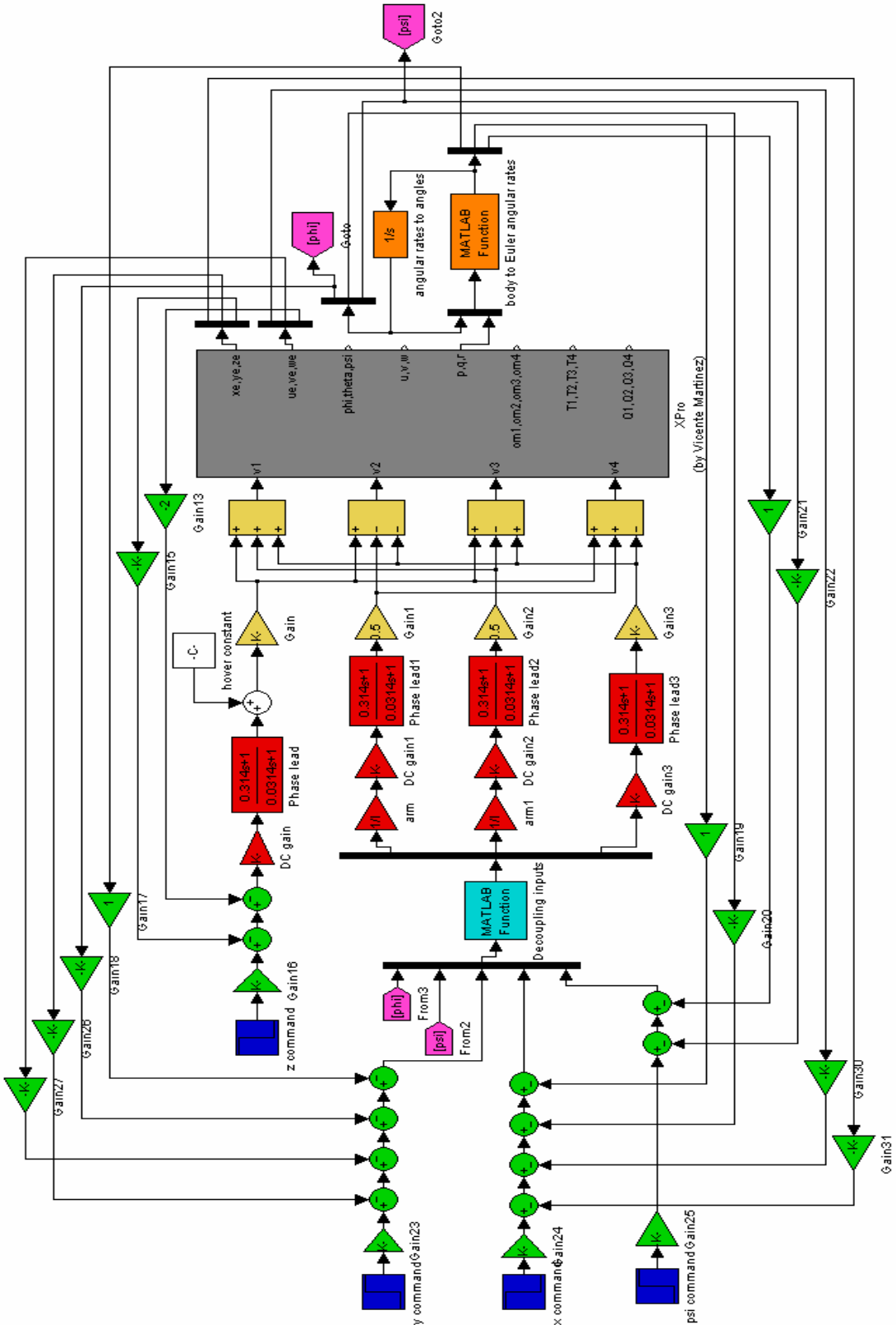


Figure 0.25 : latest version of the Simulink model used for trajectory tracking

Appendix 4.2: Generating trajectories

Matlab functions used to generate a circle in 120 seconds:

Generating x command:

```
function x=x_signal_building(t)
    w=2*pi/120;
    x=cos(w*t)-1;
```

Generating y command:

```
function y=y_signal_building(t)
    w=2*pi/120;
    y=sin(w*t);
```

