ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Winter 2024

Model Predictive Control - Rocket Mini Project

Group G:
Selma BENHASSINE (SCIPER 300148)
Salim BOUSSOFARA (SCIPER 311800)
Oussama JAFFAL (SCIPER 301816)

Table of Contents

1 Introduction

1.1 System Dynamics

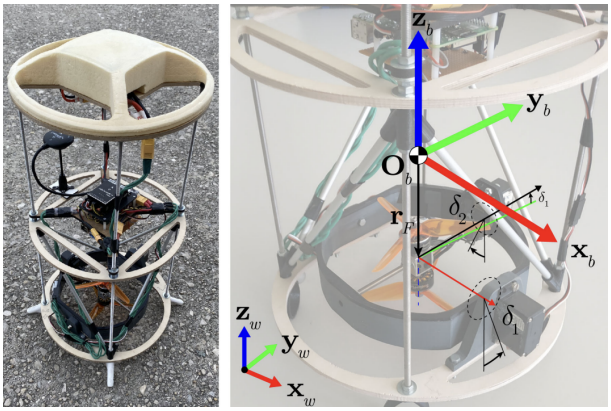


Figure 1: Rocket-shape drone

In this project, we will develop MPC controllers to control a rocket-shaped drone. To do so, one should understand the dynamics of the system which will be described by a 12-state vector

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\Omega}^T & \boldsymbol{\Phi}^T & \mathbf{v}^T & \mathbf{p}^T \end{bmatrix}^T$$

where $\boldsymbol{\Omega}$ are the angular velocities about the body axes, $\boldsymbol{\Phi}$ represent the attitude of the body frame with respect to the world frame, $\mathbf{v}$ and $\mathbf{p}$ represent respectively the velocity and the position expressed in the world frame.

The model will have as input vector:

$$\mathbf{u} = \begin{bmatrix} \delta_1 & \delta_2 & P_{\text{avg}} & P_{\text{diff}} \end{bmatrix}^T$$

where δ_1 and δ_2 are the deflection angles of servo 1 and servo 2, respectively, up to $\pm 15\circ$ ($= 0.26$ rad). $P_{\text{avg}} = (P_1 + P_2)/2$ and $P_{\text{diff}} = P_2 - P_1$ with P_1 and P_2 the power settings of motor 1 and 2.

The model will, then, control the combined propellers that produce a thrust force F whose magnitude depends on P_{avg} and a differential moment M_Δ whose magnitude depends on P_{diff}. The following image shows how they will be applied with $\mathbf{e}_F(\delta_1, \delta_2) = \begin{bmatrix} \sin(\delta_2) \\ -\sin(\delta_1) \cdot \cos(\delta_2) \\ \cos(\delta_1) \cdot \cos(\delta_2) \end{bmatrix}$

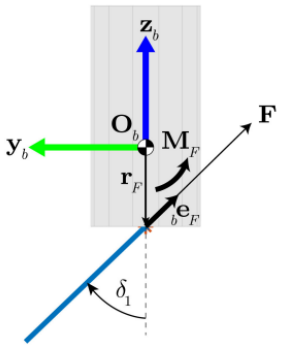


Figure 2: Side view of the rocket

When the axis is tilted, the resulting force and moment acting on the body, expressed in body frame, are:

- ${}_b\mathbf{F} = F \cdot_b \mathbf{e}_F$
- ${}_b\mathbf{M} = M_\Delta \cdot_b \mathbf{e}_F +_b \mathbf{M}_F$, with ${}_b\mathbf{M}_F = \mathbf{r}_F \times_b \mathbf{F}$

Understanding this is crucial to properly understand the linear and angular dynamics:

- **Linear dynamics expressed in the inertial frame:**
  $\dot{\mathbf{v}} = \mathbf{T_{wb}} \cdot_b \mathbf{F}/m - \mathbf{g}$; $\mathbf{T_{wb}}$ being defined as ${}_w\mathbf{F} = \mathbf{T_{wb}} \cdot_b \mathbf{F}$, m the rocket mass, and $\mathbf{g} = [0\ 0\ g]^T$
- **Angular dynamics:**
  $\dot{\omega} = \mathbf{J}^{-1}(-\omega \times \mathbf{J}\omega +_b \mathbf{M})$; J being the inertia matrix of the rocket

Having the dynamics, we can analyze the kinematics of the attitude that is defined by $\varphi = [\alpha\beta\gamma]^T$.

$$\dot{\varphi} = \frac{1}{\cos\beta}\begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma\cos\beta & \cos\gamma\cos\beta & 0 \\ -\cos\gamma\sin\beta & \sin\gamma\sin\beta & \cos\beta \end{bmatrix}\omega$$

Putting all together, we get the dynamic equations for the rocket $\dot{x} = f(x, u)$

2 Linearization

2.1 Independent Subsystems

The system is described by dynamic equations $\dot{x} = f(x, u)$. The rocket system comprises 12 states where the state vector is denoted by $x = [\vec{\omega}, \vec{\phi}, \vec{v}, \vec{p}]^T$, representing angular velocities (ω), Euler angles (ϕ), velocity (v), and position (p). The project description indicates a link between certain states and inputs: thrust vector angles (δ_1 and δ_2) to positions, average throttle (P_{avg}) to height, and differential throttle (P_{diff}) to roll angle. This linkage allows for the separation into four independent subsystems, each influenced by specific inputs.

The linearisation process assumes small variations around the nominal point. Small angle approximation is justified for small angles of α and β. The rocket's movements are consequences of applied forces and moments. The small angle variations lead to an approximation on the axis along which the force and moment are applied.

The subsystems and specific inputs are given by:

Subsystem X: The angular velocity ω_y, rotation β around the Y-axis, velocity v_x, and position x describe the rocket's motion along the X-direction. The control input δ_2 influences the states β, ω_y, x, and v_x in this subsystem.

Subsystem Y: The angular velocity ω_x, rotation α around the X-axis, velocity v_y, and position y describe the rocket's motion along the Y-direction. The control input δ_1 influences the states α, ω_x, y, and v_y in this subsystem.

Subsystem Z: The velocity v_z and position z together describe the rocket's motion along the Z-direction. The physical input P_{avg} influences the states v_z and z in this subsystem.

Subsystem Roll: The angular velocity ω_z and rotation γ around the Z-axis describe the rocket's roll motion. The input P_{diff}, representing the differential throttle, influences the states ω_z and γ in this subsystem.

In summary, the separation into independent subsystems is both mathematically justified by the structure of the dynamic equations and physically justified by the influence of specific states on the behavior of each subsystem. This decomposition facilitates modular control design and simplifies the analysis and synthesis of the overall rocket control system.

3 Design MPC Controllers for each Subsystem

3.1 Design MPC Regulators

3.1.1 Design Procedure

The MPC controller is designed with a dual-horizon approach to optimize system performance and stability. Finite Horizon control is achieved through LQR, while Infinite Horizon stability is assured via a terminal set constraint $x_N \in X_f$ and a terminal cost matrix Q_f, derived from the discrete-time algebraic Riccati equation. The cost function that is minimized by the controller is defined as the following.

$$\begin{aligned}
\text{Cost Function:} \quad & J^*(x) = \min \sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T Q_f x_N \\
\text{State Dynamics:} \quad & x_{i+1} = A x_i + B u_i \\
\text{State Constraints:} \quad & F x_i \leq f \\
\text{Input Constraints:} \quad & M u_i \leq m \\
\text{Terminal Set Constraints:} \quad & x_N \in X_f
\end{aligned}$$

The state and input cost matrices Q and R influences the tradeoff between optimizing performance and minimizing control effort and is also tailored to each subsystem. The constraint matrices F, f, M, m, are tuned according to the specific controller, and are defined by the following constraints on state and input.

$$\begin{aligned}
\textbf{Subsystem X:} \quad & \text{State Constraint:} \quad |\beta| \leq 7^\circ \\
& \text{Input Constraint:} \quad |\delta_2| \leq 15^\circ \\
\textbf{Subsystem Y:} \quad & \text{State Constraint:} \quad |\alpha| \leq 7^\circ \\
& \text{Input Constraint:} \quad |\delta_1| \leq 15^\circ \\
\textbf{Subsystem Z:} \quad & \text{Input Constraint:} \quad 50\% \leq P_{\text{avg}} \leq 80\% \\
\textbf{Subsystem Roll:} \quad & \text{Input Constraint:} \quad |P_{\text{diff}}| \leq 20\%
\end{aligned}$$

In implementing the MPC controller, the system is governed by a cost function $J^*(x)$, comprising stage costs over a finite horizon and a terminal cost for infinite stability. The state dynamics $x_{i+1} = Ax_i + Bu_i$ and associated constraints ($Fx_i \leq f$ and $Mu_i \leq m$) guide the optimization process, ensuring both state and input recursive satisfaction. This controller design is used for all 4 subsystems, with its parameters tuned accordingly.

