École Polytechnique Fédérale de Lausanne

Mini-Project:Development of a MPC controller to fly a rocket prototype.

MICRO-425 Model Predictive Control

Baiyu Peng 352384
Marie Ethvignot 314654
Marine Moutarlier 310703

April 24, 2025

Contents

1 Introduction

This project, divided in different parts, is used to make a rocket fly. In this project we did not model the rocket ourselves but used a given nonlinear model instead. The goal is to make it fly with drone racing propellers instead of the rocket engine. There are different important sets to this project. We will start with the system definition of the rocket, then we will go into details about forces and moments, linear and angular dynamics as well as attitude kinematics. All of these parameters are necessary to model the rocket.

2 System dynamics

The system dynamics is made out of differents parameters. The initial stages of crafting our controller are pivotal, beginning with the fundamental task of constructing a model for the system dynamics grounded in physical principles. Our focus will be on deriving a nonlinear model, taking into account the body frame and the world frame. This process allows us to formulate a comprehensive 12-state description of the system, encapsulated within a singular vector.

$$\boldsymbol{x} = \begin{bmatrix}\boldsymbol{\omega}^T & \boldsymbol{\phi}^T & \boldsymbol{v}^T & \boldsymbol{p}^T\end{bmatrix}^T, \quad [\boldsymbol{x}] = \begin{bmatrix}\text{rad/s} & \text{rad} & \text{m/s} & \text{m}\end{bmatrix}^T$$

where

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

represent the angular velocities about the body axes.

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

represents the Euler angles for the dimensions x, y, z, and finally the velocity v

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

and the position p

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

expressed in the world frame as $v = \dot{p}$.

The input of the model is given by

$$\boldsymbol{u} = \begin{bmatrix}\delta_1 & \delta_2 & P_{\text{avg}} & P_{\text{diff}}\end{bmatrix}^T, \quad [\boldsymbol{u}] = \begin{bmatrix}\text{rad} & \text{rad} & \% & \%\end{bmatrix}^T$$

where δ_1 and δ_2 are the deflection angles of servo 1 (about **x**b) and servo 2 (about rotated **y**b), respectively, up to $\pm 15 \circ$ (= 0.26 rad) and $P_{\text{avg}} = \frac{P_1+P_2}{2}, \quad P_{\text{diff}} = P_2 - P_1$.

To maintain the throttle of each individual motor within the range [0, 100%] while allowing P_{diff} to be up to ±20%, we must limit the valid range for P_{avg} to [20%, 80%].

The combined propellers produce a thrust force of magnitude $F(P_{\text{avg}})$ and a differential moment of magnitude $M_\Delta(P_{\text{diff}})$. When the axis is tilted, the thrust vector introduces another moment about the center of mass, $\mathbf{bMF} = \mathbf{rF} \times \mathbf{bF}$, so that the resulting force and moment acting on the body are expressed in the body frame. We use them to compute the acceleration of the center of mass and the angular dynamics in the frame:

$$\dot{v} = T\mathbf{bF} - g\mathbf{wb} - \frac{\mathbf{wb} \times \mathbf{m}}{m}$$

$$\dot{x} = J^{-1}\left(-\boldsymbol{\omega} \times J\boldsymbol{\omega} + \mathbf{M}\right)$$

where $T_{\text{wb}}(\psi)$ is the direction cosine matrix that transforms a body vector to a world frame vector. The rate of change of the Euler angles is a function of the attitude $\boldsymbol{\phi} = \begin{bmatrix}\alpha & \beta & \gamma\end{bmatrix}^T$, expressing the rotating body frame, and the angular velocity in the body frame according to the kinematic differential equation:

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

When we put all of this together, we get the dynamic equations for the rocket: $\dot{x} = f(x, u)$.

We simulate the rocket with various step inputs to confirm that the dynamics responds as expected. We are trying out if the rocket behaves as it should. By that we manipulate the different values of the vector u.

The control input vector **u** is defined as follows:

$$\mathbf{u}^T = \begin{bmatrix}\text{deg2rad}(u(1)) & \text{deg2rad}(u(2)) & u(3) & u(4)\end{bmatrix}$$

This implies that:

u(1) : Pitch angle (converted from degrees to radians, with values between [-15,15])
u(2) : Roll angle (converted from degrees to radians [-15,15])
u(3) : Thrust along the z-axis, with values between [50,80] (in %)
u(4) : Yaw rate, with values between [-20,20] (in %)

This control input is used to simulate the rocket's behavior, providing specific values for pitch, roll, thrust, and yaw rate.

After a bit of trial and error and simulation, we realized that,

1. **To make the rocket ascend vertically without tipping over** we can set the u to :

$$\mathbf{u}^T = \begin{bmatrix}0 & 0 & 80 & 0\end{bmatrix}$$

   u(1): Pitch angle is set to 0. This means that the rocket is not pitched upward or downward; it maintains a level orientation about its lateral axis.

   u(2): Roll angle is set to 0. This ensures that the rocket doesn't roll about its longitudinal axis. It remains stable and does not tilt to the side.

   u(3): Thrust along the z-axis is set to 80. This provides sufficient upward force to overcome gravity and make the rocket ascend vertically.

   u(4): Yaw rate is set to 0. There is no rotational motion around the vertical axis, ensuring that the rocket maintains a straight trajectory without any spinning.

2. **To make the rocket descend vertically without tipping over** we can set the u to :

$$\mathbf{u}^T = \begin{bmatrix}0 & 0 & 50 & 0\end{bmatrix}$$

   Same as before, just u(3) is set to the minimum value 50. Setting this to 50 represents a downward thrust. This component counteracts the gravitational force, causing the rocket to descend.

3. **Rotate its body along x**

$$\mathbf{u}^T = \begin{bmatrix}\text{angle\_x} & 0 & \text{thrust} & 0\end{bmatrix}$$

u(1): Pitch Angle - Set this to the desired pitch angle, representing the rotation about the x-axis. The value should be between [-15,15] and will change the amount of spinning or the direction (clockwise or counter clockwise)

u(2): Roll Angle - Set to 0, indicating no roll motion.

u(3): Thrust along the z-axis - Adjust as needed to maintain the desired rotation. The value shoiuld be between [50,80]

u(4): Yaw Rate - Set to 0, as we are not introducing yaw rotation in this case.

4. **Rotate its body along y**

$$\mathbf{u}^T = \begin{bmatrix} 0 & \text{angle\_y} & \text{thrust} & 0 \end{bmatrix}$$

Same as before but here we change u(2) instead of the u(1), this will change the direction of the rotation and create a rotation around y

5. **Rotate its body along z**

$$\mathbf{u}^T = \begin{bmatrix} 0 & 0 & \text{thrust} & \text{angle\_z} \end{bmatrix}$$

(u(1) : Pitch Angle - Set to 0, indicating no pitch motion.

(u(2) : Roll Angle - Set to 0, indicating no roll motion.

(u(3) : Thrust along the z-axis - Adjust as needed to maintain the desired rotation. The value should be between [50, 80].

u(4): Yaw Rate - Set this to the desired yaw rate, representing the rotation about the z-axis. The value should be between [-20, 20] and will change the amount of spinning or the direction (clockwise or counterclockwise).

