Ecole Polytechnique
Fédérale de Lausanne

ME-425: Model Predictive Control

Mini-Project
MPC controller for a Rocket Prototype

Zahra Taghizadeh - 369116
Giada Ehrlich -370768
Andrea Grillo - 371099

Group I

Professor
Colin Jones

Fall 2023

Contents

1 Introduction

The objective of this project for the Model Predictive Control course is to implement an MPC controller for a rocket prototype, evaluating different methods and techniques to achieve different goals. The project has been developed in the Matlab environment, starting from a given physical model of the rocket and a code template for running the simulations and plotting the results.

First, we simulated the rocket dynamics using arbitrary inputs, to better understand how the system behaves. Then we linearized the given system and divided it into 4 independent subsystems. We have developed MPC regulators for each of the subsystems and then improved them to track a reference. The 4 MPC controllers allowed us to track a complex reference trajectory and evaluate the performance of the linear controllers applied to a nonlinear system. To ensure the feasibility of the MPC problems in this case, we softened the constraints. We have then implemented an offset-free tracking controller, able to track the reference despite a model mismatch, that will consist of a different rocket mass and a time-varying mass. Finally, we have designed a Nonlinear MPC controller for the rocket, evaluated its performance and compared it to the linear ones. We have also evaluated the effects of time delay on the stability of the closed-loop system.

2 Deliverable 2: Linearization and subsystem separation

2.1 Description of the system

The rocket is a complex system which can move in 3D space with 6 degrees of freedom.
Onboard the prototype there are 4 actuators: two motors with propellers to provide the thrust, and two servo motors to control the deflection of the propellers.
The nonlinear model describing the rocket dynamics has 12 states and 4 inputs, shown below:

$$\vec{x} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ \alpha \\ \beta \\ \gamma \\ v_x \\ v_y \\ v_z \\ x \\ y \\ z \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ P_{avg} \\ P_{diff} \end{bmatrix}$$

2.2 Linearization

We first found the trim point, which is the steady-state equilibrium point obtained by setting the derivative of the state to 0. This is done by using the Rocket class *trim* function, which returns the *xs* and *us* variables, respectively trim state and input.

Below is the obtained state and input:

$$\vec{x}_s = \begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\\ \alpha\\ \beta\\ \gamma\\ v_x\\ v_y\\ v_z\\ x\\ y\\ z\end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \qquad \vec{u}_s = \begin{bmatrix}\delta_1\\ \delta_2\\ P_{avg}\\ P_{diff}\end{bmatrix} = \begin{bmatrix}0\\0\\56.6667\\0\end{bmatrix}$$

As we can see from the trim point, the rocket is stationary in a vertical position. The inputs are all null, except for the P_{avg} value which is non-zero as the rocket needs to stay propelled vertically to oppose gravity.
Then we used the *linearize* function to get a linear system, obtained by linearizing the original dynamics of the rocket around the previously computed steady state point.
The state-space model of the obtained linear system is the following:

$$\dot{\vec{x}} = \begin{bmatrix}0&0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0&0\\1&0&0&0&0&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0&0&0&0&0\\0&0&1&0&0&0&0&0&0&0&0&0\\0&0&0&0&9.81&0&0&0&0&0&0&0\\0&0&0&-9.81&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&1&0&0&0&0&0\\0&0&0&0&0&0&0&1&0&0&0&0\\0&0&0&0&0&0&0&0&1&0&0&0\end{bmatrix}\vec{x} + \begin{bmatrix}-55.68&0&0&0\\0&-55.68&0&0\\0&0&0&-0.104\\0&0&0&0\\0&0&0&0\\0&0&0&0\\0&9.81&0&0\\-9.81&0&0&0\\0&0&0.1731&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\vec{u}$$

As the linearization is done around a vertical trim, and the rocket has a highly non-linear behaviour, to ensure that the linear controller obtained works, the system will have to be constrained to not deviate too much from a vertical position.

2.3 Separation of subsystems

The next task consisted of the separation of the linearized system into 4 different subsystems. This is done using the *decomopose* method of the rocket class.
In the matrices shown above it is easy to see that each input has influence only on a limited set of states and that a lot of states are decoupled and do not influence each other.

- **Subsystem X** - The δ_2 input represents the rotation of the propeller, which causes a variation in the ω_y rotational speed and v_x linear speed, which consequently causes a variation in the angle β and of the position along the x axis.

$$\dot{\vec{x}}_x = \begin{bmatrix}0&0&0&0\\1&0&0&0\\0&9.81&0&0\\0&0&1&0\end{bmatrix} \cdot \begin{bmatrix}\omega_y\\ \beta\\ v_x\\ x\end{bmatrix} + \begin{bmatrix}-55.68\\0\\9.81\\0\end{bmatrix} \cdot \delta_2$$

- **Subsystem Y** - analogous to the X one.

$$\dot{\vec{x}}_y = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -9.81 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \omega_x \\ \alpha \\ v_y \\ y \end{bmatrix} + \begin{bmatrix} -55.68 \\ 0 \\ -9.81 \\ 0 \end{bmatrix} \cdot \delta_1$$

- **Subsystem Z** - The P_{avg} input only influences the speed along the axis z(v_z), which in turn causes a variation of the position along the same axis.

$$\dot{\vec{x}}_z = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_z \\ z \end{bmatrix} + \begin{bmatrix} 0.1731 \\ 0 \end{bmatrix} \cdot P_{avg}$$

- **Subsystem Roll** - The roll angle(γ), represents the rocket's rotation about the z-axis. This angle is regulated by the input P_{diff}, which induces changes in the angular velocity(ω_z) and consequently alters the roll angle.

$$\dot{\vec{x}}_\gamma = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \omega_z \\ \gamma \end{bmatrix} + \begin{bmatrix} -0.104 \\ 0 \end{bmatrix} \cdot P_{diff}$$

One important thing to note is that this complete decoupling of the subsystems is not present in the original nonlinear system.
For example, the δ inputs generate a variation of the z position, because the thrust generated by the propellers is not vertical anymore.
This separation is only valid when the states are close to the trim point.

