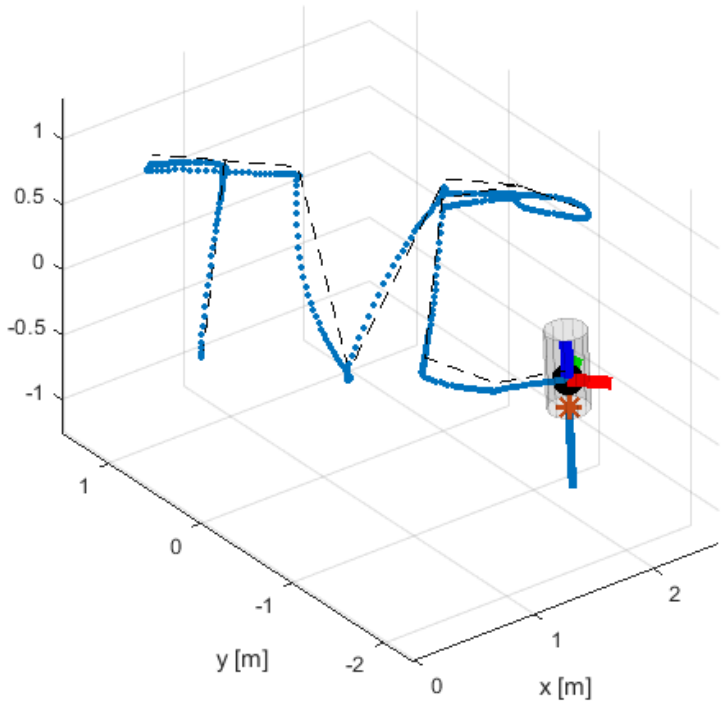
WS 2023-2024, ME-425 January 11, 2024

Mini-Project

Project Report - Model Predictive Control

Gaston Wolfart - 311916
Tim Lücking - 315026
Dimitri Jacquemont - 310572



Professor: Colin Jones

Deliverable 2.1

By studying the matrices **A** and **B** it is possible to separate the system into four independent systems. We can see that all subsystems are linearly independent if we divide the system into the following subsystems:

- X-axis: States: ω_y, β, x, v_x. Input: δ_2
- Y-axis: States: ω_x, α, y, v_y. Input: δ_1
- Z-axis: States: z, v_z. Input: P_{avg}
- Roll: States: ω_z, γ. Input: P_{diff}

From an intuitive/mechanical perspective, we can see that every input impacts only the state of their subsystem, and this enables the separation into subsystems:

- X-axis: When we change δ_2, the rocket turns around the y-axis, acting on the states ω_y and β. It also orients the thrust vector so that it has a component in the x direction, thus impacting x and v_x.
- Y-axis: When we change δ_1, the rocket turns around the x-axis, acting on ω_x and α. It also orients the thrust vector so that it has a component in the y direction, thus impacting y and v_y.
- Z-axis: Increasing the average of the thrust will only impact the altitude z and the velocity v_z, as the propellers turn in opposing directions.
- Roll: Increasing the difference of the propellers, will create an imbalance in the moment around the z-axis, causing a rotation around the z-axis, therefore acting only on ω_z and γ.

If we take for example roll and the Z-axis, we can see that even if we act on the same physical input (the propellers), we can still have an independent separation thanks to the abstraction of P_{avg} and P_{diff} that do not impact each other. Increasing the average will only increase v_z (and z) and will not change the roll. Similarly, decreasing the difference between the propellers will make the rocket turn on itself without acting on v_z.

Deliverable 3.1

This section details the implementation and design choices of the controllers, focusing on the design procedure that ensures recursive constraint satisfaction. It includes an explanation of the tuning parameters used - Q, R, H, and the terminal components. The design process of the terminal set and the application of tuning parameters are also discussed. Additionally, the terminal invariant sets for each dimension are illustrated, accompanied by open-loop and closed-loop plots. These plots demonstrate the system's behavior starting from a stationary position at 3 meters from the origin (x, y, and z axes) and at 30 degrees for roll.

Design procedure ensuring recursive constraint satisfaction

Recursive constraint satisfaction is the requirement for the system to consistently meet state and control constraints at each step of the prediction horizon. MPC design ensures that constraints are satisfied at each step by formulating the constraints within the prediction horizon and the application of terminal constraints. The length of the prediction horizon affects the ability to satisfy constraints recursively. A longer horizon might provide better foresight, but at the cost of increased computational complexity.

The optimization problem to solve :

$$\begin{aligned} J^*(x) = \min_{x,u} \quad & x_N^\top Q_f x_N + \sum_{i=0}^{N-1} \left(x_i^\top Q x_i + u_i^\top R u_i \right) \\ \text{subject to} \quad & x_{i+1} = A x_i + B u_i, \\ & F x_i \leq f, \\ & M u_i \leq m, \\ & x_N \in \mathcal{X}_f, \\ & x_0 = x. \end{aligned} \tag{1}$$

Matrix Q and R are the performance weights. The values of Q_f and X_f are chosen to simulate an infinite horizon. A and B describe the model of the system, M and F model the constraints that have to be satisfied.

Choice of tuning parameters

Selecting the Q and R cost matrix requires balancing tracking performance and control effort. The Q matrix (state penalty) emphasize the importance of maintaining certain states. Larger values penalize deviations more severely, encouraging closer adherence to the desired state. This must be balanced to avoid excessive control actions. The R matrix (control penalty) regulates the intensity of control actions. Higher values discourage aggressive control inputs, helping to prevent aggressive system behavior. The selection process involves trial and error, aiming for a balance that meets system performance and stability requirements. In the Q matrix, higher weights are typically assigned to the tracking variables such as `x`, `y`, `z`, and `roll`. This is because these are the main variables that the system aims to monitor and control accurately.

For **controller X**, constraint $|\beta| \leq 10°$ and $|\delta_2| \leq 15°$ are ensured by matrix F and f, M and m. The cost matrices for the sub-system are the following Q and R.

$$F = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad M = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad f = \begin{bmatrix} 0 \\ 0.1745 \\ 0 \\ 0 \\ 0 \\ 0.1745 \\ 0 \\ 0 \end{bmatrix} \quad m = \begin{bmatrix} 0.26 \\ 0.26 \end{bmatrix} \quad Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix} \quad R = 1$$

For **controller Y**, constraint $|\alpha| \leq 10°$ and $|\delta_1| \leq 15°$ are ensured by matrix F and f, M and m. The cost matrices for the sub-system are the following Q and R.

$$F = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad M = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad f = \begin{bmatrix} 0 \\ 0.1745 \\ 0 \\ 0 \\ 0 \\ 0.1745 \\ 0 \\ 0 \end{bmatrix} \quad m = \begin{bmatrix} 0.26 \\ 0.26 \end{bmatrix} \quad Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix} \quad R = 1$$

For **controller Z**, constraint $50\% \leq P_{avg} \leq 80\%$ is ensured by matrix M and m. The cost matrices for the sub-system are the following Q and R.

$$M = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad m = \begin{bmatrix} 80 \\ -50 \end{bmatrix} \quad Q = \begin{bmatrix} 1 & 0 \\ 0 & 20 \end{bmatrix} \quad R = 1$$

For **controller ROLL**, constraint $|P\text{diff}| \leq 20\%$ is ensured by matrix M and m. The cost matrices for the sub-system are the following Q and R.

$$M = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad m = \begin{bmatrix} 20 \\ 20 \end{bmatrix} \quad Q = \begin{bmatrix} 1 & 0 \\ 0 & 100 \end{bmatrix} \quad R = 0.01$$

The prediction horizon H is the time window over which future system behavior is predicted and optimized. A longer horizon can provide a more comprehensive view of the system's future behavior, leading to more optimized control actions, however, it also increases computational complexity. A shorter horizon reduces computational complexity but might miss longer-term considerations. In this project, H was chosen based on empirical testing and trade-offs between performance and computational feasibility. The prediction horizon in this deliverable is set to 4, which lead to satisfying results.