6. **Flying along x,y,z**
Combining all the elements of u will create a flying movement in all the axes. Setting u(1) to 0 will result in flying only along x and z, while setting u(2) will do the same but only for y and z. To fly only along the z axis, see the first item about ascending and descending. Any other u vector with non zero variables will result in a flight along the x, y and z axis.

7. **Hovering**

To achieve hovering, where the rocket counters the force of gravity, we need to carefully balance the thrust and other control parameters. The control input vector u can be set to:

$$\mathbf{u}^T = \begin{bmatrix} 0 & 0 & 62.5 & 0 \end{bmatrix}$$

u(1): Pitch Angle - Set to 0, indicating no pitch motion.

u(2): Roll Angle - Set to 0, indicating no roll motion.

u(3): Thrust along the z-axis - After a bit of trial and error, the rocket stays in place for around a trust of 62.5

u(4): Yaw Rate - Set to 0, indicating no yaw rotation.

3 Linearization

In the first part of the project, we are going to control a linearized version of the rocket.

We start by find a trim point. A trim point is the steady-state condition where the system is in equilibrium. In MATLAB, the **trim** function is commonly used for this purpose. The trim function adjusts the inputs u to find a state where the dynamics f(x,u) are in equilibrium, i.e., $f(xs, us) = 0$ After execution of the trim function with our system, we obtain:

$$\mathbf{x_s}^\top = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{u_x}^\top = \begin{bmatrix} 0 & 0 & 56.6667 & 0 \end{bmatrix}$$

The **linearize** function is then used to linearize the nonlinear rocket model around the trim point. The resulting linearized state-space representation around the trim point is given by $\dot{x} = A(x - x_s) + B(u - u_s)$, where A and B are the state and input matrices.

We can separate the linearized rocket into different sub-systems because the rocket's motions along the different axes are independent. Indeed, from a physical point of view it is clear that, for example, when the first servo motor is actuated and rotates around the x_b axis, it cannot induce a change of position along this same axis. Indeed, a change in the deflection angle δ_1, which means a rotation of the first servo motor around the x axis of the body (which also implies a change of the first Euler angle), only induces a change in angular velocity along the y_b axis, the position and velocity along the y axis (in the world frame). This yields that one independent sub-system (sys_y in our code) will concern the change of the rotation around the x_b axis, the corresponding Euler axis, the position and velocity along the y axis, depending on these four states and the deflection angle δ_1 of the first servo motor, as input. In a similar way, it is clear, from a physical point a view, that when we change the rotation of the second servo (that rotates around the y_b axis) it will only induce a change in the angular velocity around y_b, the corresponding Euler angle and the position and velocity along the x axis of the world frame. As for sys_z, the z velocity is directly controlled by the average throttle which produces the lifting force. For sys_roll, the z rotating velocity is proportional to the difference of the two throttles, which produce torques on the drone.

From the complete linearized system, we can extract four independent sub-systems and create a different MPC for each of the subsystems.

We compute the four independent systems above using this function:

[sys_x, sys_y, sys_z, sys_roll] = rocket.decompose(sys, xs, us) we find:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

For sys_x:

$$\begin{bmatrix} \dot{w_y} \\ \dot{\beta} \\ \dot{v_x} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 9.81 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} w_y \\ \beta \\ v_x \\ x \end{bmatrix} + \begin{bmatrix} -55.68 \\ 0 \\ 9.81 \\ 0 \end{bmatrix} \delta_2 \text{ and } y = x$$

For sys_y:

$$\begin{bmatrix} \dot{w_x} \\ \dot{\alpha} \\ \dot{v_y} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -9.81 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} w_x \\ \alpha \\ v_y \\ y \end{bmatrix} + \begin{bmatrix} -55.68 \\ 0 \\ -9.81 \\ 0 \end{bmatrix} \delta_1$$

For sys_z:

$$\begin{bmatrix} \dot{v_z} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} v_z \\ z \end{bmatrix} + \begin{bmatrix} 0.1731 \\ 0 \end{bmatrix} P_{\text{avg}}$$

For sys_roll:

$$\begin{bmatrix} \dot{\omega_z} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \omega_z \\ \gamma \end{bmatrix} + \begin{bmatrix} -0.104 \\ 0 \end{bmatrix} P_{\text{diff}}$$

4 Design MPC Controllers for Each Sub-System

4.1 Deliverable 3.1

Recursive constraint satisfaction We start by outlining the state and control constraints associated with each subsystem, with all angles expressed in radians.

x-subsystem

$$\begin{aligned} |\beta| &\leq 0.1745, \\ |\delta_2| &\leq 0.26, \end{aligned} \tag{1}$$

y-subsystem

$$\begin{aligned} |\alpha| &\leq 0.1745, \\ |\delta_1| &\leq 0.26, \end{aligned} \tag{2}$$

z-subsystem

$$50 \leq P_{ave} \leq 80, \tag{3}$$

roll-subsystem

$$|P_{diff}| \leq 20, \tag{4}$$

To ensure recursive constraint satisfaction for α and β under other input constraints, we introduce the concept of terminal set and enforce the terminal set constraint in the MPC problem. The terminal set $\S_f$ is induced by a LQR controller $u(x) = Kx$, which is solved by Matlab. We run the pre-set algorithm to compute the maximum invariant set of the LQR controller. The 2-D visualization of the state constraints and the identified invariant set of x-subsystem and y-subsystem are presented below. The state constraints should be unbounded on the wy direction, but due to some unknown bug in Matlab plotting function it's mistakenly bounded. Note that since there is no state constraint for the other two subsystems, they don't require terminal set constraints.

Parameters choice The main parameters involved in the MPC are Q, R and H. For Q, we use a diagonal matrix for every subsystem.

First we study the x-subsystem. We initially assign a weight of 0.5 to β. Given that the position x is typically one order of magnitude greater than the angles, we choose a weight of 5 for x. The

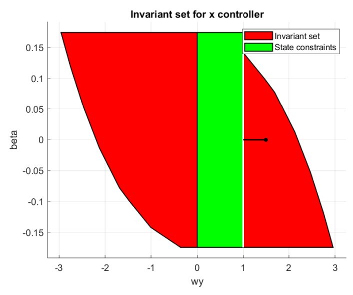

Figure 1: The invariant set of x-subsystem. The state constraints should be unbounded also in negative wy direction. But due to some unknown issues in the Matlab plotting function, there is a fake bound which actually doesn't exist.

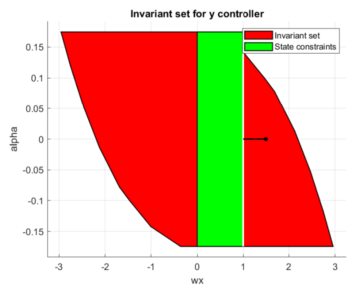

Figure 2: The invariant set of y-subsystem.

(angular) velocities have a more direct and sensitive influence on the stability of the entire rocket system. Therefore for w_y we choose 1, and for v_x we choose 10, both higher than the weights for the positions. The input penalty R is set to as 1.

