January 2021 - Master I

Model Predictive Control - Project Report

Ecole polytechnique fédérale de Lausanne

MPC Controller to fly a rocket prototype

GREBICI Yucef
GUILLAUME Camille
HERMANN Théo

Group 46

Contents

1 Introduction

During this project for the course ME-425: Model Predictive Control given by Dr. Colin Jones, we will develop several different controllers for a rocket prototype. Our role is to implement our MPC controller Matlab code into the provided simulation. First, we will simulate the rocket dynamics by hands, in order to understand how it react to the different input commands, then using linearized dynamics we will divide the rocket dynamics into 4 independent subsystems to develop MPC regulators for each of them. Afterwards, we will implement a tracking controller allowing the rocket to track a given point. This reference tracking MPC controller will be used to make the rocket track a complex reference path. We will then implement an offset free tracking controller, which will be able to track the same path reference despite a mass mismatch. Finally, we will design a MPC controller for the Non Linear Dynamics and compare its performance against the linear controllers.

2 Deliverable 2: Linearization

The state vector we are working on in this project is described as:

$$\overrightarrow{state} = \overrightarrow{x} = \begin{bmatrix} \omega_x & \omega_y & \omega_z & \alpha & \beta & \gamma & v_x & v_y & v_z & x & y & z \end{bmatrix}^T \tag{1}$$

And the input vector we are working with is:

$$\overrightarrow{inputs} = \overrightarrow{u} = \begin{bmatrix} \delta_1 & \delta_2 & P_{avg} & P_{diff} \end{bmatrix}^T \tag{2}$$

The system will be linearized around the steady state, to have $f(x_s, u_s) \approx 0$. By running the *rocket.trim* function we find the x_s and u_s that validate the steady state condition. We obtain for the steady–state state vector:

$$\overrightarrow{x_s} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3}$$

and we obtain for the steady–state input vector:

$$\overrightarrow{u_s} = \begin{bmatrix} 0 & 0 & 56.6667 & 0 \end{bmatrix}^T \tag{4}$$

The steady state vector corresponds to a vertical rocket, totally immobile. The values in the steady state input vector imply that in steady state, the drone remains propelled vertically (in order to overcome gravity). The rocket remains stationary. The system must therefore be constrained not to deviate too much from these values.
With this information in our knowledge, we can observe the matrices resulting from the linearization around the steady state state and input vectors, by executing *rocket.linearize(x_s,u_s)* and sub-divide them into 4 independent sub-systems:

- An independent system depending on the δ_1 input.

$$\overrightarrow{\dot{x_y}} = \begin{bmatrix} \dot{\omega_x} \\ \dot{\alpha} \\ \dot{v_y} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -9,81 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} \omega_x \\ \alpha \\ v_y \\ y \end{bmatrix} + \begin{bmatrix} -55,68 \\ 0 \\ -9,81 \\ 0 \end{bmatrix} * \delta_1 \tag{5}$$

- An independent system depending on the δ_2 input.

$$\overrightarrow{\dot{x_x}} = \begin{bmatrix} \dot{\omega_y} \\ \dot{\beta} \\ \dot{v_x} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 9,81 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} \omega_y \\ \beta \\ v_x \\ x \end{bmatrix} + \begin{bmatrix} -55,68 \\ 0 \\ 9,81 \\ 0 \end{bmatrix} * \delta_2 \tag{6}$$

- An independent system depending on the P_{avg} input.

$$\overrightarrow{\dot{x_z}} = \begin{bmatrix} \dot{v_z} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} \omega_z \\ z \end{bmatrix} + \begin{bmatrix} 0,1731 \\ 0 \end{bmatrix} * P_{avg} \tag{7}$$

- An independent system depending on the P_{diff} input.

$$\overrightarrow{\dot{x}_{roll}} = \begin{bmatrix} \dot{\omega_z} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} \omega_z \\ \gamma \end{bmatrix} + \begin{bmatrix} -0,104 \\ 0 \end{bmatrix} * P_{diff} \tag{8}$$

Note that each input influences a number of variables.
Thus, the rotation of an angle δ_1 around the axis x of the thruster causes a non zero speed v_y, therefore displacement along y, as well as a rotational speed around x, w_x, and an angle variation β.
The same happens for δ_2 around the axis y of the thruster causes a non zero speed v_x, therefore displacement along x, as well as a rotational speed around y, w_y, and an angle variation α.
Note that the angle of rotation along z, γ, which we call the roll angle and its speed of rotation is only modified by the rotational propulsion P_{diff}.
P_{avg} only modifies z and the speed of translation according to z, v_z, because it is the vertical propulsion under the rocket.
Of course, in reality, a rotation of δ_1 influences the position z, because P_{avg} is propelled less vertically. But as our system is linearized around the vertical, the subsystems described above only make sense when the states are close to this linearized state.

3 Deliverable 3: MPC controllers for each sub-systems

In the previous part we linearized the rocket system and divided it into 4 independent sub-systems for x, y, z and roll. We will now design two controllers based on Linear Model Predictive Control. The first controller being a regulator it will pull the system to the origin and the second one is a reference tracking controller allowing to set a reference point different from the origin and making the rocket position converge to this reference point. Furthermore, the choice of used tuning parameters for each controller will be reviewed. Finally, states and inputs variables will be visualized to monitor the performance of the controllers.

3.1 Deliverable 3.1: MPC regulators

3.1.1 Ensuring recursive feasibility

For the practical implementation of the MPC controller we have to split the ideal infinite horizon problem into two sub-problems to provide realistic computational cost. From step 0 to N we will formulate the Finite Horizon MPC with states and inputs constraints. Then, from step N to infinity we will drop the constraints and solve the optimization problem to find a control law and its associated invariant terminal set. We also need that the cost from the state x_N to the origin to be finite and described by a Lyapunov function in the terminal set. The terminal control problem results is an unconstrained LQR problem. We find a control law $\kappa_f(x)$ and invariant terminal set X_f ensuring that all state and inputs constraints are verified. The terminal cost is $V_f(x_N) = x_N^T Q_f x_N$, where Q_f is the solution of the Discrete Algebraic Riccati equation (DARE).

This design results in the following MPC problem formulation for the subsystem x. The design for the other subsystems y, z and roll is similar:

$$\boxed{\begin{gathered} J^*(x) = \min_{x,u} \left(\sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T Q_f x_N \right) \\ \text{s.t. } \mathrm{x}_{i+1} = A x_i + B u_i \\ \mathrm{Fx}_i \leq f \\ \mathrm{Mu}_i \leq m \\ \mathrm{x}_N \in X_f \end{gathered}}$$

Matrix A and B represent the linearized model dynamics, F and f are state constraints and M and m are inputs constraints. Each subsystem has different constraints summarised in the table below. We drop the state constraints for the z and roll subsystems because there is no specification of state constraints for those subsystems.

Subsystem	State	Input
MPC x controller	$\lvert\beta\rvert \leq 5°$=0.0873 rad	$\lvert\delta_2\rvert \leq 15°$
MPC y controller	$\lvert\alpha\rvert \leq 5°$=0.0873 rad	$\lvert\delta_1\rvert \leq 15°$
MPC z controller		$50\% \leq P\_avg \leq 80\%$
MPC γ controller		$\lvert P\_diff\rvert \leq 20\%$

3.1.2 Parameters tuning method

MPC controllers need the tuning of the following parameters: the horizon length N, the matrix Q and the matrix R which are both involved in the cost function, and the terminal components.

The horizon length parameter N has influence on the region of attraction and the speed of convergence. When picking a N value too small the problem becomes unfeasible, indeed the controller can not find any solution to reach the terminal set in only N steps. However, picking a big value for N results in increasing the computational cost.

The cost matrices Q and R play a role in the minimized cost function and also in the size of the terminal set. Incorrect choice for those matrices could lead to unfeasibility. An intuitive interpretation for the tuning of those matrices is that Q refers to the "performance" of the system (how fast it will converge for a specific state) while R refers to the "efficiency" (avoiding too much actuator effort). For our case we want to have a settling time of 8s while avoiding oscillations in the input. We defined settling time as the time before the signal enters a region of 5% around the desired value (here the origin). We started by using $Q = I$ and $R = 1$, first adjusting the ratio between both and then adjusting the individual weights of the Q matrix. Putting more weight on one individual weight of the Q matrix allows us to penalise more this state variable in the cost function and therefore to force it to get faster to the origin. This can solve for example oscillation and can reduce the settling time. However this can also result in overshooting, so we need to take these two effects into account and counterbalance them when tuning our parameters. This is the same reasoning when putting more weight on R compared to Q or on Q compared to R.