The terminal component includes a terminal cost and terminal constraints. It is crucial for ensuring the stability and feasibility of the control system:

- **Terminal Cost** $x_N^\top Q_f x_N$: This is an additional cost function applied to the state at the end of the prediction horizon. It is designed to ensure that the system's final state is within a stable region. In this project, the terminal cost is chosen based on the solution to the associated discrete-time algebraic Riccati equation (Q_f in this context).
- **Terminal Constraints** $x_N \in \mathcal{X}_f$: These constraints are applied to the state at the end of the prediction horizon. They ensure that the final state lies within a set that is invariant under the system dynamics and safe to operate in. The choice of terminal constraints is crucial for guaranteeing that the control action will not lead to violations of state constraints in the future.

Plot of terminal invariant set

The role of terminal sets is to ensure stability and recursive feasibility. Terminal sets act as constraints on the final state in the prediction horizon. To refine the terminal sets, iterative methods involving starting with an initial set and repeatedly applying system dynamics and constraints to shrink it to a feasible set were used. The terminal set is tailored to the specific dynamics and constraints of the system, ensuring that the states in the set are reachable and maintainable under the given constraints.

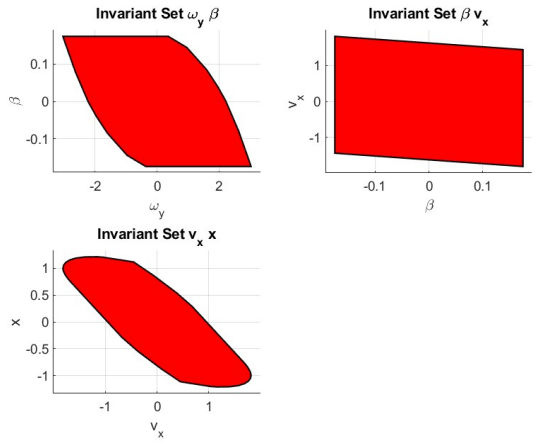


Figure 1: Terminal Invariant Set X

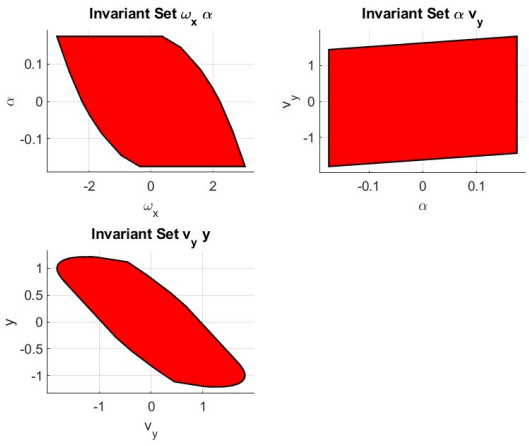


Figure 2: Terminal Invariant Set Y

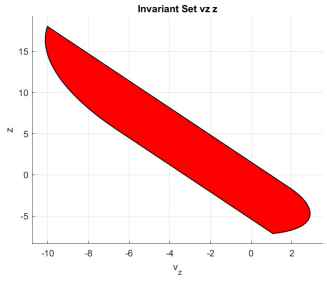


Figure 3: Terminal Invariant Set Z

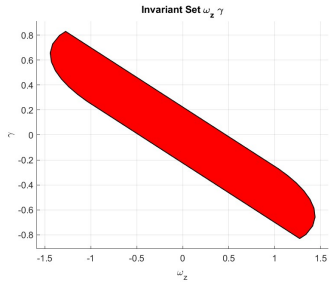


Figure 4: Terminal Invariant Set ROLL

Open-Loop & Closed-Loop plots

The open-loop and closed-loop plots for each dimension of the system starting stationary at 3 meters from the origin (for x, y, and z) or stationary at 30° for roll for each dimension of the system are presented, showcasing the system's performance under both control strategies.