Generating z command:

```
function z=z_signal_building(t)
    w=2*pi/120;
    z=sin(w*t);
```

Matlab functions used to generate a sinusoidal trajectory with a period equalled to 60 seconds:

Generating x command:

```
function x=x_signal_building2(t)
    w=2*pi/60;
    x=100*sin(w*t);
```

Generating y command:

```
function y=y_signal_building2(t)
    y=1.6667*t;
```

Generating z command:

```
function z=z_signal_building2(t)
    w=2*pi/60;
    z=50*sin(w*t);
```

Matlab code used to plot the simulated results and compare them with the input trajectory :

```
t=0:0.01:120;
w=2*pi/120;
x=cos(w*t)-1;
y=sin(w*t);
z=sin(w*t);
plot3(x,y,z,'r')
hold on
plot3(xpos,ypos,zpos,'m')
```

Appendix 5.1: Video of step response

Matlab code used to create the video file of the step response:

```
clear mex;
t=20;
x_com=-20;
y_com=20;
z_com=-10;
x_init=0;
y_init=0;
z_init=0;
psi_init=0;

mov =
avifile('flight_trajectory.avi','Compression','None','Quality',100,'
fps',10)

video = figure(1)

hold on
plot3(xpos,ypos,-zpos);
grid on
axis([-25 5 -5 25 0 15])
az=60;
el=25;
view(az,el);
hold on
plot3([x_init x_com],[y_init y_com],[-z_init -z_com],'r');

title('Trajectory following');

X=0;
Y=0;
Z=0;
radius=5;

hold on

for j=1:10:t*100+1;
    phi=roll(j);
    theta=pitch(j);
    psi=yaw(j);
    X=xpos(j);
    Y=ypos(j);
    Z=-zpos(j);
    DCM=[sin(theta)*sin(phi)*sin(psi)+cos(psi)*cos(theta)
sin(phi)*sin(theta)*cos(psi)-cos(theta)*sin(psi) sin(theta)*cos(phi)
;
        cos(phi)*sin(psi) cos(psi)*cos(phi) -sin(phi) ;
        cos(theta)*sin(psi)*sin(phi)-sin(theta)*cos(psi)
cos(theta)*sin(phi)*cos(psi)+sin(theta)*sin(psi)
cos(theta)*cos(phi)];
    XYZ1=DCM*[radius ; 0 ; 0];
```

```
    XYZ2=DCM*[0 ; radius ; 0];
    quad_vertices=[X+XYZ1(1) Y+XYZ1(2) Z-XYZ1(3)+0.1 ; X-XYZ1(1) Y-
XYZ1(2) Z+XYZ1(3)+0.1 ; X-XYZ1(1) Y-XYZ1(2) Z+XYZ1(3)-0.1 ;
X+XYZ1(1) Y+XYZ1(2) Z-XYZ1(3)-0.1 ; X+XYZ2(1) Y+XYZ2(2) Z-
XYZ2(3)+0.1 ; X-XYZ2(1) Y-XYZ2(2) Z+XYZ2(3)+0.1 ; X-XYZ2(1) Y-
XYZ2(2) Z+XYZ2(3)-0.1 ; X+XYZ2(1) Y+XYZ2(2) Z-XYZ2(3)-0.1];
    quad_faces=[1 2 3 4 ; 5 6 7 8];
    h1=patch('Vertices',quad_vertices,'Faces',quad_faces);

    M(j)=getframe(video);
    mov=addframe(mov,M(j));
    delete(h1);
end

mov=close(mov);
```

