Control of a convertible VTOL

Alessandro Ferraioli
Supervisor: A. Astolfi

July 2019

List of Figures

Contents

Acknowledgements

I would like to express my gratitude to my supervisor Alessandro Astolfi for the useful comments, remarks and engaging conversation through the learning process of this masters thesis. Also, I would like to thank my colleagues who shared in this journey through these wonderful two years. I would especially like to thank my closest friends who have supported me, by keeping me harmonious and helping me to put the pieces together. I will be forever grateful for your love. Furthermore, I'd like to thank my parents and my siblings for their faith in me. Finally, I want to thank the University of Tor Vergata, which granted me the opportunity to embark on Erasmus, which has been one of most stimulating experiences of my life, where I found my second family; my flatmates. I thank them for inspiring me during these last nine months which I will always remember.

Failure is an option here. If things are not failing, you are not innovating enough.
Elon Musk

Abstract

The aim of this thesis is the study and the control of a special class of a convertible *VTOL* with four rotors and fixed wings. This vehicle combines the high-speed cruise capabilities of a conventional airplane with the hovering capabilities of a helicopter. Initially, the different behaviours are separately studied, then a complete simulation from take-off to landing is proposed. A nonlinear control strategy using: feedback-linearization, backstepping and bounded-saturation control is proposed. The convergence analysis is based on Lyapunovs and LaSalle method.
Finally a study of a swarm-formation in hover-flight is proposed using a potential function control law.
So, the main contributions of this work are the following

- Providing a full simulation from take-off to landing using a state-machine approach in order to manage the transition.
- Using a backstepping with bounded input approach in order to stabilize the hover flight mode.
- Using a backstepping approach in order to stabilize the transition flight with a non-constant thrust.

1 Introduction

1.1 History

The idea of vertical flight has been around for thousands of years and sketches for a VTOL (helicopter) shows up in Leonardo da Vinci's sketch book. Manned VTOL aircraft, in the form of primitive helicopters, first flew in 1907 but would take until after World War Two to perfect.
In addition to helicopter development, many approaches have been tried to develop practical aircraft with vertical take-off and landing capabilities including Henry Berliner's 1922–1925 experimental horizontal rotor fixed wing aircraft, and Nikola Tesla's 1928 patent and George Lehberger's 1930 patent for relatively impractical VTOL fixed wing airplanes with a tilting engines.

The use of vertical fans driven by engines was investigated in the 1950s. The US built an aircraft where the jet exhaust drove the fans, while British projects not built included fans driven by mechanical drives from the jet engines.

A different British VTOL project was the *Gyrodyne*, where a rotor is powered during take-off and landing but which then freewheels during flight, with separate propulsion engines providing forward thrust. Starting with the Fairey *Gyrodyne*, this type of aircraft later evolved into the much larger twin-engined Fairey Rotodyne, that used tipjets to power the rotor on take-off and landing but which then used two Napier Eland turboprops driving conventional propellers mounted on substantial wings to provide propulsion, the wings serving to unload the rotor during horizontal flight. The Rotodyne was developed to combine the efficiency of a fixed-wing aircraft at cruise with the VTOL capability of a helicopter to provide short haul airliner service from city centres to airports.[1]

Fixed-wing and rotor craft aerial vehicles, as representations of aircraft with conventional structures, have played irreplaceable roles in aviation development for a long time. Takeoff environment affects normal flight and they cannot hover at a fixed position. Rotor craft aerial vehicles, such as helicopters, which are driven by propellers with a swash plate in the vertical direction, can hover in one spot, but cruise speed and flight endurance are low. To combine the advantages of both vehicles, researchers worldwide have focused on aircraft with both high-speed cruise and vertical takeoff and landing (VTOL) ability. This led to the concept of tilt rotor aerial vehicles.

Because of the variable rotor tilt angle, the most attractive characteristic of tilt rotor aerial vehicles is that they possess the capability to hover in place, as helicopters, while achieving much higher cruise speeds than conventional rotor craft in air- plane mode.

A vertical take-off and landing (VTOL) aircraft is one that can hover, take off, and land vertically. This classification can include a variety of types of

aircraft including fixed-wing aircraft as well as helicopters and other aircraft with powered rotors, such as cyclogyros/cyclocopters and tiltrotors.
The applications are widely diversified, from the military to the civilian purpose. VTOL technology means aircraft can theoretically take off and land almost anywhere, making them far more flexible. They're also able to perform various manoeuvres not possible with a conventional plane; a significant advantage for aircraft in combat situations.
The classic scheme of the VTOL is showed in the following figure

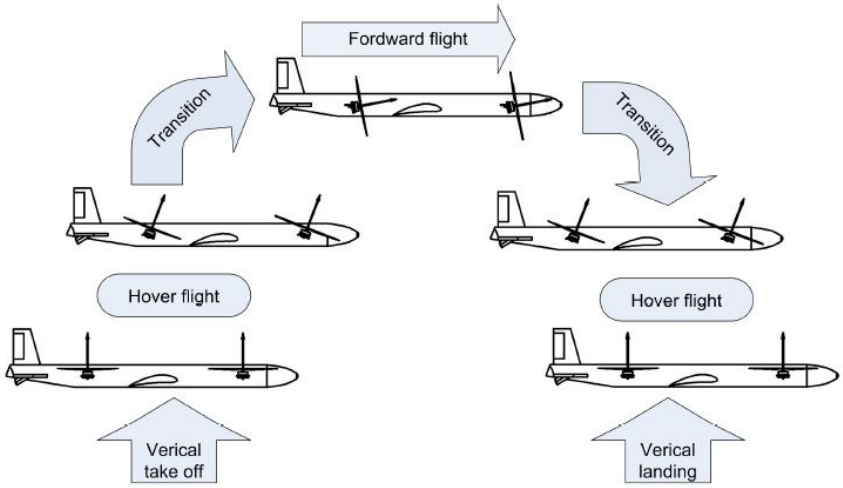


Figure 1: Operational scheme

1.2 Body frames

In this section will be exposed the physical model used in this thesis. The equations of the motion presents are for the longitudinal flight.
It is necessary define three different *frames* in order to describe the movement of the aircraft and the forces acting on it.

1. F^i is the inertial earth-fixed frame with axis (x^i, y^i, z^i).
2. F^b is the body-fixed frame wit axis (x^b, y^b, z^b).
3. F^a is the aerodynamic frame wit axis (x^a, y^a, z^a).

A graphical description of the first two frames is provided in the following figure

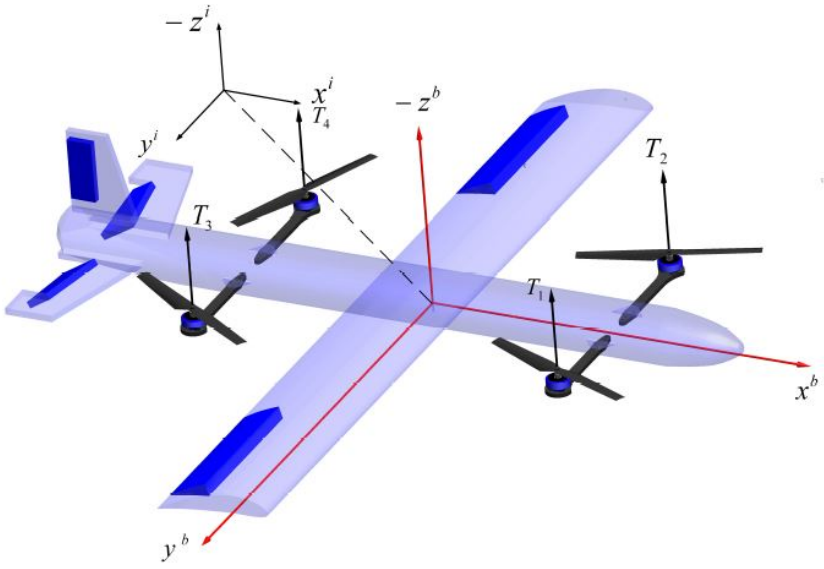


Figure 2: Definition of F^i and F^b

It is worthwhile to be pointed out that the z^i axis is oriented as the opposite of the standard choice. In this way the vector of gravity g will have only positive components.
Considering the following figure, which defines the convention of the angles used in this work, it is possible to define two matrices of transformation :R^{bi} and R^{ab}, respectively $F^b \rightarrow F^i$ and $F^a \rightarrow F^b$.

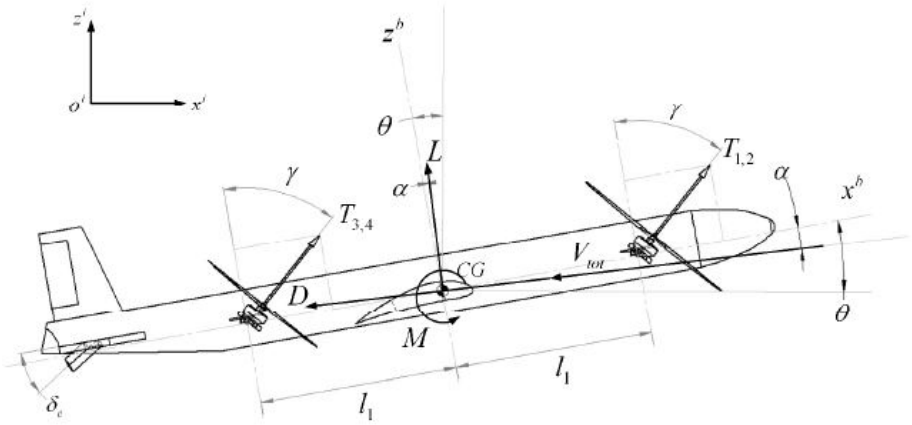


Figure 3: Body's diagram

The matrices(based on the above figure) are the follow

$$R^{bi} = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} R^{ab} = \begin{bmatrix} cos(\alpha) & 0 & sin(\alpha) \\ 0 & 1 & 0 \\ -sin(\alpha) & 0 & cos(\alpha) \end{bmatrix}$$

Using the definition of the angle γ, it is possible to split the flight of a VTOL into three different modes

1. *Hover-flight* mode (HF) is characterized by the angle $\gamma \leq \frac{\pi}{6}$. The goal of this phase will be stabilize the aircraft with angle $\theta = 0$,and the position $(x, z) = (x_{des}, z_{des})$.

2. Before the angle γ reaches the value close to $\frac{\pi}{2}$ we can identify another flight mode, which is the link between the *hover-flight* mode and the last one. This mode is so-called : *transition-flight* mode (TF).

3. When the angle γ reaches the value of $\frac{\pi}{2}$ the airplane behaves like a normal airplane and the *lift force* L is sufficient to stabilize the system. This mode is the so-called *forward-flight* mode (FF).

1.3 Aerodynamics

In order to proceed with the aerodynamics's analysis is worth to state the following assumptions

- The aircraft is described as rigid-body.
- The mass will be considered constant.

- The distance between the aerodynamics's center and the gravity's center is 0.

Under the above assumptions is possible to define the vehicle's aerodynamic. It is possible to identify three *wind components* :

$$V = V_p(\gamma) + V_e + V_b$$

where $V_p(\gamma)$ is the wind generated by rotors and it is a function of the angle γ, the components V_e, V_b are respectively the wind generated by the external wind and the body's translation.
It is possible to define the V_p component using the following discontinue function $\zeta(\gamma) = 1 \cdot a(\gamma - \frac{\pi}{3})$, where the function $a(x) \neq 0 \leftrightarrow x > 0$.
In this way is possible to decompose the vector V into two components (u, w) as the following figure shows

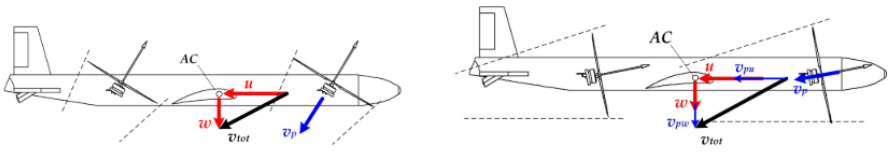


Figure 4: Wind Velocity

where the first figure shows the airplane in the *hover-flight* mode, and the second one shows the airplane in the *forward-flight* mode. Bear in mind the above figure is possible to define the following equations

$$V_u = (u + \zeta(\gamma)v_p sin(\gamma) + v_{eu})i_b$$
$$V_w = (w + \zeta(\gamma)v_p cos(\gamma) + v_{ew})k_b$$

Considering the rotors as a simple *actuator-disc* is possible to define the wind generated by the rotors in the following way

$$v_{pi} = \sqrt{\frac{2T_i}{\rho A_p}} \tag{1}$$

where A_p is the total disc area generated by rotational movement of the blades of the four rotors, ρ is the density of the air and the T_i is the thrust generated by the $i^{th} - rotor$. It is possible to calculate the thrust in the following way

$$T = K_l \sum_{i=1}^{4} w_i^2$$

where K_l is the lift factor depending on the aerodynamic parameters of the propeller.
In order to calculate the *lift-force*, the *drag-force* and the *pitching-moment* is necessary to obtain the following coefficients

$$\begin{aligned} C_l &= C_{l\alpha}\alpha \\ C_d &= C_{dp} + C_{di} \\ C_m &= C_{m\alpha}\alpha + C_{m\delta}\delta_e \end{aligned}$$

It is worthwhile to be pointed out that into the definition *pitching-coefficient* C_m, the most important part is the coefficient $C_{m\delta}$, which is the mapping between the angle δ_e of deflection and the *pitching-moment* generated.
Using the above definitions, it is possible to define the *lift-force* L, the *drag-force* D and the *pitching-moment* M as follow

$$\begin{aligned} L &= \frac{1}{2}C_l\rho V^2 S \\ D &= \frac{1}{2}C_d\rho V^2 S \\ M &= \frac{1}{2}C_m\rho V^2 S\bar{c} \end{aligned} \tag{2}$$

where S is the area of the wing and $\bar{c}$ is the wing chord.

1.3.1 Pitching moment

In aerodynamics, the pitching moment on an airfoil is the moment (or torque) produced by the aerodynamics's force on the airfoil if that force is applied not at the center of pressure, but at the aerodynamics's center of the airfoil.[2]
The *pitching-moment's* coefficient C_m is important in the study of the longitudinal static stability of aircraft and missiles and it is defined as follow :

$$C_m = \frac{M}{qSc}$$

where as usual M is the pitching moment, q is the dynamic pressure, S is the wing area and c is the length of the chord of the airfoil. It is worthwhile to be pointed out that the C_m is dimensionless. The pitching moment is, by convention, considered to be positive when it acts to pitch the airfoil in the nose-up direction.
In the following figure is plotted a typical plot of the pitching moment respect to the angle of attack α

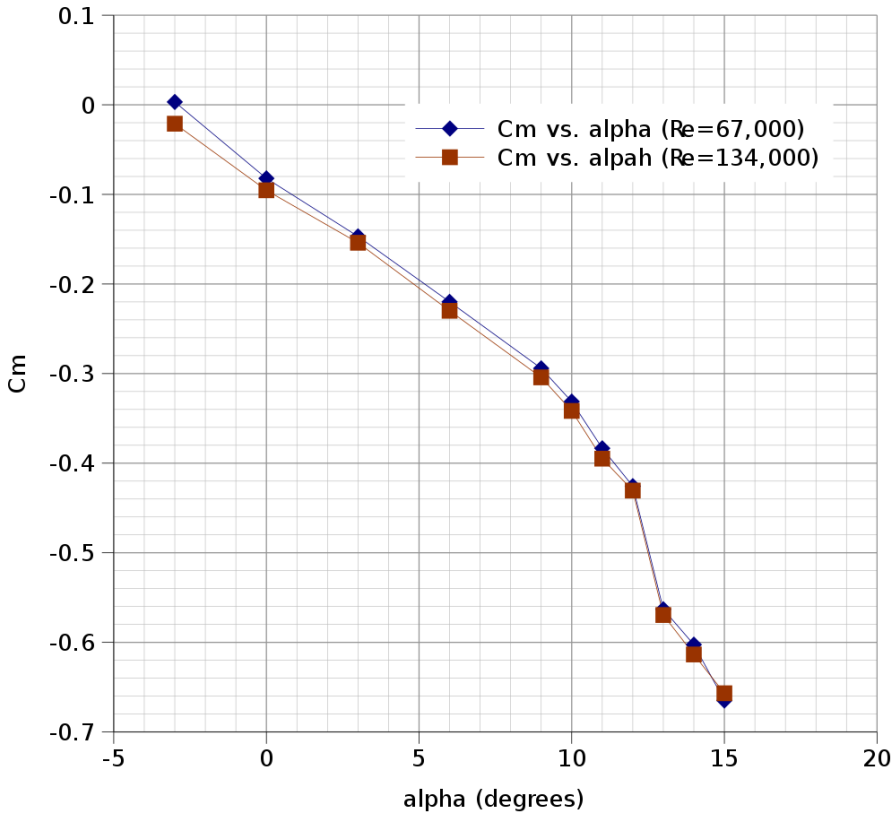


Figure 5: C_m vs α

The elevator deflection angle is taken negative when the trailing edge is deflected upward and positive when the trailing edge is deflected is downward.[3]

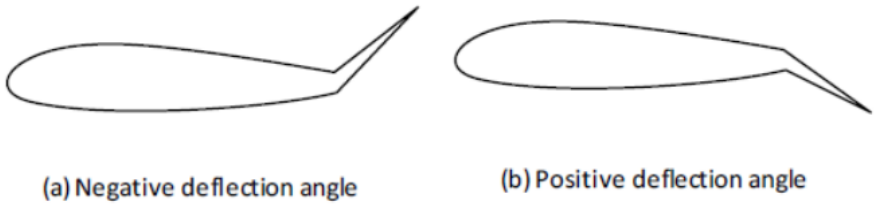


Figure 6: Deflection angle

1.3.2 Lift coefficient

The lift coefficient C_L, as the C_m is a dimensionless coefficient and it is defined as follow [4]

$$C_L = \frac{L}{qS}$$

where L is the lift force, S, is the relevant surface area and q, is the fluid dynamic pressure.
It is common to show, for a particular airfoil section, the relationship between section lift coefficient and angle of attack.

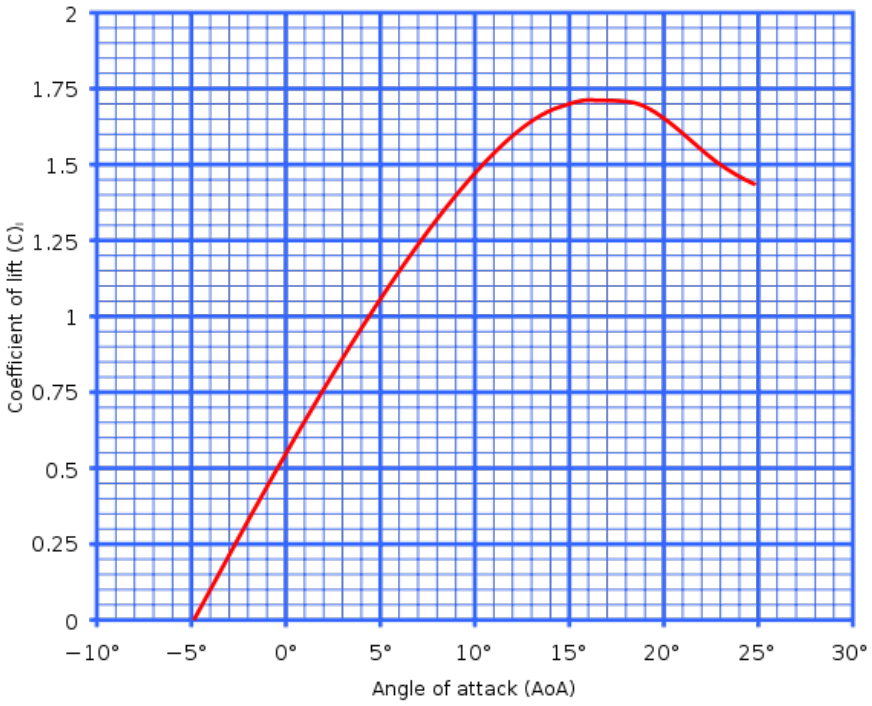


Figure 7: C_L vs α

1.4 Moments

As well known, the *moments* are generated by the forces shifted from the center of gravity. The total moments acting on the airplane are defined as follow

$$\tau = \tau_T + \tau_M + \tau_G$$

where τ_T is the moment due to the difference of thrust $\Delta_T = T_{34} - _{12}$, τ_M is the moment due to the airfoil's pitching moment, and it will be the control input

during the *forward-flight* mode, where the pitching moment will be generated using the deflection angle on the airfoil-tail and the τ_G is the moment due to the gyroscopic effect, which is neglected.
The moment τ_T is defined as follow

$$\tau_T = l_1(-T_{34}cos(\gamma) + T_{12}cos(\gamma))j_b$$

where j_b is the versor of the y axis of the body-frame, according to the figure (2).

1.5 Mathematical model

Under the assumption that the system can be modeled as rigid body [5], it is possible to write the equation of the dynamic in the following way

$$\begin{aligned} m\ddot{x} &= -T_{34}sin(\theta+\gamma) - T_{12}sin(\theta+\gamma) - Lsin(\theta-\alpha) - Dcos(\alpha-\gamma) \\ m\ddot{z} &= Dsin(\theta-\alpha) - T_{12}cos(\theta+\gamma) - Lcos(\theta-\alpha) - T_{34}cos(\theta+\gamma) + g \\ I_{yy}\ddot{\theta} &= M + l_1\tau_b + I_p p(w_1 - w_2 - w_3 + w_4) \end{aligned} \tag{3}$$

In the *forward-flight* mode, in addition to the dynamic of the x and z axis, it is necessary to describe the dynamic of wind V, the angle of attack α and the pitch-rate. In this regime, the aerodynamic forces became widely different from zero, actually they became the most important forces in the system. The dynamic of the system in this behaviour can be described as follow

$$\begin{aligned} \dot{V} &= \frac{1}{m}(-D + T_t cos(\alpha) - mg(cos(\alpha)sin(\theta) - sin(\alpha)cos(\theta)) \\ \dot{\alpha} &= \frac{1}{Vm}(-L - T_t sin(\alpha) + mg(cos(\alpha)cos(\theta) + sin(\alpha)sin(\theta))) + q \\ \dot{\theta} &= q \\ \dot{q} &= \frac{1}{I_{yy}}M \end{aligned} \tag{4}$$

It is worthwhile to be pointed out that the angle θ and the angle α lie in the same vertical plane, hence we can define the *path angle* $\gamma = \theta - \alpha$

2 Control hover flight mode

In this section will be developed a controller for the hover-flight using two different approach : *feedback-linearization* and *backstepping*. It is worthwhile to be pointed out that this phase is one of the most important, because it will influence the next two phases.

2.1 Control using feedback-linearization

In this mode the airplane will have the angle of the motors $\gamma = 0$, so they will be oriented as the $z - axis$. We will consider the gravity force g as external-constant disturbance on the dynamics of the system along the $z - axis$, which is oriented in the same direction of the vector g. It is obviously that in this mode the angle θ between the $x - axis$ of the inertial-fixed frame and the $x - axis$ of the body-fixed frame will be close to zero, so we can assume that the case θ close to $\frac{\pi}{2}$ is avoided.