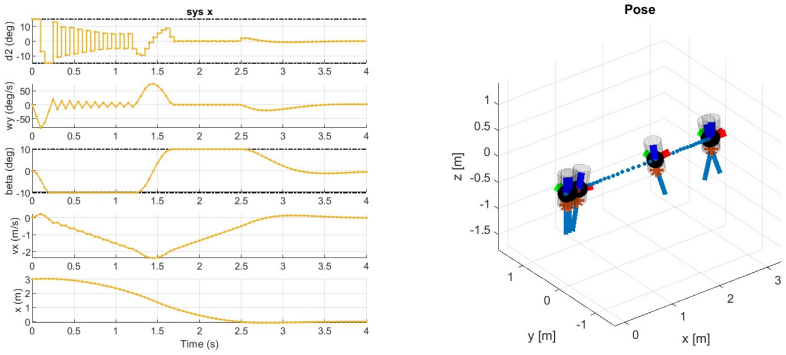


Figure 5: System X Open-Loop

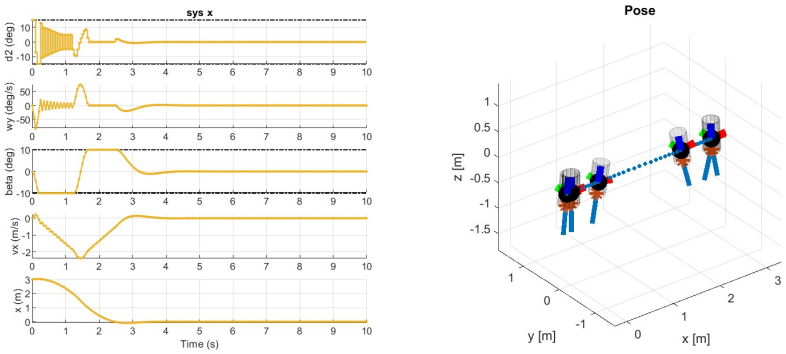


Figure 6: System X Closed-Loop

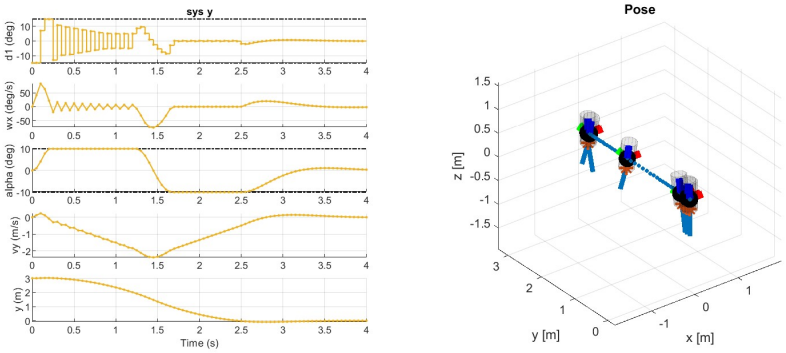


Figure 7: System Y Open-Loop

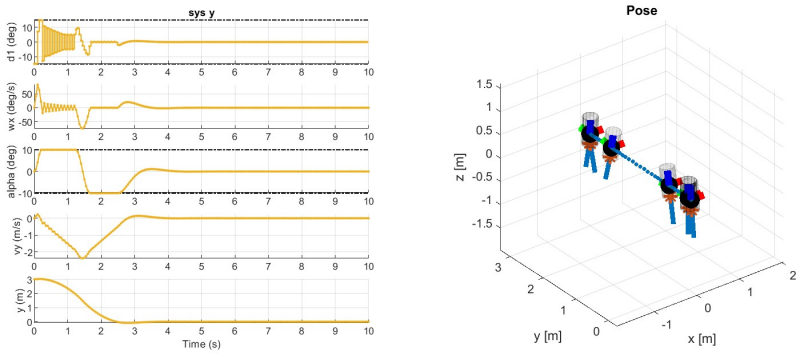


Figure 8: System Y Closed-Loop

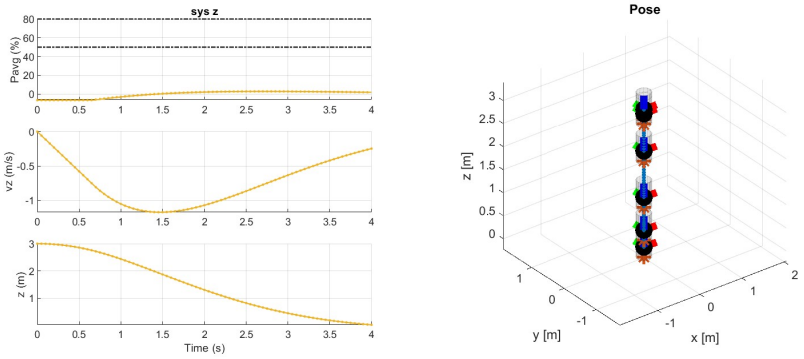


Figure 9: System Z Open-Loop

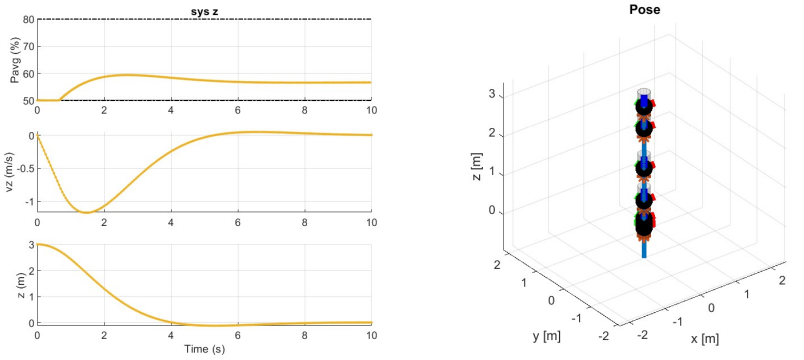


Figure 10: System Z Closed-Loop

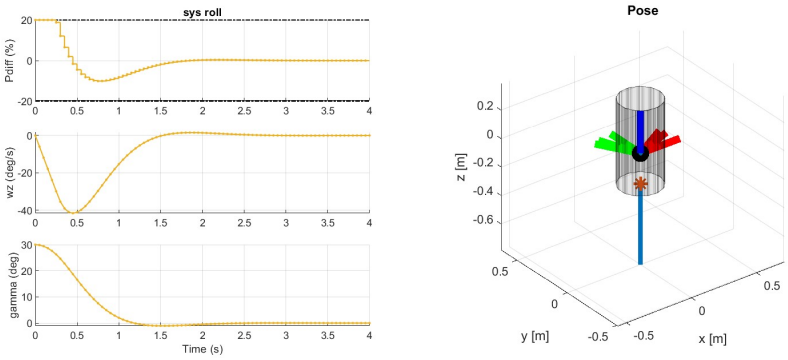


Figure 11: System ROLL Open-Loop

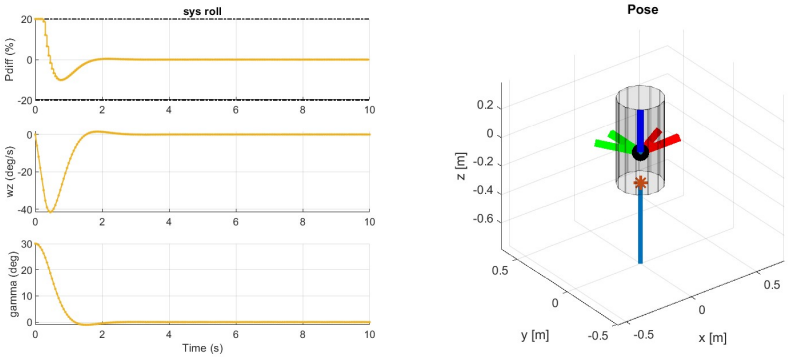


Figure 12: System ROLL Closed-Loop

The following design was implemented in Matlab to ensure recursive constraint satisfaction.

Algorithm 1 Design procedure ensuring recursive constraint satisfaction

Require: System Model
Define cost matrices Q and R
Define state and control constraints matrices F, f, M, and m
Compute LQR gain K and terminal cost Q_f using dlqr
Initialize and compute maximal invariant set X_f
Define constraints *con* and objective *obj*
for each step in prediction horizon **do**
  Update *con* and *obj* with state dynamics and cost
end for
Add terminal constraints to *con*
Add terminal cost to *obj*

Deliverable 3.2

In this section, the task involved extending the controllers for tracking constant references in position for x, y, z, and an angle in radians for roll, without the requirement of an invariant terminal set. It covers the modifications made to the four distinct controllers - X, Y, Z, and ROLL. Included are detailed explanation of the design procedure and tuning parameters, along with open-loop and closed-loop plots for each dimension, illustrating tracking from the origin to -4 meters for x, y, z, and to 35 degrees for roll.

Explanation of design procedure & choice of tuning parameters

The optimization problem and constraints stay the similar to question 3.1 however computation of the terminal sets are removed, and delta formulation for tracking are defined.

$$\Delta x = x - x_s, \quad \Delta u = u - u_s$$

The objective function and constraints of the controller are thus re-expressed as:

$$\text{con} : \begin{cases} \sum_{i=1}^{N-1} X_{i+1} = A \cdot X_i + B \cdot U_i, \\ \sum_{i=2}^{N-1} F \cdot X_i \leq f, \\ \sum_{i=1}^{N-1} M \cdot U_i \leq m, \\ x_N \in X_f \end{cases}$$

$$\text{obj} = \Delta x_N^\top Q_f \Delta x_N + \sum_{i=2}^{N-1} \Delta x_i^\top Q \Delta x_i + \sum_{i=1}^{N-1} \Delta u_i^\top R \Delta u_i$$

To compute a feasible steady-state target for the system, the steady-state state x_s and input u_s that would allow the system to maintain this reference value are computed. A simple objective function subject to constraints ensure the steady-state conditions are met. The computed x_s and u_s are then used as targets for the MPC controller to track. The objective and constraint function used in the computation of steady state x_s and input u_s :

$$\text{con} : \begin{cases} M \cdot u_s \leq m, \\ F \cdot x_s \leq f, \\ x_s = A \cdot x_s + B \cdot u_s, \\ y_{\text{ref}} = C \cdot x_s + D \cdot u_s \end{cases} \qquad \text{obj} = \quad u_s^T u_s$$

The choice of parameters Q, R, H stays the same as for question 3.1.

Open-Loop and Closed-Loop plots

The plots illustrate the system's response in open-loop and closed-loop modes for each spatial dimension as it moves from the starting point to a set target: specifically, transitioning from the origin to a position 4 meters in the negative direction for the x, y, and z axes, and rotating to achieve a 35-degree angle for roll.