The terminal cost matrix Q_f is obtained using the LQR solution from DARE using the function $dlqr(A, B, Q, R)$ in Matlab.

3.1.3 Parameters tuning choice

We get the following tuning for regulation:

Subsystem	Q	R	H[s]	N
MPC x controller	Q(1,1)=1 Q(2,2)=5 Q(3,3)=0.1 Q(4,4)=0.1	20	2.5	50
MPC y controller	Q(1,1)=1 Q(2,2)=5 Q(3,3)=0.1 Q(4,4)=0.1	20	2.5	50
MPC z controller	Q(1,1)=60 Q(2,2)=60	35	5	100
MPC γ controller	Q(1,1)=50 Q(2,2)=50	0.1	5	100

We can see for example for subsystems x and y that in the cost function we put more weight on the first two state variables which are respectively ω_y and β for the x subsystem and ω_x and α for the y subsystem. For subsytems z and γ we set Q bigger than R.

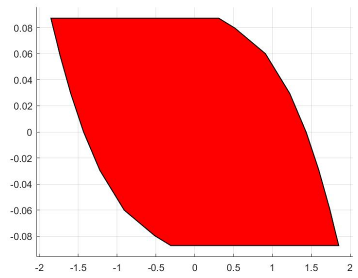
3.1.4 Terminal Invariant Set

To compute the terminal set for each subsytem, we use the following algorithm which finds the maximal invariant set based on the state and input constraints and on the following property:
A set C is a control invariant set if and only if C $\subseteq pre(C)$.

