MPC report

MPC controller to fly a rocket prototype

Groupe 32
Baptiste Savioz 284253
Hendrik Hilsberg 274984
Constance Gontier 288443

Model Predictive Control (ME-425)
Prof. Colin Jones
EPFL
January 2022

Contents

1 Introduction and presentation of the system dynamics

The goal of the project is to develop an MPC controller for a provided model of a rocket and optimize its control. This is a prototype, and the rocket engine was replaced with two drone propellers that counter-rotate in order to cancel their torques in stationary flight. The mechanism that links the propellers to the body of the rocket enables them to be tilted. Let's look at the nonlinear model of the system dynamics. In order to obtain the model, we have two reference frames. One is attached to the center of mass of the rocket(called body frame), and the second is a fix world frame. Our system has 12 states, and the state vector is the following:

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

where $\mathbf{w}$ are the angular velocities about the body axes, $\boldsymbol{\varphi}$ represent the attitude of the body frame with respect to the world frame frame[1]. The velocity $\mathbf{v}$ and position $\mathbf{p}$ are expressed in the world frame.

$$\boldsymbol{\varphi} = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix}^T$$

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

$$\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$$

$$\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^T$$

The input $\mathbf{u}$ has four components. δ_2 and δ_2 are the deflection angles of the two servo motors, responsible for the orientation of the propellers. For the last two components, instead of using the power settings of propeller one and two, a choice was made to use the throttle average and the throttle difference. So $P_{diff} = P_2 - P_1$ and $P_{avg} = (P_1 + P_2)/2$.

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

δ_2 and δ_2 are limited to $\pm 15°$.To hold the throttle of each individual motor within [0, 100%] while P_{diff} might be up to $\pm$20%, we have to limit the valid range for P_{avg} to [20%, 80%][1].
While the force produced by the propellers depends on P_{avg}, the moment depends on P_{diff}. F(P_{avg}) and M(P_{diff}) apply along the following vector:

$$\mathbf{e}_F(\delta_1, \delta_2) = \begin{bmatrix} \sin\delta_1 \\ -\sin\delta_1 \cos\delta_2 \\ \cos\delta_1 \cos\delta_2 \end{bmatrix}$$

This vector represents the axis of the motor, that depends on the tilt applied by the servo motors. When tilted, the propellers produce another moment:

$${}_b\mathbf{M}_F = \mathbf{r}_F \times_b \mathbf{F}$$

We finally obtain the following force and moment:

$${}_b\mathbf{F} = F \cdot_b \mathbf{e}_F$$

$${}_b\mathbf{M} = M_\Delta \cdot_b \mathbf{e}_F +_b \mathbf{M}_F$$

The acceleration of the center of mass in the inertial (world) frame is given by[1]:

$$\dot{\boldsymbol{v}} = \boldsymbol{T}_{wb} \cdot_b \boldsymbol{F}/m - \boldsymbol{g}$$

where $\mathbf{T}_{wb}$ is a transformation matrix that let's us get from the body to world frame, m is the mass of the rocket, and $\mathbf{g}$ is the gravitational acceleration constant.
The angular acceleration, where $\mathbf{J}$ is the inertia of the flying object.

$$\dot{\boldsymbol{\omega}} = \boldsymbol{J}^{-1}(-\boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} +_b \boldsymbol{M})$$

$$\mathbf{g} = \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T$$

Finally, the derivative of the Euler angles:

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

Putting all these equations together leads to the dynamic equation of the rocket $\dot{x} = f(x, u)$.

2 Linearization

We are now provided with a non linear model for our rocket. However, it is common practice to linearize the model and compute an MPC controller on the linearized version, because it is simpler, computationally more affordable and many tools exist to deal with a linear problem.

In order to linearize the non-linear model, we need to have a point to linearize around. This is chosen to be a steady state point at the origin and computed with the provided Matlab function *rocket.trim()*. The only non-zero value is P_{avg} since the rocket needs some vertical propulsion to stay in the air.

$$x_s = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T, \quad u_s = \begin{bmatrix} 0 & 0 & 56.667 & 0 \end{bmatrix}^T$$

Now, we have our point to linearize around, we can derive the linear model with the provided function *rocket.linearize(xs,us)*. This gives us the following continuous linear model:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases} \tag{1}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9.81 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -9.81 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -55.68 & 0 & 0 & 0 \\ 0 & -55.68 & 0 & 0 \\ 0 & 0 & 0 & -0.104 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -9.81 & 0 & 0 & 0 \\ 0 & 9.81 & 0.1731 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C = I \in \mathbb{R}^{n_x \times n_x}, \quad D = 0 \tag{2}$$

By studying the matrices and especially the control matrix B, we understand that we can divide our system into four independent subsystems. This makes sense when we think of the point we linearize the system around. Indeed, the world frame and body frame were aligned, which effectively decouples some variables. Therefore the control variable d_1 rotates the rocket around $\vec{x}_B$, thus directly acting on ω_x (wrt. the body frame). Since world and body frames are aligned, it also acts directly and only on v_y (wrt to the world frame), making the rocket move along y. We have our first independent system *sys_ y* with the state variables: $x = (\omega_x, \alpha, v_y, y)$

and the control variable: $u = d_1$. α and y are states that are not directly accessed by the control variable d_1 but only their rates, it is thereby evident that we must include them into the sub-system. This is easily confirmed by looking at the state matrix A. The same reasoning can be applied for d_2 and the sub-system sys_x. The last two subsystems are straightforward. The control variable P_{avg} acts directly on $\dot{\vec{z_B}}$. It pushes the rocket up. Since $\vec{z_B}$ is aligned with $\vec{z_W}$, P_{avg} also acts directly on v_Z. Since v_z is modified directly by P_{avg}, so is z indirectly. For the last control variable P_{diff}, it is even simpler, since the variable it accesses are expressed in the same frame (body frame) as the state variable. Thus, this subsystem is not dependant on linearization.

We have now four independent subsystems because of where we did the linearization. If we had linearized around a point where the rocket had an angle for example, it wouldn't be possible to find the decoupled subsystem. Indeed, P_avg wouldn't only act on v_z but also on v_x and v_y, etc. Therefore we should add additional constraints on some state variables in order to stay close to the linearization point and minimize the error between the actual and the linear model. These constraints are detailed afterwards, but will typically concern the rocket angles.