3.1.2 Tuning Parameters

Here we will go more into detail about our tuning choices for F, f, M, m, Q and R for the X, Y, Z, and Roll MPC Controllers.

Subsystem X: State Cost: $Q = 10 \cdot I$

Input Cost: $R = 1$

State Constraints: $F = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, \quad f = \begin{bmatrix} 0.1745 \\ 0.1745 \end{bmatrix}$

Input Constraints: $M = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad m = \begin{bmatrix} 0.26 \\ 0.26 \end{bmatrix}$

The subsystem X controller is pretty straightforward. The simulation settled in 7 seconds without needing to make many changes to Q and R. The state and input constraints correspond to those previously defined for β and δ_2.

Subsystem Y: State Cost: $Q = 10 \cdot I$

Input Cost: $R = 1$

State Constraints: $F = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, \quad f = \begin{bmatrix} 0.1745 \\ 0.1745 \end{bmatrix}$

Input Constraints: $M = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad m = \begin{bmatrix} 0.26 \\ 0.26 \end{bmatrix}$

The subsystem Y controller is also simple. The simulation settles in 7 seconds as well with the same Q and R values. The state and input constraints correspond to those previously defined for α and δ_1.

Subsystem Z: State Cost: $Q = 10 \cdot I$

Input Cost: $R = 1$

State Constraints: $F = \begin{bmatrix} 0 & 0 \end{bmatrix}, \quad f = \begin{bmatrix} 0 \end{bmatrix}$

Input Constraints: $M = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad m = \begin{bmatrix} 80 - 56.667 \\ -50 + 56.667 \end{bmatrix}$

The subsystem Z controller was also able to settle within the time constraint given, but the input constraints do not exactly match those defined for P_{avg} previously. They had to be modified according to the trim point us, in which there was an offset of 56.667. There are no state constraints.

Subsystem Roll: State Cost: $Q = 20 \cdot I$

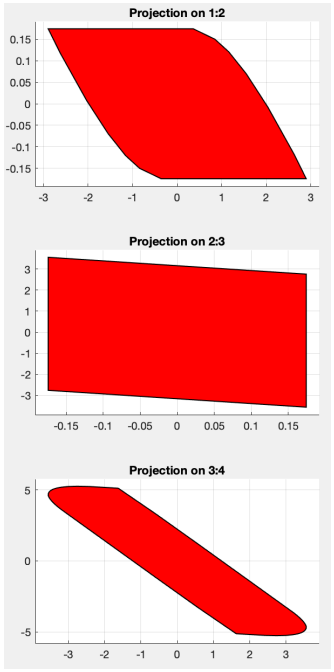
Input Cost: $R = 1$

State Constraints: $F = \begin{bmatrix} 0 & 0 \end{bmatrix}, \quad f = \begin{bmatrix} 0 \end{bmatrix}$

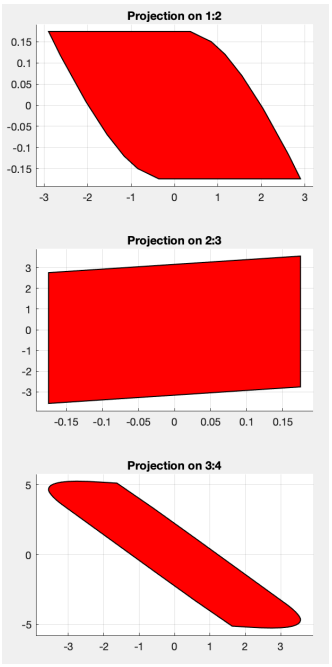
Input Constraints: $M = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad m = \begin{bmatrix} 20 \\ 20 \end{bmatrix}$

The subsystem Roll controller is similar to the others, with no state constraints and the input constraint on P_{diff} reflecting those prescribed earlier.

3.1.3 Terminal Invariant Sets

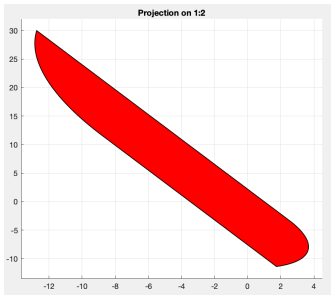


(a) Terminal Invariant Set for X

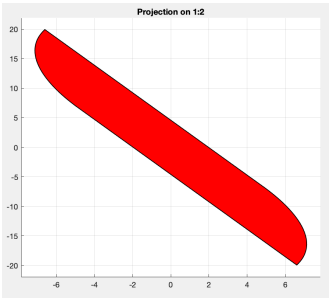


(b) Terminal Invariant Set for Y

Figure 3: Terminal Invariant Sets for X and Y



(a) Terminal Invariant Set for Z



(b) Terminal Invariant Set for Roll

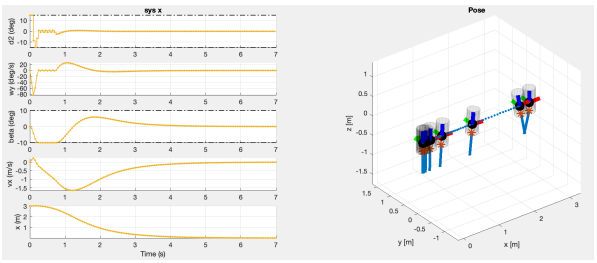
Figure 4: Terminal Invariant Sets for Z and Roll

We notice that the invariant sets of X and Y are almost identical, as well as those of Z and Roll.

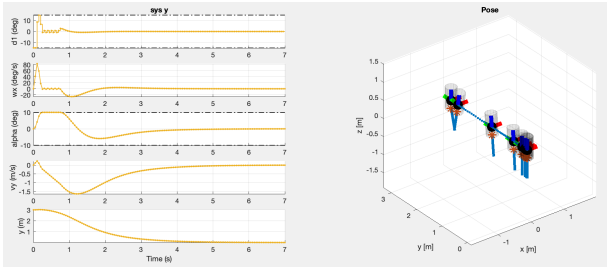
The initial states of the open and closed loop trajectories in the following portions are the following:

$$
\begin{aligned}
x_0 &= [w_y\ \beta\ v_x\ x] &&= [0\ 0\ 0\ 3] \\
y_0 &= [w_x\ \alpha\ v_y\ y] &&= [0\ 0\ 0\ 3] \\
z_0 &= [v_z\ z] &&= [0\ 3] \\
\text{roll}_0 &= [w_z\ \gamma] &&= [0\ 40°]
\end{aligned}
$$