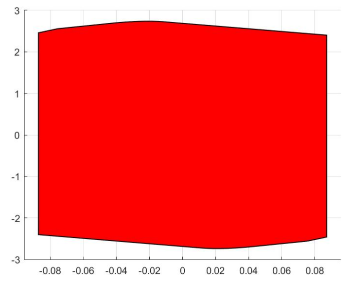
```
Conceptual Algorithm to Compute Invariant Set

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

Figure 1: Algorithm that computes a control invariant set

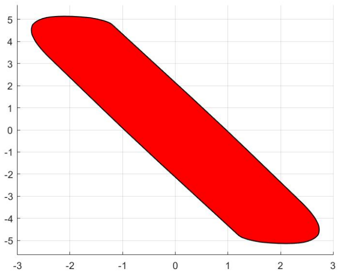
We obtain the following terminal sets given our constraints and cost matrices.



(a) Projection on x_1 and x_2

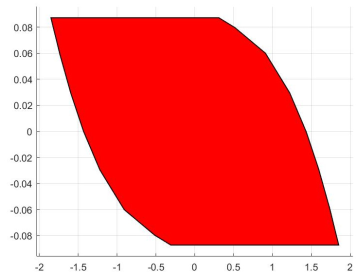


(b) Projection on x_2 and x_3

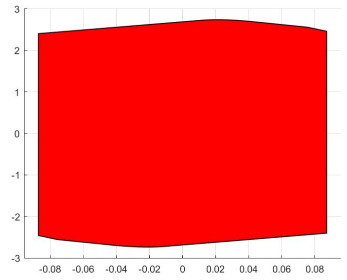


(c) Projection on x_3 and x_4

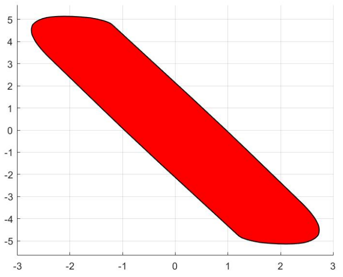
Figure 2: Terminal set of subsystem x projected in 2 dimensions



(a) Projection on x_1 and x_2



(b) Projection on x_2 and x_3



(c) Projection on x_3 and x_4

Figure 3: Terminal set of subsystem y projected in 2 dimensions

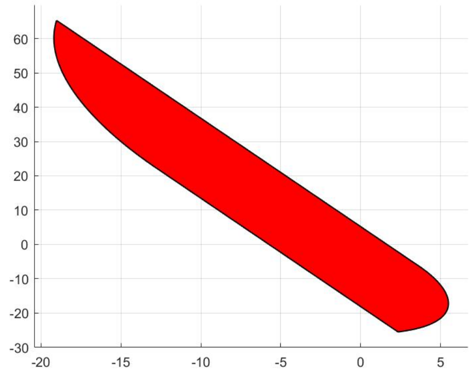


Figure 4: Terminal set of subsystem z

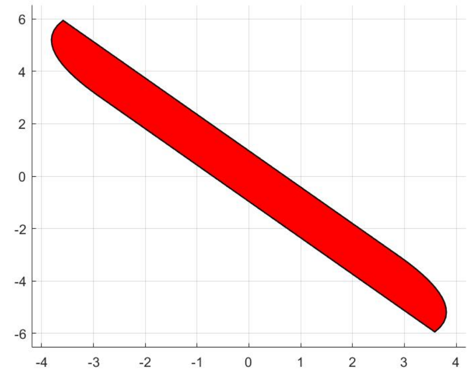


Figure 5: Terminal set of subsystem γ

3.1.5 Regulation Simulation

We simulate our four controllers and obtain the following graphs. We can see that with our tunings, all controllers manage to converge and stabilize to the origin within the maximum settling time of 8s. We can also see that the constraints are recursively satisfied while avoiding inputs oscillations. In subsystem z, one can notice on Figure 8 a small overshoot of the state variable z between 8s and 12s. However its minimum is at -0.148m which is smaller than our limit of 5% overshoot at -0.25m.

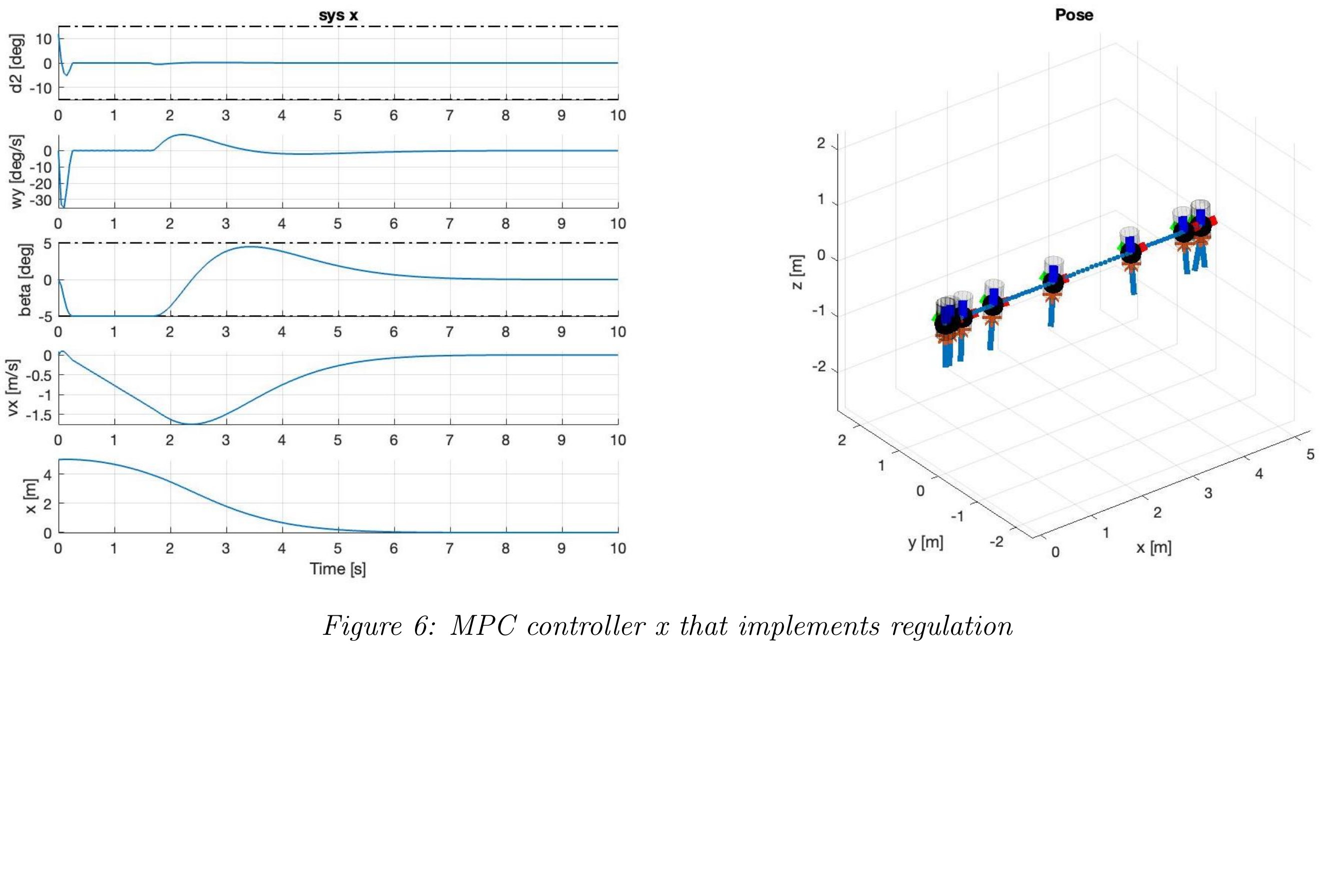


Figure 6: MPC controller x that implements regulation

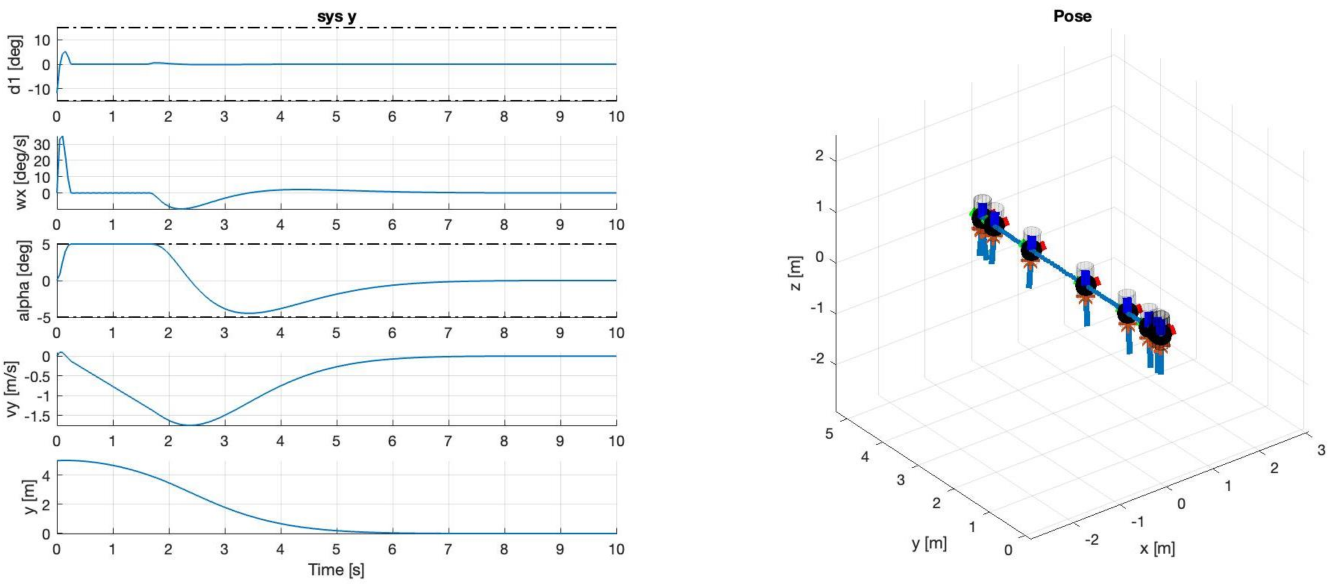


Figure 7: MPC controller y that implements regulation

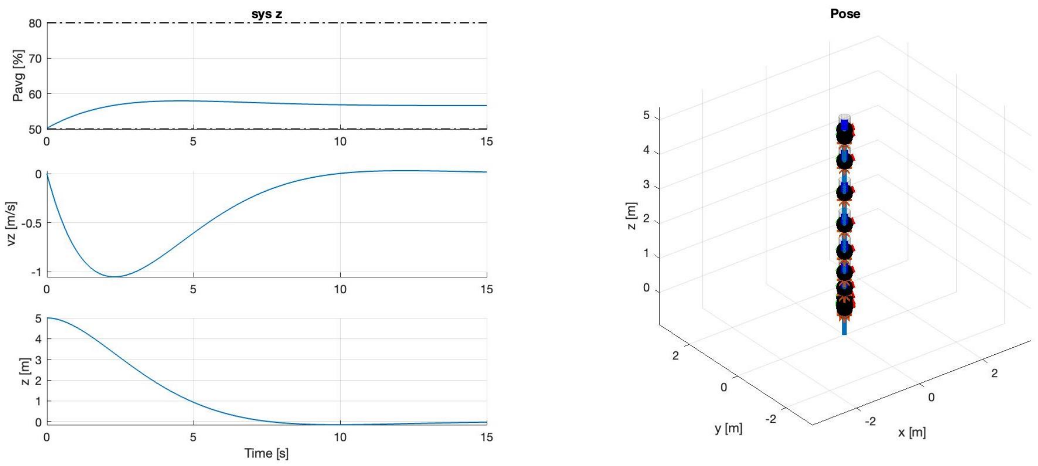


Figure 8: MPC controller z that implements regulation

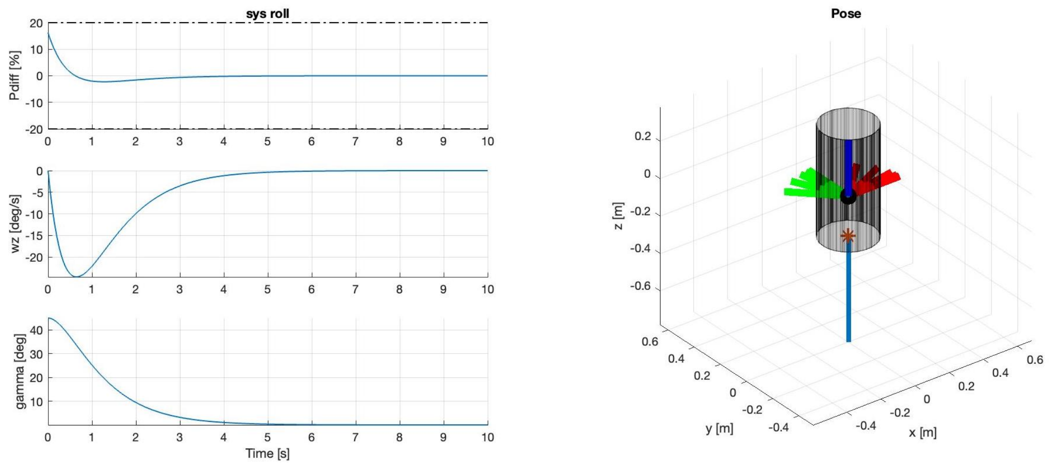


Figure 9: MPC controller γ that implements regulation

3.2 Deliverable 3.2: MPC Tracking Controller

The first step for the design of the MPC Tracking controller is to find the steady-state and input (x_s, u_s) associated to the reference output r that we want our system to reach. We will solve this steady-state problem for each individual subsystem following a similar approach. However, we will drop the terminal set constraint on the states.

Assuming that the reference target point is feasible, the steady-state problem for the different subsystems can be formulated as:

$$\boxed{\begin{gathered}\min_{u_s} = u_s^T R_s u_s \\ \\ \text{s.t. } \mathrm{x}_s = Ax_s + Bu_s \\ \mathrm{Cx}_s = r \\ \mathrm{Fx}_s \leq f \\ \mathrm{Mu}_s \leq m\end{gathered}}$$

By solving the equation above using the gurobi solver we are able to know the steady state and inputs in order to reach the reference point. Now that we have computed the steady-states and input (x_s, u_s) the tracking problem can be formulated as a standard regulation MPC problem with the only difference being that we have shifted the origin to the steady-states we want to reach.

After shifting the origin in the MPC regulator formulation, the reference tracking MPC problem can be formulated as:

$$\boxed{\begin{gathered}J^*(x) = \min_{x,u} \left(\sum_{i=0}^{N-1} ((x_i - x_s)^T Q (x_i - x_s) + (u_i - u_s)^T R (u_i - u_s)) + (x_N - x_s)^T Q_f (x_N - x_s) \right) \\ \\ \text{s.t. } \mathrm{x}_{i+1} = Ax_i + Bu_i \\ \mathrm{Hx}_i \leq h \\ \mathrm{Mu}_i \leq m \\ \mathrm{x}_0 = x.\end{gathered}}$$

3.2.1 Tuning parameters

We tuned our parameters according to the method described in [3.1.2]. We kept the same tuning as for regulation except for the horizon that we made smaller while staying long enough to allow good reference tracking.

Subsystem	Q	R	H[s]	N
MPC x controller	Q(1,1)=1 Q(2,2)=5 Q(3,3)=0.1 Q(4,4)=0.1	20	0.5	10
MPC y controller	Q(1,1)=1 Q(2,2)=5 Q(3,3)=0.1 Q(4,4)=0.1	20	0.5	10
