ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MODEL PREDICTIVE CONTROL
MINI-PROJECT

Group AF
Frédéric Pili (324476), Pierre Vinet (312918), Julie Bohning(296023)

12 janvier 2024

Contents

1 Introduction

In order to get familiar with model predictive control, we are asked in this project design several MPC controllers for a small scale prototype of a rocket where the engine is replaced by high-performance drone racing propellers.

In this section, we will recall the provided nonlinear model.

We are using a 12 state description of the system :

$\boldsymbol{x} = [\boldsymbol{\omega}^T\ \boldsymbol{\varphi}^T\ \boldsymbol{v}^T\ \boldsymbol{p}^T]^T\ ,\ \ [\boldsymbol{x}] = [rad/s\ \ rad\ \ m/s\ \ m]$

where $\boldsymbol{\omega} = [\omega_x\ \ \omega_y\ \ \omega_z]^T$ are the angular velocities about the body axes, $\boldsymbol{\varphi} = [\alpha\ \ \beta\ \ \gamma]^T$ repesent the attitude of the body frame with respect to the world frame, $\boldsymbol{v} = [v_x\ \ v_y\ \ v_z]^T$ and $\boldsymbol{p} = [x\ \ y\ \ z]^T$ are the velocity and position expressed in the world frame.

The input vector of the model is $\boldsymbol{u} = [\delta_1\ \ \delta_2\ \ P_{avg}\ \ P_{diff}]^T$, $\ [\boldsymbol{u}] = [rad\ \ rad\ \ \%\ \ \%]^T$ where δ_1 and δ_2 are the deflection angles of servo 1 and servo 2 respectively, P_{avg} is the average throttle and P_{diff} is the throttle difference between the motors.

With the constraints $\delta_1, \delta_2 \in [-15°, +15°](= 0.26rad)$, $\quad P_{avg} \in [20\%, 80\%]$, $\quad P_{diff} \in [-20\%, +20\%]$

The dynamic of the model gives us the nonlinear equation $\dot{x} = f(x, u)$ which have been implemented in the Matlab Rocket class.

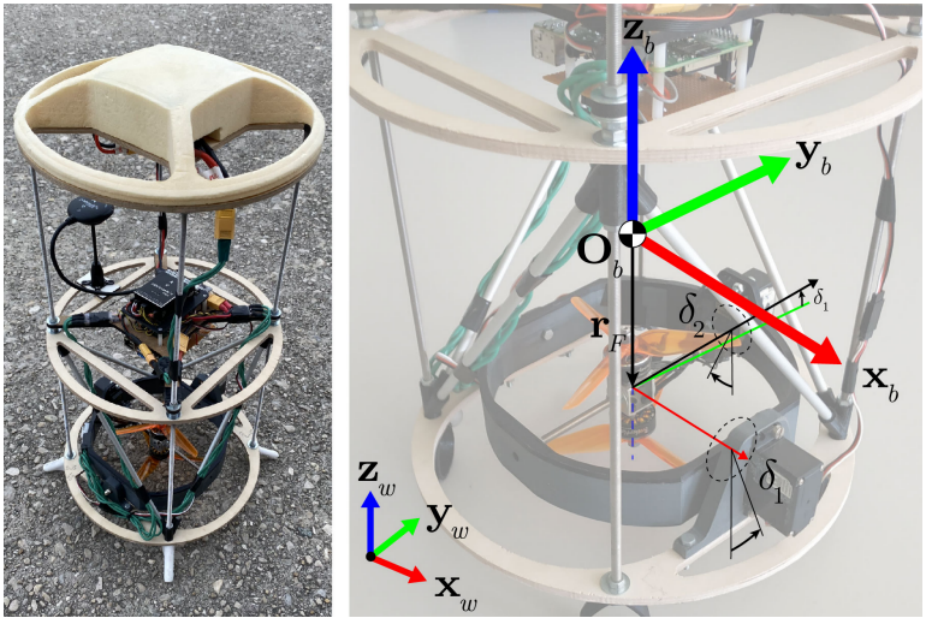

FIGURE 1.1
Rocket prototype

2 Linearization

2.1 DELIVERABLE

In this part we are required to trim and linearize the system. This means that we search for an equilibrium point (x_s, u_s) such that $f(x_s, u_s) = 0$. We then linearize the system around this point $\dot{x} \approx A(x - x_s) + B(u - u_s)$. After being trimmed and linearized, the system can be divided into four subsystems :

$\dot{x}_x = A_x x_x + B_x u_x, \quad y_x = C_x x_x + D_x u_x$

$\dot{x}_y = A_y x_y + B_y u_y, \quad y_y = C_y x_y + D_y u_y$

$\dot{x}_z = A_z x_z + B_z u_z, \quad y_z = C_z x_z + D_z u_z$

$\dot{x}_{roll} = A_{roll} x_{roll} + B_{roll} u_{roll}, \quad y_{roll} = C_{roll} x_{roll} + D_{roll} u_{roll}$

From an intuitive perspective, this division into four subsystems is possible because each of the subsystems can be controlled independently from the others. To move in the x and y dimensions, we only need to change δ_2 and δ_1 respectively, to move in the z dimension, we only need to change P_{avg} and to roll the system, we only need to change P_{diff}

We can see by analyzing the linearized system B matrix (input matrix) that each input only affects one dimension. For instance, δ_1 affects ω_x and v_y which both acts on the y dimension, δ_2 affects ω_y and v_x which both acts on the x dimension, P_{avg} only affects v_z and P_{diff} only affects ω_z

B =

	d1	d2	Pavg	Pdiff
wx	-55.68	0	0	0
wy	0	-55.68	0	0
wz	0	0	0	-0.104
alpha	0	0	0	0
beta	0	0	0	0
gamma	0	0	0	0
vx	0	9.81	0	0
vy	-9.81	0	0	0
vz	0	0	0.1731	0
x	0	0	0	0
y	0	0	0	0
z	0	0	0	0

FIGURE 2.1
Linearized system B matrix

3 Design MPC Controllers for Each Sub-System

3.1 DELIVERABLE

In this section we are asked to design an MPC controller for each subsystems.

To be able to stay in the linearized domain, we need to add additional constraints on the angles α and β. We also add a constraint on the average throttle to limit the downward acceleration.

$|\alpha| \leq 10° = 0.1745\, rad, \quad |\beta| \leq 10° = 0.1745\, rad, \quad 50\% \leq Pavg \leq 80\%$

We also need to take in account the trim points x_s and u_s for our constraint design. $x_s = \mathbf{0}$ and $u_s = \begin{bmatrix} 0 \\ 0 \\ 56.6667 \\ 0 \end{bmatrix}$

These result in the constraint matrices :

$Fx \leq f, \quad Mu \leq m$

x subsystem, ω_y, β, v_x, x as states, δ_2 as input :

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} x_x \leq \begin{bmatrix} 0.1745 \\ 0.1745 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ -1 \end{bmatrix} u_x \leq \begin{bmatrix} 0.26 \\ 0.26 \end{bmatrix}$$

y subsystem, ω_x, α, v_y, y as states, δ_1 as input :

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} x_y \leq \begin{bmatrix} 0.1745 \\ 0.1745 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ -1 \end{bmatrix} u_y \leq \begin{bmatrix} 0.26 \\ 0.26 \end{bmatrix}$$

z subsystem, v_z, z as states, P_{avg} as input :

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} x_z \leq 0\,, \quad \begin{bmatrix} 1 \\ -1 \end{bmatrix} u_z \leq \begin{bmatrix} 80 - u_{s,z} \\ -50 + u_{s,z} \end{bmatrix}$$

roll subsystem, ω_z, γ as states, P_{diff} as input :

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} x_z \leq 0\,, \quad \begin{bmatrix} 1 \\ -1 \end{bmatrix} u_z \leq \begin{bmatrix} 20 \\ 20 \end{bmatrix}$$

The design of Q for each subsystem is done by setting it to the identity matrix and then increasing the cost corresponding to states saturating to the maximal or minimal constraints. The design of R is done by setting it to 1 and then increasing the cost of inputs when we observe oscillations. We want to avoid oscillations around 0 at all cost for the z position to avoid collision with the ground (assuming that 0 is the ground). We also need to minimise the settle time to keep it under 7 seconds. By doing this method we find good performances for :

$$Q_{x,y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 150 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Q_z = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, Q_{roll} = \begin{bmatrix} 10 & 0 \\ 0 & 20 \end{bmatrix}$$

and

$R_{x,y} = 10$, $R_z = 1$, $R_{roll} = 1$

Since we are not given any computation time constraint, we set the horizon length H to a high value (10 seconds) to have a better controlled input.