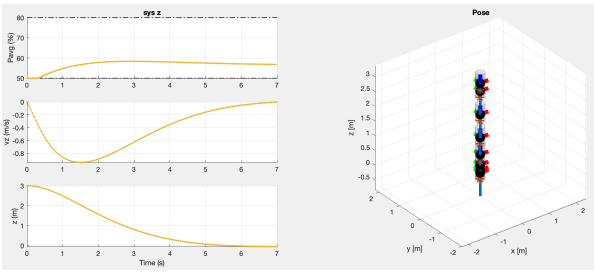
3.1.4 Open-Loop Trajectory



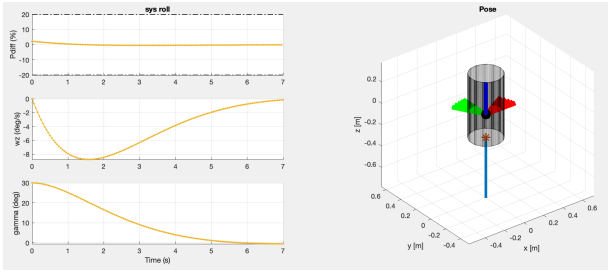
(a) Open-Loop Trajectory for X



(b) Open-Loop Trajectory for Y

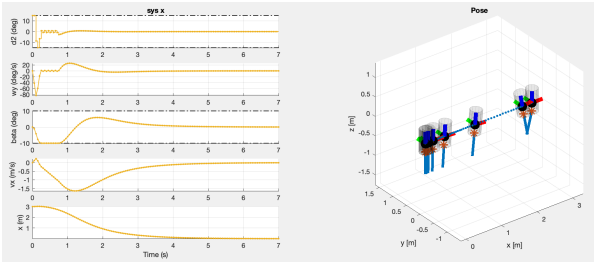


(c) Open-Loop Trajectory for Z

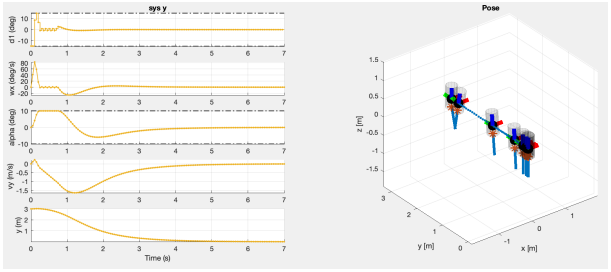


(d) Open-Loop Trajectory for Roll

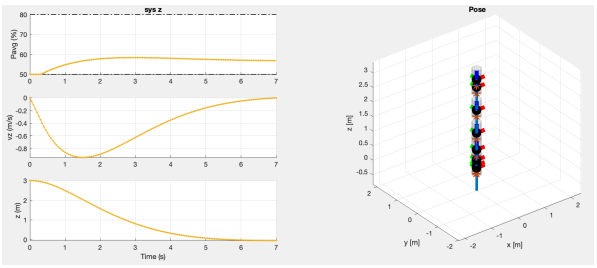
3.1.5 Closed-Loop Trajectory



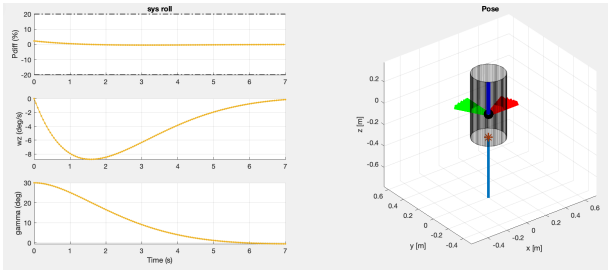
(a) Closed-Loop Trajectory for X



(b) Closed-Loop Trajectory for Y



(c) Closed-Loop Trajectory for Z



(d) Closed-Loop Trajectory for Roll

3.2 Design MPC Tracking Controllers

3.2.1 Design Procedure

In this section, we develop an MPC tracking controller using YALMIP as before. The setup involves defining the predictive state and input trajectories, denoted by X and U, respectively. Deviation variables are introduced for tracking the objective:

$$x_{\text{shift}} = x - xs$$
$$u_{\text{shift}} = u - us$$

These variables are then incorporated into the optimization problem outlined in Part 3.1. The specific constraints and objectives for each controller are as follows:

$$\text{Con} = \begin{bmatrix} X_{i+1} == AX_i + BU_i \\ FX \leq f \\ MU \leq m \end{bmatrix}$$

$$\text{Obj} = \sum_{i=1}^{N-1} (x_{\text{shift}}^T Q \Delta x_{\text{shift}} + u_{\text{shift}}^T R u_{\text{shift}}) + (\Delta x_{\text{shiftN}}^T Q_f x_{\text{shiftN}})$$

As in part 3.1, Q and R are cost matrices, and Q_f is the solution to the discrete-time algebraic Riccati equation. In fact, the only difference above with part 3.1 is the objective, for which the deviated states and inputs are used. The tracking portion of the controller is defined in the steady state function, which aims to find a feasible steady-state target (xs, us) that satisfies the system constraints.

$$\text{Con} = \begin{bmatrix} M \cdot us \leq m \\ F \cdot xs \leq f \\ xs == A \cdot xs + B \cdot us \\ ref == C \cdot xs + D \end{bmatrix}$$

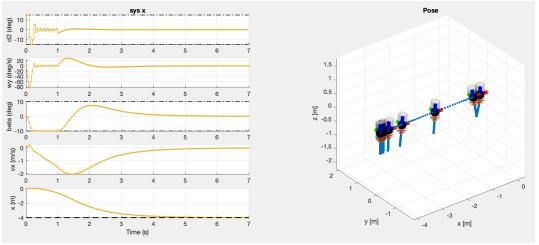
$$\text{Obj} = us^2$$

Here, ref is the reference position. This formulation aims to find a feasible steady-state target (xs, us) that satisfies the system constraints and follows the reference path.

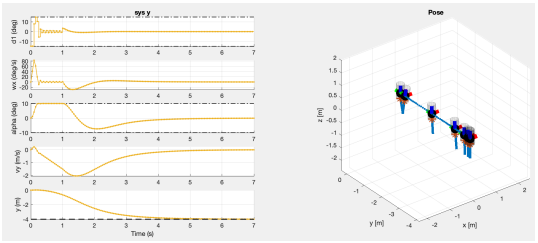
3.2.2 Tuning Parameters

All of our constraints and cost matrices remain identical to those of part 3.1. The initial states are 0 for all subsystems and all variables. The controllers track a reference of -4 for x, y, z, and 35° for γ.

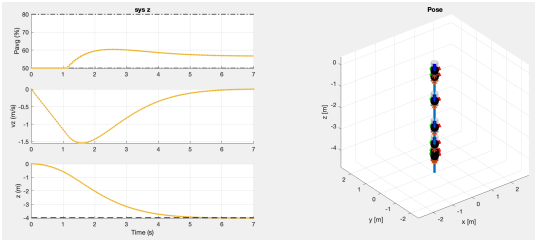
3.2.3 Open-Loop Trajectory



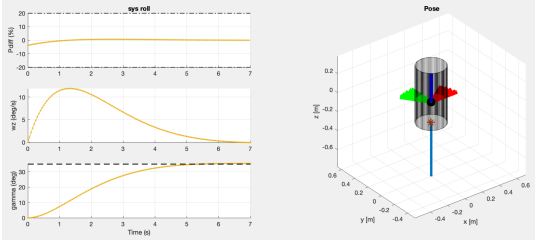
(a) Open-Loop Trajectory for X



(b) Open-Loop Trajectory for Y

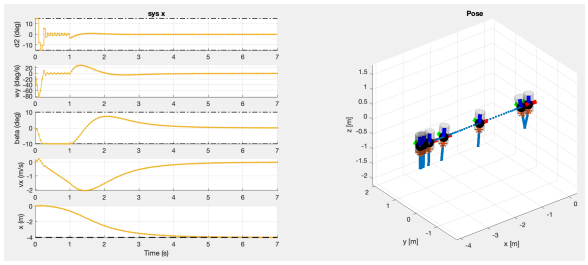


(c) Open-Loop Trajectory for Z

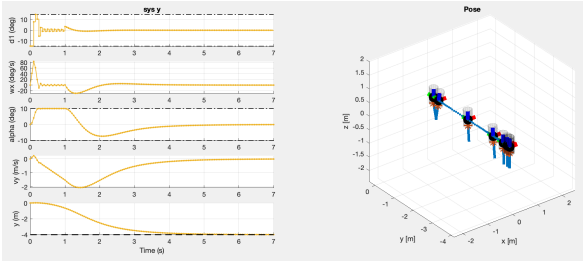


(d) Open-Loop Trajectory for Roll

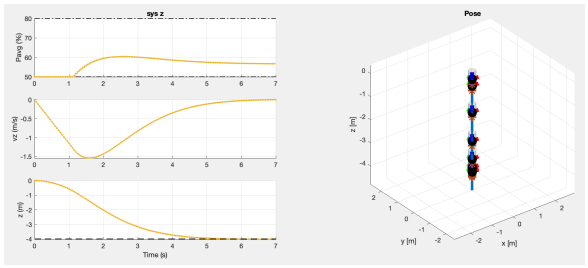
3.2.4 Closed-Loop Trajectory



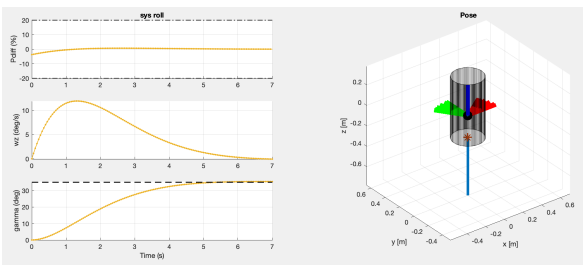
(a) Closed-Loop Trajectory for X



(b) Closed-Loop Trajectory for Y



(c) Closed-Loop Trajectory for Z



(d) Closed-Loop Trajectory for Roll

4 Simulation with Nonlinear Rocket

4.1 Tracking Reference Trajectory

These are the plots created by the open and closed loop trajectories for the nonlinear rocket with our linear controllers.