Appendix 5.2: Video of trajectory tracking

Matlab code used to create the video file of the trajectory following:

```
clear mex;

w=2*pi/60;
x=100*sin(w*t);
y=1.6667*t;
z=50*sin(w*t);

mov =
avifile('flight_trajectory.avi','Compression','None','Quality',100,'
fps',10)

video = figure(1)

hold on
plot3(xpos,ypos,-zpos);
grid on
axis([-125 125 -40 210 -60 60])
az=35;
el=15;
view(az,el);
hold on
plot3(x,y,-z,'r')

title('Trajectory following');

X=0;
Y=0;
Z=0;
radius=40;

hold on

for j=1:10:120*100+1;
    phi=roll(j);
    theta=pitch(j);
    psi=yaw(j);
    X=xpos(j);
    Y=ypos(j);
    Z=-zpos(j);
    Xc=x(j);
    Yc=y(j);
    Zc=-z(j);
    DCM=[sin(theta)*sin(phi)*sin(psi)+cos(psi)*cos(theta)
sin(phi)*sin(theta)*cos(psi)-cos(theta)*sin(psi) sin(theta)*cos(phi)
;
        cos(phi)*sin(psi) cos(psi)*cos(phi) -sin(phi) ;
        cos(theta)*sin(psi)*sin(phi)-sin(theta)*cos(psi)
cos(theta)*sin(phi)*cos(psi)+sin(theta)*sin(psi)
cos(theta)*cos(phi)];
    XYZ1=DCM*[radius ; 0 ; 0];
```

```
    XYZ2=DCM*[0 ; radius ; 0];
    quad_vertices=[X+XYZ1(1) Y+XYZ1(2) Z-XYZ1(3)+0.1 ; X-XYZ1(1) Y-
XYZ1(2) Z+XYZ1(3)+0.1 ; X-XYZ1(1) Y-XYZ1(2) Z+XYZ1(3)-0.1 ;
X+XYZ1(1) Y+XYZ1(2) Z-XYZ1(3)-0.1 ; X+XYZ2(1) Y+XYZ2(2) Z-
XYZ2(3)+0.1 ; X-XYZ2(1) Y-XYZ2(2) Z+XYZ2(3)+0.1 ; X-XYZ2(1) Y-
XYZ2(2) Z+XYZ2(3)-0.1 ; X+XYZ2(1) Y+XYZ2(2) Z-XYZ2(3)-0.1];
    quad_faces=[1 2 3 4 ; 5 6 7 8];
    com_vertices=[Xc+1 Yc+1 Zc+1 ; Xc-1 Yc+1 Zc+1 ; Xc-1 Yc-1 Zc+1 ;
Xc+1 Yc-1 Zc+1 ; Xc+1 Yc+1 Zc-1 ; Xc-1 Yc+1 Zc-1 ; Xc-1 Yc-1 Zc-1 ;
Xc+1 Yc-1 Zc-1 ];
    com_faces=[1 2 3 4 ; 5 6 7 8 ; 1 2 5 6 ; 3 4 7 8 ; 2 3 6 7 ; 1 4
5 8];
    h1=patch('Vertices',quad_vertices,'Faces',quad_faces);
    h2=patch('Vertices',com_vertices,'Faces',com_faces);
    M(j)=getframe(video);
    mov=addframe(mov,M(j));
    delete(h1);
    delete(h2);
end

mov=close(mov);
```

Appendix 6: Contents of the enclosed DVD

Files related to the full non linear model (created by Vicente Martinez):

- ✓ **quadrotor0_sf.c**
- ✓ **quadrotor0_sf.h**
- ✓ **quadrotor0_sf.dll**

Matlab files:

- ✓ **body_rates_to_euler.m**: function executed by Simulink
- ✓ **decoupling_inputs.m**: function executed by Simulink
- ✓ **initialization_trajectory_tracking.m**: code to initialize the Simulink model "proper_feedback_trajectory_tracking.mdl" which is not considered in the interface
- ✓ **interface.m**: function executed by "interface.fig"
- ✓ **video.m**: code to create videos of trajectory tracking
- ✓ **x_signal_building.m**: function executed by Simulink
- ✓ **y_signal_building.m**: function executed by Simulink
- ✓ **z_signal_building.m**: function executed by Simulink

Simulink models:

- ✓ **proper_feedback_guide_LQR.mdl**: model executed by "interface.fig". Investigates step responses with LQR controller.
- ✓ **proper_feedback_guide_PID.mdl**: model executed by "interface.fig". Investigates step responses with PID controller.
- ✓ **proper_feedback_trajectory_tracking.mdl**: model executed manually. Investigates the trajectory tracking task (PID controller).