MPC z controller	Q(1,1)=60 Q(2,2)=60	35	0.5	10
MPC γ controller	Q(1,1)=50 Q(2,2)=50	0.1	0.5	10

3.2.2 Reference tracking Simulation

We simulate our four controllers and obtain the following graphs. We can see that with our tunings, all controllers manage to converge and stabilize to the reference within the maximum settling time of 8s. We can also see that the constraints are recursively satisfied while avoiding inputs oscillations at steady state. In subsystem z, one can notice on Figures 12 and 14 again a small overshoot between 8s and 12s. However its minimum is at -5.148m which is smaller than our limit of 5% overshoot at -5.25m.

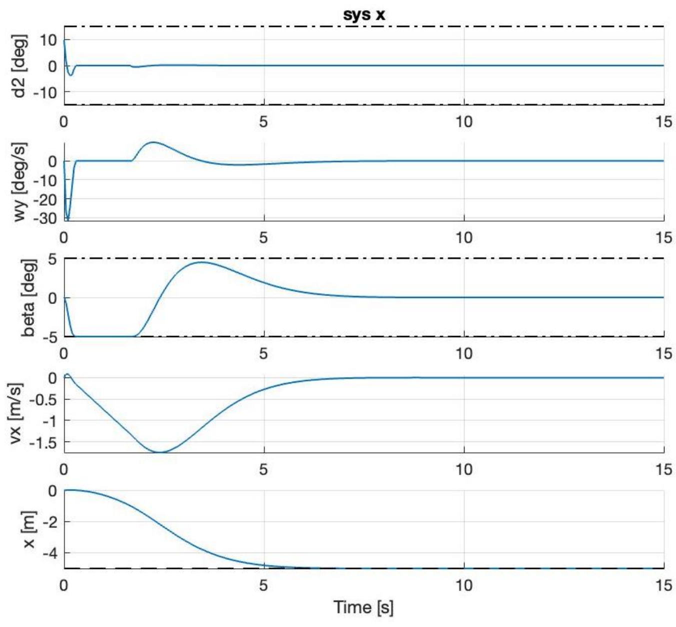


Figure 10: MPC controller x that implements reference tracking

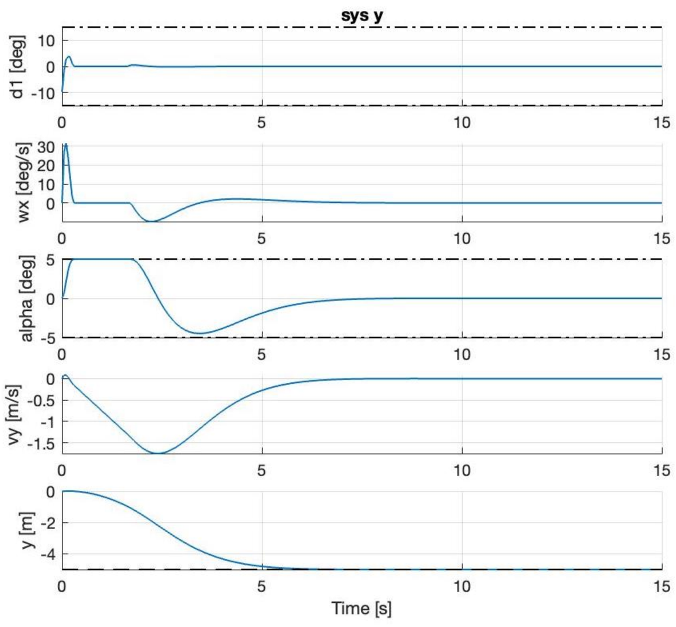


Figure 11: MPC controller y that implements reference tracking

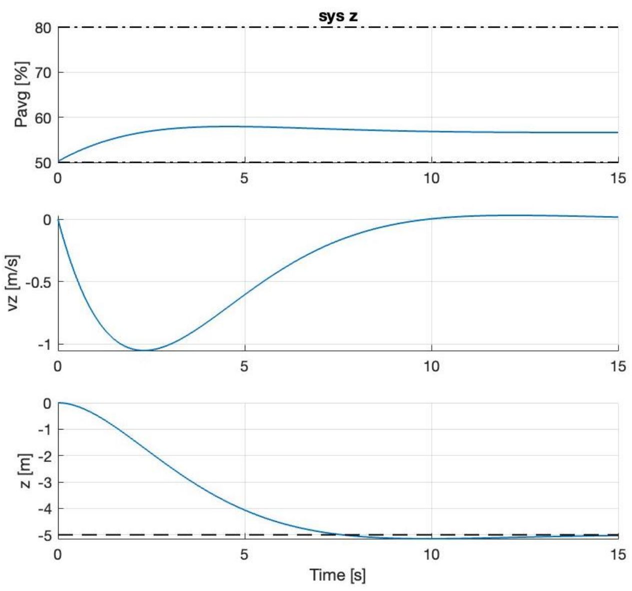


Figure 12: MPC controller z that implements reference tracking

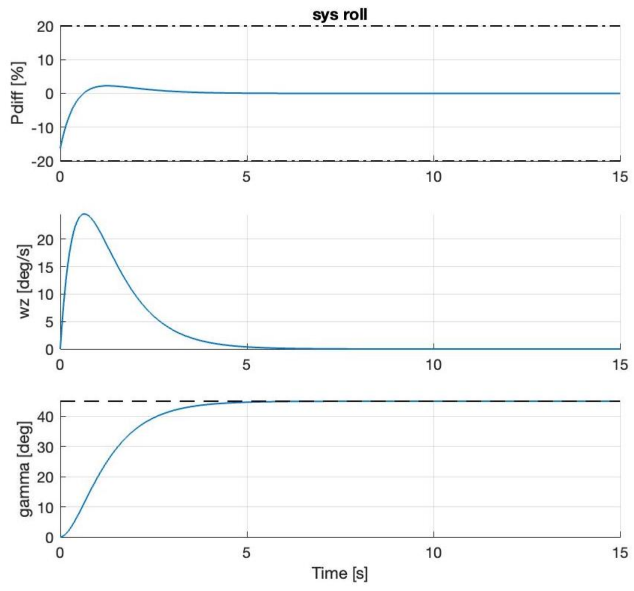


Figure 13: MPC controller γ that implements reference tracking

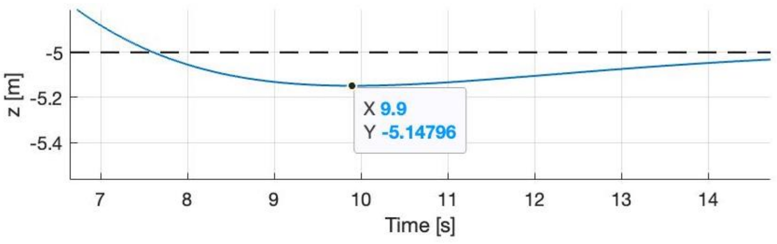


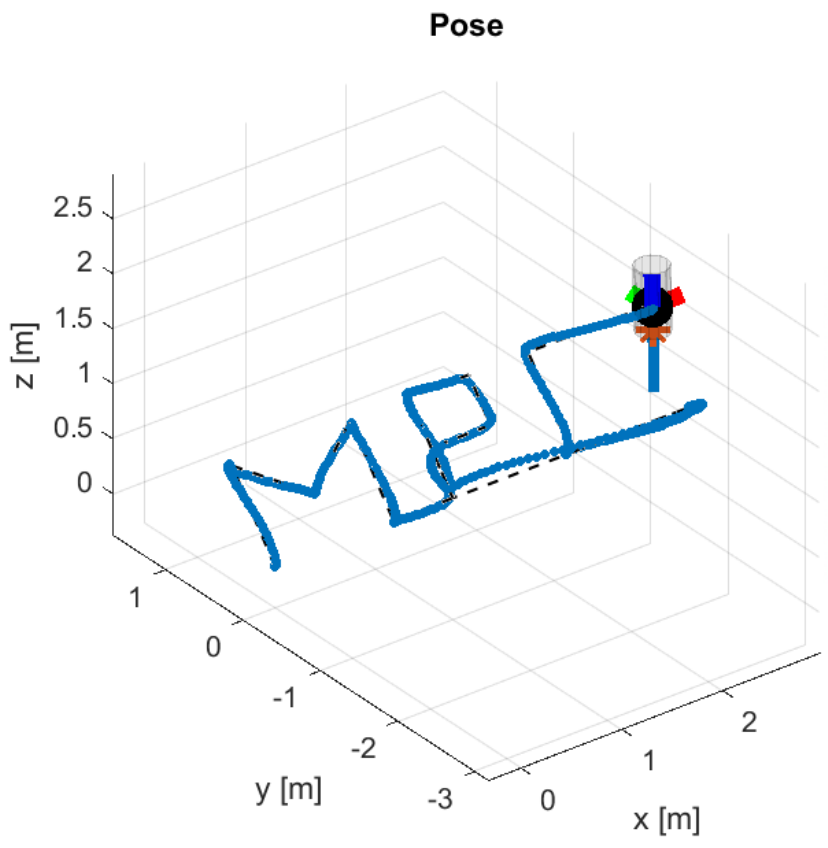
Figure 14: Subsystem z overshoot on state variable $x_4 = z$

4 Deliverable 4: Simulation with non linear rocket

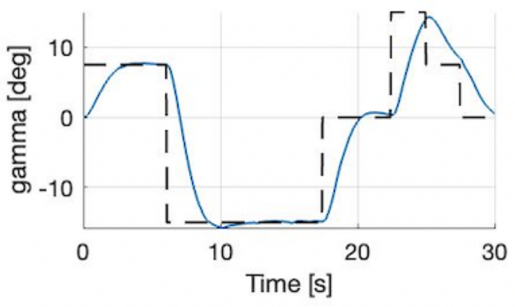
In this part we use our 4 reference tracking linear controllers as defined in 3.2 to track a reference path that writes "MPC" starting from the origin as initial state. To simulate the total rocket system we merge our four subs-systems together. To be able to run the nonlinear simulation with our linear subsystems, and to solve some infeasibility problems we decided to drop the first state constraint which allows the controller to have one time-step to get within the states constraints. Therefore we also removed this first state step from the objective function.

4.1 Reference path tracking of the total system

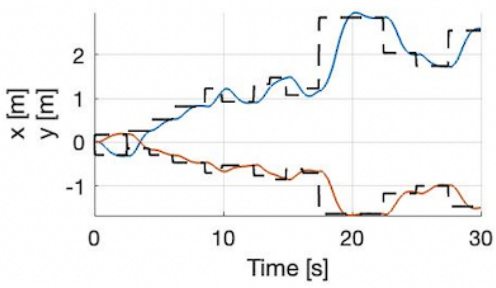
We get the following path tracking:



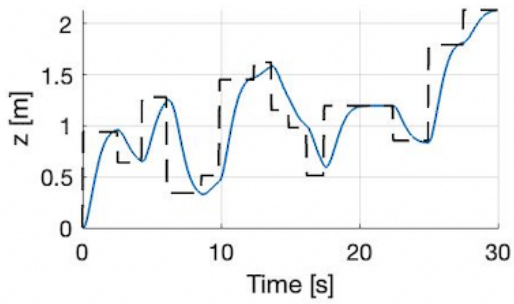
(a) Path tracking of the total system



(b) Path tracking of γ subsystem



(c) Path tracking of x and y subsystems

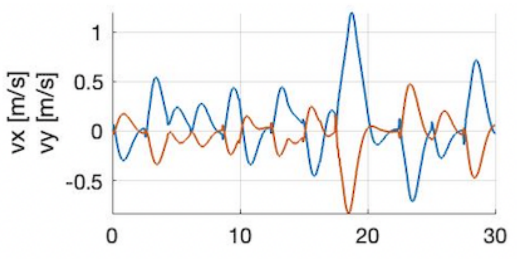


(d) Path tracking of z subsystem

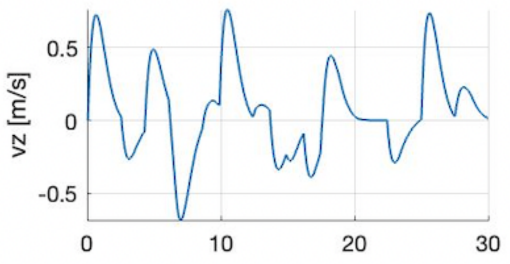
Figure 15: Path tracking simulation of the total linearized system with non linear rocket

We can see on Figure 15a that our choices and tunings allow the controller to perform good reference tracking although we did not manage to track correctly the bottom of the "P" letter. At this point the next reference point that corresponds to the start of the "C" letter is the largest distance the controller has to pursue. The predictive characteristic of MPC optimization, makes the controller anticipate this next point and therefore it doesn't finish the P letter.

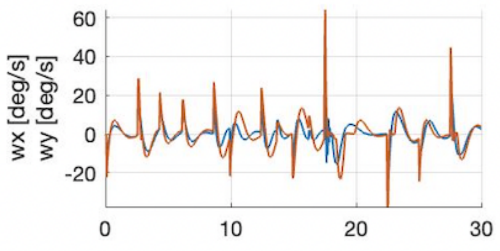
4.2 State and input variables evolution during tracking



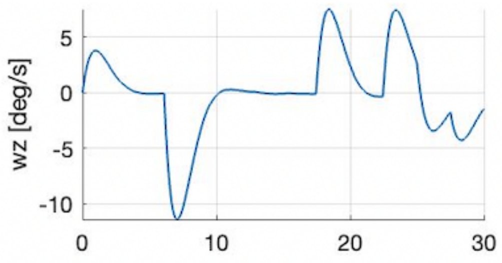
(a) v_x and v_y state variables respectively related to subsystems x and y



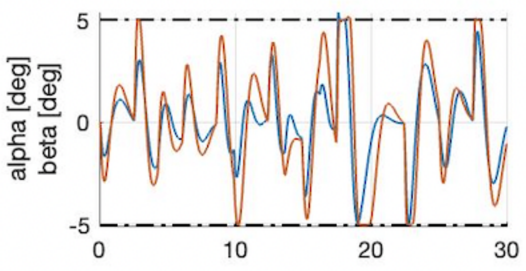
(b) v_z state variable related to subsystem z



(c) ω_y and ω_x state variables respectively related to subsystems x and y



(d) ω_z state variable related to subsystem γ



(e) β and α state variables respectively related to subsystems x and y

Figure 16: State variables evolution during tracking related to subsytems x, y, z and γ

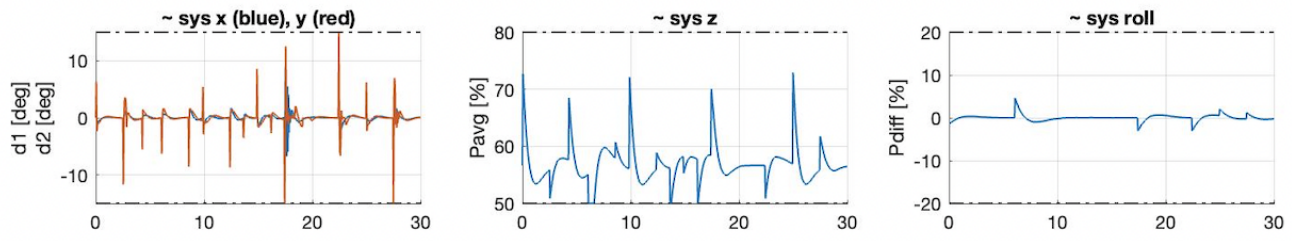


Figure 17: Input variables evolution during tracking related to subsytems x, y, z and γ

We can notice on Figure 16e that there is violation of the angle constraint of β until 5.36° at 17.6s and of α until 5.13° at 18.4s. This is due to the first step state constraint removal. We found that this violation was acceptable considering that it allowed us to tune our controller to get good tracking performance.