Regarding the horizon H, to achieve faster computation speed, it is preferable to use a shorter H if the performance is not compromised. Hence we employ a horizon of 1s, equivalent to 20 discrete steps. We also tested 0.5s but it often failed to find a feasible solution.

The y-subsystem shares a similar structure as the x-subsystem. Therefore we employ the same Q, R and H settings.

The z-subsystem's control input is P_{ave}, which falls within the range of 50-80. Thus we first reduce the penalty for control to $R = 0.0001$. For Q, we follow the setting of the other subsystems and choose 10 for velocity and 5 for position. The horizon remains 1.

The roll-sub system's control input is P_{ave}, which is between -20-20. Thus we similarly reduce the penalty for control to $R = 0.0001$. For Q, we also choose 10 for velocity and 5 for position. The horizon remains 1.

Trajectory plots

The open-loop and closed-loop trajectories for all the four subsystems are shown below. Our closed-loop controllers achieve good performance for all the four subsystems.

4.2 Deliverable 3.2

Design procedure and parameters To track the desired output r, we have to find a steady state and input, which is accomplished by solving the following constrained optimization problem.

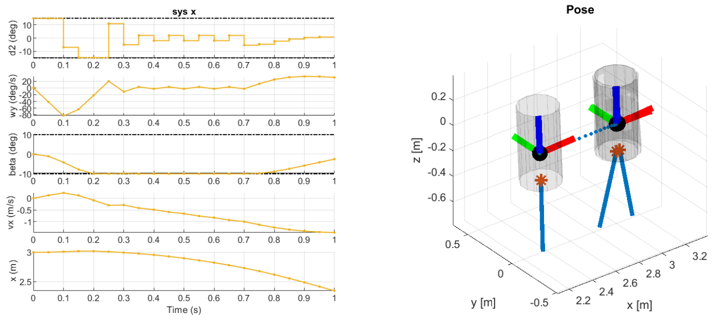

Figure 3: The open-loop plots for x-subsystem.

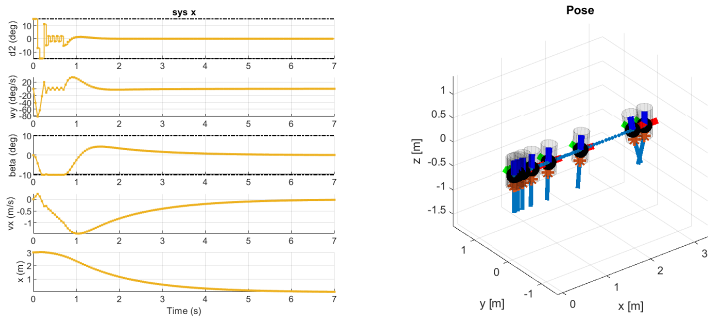

Figure 4: The closed-loop plots for x-subsystem.

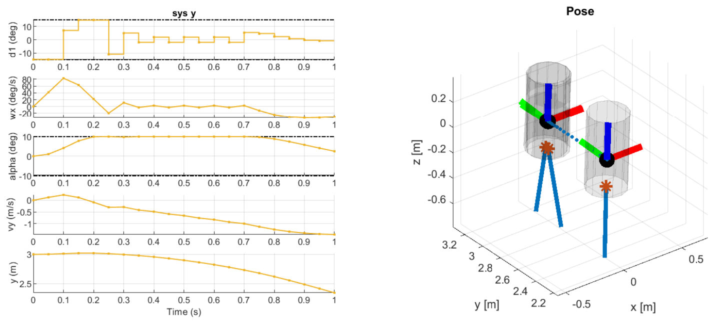

Figure 5: The open-loop plots for y-subsystem.

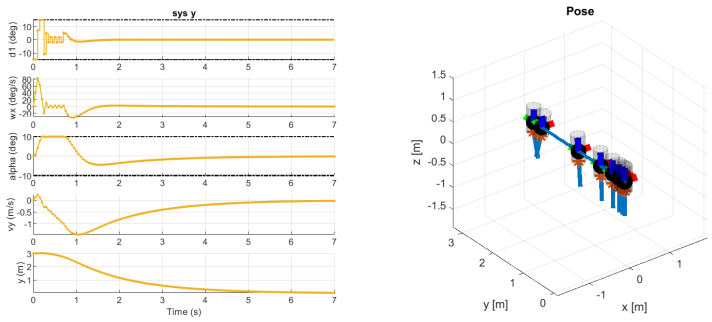

Figure 6: The closed-loop plots for y-subsystem.

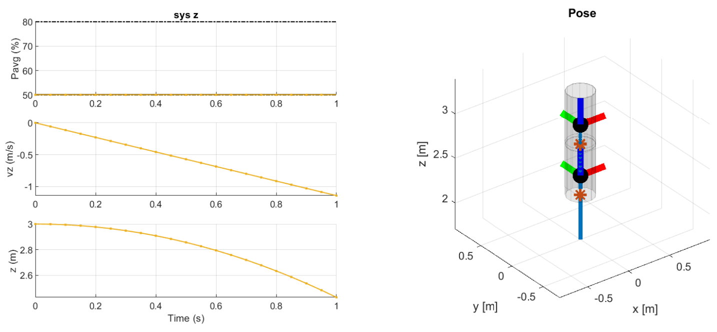

Figure 7: The open-loop plots for z-subsystem.

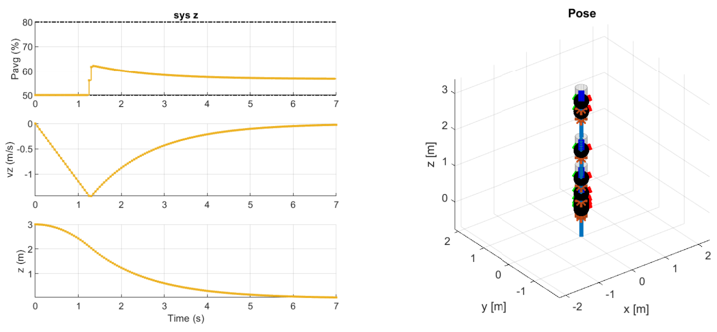

Figure 8: The closed-loop plots for z-subsystem.

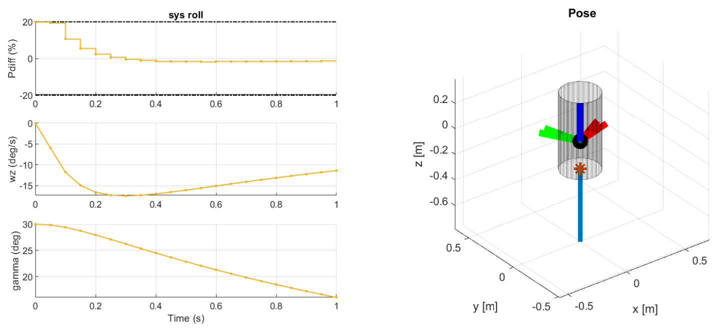

Figure 9: The open-loop plots for roll-subsystem.