Furthermore, it is important to notice that the linearized state and input variables were trimmed by (x_s, u_s). And we have then linearized our controller around this point[1].

$$\dot{x} \cong A(x - x_s) + B(u - u_s) \tag{3}$$

We have to take this into account when applying the computed optimal input sequence to the system. This is already carefully done by the code framework. We just need to be careful with the constraints, which means to correctly shift them.

3 Design MPC controllers for each sub-system

For this part, we design a recursively feasible, stabilizing MPC controller for each of dimension x, y, z and roll that can track step references with the continuous time linear model derived previously. However, the MPC framework we will use, uses a discrete time model which means we first need to discretize our linear model. This can be done with the matlab function *c2d(sys, Ts)*. As our linearization is approximate, we must place constraints on the maximal angles that the rocket can take so that our approximation is valid [1]:

$$|\alpha| \leq 5^\circ = 0.0873 \text{ rad}$$
$$|\beta| \leq 5^\circ = 0.0873 \text{ rad}$$

These constraints can be expressed as constraints on the state (for x and y) so we would satisfy: $F * x \leq f$:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} * x \leq \begin{bmatrix} 0.0873 \\ 0.0873 \end{bmatrix}$$

We also have constraints on all of the inputs of the system, we want : $M * u \leq m$. We can decompose the constraints on the input vector **u** into 4 constraints on the controllers of each dimension. Here are the constraints on the inputs of the controllers for the dimensions x and y:

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} * u \leq \begin{bmatrix} 0.26 \\ 0.26 \end{bmatrix}$$

For the z dimension, experiments with the prototype have shown that the rocket descends too quickly when less than 50% average throttle is given. For safety reasons, we want to limit the downward acceleration that can occur to the rocket, and we therefore require a minimum average throttle of 50% at all times: [1]

$$50\% \leq P_{avg} \leq 80\%$$

As we linearized around (x_s, u_s), we should not forget to substract the offset linked to this linearization. So $M * u \leq m$ is expressed:

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} * u \leq \begin{bmatrix} 80 - us \\ -50 + us \end{bmatrix}$$

For the roll dimension, we have a constraint on the input so that P_{diff} can be up to +/- 20 %. $M * u \leq m$ is expressed :

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} * u \leq \begin{bmatrix} 20 \\ 20 \end{bmatrix}$$

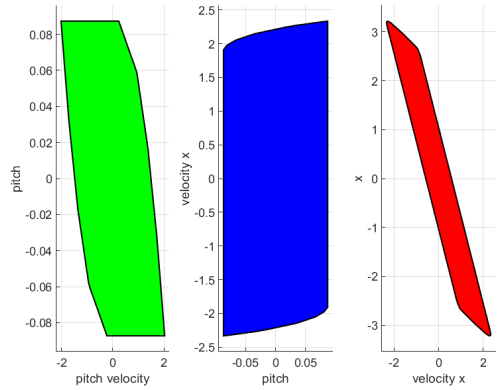
Deliverable 3.1

For this first part, we designed controllers for each subsystem that ensure the constraint satisfactions and track to the origin. We designed our controllers based on this standard MPC framework:

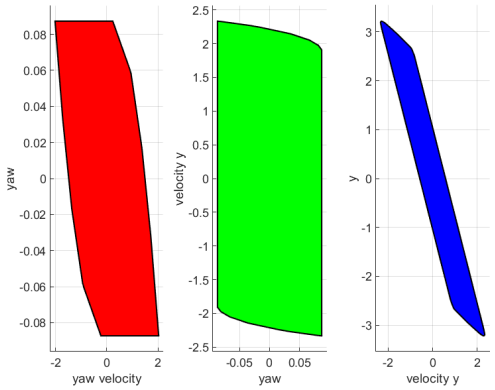
$$\begin{aligned} \min_{\mathbf{u}} \quad & \sum_{i=0}^{N-1} x_i{}^T Q x_i + u_i{}^T R u_i + x_N{}^T Q_f x_N \\ \text{s.t.} \quad & x_i \in \mathbb{X} \qquad\qquad i \in \{1, \dots, N-1\} \\ & u_i \in \mathbb{U} \qquad\qquad i \in \{0, \dots, N-1\} \\ & x_N \in \mathcal{X}_f \\ & x_{i+1} = A x_i + B u_i \end{aligned}$$

Figure 1: standard MPC framework[2]

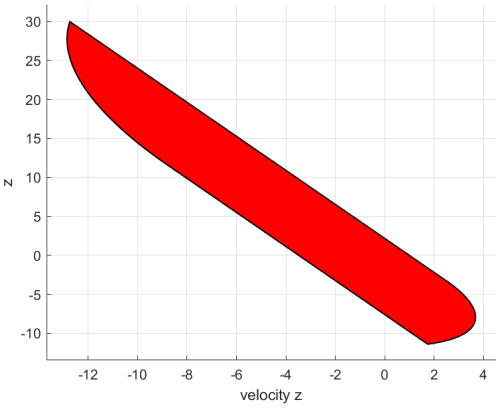
To ensure recursive constraint satisfaction, we used a terminal set. The goal is that the final predicted state ends in a controlled invariant set so that the system will be able to stay in it for all the time and therefore ensure recursive constraint satisfaction. To do so, we solved the *DARE* equation of an LQR controller thanks to the *dlqr* function of Matlab. This function gives us the optimal gain matrix K such that the state-feedback law minimizes the quadratic cost function so that we can compute a controlled invariant set for an LQR: $\chi_f = K * x$. We computed the terminal sets for dimensions x and y by projecting the polytopes in a 2D space. For the z dimension as well as for the roll angle, as they were already in 2 dimensions, we only used MPT3 to compute the invariant sets.
Here is a plot of the terminal invariant sets for each of the dimensions:



