Model Predictive Control - Mini-Project Report

Group 19:

Nathan Kammoun

Adrien O'Hana

Elie Chelly

Date : March 18, 2022

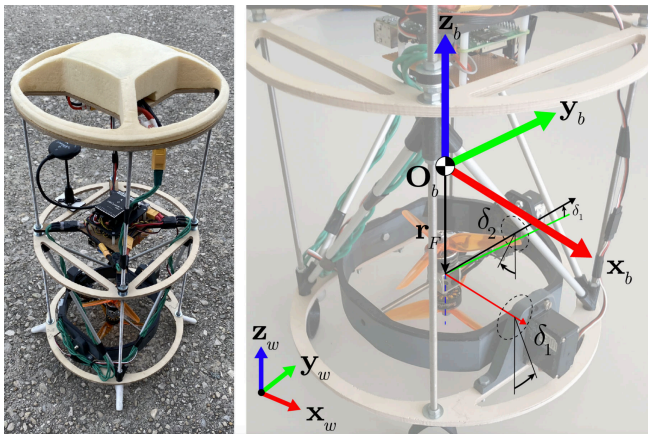


Figure 1: Dual propeller Rocket

System Definition

The goal of this project is to implement a controller for a given system. We briefly redefine this system below, which describes a small scale rocket prototype equipped with a propeller pair mounted on a gimbal. The propellers can be tilted by two servos.

State vector : $\mathbf{x} = [w_x\ w_y\ w_z\ \alpha\ \beta\ \gamma\ v_x\ v_y\ v_z\ x\ y\ z]^T$

- w_x, w_y, w_z are the angular velocities about the body axes.
- α, β, γ represent the attitude of the body frame in the world frame.
- v_x, v_y v_z and x, y and z represent the velocity and position in the world frame.

Input vector : $\mathbf{u} = [\delta_1\ \delta_2\ P_{avg}\ P_{diff}]^T$

- δ_1 δ_2 are the deflection angles of servos 1 and 2
- P_{avg} is the average of both throttles
- P_{diff} is the difference between the two throttles

Input constraints :
The following constraints on the inputs hold for the entire project, additional state constraints are introduced in the different sections.

$|\delta_1| \leq 15\,\text{deg}$

$|\delta_2| \leq 15\,\text{deg}$

$|P_{diff}| \leq 25\%$

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

Deliverable 2.1

This part of the project is dedicated to the control of a linearized version of the rocket's dynamics.

Explain from an intuitive physical / mechanical perspective, why this separation into independent subsystems occurs.

The system is first trimmed in order to find a steady state and input pair for which the next state is 0 (the rocket should be hovering). We then linearize the non-linear model around this steady state (x_s, us) to obtain a linear system. The resulting A, B, C, and D matrices can be studied to identify four independent linear subsystems which are intuitively explained below.

A small input deviation from the steady-state input for P_{avg} will result in a constant acceleration along z (Newton's law), therefore affecting only z and v_z. Around the steady-state all other state variables remain unaffected.

The roll is only affected by P_{diff} which creates an imbalance between the torques created by the propellers rotating in opposite directions. It therefore only affects the robot's angle around its z-axis. As long as P_{avg} remains constant the deviation from the steady-state input for P_{diff} should only affect γ.

If some input δ_1 is given (the angle of inclination of the thrust vector around the x-axis in the body frame), it will result in a translation along the x axis. This is because the attitude angles are small: the rocket will therefore not lose altitude (so we suppose no impact on z). The input affects affect v_x and x according to the formula $sin(\beta + \delta_1) \times g = \dot{v_x}$. Here since we are working with small angles the previous formula becomes $(\beta + \delta_1) \times g = \dot{v_x}$. The same goes for y with δ_2.

These independent systems' dynamics should be a valid approximation if the state remains in the neighborhood of x_s.

Deliverable 3.1

The goal is to design for each of the dimensions x, y, z and γ (roll), a recursively feasible, stabilizing MPC controller that can track step references. The MPC controllers need to have the following properties :

1. Recursive satisfaction of the input and angle constraints
2. Stabilization of the system to the origin
3. Settling time of around eight seconds when starting stationary at five meters from the origin (for x, y and z) or stationary at 45 for roll

Design Procedure

Explanation of design procedure that ensures recursive constraint satisfaction.