Interface:

- ✓ **interface.fig**: interface used to run more easily the step response models (PID and LQR)

Video files:

- ✓ **Real time step response.wmv**: video of the quadrotor's response to a step input (see 8.1 for the video description)
- ✓ **Real time trajectory following.wmv**: video of the quadrotor's response to a specific trajectory input (see 8.2 for the video description)

Pdf version of the report:

- ✓ **Modelling and Linear Control of a Quadrotor.pdf**: pdf version of this report

Help to use correctly the software:

- ✓ **How to run simulations.doc**

Appendix 7: Software interface

Printscreens of the interface:

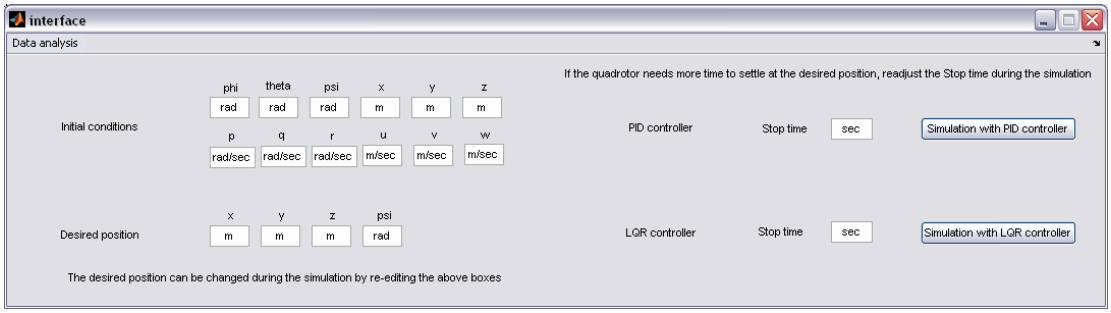


Figure 0.26 : printscreen of the interface used to run the simulations

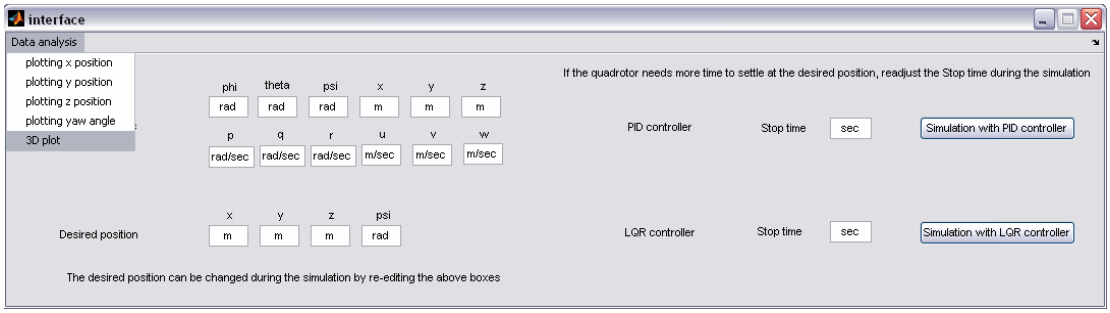


Figure 0.27 : printscreen of the interface used to analyse the data from the simulations

Matlab code of the guide used for the software interface:

- ✓ Programming callbacks for Edit Text boxes: initial conditions

```
function edit1_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit1=user_string;
guidata(hObject, handles);

function edit2_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit2=user_string;
guidata(hObject, handles);

function edit3_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit3=user_string;
guidata(hObject, handles);

function edit4_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit4=user_string;
guidata(hObject, handles);

function edit5_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit5=user_string;
guidata(hObject, handles);

function edit6_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit6=user_string;
guidata(hObject, handles);

function edit7_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit7=user_string;
guidata(hObject, handles);

function edit17_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit17=user_string;
guidata(hObject, handles);

function edit18_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit18=user_string;
guidata(hObject, handles);

function edit19_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit19=user_string;
guidata(hObject, handles);

function edit20_Callback(hObject, eventdata, handles)
```

```
user_string = get(hObject,'string');
handles.edit20=user_string;
guidata(hObject, handles);

function edit21_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit21=user_string;
guidata(hObject, handles);
```