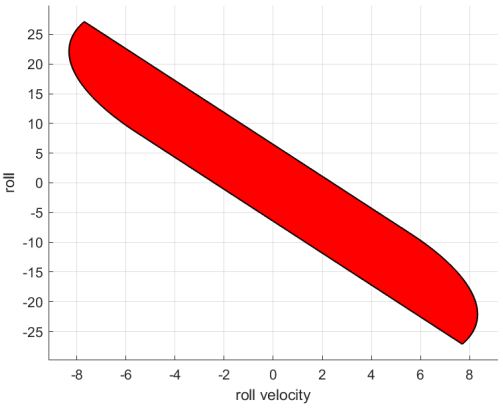
(a) projection of the terminal set for the x controller



(b) projection of the terminal set for the y controller



(c) Terminal set for the z controller



(d) Terminal set for the roll controller

Figure 2: Terminal set for each of the sub-system

Our controllers have to minimize the stage cost function: $l(x_i, u_i) = x_i^T * Q * x_i + u_i^T * R * u_i$, Q being the matrix that penalizes the error on the states and R being the matrix that penalizes the error on the inputs. What matters is the ratio between Q and R. We are aware of the fact that we should compare variables that have been normalized. By simplicity, we choose not to normalize R and Q but to tune them accordingly to the importance we want to give to penalizing errors on the inputs and on the states. To tune R and Q, we started a first simulation with two identity matrices. We have noticed that the inputs were heavily used with an overshot for x and y. This is why we chose to increase R for x and y until we had a satisfying simulation. On the other hand, we slightly increased the weight on Q for z and roll in order to have a faster response. We are satisfied with our tuning that allows a more reasonable use of the inputs with no significant deviation on the states. Here are the matrices we used in our code:

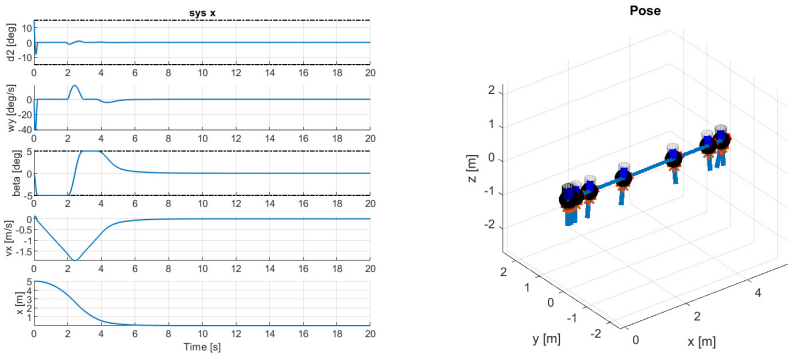
$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R = 100 * \begin{bmatrix} 1 \end{bmatrix}$$

H describes the horizon length (we have N the number of horizon steps : $N = \frac{H}{T_s}$). We have to find a compromise between not having too small of a value of H which would not allow a good prediction for the path, and not having too important of a value which would be too computationally heavy.
We noticed that a value of H between 2 and 2,5 corresponds to the limit of instability and that we could obtain good result from 3. We tried horizon up to 10 seconds without noticing significant improvement on the results.

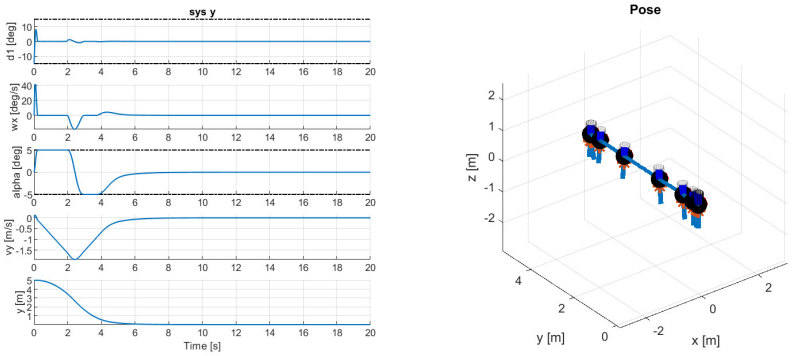
We finally chose H = 4 which seemed to be a good compromise between performance and computational cost.

To ensure stability, we introduced a terminal cost that we found by solving the equation of an LQR controller (the discrete time Ricatti's equation). For that we used the *dlqr* function of Matlab which gives us the P value to compute the terminal cost: $V_f = x_N^T * P * x_N$.

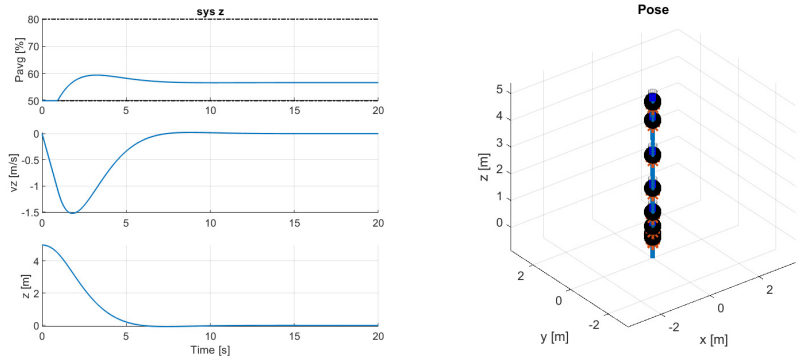
Here are the plots of the simulations for each of our sub-controllers starting stationary at 5 meters from the origin for x, y and z or stationary at 45° for roll. We are satisfied with our results: indeed the settling time is lower than 8 seconds and we have a smooth behavior for the variables (no overshoot anymore).