The terminal invariant sets are computed as follows :

We first compute the discrete time LQR controller gain for the unconstrained system K. Then we compute the first polytope X_f such that $Fx \leq f, \quad MKu \leq m$ and $A_{cl} = A + BK$ such that $x_{i+1} = A_{cl}x_i$. We then compute the polytope corresponding to $A_{cl}x$ for $x \in X_f$ and intersect it with the first polytope. We then repeat the two last steps until the two intersected polytope are equal. This method allows us to compute a polytope such that for every $x \in X_f, \quad A_{cl}x \in X_f$.

Here are the terminal sets and the performances for the x and y dimensions :

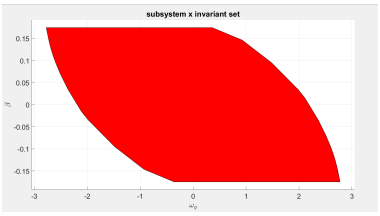

FIGURE 3.1
x terminal set, ω_y-β

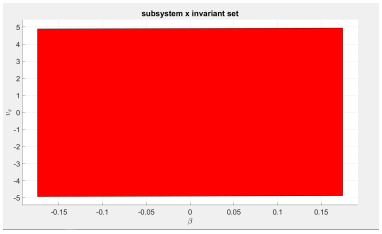

FIGURE 3.2
x terminal set, β-v_x

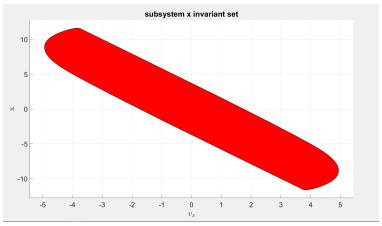

FIGURE 3.3
x terminal set, v_x-x

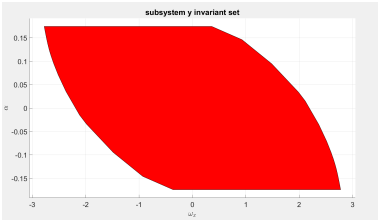

FIGURE 3.4
y terminal set, ω_x-α

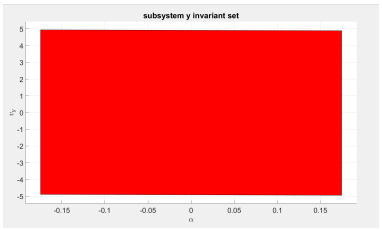

FIGURE 3.5
y terminal set, α-v_y

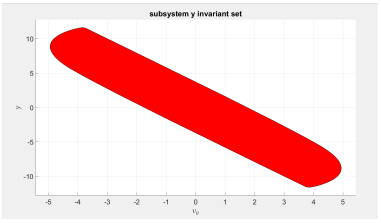

FIGURE 3.6
y terminal set, v_y-y

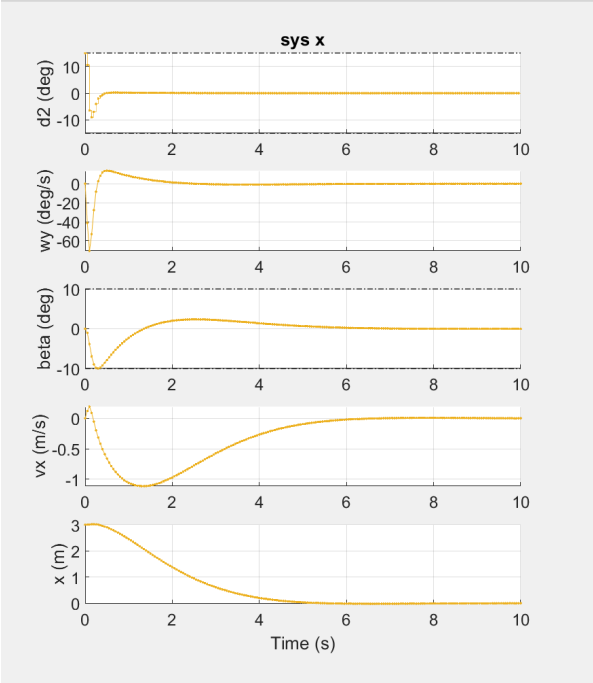

FIGURE 3.7
x performances open-loop

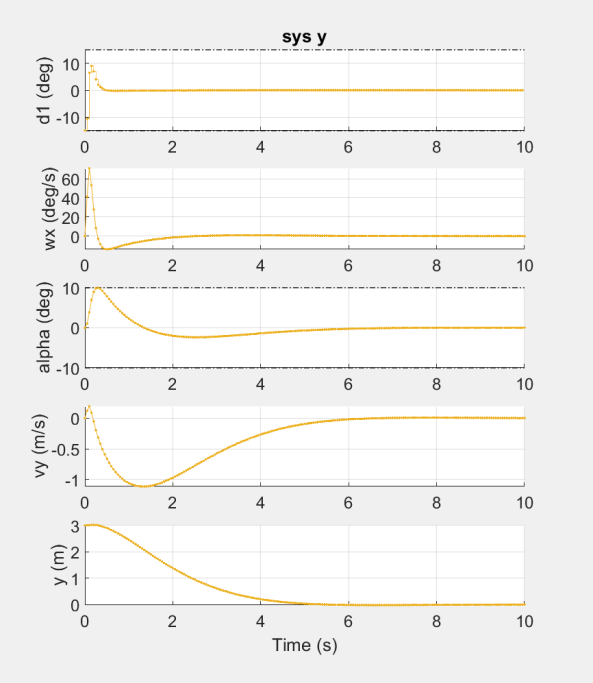

FIGURE 3.8
y performances open-loop

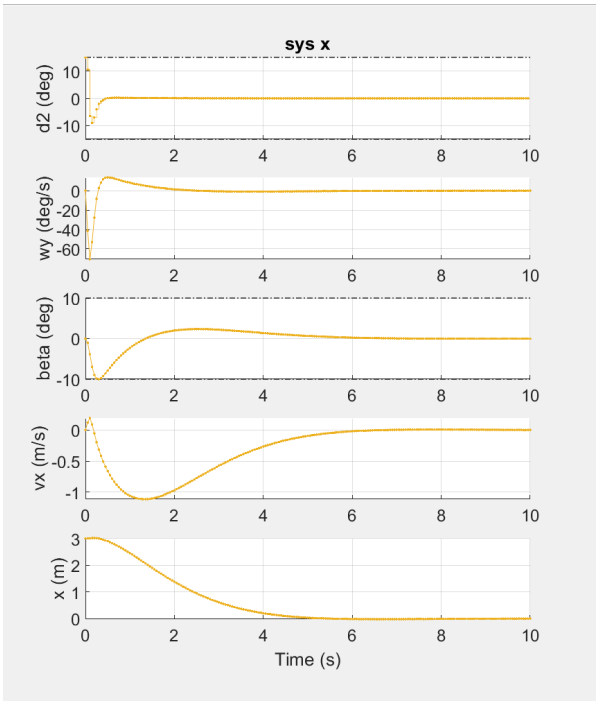

FIGURE 3.9
x performances closed-loop

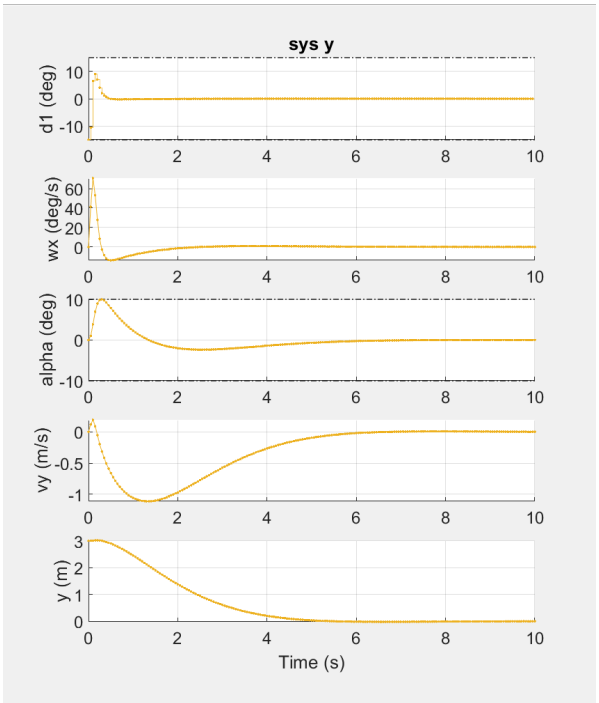

FIGURE 3.10
y performances closed-loop

Here are the terminal sets and the performances for the z dimension :