✓ Programming callbacks for Edit Text boxes: desired position

```
function edit13_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit13=user_string;
guidata(hObject, handles);
assignin('base','xcom',str2double(get(hObject,'string')))
try
    set_param('proper_feedback_guide_PID', 'SimulationCommand',
'pause')
    set_param('proper_feedback_guide_PID/Constant3','Value','xcom')
    set_param('proper_feedback_guide_PID', 'SimulationCommand',
'continue')
catch
end
try
    set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'pause')
    set_param('proper_feedback_guide_LQR/Constant3','Value','xcom')
    set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'continue')
catch
end


function edit14_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit14=user_string;
guidata(hObject, handles);
assignin('base','ycom',str2double(get(hObject,'string')))
try
    set_param('proper_feedback_guide_PID', 'SimulationCommand',
'pause')
    set_param('proper_feedback_guide_PID/Constant2','Value','ycom')
    set_param('proper_feedback_guide_PID', 'SimulationCommand',
'continue')
catch
end
try
    set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'pause')
    set_param('proper_feedback_guide_LQR/Constant2','Value','ycom')
    set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'continue')
catch
end
```

```
function edit15_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit15=user_string;
guidata(hObject, handles);
assignin('base','zcom',str2double(get(hObject,'string')))
try
    set_param('proper_feedback_guide_PID', 'SimulationCommand',
'pause')
    set_param('proper_feedback_guide_PID/Constant1','Value','zcom')
    set_param('proper_feedback_guide_PID', 'SimulationCommand',
'continue')
catch
end
try
    set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'pause')
    set_param('proper_feedback_guide_LQR/Constant1','Value','zcom')
    set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'continue')
catch
end


function edit16_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit16=user_string;
guidata(hObject, handles);
assignin('base','psicom',str2double(get(hObject,'string')))
try
    set_param('proper_feedback_guide_PID', 'SimulationCommand',
'pause')

set_param('proper_feedback_guide_PID/Constant4','Value','psicom')
    set_param('proper_feedback_guide_PID', 'SimulationCommand',
'continue')
catch
end
try
    set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'pause')

set_param('proper_feedback_guide_LQR/Constant4','Value','psicom')
    set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'continue')
catch
end
```

✓ Programming callbacks for Edit Text boxes: stop time

```
function edit23_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit23=user_string;
guidata(hObject, handles);
try
    if
get_param('proper_feedback_guide_PID','SimulationStatus')=='running'
,
        set_param('proper_feedback_guide_PID', 'SimulationCommand',
'pause')
        set_param('proper_feedback_guide_PID', 'StopTime',
get(hObject,'String'))
        set_param('proper_feedback_guide_PID', 'SimulationCommand',
'continue')
    end
catch
end


function edit24_Callback(hObject, eventdata, handles)
user_string = get(hObject,'string');
handles.edit24=user_string;
guidata(hObject, handles);
try
    if
get_param('proper_feedback_guide_LQR','SimulationStatus')=='running'
,
        set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'pause')
        set_param('proper_feedback_guide_LQR', 'StopTime',
get(hObject,'String'))
        set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'continue')
    end
catch
end
```