The system has *2-DOF* given by the sum of the thrusts of the 4 motors T_t and the difference of thrusts between the two motors in front and the two in the back T_d. The dynamics of the system is given by the following differential equations :

$$\begin{aligned} \ddot{x} &= -\frac{T_t \cdot sin(\theta)}{m} \\ \ddot{z} &= -\frac{T_t \cdot cos(\theta)}{m} + g \\ \ddot{\theta} &= -\frac{l_1 \cdot T_d}{I_{yy}} \end{aligned} \tag{5}$$

It is worthwhile to point out that as we said, the disturbance g has the same direction of the *z-axis*. The controller will follow the *feedback-linearized* approach. We are going to use one of the *DOF* in order to stabilize the dynamics on the *z-axis*. Re-writing the dynamic of the *z-axis* in the standard form :

$$\dot{x} = f(x) + g \cdot u$$

and taking the state $x = [x_1, x_2]^T$, it is possible to re-write the dynamic as follow

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= -\frac{T_t \cdot cos(\theta)}{m} + g \end{aligned} \tag{6}$$

Taking $y = x_1$ as output of the system it is to verify that the relative degree of the system is $r = 2$, then we can choose the controller as follow : $\ddot{x_1} = \dot{x_2} = u_z$, hence $T_t = -\frac{m \cdot u_z - mg}{cos(\theta)}$.

The closed loop system behaves like a standard *double integrator* in the following form : $\ddot{z} = u_z$, where u_z is the virtual control. An easy choice of u_z is the standard controller $u_z = -k_{pz}(z - z_{des}) - k_{dz}\dot{z}$, which stabilize the dynamic around the equilibrium point $z_{eq} = [z_{des}, 0]$. It is worthwhile to point out, that the singularity of the controller, which occurs for value of $\theta \approx \frac{\pi}{2}$ is avoided as specified before.
Taking into account the dynamics of x and θ we can use the last DOF in order to stabilize those last two dynamics. Considering the closed-loop system for the dynamic on the $x - axis$ we obtain the follow result

$$\ddot{x} = (u_z - g)tan(\theta) \tag{7}$$

We can consider θ as virtual control input, so we obtain

$$\begin{aligned} \theta_{des} = u_x = atan(\frac{v_x}{u_z - g}) \\ v_x = -k_{px}(x - x_{des}) - k_{dx}\dot{x} \end{aligned} \tag{8}$$

where the controller v_x stabilizes the double integrator $\ddot{x} = v_x$. In order to find a controller T_d, such that $\tilde{\theta} = \theta - \theta_{des} \to 0$ we can use the following approach: Picking the following *Lyapunov-function*

$$V(\tilde{\theta}, \dot{\theta}) = \frac{1}{2} I_{yy}\dot{\theta}^2 + ln(cosh(\tilde{\theta}))$$

and taking the derivative along the solution we end up to $\dot{V} = \dot{\theta}(-l_1 T_d + tanh(\tilde{\theta}))$ where we want to set ≤ 0. Picking the following controller

$$T_d = \frac{tanh(\tilde{\theta} + tanh(\dot{\theta}))}{l_1}$$

then $\dot{V} = -\dot{\theta}tanh(\dot{\theta}) \leq 0$. The asymptotically stability of the system can be proved using the Lasalle's invariance principle. In fact, it is easy to prove that 0 is the largest invariant set contained in $Ker[\dot{V}] := \{x : \dot{V}(x) = 0\}$. Taking the following sets

$$K_0 = [\dot{\theta} = 0] \quad K_1 = [K_0 \quad \& \quad tanh(\tilde{\theta}) = 0] = [\dot{\theta} = 0 \,\&\, \tilde{\theta} = 0]$$

we obtain that the origin is the largest invariant set contained in $Ker[\dot{V}]$.

2.1.1 Simulation using Feedback Linearization

In this section will be plotted the results of the simulations in *hover-flight* mode using the feedback linearization approach. As it has already been announced the results are not so good, so the controller will need to be improved using a backstepping approach.
The model has been simulated in *MATLAB/SIMULINK* environment using the following values :

$$\theta_{des} = 0 \quad x_{des} = 4 \quad z_{des} = 10$$
$$\theta_0 = \frac{\pi}{8} \quad x_0 = 0 \quad z_0 = 5$$

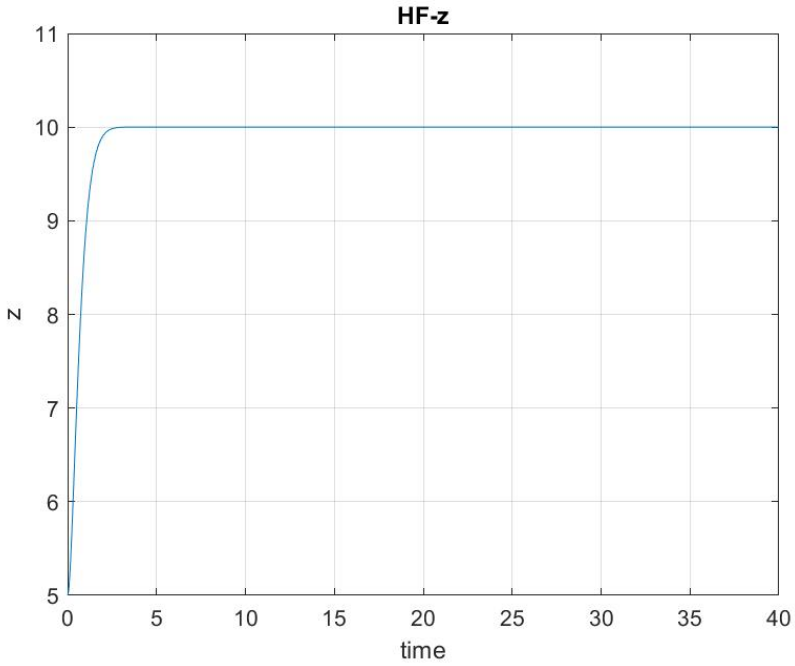


Figure 8: Hover Flight z position

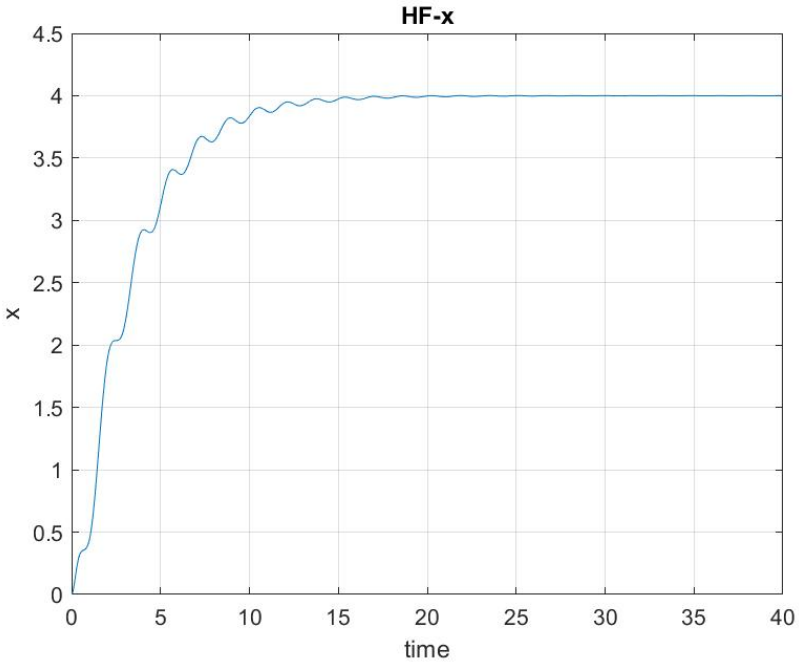


Figure 9: Hover Flight x position

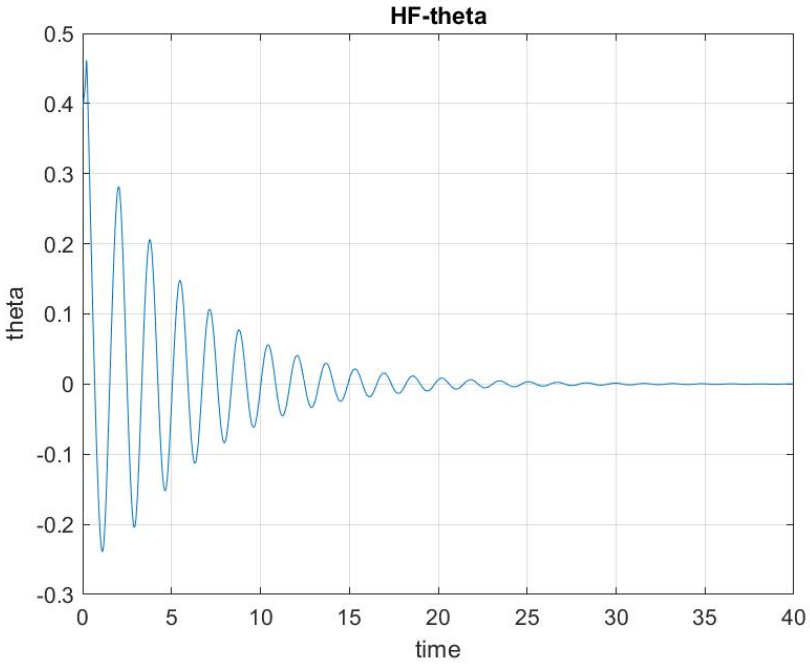


Figure 10: Hover Flight θ angle

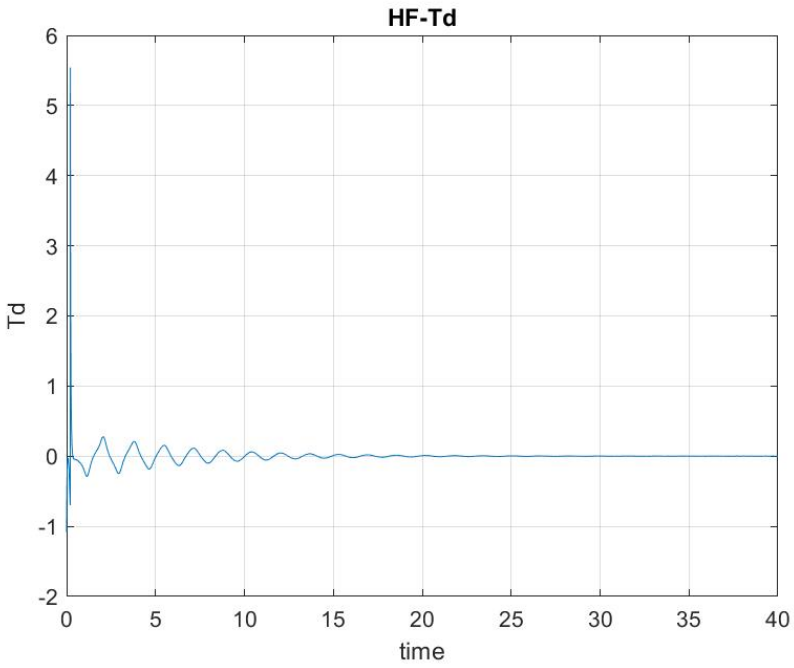


Figure 11: Hover Flight T_d control

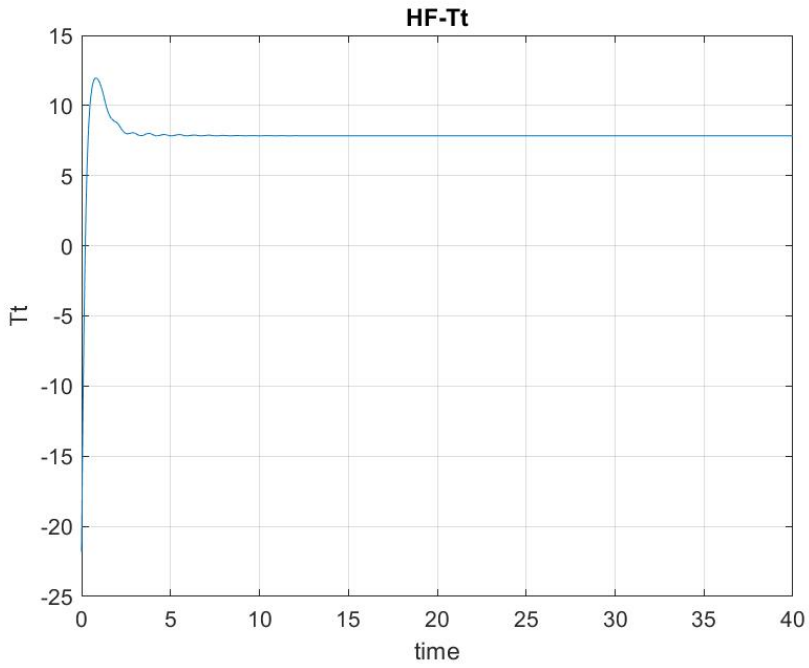


Figure 12: Hover Flight T_t control

2.2 Control using Backstepping

2.2.1 Recall about saturation function

The saturation function $\sigma_a(x)$ is defined as follow

$$\sigma_a(x) = \begin{cases} x & if\ |x| \leq a \\ \dfrac{a|x|}{x} & if\ |x| > a \end{cases}$$

In the following figure has been plotted the plot of the *saturation function*

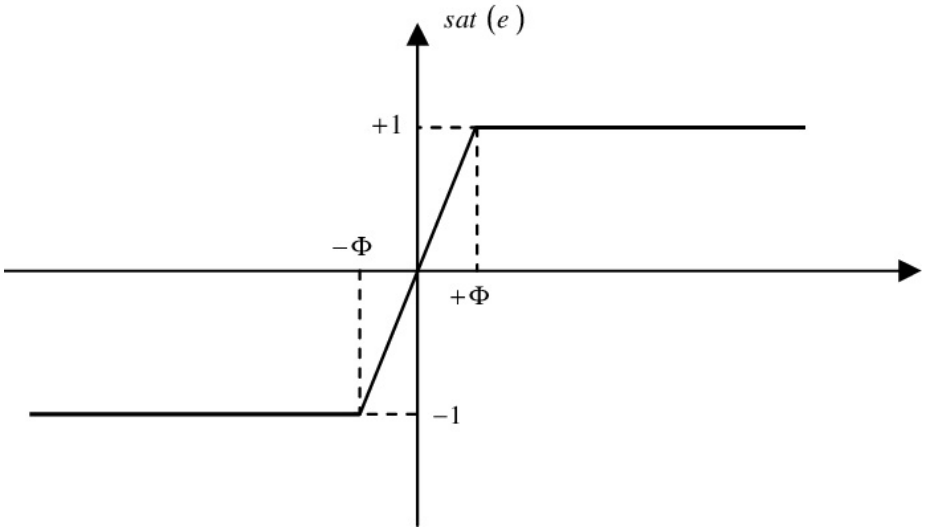


Figure 13: Saturation function

The derivative of the saturation function is defined as follow

$$\dot{\sigma}_a(x) = \begin{cases} 1 & if\ |x| \leq a \\ 0 & if\ |x| > a \end{cases}$$

and obviously the second derivative $\ddot{\sigma}_a(x) = 0$. It is worthwhile to be pointed out some properties of the saturation function [6], because they will be useful in the analysis of the stability of the controller, which will be developed in the next section.

1. $\int_0^x \sigma_a(s)ds > 0$ for any $x \neq 0$
2. $\int_0^x \sigma_a(s)ds > 0$ for any $x \neq 0$
3. $(\sigma_a(y) - \sigma_a(x+y))y \leq 0$

2.2.2 Control Strategy

In this section will be developed a better and more robust controller for the hovering flight using a backstepping approach as proposed in [7], [8] and [9]. Taking into account again the dynamics of the *hover-flight* mode we can split the problem , as we already did in the previous section,into two part:
• Stabilization on the *z-axis*.
• Stabilization of the angle θ and on *x-axis*.
For the *z-dynamic* will be adopted a *feedback-linearization* approach. So defining the controller

$$\begin{aligned} T_t &= \frac{m(g-u_z)}{cos(\theta)} \\ u_z &= \sigma_{m_1}(\sigma_{m_2}(\tilde{z}) + \dot{z}) \end{aligned} \tag{9}$$

where $\sigma_a(x)$ is the saturation function and $\tilde{z} = z - z_{des}$. The closed-loop system will result as follow

$$\begin{aligned} \ddot{z} &= u_z \\ \ddot{x} &= tan(\theta)(u_z - g) \\ \ddot{\theta} &= -\frac{l_1 \cdot T_d}{I_{yy}} \end{aligned} \tag{10}$$

Defining the following dynamic error, as the backstepping procedure proposes:

$$\begin{aligned} z_1 &= v_1 - k_2\theta \\ v_1 &= \sigma_a(\tilde{x}) + \sigma_b(\dot{x}) \end{aligned}$$

where v_1 is the virtual controller. From the above equation, we can define $\theta = \frac{1}{k_2}(v_1 + z_1)$. Taking the derivative we obtain:

$$\dot{z_1} = \dot{\sigma_a}(\tilde{x})\dot{x} - k_2\dot{\theta} + \dot{\sigma_b}(\dot{x})tan(\theta)(u_z - g)$$

Defining the following dynamic of the error

$$\begin{aligned} z_2 &= \dot{\sigma_a}(\tilde{x})\dot{x} - k_2\dot{\theta} - \eta \\ \eta &= -cz_1 - tan(\dot{\theta})\dot{\sigma_b}(\dot{x})(u_z - g) \end{aligned} \tag{11}$$

As usual, taking the derivative of the z_2 we obtain our controller.

The controller calculated using the backstepping approach is the follow

$$T_d = \frac{1}{w \cdot k_2}(k_3 z_2 + \Phi + z_1) \tag{12}$$

where $w = \frac{-I_{yy}}{l_1}$. The controller is such that $(\tilde{x}, \tilde{z}, \theta) \to 0$ as $t \to \infty$

2.2.3 Stability proof

In order to prove the asympotitcally stability of the control strategy proposed we can pick the following Lyapunov function:

$$V_z = \frac{1}{2}z_1^2 + \frac{1}{2}z_2^2$$

Taking the derivative of V_z along the solution we obtain

$$\dot{V}_z = -cz_1^2 - k3z_2^2 \tag{13}$$

Picking the value of $\lambda = min(c, k_3)$ we can define an upper-bound for the $\dot{V}_z$ as follow :

$$\dot{V}_z \leq -\lambda V_z$$

Due to that $V_z > 0$ we proved the asymptotically stability of the controller (16).

2.2.4 Simulation using Backstepping

$$\theta_{des} = 0 \quad x_{des} = 5 \quad z_{des} = 10$$
$$\theta_0 = \frac{\pi}{8} \quad x_0 = 0 \quad z_0 = 5$$

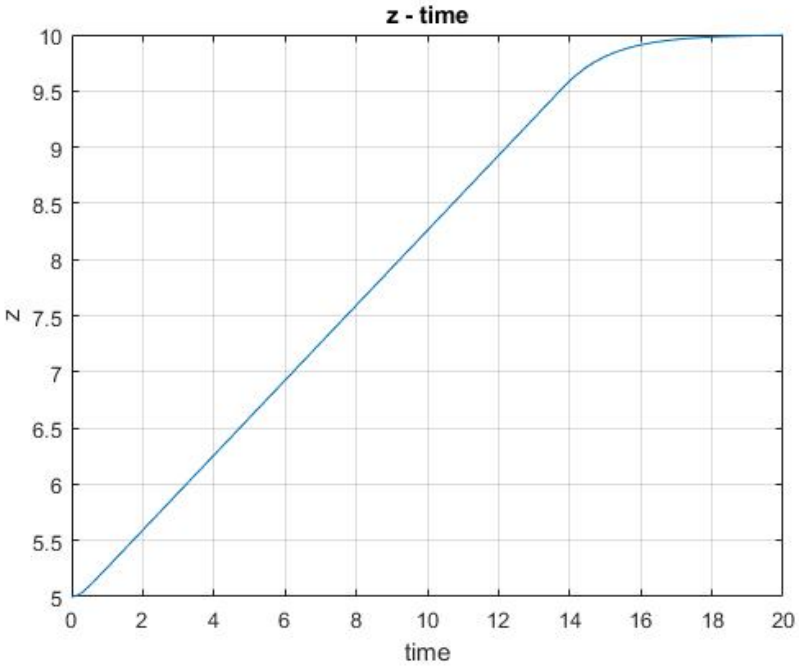


Figure 14: Hover Flight z position

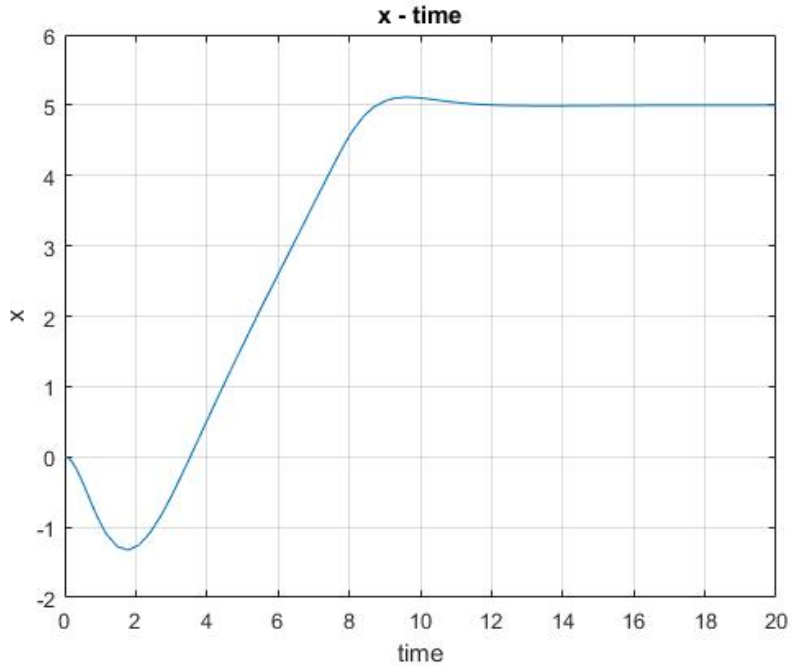


Figure 15: Hover Flight x position

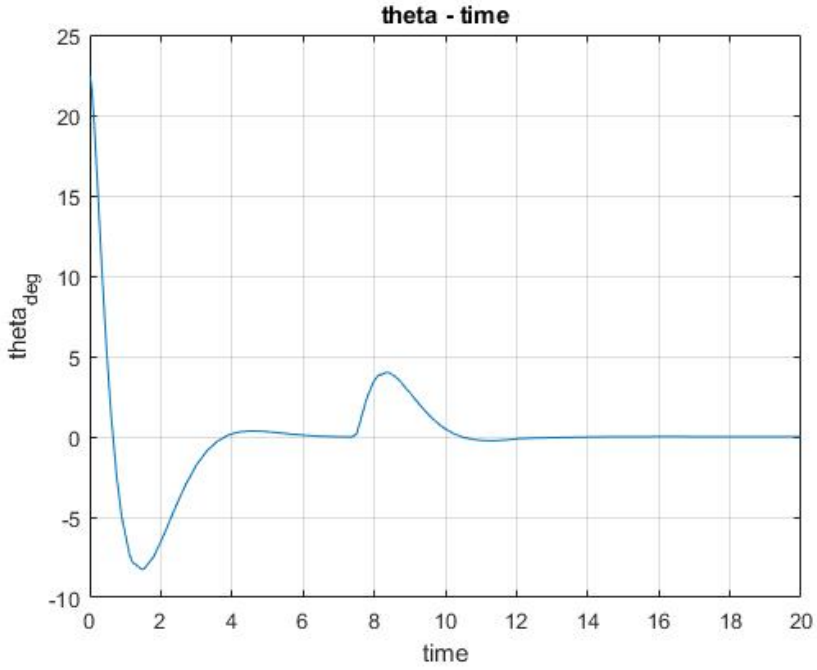


Figure 16: Hover Flight θ angle

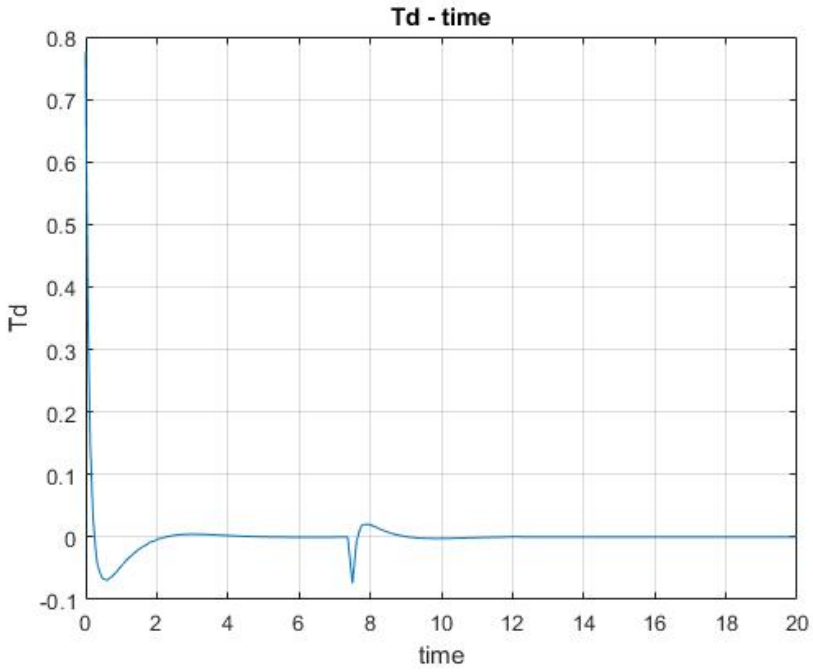


Figure 17: Hover Flight T_d control

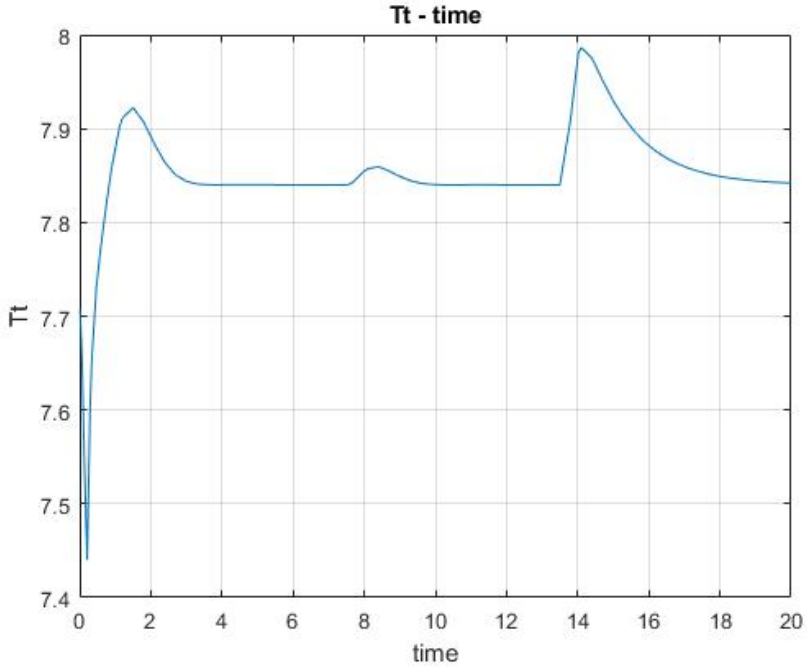


Figure 18: Hover Flight T_t control

2.2.5 Simulation with disturbances

In this section will be showed the results obtained simulating the model in presence of disturbances on the angle θ and a variation of the *payload*.
The references and the initial conditions are the same of the previous simulation.
The disturbances used are plotted in the first two figures below