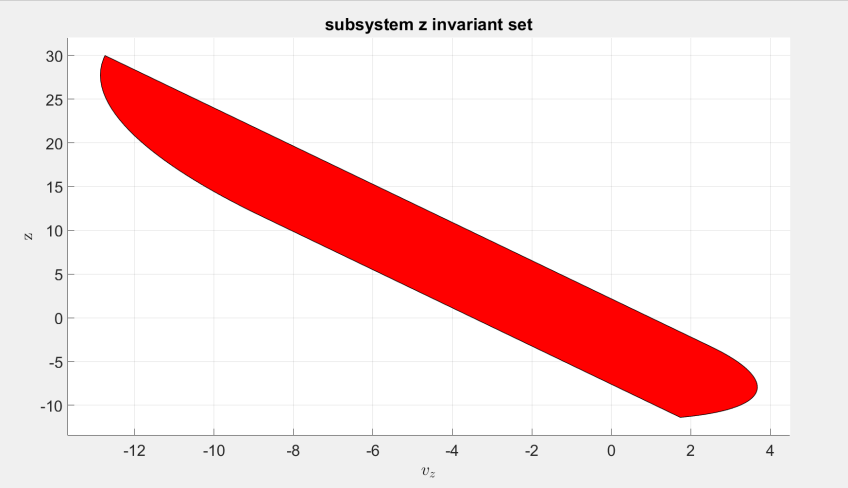

FIGURE 3.11
z terminal set

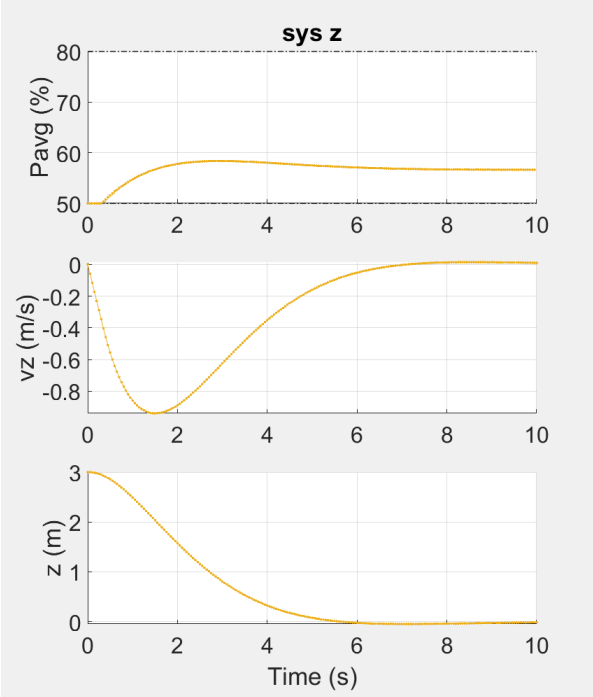

FIGURE 3.12
z performances open-loop

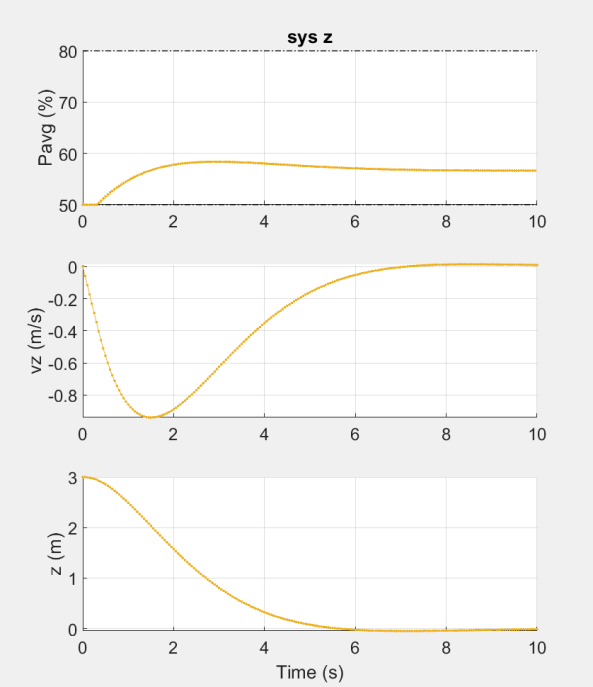

FIGURE 3.13
z performances closed-loop

Here are the terminal sets and the performances for the roll dimension :

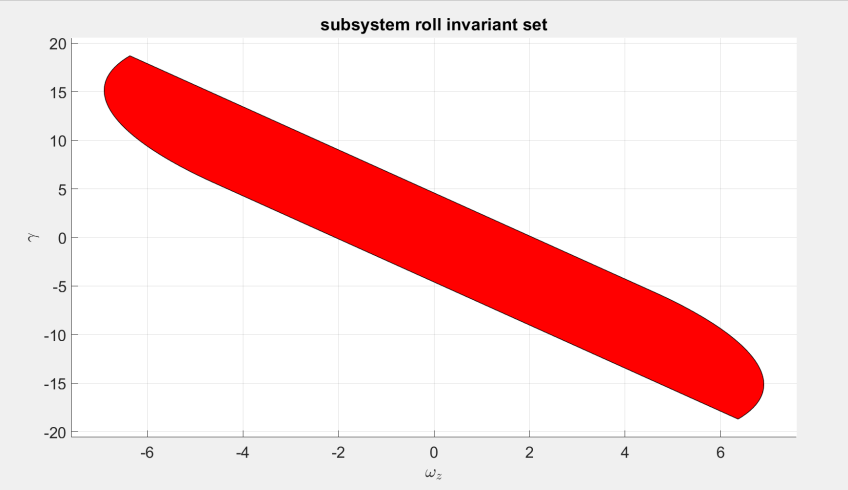

FIGURE 3.14
roll terminal set

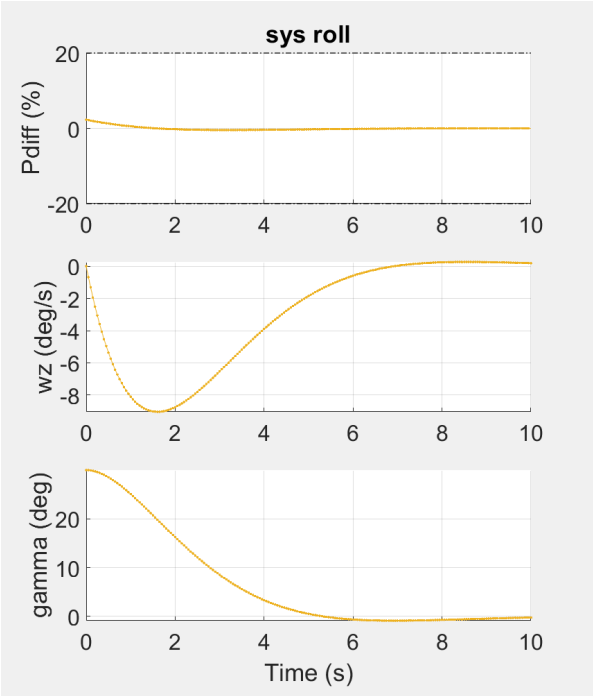

FIGURE 3.15
roll performances open-loop

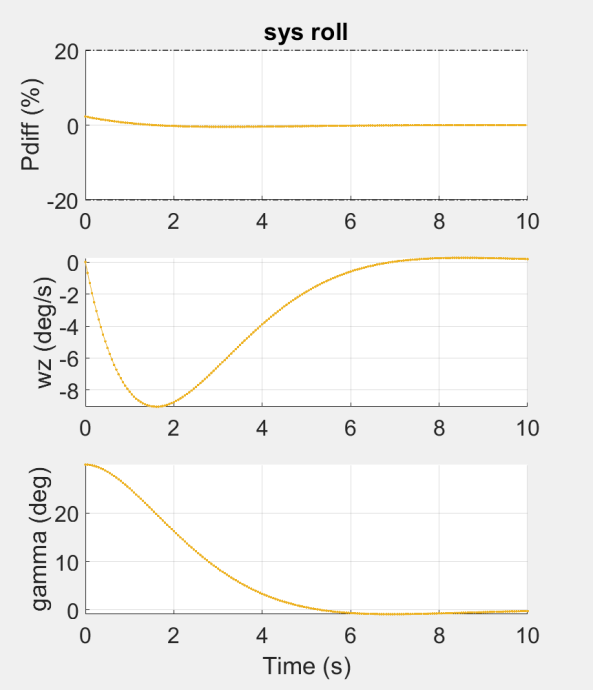

FIGURE 3.16
roll performances closed-loop

The closed-loop and open-loop simulations appear identical because we select a high value for the horizon length.

3.2 DELIVERABLE

In this section we have to extend our controllers so that they can track constant references.

The reasoning is the same as in the first section, but instead of minimizing over x and u, we minimize over $\Delta x = x - x_s$ and $\Delta u = u - u_s$

We choose the target state and input xs by resolving this problem :

$min\ u_s^T R_s u_s$

such that

$x_s = Ax_s + Bu_s, \quad Cx_s = ref, \quad Fx_s \leq f, \quad Mu_s \leq m$

Here we choose $R_s = 1$

We also dropped the requirement of invariant, so we don't need to compute the terminal set.