3 Deliverable 3: MPC controllers for the subsystems

In this part, the goal is to create 4 different MPC controllers for the 4 subsystems. We started by designing simple regulators aimed at tracking a zero state value and then improved them to track a reference value.

3.1 MPC regulators

3.1.1 Ensuring recursive feasibility

The required properties of the regulators are the following:

- **Ensuring recursive feasibility and constraint satisfaction**
- **Regulation:** stabilization of the system to the origin
- **Performance requirements:** settling time for given initial state

To address the computational limitations of implementing an infinite horizon in the MPC problem, we established a terminal set to guarantee recursive feasibility. This terminal set is based on the invariant set of an unconstrained LQR(Linear Quadratic Regulator) controller. By ensuring that the final state of the predicted trajectory falls within this LQR invariant set, we can confidently assert the existence of at least one feasible control input – specifically, the one provided by the LQR, that can effectively manage the system.

The algorithm used to compute the terminal set employs the closed-loop matrix $A+BK$, where K represents the gain matrix of the LQR controller. This approach effectively renders the system autonomous by leveraging the closed-loop dynamics. The details of the algorithm are as follows:

```
$\Omega_0 \leftarrow \mathbb{X}$
loop
    $\Omega_{i+1} \leftarrow \mathrm{pre}(\Omega_i) \cap \Omega_i$
    if $\Omega_{i+1} = \Omega_i$ then
        return $\mathcal{C}_\infty = \Omega_i$
    end if
end loop
```

Figure 1: Terminal set algorithm

The LQR controller was also used to define the terminal cost, associated with the last iteration of the finite horizon problem. The terminal cost is defined as:

$$V_f = x_N^T Q_f x_N$$

Where Q_f is the solution of the Discrete Algebraic Riccati Equation ($DARE$).

The resulting formulation of the MPC problem is the following:

$$J^*(x) = \min_{x,u} \sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T Q_f x_N$$

$$\begin{aligned} s.t. \quad & x_{i+1} = Ax_i + Bu_i \\ & Fx_i \le f \\ & Mu_i \le m \\ & x_N \in X_f \end{aligned}$$

Where A,B are the matrices of the model dynamics, and F, f and M, m represent respectively the state and input constraints. In the table below the enforced constraints are summarized :

Subsystem	State	Input
x controller	$\lvert\beta\rvert \le 15°$	$\lvert\delta_2\rvert < 15°$
y controller	$\lvert\alpha\rvert \le 15°$	$\lvert\delta_1\rvert < 15°$
z controller	-	$50\% \le P_{avg} \le 80\%$
γ controller	-	$\lvert P_{diff}\rvert \le 20\%$

Table 1: State and input constraints for each subsystem

3.1.2 Choice of tuning parameters

- **H - Horizon length:** The horizon length has an influence on the region of attraction and the speed of convergence. A longer horizon offers benefits, but it also comes with an increase in the number of computational steps. If the horizon is too short, the controller cannot generate a trajectory to reach the terminal set in the horizon timespan, so the problem becomes unfeasible. The horizon length has been set to $1s$, which at $1/20s$ sampling time means that a 20-step trajectory will be generated each time.

- **Q, R - Cost matrices:** The Q and R cost matrices have been used to generate the cost function for the MPC problem and to generate the terminal set and cost for the unconstrained LQR controller. So they influence both the MPC-generated trajectory and in the terminal set and cost. The tuning has been divided into two steps. The first step consisted of tuning the relative weight of Q, and R to ensure that the cost of being in a certain state and the cost of applying a certain input are well balanced. Having a bigger weight on R than on Q means that the controller will be less aggressive, while the opposite situation leads to a very fast response of the system, but it also may lead to undesired effects like overshoot and oscillation around the setpoint. In this first part, we have found no need to do a more fine-grained tuning on the Q matrix, by setting different weights to the individual state variables.

The chosen parameters are the following:

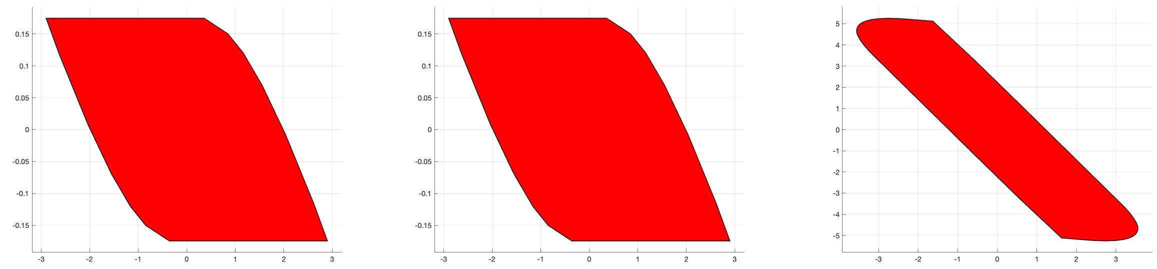
Subsystem	Q	R
x controller	Q(1,1) = 10, Q(2,2) = 10, Q(3, 3) = 10, Q(4,4) = 10	0.1
y controller	Q(1,1) = 10, Q(2,2) = 10, Q(3, 3) = 10, Q(4,4) = 10	0.1
z controller	Q(1,1) = 10, Q(2,2) = 10	1
γ controller	Q(1,1) = 20, Q(2,2) = 20	1

Table 2: Tuning Parameters

3.1.3 Plots 3.1

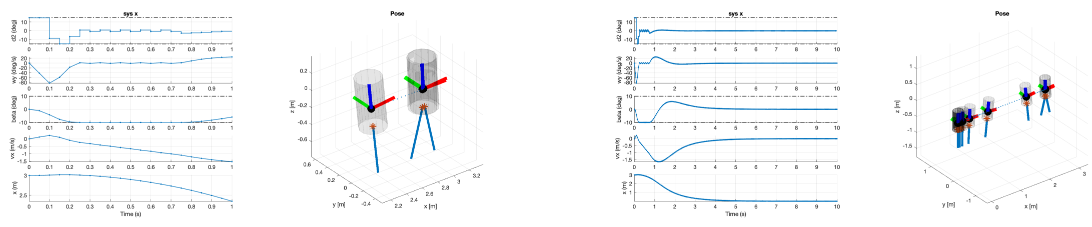
We have simulated the 4 controllers and the results are shown below. We can observe that with the design and tuning, all controllers manage to converge and regulate the state, ensuring recursive feasibility and constraint satisfaction, as well as the satisfaction of the performance objectives that were required.