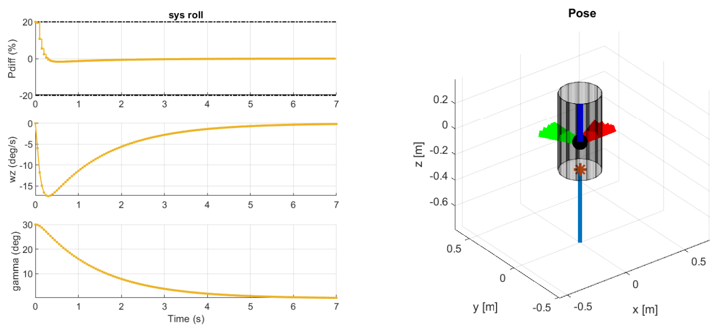

Figure 10: The closed-loop plots for roll-subsystem.

$$\begin{aligned}
&\max_{x_s,u_s} u_s^2 \\
&\text{s.t. } x_s = Ax_s + Bu_s, \\
&r = Cx_s, \\
&\text{and other state and inputconstraints as defined in the last section.}
\end{aligned} \tag{5}$$

With the obtained steady state and input, the original MPC problem can be reformulated into:

$$\begin{aligned}
&\max_{U} \sum_{i=1}^{N-1} \Big[(x_i - x_s)^T Q (x_i - x_s) + (u_i - u_s)^T R (u_i - u_s)\Big] + (x_N - x_s)^T P (x_N - x_s) \\
&\text{s.t. The state and input constraints as defined in the last section.}
\end{aligned} \tag{6}$$

where N is the total predicted time steps, P is the Raccati matrix. Note that as indicated in the project description, the terminal set constraint is dropped out in this task.

All the parameters for this tracker are consistent with the settings for the regulator , since the plant remains the same and the experimental tracking performance is good.

Trajectory plots The open-loop and closed-loop trajectories for all the four subsystems are depicted below. Our closed-loop controllers demonstrate precise tracking performance for all the four subsystems.

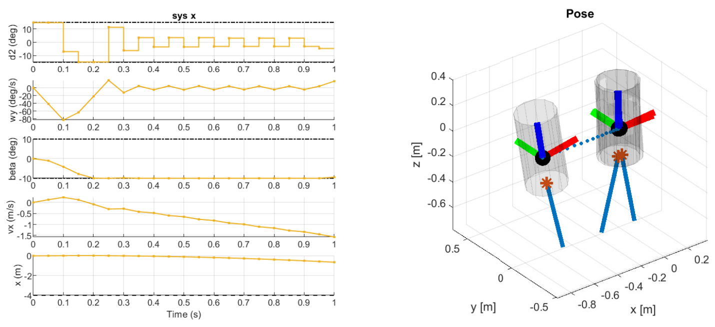

Figure 11: The open-loop tracking plots for x-subsystem.

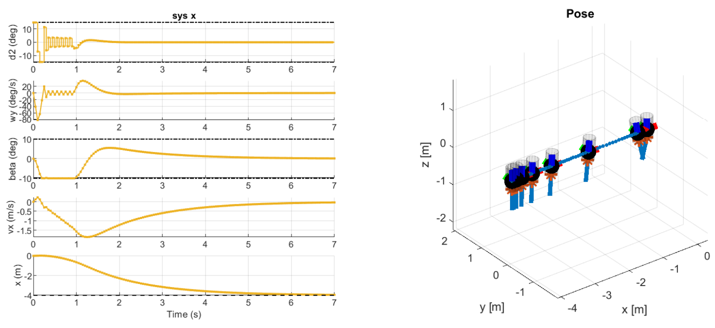

Figure 12: The closed-loop tracking plots for x-subsystem.

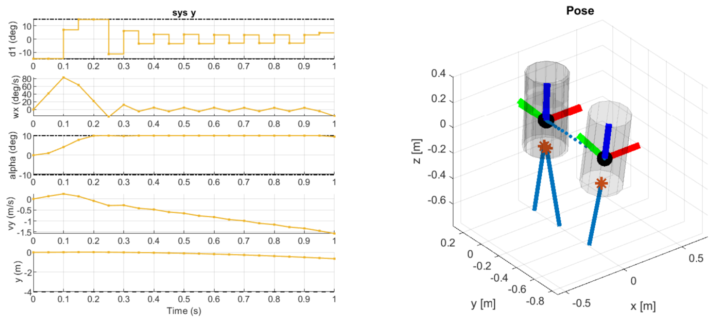

Figure 13: The open-loop tracking plots for y-subsystem.

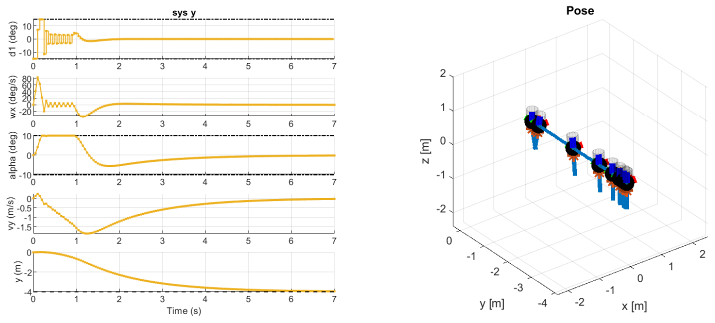

Figure 14: The closed-loop tracking plots for y-subsystem.

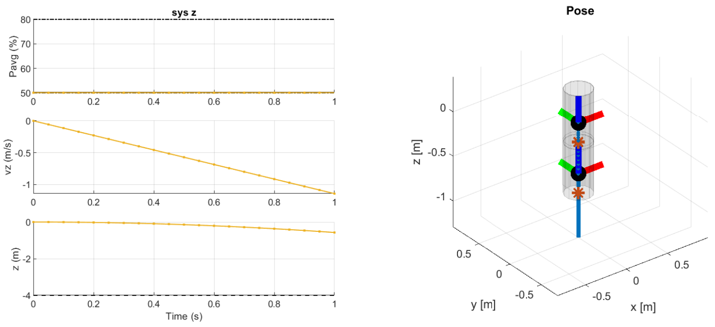

Figure 15: The open-loop tracking plots for z-subsystem.

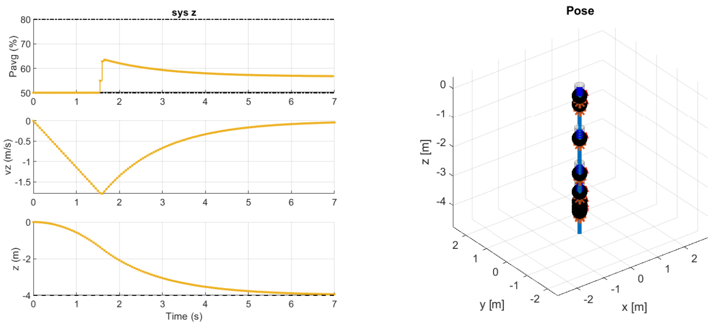

Figure 16: The closed-loop tracking plots for z-subsystem.