We simulated the system starting at the origin and tracking a reference to -4 meters (for x, y and z) and to 35° for roll. It seems that adding a reference or dropping the requirement of invariant added some overshoot on the z position and on γ, so we increased the cost of these states in the Q_z and Q_{roll} matrices to have better performances. The overshoot remains even with extremely large costs, so we decided to stay within a reasonable range of costs while keeping the overshoot small and within 7 seconds of simulation.

$$Q_z = \begin{bmatrix} 1 & 0 \\ 0 & 250 \end{bmatrix}, \quad Q_{roll} = \begin{bmatrix} 1 & 0 \\ 0 & 250 \end{bmatrix}$$

Here are the performances for the x and y dimensions :

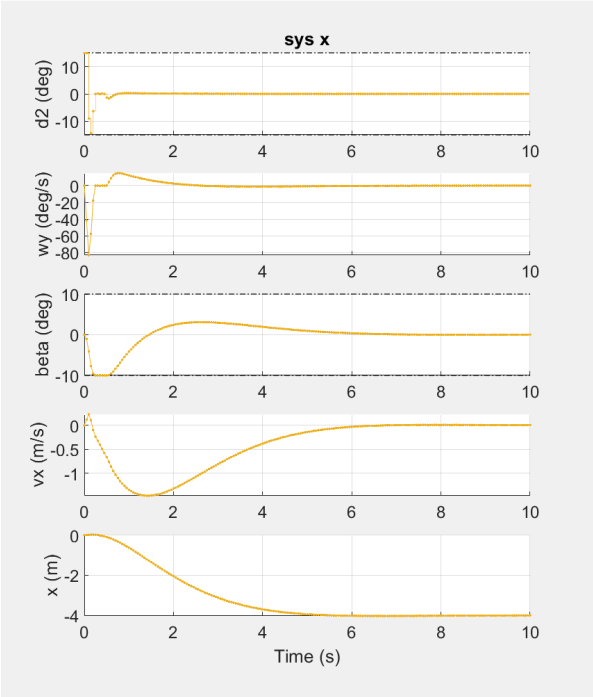

FIGURE 3.17
x performances with tracking open-loop

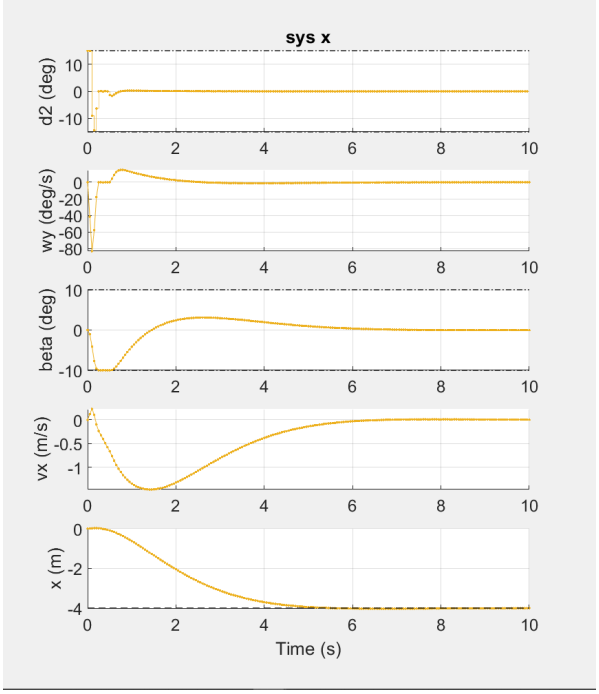

FIGURE 3.18
x performances with tracking closed-loop

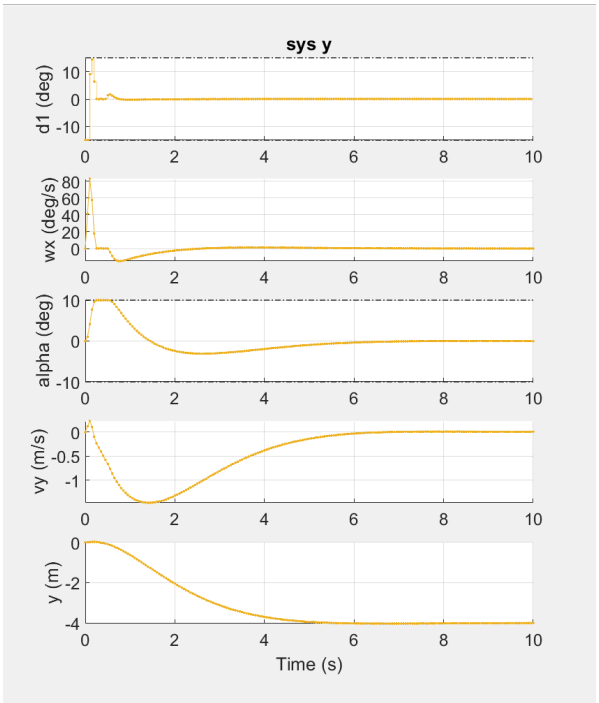

FIGURE 3.19
y performances with tracking open-loop

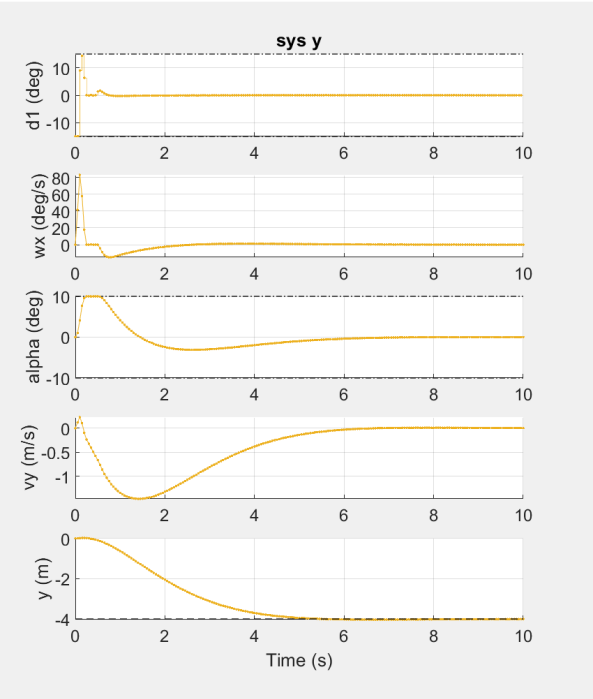

FIGURE 3.20
y performances with tracking closed-loop

Here are the performances for the z dimension :

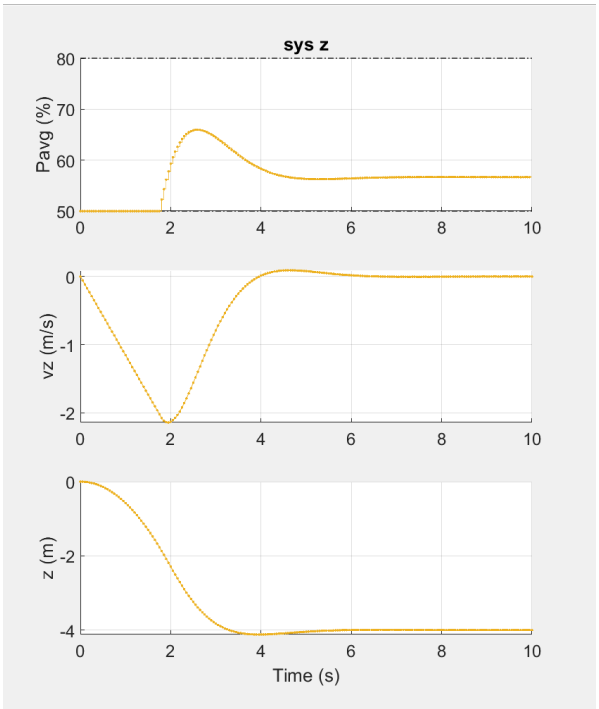

FIGURE 3.21
z performances with tracking open-loop

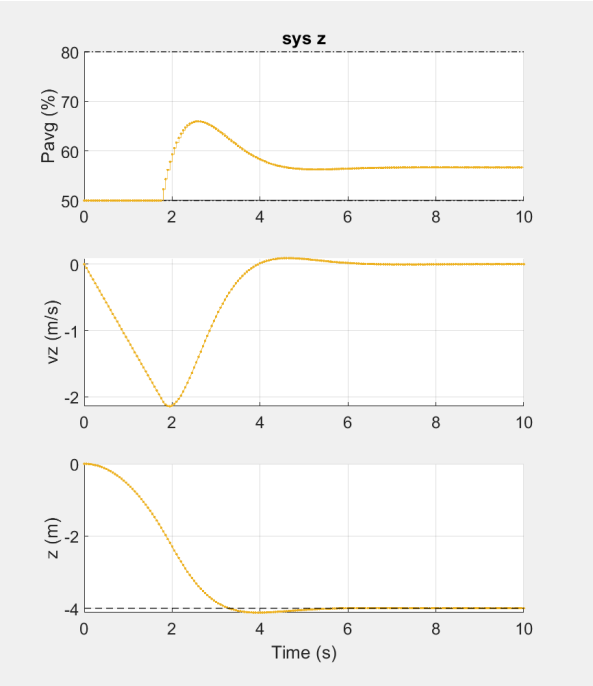

FIGURE 3.22
z performances with tracking closed-loop

Here are the performances for the roll dimension :