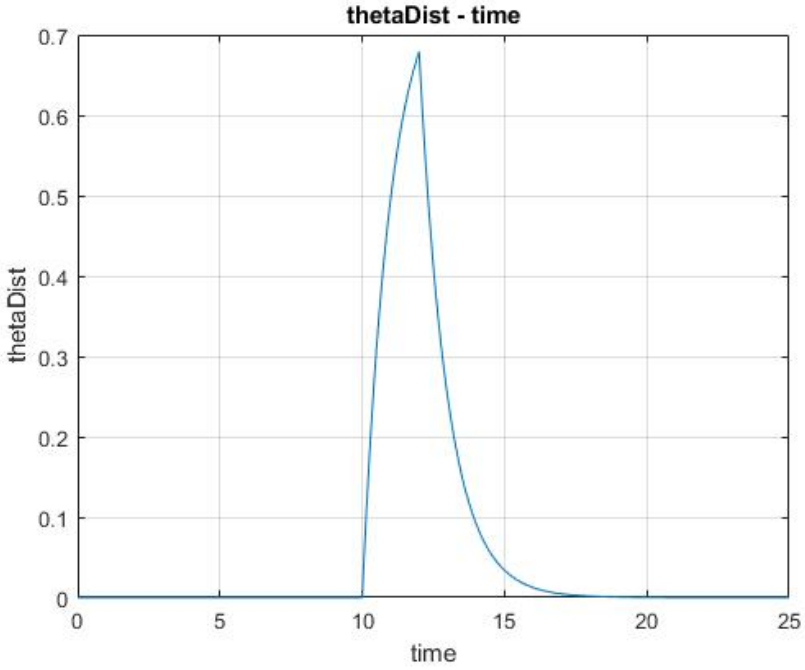


Figure 19: Disturbance on θ

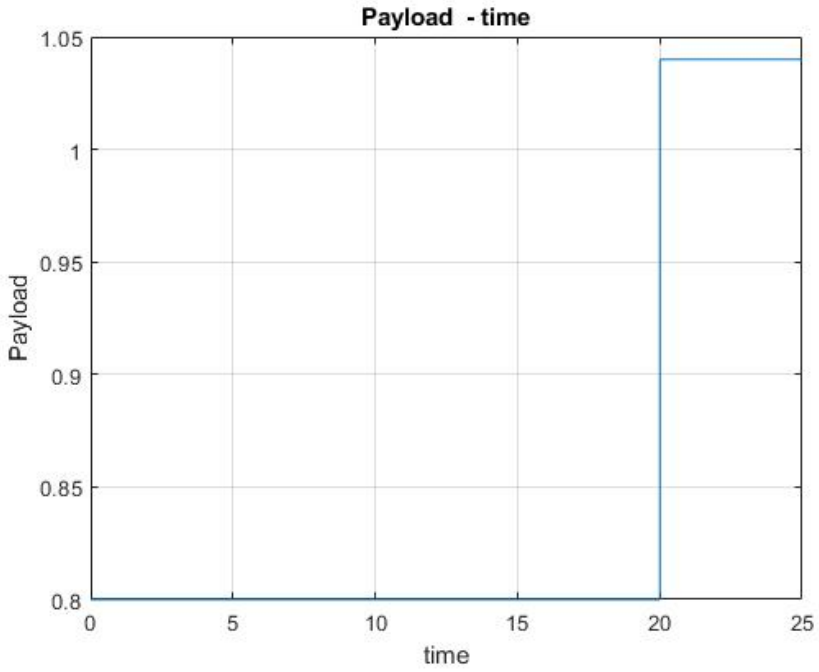


Figure 20: Variation of payload

The disturbance on θ starts at $t = 10$ and it has an amplitude of $\frac{\pi}{4}$, which is a really high value considering a realistic scenario. The *payload* will be increased of +30%

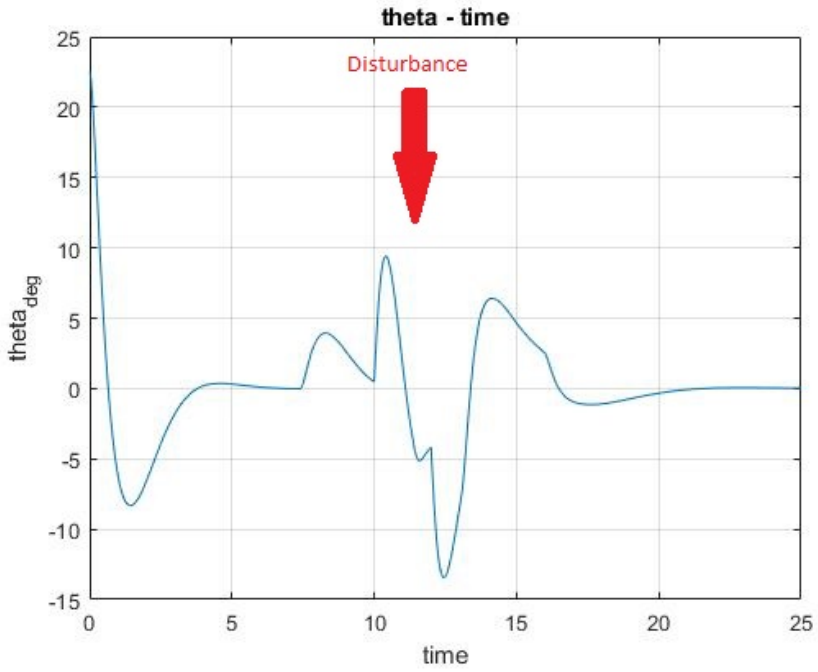


Figure 21: Hover Flight θ angle

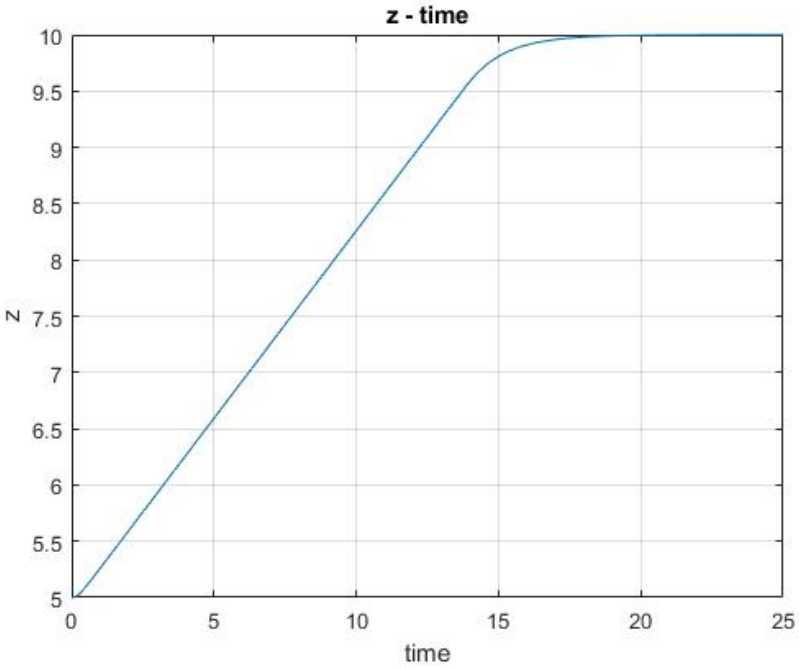


Figure 22: Hover Flight z position

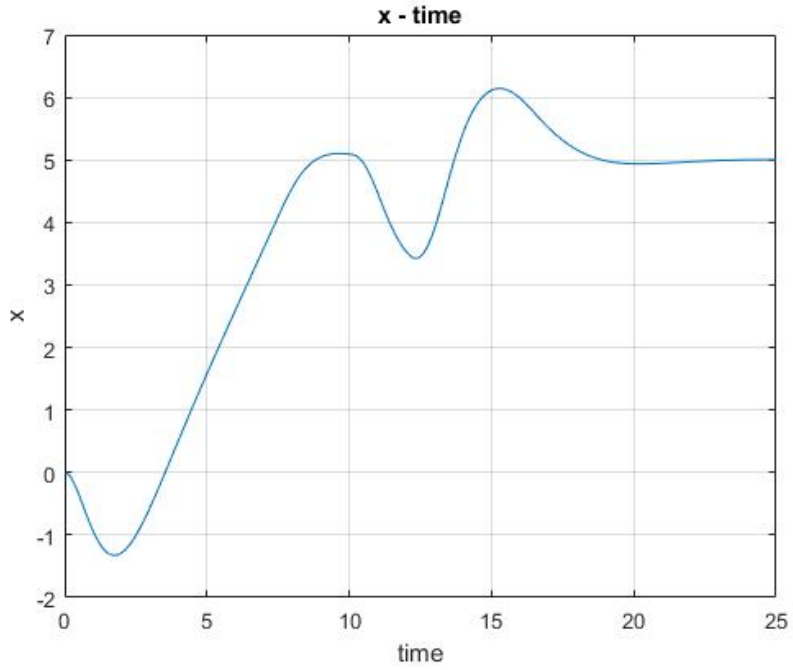


Figure 23: Hover Flight x position

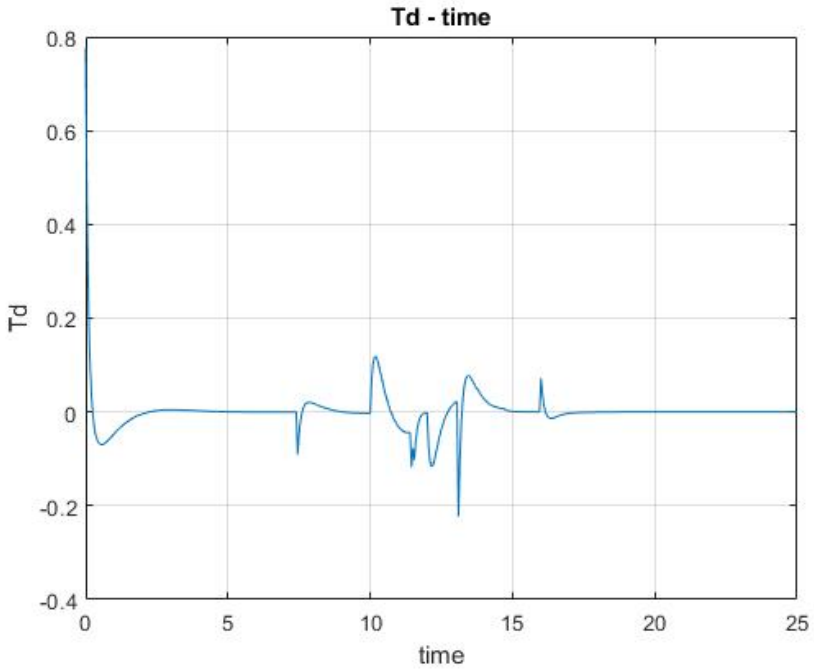


Figure 24: Hover Flight T_d control

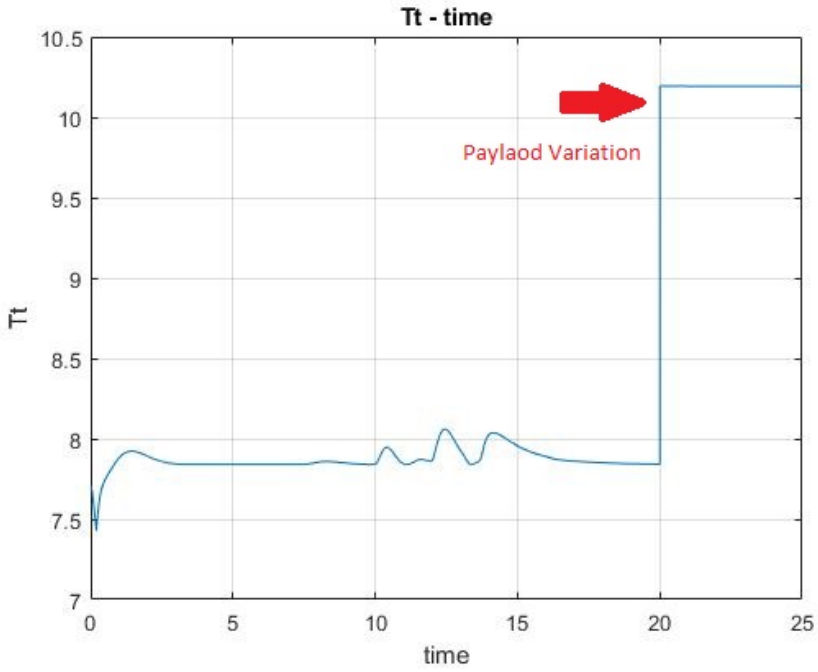


Figure 25: Hover Flight T_t control

Due to the variation of the payload the control input T_t will be increased in order to stabilize the gravity's force mg.

3 Control forward flight mode

3.1 Control Strategy

The flight path angle Γ is controlled using the backstepping algorithm.
In order to use this approach the dynamics of the vehicle has been re-written in the following form :

$$\begin{aligned} \dot{x} &= f(x) + \alpha \\ \dot{\alpha} &= f_1(x, \alpha) + q \\ \dot{q} &= f_2(\alpha) u \end{aligned} \tag{14}$$

where u is the control of the angle of the elevator δ_e and the functions and the state's variables are defined as follow:

$$\begin{aligned} f(x) &= -\frac{g}{V} \cdot cos(\frac{C_{l\alpha} x}{mV}) \\ f_1(x, \alpha) &= \frac{g}{V} cos(\frac{C_{l\alpha} x}{mV}) - \frac{C_{l\alpha} \alpha}{mV} \\ g_2(\alpha) &= \frac{C_{m\delta_e}}{I_{yy}} \end{aligned} \tag{15}$$

where $x = \dfrac{mV\Gamma}{C_{l\alpha}}$, α = angle of attack, q = pitch rate.
Following the standard procedure for the backstepping, an easy choice of Lyapunov function is $V_1(x) = 0.5 \cdot x^2$. Taking the derivative along the solution we can find that :

$$\dot{V}(x) = x(f(x) + \phi_1(x))$$

where $\phi_1(x)$ is the *virtual controller*. The easiest choice of the virtual control, in order to achieve $\dot{V_1}(x) < 0$ is $\phi_1(x) = -f(x) - k_1 x$, where $k_1 > 0 \in R$.
We can re-write the system defining the following error :

$$\begin{aligned} e_1 = \alpha - \phi_1(x) \rightarrow \dot{e_1} &= f_1(x, \alpha) + q - \dot{\phi_1}(x) \\ \dot{\phi_1}(x) = \frac{\partial \phi_1(x)}{\partial x} \dot{x} &= (\frac{-df(x)}{dx} - k_1) \cdot (f(x) + \alpha) \end{aligned} \tag{16}$$

Taking into account the dynamics of e_1, we can define the following Lyapunov function $V_2(x, e_1) = 0.5x^2 + 0.5e_1^2$. As the previous step, taking the derivative of V_2 along the solution we can find the following result :

$$\dot{V_2}(x) = -k_1 x^2 + e_1(x + f_1 - \dot{\phi_1} + \phi_2)$$

where in this case ϕ_2 is the *virtual control* as well.
Picking $\phi_2 = -x - f1 + \dot{\phi_1} - k_2 e_2$ we easily reach $V_2(x) < 0$. Defining the follow error $e_2 = q - \phi_2(x, e_1)$ and calculating the dynamics of e_2, we can find the following expressions :

$$\dot{e_2} = f_2 + g_2 u - \dot{\phi}_2$$
$$\dot{\phi}_2 = \frac{\partial \phi_2}{\partial x}\dot{x} + \frac{\partial \phi_2}{\partial e_1}\dot{e_1} = (-\frac{df_1}{dx} - 1 + \frac{\dot{\phi_1}}{dx})\dot{x} - k_1\dot{e_1}.$$

Defining the following Lyapunov function: $V_3(x) = V_2(x) + 0.5e_3^2$ and taking the derivative along the solution we can find as follow :

$$\begin{aligned} \dot{V_3}(x) &= -k_1 e_1^2 - k_2 e_1^2 + e_2(e_1 + f_2 + g_2 u - \dot{\phi}_2) \\ u &= \frac{1}{g_2}(-f_2 - e_1 - k_3 e_2 + \dot{\phi}_2) \end{aligned} \tag{17}$$

Taking the *real control* u as chosen in (3) we can reach our purpose.
From the value of x is easy to find the *the angle of path* as follow : $\Gamma = \frac{C_{l\alpha}}{mV}$.

3.1.1 Simulation Forward Flight

The initial conditions have been set to zero and the $\gamma_{des} = 3°$.

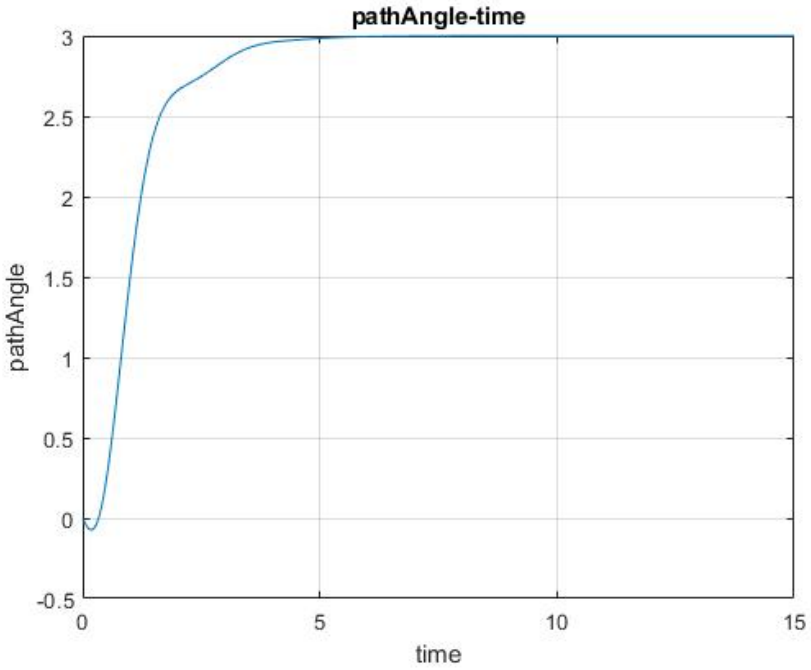


Figure 26: Forward Flight γ path angle

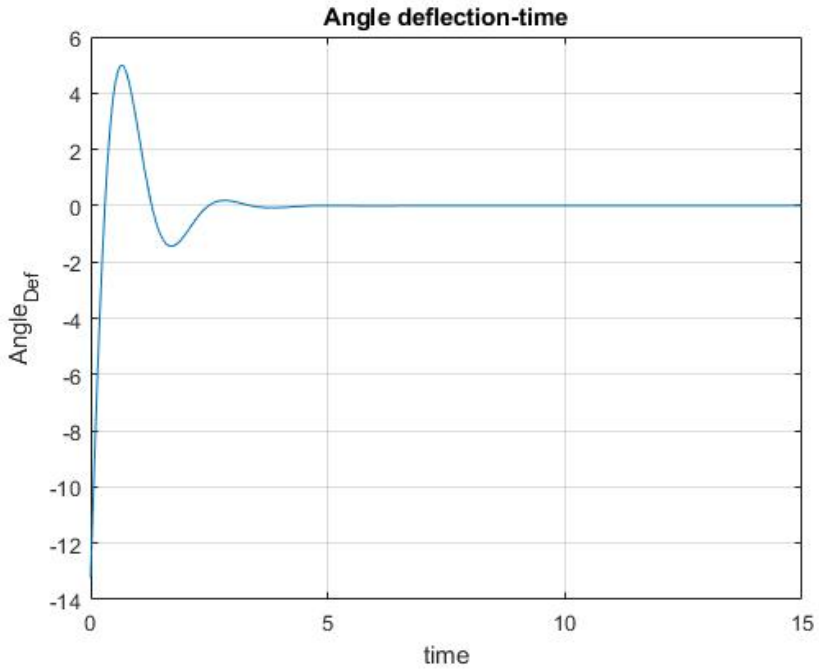


Figure 27: Forward Flight δ_e control

4 Transition flight mode

The *transition-flight* mode is the link between the *hover-flight* mode e *forward-flight* mode. This could be considered as the most complex dynamics. We take some assumptions for this mode:
• The angle $\theta \approx 0$ which is a good assumption, because this flight mode will occur after the *hover-flight* mode, where one of the scope is to stabilize the angle θ to 0.
• The term $u_e\dot{x}^2$ is the effect of the elevator deflection on the tail, and it will be different from zero iff $l\dot{x}^2 \geq mg$, where the *LHS* is the lift force.