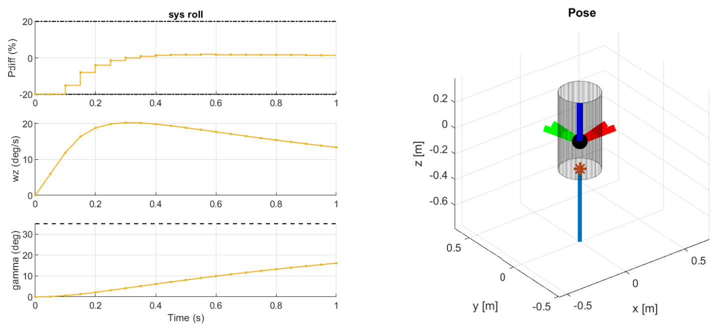

Figure 17: The open-loop tracking plots for roll-subsystem.

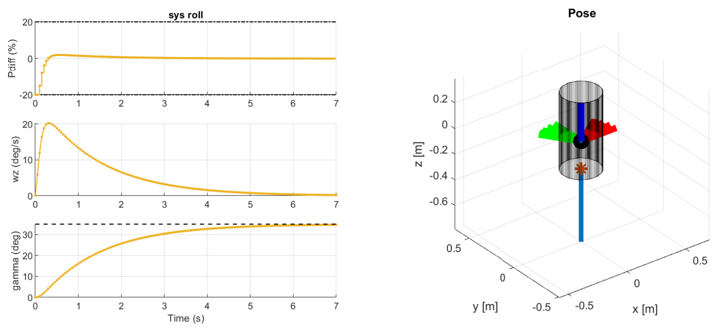

Figure 18: The closed-loop tracking plots for roll-subsystem.

5 Simulation with Nonlinear Rocket

5.1 Deliverable 4.1

We merge all the four linear controllers to obtain a full controller and apply it to the nonlinear dynamics. Fortunately, the first attempt is already successful: as there is no constraint violation and the controller decently tracks the complex trajectory.

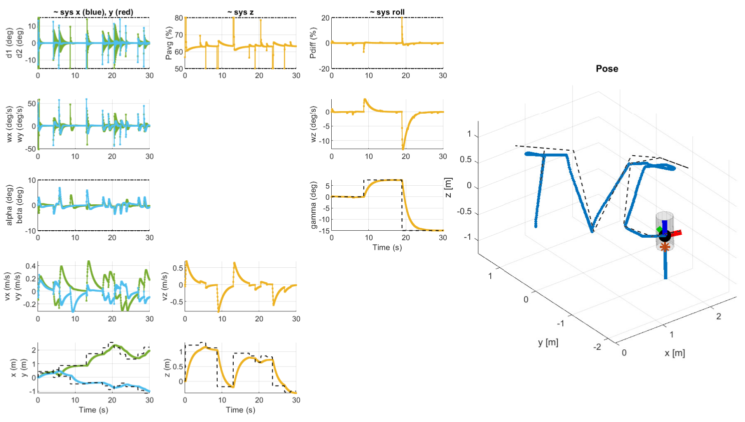

Figure 19: The closed-loop tracking plots for the nonlinear model.

Despite the success, we have observed that the tracker exhibits a slow response, especially in z, and the tracking trajectory is not perfect. To improve it , we have undertaken the following attempts and finally selected the last one:

1. **Decreasing the control penalty. (failed)** We suspect that the slow control is due to too high control penalty so we decrease all the control penalty to one tenth of the original values. However, it doesn't make any difference.
2. **Increasing prediction horizon. (failed)** Theoretically longer prediction horizon should have better overall performance, and 1s horizon may be too short for a 30s control problem. Thus we tried 2s and 4s but didn't see obvious improvement.
3. **Decreasing all the (angular) velocity penalty. (failed)** The slow response may come from too large penalty to the velocity. Thus we decrease all the velocity penalty to one tenth. Unfortunate this leads to infeasible problem.
4. **Decreasing the (angular) velocity penalty except for w_y and w_x. (successful)** The in-feasibility may come from the model mismatch, which is serious when α, β, $\delta_{1,2}$ are far from zero. Thus this time we remain the penalty weight for w_y and w_x. It finally achieve faster control response and preciser tracking performance, as presented in Fig. 20.

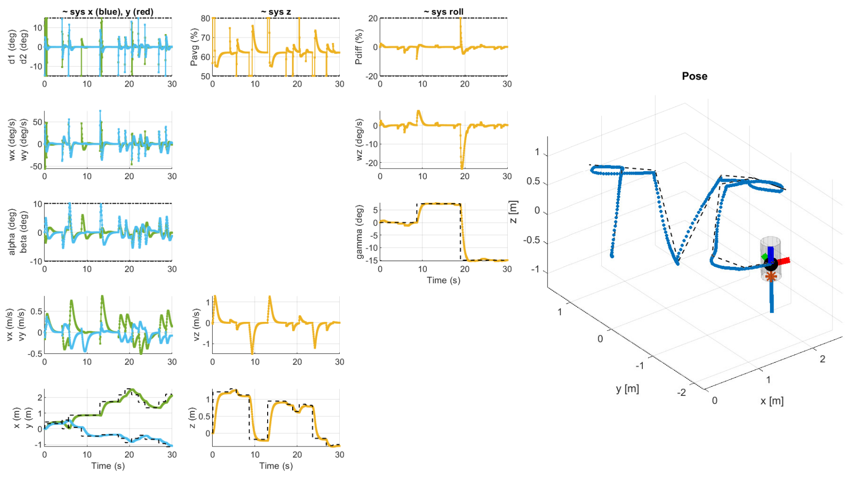

Figure 20: The improved closed-loop tracking plots for the nonlinear model.

Since the improved tracker already work very well , we didn't try the soft-penalty method in this task.

6 Offset-Free Tracking

In this section, we assume that the mass of the rocket is significantly different from what we have modeled, and we want to extend the z-controller to compensate. We assume the mass offset enters the dynamics of the system in the z-direction according to:

$$\mathbf{x}^{+} = \mathbf{Ax} + \mathbf{Bu} + \mathbf{Bd} \tag{7}$$

where d is a constant, unknown disturbance.