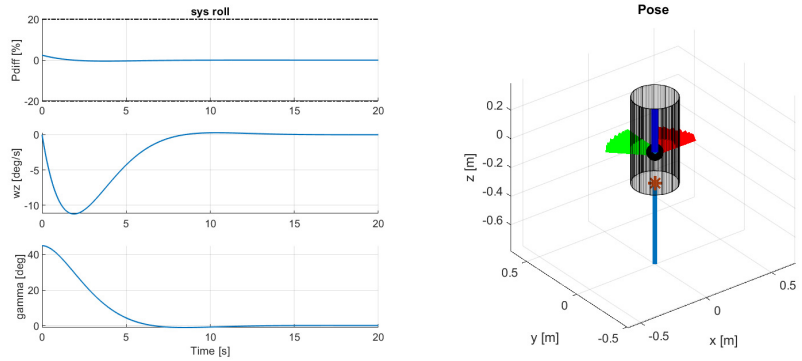
(a) simulation of the x sub-controller, starting at x_0



(b) simulation of the y sub-controller, starting at x_0



(c) simulation of the z sub-controller, starting at x_0



(d) simulation of the roll sub-controller, starting at x_0

Figure 3: Simulation of the four sub-controllers, starting at x_0

Deliverable 3.2

Now, we have a MPC framework for each subsystem, that should drive the states variables to the origin. The next step is to extend the framework in order to make the controller able to track a constant reference. To do so, we first need to reformulate the MPC controller with the delta formulation as seen in class:

$$\begin{aligned}
\min \ & \sum_{i=0}^{N-1} \Delta x_i^T Q \Delta x_i + \Delta u_i^T R \Delta u_i + V_f(\Delta x_N) \\
\text{s.t. } & \Delta x_0 = \Delta x \\
& \Delta x_{i+1} = A\Delta x_i + B\Delta u_i \\
& H_x \Delta x_i \leq k_x - H_x x_s \\
& H_u \Delta u_i \leq k_u - H_u u_s \\
& \Delta x_N \in \mathcal{X}_f
\end{aligned}$$

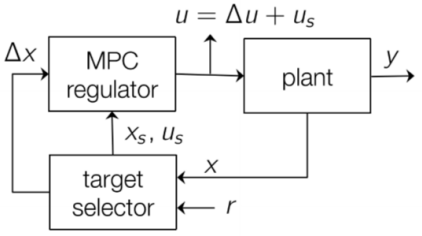


Figure 4: MPC framework with delta formulation for tracking[2]

We modified the function *setup_controller* for each subsystem according to the delta formulation, which means replacing x and u by $x - x_{ref}$ and $u - u_{ref}$ everywhere except for the state and input constraints. We should add the reference afterwards when we want to apply the computed input sequence to the rocket, but again the provided framework takes care of this for us. The next step is to actually compute the references x_{ref} and u_{ref}. By definition they are steady-state values which means they satisfy:

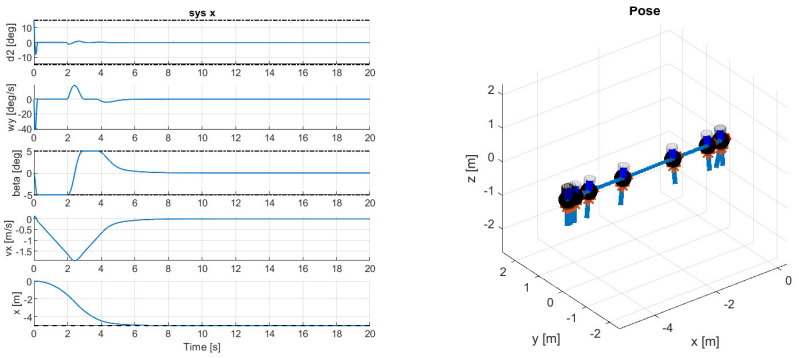
$$x_s = Ax_s + Bu_s \tag{4}$$

Then we want to track a reference, which means that we want the output to be equal to the reference:

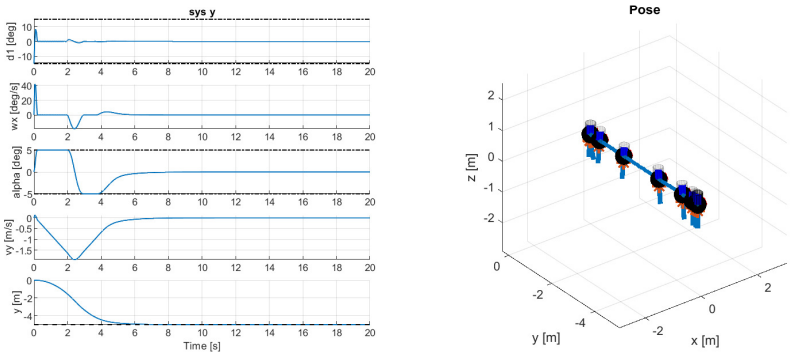
$$y = Cx_s + Du_s = ref \tag{5}$$

Finally we can setup this as a minimization problem where u_s is minimized with a quadratic cost function in order to satisfy the two constraints above (equations (4) and (5)) and the corresponding input constraints (we assumed the reference was feasible). We filled the function *setup_steady_state_target()* accordingly for each subsystem controller. It is important not to confuse the previously poorly defined (x_s, u_s) which is the point around which we linearized the rocket model, with (x_s, u_s) the computed steady state that the controller tracks as a reference.

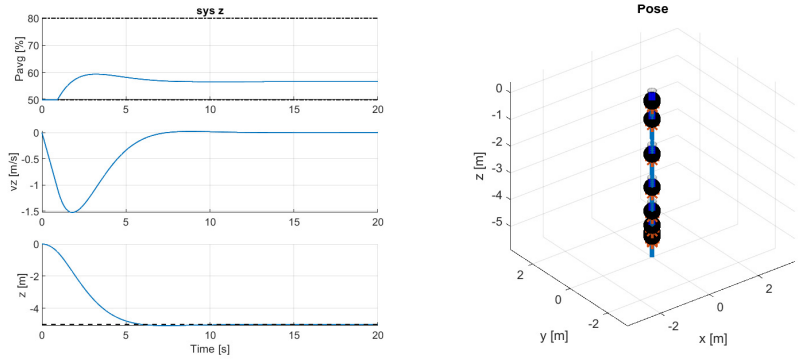
We now have our controllers for each sub-system completed so that they can track a constant reference. The horizon length and tuning for R and Q is no different than from the previous part. Indeed, it is the exact same problem (for the given x_0 and reference in the project) we're solving except that it's now shifted. We are happy with our results shown in figure 5. The controllers act just as expected and can now effectively track a given reference.