- **X regulator**



projection on ω_y and β projection on β and v_x projection on v_x and x

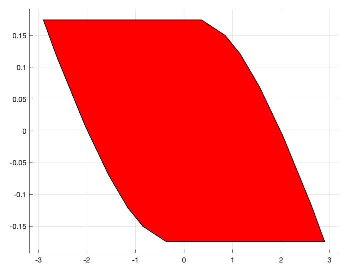
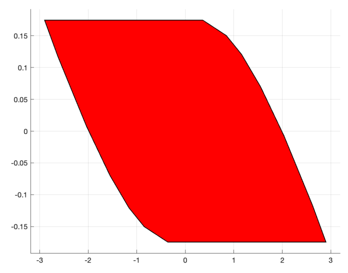
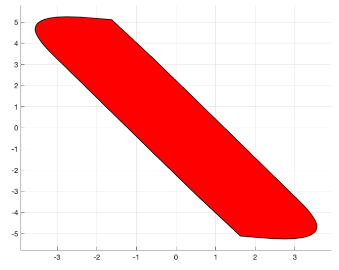
Figure 2: X regulator terminal set projections



Open loop trajectory Closed loop trajectory

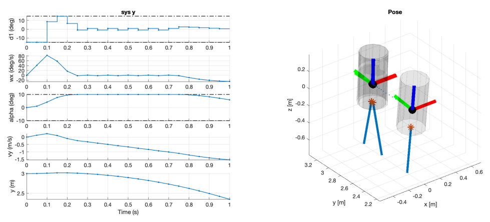
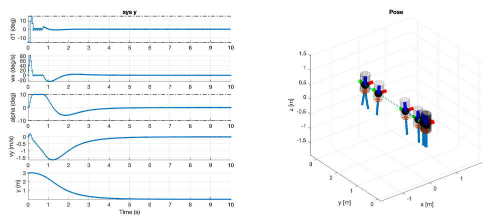
Figure 3: X regulator

- **Y regulator**

projection on ω_x and α projection on α and v_y projection on v_y and y

Figure 4: Y regulator terminal set projections

Open loop trajectory Closed loop trajectory

Figure 5: Y regulator

- **Z regulator**

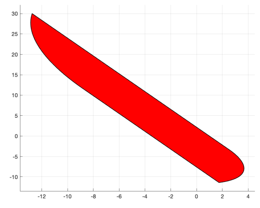
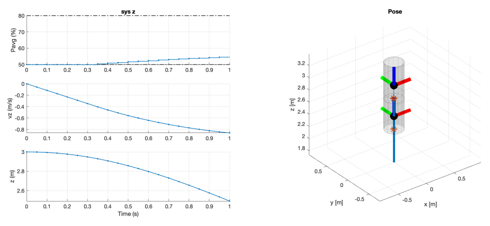
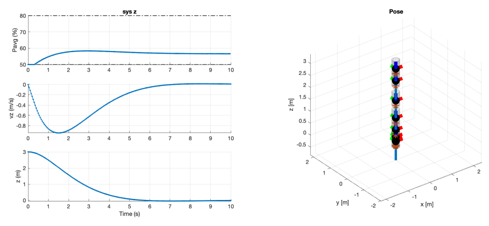


Figure 6: Z regulator terminal set

Open loop trajectory Closed loop trajectory

Figure 7: Z regulator

- **Roll regulator**

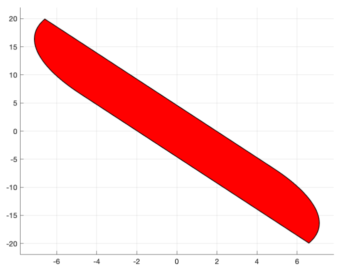
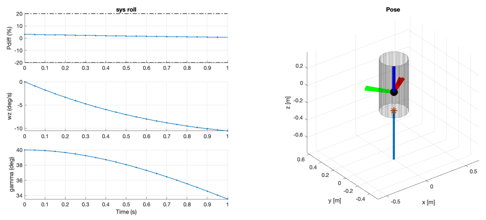
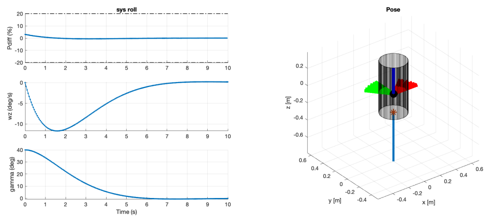


Figure 8: Roll regulator terminal set



Open loop trajectory



Closed loop trajectory

Figure 9: Roll regulator

3.2 MPC reference tracking controllers

In this part, we had to improve our controllers not to be just regulators but to be able to track a reference value.
The first step to achieve this objective is to set up the optimization problem to find the steady-state point (x_s, u_s) associated with the reference value ref that we want to track.
We have used the following formulation for the optimization problem of the steady-state calculation:

$$\min_{u_s} u_s^T R_s u_s$$

$$s.t. \quad \begin{bmatrix} I-A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ ref \end{bmatrix}$$

$$F x_i \leq f$$

$$M u_i \leq m$$

After having computed the steady-state point to be tracked, the formulation of the MPC problem has been adapted to the new *delta* situation:

$$J^*(x) = \min_{x,u} \sum_{i=0}^{N-1} (x_i - x_s)^T Q (x_i - x_s) + (u_i - u_s)^T R (u_i - u_s) + (x_N - x_s)^T Q_f (x_N - x_s)$$

$$s.t. \quad x_{i+1} = A x_i + B u_i$$

$$F x_i \leq f$$

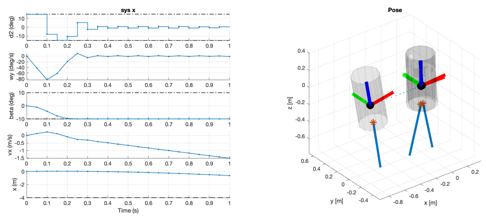
$$M u_i \leq m$$

Note that the terminal set computation and constraint have been dropped.