6.1 Deliverable 5.1 : Design an offset-free tracking controller for the z-controller.

To take this constant disturbance into account, we design a state and disturbance estimator based on the following augmented model:

$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{d}_{k+1} \end{bmatrix} = \bar{A} \begin{bmatrix} \hat{x}_k \\ \hat{d}_k \end{bmatrix} + \bar{B}u_k + \begin{bmatrix} L_x \\ L_d \end{bmatrix} \left(\bar{C} \begin{bmatrix} \hat{x}_k \\ \hat{d}_k \end{bmatrix} - y_k \right)$$

We therefore updated our controller (from the previous deliverable) to this augmented model to reject the disturbance and track the references with no offset. To do this, we chose the matrices A, B and C to be:

$$\bar{A} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix}; \bar{B} = \begin{bmatrix} B \\ 0 \end{bmatrix}; \bar{C} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix};$$

In order to have stable error dynamics that converge to zero, we chose L using the **place** Matlab function. We chose poles inferior to 1 (p = [0.6 0.7 0.8]) to ensure stability. The parameters of the place function are the matrices $\bar{A}'$, $\bar{C}'$ and the vector p containing the poles. To get the fastest and most accurate tracking, we also tuned the parameters Q and R of our controller. We finally chose them to be $Q = \begin{bmatrix} 40 & 0 \\ 0 & 300 \end{bmatrix}$ and R = 0.0001. The results of our simulation of the system with an augmented mass with and without off-set free tracking are shown by the following figures:

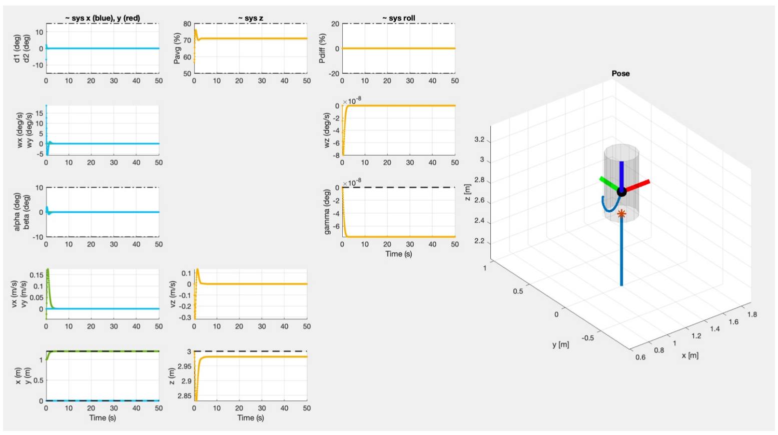

Figure 21: Simulation of the system with mass = 2.13 kg with the original controller (no offset-free tracking) during 50 seconds

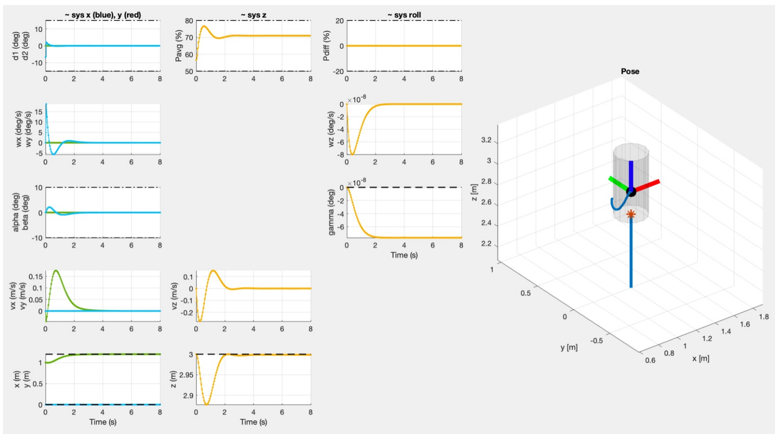

Figure 22: Simulation of the system with mass = 2.13 kg with the offset-free tracking controller during 8 seconds

In both figures, on the plot showing the position of the rocket over time along the z axis, we can notice the disturbance with a drop in the z-position curve in the first few seconds. We tried to get the shortest drop possible (in time) by tuning the parameters Q and R of our controller. The first figure (21) represents the results of the simulation with a mass of 2.13 kg but without the off-set free tracking controller. We can clearly see on the plot of the position along the z-axis that there is a constant offset between the reference and the position of the rocket (after the drop). The second figure (22) represents the results with the augmented mass but with the offset-free tracking controller. On the plot of the position of the rocket along the z axis, we can clearly see that after the drop corresponding to the disturbance, there is no offset anymore. This shows us that our offset-free tracking controller is successful.

6.2 Deliverable 5.2: Simulate the system with changing mass.

In this section, on top of our augmented mass, we now take into account that the mass of the rocket changes over time. Since the rocket burns fuel, its mass decreases over time during the flight. We take the controller of the previous section 5.1 and try to our off-set free tracking and add a mass change rate of -0.27. At first, we encountered a problem of in-feasibility in the steady-state target optimizer. This is due to the fact that the rocket becomes so light that the power constraint is violated, because a power lower than 50 % is required. To solve this problem, we dropped the constraint and tuned the matrix Q and set it to Q = $\begin{bmatrix} 50 & 0 \\ 0 & 1000 \end{bmatrix}$. The results of this simulation during 12 seconds and then during 20 seconds are the following:

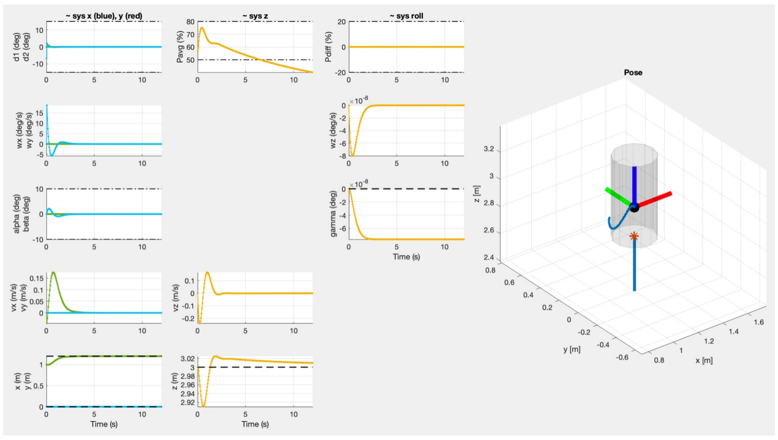

Figure 23: Simulation of the system with mass changing over time during 12 seconds

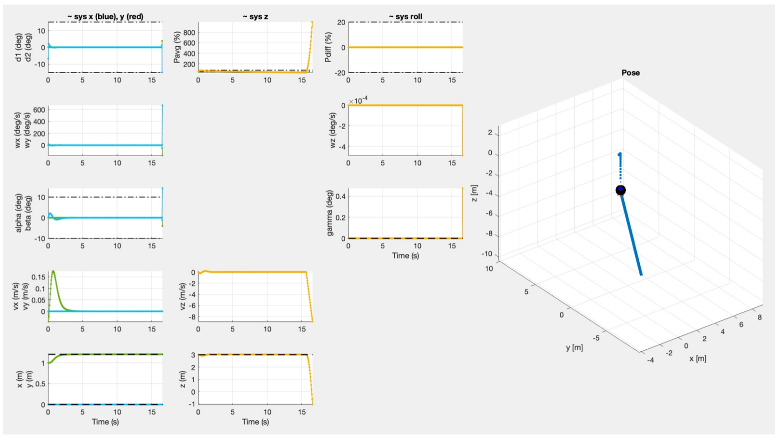

Figure 24: Simulation of the system with mass changing over time during 20 seconds

On the first figure, we can see that for the simulation of the system with changing mass during 12 seconds, the position in z still has an offset for a longer time than with a constant disturbance but it eventually approaches the goal position. We can explain it by the fact that the eigenvalues of $\bar{A} + \bar{L}C$ are too big so the estimator dynamics are too slow. Maybe if the time of update of estimator was reduced, this issue could be improved. How ever, we can see that for a longer simulation of 20 seconds (on the second figure), after 15 seconds, the rocket crashes. Indeed, at this point that the rocket can't provide any more thrust since it has used all its fuel.