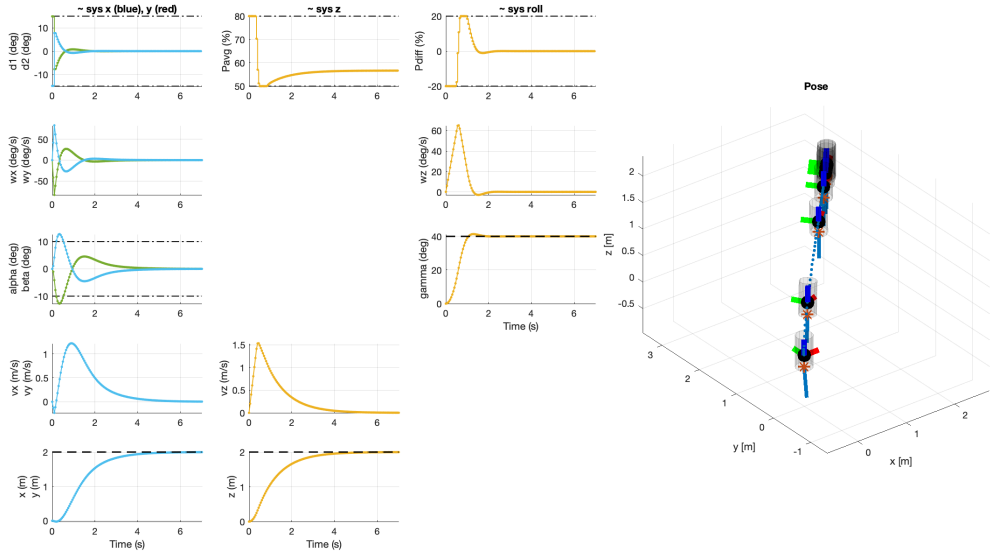


Figure 9: Open-Loop Trajectory for Nonlinear Rocket

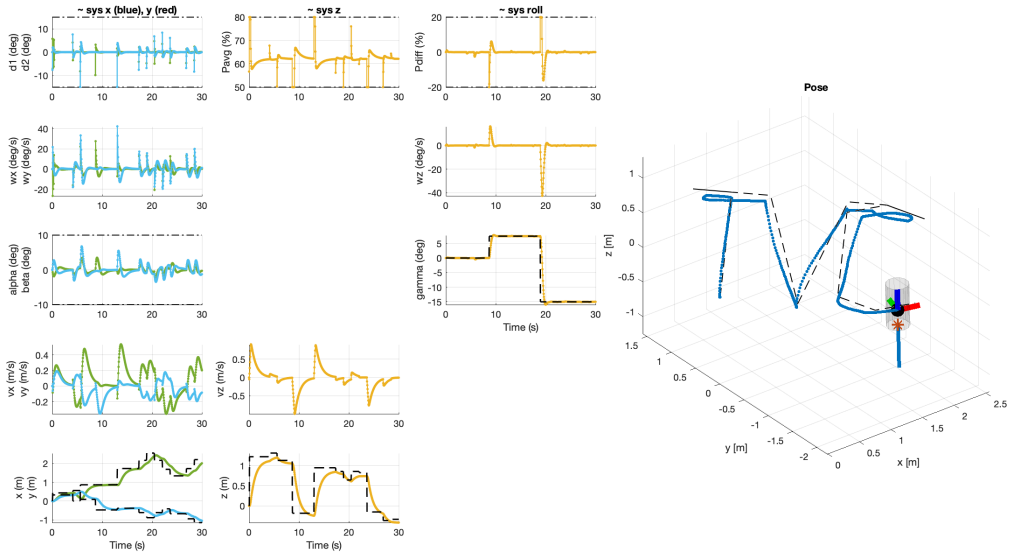


Figure 10: Closed-Loop Trajectory for Nonlinear Rocket

4.1.1 Tuning Parameters

Here we will go more into detail about our changes to Q and R for the X, Y, Z, and Roll MPC Controllers for the nonlinear rocket.

$$\textbf{Subsystem X:} \quad \text{State Cost:} \quad Q = 50 \cdot I$$
$$\text{Input Cost:} \quad R = 1$$

The subsystem X controller is remains straightforward. The state cost increases due to the mismatch between the linear controller and the nonlinear rocket.

$$\textbf{Subsystem Y:} \quad \text{State Cost:} \quad Q = 50 \cdot I$$
$$\text{Input Cost:} \quad R = 1$$

The subsystem Y controller is identical to the X controller.

$$\textbf{Subsystem Z:} \quad \text{State Cost:} \quad Q = 5000 \cdot I$$
$$\text{Input Cost:} \quad R = 0.5$$

The subsystem Z controller had the biggest model mismatch and worst tracking ability. The state cost Q needed to be very high to reach the target trajectory, reflecting the importance of precise state tracking in P_{avg}. To balance the state cost, the input cost R was reduced, creating a compromise between state tracking and control effort.

$$\textbf{Subsystem Roll:} \quad \text{State Cost:} \quad Q = \begin{bmatrix} 1 & 0 \\ 0 & 500 \end{bmatrix}$$
$$\text{Input Cost:} \quad R = 0.01$$

There is a large emphasis on tracking γ as reflected in the state cost Q. The input cost R is set to 0.01, indicating a strong preference for minimizing control effort.

4.2 Soft State Constraints

A slack variable was introduced for the inequality constraints of the x, y, z and roll states for both the controller and steady-state target optimization. This ensures a soft enforcement of the constraints, making the problem feasible under model mismatch, and contributing to the robustness of the control system.

5 Offset-Free Tracking

In this part, we assume that the mass of the rocket changes from what we have already modeled. The dynamics if the system in z-direction are, therefore, changed to:

$$x^+ = Ax + Bu + Bd$$

with d being a constant, unknown disturbance simulating the variation of the mass which we will compensate by extending the z-controller to track setpoint references with no offset.

5.1 Design Offset-Free Tracking Controller for Z

5.1.1 Design Procedure

To design an offset-free tracking controller, we need to incorporate an observer to estimate the offset and the state of the system and update the controller to reject the disturbance and track setpoint references with no offset. The estimator is based on the following augmented model:

$$\begin{bmatrix}\hat{x}_{k+1}\\ \hat{d}_{k+1}\end{bmatrix} = \begin{bmatrix}A & B_d\\ 0 & I\end{bmatrix}\begin{bmatrix}\hat{x}_k\\ \hat{d}_k\end{bmatrix} + \begin{bmatrix}A\\ 0\end{bmatrix}u_k + \begin{bmatrix}L_x\\ L_d\end{bmatrix}(C\hat{x}_k + C_d\hat{d}_k - y_k)$$

where $\hat{x}$ and $\hat{d}$ represent the state and disturbance estimates. From this, we can easily derive the error dynamics:

$$\begin{bmatrix}x_{k+1} - \hat{x}_{k+1}\\ d_{k+1} - \hat{d}_{k+1}\end{bmatrix} = \left(\begin{bmatrix}A & B_d\\ 0 & I\end{bmatrix} + \begin{bmatrix}L_x\\ L_d\end{bmatrix}\begin{bmatrix}C & C_d\end{bmatrix}\right)\begin{bmatrix}x_k - \hat{x}_k\\ d_k - \hat{d}_k\end{bmatrix}$$

We can, therefore, construct the new A, B and C matrices:

- $\bar{A} = \begin{bmatrix}A & B\\ 0 & I\end{bmatrix}$
- $\bar{B} = \begin{bmatrix}B\\ 0\end{bmatrix}$
- $\bar{C} = \begin{bmatrix}C & 0\end{bmatrix}$

5.1.2 Tuning Parameters

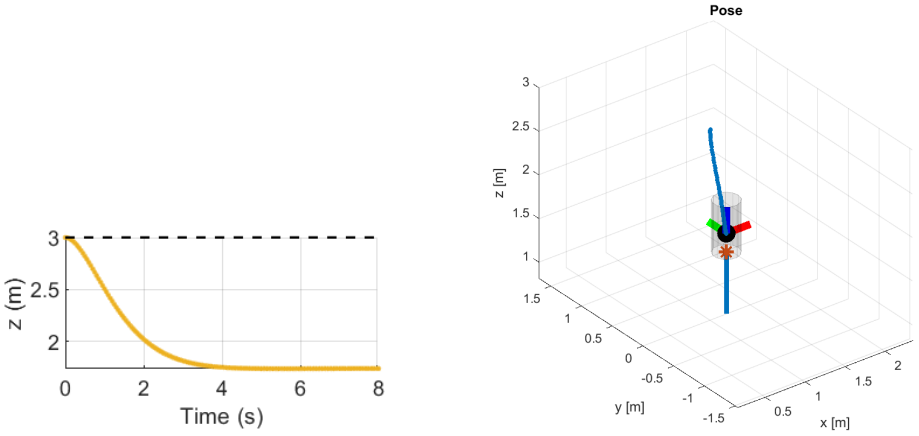
The observer gain matrix, $\bar{L}$, is fine-tuned manually to stabilize the error dynamics, ensuring they not only remain stable but also converge to zero. This fine-tuning involves adjusting the poles within the augmented observer loop, expressed as $\bar{A} + \bar{L}\bar{C}$, so that rapid response and minimal error are assured, given by the selected poles $\begin{bmatrix}0.1, & 0.2, & 0.3\end{bmatrix}$. While integrating an observer alters the system configuration, the fundamental control objectives are preserved. This is reflected by maintaining the established values of Q and R matrices from section 3.1, meaning that the primary aims of system performance—error minimization and control effectiveness— remain unchanged despite the new estimator dynamics.

Listing 1: setup_estimator

```
A = mpc.A;
B = mpc.B;
C = mpc.C;
nx  = size(A,1);
nu  = size(B,2);
ny  = size(C,1);
A_bar = [A, B;
         zeros(1,nx), ones(nu)];
B_bar = [B;
         zeros(1,nu)];
C_bar = [C, zeros(ny,1)];
L = -place(A_bar',C_bar',[0.1,0.2,0.3])';
```

5.1.3 Changing Mass on Original Controller

We have a dramatic change. The rocket goes down and stabilizes at a value which is much lower than the desired reference. This is in accordance with the expected behavior. In fact, the designed controller was not designed to be robust to disturbances.



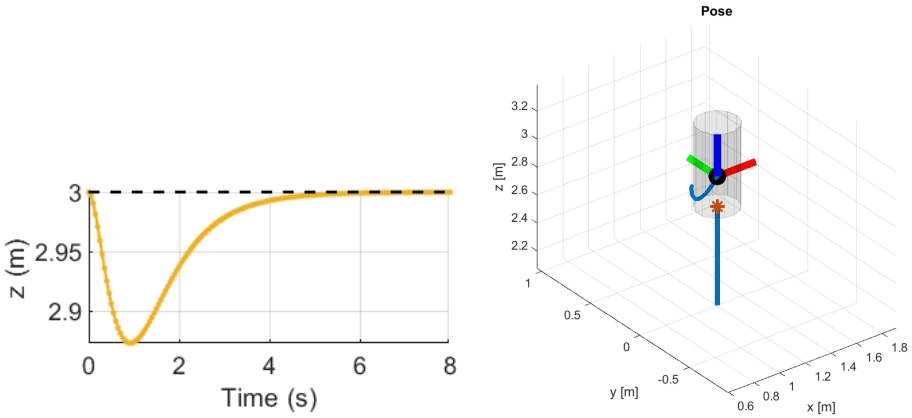
(a) Evolution of the height

(b) Rocket's trajectory

Figure 11: Original controller with m = 2.13

5.1.4 Changing Mass on Offset-Free Tracking Controller

We can see that we obtain a first drop that is relatively quickly corrected to reach the desired reference without any offset in less than 5s. This shows that the controller is robust to a constant unknown disturbance since it succeeds in tracking the reference without any offset despite the change in the mass.



(a) Evolution of the height (b) Rocket's trajectory

Figure 12: Offset-free tracking controller with m = 2.13

The influence of the poles of the observer can be observed by comparing the 2 following images:

(a) L = [0.7, 0.8, 0.9] (b) L = [0.1, 0.2, 0.3]

Figure 13: Evolution of the height for different controllers

(a) L = [0.7, 0.8, 0.9] (b) L = [0.1, 0.2, 0.3]

Figure 14: Evolution of the velocity on the z-axis for different controllers

(a) L = [0.7, 0.8, 0.9] (b) L = [0.1, 0.2, 0.3]

Figure 15: Evolution of the average power for different controllers

We can, clearly, see that both lead to an offset-free tracking but the bigger poles present a much higher drop in altitude and, then, a small overshoot before stabilizing. The same pattern is observed for the velocity on the z-axis as well for the average power.

5.2 Changing Mass Simulation

In this part, we assume that half of the initial rocket mass is actually fuel, and it will decrease depending on the power consumption of the motor.

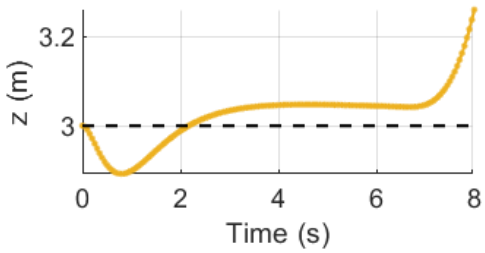
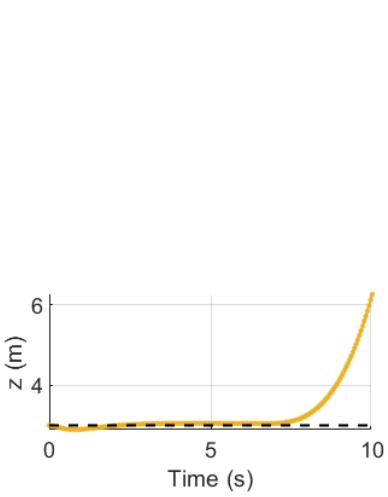


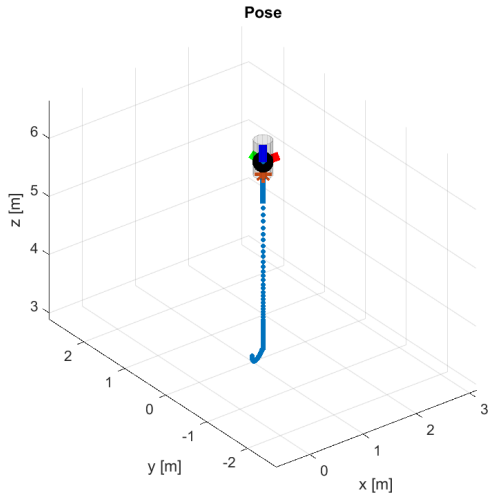
Figure 16: Evolution of the altitude for offset-free tracking controller with m = 2.13 and mass-rate = -0.27 for 8s

Our offset-free tracking controller does not seem to work well. In fact, a tracking offset in height appears which is not surprising since it is not designed to be robust to such dynamic disturbances. A possible way to handle dynamic changes in the disturbance is to use robust MPC which considers the worst-case scenarios within this optimization problem.

We observe that at the end of the time frame the rocket seems to rise up. We, therefore, increase the time frame to analyze this behaviour.