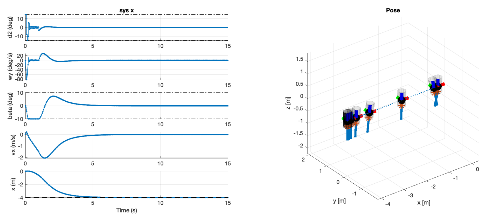
3.2.1 Plots 3.2

Also in this case the results of the simulations for the 4 controllers satisfy the performance requirements. There has been no need to adapt the tuning of the cost matrices, as the tracking is good.

- **X controller**



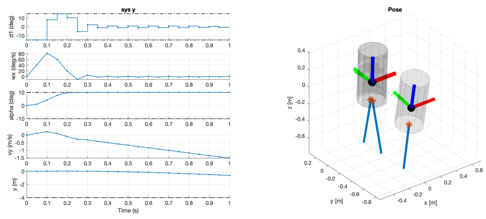
Open loop trajectory



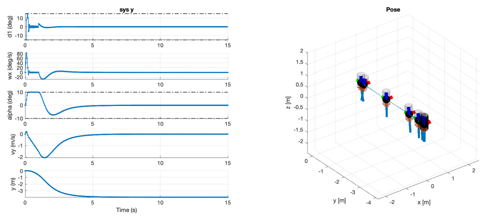
Closed loop trajectory

Figure 10: X controller

- **Y controller**



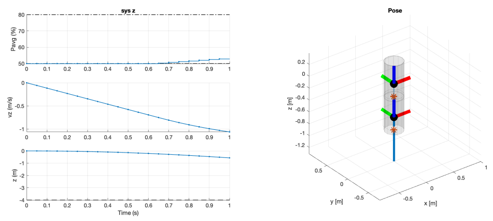
Open loop trajectory



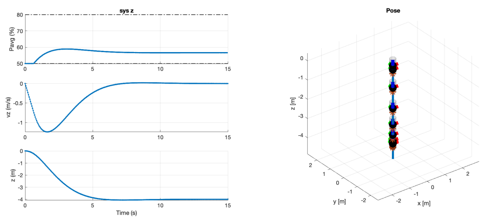
Closed loop trajectory

Figure 11: Y controller

- **Z controller**



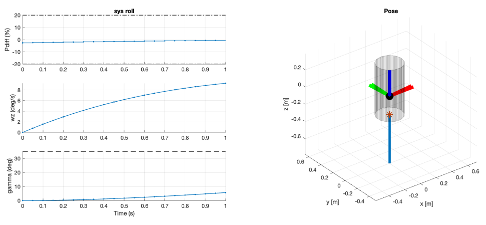
Open loop trajectory



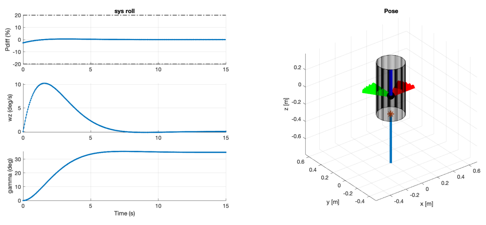
Closed loop trajectory

Figure 12: Z controller

- **Roll controller**



Open loop trajectory



Closed loop trajectory

Figure 13: Roll controller

4 Deliverable 4: Linear MPC applied to nonlinear model

In this report section, we assess the efficacy of four controllers through simulations in a nonlinear system setting. Starting from the initial state at the origin, our goal is to evaluate each controller's performance and its real-world applicability. A critical aspect of our analysis is addressing the model mismatches encountered when applying linear controllers to a nonlinear framework. To address this, we first closely analyzed the trajectory plots to identify the specific states leading to such violations and tuned the parameters to prevent this. Furthermore, in pursuit of a more robust solution to tackle randomized model mismatches, we used the concept of formulating soft state constraints. By integrating slack variables into our control framework, we aimed to ensure the feasibility of the problem even under circumstances of model mismatch.

4.1 Adapting Controller Tuning

The parameters used to build the controllers are presented in the following table. The parameters have been modified and tuned according to the method described in the previous section, to achieve a good performance and keep the problem feasible in the context of the given reference trajectory, before changing to the soft-constrained formulation of the problem. A more fine-grained tuning has been executed on the Q matrix, by setting different weights to the individual state variables.

Subsystem	Q	R	S	s
x controller	Q(1,1) = 40, Q(2,2) = 200, Q(3, 3) = 10, Q(4,4) = 100	0.04	100	50
y controller	Q(1,1) = 40, Q(2,2) = 200, Q(3, 3) = 10, Q(4,4) = 100	0.04	100	50
z controller	Q(1,1) = 20, Q(2,2) = 200	0.01	-	-
γ controller	Q(1,1) = 50, Q(2,2) = 75	0.05	-	-

Table 3: Tuning Parameters

When relaxing the constraints using slack variables, the amount of constraint violation is penalised by employing linear and quadratic penalties:

$$\epsilon^T S \epsilon + s^T \epsilon$$

The s and S parameters are responsible for adjusting the effect of them respectively. Increasing s, results in increasing peak violations and decreasing duration. On the other hand, an increase in S leads to the hardening of the soft constraints. Therefore, we opted for substantially larger values of s and S to reduce the chance of violating constraints over an extended period and to a considerable amount. Considering the state variables, higher weights were assigned to angular velocities to avoid too aggressive manoeuvres and minimize angle constraints. Additionally, to better track the trajectory, the positions' weights also increased. R parameter was also decreased, which generally made the tracking more accurate.

4.2 Plots 4.1

The following plots illustrate the successful path tracking achieved in our simulations.