4.1 Control strategy

Under the above assumptions the dynamic of the system will be the following

$$\begin{aligned} \ddot{x} &= Tsin(\gamma) - d\dot{x}^2 \\ \ddot{z} &= Tcos(\gamma) + l\dot{x}^2 + u_e\dot{x}^2 - mg \\ \ddot{x} &= Tsin(\gamma) - d\dot{x}^2 \\ \ddot{z} &= Tcos(\gamma) + l\dot{x}^2 - mg \end{aligned} \tag{18}$$

where the $d\dot{x}^2$ is the *drag force*, γ is the angle between the axis z_b and the vector T of thrust generated by the 4 motors.
The angle γ will change during the maneuver from $\gamma = 0 \rightarrow \dfrac{\pi}{2}$, so from the *hover-flight* mode to *forward-flight* mode, using the following function

$$\gamma(t) = \gamma_s t$$

where $\gamma_e > 0$ defines the speed of converging to the value of saturation $\frac{\pi}{2}$.
Focusing on the dynamic of the *z-axis* we can develop a controller, using a backstepping approach, in order to stabilize this axis before the condition $l\dot{x}^2 \geq mg$ holds.
Defining the following vector of state $[x_1, x_2] = [z, \dot{z}]$ the goal is to set $[x_1, x_2] = [x_{1des}, 0]$. So the equations became

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= Tcos(\gamma) + l\dot{x}^2 - mg \end{aligned} \tag{19}$$

It is easy to initialize the backstepping's procedure picking the first virtual controller $\Phi_z = -k_1(x_1 - x_{1des})$, where $k_1 \geq 0$. Adding and subtracting the virtual controller from the dynamic of x_1 we end up in

$$\begin{aligned} \dot{x}_1 &= e_1 - k_1(x_1 - x_{1des}) \\ e_1 &= x_2 - \Phi_z \rightarrow \dot{e}_1 = \dot{x}_2 - \dot{\Phi}_z \end{aligned}$$

where $\dot{\Phi}_z = \frac{\partial \Phi_z}{x_1}\dot{x}_1 = -k_1(e_1 - k_1(x_1 - x_{1des})$. For the dynamic of e_1 we can pick the real controller T as follow

$$T = \frac{1}{cos(\gamma)}(-x_1 - l\dot{x}^2 + mg + \dot{\Phi}_z - k_2 e_1) \tag{20}$$

where $k_2 \geq 0$. Using the following Lyapunov function we can prove that the origin is asymptotically stable : $V(x_1, e_1) = 0.5x_1^2 + 0.5e_1^2$. Taking the derivative along the solution we obtain $\dot{V} = -k_1 x_1^2 - k_2 e_1^2$, hence the origin is globally asymptotically stable.
When the condition $l\dot{x}^2 \geq mg$ holds, we can start to use the controller u_e. In this condition the velocity will be sufficiently to generate enough lift force and we are going to use the controller proposed for the *forward-flight* mode.

4.2 Simulations

As the previous simulations the initial conditions have been set to zero and the reference $z_{des} = 10$.

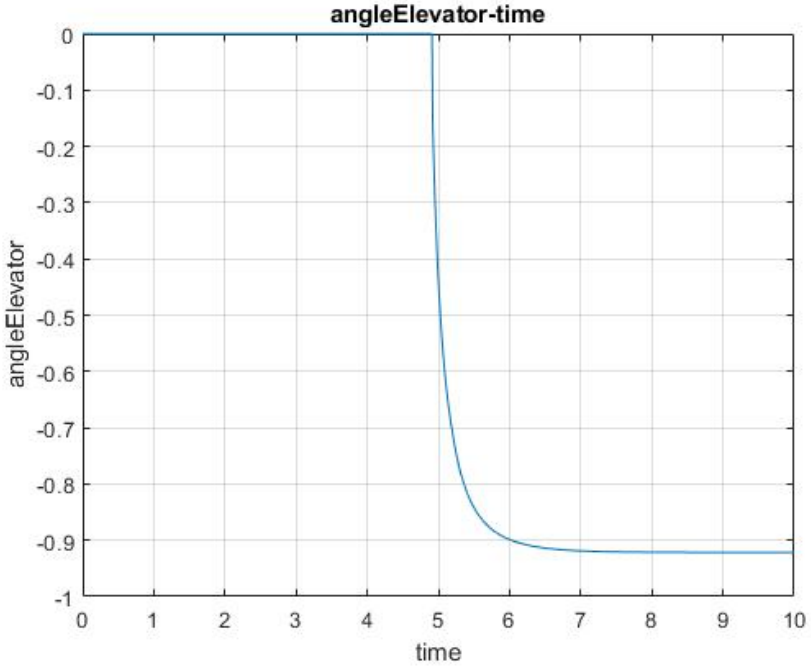


Figure 28: Transition Flight δ_e control

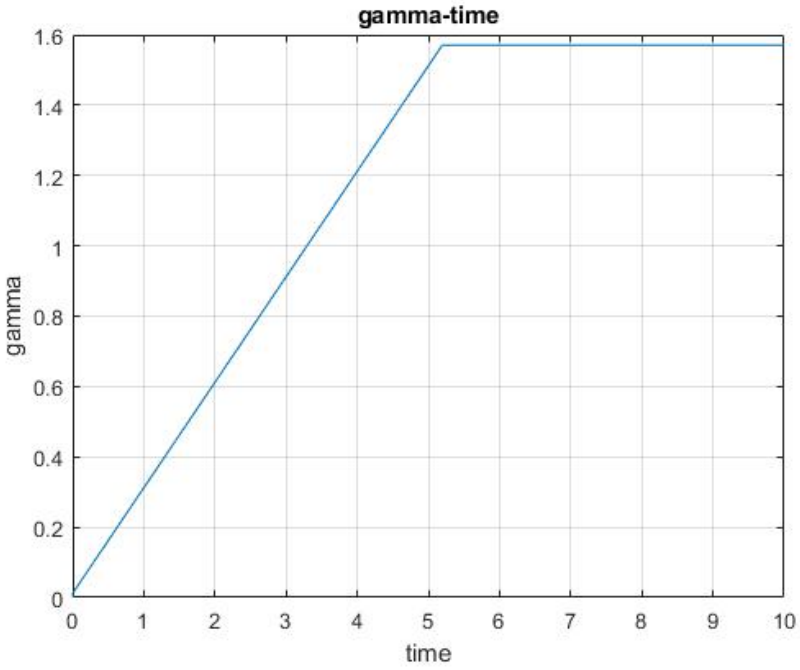


Figure 29: Transition Flight γ(angle motor)

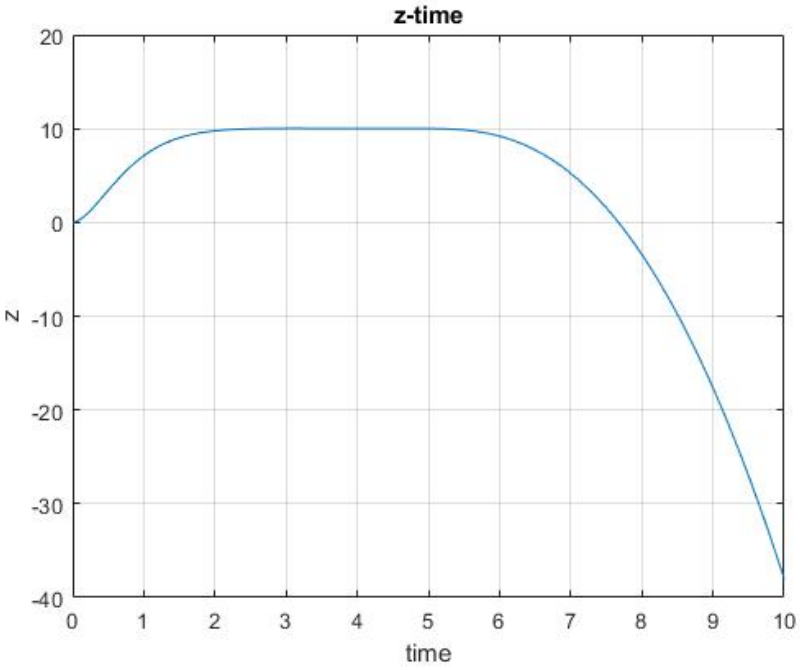


Figure 30: Transition Flight z

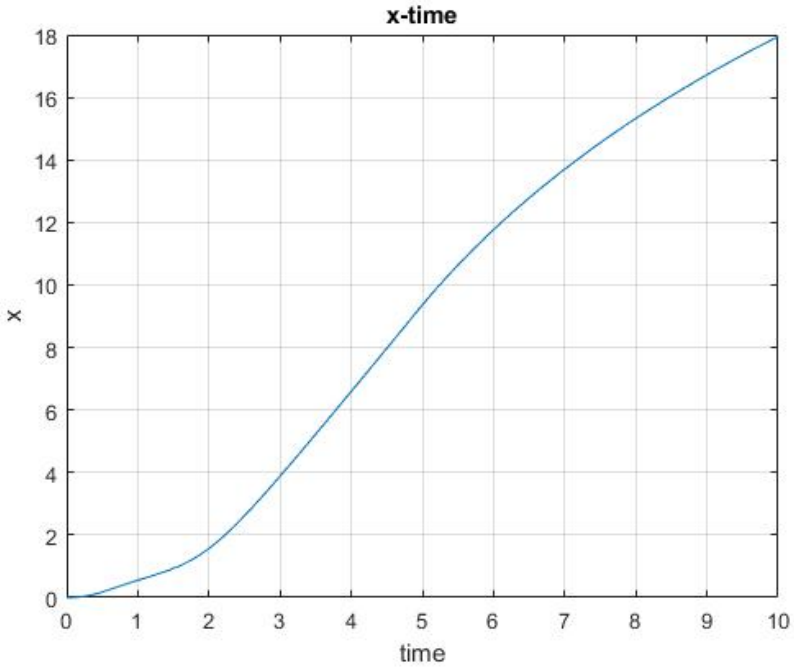


Figure 31: Transition Flight x

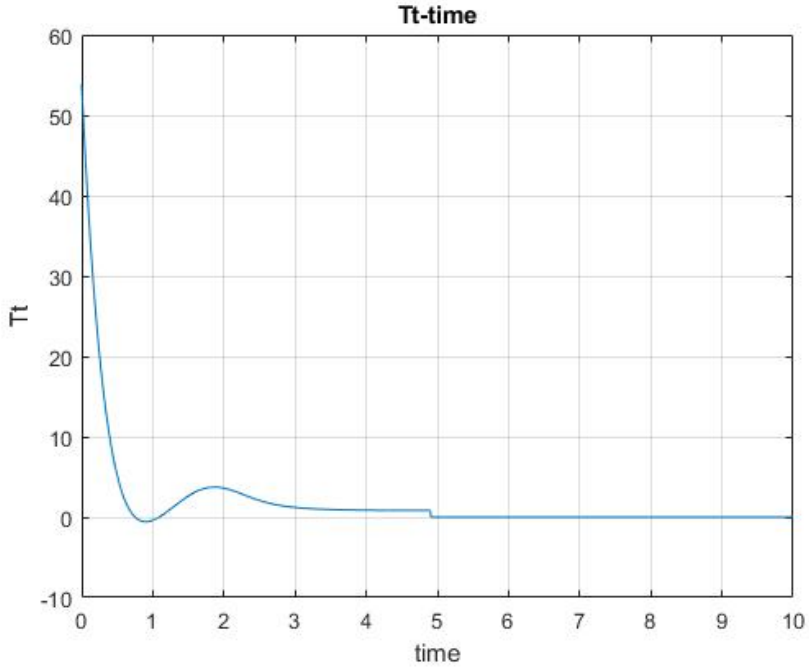


Figure 32: Transition Flight T_t

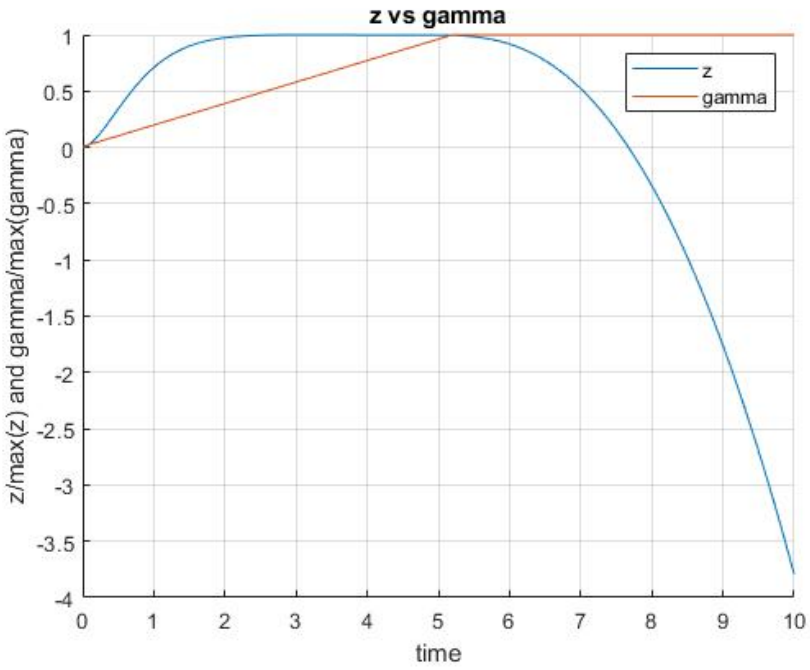


Figure 33: z vs γ

The last figure shows the z evolution vs the γ evolution, both of them normalized to 1. As it possible to see the z will start decreasing as soon as the angle γ reaches the value of $\frac{\pi}{2}$. At this point, as we are going to show in the simulation of the complete dynamic, the controller of the *forward-flight* mode will be activated.

5 Complete Dynamic

Up to this point all the three dynamics of the system, *hover-flight*, *transition-flight*, *forward-flight* modes have been controlled. So it is possible to merge all together the three different modes in order to simulate the system from the taking-off phase to the landing phase.

5.1 Finite state machine

In order to have a single work-flow of transition a *finite state machine* approach has been implemented. The machine is defined on 5 different states.
In the following figure is figured the machine

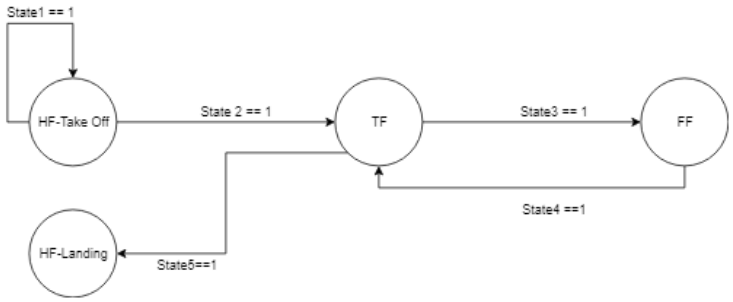
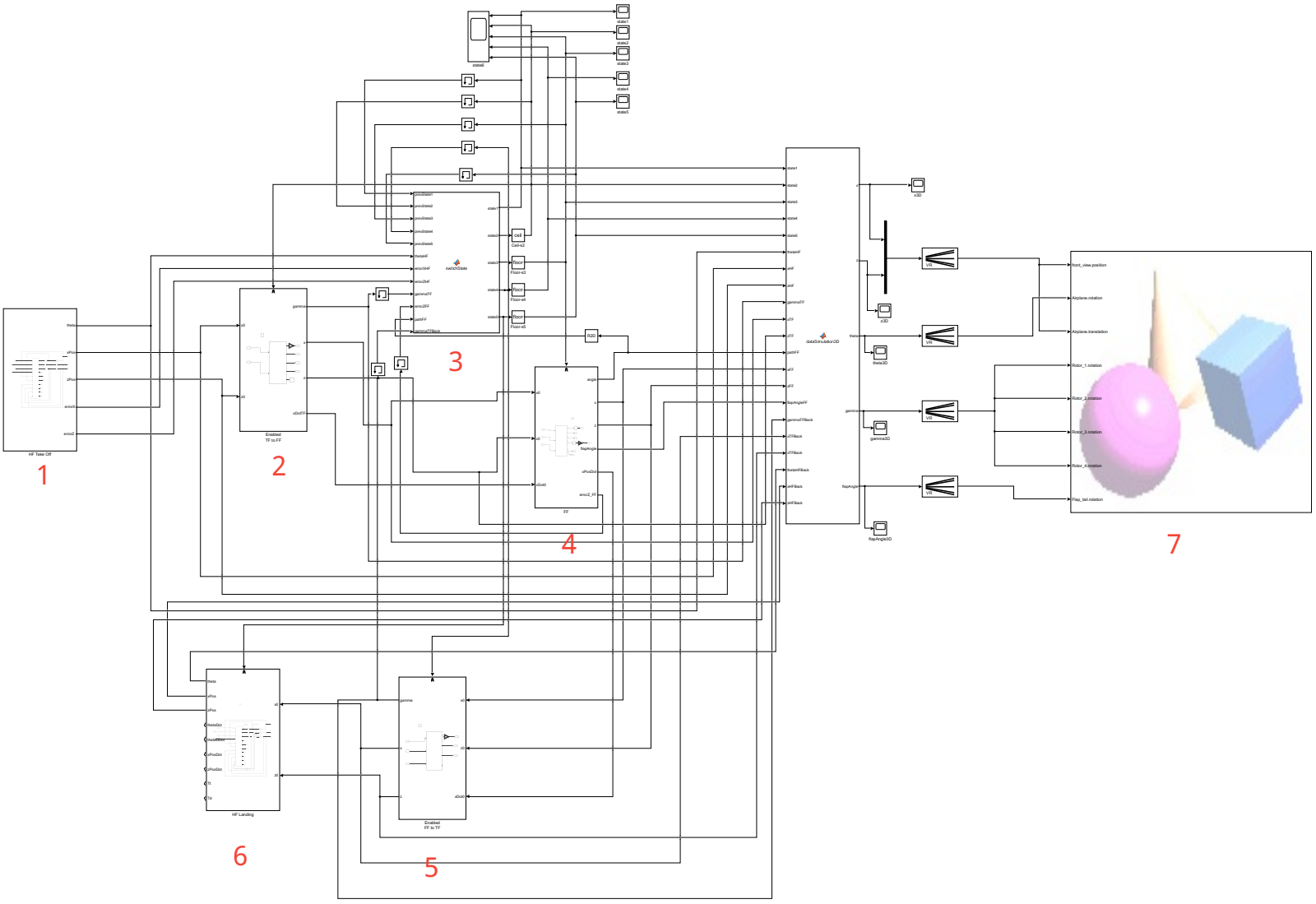


Figure 34: Finite state machine

The *state-i*'s variables are binary variables. The transition from a state to the next one happens when the variable state changes his value $0 \rightarrow 1$.
It is worthwhile to be pointed out that the *state-i*'s variables are mutually dependent, which means that at some instant t only one of them is 1 and the others must be 0.

5.1.1 MATLAB/Simulink environment

The system has been implemented in *MATLAB/Simulink* environment. In the following page the complete scheme used in Simulink is showed.


1
2
3
4
5
6
7

The core of the scheme is the block *switch state(3)*. It has as input the signal of the all different dynamics and it needs to change the state variable in order to active the different flight mode. Instead the blocks *1,2,4,5,6* are respectively the block dynamics of the *HF-Take off, TF, FF, TF back and HF-Landing*. The block *(7)* is the one that manages the *3D-Simulation* and it has as input the vector that contains the following information

$$[x, z, \theta, \gamma, \delta_e]$$

where the x and z are the positions of the vehicle, θ is the angle that it forms with the horizontal axis, γ is the angle of rotation of the 4-motors and δ_e is the flap-angle of the tail.

5.1.2 3D World

The 3D world for the graphic simulation has been created using the tool *VR Sink - Simulink*. In the following two figures is showed the 3D world realization where the simulation will be showed

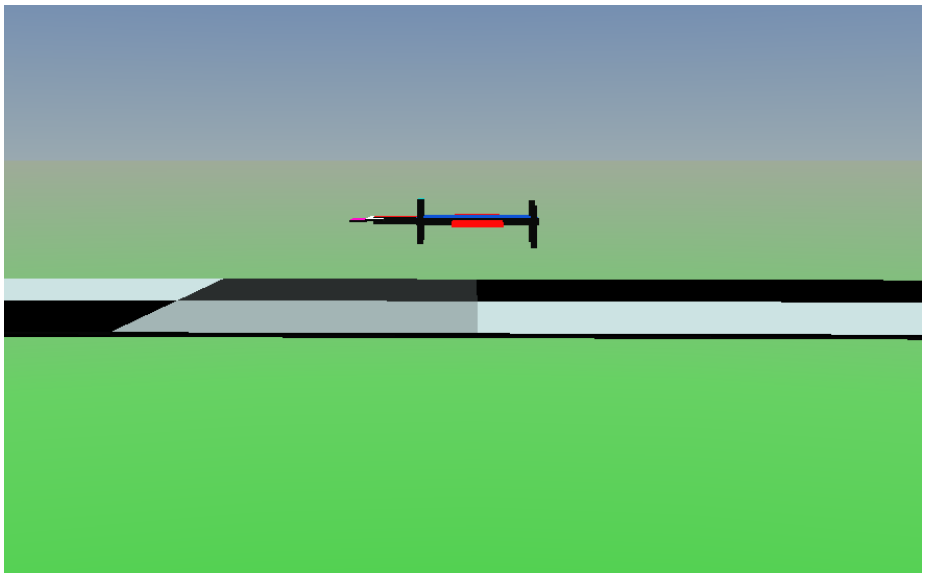


Figure 35: Front view 3D World

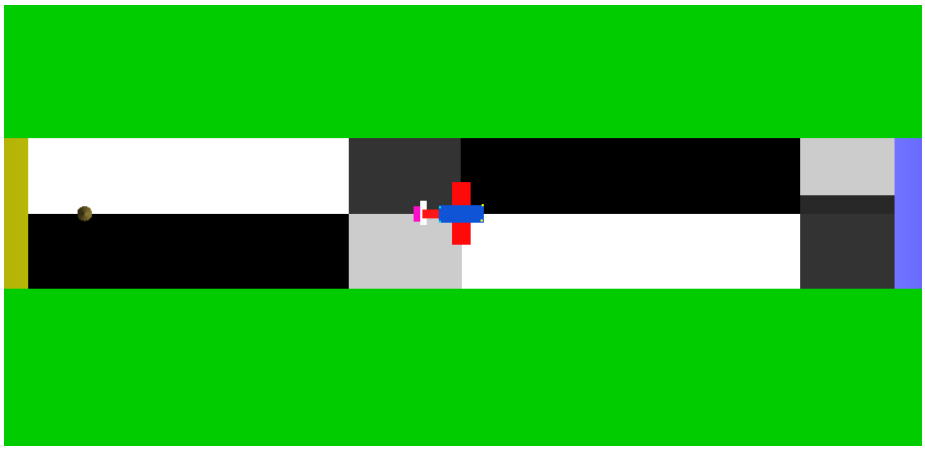


Figure 36: Top view 3D World

Considering the *top-view* it is possible to recognize the *yellow-zone* which will be the starting point of the trajectory and the *blue-zone* which will be ending point.

The 3D world has the following dimensions :

- The *terrain* is scale = $[120, 30, 50]$ and position $[0, 0, 0]$
- The *starting-point* is scale = $[5, 5, 20]$ and position $[-60, 0, 0]$
- The *ending-point* is scale = $[5, 5, 20]$ and position $[60, 0, 0]$

where scale and position are vectors in $[x, y, z]$ (the *y-axis* into the 3D-World corresponds to the *z-axis* of the dynamic of the system).