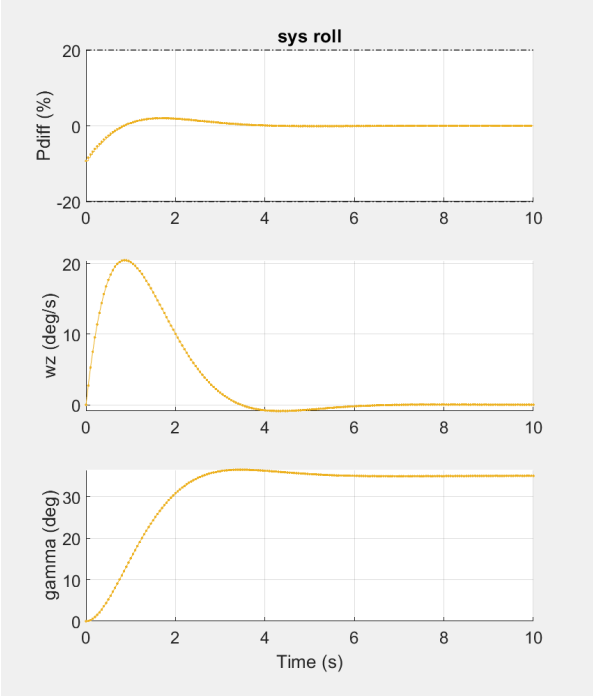

FIGURE 3.23
roll performances with tracking open-loop

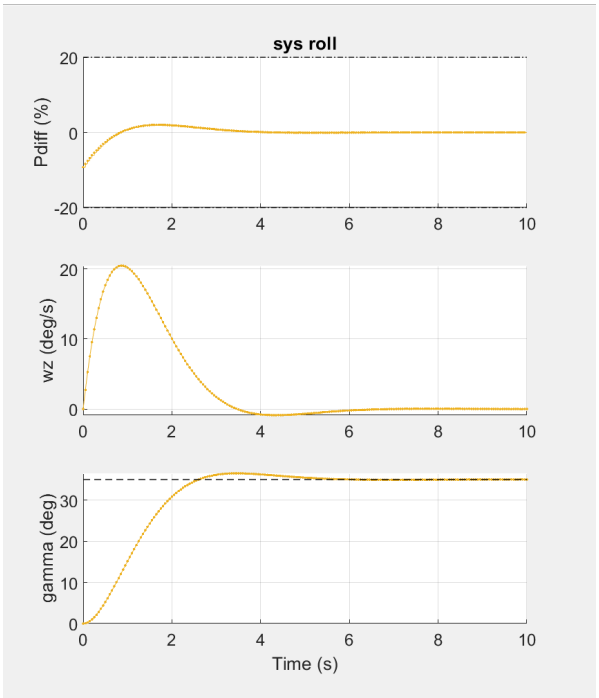

FIGURE 3.24
roll performances with tracking closed-loop

As for the previous section, the open-loop and closed-loop simulations look very similar due to a high value for the horizon length H.

4 Simulation with Nonlinear Rocket

4.1 DELIVERABLE

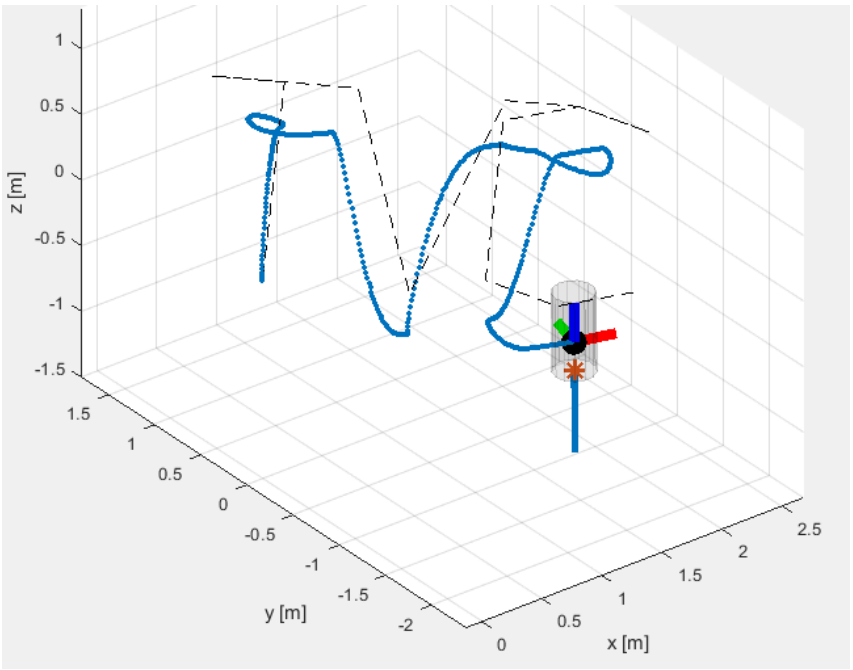

FIGURE 4.1
Nonlinear simulation of the Rocket path (blue) vs Reference (dash)

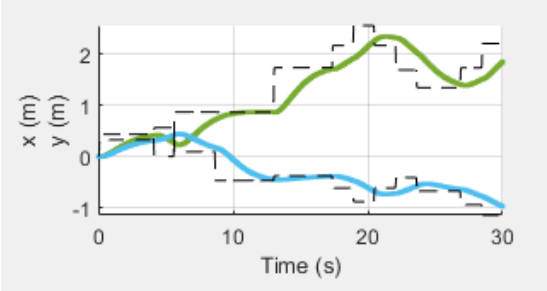

FIGURE 4.2
Rocket path vs reference (dashed line) in the x (green) and y (blue) dimension

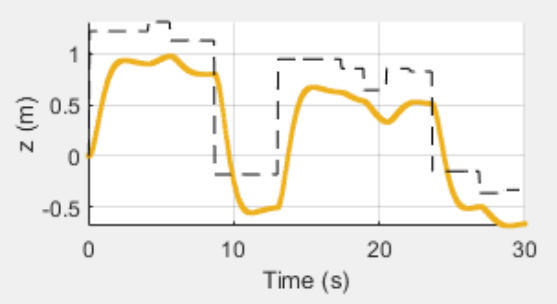

FIGURE 4.3
Rocket path vs reference (dashed line) in the z (orange) dimension

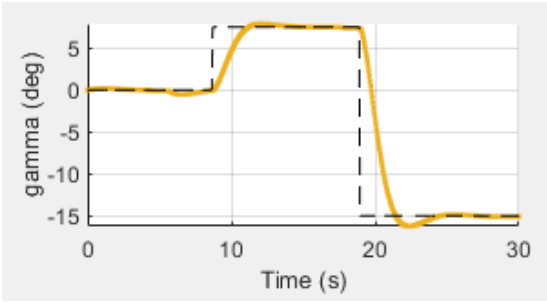

FIGURE 4.4
Rocket angle (orange) vs reference (dashed line)

The 3D plot reveals the rocket's trajectory over time, with the solid blue line representing the actual path and the dashed black line indicating the reference path. The rocket starts with good alignment in the x and y dimensions but lacks aggressiveness in tracking the reference. It then deviates trying to reference the letter "C". It deviates also while getting back from a path, forming loops that diverge from the intended route. For the z dimension, the rocket consistently undershoots the reference altitude, indicating ineffective vertical control. The rocket's roll angle is well tracked with a little delay in response to a step change and slightly overshoots the reference (dashed line).

The primary cause of the ineffective tracking is the model mismatch between the linear controllers and the non-linear simulation. Since the linear MPC uses a simplified prediction model that assumes linear system behavior, it cannot accurately predict the non-linear responses of the actual system. This discrepancy leads to control inputs that are not optimal for the non-linear dynamics, resulting in the observed deviation from the reference path and altitude.

To mitigate the noted undershoot in the z-axis, a manual adjustment was made to the steady-state control variable Pavg. Specifically, Pavg was increased from f 56.6666% to 61%. This intervention aimed to correct for the persistent shortfall in altitude as visible in the following figure.