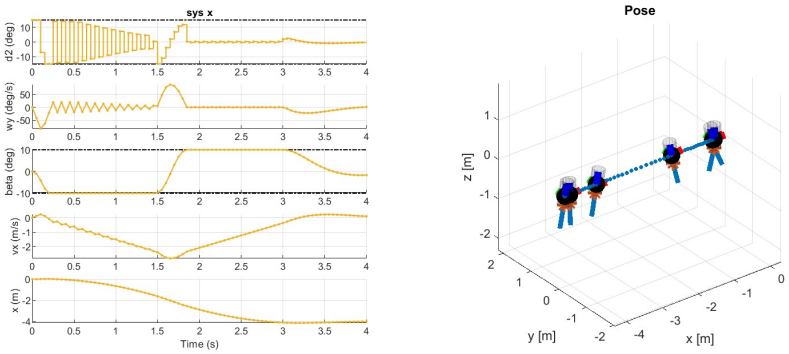


Figure 13: System X Open-Loop

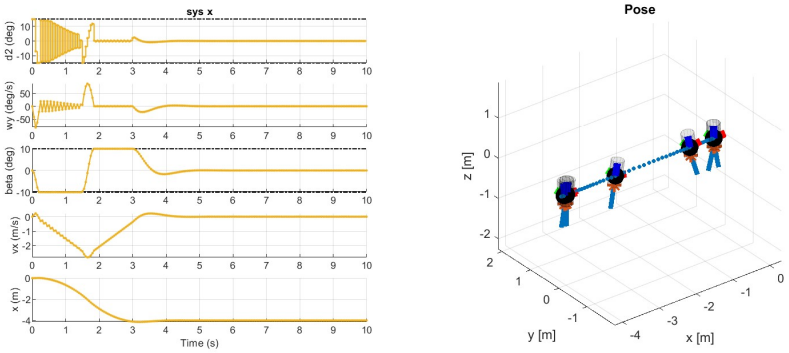


Figure 14: System X Closed-Loop

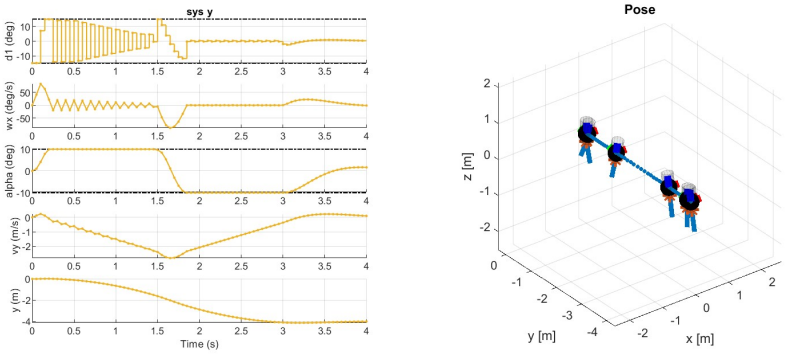


Figure 15: System Y Open-Loop

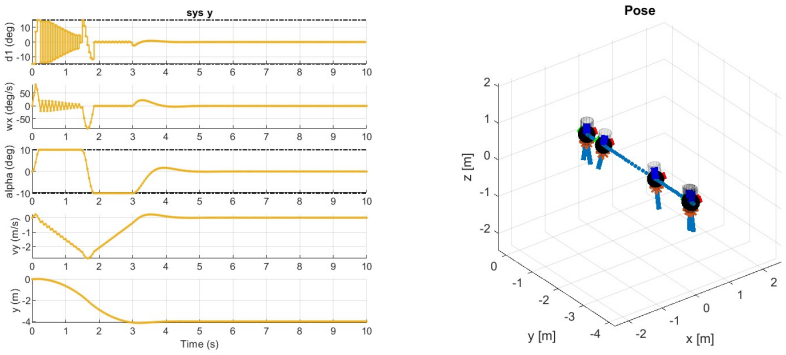


Figure 16: System Y Closed-Loop

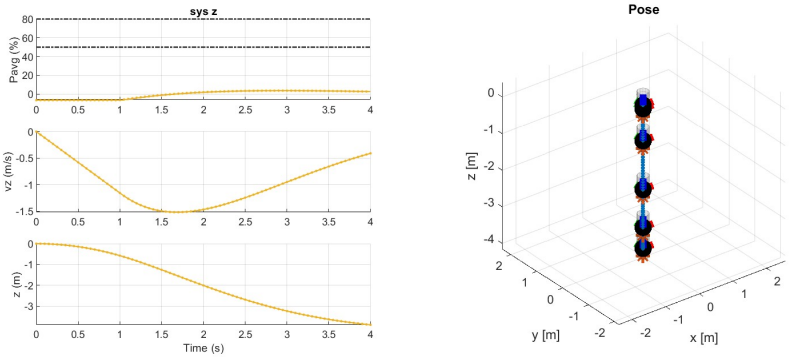


Figure 17: System Z Open-Loop

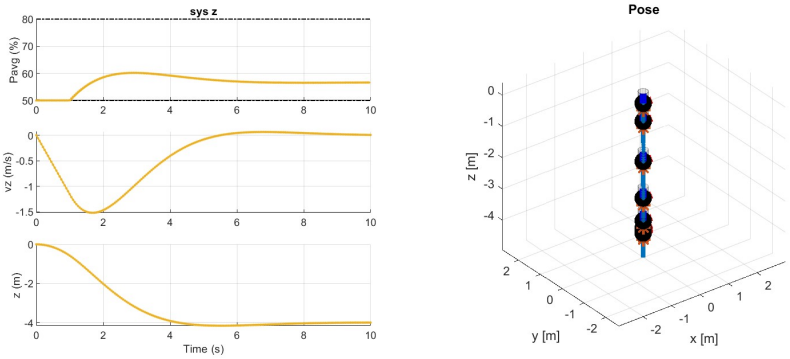


Figure 18: System Z Closed-Loop

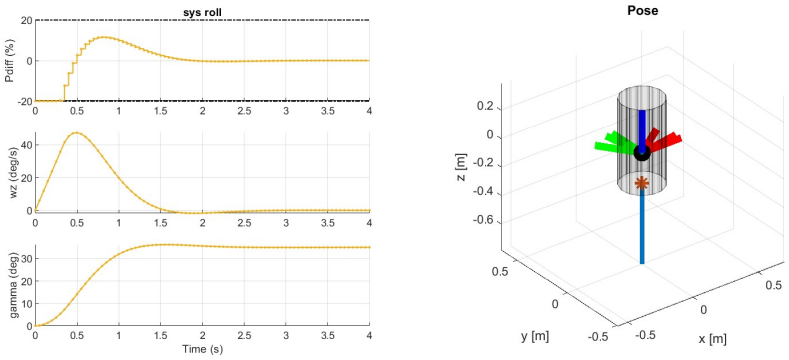


Figure 19: System ROLL Open-Loop

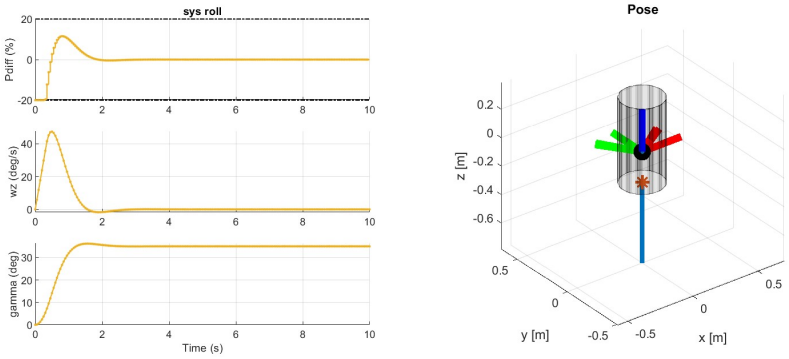


Figure 20: System ROLL Closed-Loop

Deliverable 4.1

This section presents graphs of the controllers successfully tracking the reference path and reference roll angle.

To facilitate controller tuning, rather than relying on trial and error to find robust tuning weights for any potential model mismatches, soft state constraints were introduced using slack variables. This approach helps maintain the feasibility of the problem even under model mismatch.