5.2 State Simulation transition

In the following figures are showed the values of the states and their evolution during the simulation. The complete state evolution is plotted in the following figures

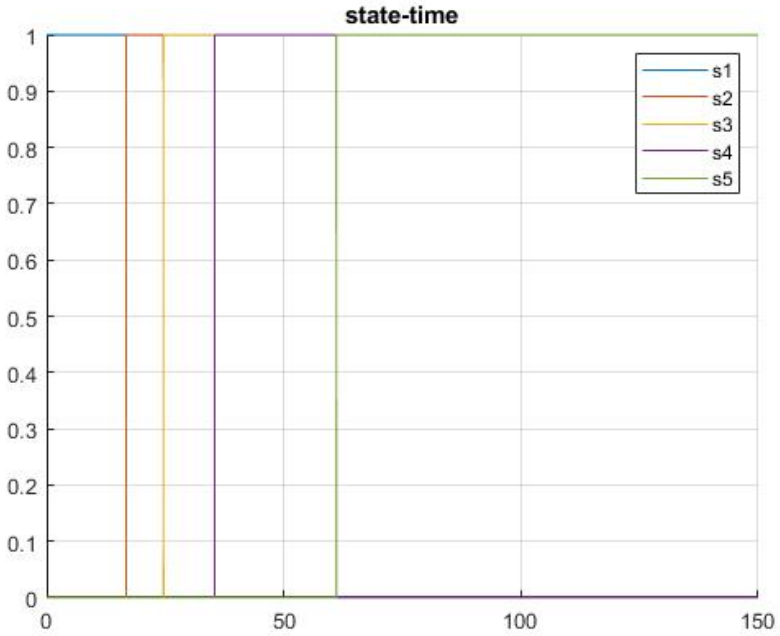


Figure 37: State transitions

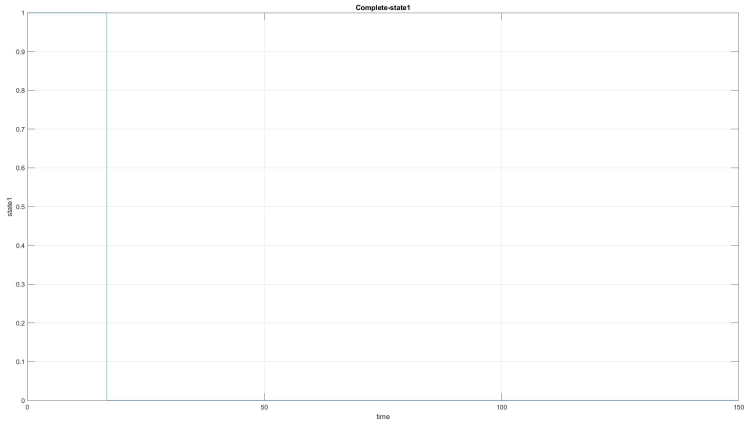


Figure 38: State 1 - HF taking off

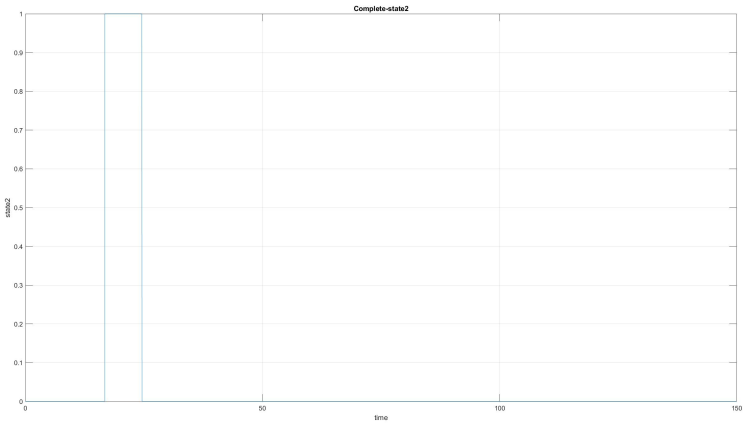


Figure 39: State 2 - TF - HF to FF

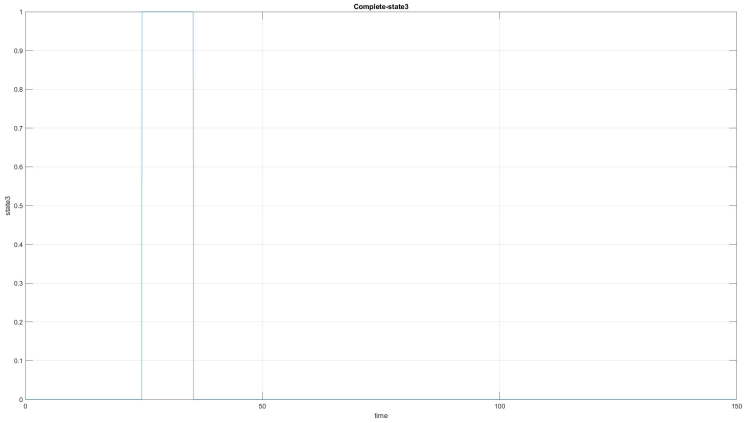


Figure 40: State 3 - FF

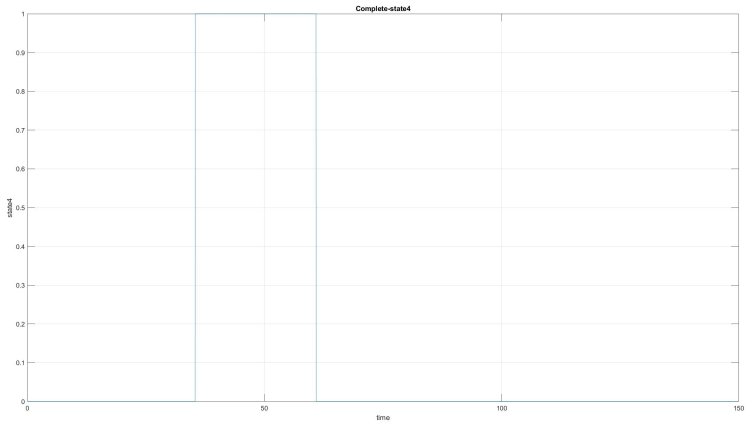


Figure 41: State 4 - TF - FF to HF

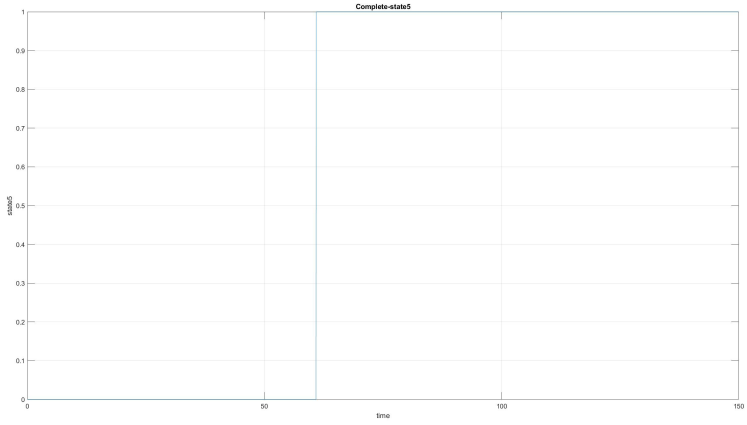


Figure 42: State 5 - HF landing

5.3 Simulations

The following figures show the simulation of the *complete-dynamic* of the system, from the take-off to the landing.
The system have been simulated using the following set of values

- Start $= [-60, 5, 0]$
- Take off reference $= [-50, 10, 0]$
- Landing reference $= [60, 5, 0]$

where the vector as usual are in the form $[x, z, \theta]$.

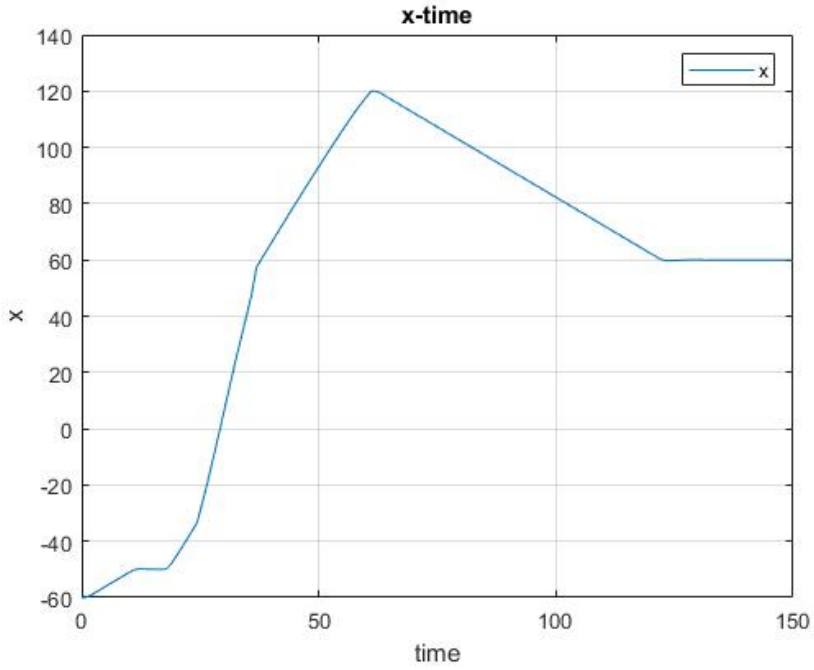


Figure 43: Evolution of x position

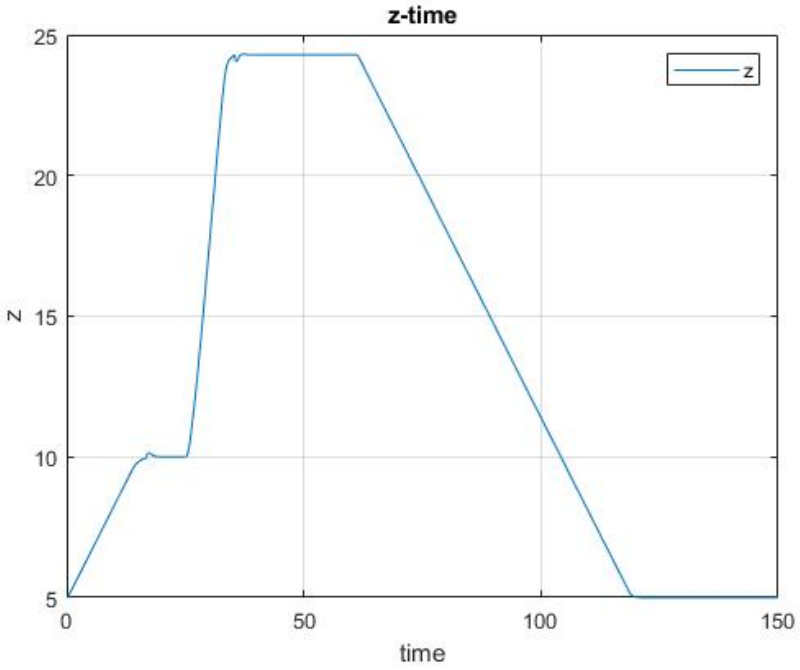


Figure 44: Evolution of z position

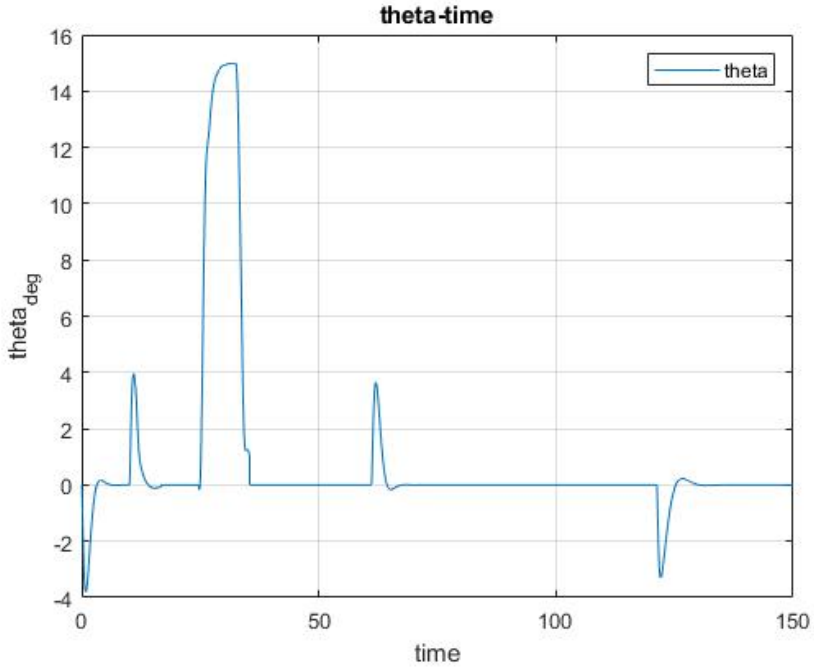


Figure 45: Evolution of θ

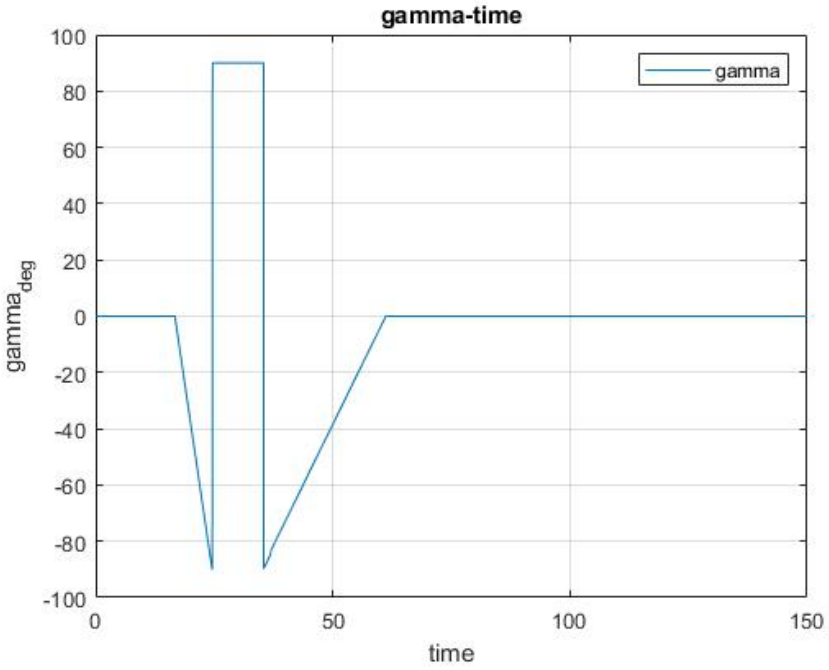


Figure 46: Evolution of γ

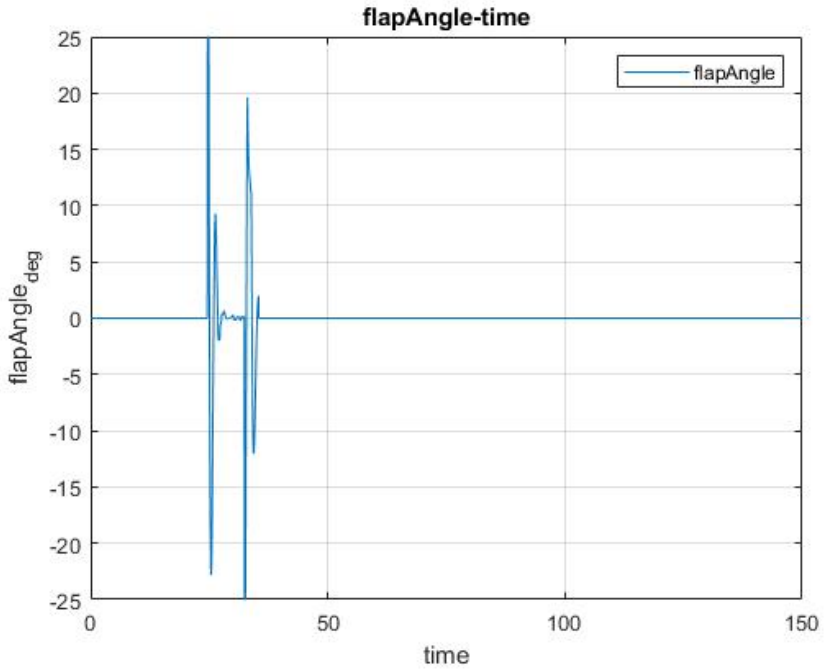


Figure 47: Evolution of δ_e

5.4 Simulations with disturbance

The following figures show the simulation of the *complete-dynamic* of the system, from the take-off to the landing in presence of disturbance.
The system have been simulated using the following set of values

- Start $= [-60, 5, 0]$
- Take off reference $= [-50, 10, 0]$
- Landing reference $= [60, 5, 0]$

where the vector as usual are in the form $[x, z, \theta]$.

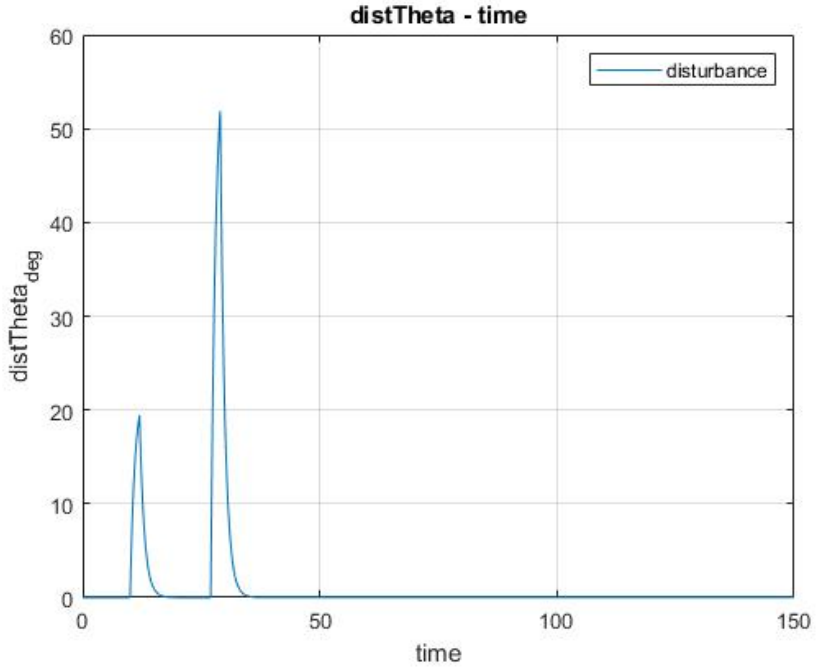


Figure 48: Disturbance

The disturbance is divided into two parts :

- The first part is a disturbance of magnitude $\frac{\pi}{4}$ applied on the *hover-flight* mode at $t = 10$.
- The second one is a disturbance of magnitude $\frac{\pi}{3}$ applied on the *forward-flight* mode at $t = 26$.

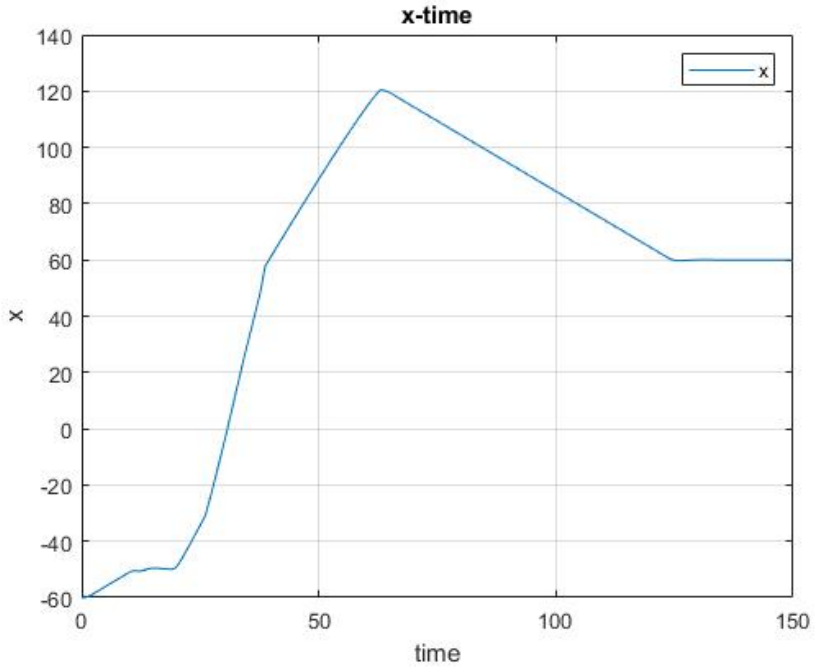


Figure 49: Evolution of x position

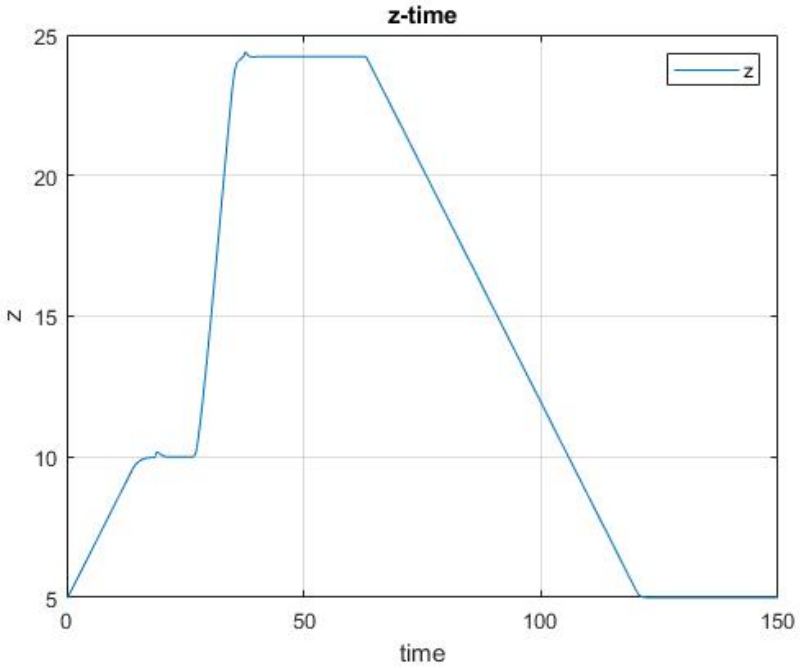


Figure 50: Evolution of z position

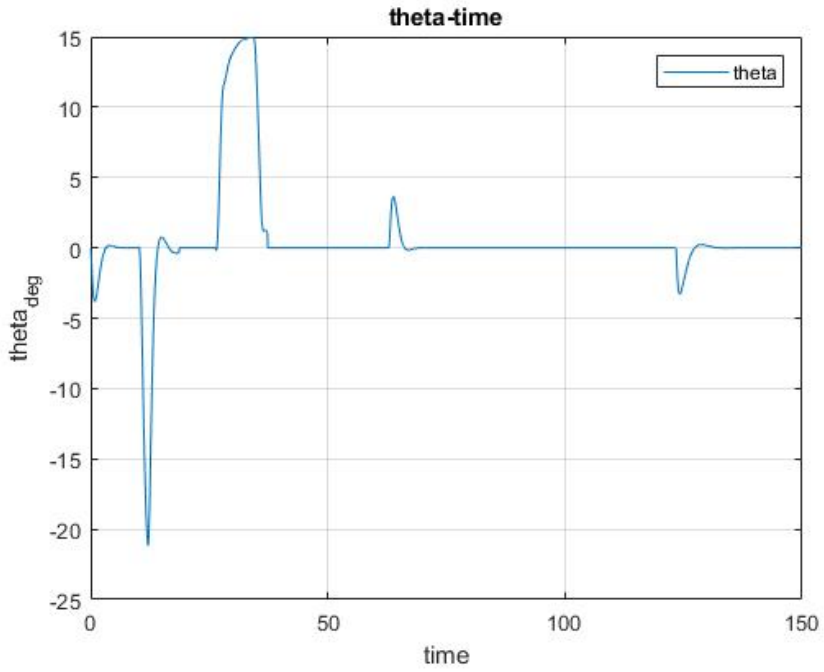


Figure 51: Evolution of θ

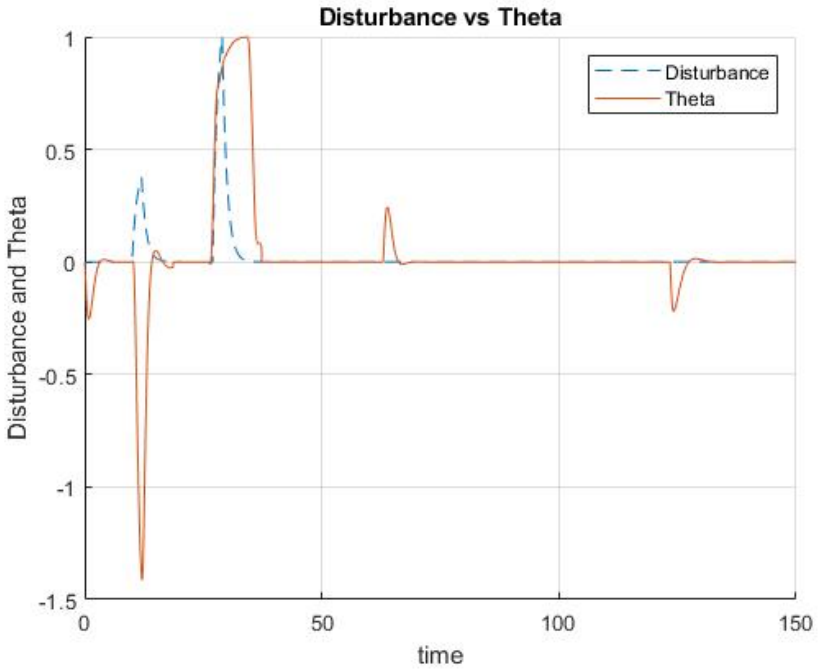


Figure 52: Evolution of θ vs Disturbance

In the last figure has been plotted the θ evolution and the disturbance's evolution, both of them normalized to 1. We can clearly see the effect of the disturbance and the reation of the system.