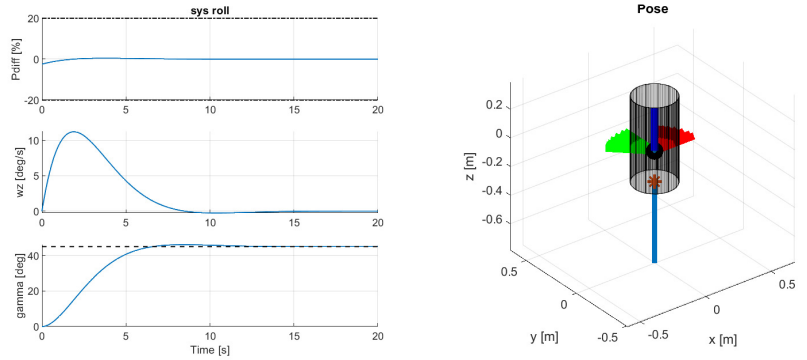
(a) X controller tracking a constant reference



(b) Y controller tracking a constant reference



(c) Z controller tracking a constant reference



(d) Roll controller tracking a constant reference

Figure 5: Each sub-controller tracking a constant reference

4 Simulation with nonlinear rocket

In this simulation the goal was to track a given path with the nonlinear rocket. However our controllers are linear so we had to adapt our weight matrices to avoid constraint violation and better tracking. We put more weight on the angular velocities and the position for the x and y controllers. The z controller has a higher weight on the position. For the roll controller we increased the weight on the γ angle. We also observed better tracking with a higher weight on the inputs δ_2 and δ_1 in the x and y controllers respectively. Finally, the time horizon $H = 4$ chosen previously remains a reasonable compromise between performance and computation and we decided to keep it. All these modifications on the Q and R matrices improved the performance of the rocket and the values are:

$$Q_x = \begin{bmatrix} 10 & & & (0) \\ & 1 & & \\ & & 1 & \\ (0) & & & 10 \end{bmatrix}, Q_y = \begin{bmatrix} 10 & & & (0) \\ & 1 & & \\ & & 1 & \\ (0) & & & 10 \end{bmatrix}, Q_z = \begin{bmatrix} 1 & 0 \\ 0 & 100 \end{bmatrix}, Q_{roll} = \begin{bmatrix} 1 & 0 \\ 0 & 80 \end{bmatrix}$$

$$R_x = 5, \quad R_y = 5, \quad R_z = 1, \quad R_{roll} = 1,$$

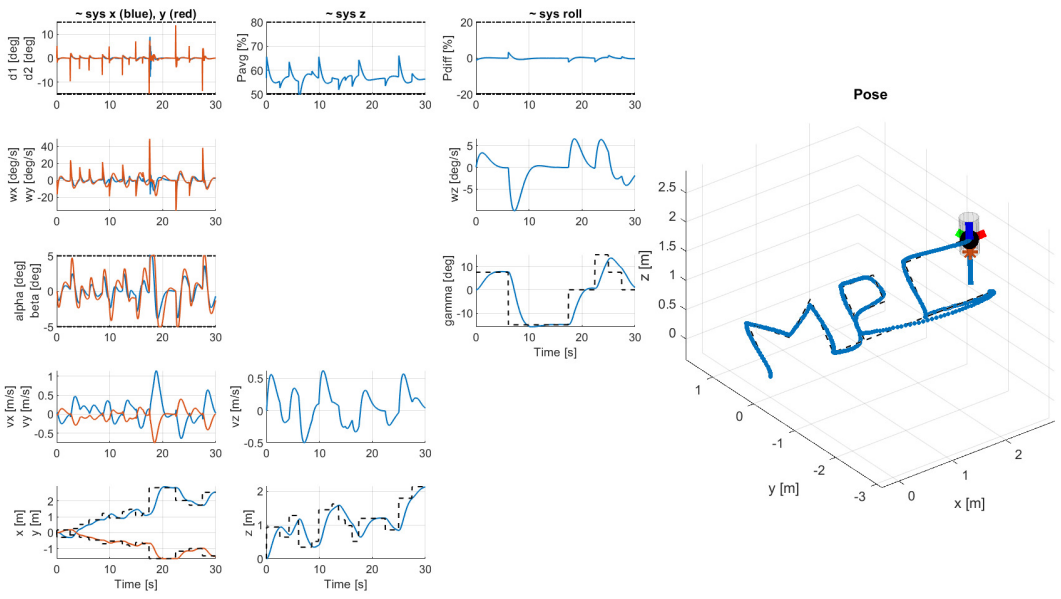


Figure 6: linear MPC controller tracking a path with the non linear rocket model

5 Offset-free tracking

In this section our goal was to cancel an unknown, constant perturbation in the z-direction. We had to adapt our z-controller in order to reject the disturbance and avoid having an offset, while tracking a reference. In the case below, we don't implement an observer and just use the controller from the last section, we can clearly see the offset created by the change of mass:

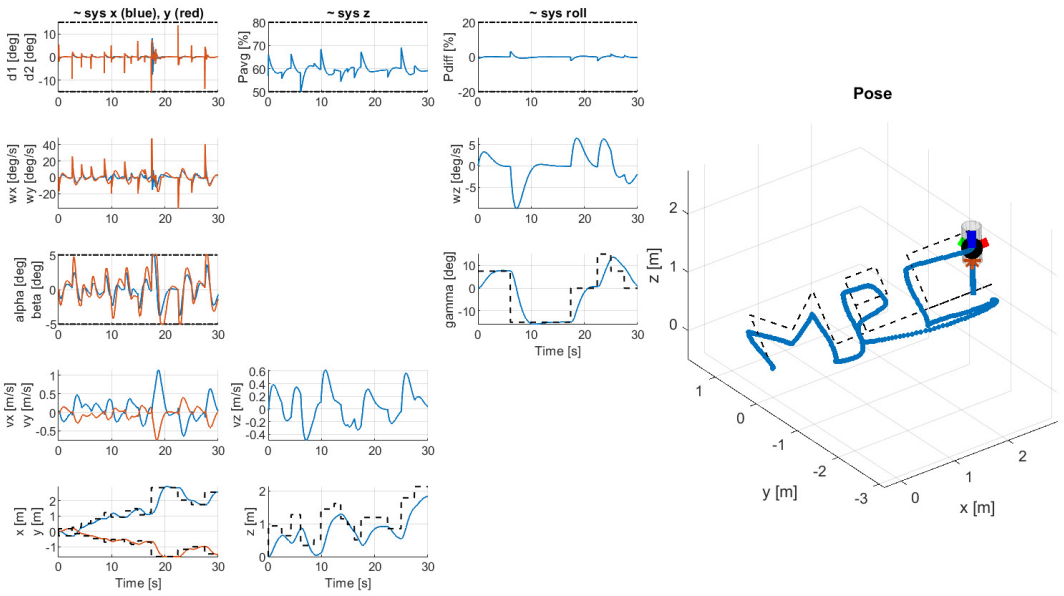


Figure 7: linear MPC controller with disturbance, without observer tracking references with the non linear rocket model