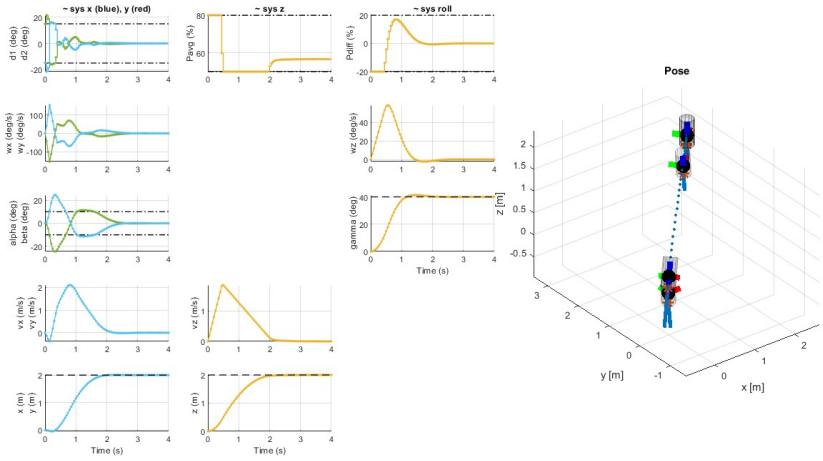


Figure 21: Optimal Open-Loop Trajectory

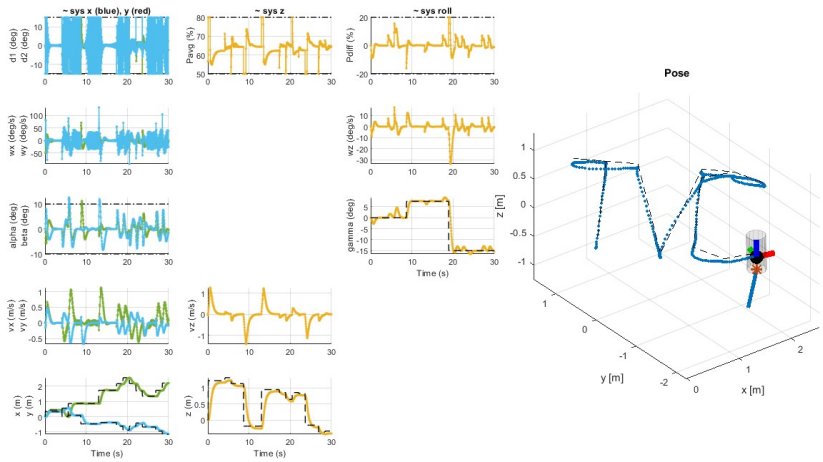


Figure 22: Controllers Tracking Reference Path

To accomplish the task of writing the letters "TVC," the matrices Q and R were adjusted. The R matrix values were lowered to facilitate more intense control actions. The weights in the Q matrices were significantly increased, particularly for the tracking variables x, y, z, and roll. This adjustment prioritizes these specific states, ensuring precise control and adherence to the desired trajectory.

$$Q_x = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix} \quad Q_y = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix} \quad Q_z = \begin{bmatrix} 50 & 0 \\ 0 & 200 \end{bmatrix} \quad Q_{\text{roll}} = \begin{bmatrix} 1 & 0 \\ 0 & 200 \end{bmatrix}$$

$$R_x = 0.01 \quad R_y = 0.01 \quad R_z = 0.01 \quad R_{\text{roll}} = 0.01$$

Deliverable 5.1

In this deliverable, the goal was to implement offset-free tracking for the z-direction only. The model needs to be augmented to include a constant disturbance.

$$\begin{aligned} x_{k+1} &= Ax_k + Bx_k + B_d d_k \\ d_{k+1} &= d_k \\ y_k &= Cx_k C_d d_k \end{aligned}$$

This disturbance now also needs to be estimated using an estimator:

$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{d}_{k+1} \end{bmatrix} = \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} \hat{x}_k \\ \hat{d}_k \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k + \begin{bmatrix} L_x \\ L_d \end{bmatrix} \left(C\hat{x}_k + C_d \hat{d}_k - y_k \right) \tag{2}$$

where $\hat{x}$ and $\hat{d}$ are the estimate of the state and disturbance.
We can get the error dynamics from eq. 3:

$$\begin{bmatrix} x_{k+1} - \hat{x}_{k+1} \\ d_{k+1} - \hat{d}_{k+1} \end{bmatrix} = \left(\begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} + \begin{bmatrix} L_x \\ L_d \end{bmatrix} + \begin{bmatrix} C & C_d \end{bmatrix} \right) \begin{bmatrix} x_k - \hat{x}_k \\ d_k - \hat{d}_k \end{bmatrix} \tag{3}$$

The estimator L is computed using the Matlab function `place()` with $\tilde{A} = \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix}$ and $\tilde{C} = \begin{bmatrix} C & C_d \end{bmatrix}$, with $B_d = B$ and $C_d = 0$.

The estimator is set up in the `setup_estimator`. This estimator needs to be tuned with some poles. Poles with a smaller norm have a faster estimation process but an increased overshoot. We first tried with the poles from exercise 5 and tuned them down a bit to have a fast estimation (important for a rocket) while not having too much of an overshoot. The weights chosen for the poles were:

$$p = [0.2, 0.3, 0.4]$$

Note: The poles were chosen to be on the unit circle to generate a stable system.

The simulation setup is to fly from an x_0 where $\begin{bmatrix} x & y & z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 3 \end{bmatrix}$ to a target reference: $\begin{bmatrix} x_{ref} & y_{ref} & z_{ref} \end{bmatrix} = \begin{bmatrix} 1.2 & 0 & 3 \end{bmatrix}$.

We can see in Figure 23 the impact of changing the mass (mass = 2.13kg) on our original controller from Part 4. We see on the Z-axis that the rocket shifts down a bit from the supposed 3m to around 2.85m.

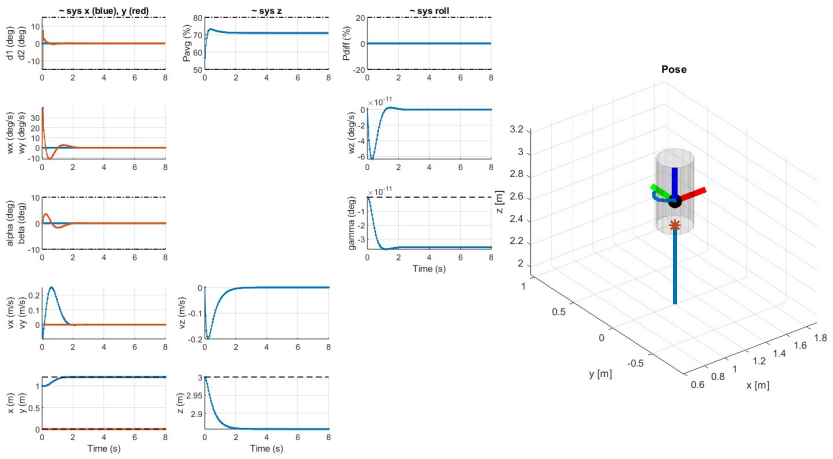


Figure 23: Controller from Part 4, without z estimator with modified of mass = 2.13kg

After implementing our z estimator (see Figure 24), we can see that even though there is an overshoot at the beginning of the simulation (Z-axis), the rocket manages to track back to 3m after 2 seconds.