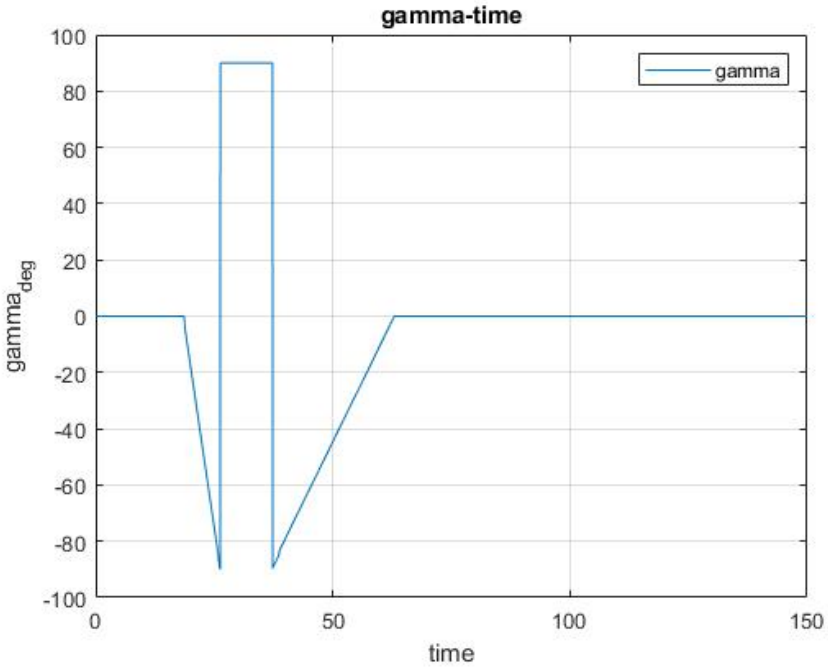


Figure 53: Evolution of γ

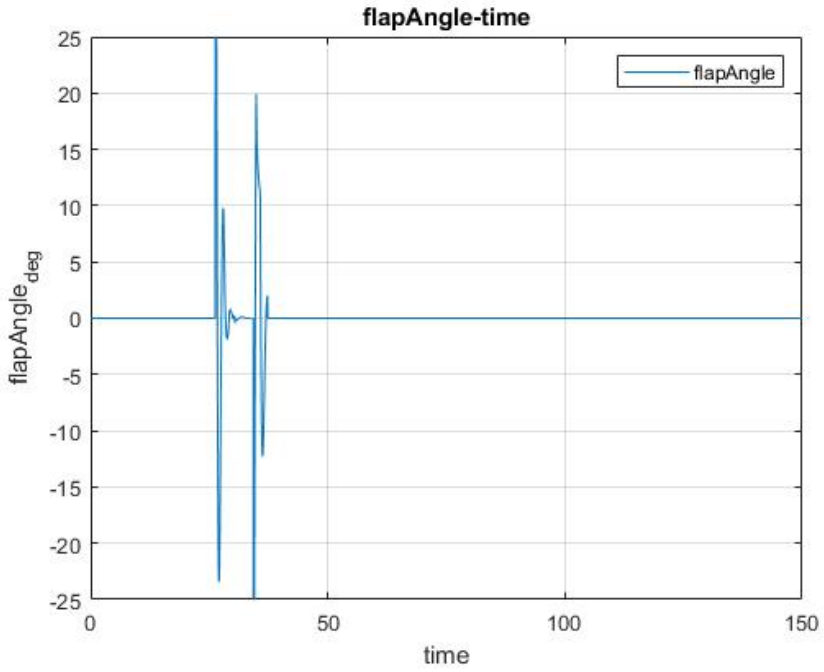


Figure 54: Evolution of δ_e

6 Swarm Control

Swarm behaviour, or swarming, is a collective behaviour exhibited by entities, particularly animals, of similar size which aggregate together, perhaps milling about the same spot or perhaps moving enmasse or migrating in some direction. From a more abstract point of view, swarm behaviour is the collective motion of a large number of self-propelled entities. Swarm behaviour was first simulated on a computer in 1986 with the simulation program boids.[10].

6.1 Centralized vs decentralized control

In the context of a small swarm there are two main categories of approach : *centralized control* and *decentralized control*.
The first one is the method where each swarm member(the single UAV) is controlled by a central controller. The member are fully dependent on the central controller, which means that a fault/error of the central controller messes up all the formation. From this point of view this kind of method is not widely used .
In *decentralized control* the individual member have their own controller. In this way the members of the swarm are autonomous. In this work we are going to use the decentralized approach.

6.2 Potential functions approach

Suppose N is the number of members in a swarm and collect the positions vector inside $r = [r_1, r_2, ..., r_N]^T$, where $r \in R^{3xN}$. Based on the potential field idea, the swarm is modeled as *potential function*, and the control law will try to avoid to came across the region where the potential energy is higher.
Defining the following function [11]:

$$J_{ij}(r_i, r_j) = \frac{1}{2}a \left\| r_i - r_j \right\|^2 + \frac{bc}{2} e^{\frac{-\|r_i - r_j\|^2}{c}} \tag{21}$$

it is possible to define the following potential function

$$J(r) = \sum_{i}^{N-1} \sum_{j=i+1}^{N} J(r_i, r_j) \tag{22}$$

The above function fulfills the following assumptions

- The potential function is symmetric with respect to the position vector hence $\nabla r_i J_{ij}(r_i, r_j) = -\nabla r_j J_{ij}(r_i, r_j)$
- It is possible to identify two different part inside the function: *attractive* and *repulsive* $J_{ij}(r_i, r_j) = J_a + J_r$
- There exist a unique distance δ_{ij} such that the repulsive and the attractive part are balanced.

Under the above assumptions the following theorems hold [11]

- The center of the swarm is given by $\bar{r} = \frac{1}{N}\sum r$ and it is stationary for all time t.
- If J(r) is lower-bounded then for any initial condition $r(t) \to \Omega$ as $t \to \infty$
- The swarm size will be bounded for all the time t.

In this way it is possible to define the *steering command* as follow

$$\dot{r}_i = -\nabla_{r_i} J(r), \quad i = 1..N \tag{23}$$

Using the defined function J_{ij} the steering command became

$$\dot{r}_i = -\sum_{j=1, j\neq i}^{N} (r_i - r_j)[a - be^{\frac{-\|r_i - r_j\|^2}{c}}] \tag{24}$$

It is worthwhile to be pointed out that the parameter a, b, c define the shape and the distance between each swarm member.
Using this approach it is just needed to add the *steering command* to the vector velocity of the model in order to obtain a *swarm-form*.
In the longitudinal case is worthwhile to be pointed out that it is necessary just a term of the potential energy on one axis (x *or* z) in order to maintain an avoiding-collide formation. In fact, as long as one of the two coordinates is different from the others, there is no way to collide. In this work the *x-axis* has been chosen.

6.2.1 Obstacle avoidance

Using the potential function approach it is really easy to implement obstacle avoidance into the control law. The obstacle will basically be behaving like a static member of the swarm. It is just necessary to add an another *extra-repulsive* term inside the steering command as follow

$$\dot{r}_{obs} = -\sum_{j=1, j\neq i}^{N} (r_i - r_{obs})[b_{obs} e^{\frac{-\|r_i - r_{obs}\|^2}{c_{obs}}}] \tag{25}$$

where the terms c_{obs} and b_{obs} define the *power* of the repulsive term of the obstacle. The exponential term is now used as repulsive part and there is no attraction in order to avoid them.
A graphical representation [12]of the potential field with obstacle is showed in the following figure where it is easy to notice the obstacle into the potential field

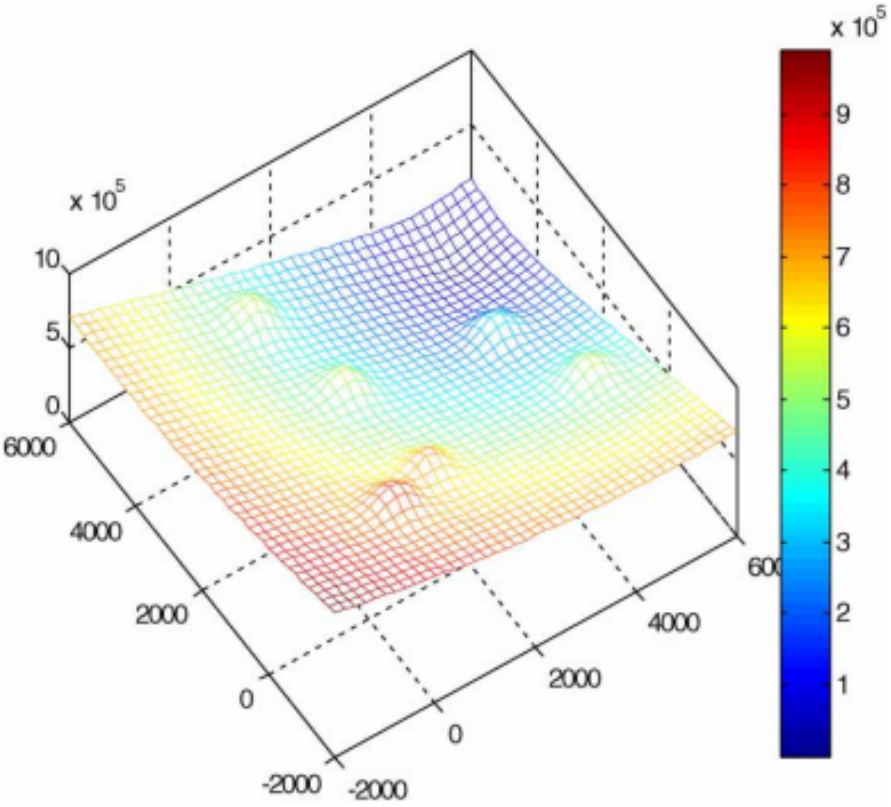


Figure 55: Potential Field with obstacle

6.3 Simulations

In the following figure are plotted the result of the simulations of a swarm-UAV with $N = 3$. The model used is the same of the *hover-flight* mode used before. The initial condition are defined as follow

$$x_0^1 = [10, 0]$$
$$x_0^2 = [0, 0]$$
$$x_0^3 = [-10, 0]$$

In order to prove the effectiveness of the control law proposed the reference has been set to $x_{ref} = [60, 10]$ and in this way it is possible to show that the all UAVs will avoid each other.

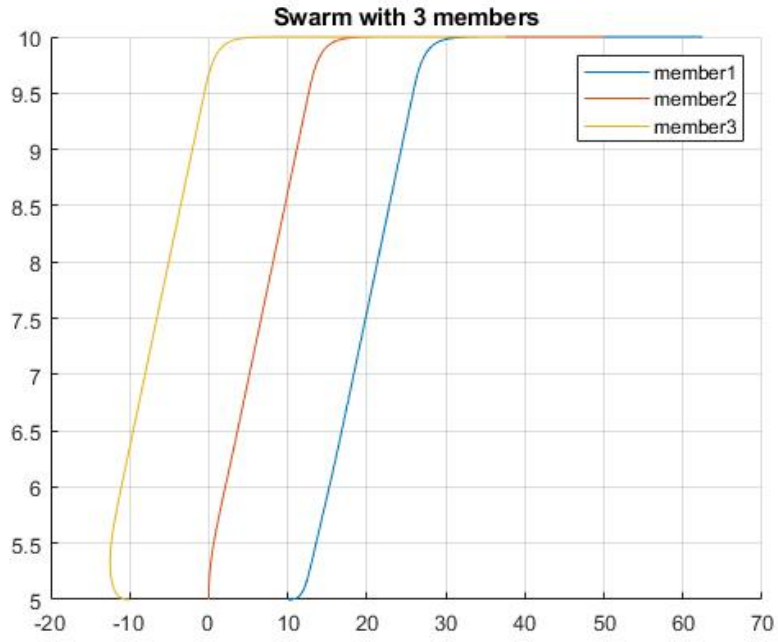


Figure 56: Swarm position (x,z)

A 3D simulation of the swarm-model is provided aswell.

7 Draw Path

Finally another function has been implemented in order to test the performances of the *hover-flight* mode which is the one that gives to us more freedom and on the *swarm* formation.

7.1 Drawing path

In order to create a different and various path has been developed a script *MATLAB* which makes a different path using the mouse.
A simple *star-path* drew using the script is showed in the following figure

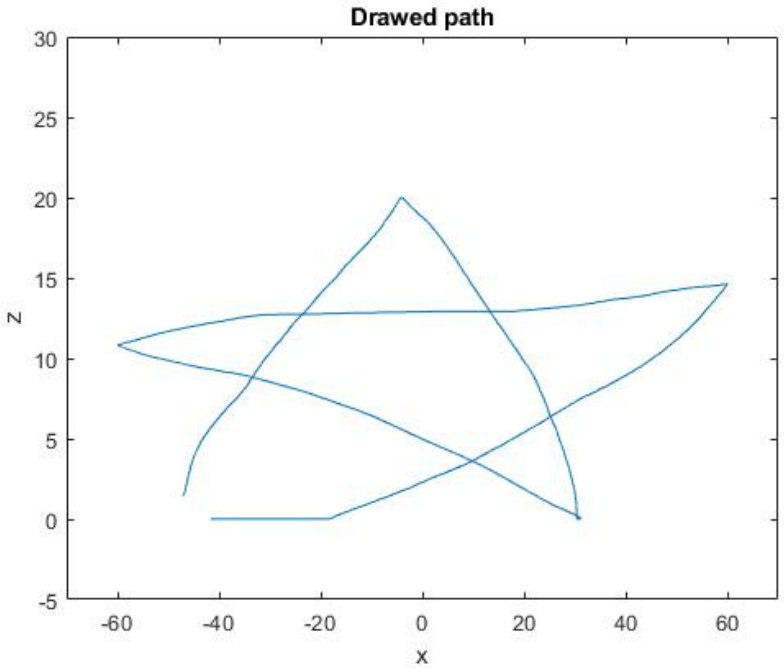


Figure 57: Star path

It is worthwhile to be pointed out that as the script is developed, there are no limits to the shape of the path that could be tested.
In order to set the reference for the system the shape drew into the *MATLAB* script must be discretized into different point. The choice of the number of point that are used as references point of the system influences the performance of the tracking of the shape. For the *Star-path* plotted above a good choice is to get around 500 points on a time window of *50s*(as showed on the code in the appendix).

7.2 Hover Flight with draw path

As we already said the performance of tracking of the path depends on the number of points. In the *hover-flight* mode a good choice is 500 points and the performances are good. In the following figure is showed the result of the tracking and the reference used

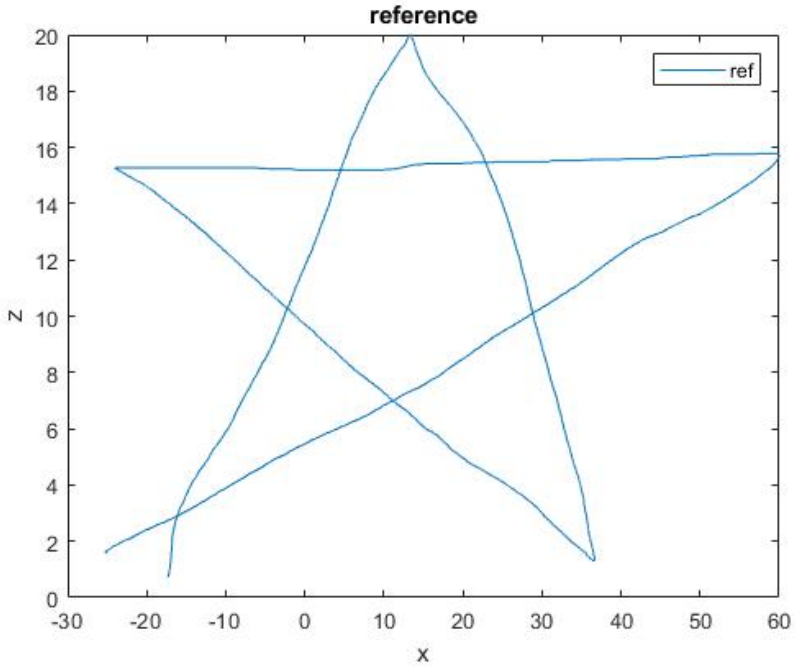


Figure 58: Reference

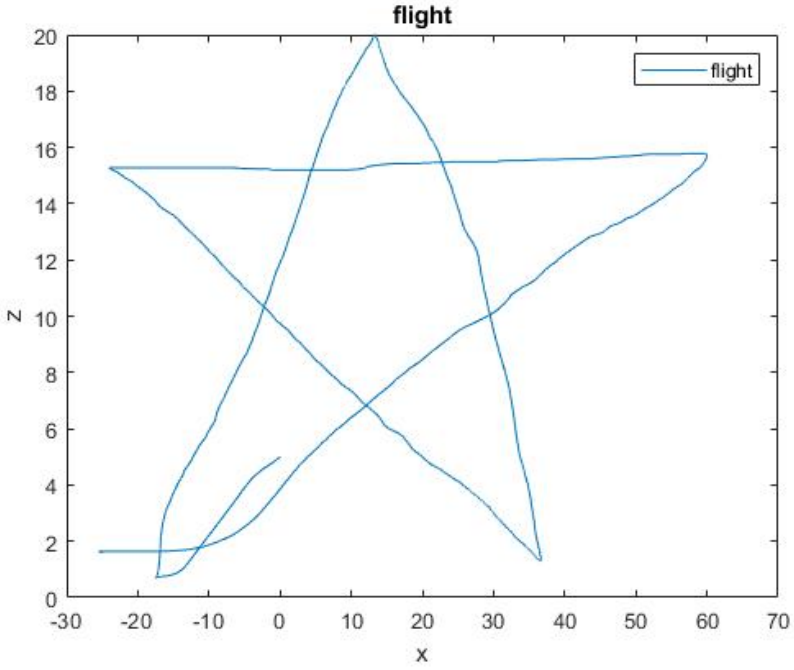


Figure 59: Flight path

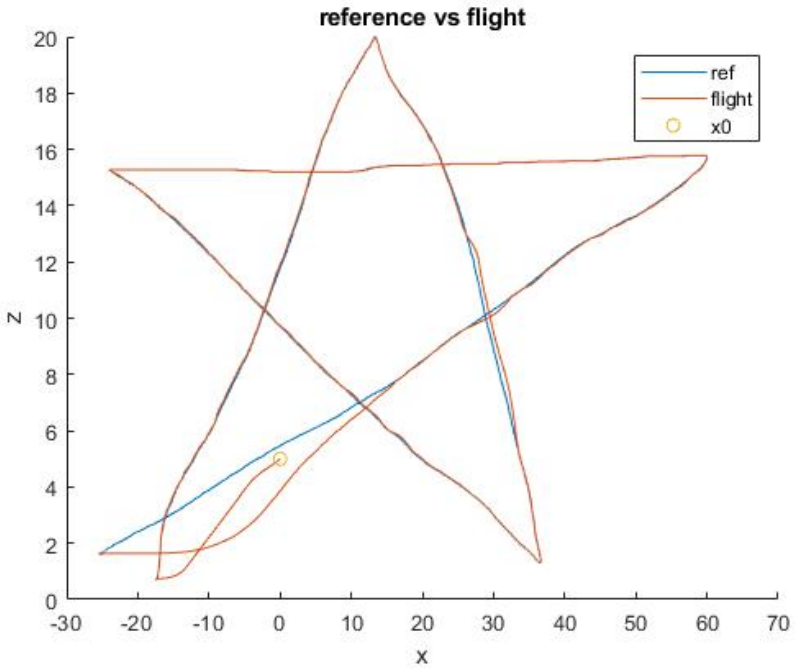


Figure 60: Flight path vs Reference

As showed and as expected the system goes to the *starting-point* of the path, and then it will follow the trajectory. Using a less amount of points the performance are getting worst.

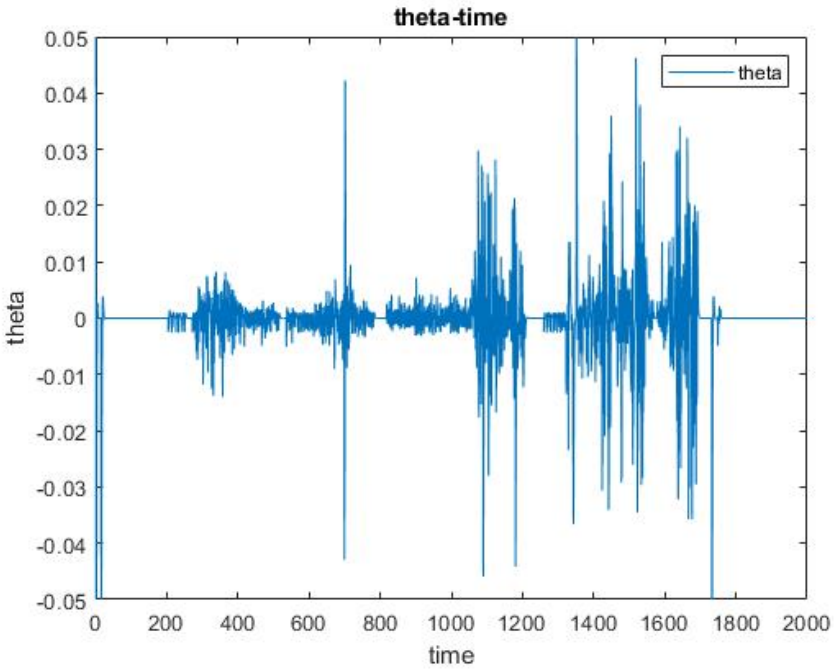


Figure 61: Hover Flight theta draw path

Due to the imprecision of the mouse used to shape the path, and due to the small vibration of the hand, the path will result not perfectly clear, which means that the θ angle of the vehicle will be inaccurate as showed in the figure above. A filter as future work could be implemented in order to clean the path. As the previous simulations, a *3D-simulation* is provided as well.