The new dynamics of the system are the following:

$$\mathbf{x}^+ = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{B}d$$

The presence of a disturbance influences the system and brings the rocket to diverge from its nominal dynamics. Our system has to converge to the set point. We are therefore going to estimate our states and the disturbance using the output measurement and the inputs that we applied. Computing these estimates will allow us to cancel the offset. We create an augmented model to include the estimated disturbance and states. We can then compute the bias induced by the disturbance and adjust our target condition and obtain a steady state target and input, to cancel the offset.

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

Figure 8: State and disturbance estimator based on the augmented model [2]

In our case, B_d is the same as B, and we don't have any disturbances in our output, so $C_d = 0$. The estimator gain is computed by placing its eigenvalues in order to minimize the variance of the estimation error. The eigenvalues shouldn't be greater than 1, and we chose the values by trying to get the best performance. We re-write our problem in Delta-formulation to include the calculated steady state and track the reference.

$$\min \sum_{i=0}^{N-1} \Delta x_i^T Q \Delta x_i + \Delta u_i^T R \Delta u_i + V_f(\Delta x_N)$$

$$\text{s.t. } \Delta x_0 = \Delta \hat{x}$$

$$\Delta x_{i+1} = A\Delta x_i + B\Delta u_i$$

$$H_x \Delta x_i \leq k_x - H_x x_s$$

$$H_u \Delta u_i \leq k_u - H_u u_s$$

$$\Delta x_N \in \mathcal{X}_f$$

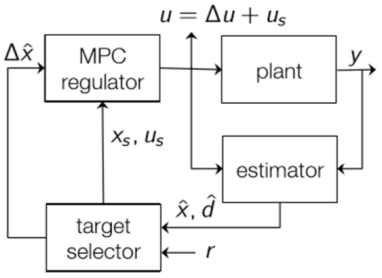


Figure 9: MPC delta formulation for tracking with an observer framework [2]

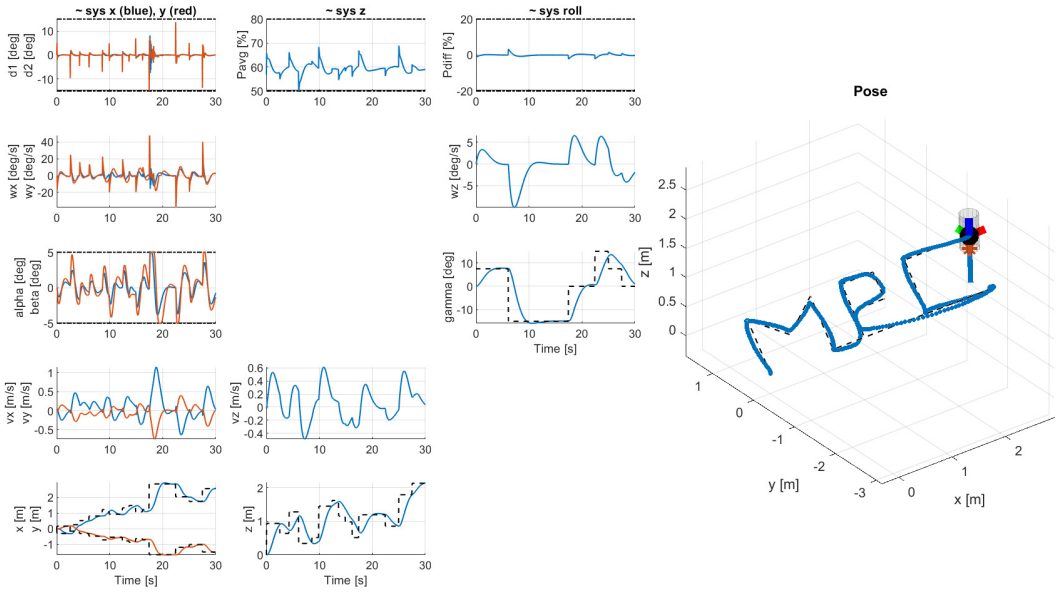


Figure 10: linear MPC controller with disturbance observer tracking references with the non linear rocket model

The tracking is satisfying. To obtain this result, instead of adjusting the weights of the matrices Q and R and time horizon H, we chose the eigenvalues of the observer to obtain good performances.

6 Nonlinear MPC

In this section we will derive a whole new controller for the rocket. We don't use a linear model approximation of our rocket anymore and instead we will tackle the non-linearity of the problem. The problem formulation is now simpler since we don't have to take into account any limitations from linear model. However, the complexity is now embedded in solving the minimization problem. This will be done using the solver *casadi*.

The constraints on α and β were there to keep the actual problem close to the linear model, thus we don't need them anymore. However, since we use Euler's angle to describe the rocket orientation, it has a singularity at $\beta = 90°$ and so we have to limit $|\beta| \leq 85°$.