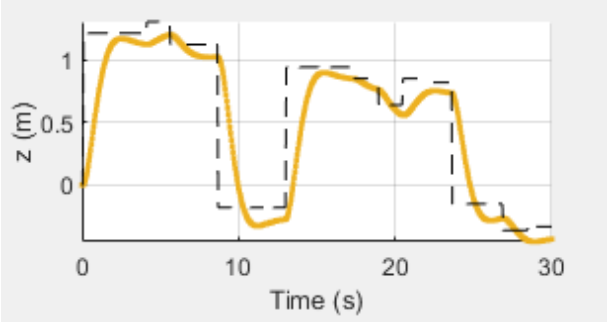

FIGURE 4.5
Rocket path vs reference (dashed line) in the z (orange) dimension increasing Pavg to 61%

Additionally, in order to more accurately trace the reference, an alteration was made to the controller's aggressiveness by adjusting the weight matrix Q within the MPC controllers. The diagonal entries of Q were increased, to enhance the responsiveness of the controller, allowing for more rapid and assertive corrections when discrepancies are detected.

5 Offset-Free Tracking

5.1 DELIVERABLE

To achieve offset-free tracking, the control system was augmented with an additional state dedicated to estimating the disturbance. In our case, it is a Leuenberger estimator is defined by poles at 0.1, 0.5, and 0.8. In our model, the disturbance is treated as a constant yet unknown variable, influencing the dynamics in the z-direction. Figure 5.1 and figure 5.2 present a comparative analysis of the system's response with and without the offset-free tracking implementation.

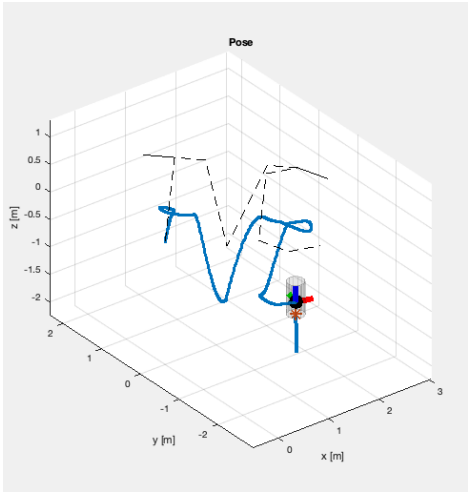

FIGURE 5.1
Simulation of a heavier Rocket's path (blue) vs Reference (dash)

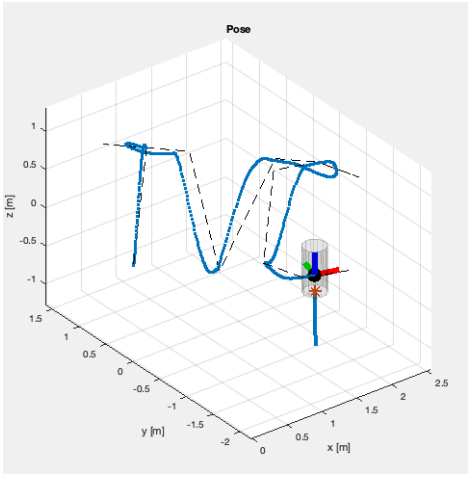

FIGURE 5.2
Simulation of a heavier Rocket's path (blue) vs Reference (dash) with offset-free tracking

In order to realize better the correction, you can see in figure 5.5 and figure 5.5 another comparative analysis of the system's response with and without the offset-free tracking implementation. This time, the simulation was carried out over an 8-second duration, commencing from an initial state of $x_0 = [\text{zeros}(1,9), 1, 0, 3]'$ and a reference point $\text{ref} = [1.2, 0, 3, 0]'$.

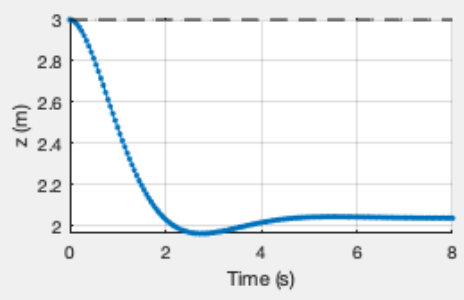

FIGURE 5.3
z performance without offset-free tracking

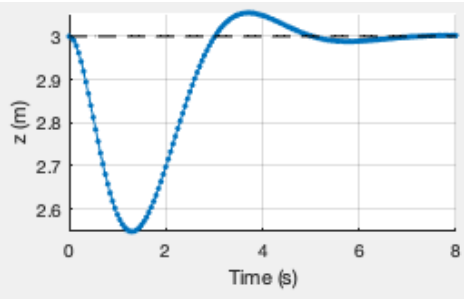

FIGURE 5.4
z performance with offset-free tracking

5.2 DELIVERABLE

When factoring in the decreasing fuel mass at a constant rate, we observe a notable impact on the average thrust, denoted as P_{avg}. As the rocket progressively loses weight due to fuel consumption, a corresponding linear adjustment in thrust is necessitated to meet the desired flight objectives. This phenomenon is clearly illustrated in the two graphs below, which contrast the behavior of P_{avg} in scenarios of constant mass versus diminishing mass.

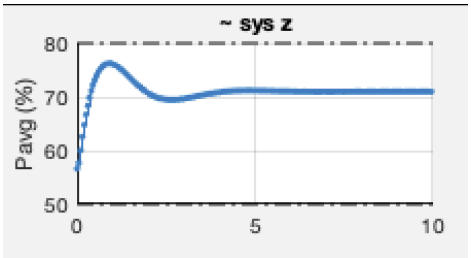

FIGURE 5.5
P_{avg} through time with constant fuel mass

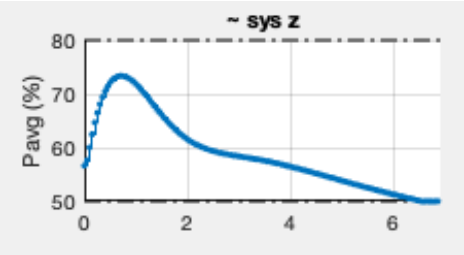

FIGURE 5.6
P_{avg} through time with decreasing fuel mass

In the simulation's initial phase, a tracking offset in height is observed, primarily due to the estimator's inability to quickly adapt to the rocket's rapidly changing mass as fuel is consumed. To address this issue and achieve offset-free tracking in the scenario of decreasing mass, the estimator could be modified by integrating mass dynamics directly into its model. This integration would enable the estimator to account for the decreasing mass in its state predictions, leading to more accurate and responsive tracking performance in the case of mass variation.

Addressing your question about the distinct behaviors observed in the rocket trajectory simulation with time-varying mass : Initially, as fuel is consumed, the rocket becomes lighter and over-responds to control inputs. Mid-simulation, the system's increased responsiveness requires stability adjustments. Towards the end, a significant challenge arises due to fuel depletion, as indicated by an error message. This error, related to thrust-dependent mass dynamics, underscores the complexities of managing a system with diminishing fuel, preventing an extended simulation and highlighting the intricacies of controlling such a dynamic system.

6 Nonlinear MPC

6.1 DELIVERABLE

In this section we have to design a non-linear MPC. For this we design a cost function based on the non-linear function :

$J(x) = min \sum_{i=1}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T Q_f x_N$

such that :

$x_{i+1} = f_{discrete}(x_i, u_i), \quad Fx_i \leq f, \quad Mx_i \leq m$

where Qf is the terminal cost computed with the linearized and discretized function. F, M, f and m the matrices describing the constraints :

$|\beta| \leq 75°, \quad |\delta_1| \leq 15°, \quad |\delta_2| \leq 15°, \quad 50\% \leq P_{avg} \leq 80\%, \quad |P_{diff}| \leq 20\%$

and $f_{discrete}$ is the discretized non-linear function with RK4 :

$x_{k+1} = x_k + h(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k3}{3} + \frac{k_4}{6})$

where

$k_1 = f(t_k, x_k), \;\; k_2 = f(t_k{+}h/2, x_k{+}h/2k_1), \;\; k_3 = f(t_k{+}h/2, x_k{+}h/2k_2), \;\; k_4 = f(t_k{+}h, x_k{+}hk_3)$

By iterative testing we defined the cost matrices Q and R as diagonal matrices :

$Q = diag(40, 40, 40, 2, 2, 1200, 25, 25, 40, 200, 200, 200)$

and

$R = diag(0.01, 0.01, 0.0001, 0.001)$

to have good open loop performances.