In the following two figures are showed the differences of performance of the tracking of the path using 1000 and 500 points.
As it is already announced the performance are much better using 1000 points, especially during the tight turns.

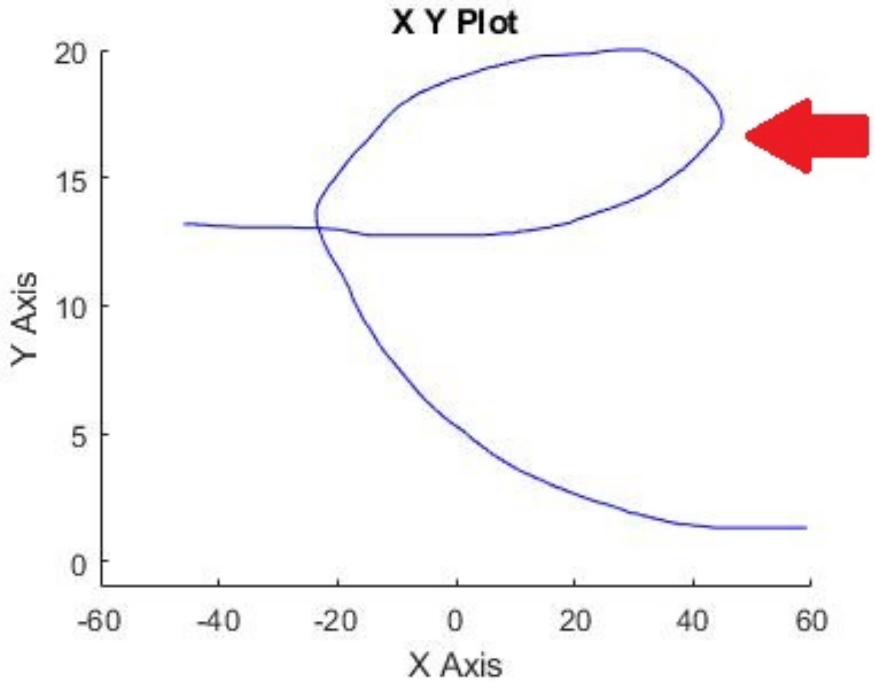


Figure 62: Reference

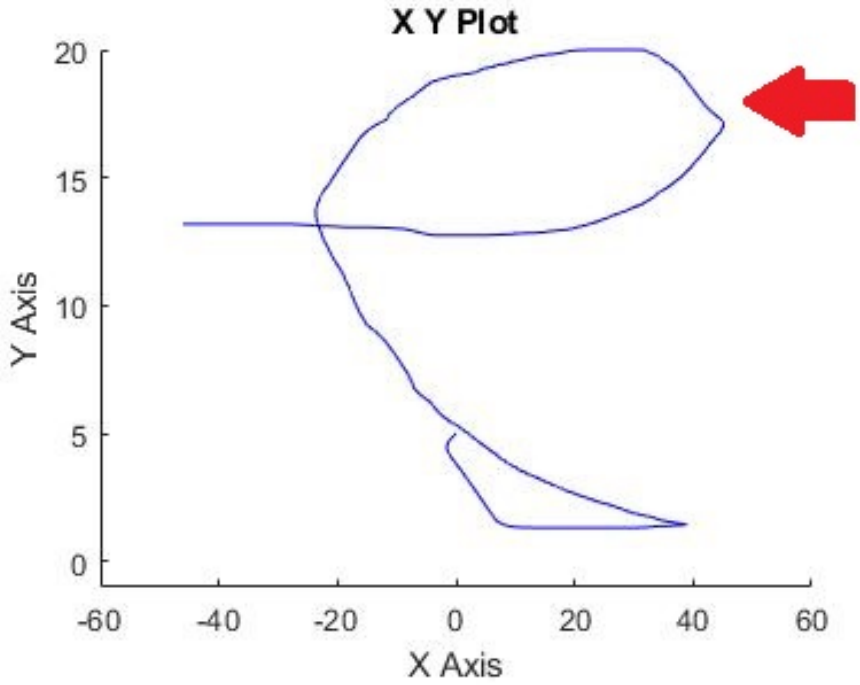


Figure 63: Using 500 points

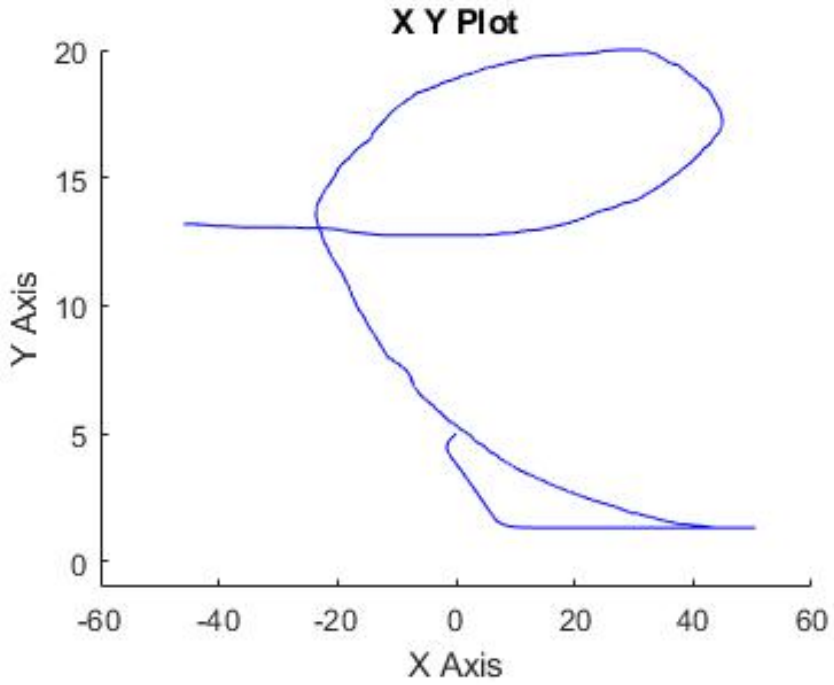


Figure 64: Using 1000 points

where the red arrows are used in order to emphasize the tight turns.

7.3 Swarm flight with draw path

The *swarm* formation has been tested with the drawing path as well. As we can clearly see from the following simulation the results are really good, which provides a confirmation about the effectiveness of the *potential-energy* approach in a swarm formation.

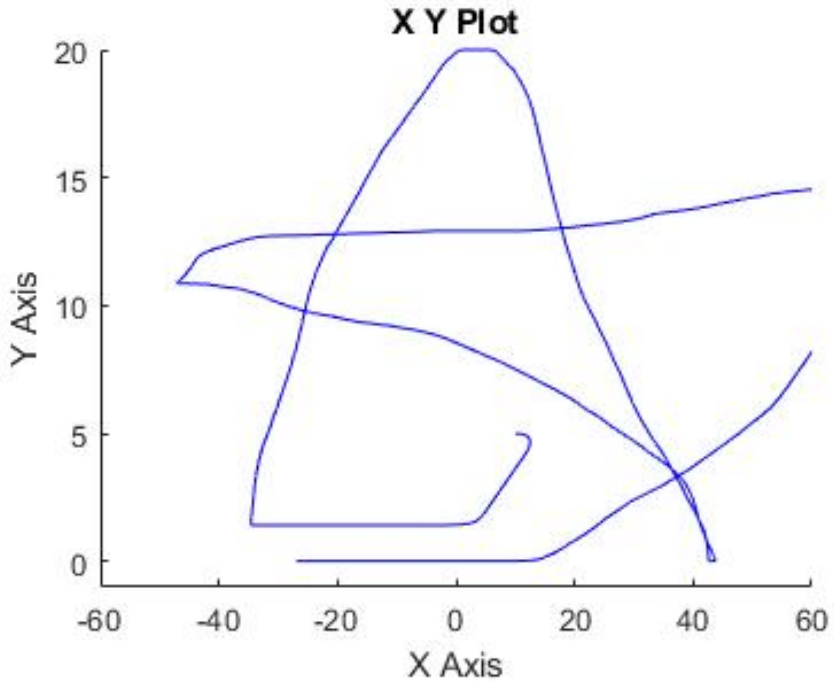


Figure 65: Swarm-1 x-z position draw path

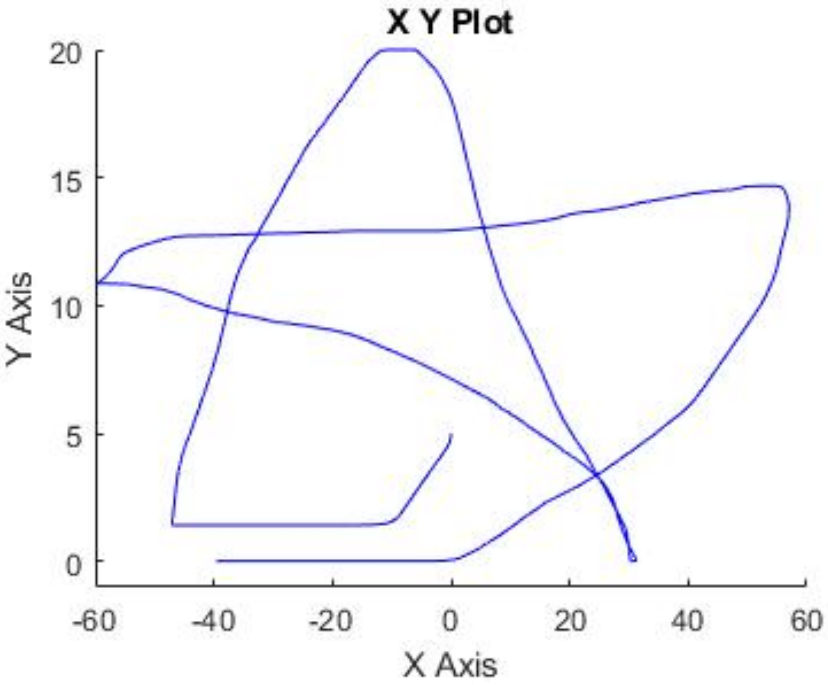


Figure 66: Swarm-2 x-z position draw path

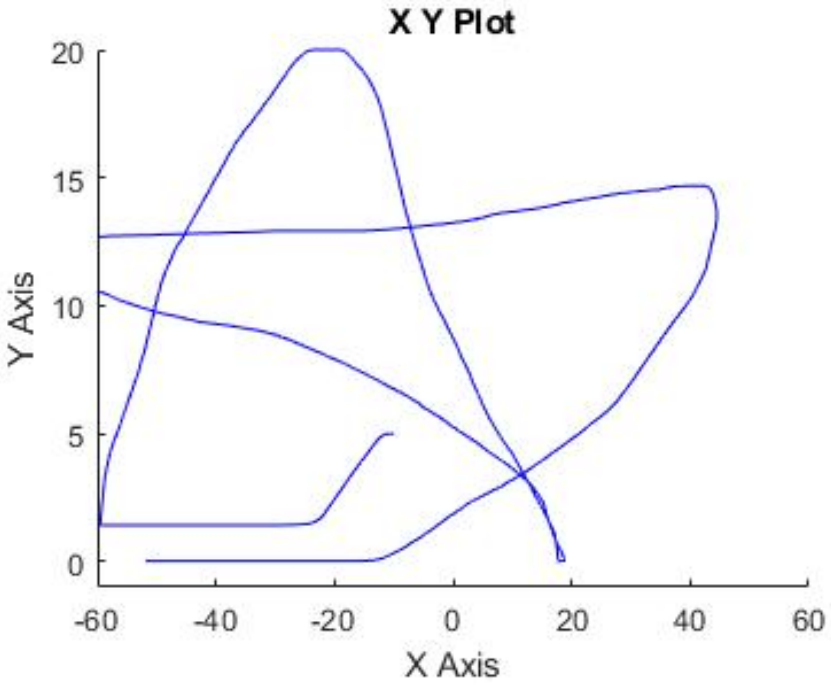


Figure 67: Swarm-3 x-z position draw path

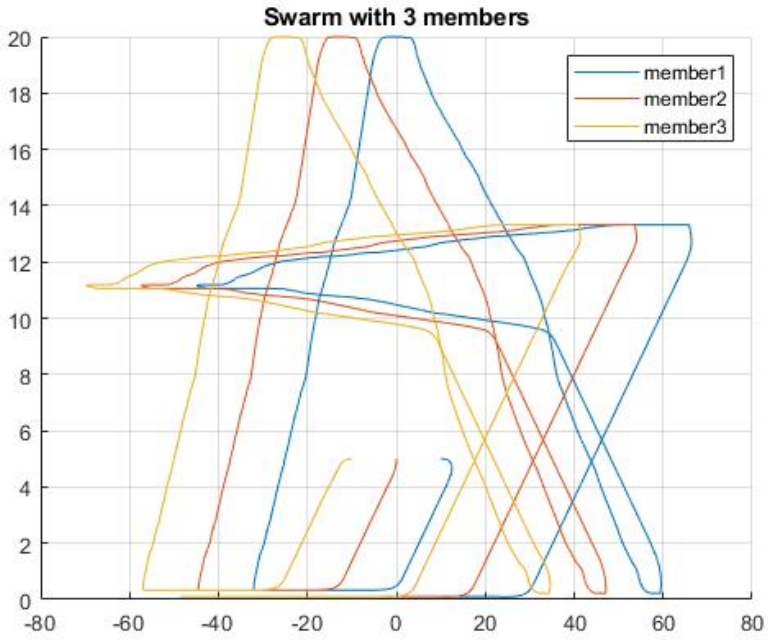


Figure 68: Swarm overlap x-z position draw path

As we can see from the last figure, due to the potential energy effect the swarm has a displacement on the x-axis in order to maintain the formation. The reference for the swarm is the same of the *hover-flight* with the drawing path.

8 Change Formation

In this section will be shown the approach used in order to solve the problem of the change of formation from an initial configuration to a target configuration using the approach proposed in [13].

8.1 Problem description

In [[14],[15]] has been proved that the following problem can be formulated as a differential game for which (approximate) solution can be determined in closed-form.

The problem of the change of formation is split into two parts :

1. First, the problem is solved for N virtual agents satisfying single-integrator dynamic ($\dot{x_i} = u_i$). In this way we are going to plan the trajectory.
2. Second, the trajectory planned in the point 1 is used as reference for the hover flight.

Each virtual agent is associated with the desired target position x_i^*. Let $\tilde{x}_i = x_i - x^*$ the error variable of each virtual agents.
The *obstacle avoidance* region is defined as follow

$$S_j = x \in R^2 : ||x - p_j|| < \rho_j^2 \quad j = 1..m$$

where $p_j \in R^2$ is the vector of the center of the obstacle and ρ_j is the radius. We assume that the norm is the *Eucledian* one.

In the same way we can define the *agent avoidance* region defining the agents as dynamic obstacles

$$D_{ij} = x \in R^2 : ||x - x_j(\bar{t})||^2 \leq (r_i + r_j)^2 \quad j = 1..N \; j \neq i \tag{26}$$

where the r_i defines the region of the safety radius.
In order to solve the problem some assumptions must be defined:

- The initial positions of the agents satisfy the obstacle avoidance condition and the agent avoidance condition.
- The final positions of the agents satisfy the obstacle avoidance condition and the agent avoidance condition.
- The configuration is feasible, which means that the static obstacles do not form a impermeable boundary about target of one or more of the agents.

8.1.1 Solving the problem

In order to solve the first part of the problem we need to introduce the vector $\zeta = [\zeta_1, .., \zeta_N] \in R^{2N}$ where $\zeta_i \in R^2$ and it denotes the dynamic extension the state x_i.
We need to define the so called *collision avoidance* functions defined as follow

$$g_i^s(\tilde{x}) = \sum_{j=1}^{m} \frac{1}{(||\tilde{x}_i + x_i^* - p_j||^2 - \rho_j^2)}$$
$$g_i^d(\tilde{x}) = \sum_{j=1, j\neq i}^{N} \frac{1}{(||(\tilde{x}_i + x_i^*) - (\tilde{x}_j + x_j^*)||^2 - \rho_i^2)}$$

where g_i^s is the function that penalize the trajectory close to the static obstacle, instead g_i^d is the function that penalize the trajectory close to the other virtual agents.

Finally the problem can be solved and the dynamics of the states x_i and the dynamic extension ζ is defined as follow

$$\begin{bmatrix} \dot{x}_i^1 \\ \dot{x}_i^2 \end{bmatrix} = -\tilde{x_i}\left(\sqrt{\alpha_i + \beta_i^s g_i^s(\zeta) + \beta_i^d(\zeta)} + \gamma_i\right) - \sum_{j=1}^{N}(\tilde{x_j} - \zeta_j)$$
$$\dot{\zeta} = -k\sum_{i=1}^{N}\left(\frac{\tilde{x}_i^T\tilde{x}_i}{2\sqrt{\alpha_i + \beta_i^s g_i^s(\zeta) + \beta_i^d(\zeta)}}\left(\beta_i^s\frac{\partial g_i^s(\zeta)^T}{\partial\zeta} + \beta_i^d\frac{\partial g_i^d(\zeta)^T}{\partial\zeta}\right) - (\tilde{x} - \zeta)\right) \tag{27}$$

where $i = 1..N$.

8.2 Simulations

For the sake of the simplicity the following simulation have been done using $N = 2$ and $m = 0$.
The initial conditions have been set as follow

$$x_1 = [15, 5] \quad x_2 = [-15, 5]$$

and the task is to switch the position of the two vehicles. The vector $\zeta_0 = [-15, 5, 5, -15]$.

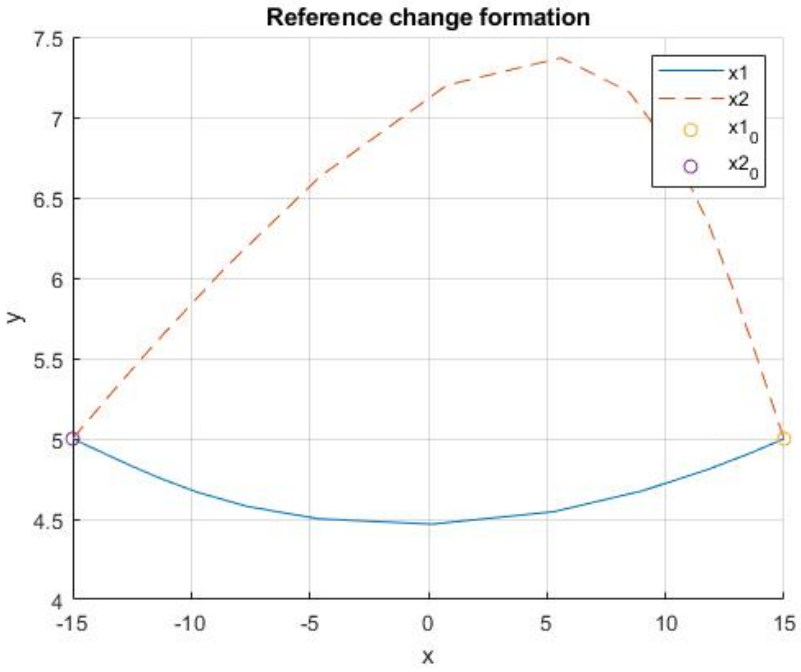


Figure 69: Trajectory planned (x,z)

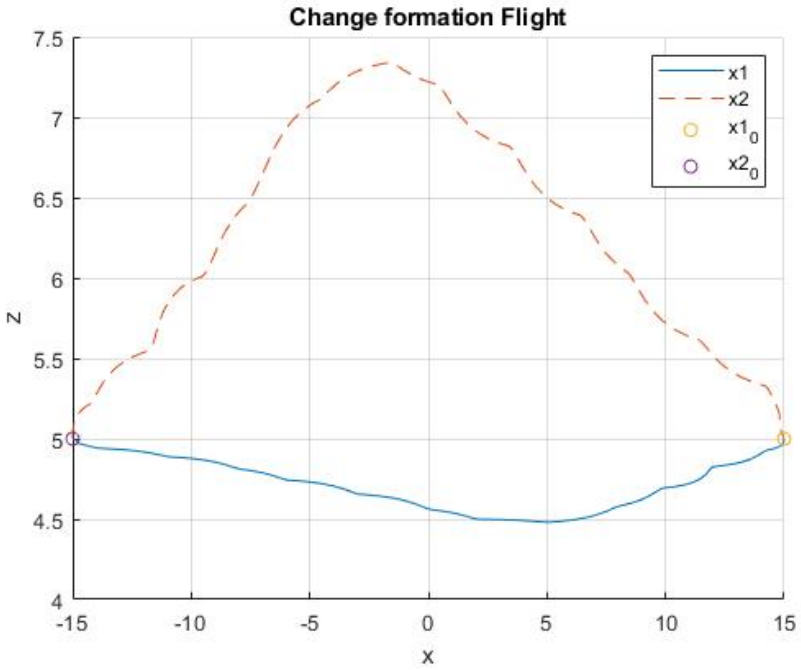


Figure 70: Flight trajectory (x,z)

In this case (as in the draw-path) the trajectory has been discretized in order to create a set of reference points for the *hover-flight*.

The 3D-simulations is provided as well.

9 Future works

Future works can be split into two parts:

- Control of the vehicle.
- Control of the swarm formation.

Future works for the first one include: improving the mathematical model of the system taking into account different shape of the airfoil in order to have a widely range of models, taking into account different kind of disturbances on the aerodynamics force and on the aerodynamics coefficients(C_L and C_m). A more general problem could be the control path of the vehicle and taking into account transverse plane.
Future works for the second one include: simulation with a really large number of members of the swarm in order to study the velocity of the control law to organize them, taking into account a centralized control law and obstacle avoid law. Currently I am working on the last aspect during my internship.
More general work could be done trying to implement the control law on a micro-controller in order to study the complexity of the algorithms and if the results are satisfying an hardware implementation could be taken into account.