4.3 Tuning parameters

Here after are the tuning parameters we chose for the reference path tracking simulation:

Subsystem	Q	R	H[s]	N
MPC x controller	Q(1,1)=50 Q(2,2)=1 Q(3,3)=1 Q(4,4)=100	100	3	60
MPC y controller	Q(1,1)=50 Q(2,2)=1 Q(3,3)=1 Q(4,4)=100	100	3	60
MPC z controller	Q(1,1)=10 Q(2,2)=100	0.3	3	60
MPC γ controller	Q(1,1)=15 Q(2,2)=150	1	3	60

We defined our tunings according to the method described in 3.1.2. One can notice that we increased the horizon compared to reference tracking in 3.2 in order to improve our prediction but we kept it small enough to get a reasonable computation time of around 21s.
Within our state variables we decided to put more weight on the angular velocities and on the positions. The weight on the angular velocities allow us to minimise angle constraints violations. The weight on the position is the key parameter to have more aggressive tracking and follow these piece-wise continuous trajectories. That is why we set it larger then the other Q weights in all of our subsystems.
For subsystems x and y we see that we did not increase the weights Q(2,2) and Q(3,3) as they did not impact much the tracking. Although we set Q(4,4) to be the largest, we kept Q(1,1) at a ratio of 50% because it allows to compensate the overshoot induced by Q(1,1) and therefore increases tracking precision. For these subsystems we increased R which had small impact compared to Q and did not prevent aggressive position tracking. This increase made the inputs less aggressive which prevents overshooting and saturating inputs.
For subsystem z we decreased R to 0.3 which made the z tracking more accurate and synchronised with x and y tracking.
For the γ subsystem, we followed the same reasoning and got good tracking performance although the roll angle didn't impact much the global reference tracking.

5 Deliverable 5: Offset free tracking

5.1 Estimator Design

We aim to apply offset free tracking on the MPC model along z. Assuming a weight change of the rocket, the model should estimate and correct the induced variations on the concerned states.
The z–system (7) is augmented as following:

$$\begin{aligned} x_{z,k+1} &= \mathbf{A} * x_{z,k} + \mathbf{B} * u_{z,k} + \mathbf{B} * d_k \\ d_{k+1} &= d_k \\ y &= \mathbf{C}_z * x_{z,k} \end{aligned}$$

Giving a new state vector:

$$x_{augm,k} = \begin{bmatrix} x_{z,k} \\ d_{z,k} \end{bmatrix} \tag{9}$$

The disturbance $\boldsymbol{d}$ is assumed to be constant. Because of the disturbance, the real value of the states can't be modeled by the model anymore, and need to be measured and recomputed assuming a constant bias made by the disturbance.
The model will make a prediction with the dynamic model he knows, and compare it to the measurements he make (the measured states are given by the $\mathbf{C}$ matrix). By comparing the prediction and the measurement, the augmented can guess its 13^{th} state, the disturbance (assumed constant for a constant reference).

The model takes now the form of the figure 18, adding to the tracking ability, the ability to apply offset-free tracking in presence of a constant disturbance.

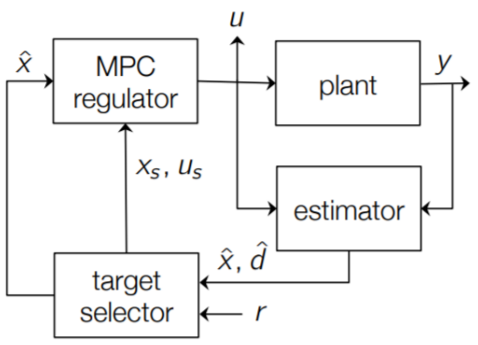


Figure 18: Model diagram of the offset–free reference MPC system.

Hence, the state and disturbance estimator takes this form:

$$e_{k+1} = \begin{bmatrix} x_{z,k+1} - \hat{x}_{z,k+1} \\ d_{k+1} - \hat{d}_{k+1} \end{bmatrix} = \left(\begin{bmatrix} A & B \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} L_x \\ L_d \end{bmatrix} \begin{bmatrix} C & 0 \end{bmatrix} \right) \begin{bmatrix} x_{z,k} - z\hat{,}x_k \\ d_k - \hat{d}_k \end{bmatrix} \tag{10}$$