In this part, the system is linearized around a steady state and broken up into four independent subsystems. The goal is to design an MPC regulator for each sub-system. The regulator's goal is to bring a system from an initial state x_0 to the steady state x_s while respecting imposed constraints such as bounds for the state x, the input u and the linearized system's dynamics. For the four sub-systems, the initial states imposed are the following:

- $x_0 = \begin{bmatrix} 0 & 0 & 0 & 5 \end{bmatrix}$
- $y_0 = \begin{bmatrix} 0 & 0 & 0 & 5 \end{bmatrix}$
- $z_0 = \begin{bmatrix} 0 & 0 & 0 & 5 \end{bmatrix}$
- $roll_0 = \begin{bmatrix} 0 & 0 & 0 & 0.785 \end{bmatrix}$

At each call of the MPC regulator, a finite horizon of inputs is computed. For this, a cost function is defined that depends on the current state, the current input and a terminal cost that will be discussed later. This cost function is minimized subject to constraints on the state and the input defined by the following inequalities:

$$\begin{aligned} Fx &\leq f \\ Mu &\leq m \end{aligned} \tag{1}$$

Checking that those inequalities are verified at all times ensures recursive constraint satisfaction.

Tuning parameters

Q, R, H, terminal components The cost function is the following:

$$J(x,u) = \sum_{i=2}^{N-1} x_i^T Q x_i + u_i^T R u_i + x_N^T Q_f x_N \tag{2}$$

Where Q, R and Q_f are positive definite weight matrices. Q and R have to be tuned. For simplicity, Q and R are diagonal matrices. As we can see in equation 2, increasing Q means that to keep J(x,u) small, x has to be small. On the opposite side, increasing R results in u having to be small. For tuning, the importance is the ratio between the weights assigned in Q and in R. This means that with a large Q and a small R, priority will be given to reaching the target at all cost, even if it means very strong input. In the opposite situation, the controller will try to bring the state to the target with the minimal input.

The goal is to reach the best compromise between a smooth input and a rapid settling time. We want to avoid oscillation of the input, because the most important design choice is to assure stability of the controller.

H is the horizon of the controller. This parameter acts on the number of iterations of the system that will be computed in the controller. It defines "how far the controller is able to see in the future". A too low H might result in a myopic controller that may not be able adapt the input in time when for example following a trajectory with sharp turns. A too long horizon results in long computation. Here, a horizon of 8 seconds was chosen.

Q_f is the weight matrix of the terminal cost. It is computed using the 'dlqr' function of matlab. It compensates for the fact that in theory, MPC should have infinite horizon which is not the case practically. Q_f uses LQR infinite horizon theory to simulate an infinite horizon from the cost function's point of view.

Terminal Invariant Sets

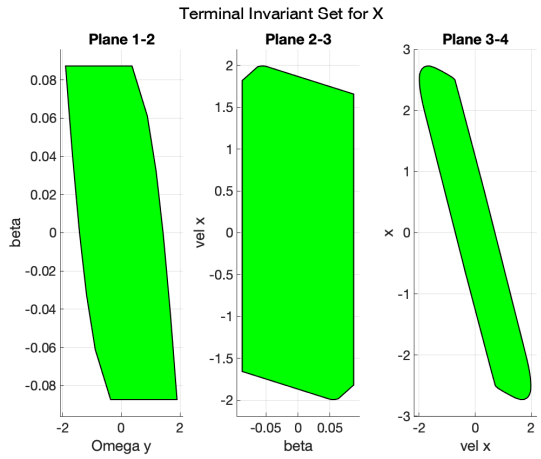
The terminal invariant sets in each case are computed using the algorithm seen in class: First, the LQR gain K is computed using matrices A, B, Q and R. A ploytope defined by the following set is created:

$$\Omega_0 = \{(x,u) \in R^{12} \times R^4 | \begin{bmatrix} FA & 0 \\ 0 & BK \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \leq \begin{bmatrix} f \\ m \end{bmatrix}\} \tag{3}$$