7 Non linear MPC

For this section, we stop the decomposition of the rocket into 4 differents parts and focus on the whole model. We complete the code in thee NmpcControl.m file

7.1 Deliverable 6.1 : Develop a nonlinear MPC controller for the rocket using CASADI

Here we want to minimize a cost function over a finite prediction horizon, subject to constraints on states and inputs, to guide a rocket system along a desired trajectory.

To develop a nonlinear MPC controller we compte the Runge-Kutta methode to find the next step of the system. We have opted for the RK4 method using k1, k2, k3 and k4 parameters as seen in class. It is method is a numerical technique for solving ordinary differential equations.

We use four evaluations of the derivative function at different points within a time step to estimate the next value of the dependent variable. Applied to our system, this gives us:

Given a first-order ODE:

$$\frac{dx}{dt} = f(X, U)$$

where X is the dependent variable describing the system, U is the input of the model, and $f(X, U)$ is the derivative function, RK4 calculates the next value x_{n+1} at time X_{n+1} using the following:

$$\begin{aligned} k_1 &= f(X, U) \\ k_2 &= f(X + \frac{h}{2}, U + \frac{k_1}{2}) \\ k_3 &= f(X + \frac{h}{2}, U + \frac{k_2}{2}) \\ k_4 &= f(X + h, U + k_3) \end{aligned}$$

$$X_{next} = X + \frac{h}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4)$$

The goal is to find an optimal cost function for the rocket to follow the given path. The cost function depend on the next step of the model, as well on on different parameters such as position, velocity, orientation, angular velocity, thrust, mass, gravity, wind, drag, as well as target position, target velocity, and target orientation, knows as weights We then compute a LQR to calculate optimal state feedback gains.LQR helps minimize the cost function, providing optimal state feedback gains that guide the system toward a desired reference trajectory. These weights are part of our matric Q in the LQR method.

we set the R matrix to

$$R_{\text{terminal}} = \begin{bmatrix} 0.01 & 0.01 & 0.01 & 0.01 \end{bmatrix}$$

They are small values because

Our final Q_{terminal} matrix is the given one: (this was computed using trial and error until the rocket fitted the TVC shape) This matrix produce good results for the default maximum roll angle $|\gamma$ ref $|$ = 15deg.

$$Q_{\text{terminal}}^{\top} = \begin{bmatrix} 1 & 1 & 1 & 200 & 200 & 200 & 1 & 1 & 1 & 50 & 50 & 50 \end{bmatrix}$$

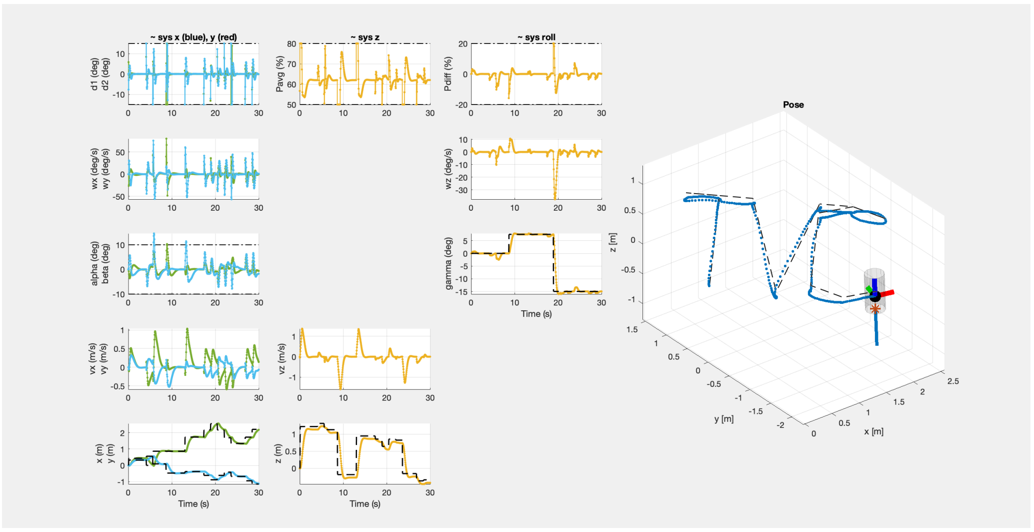

Figure 25: NMPC for the maximum roll angle at 15deg

We had to retune the paramaters for the default maximum roll angle $|\gamma$ ref $|$ = 50deg and we ended up with:

$$Q_{\text{terminal}}^{\top} = \begin{bmatrix} 1 & 1 & 200 & 360 & 200 & 200 & 1 & 1 & 1 & 50 & 50 & 50 \end{bmatrix}$$

After a bit of retuning we have this simulation:

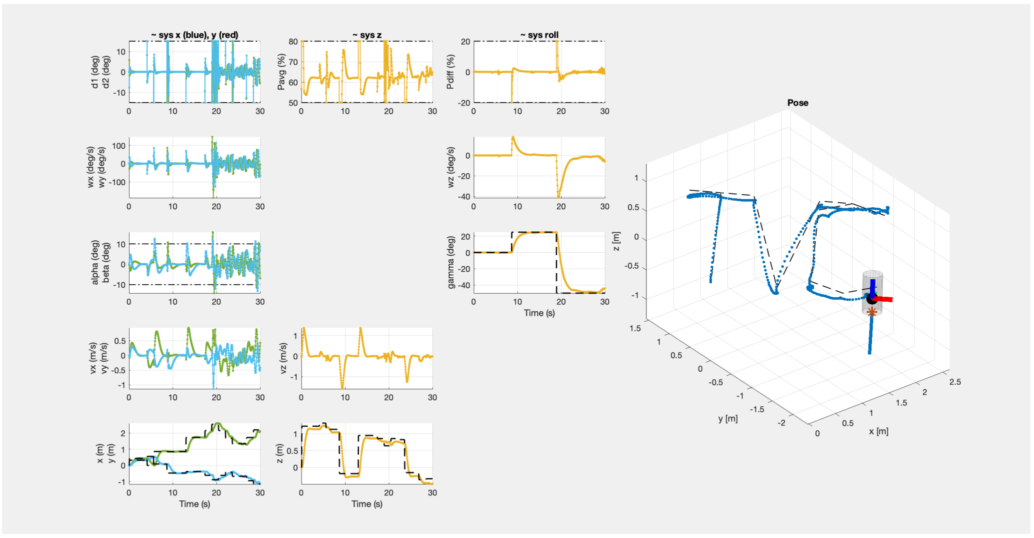

Figure 26: NMPC for the maximum roll angle at 50degrees

Its good to see that the parameters for the default maximum roll at 15deg made the rocket crashed at 50deg. That is because the control input limits might be insufficient to stabilize the rocket at a roll angle of 50 degrees. As seen:

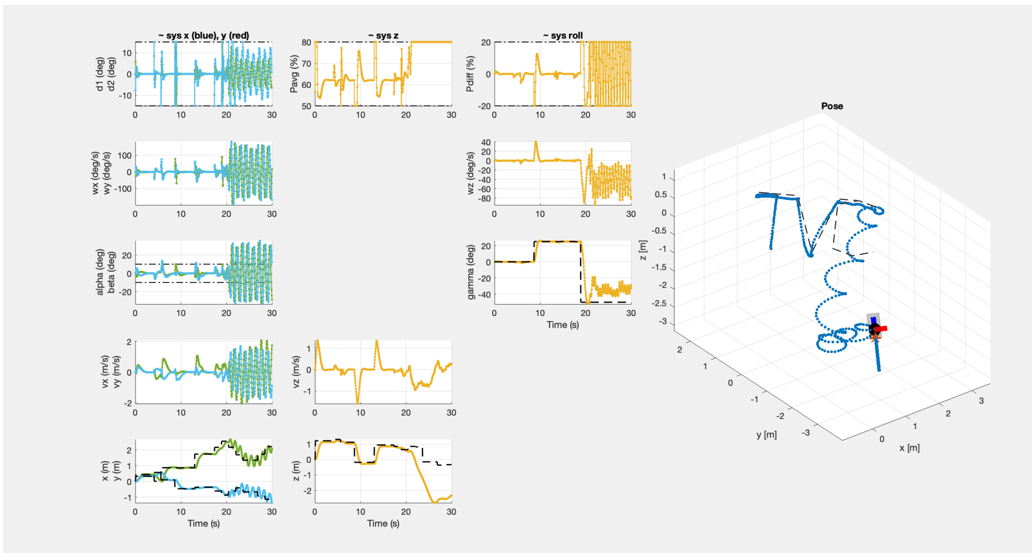

Figure 27: NMPC for the maximum roll angle at 50degrees with the weights that worked for 15deg

It seems like the Non linear MPC gives slightly better result for the the model, although the difference is not huge. Nonlinear controllers can adapt to changes better. They are more flexible and capable of handling variations. The weights are also easier to manipulate. We also did not have to do a subdivision of different problems here. The cost function was harder to compute also, and the tuning of the parameters was long as it depends on many differents parameters.

7.2 Deliverable 6.2 : NMPC with Delay / Delay Compensation

In here we implemented the get_u() fonction. We can simulate the system forward in time to account for the computational delay. the goal in this section is to account for the computational delay in the closed-loop system by predicting the state forward in time.

To do this, we computed the Euler integration to simulate the system dynamics. We used this formula based on the class for NMPC to complete the function get_u():

$x^+ = x + hf(x, u)$ where h is the sample period.

When running the simulation we find that the system is stable for a delay of 25 ms (1) but when we set it to 75 ms (3) the rocket crashes.

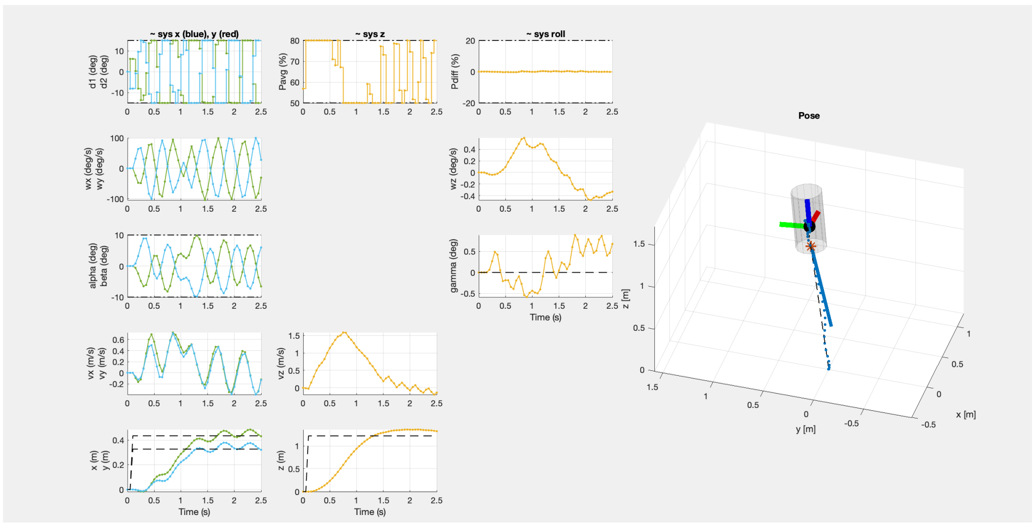

Figure 28: NMPC for a delay of 25ms

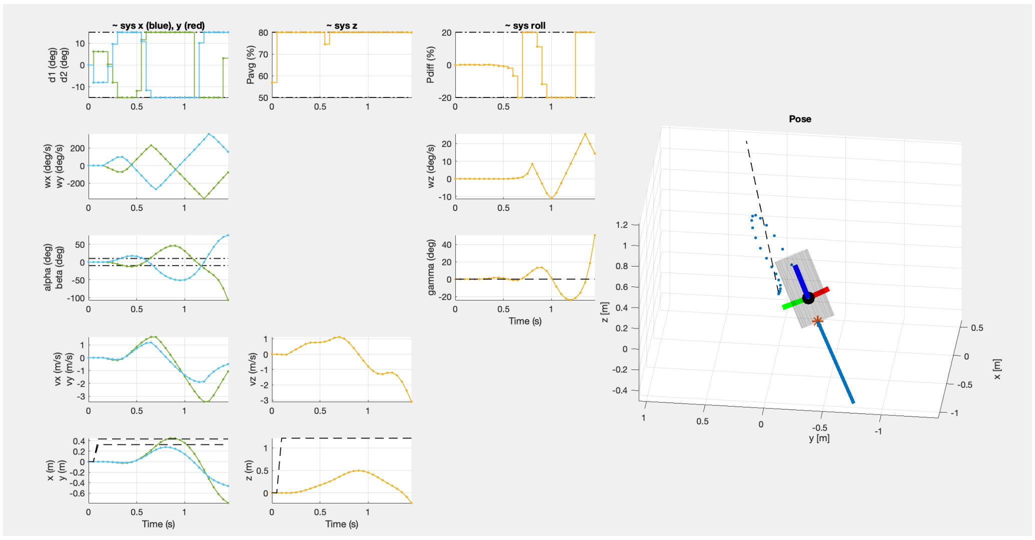

Figure 29: NMPC for a delay of 75 ms

We want to compensate this delay.

We realized now that with a partial delay of 175 ms for an actual delay of 200 ms, the system attains a steady state, although δ_1 and δ_2 have some oscillatory behavior.

If we have partial delay of 200 ms as well as an actual delay of 200 ms, the output is pretty good, and attains a steady state.

8 Conclusion

During this project, we designed a practical and robust Model Predictive Control to make a drone (recreating the dynamics of a rocket) fly and follow trajectories using a nonlinear model of the rocket. It was an interestinng project and a good way to apply the theory that we saw.