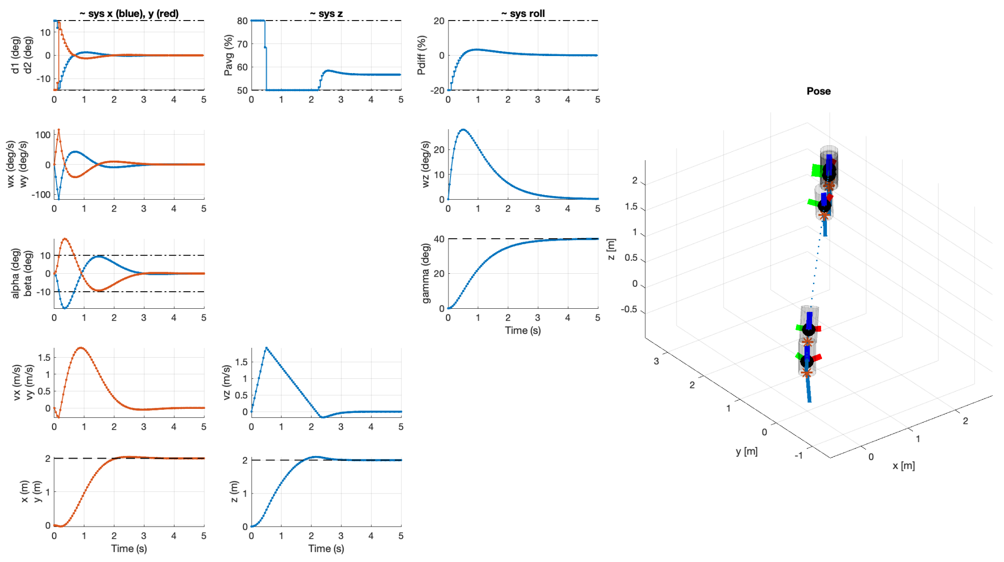


Figure 14: Linear MPC in non-linear model, open-loop trajectory

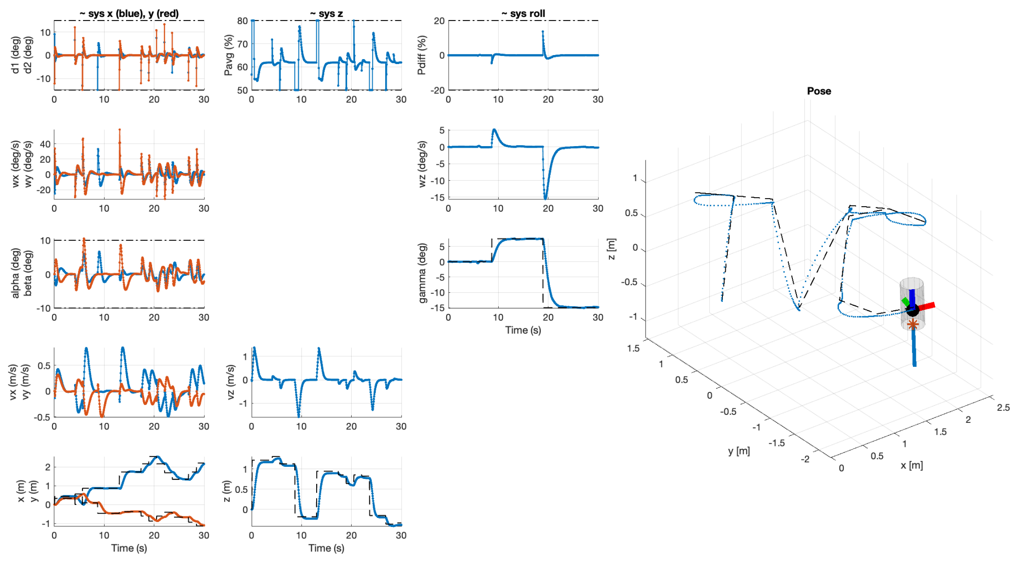


Figure 15: Linear MPC in non-linear model, closed-loop trajectory

5 Deliverable 5: Offset free tracking

In this section, the aim was to formulate a controller that could effectively counteract the influence of potential deviations in the system model via offset-free tracking. Specifically, our system encounters a

variation in the mass, which results in a constant disruptive force in terms of weight.
Consequently, the controller for the z direction had to be augmented as follows:

$$\begin{aligned} x_{z,k+1} &= A \cdot x_z + B \cdot u_k + B \cdot d_k \\ d_{k+1} &= d_k \\ y_k &= C_z \cdot x_{z,k} \end{aligned}$$

with a new augmented state:

$$x_{augm} = \begin{bmatrix} x_{z,k} \\ d_k \end{bmatrix}$$

Furthermore, the existence of a disturbance implies that the states can no longer be computed with the model itself; instead, they must be measured and recalculated, assuming a constant bias. Consequently, the introduction of an estimator is necessary to estimate both the system state and the disturbance.

The configuration in Figure 16 was applied to the model to incorporate the capability for offset-free tracking in the presence of a disturbance.

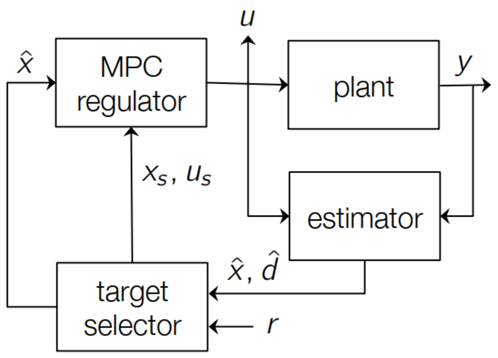


Figure 16: Offset-free tracking diagram

5.1 Estimator

The estimator for the state and the disturbance takes the following form:

$$\begin{bmatrix} x_{z,k+1} - \hat{x}_{z,k+1} \\ d_{k+1} - \hat{d}_{k+1} \end{bmatrix} = \left(\begin{bmatrix} A & B \\ 0 & I \end{bmatrix} + \begin{bmatrix} \mathbf{L}_x \\ \mathbf{L}_d \end{bmatrix} \begin{bmatrix} C & 0 \end{bmatrix} \right) \begin{bmatrix} x_{z,k} - \hat{x}_{z,k} \\ d_k - \hat{d}_k \end{bmatrix}$$

$$e_{k+1} = (A_{bar} + L_{bar} \cdot C_{bar}) \cdot e_k$$

where $\hat{x}_{z,k+1}$ and $\hat{d}_{k+1}$ are the state and disturbance estimates, respectively.