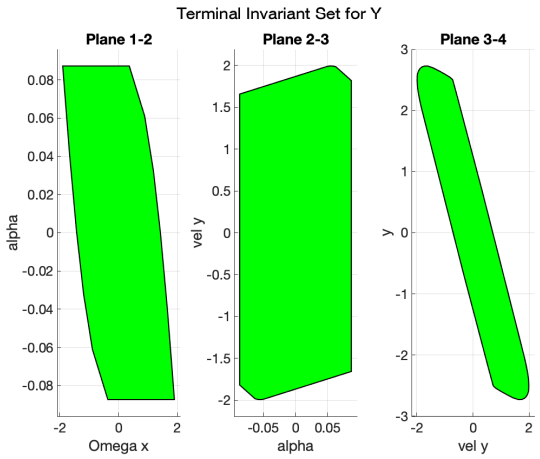
Then its preset is computed:

$$Pre(\Omega_0) = \{x | TA_{cl}x \leq t\} \tag{4}$$

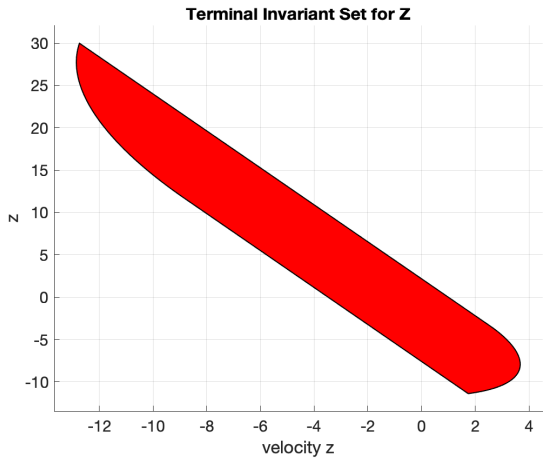
Where T and t are solution to the polytopic projection problem. Finally the intersection between Ω_0 and Pre(Ω_0) is computed and the returned set goes into $Omega_0$. The algorithm goes back to the step **compute Pre(Ω_0)** and this is looped until Ω_0 converges.



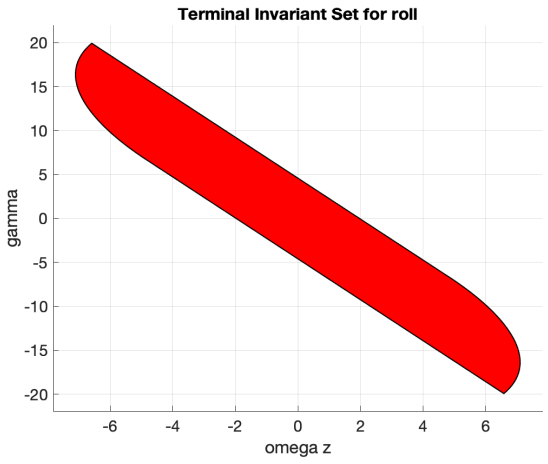
(a) Projected terminal set for X controller