✓ Programming callbacks for Pushbuttons: run simulations with PID and LQR controllers

```
function pushbutton1_Callback(hObject, eventdata, handles)
xInitial =
['[',handles.edit1,';',handles.edit2,';',handles.edit3,';',handles.e
dit4,';',handles.edit5,';',handles.edit6,';','0',';','0',';','0',';'
,handles.edit20,';',handles.edit21,';',handles.edit7,';',handles.edi
t17,';',handles.edit18,';',handles.edit19,';','151.62',';','-
151.62',';','151.62',';','-
151.62',';','0',';','0',';','0',';','0',']'];
l=0.5;
proper_feedback_guide_PID
blocks =
find_system('proper_feedback_guide_PID','BlockType','Scope');
```

```
for i = 1:length(blocks)
            set_param(blocks{i},'Open','on')
end
set_param('proper_feedback_guide_PID','InitialState',xInitial)
set_param('proper_feedback_guide_PID/Constant1','Value','zcom')
set_param('proper_feedback_guide_PID/Constant2','Value','ycom')
set_param('proper_feedback_guide_PID/Constant3','Value','xcom')
set_param('proper_feedback_guide_PID/Constant4','Value','psicom')
set_param('proper_feedback_guide_PID/Gain10','Gain',num2str(1/l))
set_param('proper_feedback_guide_PID/Gain11','Gain',num2str(1/l))
string_vert={['[',handles.edit1,' ',handles.edit2,'
',handles.edit3,']'];'1.225';['[',handles.edit4,' ',handles.edit5,'
',handles.edit6,']']};
set_param('proper_feedback_guide_PID/XPro (by Vicente
Martinez)','MaskValues',string_vert)
string_vert2={'quadrotor0_sf';'off';'eul_0_3';'rho_4';'xme_0_5'};
set_param('proper_feedback_guide_PID/XPro (by Vicente
Martinez)/quadrotor_sfcn','MaskValues',string_vert2)
set_param('proper_feedback_guide_PID','StopTime',handles.edit23)
set_param('proper_feedback_guide_PID', 'SimulationCommand', 'start')


function pushbutton2_Callback(hObject, eventdata, handles)
    m=2.353598;
    g=9.81;
    Ix=0.1676;
    Iy=0.1686;
    Iz=0.29743;
    l=0.5;
    phi=0;
    theta=0;
    psi=0;
    u1=m*g/(cos(theta)*cos(phi));
    Amoving=[0 0 0 1 0 0 0 0 0 0 0 0 ; 0 0 0 0 1 0 0 0 0 0 0 0 ; 0 0
0 0 0 1 0 0 0 0 0 0 ;
             0 0 0 0 0 0 sin(theta)*sin(phi)*u1/m -
cos(theta)*cos(phi)*u1/m 0 0 0 0 ; 0 0 0 0 0 0 cos(phi)*u1/m 0 0 0 0
0 ; 0 0 0 0 0 0 cos(theta)*sin(phi)*u1/m sin(theta)*cos(phi)*u1/m 0
0 0 0 ;
             0 0 0 0 0 0 0 0 0 1 0 0 ; 0 0 0 0 0 0 0 0 0 0 1 0 ; 0 0
0 0 0 0 0 0 0 0 0 1 ;
             0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0 0 0 ; 0 0
0 0 0 0 0 0 0 0 0 0 ];
    Bmoving=[0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ; -sin(theta)*cos(phi)/m 0
0 0 ; sin(phi)/m 0 0 0 ; -cos(theta)*cos(phi)/m 0 0 0 ; 0 0 0 0 ; 0
0 0 0 ; 0 0 0 0 ; 0 cos(psi)/Ix -sin(psi)/Iy 0 ; 0
sin(psi)/(cos(phi)*Ix) cos(psi)/(cos(phi)*Iy) 0 ; 0
sin(psi)*tan(phi)/Ix cos(psi)*tan(phi)/Iy 1/Iz ];
    Cmoving=[1 0 0 0 0 0 0 0 0 0 0 0 ; 0 1 0 0 0 0 0 0 0 0 0 0 ; 0 0
1 0 0 0 0 0 0 0 0 0 ; 0 0 0 1 0 0 0 0 0 0 0 0 ; 0 0 0 0 1 0 0 0 0 0
0 0 ; 0 0 0 0 0 1 0 0 0 0 0 0 ; 0 0 0 0 0 0 1 0 0 0 0 0 ; 0 0 0 0 0
0 0 1 0 0 0 0 ; 0 0 0 0 0 0 0 0 1 0 0 0 ; 0 0 0 0 0 0 0 0 0 1 0 0 ;
0 0 0 0 0 0 0 0 0 0 1 0 ; 0 0 0 0 0 0 0 0 0 0 0 1 ];
    Dmoving=0;
    Smoving=tf(ss(Amoving,Bmoving,Cmoving,Dmoving));
    A=Amoving;
    R=[100 0 0 0 ; 0 100000 0 0 ; 0 0 100000 0 ; 0 0 0 10];
    B=Bmoving*inv(R)*Bmoving';
```

```
    C=eye(12);
    C(1,1)=10;
    C(2,2)=10;
    C(3,3)=10;
    C(9,9)=10;
    X=are(A,B,C);
    K=inv(R)*Bmoving'*X;
    xInitial =
['[',handles.edit1,';',handles.edit2,';',handles.edit3,';',handles.e
dit4,';',handles.edit5,';',handles.edit6,';','0',';','0',';','0',';'
,handles.edit20,';',handles.edit21,';',handles.edit7,';',handles.edi
t17,';',handles.edit18,';',handles.edit19,';','151.62',';','-
151.62',';','151.62',';','-
151.62',';','0',';','0',';','0',';','0',']'];
    proper_feedback_guide_LQR
    blocks =
find_system('proper_feedback_guide_LQR','BlockType','Scope');
    for i = 1:length(blocks)
            set_param(blocks{i},'Open','on')
    end
set_param('proper_feedback_guide_LQR','InitialState',xInitial)
set_param('proper_feedback_guide_LQR/Constant1','Value','zcom')
set_param('proper_feedback_guide_LQR/Constant2','Value','ycom')
set_param('proper_feedback_guide_LQR/Constant3','Value','xcom')
set_param('proper_feedback_guide_LQR/Constant4','Value','psicom')
set_param('proper_feedback_guide_LQR/Gain10','Gain',num2str(1/l))
set_param('proper_feedback_guide_LQR/Gain11','Gain',num2str(1/l))
set_param('proper_feedback_guide_LQR/Gain7','Gain',num2str(K(1,3)))
set_param('proper_feedback_guide_LQR/Gain8','Gain',num2str(K(1,6)))
set_param('proper_feedback_guide_LQR/Gain9','Gain',num2str(K(2,2)))
set_param('proper_feedback_guide_LQR/Gain13','Gain',num2str(K(2,5)))
set_param('proper_feedback_guide_LQR/Gain14','Gain',num2str(K(2,7)))
set_param('proper_feedback_guide_LQR/Gain15','Gain',num2str(K(2,10))
)
set_param('proper_feedback_guide_LQR/Gain17','Gain',num2str(K(3,1)))
set_param('proper_feedback_guide_LQR/Gain18','Gain',num2str(K(3,4)))
set_param('proper_feedback_guide_LQR/Gain19','Gain',num2str(K(3,8)))
set_param('proper_feedback_guide_LQR/Gain20','Gain',num2str(K(3,11))
)
set_param('proper_feedback_guide_LQR/Gain21','Gain',num2str(K(4,9)))
set_param('proper_feedback_guide_LQR/Gain22','Gain',num2str(K(4,12))
)
set_param('proper_feedback_guide_LQR/Gain16','Gain',num2str(K(1,3)))
set_param('proper_feedback_guide_LQR/Gain23','Gain',num2str(K(2,2)))
set_param('proper_feedback_guide_LQR/Gain24','Gain',num2str(K(3,1)))
set_param('proper_feedback_guide_LQR/Gain25','Gain',num2str(K(4,9)))
    string_vert={['[',handles.edit1,' ',handles.edit2,'
',handles.edit3,']'];'1.225';['[',handles.edit4,' ',handles.edit5,'
',handles.edit6,']']};
set_param('proper_feedback_guide_LQR/XPro (by Vicente
Martinez)','MaskValues',string_vert)
string_vert2={'quadrotor0_sf';'off';'eul_0_3';'rho_4';'xme_0_5'};
    set_param('proper_feedback_guide_LQR/XPro (by Vicente
Martinez)/quadrotor_sfcn','MaskValues',string_vert2)
    set_param('proper_feedback_guide_LQR','StopTime',handles.edit24)
    set_param('proper_feedback_guide_LQR', 'SimulationCommand',
'start')
```