You can find then figure framework for the MPC minimization problem we used 11. We chose a quadratic cost function with a null value at the provided reference for (x, y, z, γ) in order to track a constant reference:

$$\sum_{i=0}^{N-1} [(x_i - G \cdot ref)^T Q (x_i - G \cdot ref) + (u_i - u_s)^T R (u_i - u_s)] + (x_N - G \cdot ref)^T Q_F (x_N - G \cdot ref) \tag{6}$$

With G a matrix that links correctly the desired reference (x, y, z, γ) to the corresponding variables. The terminal cost Q_F is chosen to be the solution of the *DARE* for an infinite horizon LQR controller. Therefore we still need our previously derived discrete time linear model in order to compute Q_F. The steady-state input u_s was simply the output of the function *rocket.trim*. In order to be very rigorous we should have solved a minimization problem as we did in section 3 to find the exact (x_s, u_s) to track and not directly use the provided reference nor the u_s from the trim function. Nevertheless, by looking at the observation matrix C (and D), which implies exact state measurements, and noticing that u_s doesn't change no matter the position in space because the gravity acceleration and mass are supposed constant over time and space, one can understand that solving this minimization problem won't ever have a different solution than the reference and u_s we are using directly.

$$\begin{aligned} u^\star(x_0) = \text{argmin} \sum_{i=0}^{N-1} & l(x_i, u_i) + V_f(x_N) \\ \text{s.t.} \quad x_{i+1} &= f(x_i, u_i) \quad \forall i = 0, \dots, N-1 \\ g(x_i, u_i) &\leq 0 \quad \forall i = 0, \dots, N-1 \\ h(x_N) &\leq 0 \end{aligned}$$

Figure 11: NMPC minimization problem framework [3]

In the actual implementation we didn't use a terminal set. It wasn't required in the project description and since the computation time was already very high, we didn't want to add more complexity to the problem. The performance of our system were already satisfying and we had no problem with constraint satisfaction. Furthermore, the proper implementation of a terminal set in the non-linear case may be beyond our capabilities.

The dynamics are not the simple linear dynamics we derived earlier, but instead they are the non-linear dynamics of the actual rocket. Thankfully, the implementation of the computation of the non-linear dynamics was already provided in the framework and we could simply use the function *rocket.f(x,u)*. What was left to us was the discretization of the continuous time non-linear dynamic function. We chose to discretize the function using the Runge-Kutta 4 numerical integration.

Tuning of the parameters

Before starting tuning we decrease slightly the time horizon H to 3 seconds in order to be less computationally intensive. The change in performance was not perceptible. The only parameters that were left to choose were the weights on Q and R. First of all, we paid attention to the dimension and therefore size standard of the standard deviation in order to set meaningful weights. For example an angle won't vary more than a radian while p_{avg} has a range 30 times larger. Then we chose to heavily penalize deviation from the track variables

(x, y, z, γ) and limit angular velocities (ω_x, ω_y). We get the following matrices:

$$Q = \begin{bmatrix} 10 & & & & & & & & & & & & \\ & 10 & & & & & & & & & & (0) & \\ & & 1 & & & & & & & & & & \\ & & & 1 & & & & & & & & & \\ & & & & 1 & & & & & & & & \\ & & & & & 100 & & & & & & & \\ & & & & & & 1 & & & & & & \\ & & & & & & & 1 & & & & & \\ & & & & & & & & 1 & & & & \\ & & & & & & & & & 100 & & & \\ & (0) & & & & & & & & & 100 & & \\ & & & & & & & & & & & 80 & \end{bmatrix}, \quad R = \begin{bmatrix} 1 & & & (0) \\ & 1 & & \\ & & 0.1 & \\ (0) & & & 0.3 \end{bmatrix}$$

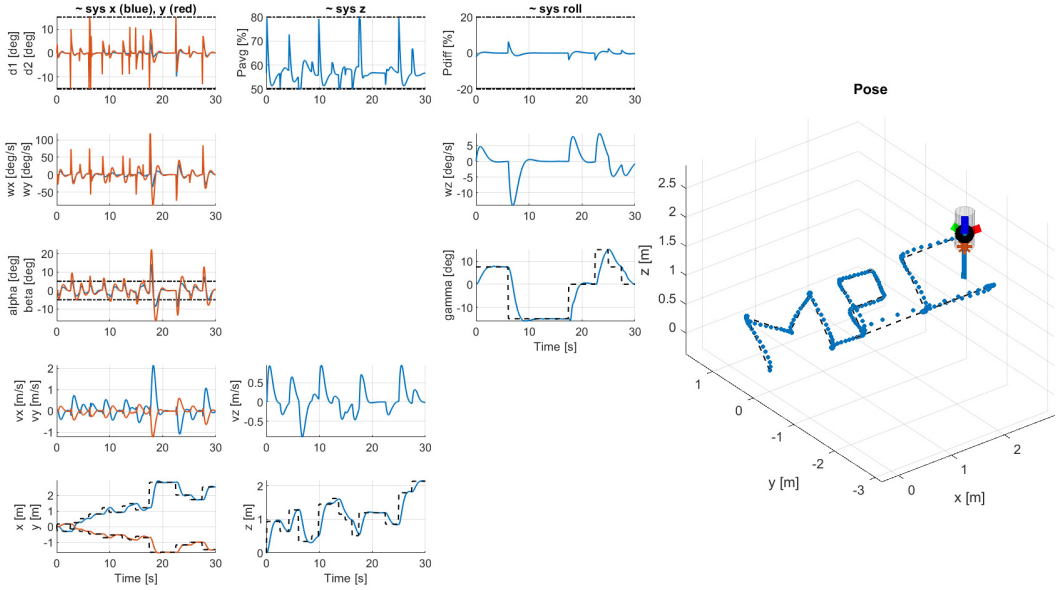


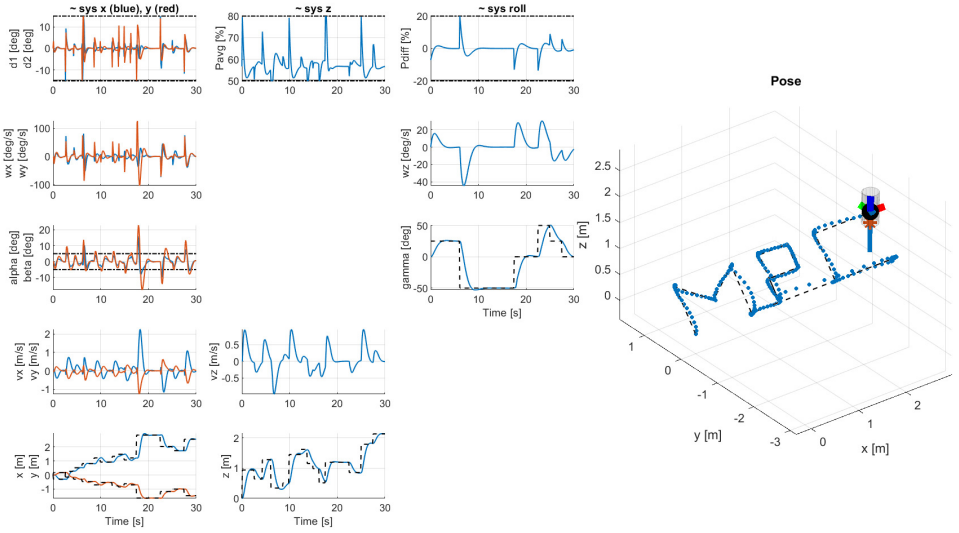
Figure 12: Non linear MPC controller tracking references with the non linear rocket model