At each time step, the state's measurements obtained through the C matrix are compared with the predicted state to estimate the disturbance in the model. To ensure the rapid convergence of the predicted state to the real state, the determination of the L_x and L_d is crucial.
It is imperative to select L_x and L_d in such a way that the error dynamics remain stable and converge to zero. In our case, they were determined through pole placement. The closer the eigenvalues are to zero, the more aggressive the system's behaviour can be, leading to possible overshooting of the disturbance

estimator. In light of this, the poles were chosen with higher values. The specific eigenvalues are the following:

$$\lambda = [0.75, 0.43, 0.87]$$

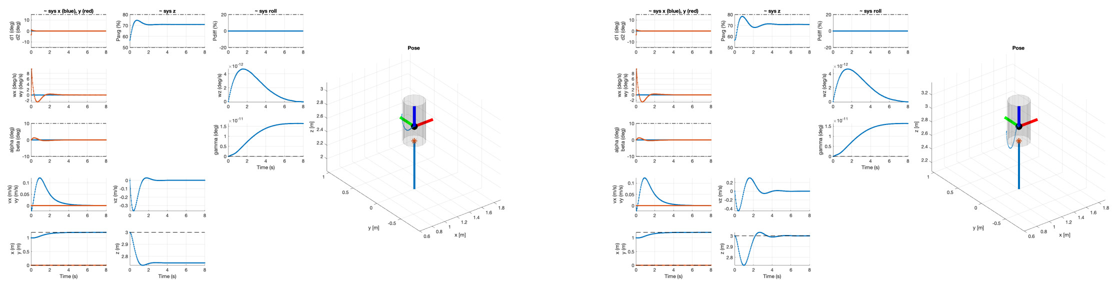
After the MPC regulator and the estimator, the target selector was modified. The model was updated to include the disturbance, and the input constraints were dropped. They were removed due to the potential infeasibility of the input in case of a very large disturbance estimation. However, it's worth noting that the MPC regulator already considers and addresses these constraints.

At last, to enhance tracking performance, the values for the cost matrices Q and R were revised. The cost associated with the input was raised to minimize fuel consumption. Regarding the state cost, the expense associated with the z position was significantly increased to highlight the importance of position tracking. The updated values are as follows:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 400 \end{bmatrix} \qquad R = 0.1$$

5.2 Constant mass

The additional mass was applied to both the tracking controller and the offset free-tracking controller. A pure tracking controller cannot detect disturbances and compute the accurate input for achieving the target state. Consequently, it fails to reach the intended point. In contrast, the offset-free tracking controller exhibits the ability to respond to disturbances and successfully attain the goal.



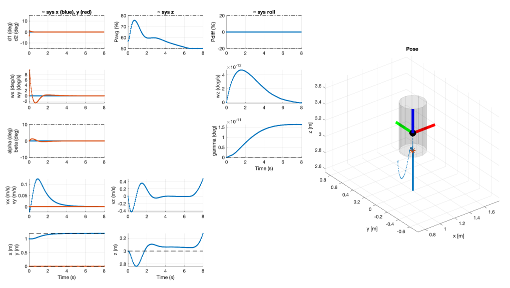
No estimator With estimator

Figure 17: Constant mass disturbance

5.3 Dynamic mass

The controller was afterwards tested with a dynamic mass, that gradually decreases as the fuel is consumed. Given that our initial models were based on the assumption of a constant disturbance, the controllers may face challenges in compensating for the disturbance's variability over time. To accommodate this dynamic disturbance, modifications can be made to the model. Specifically, the disturbance prediction equation can be updated to incorporate its correlation with the average thrust being used.

The performance of the controller with a dynamic disturbance was then evaluated. The system's initial behaviour closely mirrors the one in the , as the mass disturbance can be considered constant. As time progresses, the fuel consumption alters the mass from its initial value. Although the rocket's z position is slightly higher than the target position, it still successfully stays within a close range of the desired state. After 7 seconds the challenges faced by the rocket are primarily associated with input constraints and fuel limitations. The rockets' continuous loss of mass, causes the minimum average thrust to eventually exceed the steady-state position value. Consequently, the rocket ascends. Furthermore, between 15 to 20 seconds, it begins to descend due to insufficient fuel.


8 seconds

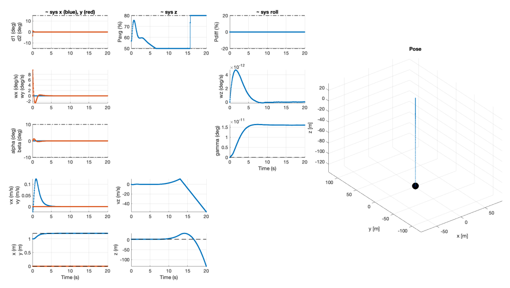

20 seconds

Figure 18: Dynamic mass disturbance

6 Deliverable 6: Nonlinear MPC

In this part, our objective is to develop a Nonlinear Model Predictive Control (NMPC) controller for the rocket system, utilizing the CASADI framework. The NMPC controller is distinct in considering the rocket's full state as its input, rather than breaking it down into separate sub-systems. This approach allows the controller to operate effectively across the entire state space of the rocket's nonlinear model. A key aspect of our design is to address the Euler angle singularity issue. Specifically, the Euler angle attitude representation in the rocket model exhibits a singularity at $\beta = 90°$. To mitigate this, we constrain the angle β within safe numerical limits, ensuring $|\beta| \leq 75°$ to avoid any potential singularity issues during operation.

6.1 Implementation of NMPC

The formulation of a nonlinear controller is similar to the one used for the linear subsystems:

$$u^* = argmin \sum_{i=0}^{N-1} l(x_i, u_i) + V_f(x_N)$$

$$\begin{aligned} s.t. \quad x_{i+1} &= \hat{f}(x_i, u_i) \\ Fx_i &\leq f \\ Mu_i &\leq m \end{aligned}$$

Where $\hat{f}$ is the discrete dynamic of the system, obtained by integrating the continuous one, as described in the following section.

We do not have complex nonlinear constraints, but just box constraints that can be modelled in the same way as the linear controllers. As we are not using the linearization around the steady state point anymore, the constraints limiting the rocket's state close to the trim point can be dropped.