✓ Programming callbacks for Pop-Up Menus: data analysis

```
function Untitled_2_Callback(hObject, eventdata, handles)
xpos=evalin('base','xpos');
ypos=evalin('base','ypos');
zpos=evalin('base','zpos');
figure(1)
plot3(xpos,ypos,-zpos);
grid on
az=45;
el=45;
view(az,el);
hold on
plot3([str2double(handles.edit4)
str2double(handles.edit13)],[str2double(handles.edit5)
str2double(handles.edit14)],[-str2double(handles.edit6) -
str2double(handles.edit15)],'r');
title('Trajectory following');

function Untitled_3_Callback(hObject, eventdata, handles)
xpos=evalin('base','xpos');
t=evalin('base','t');
figure(1)
plot(t,xpos);
title('X position vs time');

function Untitled_4_Callback(hObject, eventdata, handles)
ypos=evalin('base','ypos');
t=evalin('base','t');
figure(1)
plot(t,ypos);
title('Y position vs time');

function Untitled_5_Callback(hObject, eventdata, handles)
zpos=evalin('base','zpos');
t=evalin('base','t');
figure(1)
plot(t,zpos);
title('Z position vs time');

function Untitled_6_Callback(hObject, eventdata, handles)
yaw=evalin('base','yaw');
t=evalin('base','t');
figure(1)
plot(t,yaw);
title('Yaw angle vs time');
```