The results of our non-linear controller are shown in figure 12. We are satisfied with them, the reference is tracked with intensive but decent use of the different inputs.

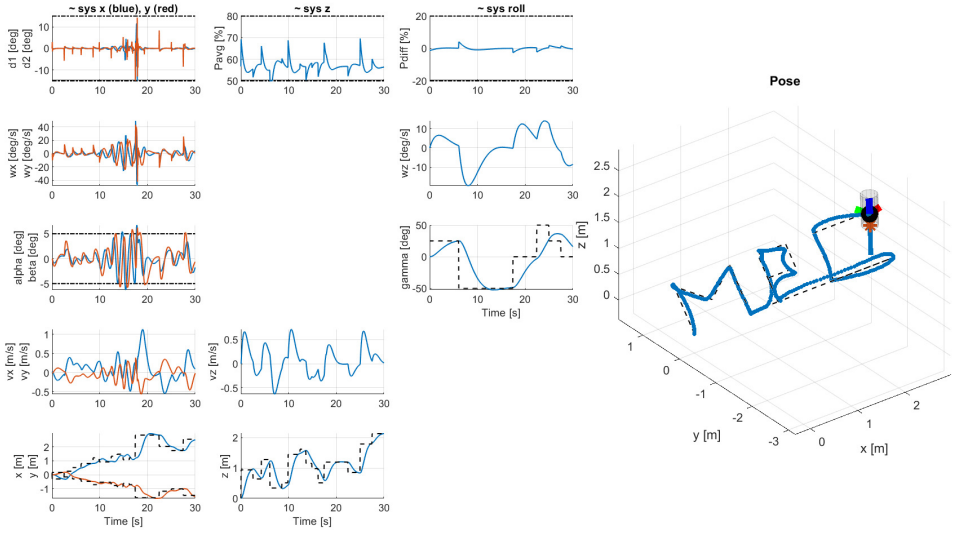
Comparison of linear and non-linear controller with higher roll angle

In a second experiment, we increased the maximum roll reference to $|\gamma_{ref}| = 50$ °. For the non linear controller, this modification didn't need any adjustment and the behaviour of the rocket was good. For the linear controller, the system has been linearized around a nominal point, the system becomes less accurate away from this nominal point. This is the case when the roll angle increases. This could lead to constraint violations and to avoid this situation we had to adjust some weights. In the roll controller, we diminished the gamma weight. Gamma is therefore going to change at a slower rate. By additionally increasing the weights of x and y in the controllers, we have better control of the position of the rocket. Furthermore, we increased the weights of ω_y, β, ω_x, and α

in the x and y controllers. This can be explained by the fact that we want α and β to respect the constraints and therefor to minimize their values faster.



(a) Non-linear MPC controller with $|\gamma_{ref}| < 50°$



(b) Linear MPC controller with $|\gamma_{ref}| < 50°$

Figure 13: Flight simulation with $|\gamma_{ref}| < 50°$ for both non-linear and linear controllers

Pros and Cons of non-linear controller versus linear controller

First of all, the non-linear controller offers some simplicity with the problem formulation. It requires less constraints. Indeed, the real constraints of the system (plus the numerical one) are: safety, actuator limit and performance constraints. However this simplicity comes with drawbacks. The solver used is more complex and requires more computational power. In real applications, this cannot be ignored anymore. In addition, finding

the real non-linear dynamics of a system can be more difficult than directly finding a linear approximation. In our case, it wasn't the case since we started from the non-linear dynamics and then we linearized them. The framework of the linear and non-linear system are very similar and thus is the complexity of the implementation. Furthermore, the results of the non-linear controller are better by a good margin and we don't have any problem operating far from the point where the model was linearized. Constraint satisfaction is more robust because the error between the model and the reality is effectively reduced. In addition, but we may loose the convexity of the problem when we use a non-linear model, and thus make it harder for the solver (or infeasible).

Finally, to summarize, the non-linear controller is better than the linear one in term of simplicity of the formulation, robustness and performances. However, it comes at the price of transferring the complexity of the problem to the solver, thus requiring more complex method, not guaranteed to converge. We may loose the convexity of the problem and the time computation can be too large to be used in practice.

7 Conclusion

This project helped us to understand the challenges of designing an MPC controller. We used a non-linear model of a rocket which was provided and had to control it. We started by simulating the behaviour of the rocket with different inputs to understand the dynamics of the system. Then we linearized the system around a trim point we computed (which is a steady-state) which let us split the system into 4 subsystems. This constitutes a first simple version of the systems we designed controllers for. Then we added constraints one by one and adapted the controller to respond to the new challenges: reject disturbance, track a reference with no offset and finally we developed a non linear controller using CASADI. This method was really interesting as it allowed to emphasize the challenges linked to each constraint of the system. We really enjoyed working on this project as we found it very playful thanks to the interface on which we can visualize the rocket and its dynamics. We are satisfied with our program and with our work as we learnt a lot about Model Predictive Control.

References

[1] Colin Jones. *Model predictive control : mini-project*. Lausanne: EPFL, 2021.

[2] Colin Jones. *Model predictive control : lecture 6 practical MPC*. Lausanne: EPFL, 2021.

[3] Colin Jones. *Model predictive control : lecture 10 non-linear MPC*. Lausanne: EPFL, 2021.