For the definition of the terminal cost V_f, we have chosen to use the cost given by an unconstrained LQR controller, as in the previous sections.

We have chosen to use a quadratic cost function, using a Q matrix for the state weights and R for the input ones. As we have to track a reference, the formulation of the cost function is therefore the following:

$$V(x) = \sum_{i=0}^{N-1} (x_i - x_{ref})^T Q (x_i - x_{ref}) + u_i^T R u_i + (x_N - x_{ref})^T Q_f (x_N - x_{ref})$$

Where Q_f is the solution of the Discrete Algebraic Riccati Equation (DARE) for the LQR controller.

6.1.1 State integration

One of the aspects that poses a challenge in the implementation of the NMPC is the computation of the evolution of the state over time, which is done by integrating the system dynamics' continuous function. During the course, various methods have been presented to achieve this result. The one that we chose to use is the *Runge-Kutta 4* method, illustrated below.
Given the dynamic model of the system:

$$\dot{x} = f(x, u)$$

The state at the next step in time is computed as follows:

$$K_1 = f(x, u)$$

$$K_2 = f(x + K_1 \frac{T_s}{2}, u)$$

$$K_3 = f(x + K_2 \frac{T_s}{2}, u)$$

$$K_4 = f(x + K_3 T_s, u)$$

$$x^+ = x + \frac{T_s}{6}(K_1 + 2K_2 + 2 * K_3 + K_4)$$

6.1.2 Tuning of the weights

The idea behind the tuning of the Q and R weights is the same as the one used for tuning the linear controller, as described in the first section. The objective is to find the right balance between the state and input cost while paying more attention to the fine-grained tuning of the individual variables to address specific issues and improve the overall performance at the end.
At the end, the following weights have been used:

$$Q = \begin{bmatrix} 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 500 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 15 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4500 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4500 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4500 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 2.5 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix}$$

6.2 Comparing Nonlinear and Linear Controllers

Nonlinear controllers offer advantages in terms of modelling accuracy and versatility, making them suitable for complex systems with nonlinear dynamics and wide operating ranges. They can effectively handle constraints, enhancing safety and stability. However, their design complexity and increased computational demands can pose challenges. In contrast, linear controllers are simpler to design and implement, particularly for systems that can be approximated by linear models, resulting in quicker development and lower implementation costs. Yet, their effectiveness is limited to a narrow range around an equilibrium point, and they struggle with efficient constraint handling. The choice between nonlinear and linear controllers hinges on the specific system and control objectives, with each approach having its trade-offs, making it essential to carefully assess the application's requirements and constraints.

The comparison between NMPC and LMPC under a substantial roll angle constraint of 50° clearly demonstrates their respective capabilities. This scenario was chosen to expose the limitations of LMPC in extreme conditions (when going away from the linearization point) while illustrating the proficiency of NMPC in managing complex dynamics.

6.3 Plots 6.1

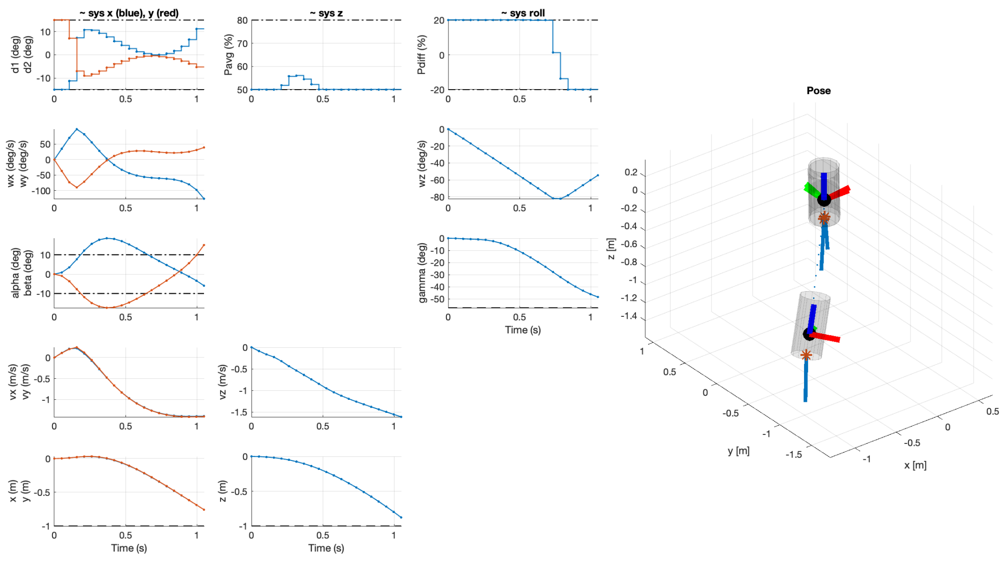
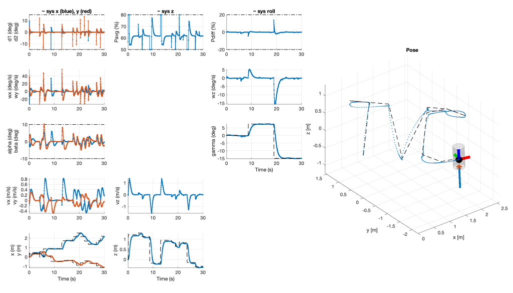
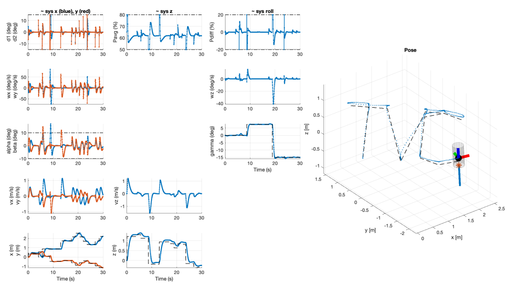


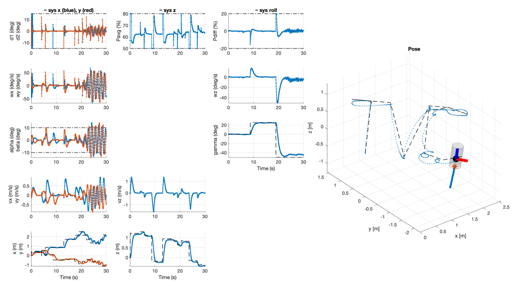
Figure 19: Open loop Nonlinear MPC trajectory