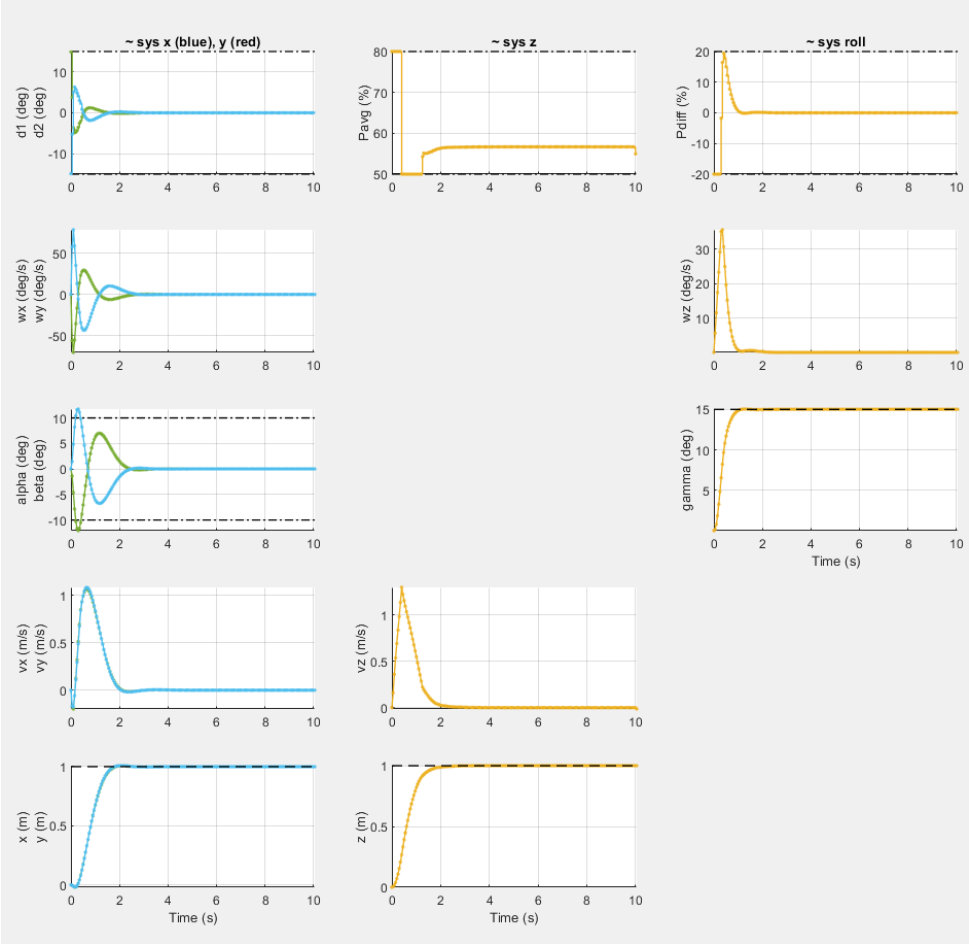

FIGURE 6.1
NMPC Open-loop performances

We have set the costs corresponding to position x, y, z and γ to be high cost to allow the controller to track efficiently those states.

We have set the costs corresponding to P_{avg} and P_{diff} to be very low (We have to assume that the propellers can change their throttles very fast). The other costs have been set by trial and error to avoid oscillation on the inputs and to try to minimize the overshoots.

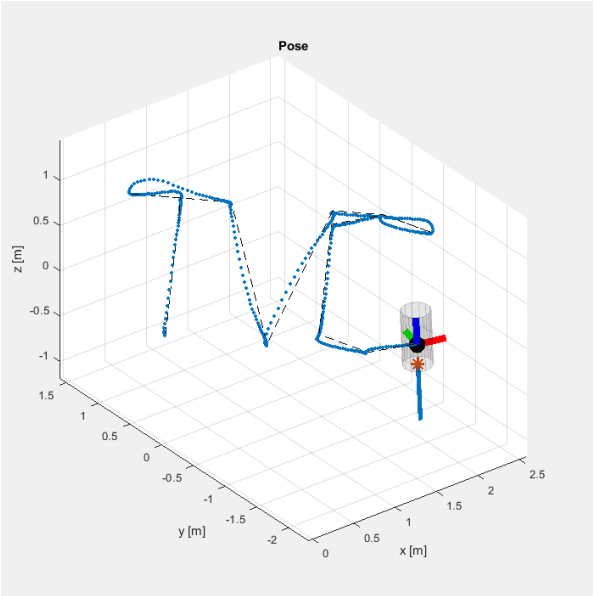

FIGURE 6.2
3D visualization, max roll = $15°$

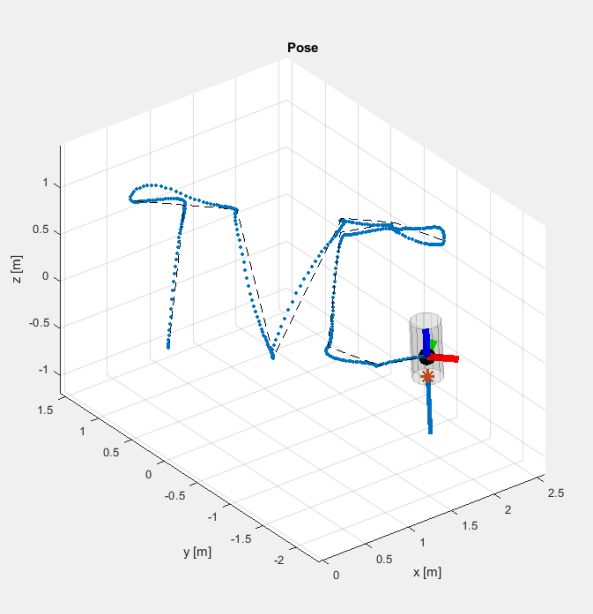

FIGURE 6.3
3D visualization, max roll = $50°$

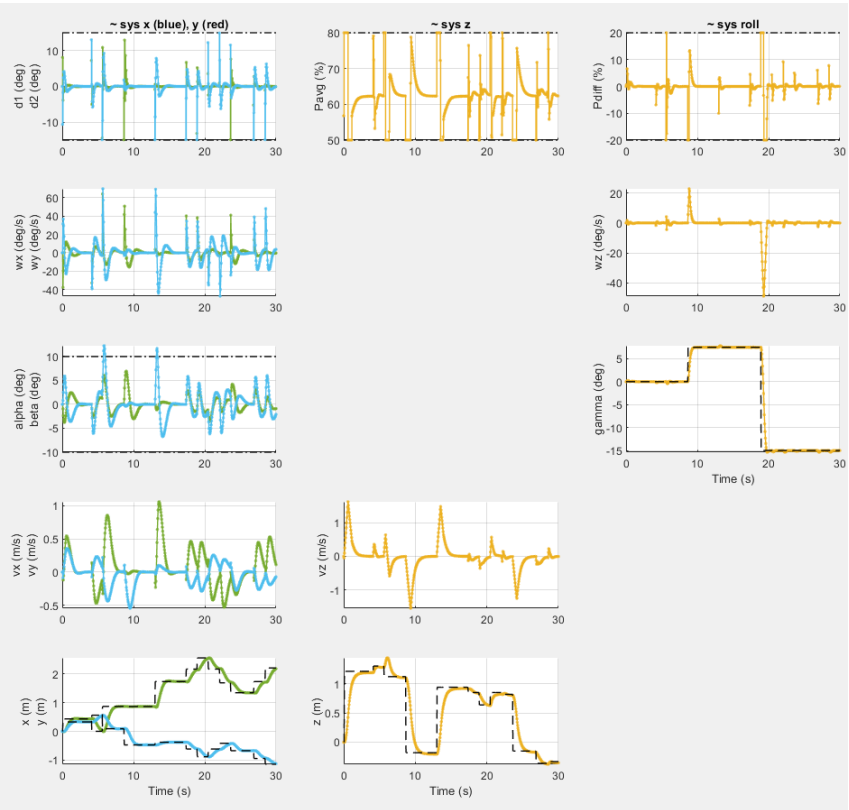

FIGURE 6.4
NMPC Controller performances, max roll = $15°$

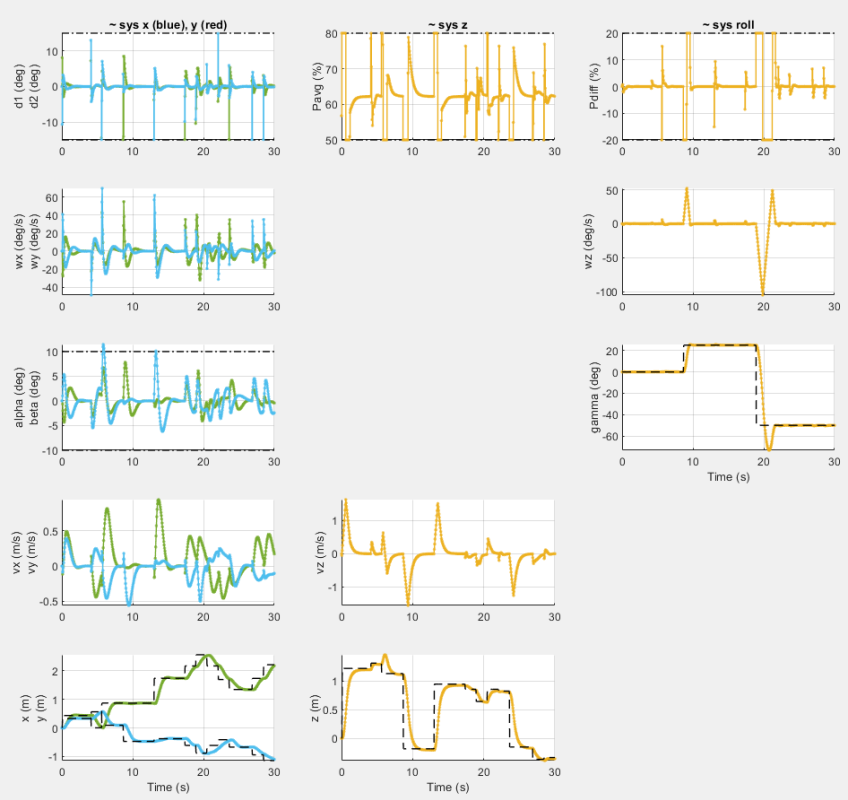

FIGURE 6.5
NMPC Controller performances, max roll = $50°$

We can see that our performances on γ for $maxroll = 15°$ is really good while our performances on x, y, and z are not as good as we would expect. The angles are not very sharp and we can observe some overshoot on z. We have set the horizon $H = 0.1$ seconds to save computation time but the results may be better with a higher horizon or with a better discretization method. The performances for $maxroll = 50°$ are the same on x, y and z as we would expect but we can observe some overshoot on γ.