Where $\hat{x}_{z,k+1}$ and $\hat{d}_{k+1}$ are estimates of the state and disturbance. From the error dynamics, choosing the right L_x and L_d by pole placement will speed up the estimation, i.e. the error between estimation and real values will converge to zero.

5.2 Estimator tuning

Estimator pole placement: L_x and L_d are chosen such that the error dynamics are stable and converge to zero. This is equivalent to having small eigenvalues of the estimation error state–update matrix (10). Eigenvalues with a small norm will speed up the estimation process, but may increase the overshoot of the estimate d. A large perturbation estimation can cause the problem of computing the set point to be infeasible.

The choice of poles depends on the situation in which we want to use our system. We choose two triplets of poles for testing: A set of fast poles, and a set of slow poles. The closer the poles are to 1, the slower they will be. Thanks to the *place* function in Matlab, the L vector is created in order to place the chosen poles into the estimation error dynamics. This leads to L_{fast} and L_{slow} being created by a set of fast and slow poles, as follows:

$$L_{fast} = -place(A'_{bar}, C'_{bar}, [0.2, 0.15, 0.1])' \tag{11}$$

$$L_{slow} = -place(A'_{bar}, C'_{bar}, [0.6, 0.5, 0.7])' \tag{12}$$

The disturbance is a mass variation, and will therefore influence the variation of the speed along z and the variation of the position z. For a mass mismatched to *2 kg* and for a constant reference to $z = 2$, the disturbance estimation can be observed on the figures 19 and 20.

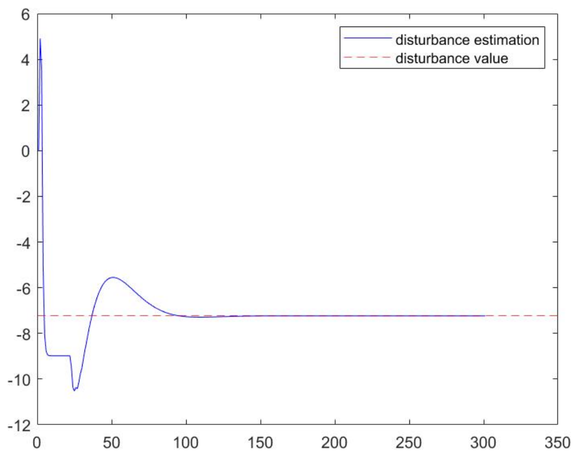


Figure 19: Disturbance estimation with fast poles.

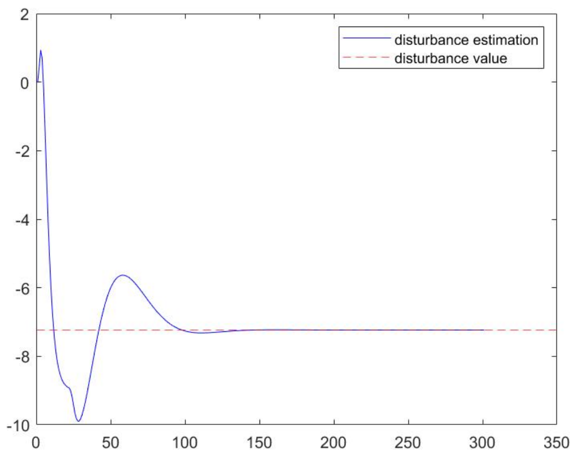


Figure 20: Disturbance estimation with slow poles.

We observe that the two sets of poles quickly estimate the disturbance. The fast pole set nevertheless esteems with more overshoot. We will see that this is a problem only when the mass of the rocket is reduced.

When the drone is overloaded, the path governed by the fast poles is more accurate, as we can see in figure 21 and 22.

On the other hand, when the rocket is lighter, the model with slower poles makes it possible to produce controllers up to 1.65 kg. While the model with the quick poles only allows 1.7 kg. A trade–off takes place between the need to be precise when gaining weight, or the need to be feasible even when the rocket weight decreases. We choose the first case, assuming that the rocket here could serve as a carrier of small loads.

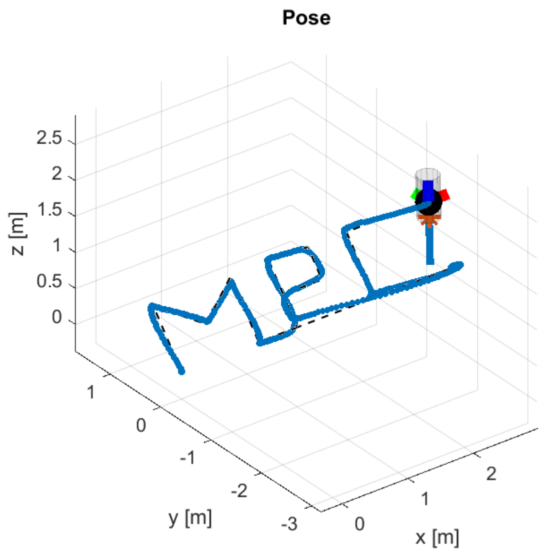


Figure 21: Path of the rocket controlled by a fast estimator.

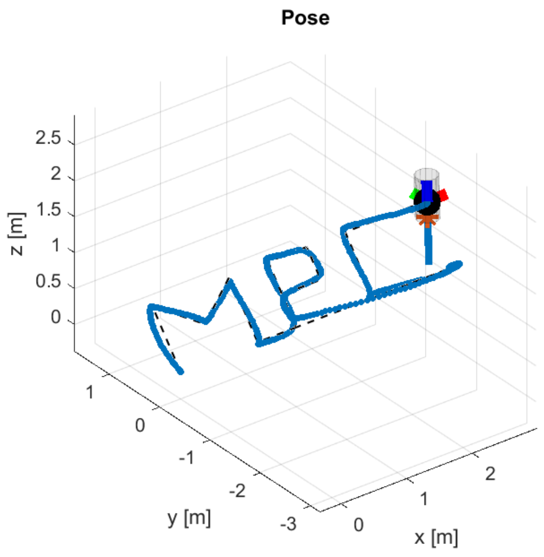


Figure 22: Path of the rocket controlled by a slow estimator.

Cost function tuning: There is also a little of re–tuning to do. The R value of the MPC z controller has to be changed. This concerns the P_{avg} input. If the rocket is loaded (mass mismatch), the rocket would need more throttle to reach the z references. That's why this input exceeds its upper limitation with the deliverable 4.1 tuning [4.3]. This value is changed to $R_{P_{avg}} = 0,5$.
This results into a less precise path reference tracking. There is a trade–off at stake: the bigger the $R_{P_{avg}}$ value is, the bigger the range of weight mismatches the rocket is capable to concede without having its P_{avg} going outside its bonds. But also, the lower the path tracking precision will be. Here we decide to have a bigger $R_{P_{avg}}$ to be able to undergo until 2 kilograms of weight. To balance the effect of lowering the P_{avg}, which will delay the time taken by z to reach its reference, we need to lower the cost of reaching the x and y references. This is done by lowering *Q(4,4)* of both y and y controllers (still of the deliverable 4.1) to 70 [4.3]. This gives back precision to the path following.