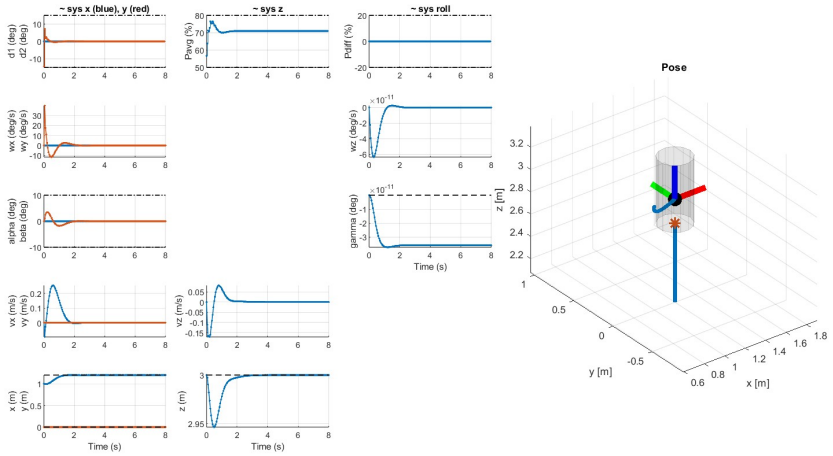


Figure 24: Controller with z estimator with modified of mass = 2.13kg

Deliverable 5.2

For this part, we want to study how our controller reacts to having a mass decrease that is related to thrust. In this setup, it is assumed that half of the rocket's initial mass is fuel and it will be reduced when consumed by the motor. Using `rocket.mass` = 2.13 and `rocket.mass_rate` = −0.27 and the same simulation setup as in 5.1.

We can see the results in Figure 25.

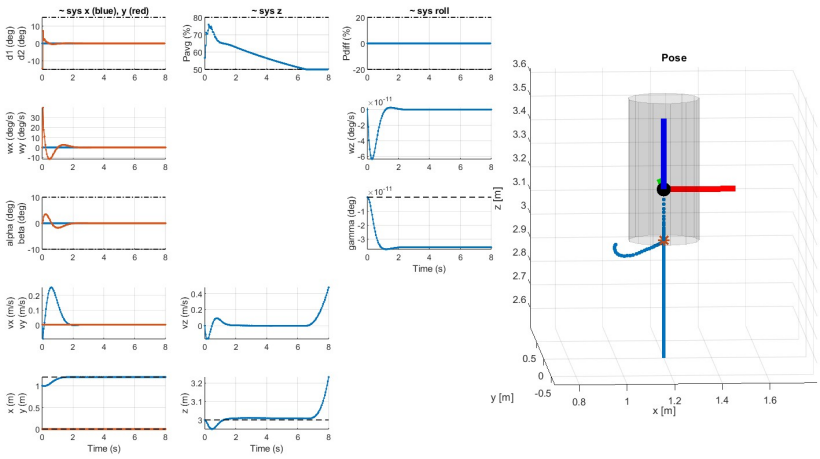


Figure 25: Controller with z estimator with decreasing mass.
$\texttt{mass\_rate} = -0.27$, $T_f = 8$s

In the first seconds of the simulation, we can see a tracking offset in the z-direction. This is because the disturbance estimator only estimates the mass change alone and does not relate it to the thrust variation. This is why, as the controller tries to reduce the thrust to account for the mass diminution, the mass continues to decrease due to fuel consumption and the controller continues to try to reduce and so on.

To account for the mass reduction, we would need to modify the estimator to have a dynamic disturbance estimation, that integrates a relation on the mass_rate.

We can see in Figure 25 that between 0 and 1.3 seconds the controller reacts to the first change in mass (mass = 2.13kg) by increasing the thrust (P_{avg}). As explained before, a new positive offset appears due to an over-correction of the controller that cannot go back to the reference because it does not model the changing mass due to fuel consumption. We can see that it tries to reduce the thrust (P_{avg}) as the mass decreases over time.

Towards the end of the simulation, we run into an unexpected behavior around 6.5 seconds. At this point, the minimum average value we can give to the propellers (P_{avg}) reaches the limit imposed by our constraint ($P_{avg} \geq 50\%$). At this point the mass of the rocket continues to decrease but the P_{avg} cannot follow due to the constraint. This causes the rocket to fly up quickly.

We simulate then for a bit longer $T_f = 20$s. The results are plotted in Figure 26.

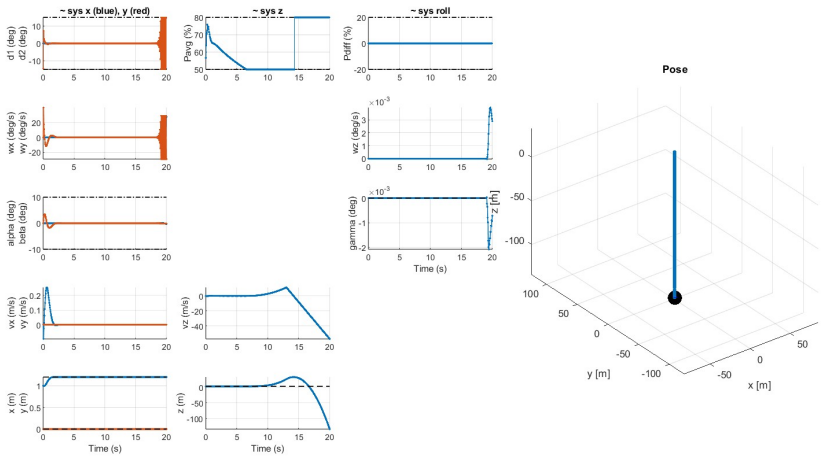


Figure 26: Controller with z estimator with decreasing mass.
`mass_rate` $= -0.27$, $T_f = 20$s

We can observe two more interesting events, From 10s until around 14s, the rocket is too light for a $P_{avf} = 50\%$, which is why it rises. After 14s, the rocket reached the point where it ran out of fuel. This causes the rocket to dive rapidly. We see the consequence of running out of fuel in the P_{avg} graph, as it tries to overcorrect by saturating to the maximum of the constraint.

Deliverable 6.1

In this section, our aim is to formulate a nonlinear MPC controller for the rocket system. The controller will take the complete state as input and generate four control commands. We will compare the performance of the NMPC controller with the MPC controller of section 4 on a fixed reference (TVC) with a maximum roll angle of $|\gamma_{ref}| = 15^o$ and then with a maximum roll angle of $|\gamma_{ref}| = 50^o$, the latter being more difficult for the MPC controller.

Controller architecture

Since the NMPC controller operates on the nonlinear model we can have less constraints on our system and thus cover the whole state space. Our only constraint on the state space is $|\beta| \leq 75^o$ to avoid the singularity at $\beta = 90^o$. For the action space we have the following constraints (see Table. 1).

The nonlinear optimization to solve is:

$\delta_{1,2} \leq 15^o$
$20\% \leq P_{avg} \leq 80\%$
$-20\% \leq P_{diff} \leq 20\%$

Table 1: Action space

$$
\begin{aligned}
J*(x) = \min_{x,u} \quad & \sum_{i=0}^{N-1} (x_i - x_{ref})^\top Q (x_i - x_{ref}) + u_i^\top R u_i \\
\text{subject to} \quad & x_{i+1} = f_{discrete}(x_i, u_i), \\
& Fx_i \leq f, \\
& Mu_i \leq m, \\
& x_0 = x.
\end{aligned} \tag{4}
$$

We utilize the Runge-Kutta 4 method to discretize the system with a sampling period of $h = rocket.Ts$.