10 MATLAB Scripts

10.1 Hover Flight

```
function [thetaDDot,xDDot,zDDot, r, Tt,Td , z1,z2,xTilde,
    zTilde] = HF_dynamics(m,g,l1,Iyy,theta,thetaDot,x,xDot
    ,z,zDot)

zDes = 10;
xDes = -50;

%saturation values
m1 = 0.9;
m2 = 2;
a = 1;
b = 2;
c= 1;

%gain of virutal controllers
kpz = 4;
kdz = 6;


k2 =  1;
k3 = 1;

%function between Td and thetaDot
g2 = -l1/Iyy;

%References
zTilde = z- zDes;
xTilde = x-xDes;


% %Control to stabilize z
r1 = sat(kpz*zTilde,m2) + kdz*zDot;
r = -sat(r1,m1);
Tt = (m*(g-r))/(cos(theta));

% %backstepping
phi1 =  sat(xTilde,a)+ sat(xDot,b);
z1 =phi1 - k2*theta;

% %z1Dot = satDot(x1Tilde,a)*xDot -k2*thetaDot + satDot(
    xDot,b)*(tan(theta)*(r-g));
```

```
39
40 % %Virtual Controller
41 n = -c*z1  - tan(theta)*satDot(xDot,b)*(r-g);
42 z2 = satDot(xTilde,a)*xDot - k2*thetaDot - n;
43
44 z1Dot = z2 - c*z1;
45
46 psi = satDot(xTilde,a)*tan(theta)*(r-g) ...
47     + sec(theta)*sec(theta)*thetaDot*satDot(xDot,b)*(r-g)
       ...
48     +c*(z1Dot)+tan(theta)*satDot(xDot,b)*satDot(zTilde+
       zDot,m1)*(r-g);
49
50 %Controller
51 Td = (psi + z1 + k3*z2)/(k2 * g2);
52
53 %Dynamics
54 xDDot = -Tt * sin(theta)/m;
55 zDDot = -Tt * cos(theta)/m + g;
56 thetaDDot = Td *g2;
57
58
59
60 end
```

10.2 Transition Flight

```
1 function [gamma,xDDot,zDDot,ue, T] = TF(xDot,z,zDot,m,g,d
      ,l,gamma)
2
3 ue = 0;
4 zDes = 10;
5 kpz = 10;
6 kdz = 4;
7
8 k1 = 2;
9 k2 = 2;
10
11
12 T = 10;
13 stepGamma = 0.01;
14
15 drag = -d * xDot^2;
16 lift = l * xDot^2;
17
18
```

```
19
20 %virtual controller
21 vz = kpz*(z-zDes)+kdz*zDot;
22
23 phi = -k1*(z-zDes);
24 e1 = zDot-phi;
25 phiDot = -k1*(e1+ phi);
26
27 %Change angle of thrusts
28 gamma =gamma+stepGamma;
29 gamma = sat(gamma, pi/2);
30
31
32 %Check if lift force > mg
33 if(lift-m*g>0 && abs(gamma)>deg2rad(85) )
34     ue = -(T*cos(gamma)+l*xDot^2-m*g+vz)/(xDot^2);
35
36 elseif(abs(gamma)< deg2rad(85))
37     T = (-(z-zDes) - lift +m*g+phiDot-k2*e1)/(cos(gamma))
          ;
38 end
39
40
41 xDDot = T*sin(gamma) + drag;
42 zDDot = T*cos(gamma) + lift + ue*xDot^2 -m*g;
43
44
45
46 end
```

10.3 Forward Flight

```
1 function [xDot,alphaDot,qDot,u ,zDot , xPosDDot,errorX,
      errorZ ] = FF(m,V,Iyy,g,CL_param,Cm_param,...
2                                          alpha_param, time,x,
                                               xPos,z,alpha,q,x0)
3 Cm = mean(Cm_param);
4
5
6 k1 = 2;
7 k2 = 2;
8
9 Kpx =0.2;
10 Kdx = 0.2;
11
12 if(alpha>71)
```

```
13      alpha = 71;
14 elseif(alpha<-10)
15      alpha = -10;
16 end
17
18 xPosDes = 40;
19 zDes = 25;
20 errorX = xPos-xPosDes;
21 errorZ =z-zDes;
22
23 if(abs(errorZ)>3)
24      xDes = 15;
25      xDesDot = 0;
26 else
27      xDes = 0;
28      xDesDot =0;
29 end
30 alphaRound = round(alpha*2)/2;
31
32 seqIndex = 1:1:length(alpha_param);
33 index = round(interp1(alpha_param,seqIndex,alphaRound));
34 CL = CL_param(index);
35
36 f = (-g/V) * cos((CL*x)/(m*V));
37 f1 = (g/V) * cos((CL*x)/(m*V)) - (CL*alpha)/(m*V);
38 f2 = 0;
39 g2 = Cm/Iyy;
40
41
42
43 df_dx =( (g*CL)/(m*V^2) ) * sin((CL*x)/(m*V));
44 df1_dx = ((-g*CL)/(m*V^2) ) *sin((CL*x)/(m*V));
45 df_ddx = ((g*CL^2)/(m*V^3)) * cos((CL * x)/(m*V));
46 %
47 % %Definition of the xDes
48 % amp = 1;
49 % freq = 2*pi/10;
50 % % xDes = amp * sin(freq*time);
51
52
53 e = x-xDes;
54
55 alphaDes = -f - k1*e + xDesDot; %Virtual : stabilaize the
       dynamics of x
56 alphaDesDot = (-df_dx -k1)*(f+alpha);
57
```

```
58
59 e1 = alpha - alphaDes;
60 qDes = -f1 -x+xDes - k1*e1 + alphaDesDot; %Virtual:
       stabilize the dynamics of alpha
61
62 e2 = q - qDes;
63
64 e1Dot = -e - k1*e1 +e2;
65
66
67
68
69
70 d_alphaDesDot_dx = -df_ddx *f - (df_dx)^2 -df_ddx * alpha
       - k1*df_dx;
71 par_qDes_x = -df1_dx - 1 + d_alphaDesDot_dx;
72 par_qDes_e1 = -k1;
73 qDesDot = par_qDes_x * (f+alpha) + par_qDes_e1 * (e1Dot);
74 u = (-f2 + qDesDot - e1 - k2*e2)/(g2);
75
76 u = sat(u,25);
77
78 %T = -Kpx * ((xPos-xPosDes))-Kdx *V;
79 T = -Kpx * ((z-zDes))-Kdx *V;
80
81 xDot = f + alpha;
82 alphaDot = f1 + q;
83 qDot = f2 + g2 *u;
84 % zDot  = V * sin(deg2rad(x));
85 % xPosDot = V * cos(deg2rad(x));
86
87 zDot = V*sin(deg2rad(x));
88
89 xPosDDot = T * sin(deg2rad(x));
90
91 end
```

10.4 State control

```
1 function [state1,state2,state3,state4,state5] =
      switchState(prevState1,prevState2,prevState3,
      prevState4,prevState5,...
2                         thetaHF,errorXHF,errorZHF,gammaTF,
                              errorZFF,pathFF,gammaTFBack)
3
4 if(prevState1 == 0 && prevState2 == 0 && prevState3==0 &&
```

```
      prevState4==0 && prevState5 == 0)
5       state1 = 1; %HF
6       state2 = 0; %TF-->FF
7       state3 = 0; %FF
8       state4 = 0; %FF-->TF
9       state5 = 0; %TF-->HF
10  else
11      state1 = prevState1; %HF
12      state2 = prevState2; %TF-->FF
13      state3 = prevState3; %FF
14      state4 = prevState4; %FF-->TF
15      state5 = prevState5;
16  end
17  tolSwitch = 0.05;
18
19  if(abs(thetaHF)<tolSwitch && abs(errorXHF)<tolSwitch &&
      abs(errorZHF)<tolSwitch && prevState1 == 1)
20      state1 =0;
21      state2 = 1;
22      state3 = 0;
23      state4 = 0;
24      state5 = 0;
25
26  end
27  if(abs(gammaTF)>=deg2rad(90) && prevState4 == 0 &&
      prevState2 == 1)
28      state1 =0;
29      state2 = 0;
30      state3 = 1;
31      state4 = 0;
32      state5 = 0;
33
34  end
35  if(abs(errorZFF)<5 && prevState3 == 1 && pathFF<1   )
36      state1 =0;
37      state2 = 0;
38      state3 = 0;
39      state4 = 1;
40      state5 = 0;
41  end
42
43  if(abs(gammaTFBack)<deg2rad(0.5) && prevState4 == 1 )
44      state1 =0;
45      state2 = 0;
46      state3 = 0;
47      state4 = 0;
```

```
48      state5 = 1;
49
50 end
51
52
53
54
55
56
57 end
```

10.5 3D Simulations

```
1 function [x,z,theta,gamma,flapAngle] = dataSimulation3D(
      state1,state2,state3,state4,state5,...
2                     thetaHF,xHF,zHF,...
3                     gammaTF,xTF,zTF,...
4                     pathFF,xFF,zFF,flapAngleFF,...
5                     gammaTFBack,xTFBack,zTFBack,...
6                     thetaHFBack,xHFBack,zHFBack)
7
8 x = 0;
9 z = 0;
10 gamma = 0;
11 theta = 0;
12 flapAngle = 0;
13
14 rescaleFF = 1;
15 if(state1 ==1)
16     x = xHF;
17     z = zHF;
18     theta = thetaHF;
19     gamma = 0;
20     flapAngle = 0;
21 end
22 if(state2 == 1)
23     x = xTF;
24     z = zTF;
25     theta = 0;
26     gamma = gammaTF;
27     flapAngle = 0;
28
29 end
30 if(state3 == 1)
31      x = rescaleFF*xFF;
32      z = zFF;
```

```
33      theta = pathFF;
34      gamma = pi/2;
35      flapAngle = flapAngleFF;
36
37 end
38 if(state4 == 1)
39     x = xTFBack;
40     z = zTFBack;
41     theta = 0;
42     gamma = gammaTFBack;
43     flapAngle = 0;
44 end
45 if(state5 ==1)
46     x = xHFBack;
47     z = zHFBack;
48     theta = thetaHFBack;
49     gamma = 0;
50     flapAngle = 0;
51 end
52
53 end
```

10.6 Swarm Potential energy

```
1 function [x1Dot,x2Dot,x3Dot] = potentialFunction(x1,x2,x3
      )
2
3 a = 0.03;
4 b = 1;
5 c = 30;
6
7 x1Dot = -(x1-x2)*(a-b*c*exp(-((x1-x2)^2)/c))-(x1-x3)*(a-b
      *c*exp(-((x1-x3)^2)/c));
8 x2Dot = -(x2-x1)*(a-b*c*exp(-((x2-x1)^2)/c))-(x2-x3)*(a-b
      *c*exp(-((x2-x3)^2)/c));
9 x3Dot = -(x3-x2)*(a-b*c*exp(-((x3-x2)^2)/c))-(x3-x1)*(a-b
      *c*exp(-((x3-x1)^2)/c));
10
11
12
13 end
```

10.7 GeLoopPos.m

```
1 function [x,z] = getLoopPos()
2 clc;
```

```
clear all;
close all;
max_xMap = 60;
max_zMap = 20;
global KEY_IS_PRESSED
KEY_IS_PRESSED = 0;
x =[];
z =[];
gcf
set(gcf, 'KeyPressFcn', @myKeyPressFcn)
while ~KEY_IS_PRESSED
        drawnow
    pos =  get(0,'PointerLocation');
    x = [x; pos(1)];
    z = [z; pos(2)];
end
x = x - (max(x) - min(x))/2;
x = (x/(max(x)))*max_xMap;
z = (z/(max(z)))*max_zMap;
plot(x,z);
title('Drawed path');
xlabel('x');
ylabel('z');
axis([-max_xMap-10,max_xMap+10,-5,max_zMap+10]);

disp('loop ended')
```

```
function myKeyPressFcn(hObject, event)
global KEY_IS_PRESSED
KEY_IS_PRESSED = 1;
disp('key is pressed')
```

10.8 changeFormation.m

```
function [x1Dot,x2Dot,y1Dot,y2Dot,psi1Dot,psi2Dot,psi3Dot
    ,psi4Dot] = fcn(x1,x2,y1,y2,psi1,psi2,psi3,psi4)


x1Des = -15;
x2Des = 5;
y1Des = 15;
```

```
 7 y2Des = 5;
 8
 9 p1 = 4;
10 p2 = -1;
11
12 rho = 3;
13 r = 1;
14 k = 1;
15 beta1 = 1;
16 beta2 = 1;
17 alpha1 = 1;
18 alpha2 = 1;
19 gamma1 = 1;
20 gamma2 = 1;
21
22
23 % psi_1 = [psi1;psi2];
24 % psi_2 = [psi3;psi4];
25
26 x1Tilde = x1-x1Des;
27 x2Tilde = x2-x2Des;
28 y1Tilde = y1-y1Des;
29 y2Tilde = y2-y2Des;
30
31
32 dg1s_dpsi11 = (-2*(psi1+x1Des-p1))/((psi1+x1Des-p1)^2+(
       psi2+x2Des-p2)^2-rho^2)^2;
33 dg1s_dpsi12 = (-2*(psi2+x2Des-p2))/((psi1+x1Des-p1)^2+(
       psi2+x2Des-p2)^2-rho^2)^2;
34
35 dg2s_dpsi21 = (-2*(psi3+y1Des-p1))/((psi3+y1Des-p1)^2 + (
       psi4+y2Des-p2)^2 - rho^2)^2;
36 dg2s_dpsi22 = (-2*(psi4+y2Des-p2))/((psi3+y1Des-p1)^2 + (
       psi4+y2Des-p2)^2 - rho^2)^2;
37
38 dg1s_dpsi = [dg1s_dpsi11;dg1s_dpsi12;0;0];
39 dg2s_dpsi = [0;0;dg2s_dpsi21;dg2s_dpsi22];
40
41
42 dg1d_dpsi11 = (-2*(psi1+x1Des-psi3-y1Des))/((psi1+x1Des-
       psi3-y1Des)^2 + (psi2+x2Des-psi4-y2Des)^2 -r^2)^2;
43 dg1d_dpsi12 = (-2*(psi2+x2Des-psi4-y2Des))/((psi1+x1Des-
       psi3-y1Des)^2 + (psi2+x2Des-psi4-y2Des)^2 -r^2)^2;
44
45
46 dg2d_dpsi11 = (2*(psi3+y1Des-psi1-x1Des))/((psi3+y1Des-
```

```
    psi1-x1Des)^2 + (psi4+y2Des-psi2-x2Des)^2 -r^2)^2;
47 dg2d_dpsi12 = (2*((psi4+y2Des-psi2-x2Des)))/((psi3+y1Des-
    psi1-x1Des)^2 + (psi4+y2Des-psi2-x2Des)^2 -r^2)^2;
48
49
50
51 dg1d_dpsi = [dg1d_dpsi11;dg1d_dpsi12;-dg1d_dpsi11;-
    dg1d_dpsi12];
52 dg2d_dpsi = [dg2d_dpsi11;dg2d_dpsi12;-dg2d_dpsi11;-
    dg2d_dpsi12];
53 %
54
55
56 const_x1Dot = sqrt(alpha1 + beta1*g_s(psi1,psi2,x1Des,
    x2Des,p1,p2,rho) + ...
57                 beta1* g_d(psi1,psi2,psi3,psi4,x1Des,
                       x2Des,y1Des,y2Des,r) ) + gamma1;
58
59 const_x2Dot = const_x1Dot;
60
61 const_y1Dot = sqrt(alpha2 + beta2*g_s(psi3,psi4,y1Des,
    y2Des,p1,p2,rho) + ...
62                 beta2* g_d(psi3,psi4,psi1,psi2,y1Des,
                       y2Des,x1Des,x2Des,r) ) + gamma2;
63
64 const_y2Dot = const_y1Dot;
65
66 sum_x1Dot = (x1Tilde-psi1);%+ y1Tilde -psi3;
67 sum_x2Dot = (x2Tilde-psi2); %+ y2Tilde- psi4;
68 sum_y1Dot = (y1Tilde-psi3);
69 sum_y2Dot = (y2Tilde-psi4);
70
71 x1Dot = -x1Tilde*(const_x1Dot)-sum_x1Dot;
72 x2Dot = -x2Tilde*(const_x2Dot)-sum_x2Dot;
73 y1Dot = -y1Tilde*(const_y1Dot)-sum_y1Dot;
74 y2Dot = -y2Tilde*(const_y2Dot)-sum_y2Dot;
75
76
77 % x1Dot = 0;
78 % x2Dot = 0;
79 % y1Dot = 0;
80 % y2Dot = 0;
81 % %
82 %
83 xTilde = [x1Tilde;x2Tilde];
84 yTilde = [y1Tilde;y2Tilde];
```

```
85
86 %PSI_1_DOT
87 const1_psi1Dot = (xTilde' *xTilde)/...
88                  (2*sqrt(alpha1 + beta1*g_s(psi1,psi2,
                        x1Des,x2Des,p1,p2,rho) + ...
89                  beta1* g_d(psi1,psi2,psi3,psi4,x1Des,
                        x2Des,y1Des,y2Des,r)));
90
91 const2_psi1Dot = (yTilde' * yTilde)/...
92                     (2*sqrt(alpha2 + beta2*g_s(psi3,psi4,
                           y1Des,y2Des,p1,p2,rho) + ...
93                  beta2* g_d(psi3,psi4,psi1,psi2,y1Des,
                        y2Des,x1Des,x2Des,r) ));
94
95 sum1_psi1Dot = const1_psi1Dot * (beta1*dg1s_dpsi(1)+beta1
       *dg1d_dpsi(1))- (x1Tilde-psi1);
96 sum2_psi1Dot = const2_psi1Dot*(beta2*dg2s_dpsi(1) + beta2
       *dg2d_dpsi(1))-(x1Tilde-psi1);
97
98 psi1Dot = -k*(sum1_psi1Dot + sum2_psi1Dot);
99
100
101 %PSI_2_DOT
102 const1_psi2Dot = (xTilde'*xTilde)/...
103                  (2*sqrt(alpha1 + beta1*g_s(psi1,psi2,
                        x1Des,x2Des,p1,p2,rho) + ...
104                  beta1* g_d(psi1,psi2,psi3,psi4,x1Des,
                        x2Des,y1Des,y2Des,r)));
105
106 const2_psi2Dot =(yTilde'*yTilde)/...
107                     (2*sqrt(alpha2 + beta2*g_s(psi3,psi4,
                           y1Des,y2Des,p1,p2,rho) + ...
108                  beta2* g_d(psi3,psi4,psi1,psi2,y1Des,
                        y2Des,x1Des,x2Des,r) ));
109
110
111 sum1_psi2Dot = const1_psi2Dot *(beta1*dg1s_dpsi(2)+beta1*
       dg1d_dpsi(2))- (x2Tilde-psi2);
112 sum2_psi2Dot = const2_psi2Dot *(beta2*dg2s_dpsi(2)+beta2*
       dg2d_dpsi(2))- (x2Tilde-psi2);
113
114
115 psi2Dot = -k*(sum1_psi2Dot + sum2_psi2Dot);
116
117
118 %PSI_3_DOT
```

```
119 const1_psi3Dot = (xTilde'*xTilde)/...
120                  (2*sqrt(alpha1 + beta1*g_s(psi3,psi4,
                         y1Des,y2Des,p1,p2,rho) + ...
121                  beta1* g_d(psi3,psi4,psi1,psi2,y1Des,
                         y2Des,x1Des,x2Des,r)));
122
123 const2_psi3Dot=  (yTilde'*yTilde)/...
124                    (2*sqrt(alpha2 + beta2*g_s(psi3,psi4,
                           y1Des,y2Des,p1,p2,rho) + ...
125                  beta2* g_d(psi3,psi4,psi1,psi2,y1Des,
                         y2Des,x1Des,x2Des,r) ));
126
127 sum1_psi3Dot = const1_psi3Dot *(beta1*dg1s_dpsi(3)+beta1*
       dg1d_dpsi(3))- (y1Tilde-psi3);
128 sum2_psi3Dot = const2_psi3Dot *(beta2*dg2s_dpsi(3)+beta2*
       dg2d_dpsi(3))- (y1Tilde-psi3);
129
130
131 psi3Dot = -k*(sum1_psi3Dot + sum2_psi3Dot);
132
133
134 %PSI_4_DOT
135
136 const1_psi4Dot = (xTilde'*xTilde)/...
137                  (2*sqrt(alpha1 + beta1*g_s(psi3,psi4,
                         y1Des,y2Des,p1,p2,rho) + ...
138                  beta1* g_d(psi3,psi4,psi1,psi2,y1Des,
                         y2Des,x1Des,x2Des,r)));
139
140 const2_psi4Dot=  (yTilde'*yTilde)/...
141                    (2*sqrt(alpha2 + beta2*g_s(psi3,psi4,
                           y1Des,y2Des,p1,p2,rho) + ...
142                  beta2* g_d(psi3,psi4,psi1,psi2,y1Des,
                         y2Des,x1Des,x2Des,r) ));
143
144 sum1_psi4Dot = const1_psi4Dot *(beta1*dg1s_dpsi(4)+beta1*
       dg1d_dpsi(4))- (y2Tilde-psi4);
145 sum2_psi4Dot = const2_psi4Dot *(beta2*dg2s_dpsi(4)+beta2*
       dg2d_dpsi(4))- (y2Tilde-psi4);
146
147
148 psi4Dot = -k*(sum1_psi4Dot + sum2_psi4Dot);
149
150 %
151 %
152 % psi1Dot = 0;
```

```
% psi2Dot = 0;
% psi3Dot = 0;
% psi4Dot = 0;
```

```
end
```

10.9 Save Plots

```
close all;
clc;
vars = who('-regexp', 'Plot');
path = 'C:\Users\ferra\Desktop\Projects\Matlab\Thesis\
    Figures';

num_plot = length(vars);

for i=1:1:num_plot
    name = vars(i);
    name = name{1};
    value = eval(name)
    split_name = (strsplit(name,'_'));

    system = split_name{2};
    var_controlled = split_name{length(split_name)-1};
    var_x_axis = split_name{length(split_name)};
    str_title = strcat(system,'-',var_controlled);

    close all
    fg=figure('Name',str_title,'units','normalized','
        outerposition',[0 0 1 1]);
    plot(value(:,1),value(:,2))
    title(str_title);
    xlabel(var_x_axis);
    ylabel(var_controlled);
    grid on;

    name_file = strcat(str_title,'.jpg');
    saveas(fg ,fullfile(path , name_file));
    disp('fig saved');
```

```
30
31 end
```

References

[1] Wikipedia, "Vtol," 2018.

[2] Wikipedia, "Pitching moment," 2018.

[3] W. Wisnoe, M. A. Zurriati, M. S. Firdaus, R. N. Fazira, R. E. M. Nasir, and W. Kuntjoro, "Experimental investigation of center elevator deflection on aerodynamics of uitm's baseline-i blended wing body (bwb) unmanned aerial vehicle (uav)," *2010 International Conference on Science and Social Research (CSSR 2010)*, pp. 108–112, 2010.

[4] Wikipedia, "Lift coefficient," 2018.

[5] G. R. Flores Colunga, J. A. Escareño, R. Lozano, and S. Salazar, "Four Tilting Rotor Convertible MAV: Modeling and Real-Time Hover Flight Control," *Journal of Intelligent and Robotic Systems*, vol. 65, pp. 457–471, Jan. 2012.

[6] "Global stabilization of the pvtol: Real-time application to a mini-aircraft," May 2017.

[7] C. Aguilar-Ibanez, H. Sossa-Azuela, and M. S. Suarez-Castanon, "Pvtol control: A backstepping approach," *2015 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*, 2015.

[8] A. R. Teel, "Global stabilization and restricted tracking for multiple integrators with bounded controls," *Systems Control Letters*, vol. 18, no. 3, p. 165–171, 1992.

[9] Teel, "A nonlinear small gain theorem for the analysis of control systems with saturation."

[10] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *SIGGRAPH Comput. Graph.*, vol. 21, pp. 25–34, Aug. 1987.

[11] V. Gazi and K. M. Passino, "A class of attractions/repulsion functions for stable swarm aggregations," 2004.

[12] K. Han, J. Lee, and Y. Kim, "Unmanned aerial vehicle swarm control using potential functions and sliding mode control," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 222, no. 6, pp. 721–730, 2008.

[13] T. Mylvaganam and M. Sassano, "Autonomous collision avoidance for wheeled mobile robots using a differential game approach," *Eur. J. Control*, vol. 40, pp. 53–61, 2017.

[14] T. Mylvaganam, M. Sassano, and A. Astolfi, "A constructive differential game approach to collision avoidance in multi-agent systems," *2014 American Control Conference*, 2014.

[15] T. Mylvaganam, M. Sassano, and A. Astolfi, "A differential game approach to multi-agent collision avoidance," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, p. 4229–4235, 2017.