(a) Evolution of the height



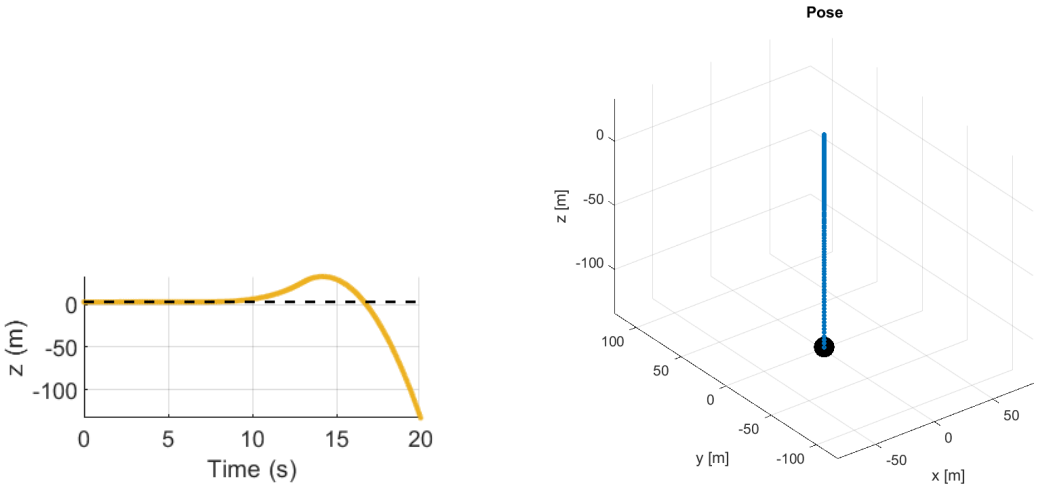
(b) Rocket's trajectory

Figure 17: Offset-free tracking controller with m = 2.13 and mass-rate = -0.27 for 10s

We observe multiple distinct properties of the trajectory:

- **Initial Steady Ascent:** Initially, the rocket seems to maintain a steady ascent. The controller is compensating for the gravitational force and providing enough thrust to lift the rocket steadily.
- **Mid-Flight Hovering or Slowed Ascent:** The rocket hovers at the desired reference without a certain offset as explain above.
- **Rapid Ascent Towards the End:** As the fuel is consumed, the mass of the rocket decreases significantly until it reaches 0 or less which is physically incorrect. However, in the simulation, the mass is erroneously allowed to go to zero due to the fuel consumption model which explains the sudden spike in ascent. No thrust is, then, needed to go upwards.

Increasing even more the time frame gives the following trajectory:



(a) Evolution of the height

(b) Rocket's trajectory

Figure 18: Offset-free tracking controller with m = 2.13 and mass-rate = -0.27 for 20s

We can observe a sudden downfall. This can be explained by the imposed simulation conditions. In fact, once all the fuel/energy is consumed, the motor will not produce any thrust anymore and the remaining rocket mass will then be again half of the initial mass. Having no thrust, a massive object will just go downwards under the effect of gravity. No limit has been given in the code, that is why it is going towards -infinity.

6 Nonlinear MPC

6.1 Design a Nonlinear MPC Controller

In this section, we will develop a Nonlinear Model Predictive Control (NMPC) that utilizes all twelve distinct states of the rocket. This approach differs from earlier methods where the rocket was divided into four separate sub-systems. The focus here will be on incorporating various equations and constraints essential for this tracking NMPC implementation.
The simulation is conducted with a standard maximum roll angle of $|\gamma_{\text{ref}}| = 15°$, and for an increased maximum roll reference of $|\gamma_{\text{ref}}| = 50°$.
Here is the non-linear optimization problem to address :

$$J^*(x) = \min \sum_{i=0}^{N-1} \left((x_i - x_{\text{ref}})^T Q (x_i - x_{\text{ref}}) + u_i^T R u_i \right) + (x_N - x_{\text{ref}})^T Q (x_N - x_{\text{ref}})$$

s.t.

$$\begin{aligned} U_{\text{lower}} - U &\leq 0; \\ U - U_{\text{upper}} &\leq 0; \\ x_0 &= X \end{aligned}$$

U_{lower} and U_{upper} are defined such that they are the lower and higher bounds for the inputs to the system and the constraints on the inputs are:

- δ_1 and δ_2 up to ± 0.26 rad
- P_{avg} between 50% and 80%
- P_{diff} up to $\pm 20\%$

The matrices Q and R are defined as follow :

$$Q = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}$$

$$R = \begin{bmatrix} 50 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix}$$

In the Q matrix of our NMPC system, we put a lot of emphasis on the gamma angle for controlling the rocket's orientation, and also on keeping the right height following the z-axis. This makes sure the rocket stays in the right direction and at the correct height, which is really important.
The use of RK4 function, representing the Runge-Kutta 4 method. This method is highly effective for integrating the dynamics of our rocket, leading us to the subsequent definition of the function:

$$\begin{aligned} k_1 &= f(X, U), \\ k_2 &= f\left(X + \frac{h}{2}k_1, U\right), \\ k_3 &= f\left(X + \frac{h}{2}k_2, U\right), \\ k_4 &= f(X + hk_3, U), \\ x_{\text{next}} &= X + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned}$$

Where h is the sampling rate.

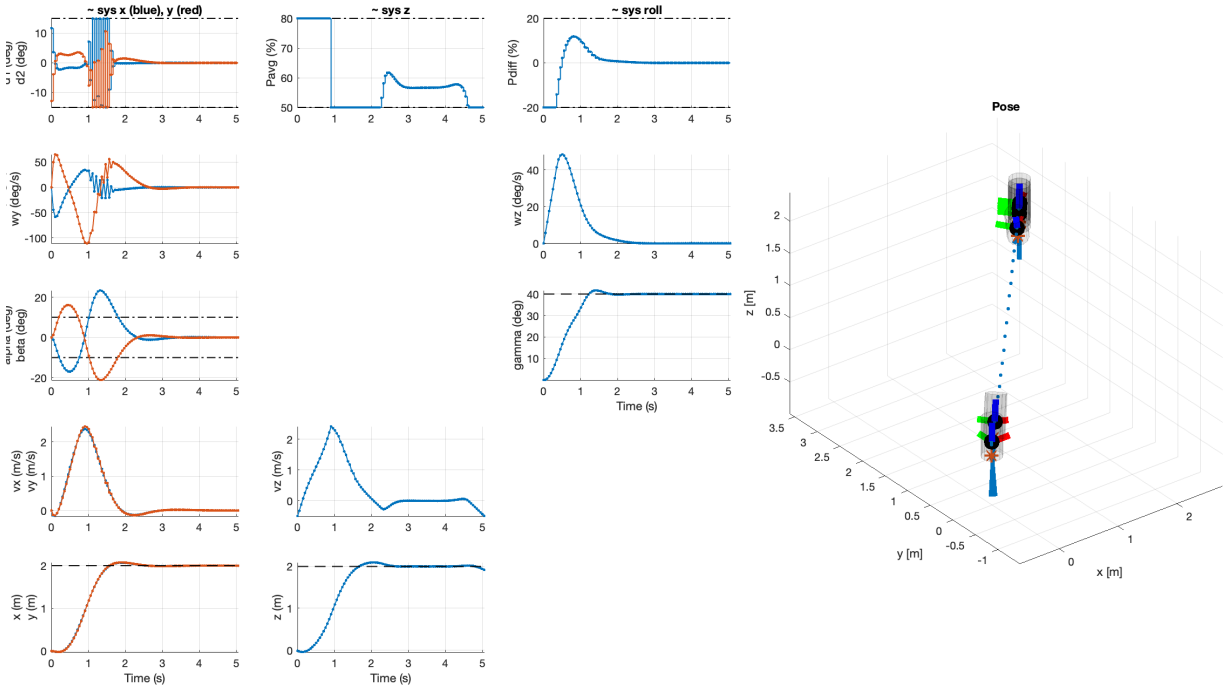


Figure 19: Plot of the open-loop system going to a reference point:

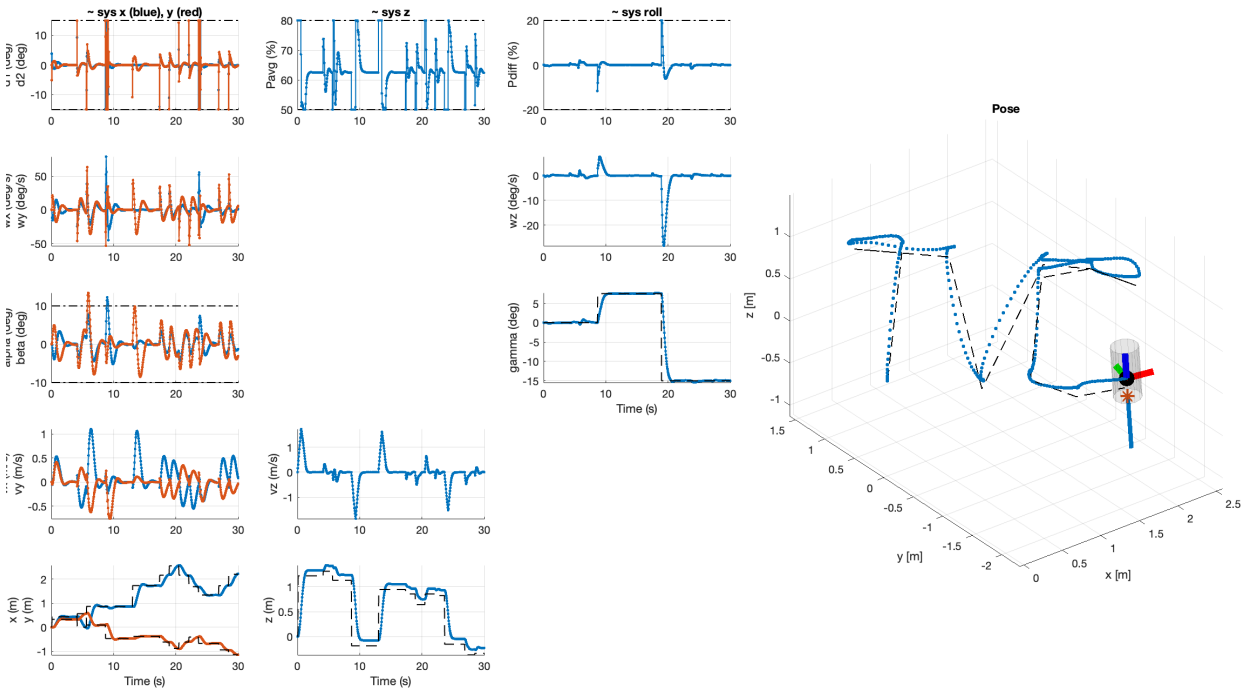


Figure 20: Plot of the closed-loop system going with a maximum roll angle of 15°:

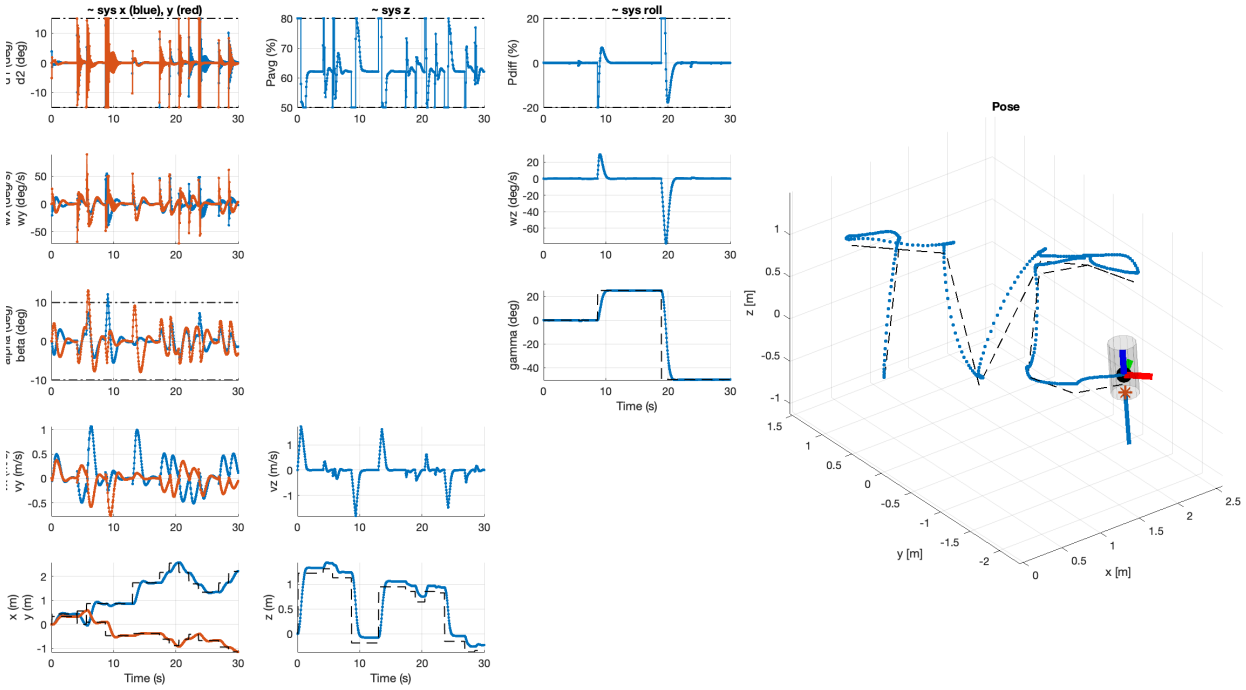


Figure 21: Plot of the closed-loop system going with a maximum roll angle of 50°:

Discuss the pros and cons of your nonlinear controller vs the linear ones you developed earlier:

A nonlinear controller is beneficial because it can handle nearly any type of model and goal. We observe that our nonlinear approach is highly effective with our rocket model and meets the tracking objective for the "TVC" pattern well.
On the downside, nonlinear controllers require substantial computational resources, which restricts us from setting the prediction horizon too long.

6.2 NMPC with Delay/Delay Compensation

How much delay is needed to observe a drop in closed-loop performance ? How much delay to make the closed-loop system unstable ?

Analyzing our system's response in a closed-loop setup with a reference point of (0.5, 0, 1, 65°) under different time delays reveals that the system starts to lose effectiveness with a delay as short as 1, or 25 milliseconds, as we start to notice the α and β angles not quite stable after 2.5sec, and turns unstable with a delay of 3, equating to 75 milliseconds. Comparing the system's behavior with a 25 ms delay against the no-delay scenario, there's noticeable irregularity in the and inputs and a slight offset in the point coordinates. Moreover, the system's instability becomes apparent at a 75 ms delay where various states, like the velocities, fail to stabilize and persistently oscillate, highlighting the system's unstable nature under these conditions.