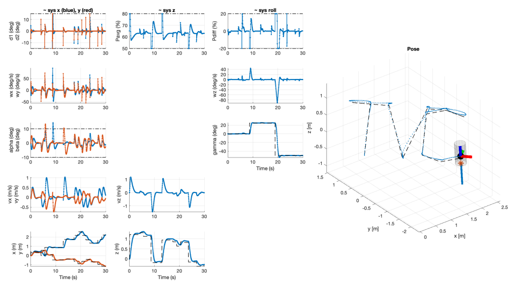
Linear MPC

Nonlinear MPC

Figure 20: LMPC and NMPC control performance at 15° roll angle constraint



Linear MPC



Nonlinear MPC

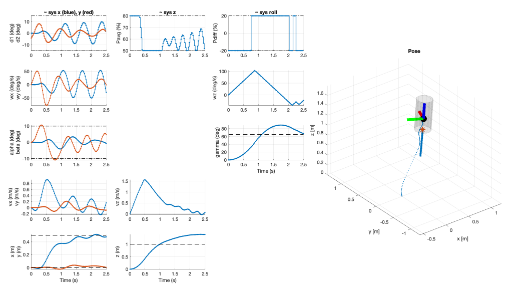
Figure 21: LMPC and NMPC control performance at 50° roll angle constraint

6.4 Effects of delay

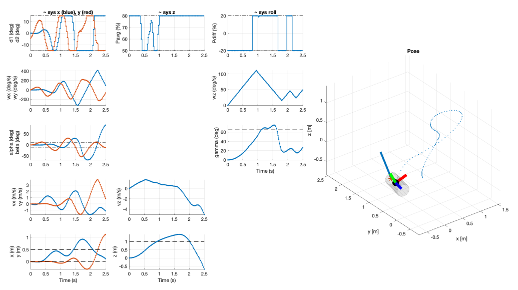
In this part, we had to evaluate the effect of delay on our controller.
As the nonlinear controller is very resource-intensive in terms of computational cost, it is difficult to have a real-time controller that can provide input at the sampling rate.
For this reason, when the controller has not computed a new input, the old one is used until there is a new value available.
We first evaluated the effects of the delay on a standard controller, and then we tried to implement a countermeasure to this problem, implementing compensation.

6.4.1 Instability of standard controller

Degradation in the performance of the controller can be observed starting from a delay of 3 steps, however it is more noticeable at 4 steps. Starting from a delay of 5 steps, equivalently $0.125s$, the system becomes unstable and is not controllable anymore.
Below is a plot of the simulation run with a delay of 4 and 5 steps:



4 steps / 0.1s delay



5 steps / 0.125s delay

Figure 22: Controller with delay without compensation

6.4.2 Delay compensation

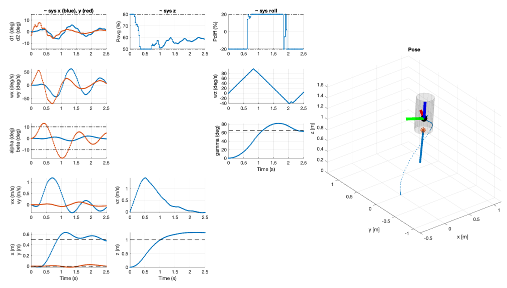
As a countermeasure to the effects of the delay, a compensator has been implemented.
At the time step k, the next input is not computed for time $k+1$, but for time $k+T_{delay}$.
To achieve this, the state dynamics has to be integrated, and for this, an Euler integration has been implemented, illustrated below:

$$x_{k+1} = x_k + T_s f(x_k, u_k)$$

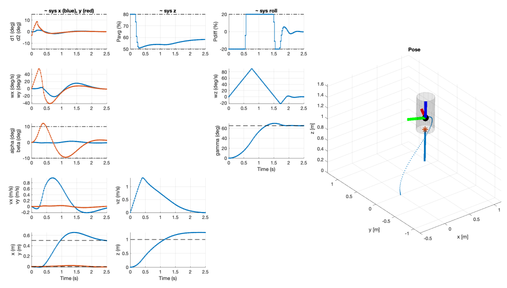
A matrix mem_u has been defined to save the computed inputs.
At the beginning of the simulation, it is initialized with the steady-state input that makes the rocket hover:

$$u_0 = \begin{bmatrix} 0 & 0 & 56.667 & 0 \end{bmatrix}^T$$

Then, at each time step the mem_u matrix is updated adding the last input computed by the MPC optimizer.
Below is a plot of the simulation of the partially and fully compensating controllers for a delay of 8 steps ($0.2s$), with 6 and 8 steps compensation respectively:



6 steps / 0.15s compensation



8 steps / 0.2s compensation

Figure 23: Controller with delay with compensation

As we can see, the performance is acceptable for the partially compensating controller, however, the input variables δ_1 and δ_2 still show a bad oscillation. For the fully compensating controller, this behaviour is not present anymore and we observe a very good performance.

7 Conclusion

In conclusion, this report presents a comprehensive journey through the development and application of various MPC strategies for a rocket prototype. The project started with a detailed analysis of the rocket's dynamics, followed by system linearization and subdivision into four subsystems. Subsequently, MPC regulators for each subsystem were meticulously designed, focusing on recursive feasibility and performance optimization. The study then progressed by implementing linear MPC in a nonlinear model, where the introduction of soft state constraints played a pivotal role in maintaining feasibility under model mismatch scenarios. The report also explores offset-free tracking to counteract disturbances, particularly variations in rocket mass. Finally, a Nonlinear MPC was developed, leveraging the full state of the rocket model and addressing Euler angle singularity issues. The performance of the NMPC was thoroughly evaluated and compared to linear controllers, highlighting the advantages and limitations of each approach. This study not only demonstrates the effectiveness of MPC in managing complex dynamic systems but also provides valuable insights into the nuances of controller design and tuning for optimal performance.