With and without offset–free controller: The figure 23 shows the difference in behaviour between a system with a disturbance regulator and one without. The disturbance is on the mass of the rocket, so it makes sense to only observe the offset along the z axis. The path without offset free tracking is therefore "flattened" according to z, while the path with it reacts by countering the disturbance and follows a trajectory identical to the path made in the deliverable 4.

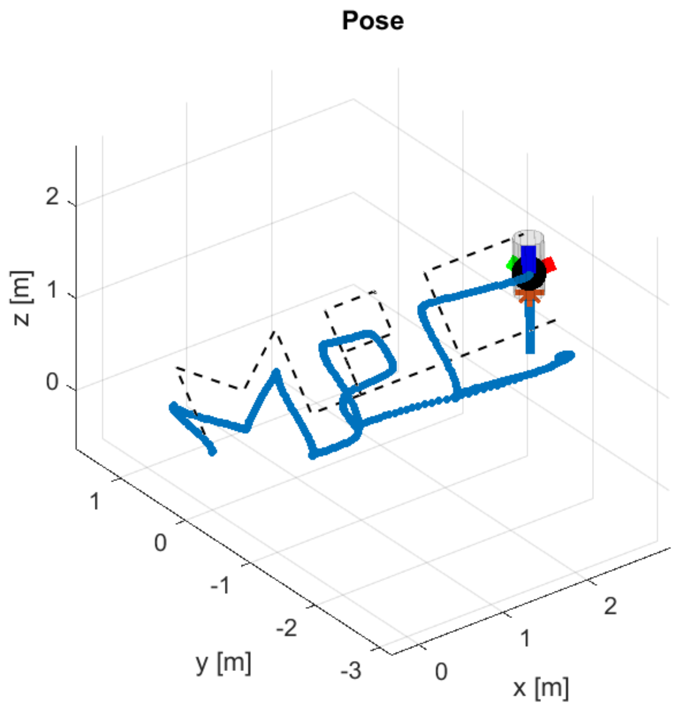


Figure 23: Path of the rocket with its mass mismatched to 2 kg, without using an offset–free regulator.

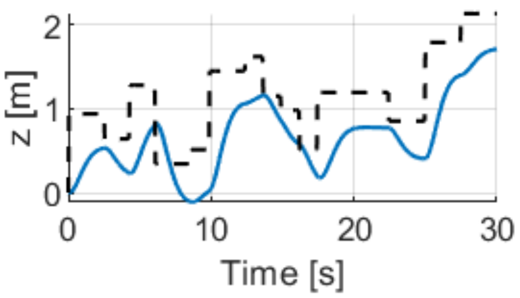


Figure 24: Plot of the z–coordinates of the rocket, with its mass mismatched to 2 kg, and without using an offset–free regulator.

6 Deliverable 6: Nonlinear MPC

For this task we didn't decompose the rocket dynamics into independent subsystems. Here, the Nonlinear MPC takes the full state x as input and input commands u are provided.

6.1 NMPC design procedure

We will now design a controller based on Non–Linear Model Predictive Control. This controller is a reference tracking controller. We will first focus on how we set up the problem so that it is applicable to a nonlinear dynamic model. Then we will see how to optimize the different free parameters.

Discretization The first difficulty with non linear MPC is to discretize the non–linear continuous model. We chose to use the *Runge–Kutta 4* method. This discretizing method is very precise, because it uses a second–order Taylor series expansion around a desired state for a given input. The gradient $\dot{x} = f(x, u)$ at a point x, for an input u is accessible from the physical plant with the matlab command *rocket.f(x,u)*. Depending on the sampling period h, the next sampled state is:

$$x_{k+1} = x_k + h * \left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right) \tag{13}$$

where

$$\begin{aligned}
k_1 &= f(x_k, u_k) \\
k_2 &= f(x_k + \frac{h}{2} * k_1, u_k) \\
k_3 &= f(x_k + \frac{h}{2} * k_2, u_k) \\
k_3 &= f(x_k + h * k_3, u_k)
\end{aligned}$$

Reference tracking The controller is still doing reference tracking. It is therefore necessary for each new reference to calculate the steady states and inputs. These variables will be used in the cost function to stretch the states and inputs concerned to the correct value. We therefore create an optimization problem similar to the 3.2 deliverable. With the only difference that we now use the Casadi solver instead of the Yalmip optimizer. The steady state is reached once the next step is equal to the previous step.

Constraints The constraints on the states related to the linearization of the system disappear. Indeed, these constraints were used to respect the approximations made to be able to model the system around steady state points. Now that we are doing Nmpc, these constraints are now useless. All that remains is to avoid the singularity in beta at 90 deg because of the Euler angles.

Cost function The last point to deal with is the cost function. As in the 3.2 deliverable, it is expressed in the expression (14).

$$V^*(x) = \sum_{i=0}^{N-1} \Big((x_i - x_s)^T Q (x_i - x_s) + (u_i - u_s)^T R (u_i - u_s)\Big) + (x_N - x_s)^T Q_f (x_N - x_s) \tag{14}$$

6.2 NMPC tuning parameters

The parameters that we have to tune are: the individual coefficients of the diagonal matrices Q and R, corresponding respectively to the weights of the associated states and inputs in the cost function that we want to minimize and the horizon length N.

Horizon length We know that N has an influence on the trade-off between computational cost and performance of the controller, we decided to use N=20 as higher values were not providing significant performance enhancement while taking too long to solve the problem.