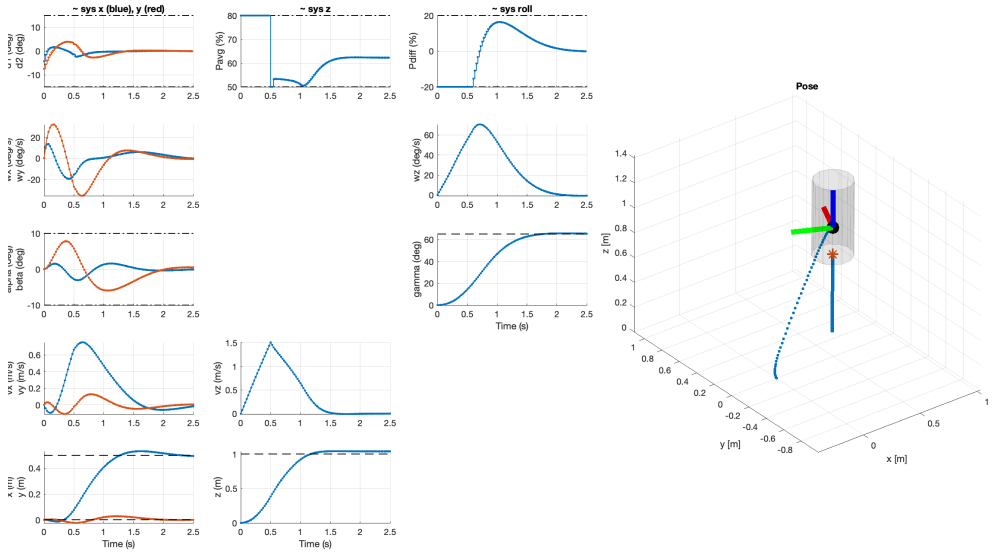


Figure 22: Plot of the closed-loop system with no delay

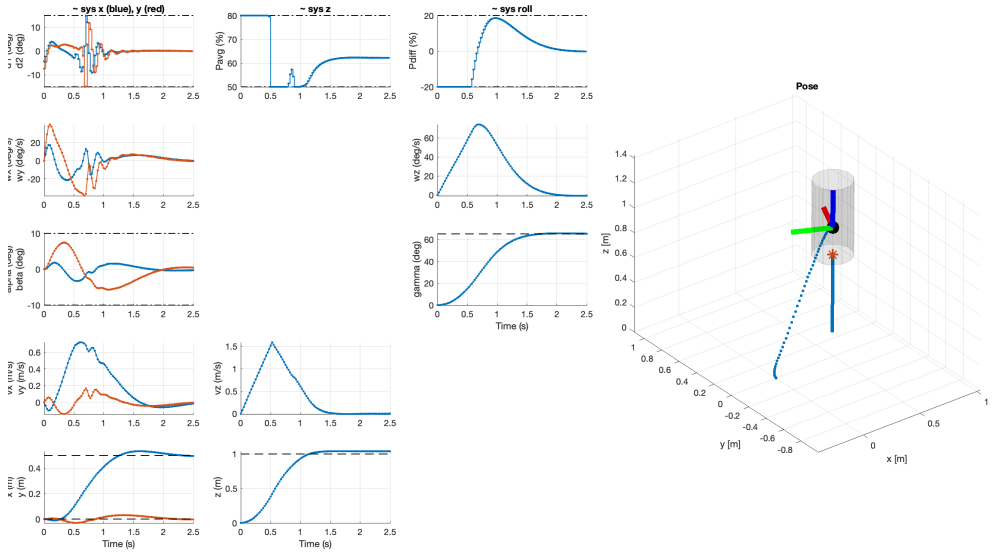


Figure 23: Plot of the closed-loop system with uncompensated delay of 1

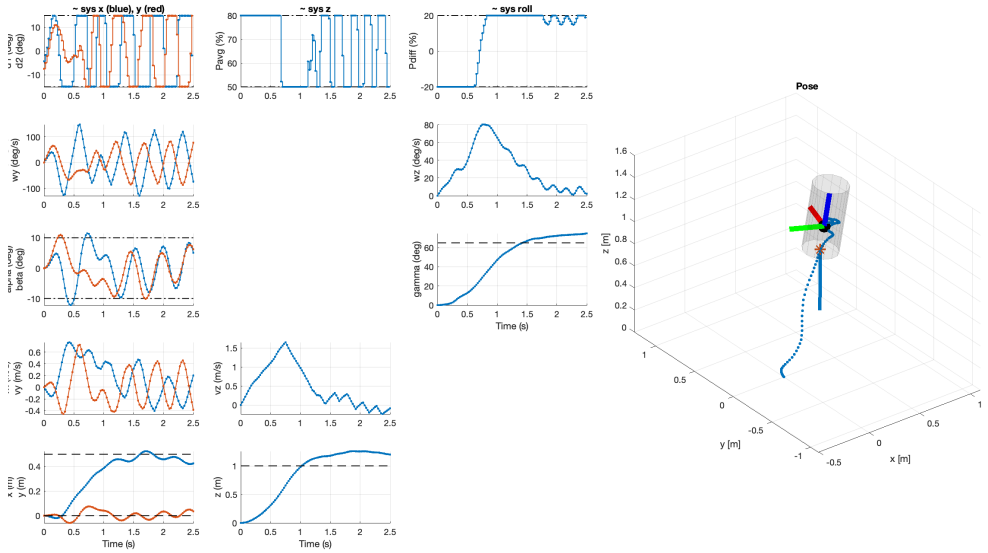


Figure 24: Plot of the closed-loop system with uncompensated delay of 3

Plot the closed-loop trajectory for a partially delay-compensating controller (compensated delay < actual delay), and a fully delay-compensating controller (compensated delay = actual delay)

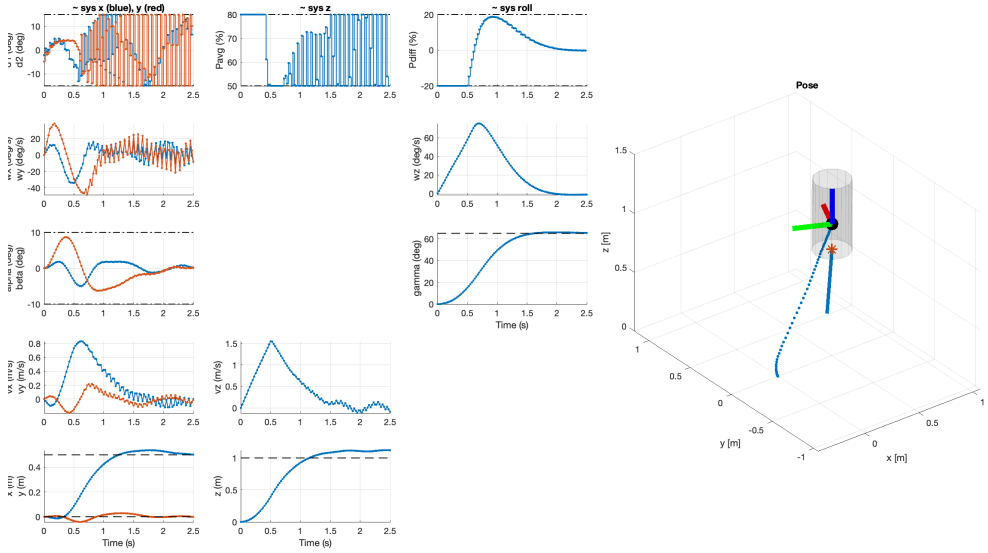


Figure 25: Plot of the closed-loop system with partial compensation

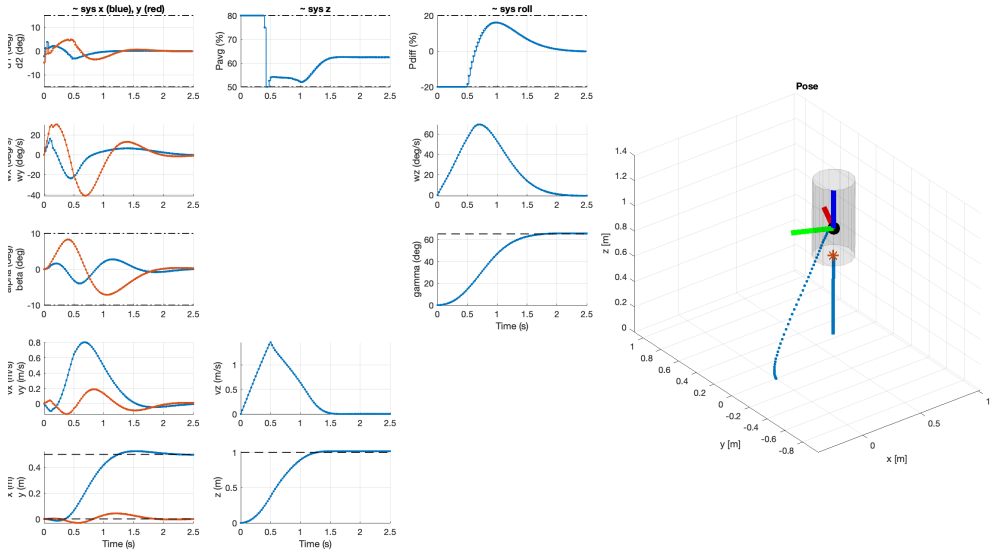


Figure 26: Plot of the closed-loop system fully compensated

In these trajectories we have implemented delay compensation. Figure 25 illustrates a partially compensated NMPC, where the compensation contributes to enhanced system stability but has is some oscillation in the delta inputs, and so they don't reach a steady state. Moving on to figure 26, we can see a fully compensated NMPC, and these outcomes closely resemble the scenario without delay in figure 22, exhibiting system instability with a smooth behavior.

7 Conclusion

To conclude, this project has been a good opportunity to put the theory that we have seen within the lectures into practice. In fact, we successfully implemented MPC on a rocket prototype, effectively handling complex dynamics and computational challenges.