As for the MPC , selecting Q and R matrices balances tracking performance and control effort. The Q matrix emphasizes maintaining specific states, penalizing deviations more severely for closer adherence. This balance avoids excessive control actions. The R matrix regulates control intensity, discouraging aggressiveness to prevent overly assertive system behavior. The process, often trial and error, aims for a balanced approach meeting system performance and stability requirements.

$$
Q = \begin{bmatrix}
2 & & & & & & & & & & & \\
& 2 & & & & & & & & & & \\
& & 1 & & & & & & & & & \\
& & & 1 & & & & & & & & \\
& & & & 1 & & & & & & & \\
& & & & & 10 & & & & & & \\
& & & & & & 1 & & & & & \\
& & & & & & & 1 & & & & \\
& & & & & & & & 2 & & & \\
& & & & & & & & & 10 & & \\
& & & & & & & & & & 10 & \\
& & & & & & & & & & & 10
\end{bmatrix}
\quad
R = \begin{bmatrix}
0.001 & & & \\
& 0.001 & & \\
& & 0.001 & \\
& & & 0.001
\end{bmatrix}
$$

Pros and cons of NMPC over MPC

A nonlinear controller offers many advantages since it can describe almost any model. For instance, our NMPC controller covers nearly the entire state space, giving it a larger feasibility set than the MPC controller. Moreover, the nonlinear model performs better for

extreme states, as shown in Fig 28 and Fig 29, where we see that the MPC controller of section 4 struggles with $|\gamma_{ref}| = 50^o$ whereas the NMPC controller has no problem with it. However, the nonlinear controller takes significantly more time to find a solution due to its higher computational demands. To achieve a competitive resolution time compared to MPC, we must either shorten the horizon or reduce the sampling rate, coming at the expense of its performance. We chose to reduce the time horizon to make a more fair comparison between our MPC and NMPC controller having H = 4 for the former and H = 2 for the latter.

Results

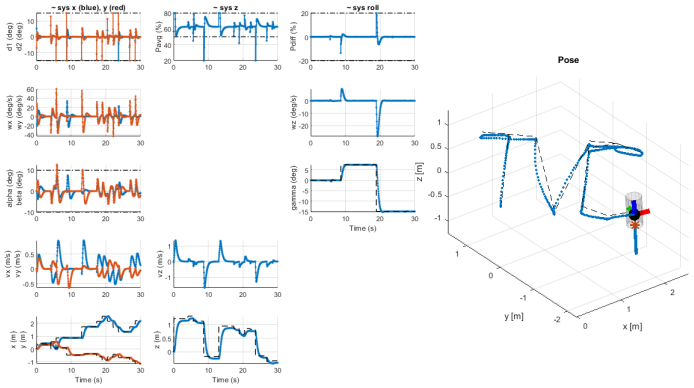


Figure 27: NMPC with $|\gamma_{ref}| = 15^o$ and H = 2

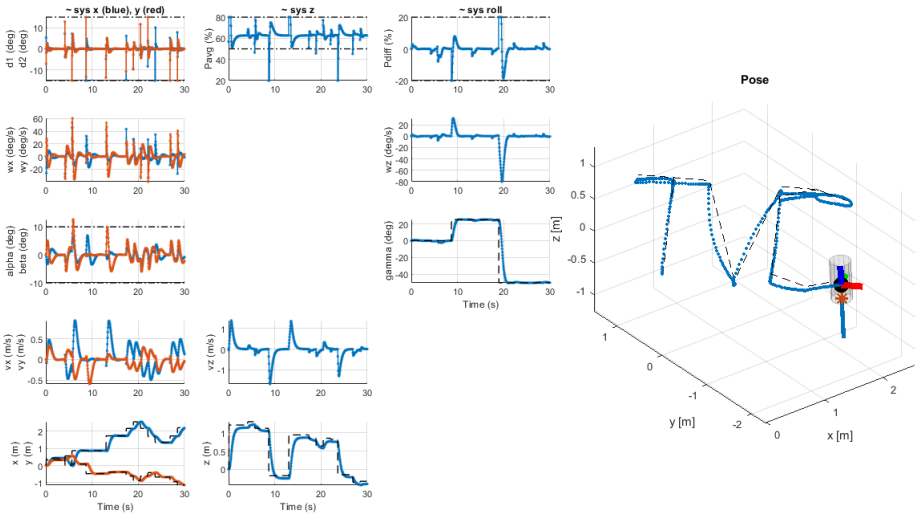


Figure 28: NMPC with $|\gamma_{ref}| = 50^o$ and H = 2

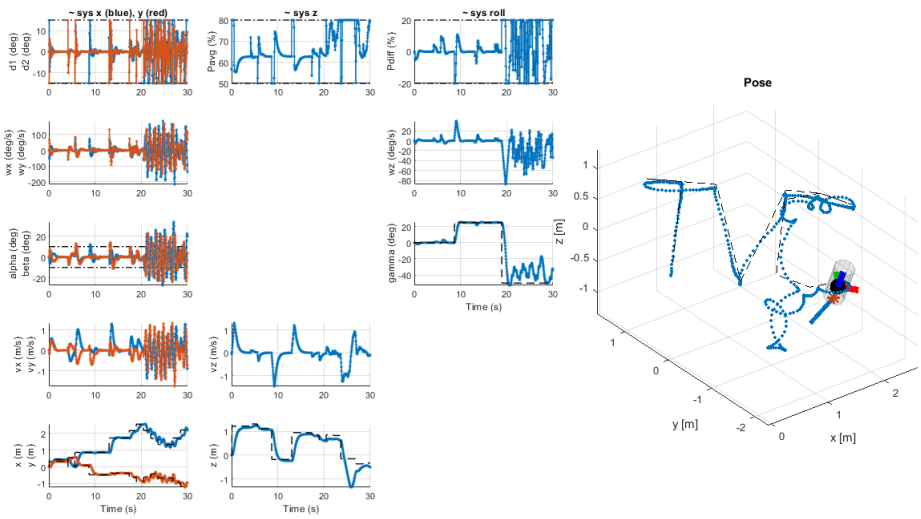


Figure 29: MPC with $|\gamma_{ref}| = 50^o$ and H = 4

Deliverable 6.2

In this section, we will add delay compensation to our NMPC controller since being more computationally demanding it can in practice lead to delays. We implemented a simple countermeasure where the controller solves the optimization for k delay steps later as seen in Fig 30.

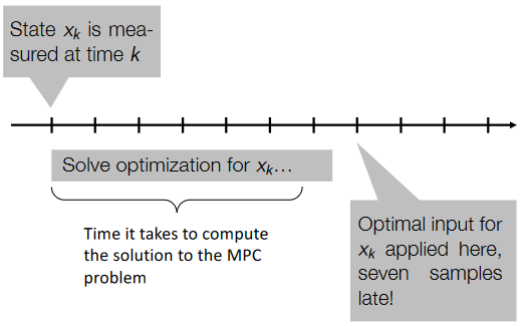


Figure 30: source: project description

Implementation

To implement this strategy we use a buffer to keep track of the last k inputs given to infer the x_{t+k} state. We initialize the buffer to k*[0,0,62,0] to keep the rocket in a hover position while waiting for the first input of the controller. To predict the state x_{t+k} we use a simple Euler integration scheme and the memory of the k last inputs.

Results

First of all we can see in Fig 31 what happens without delay compensation, the rocket is completely unstable and doesn't track the desired point.
With fully delay-compensation, the simple strategy we implemented works well for a delay of 5 as we can see in Fig 32 . For a delay of 10 we already see a drop of performance (see Fig 33), the controller still manages to track the desired point but takes longer and has a more erratic path. And with a delay of 15 the rocket completely fails. This method also works well on the previous task (TVC) with a delay of 5 or even higher (see Fig 35). The same task without the delay compensation gives us catastrophic results.
With a delay compensation smaller than the real delay, this strategy doesn't work as well. With a delay of 5 and a delay compensation of 3 (see Fig 34) we observe a drop of performance and a more oscillating behaviour.