Comparing the results we have for the linear and the non linear controller, we can say that the non linear controller can achieve better performances but have a much higher computation time, the state are correlated, meaning that changing the weight corresponding to one state can drastically change the performances on the other states, making it hard to tune the parameters, particularly avoid oscillations on δ_1 and δ_2. The linear controller cannot achieve such performances because of the distance to the trim point but is faster to compute, we can set the horizon to a much higher value and still be faster to compute. Since each of the dimensions have its own controller, one state does not affect the others as much as in then on linear case.

6.2 DELIVERABLE

In this section we are asked to design a simple delay compensation algorithm to take in account the computation time of the non linear controller in a real life situation.

First, by simulation the performances tracking a constant reference, we can see that the controller is very weak to delay. It take one sampling period of delay to see a difference on the performances and only 3 sampling periods of delay to make the closed loop system unstable.

The problem here is that the measure is taken at time k but because of delay caused by computation time, the input u_k is applied at time k+n sampling periods.

We then design a simple delay compensation algorithm by simulation the system for n sampling periods before computing the input u_{k+n} and then applying this input at the actual time k+n. We simulate the system using a simple Euler integration scheme :

$$X_{next} = X + h * f(X, U)$$

With a delay compensation algorithm, we can achieve good performances until 5 sampling periods of delay on the closed loop system. We start to notice a drop on performances with a delay of 6 sampling periods with a big overshoot on x and y and some oscillations on the z curve. Finally, the closed loop system becomes completely unstable starting from 10 sampling periods of delay. We may be able to achieve better performances by increasing the horizon and by using a better approximation than Euler to simulate the n-steps.

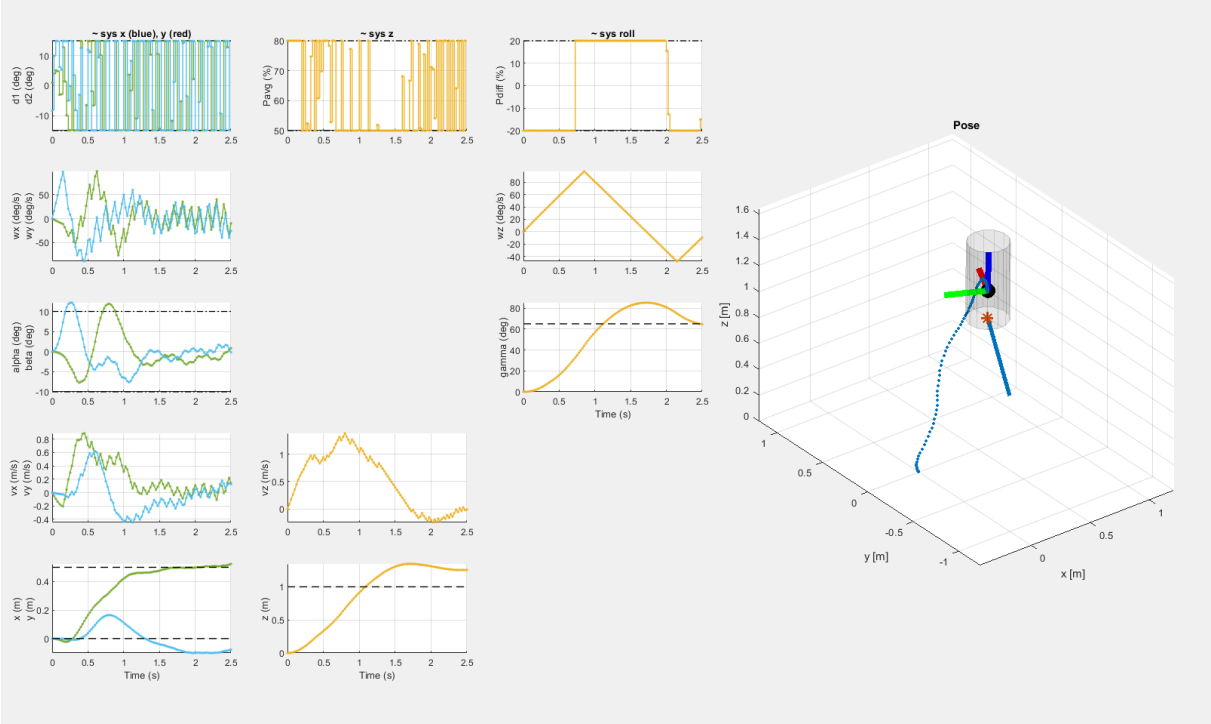

FIGURE 6.6
NMPC Controller performances, partially delay-compensating | delay = 5, compensated delay = 4

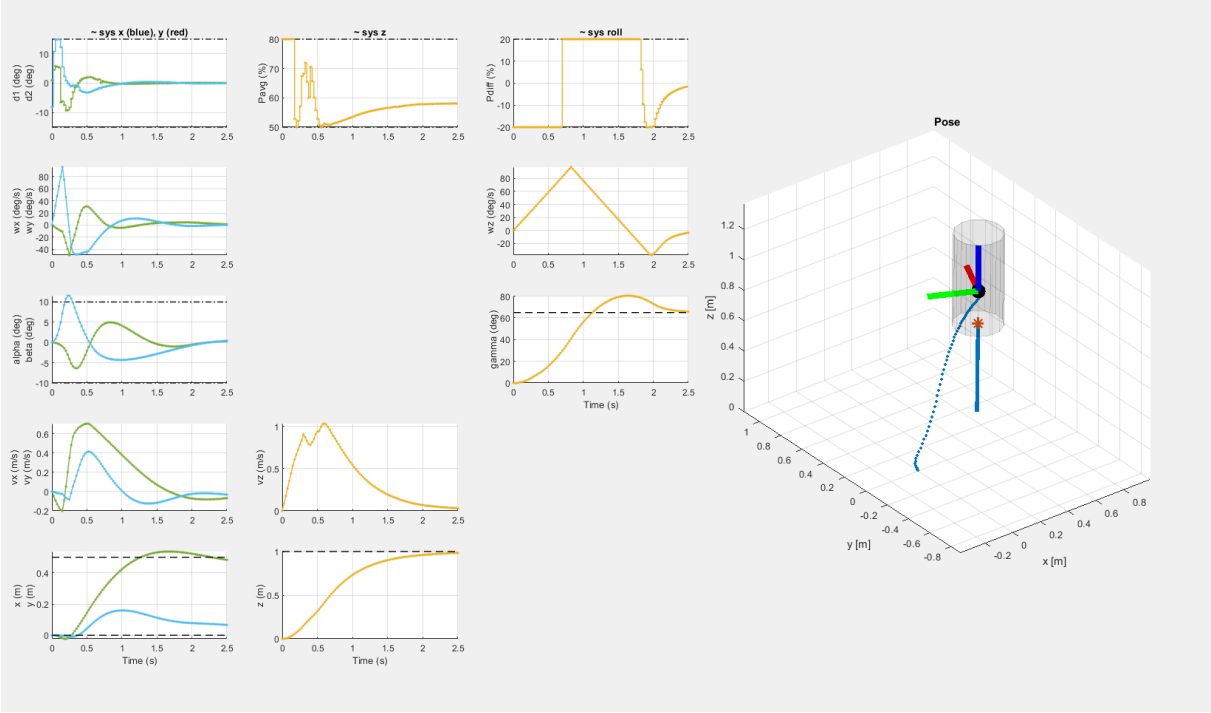

FIGURE 6.7
NMPC Controller performances, fully delay-compensating | delay = 5

We can see that for the partially delay-compensating controller, only a one sampling period of delay difference makes the closed-loop system barely stable, anything below this value make the system unstable. However, we can see that the fully delay-compensating controller can achieve good performances.