Tuning Q and R We started with the identity matrix (in cost eq.(14)) for both then we tried to increase the weight of the coefficients associated with the position x,y,z (respectively Q(10,10), Q(11,11), Q(12,12)). We can see the direct effect of increasing those weights on the convergence time for each signal; however, increasing those too much led to overshoot which was not part of the desired behaviour. Next, the idea was to increase the weights of the Q matrix associated with the translation speed states v_x, v_y, v_z (respectively Q(7,7), Q(8,8), Q(9,9)) in order to avoid rapid impulse in direction changes creating curvy trajectories or problem associated with the limited thrust. Finally, in order to track the roll angle we increased the weight of the Q matrix ($Q(6,6)$) associated with the rotation angle around z.

Our finals tuning parameters are N=20, R=I and:

$$diag(Q) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 500 & 250 & 250 & 250 & 1125 & 1125 & 1125 \end{bmatrix} \tag{15}$$

Tuning Q_f Finally, for the terminal cost we used $Q_f = Q$ and we decided to drop the terminal constraints as the horizon length was sufficient to ensure that the states were recursively feasible while not being too computationally intensive. A more precise solution might have been to linearize the system around the steady-state and compute the LQR solution for the terminal set and terminal cost.

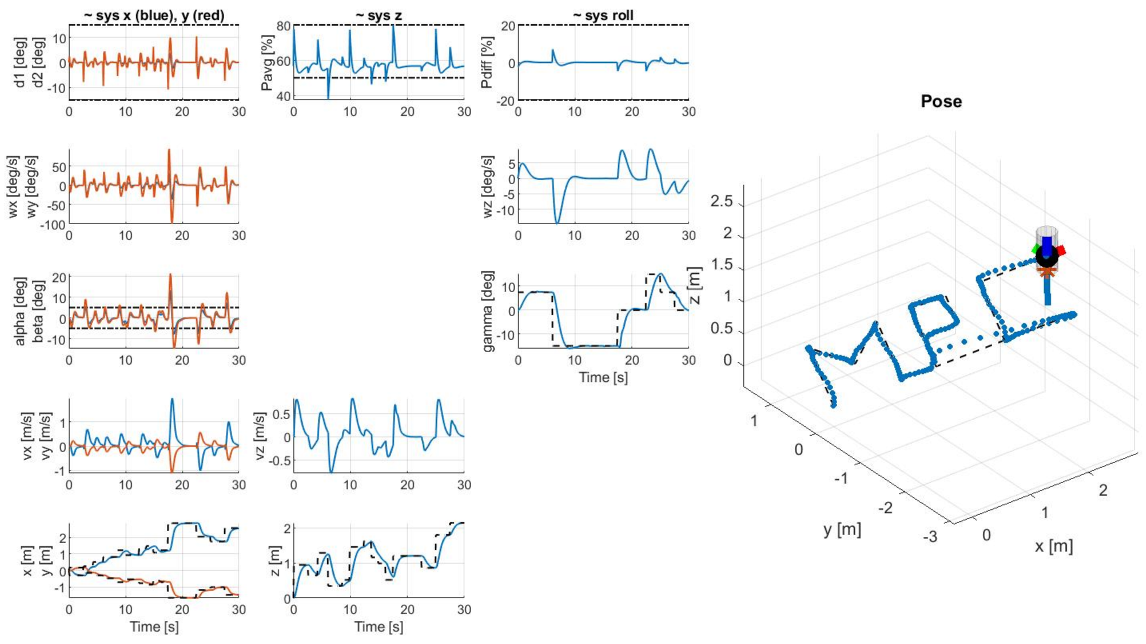


Figure 25: Path of the rocket using the NMPC controller

6.3 Linear MPC vs Nonlinear MPC comparision

When comparing the NMPC with LMPC we find multiple advantages for the Nonlinear controller. First, using the nonlinear controller allows to release the state constraints on the Euler angles associated with the linearization's limitations providing direct benefits in terms of path following. Furthermore, the Nonlinear controller exploits the full dynamical model of the rocket by directly taking as input the full state x skipping the step where we had to decompose the rocket dynamics into independent subsystems, it provides a faster response and allows to act on coupled dynamics as the previous subsystems may not be totally independent in reality. However, one could argue that the controller weights are harder to tune due to those coupling effects. Another advantage is that we do not need to linearize the system anymore, providing a model closer to reality with less approximations and more easily adaptable in case of modification of a component that could modify the dynamical model of the rocket.

To illustrate the performances improvements due to the NMPC not linearizing the dynamics, we compared the change in dynamics when the roll angle was constrained at 15° and then constrained at 50°. Comparing Fig 26 and Fig 27 we can see that the LMPC will have trouble for large roll angles as it will go away from the linearization points resulting in more approximations in the model. However, concerning the NMPC comparing Fig.28 and Fig.27 we can not see such changes in the dynamics, still presenting good performances, illustrating the impact of linearization of the model.

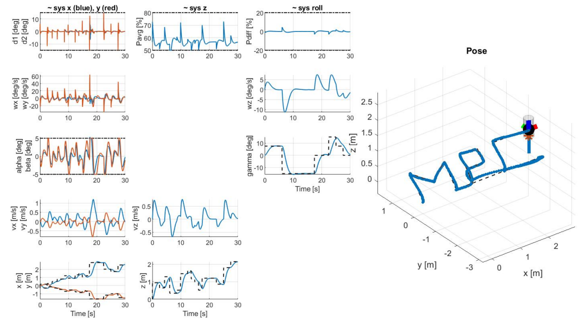


Figure 26: Path of the rocket with a LMPC controller and roll angle limitation of 15°

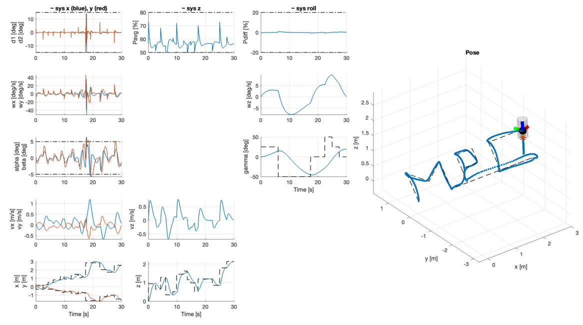


Figure 27: Path of the rocket with LMPC controller and a roll angle limitation of 50°

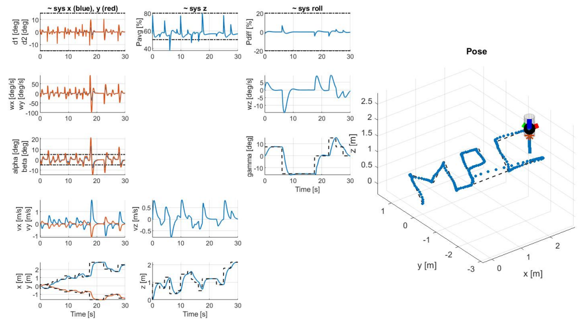


Figure 28: Path of the rocket with a NMPC controller and roll angle limitation of 15°

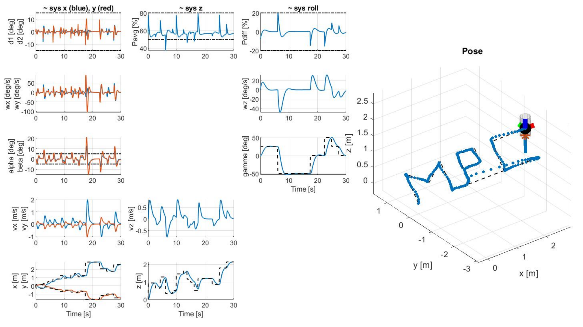


Figure 29: Path of the rocket with NMPC controller and a roll angle limitation of 50°