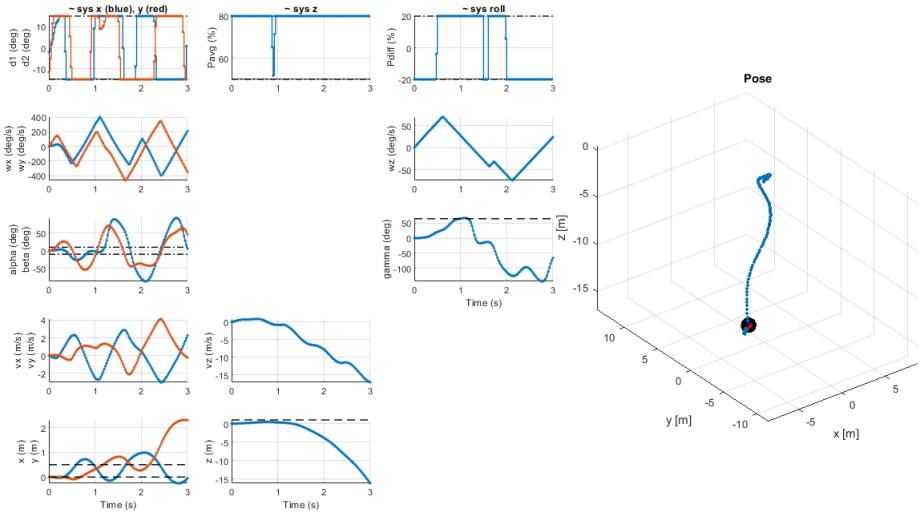


Figure 31: source: NMPC with a delay of 5 with 0 compensation

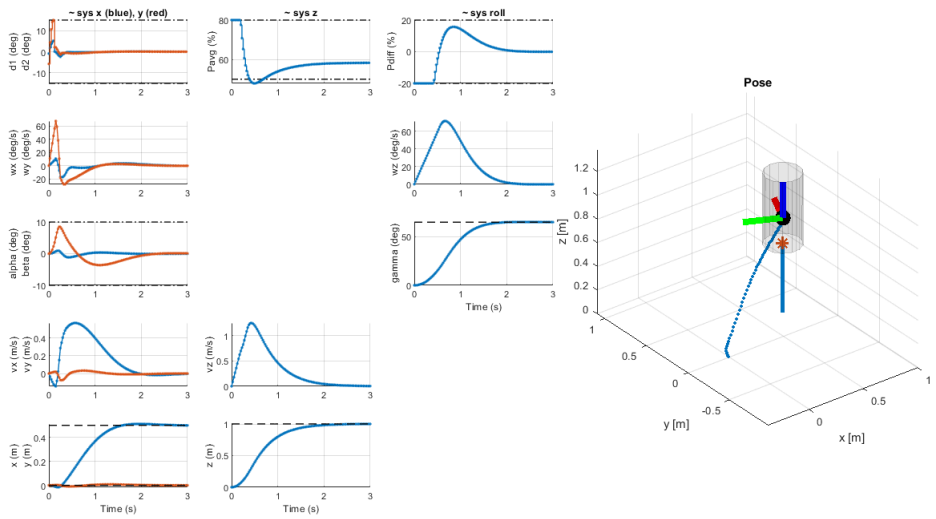


Figure 32: source: NMPC with a delay of 5 fully compensated

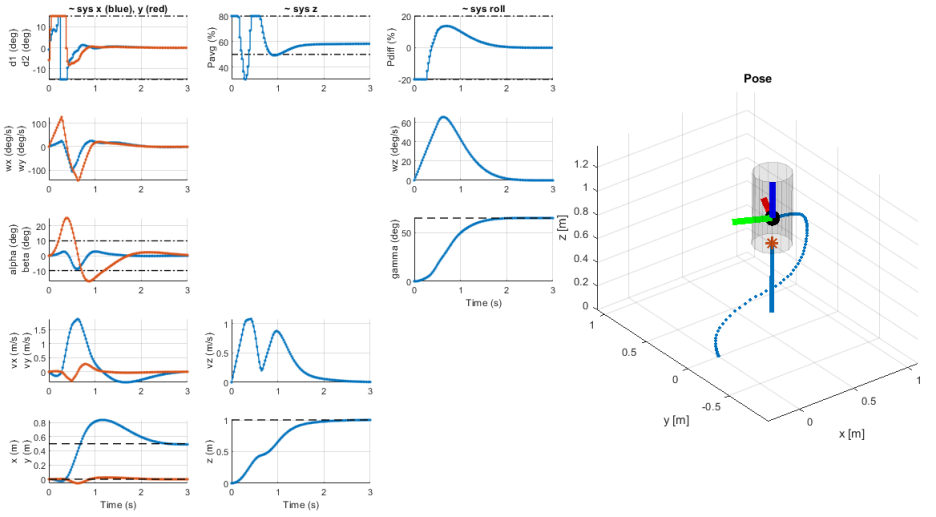


Figure 33: source: NMPC with a delay of 10 fully compensated

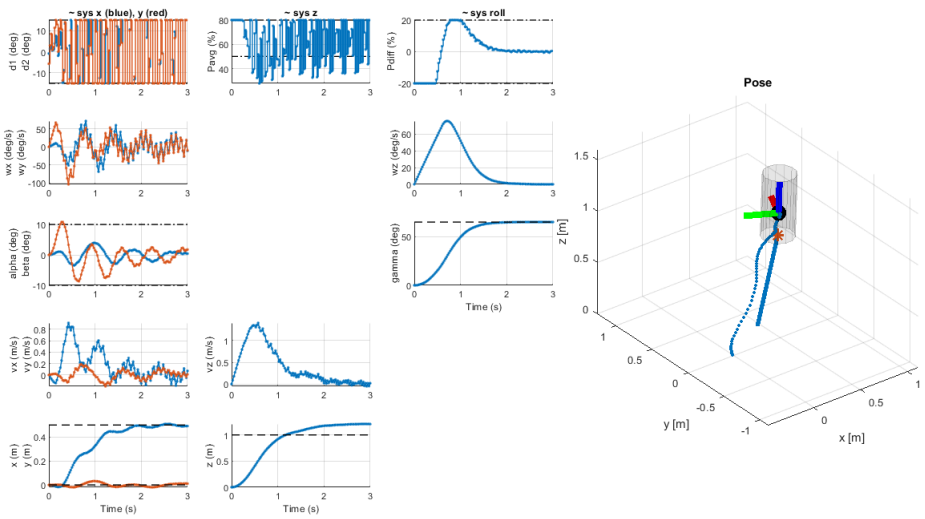


Figure 34: source: NMPC with a delay of 5 with a compensation of 3

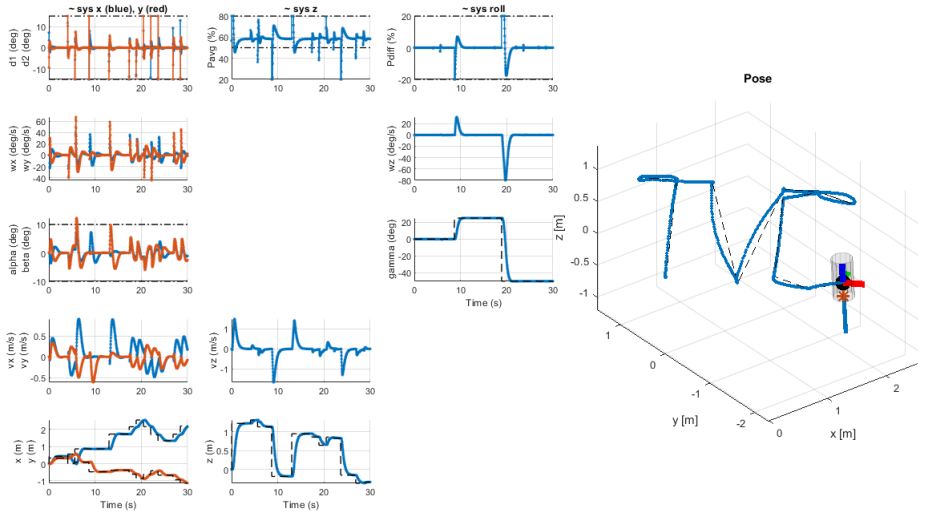


Figure 35: source: NMPC on TCV track with delay 5