(b) Projected terminal set for Y controller



(a) Terminal set for Z controller



(b) Terminal set for Roll controller

Plot for each dimension

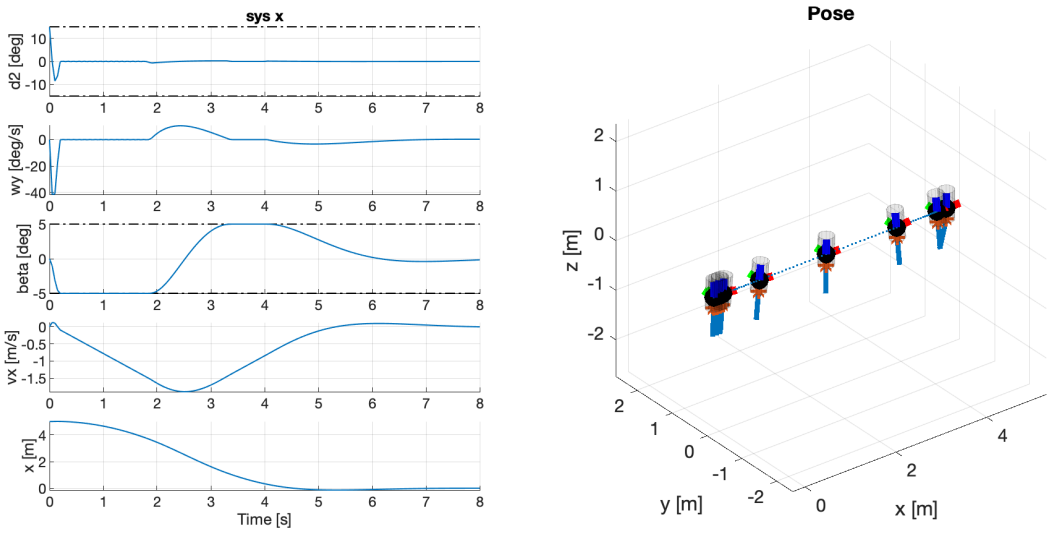


Figure 4: Evolution of states for X sub-system

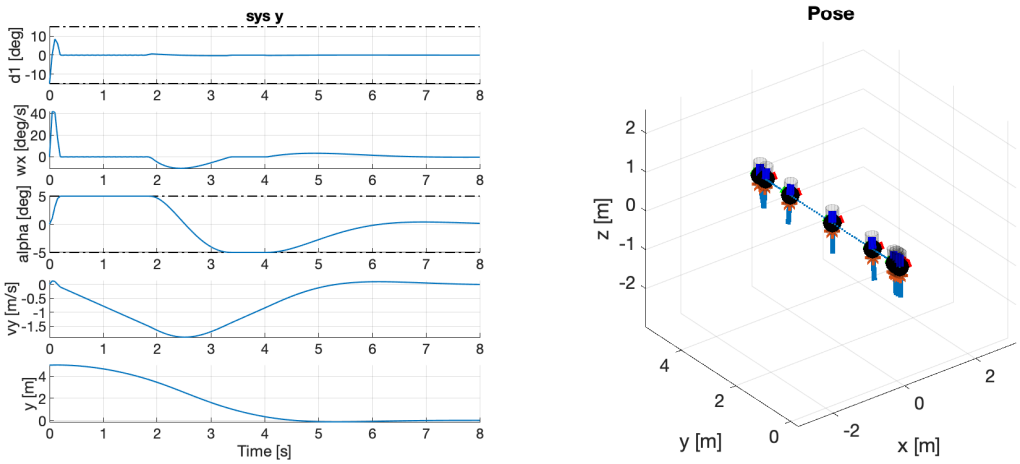


Figure 5: Evolution of states for Y sub-system

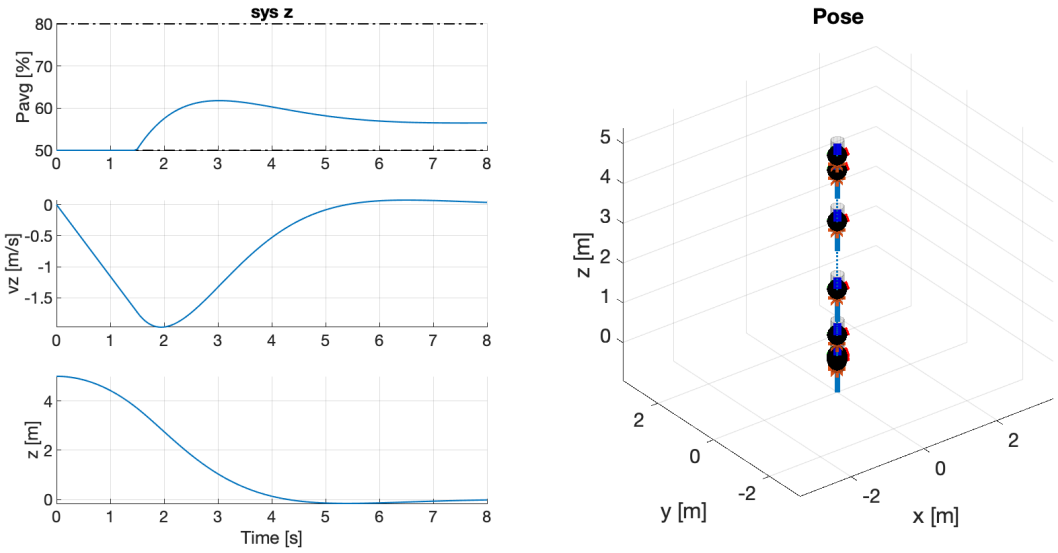


Figure 6: Evolution of states for Z sub-system

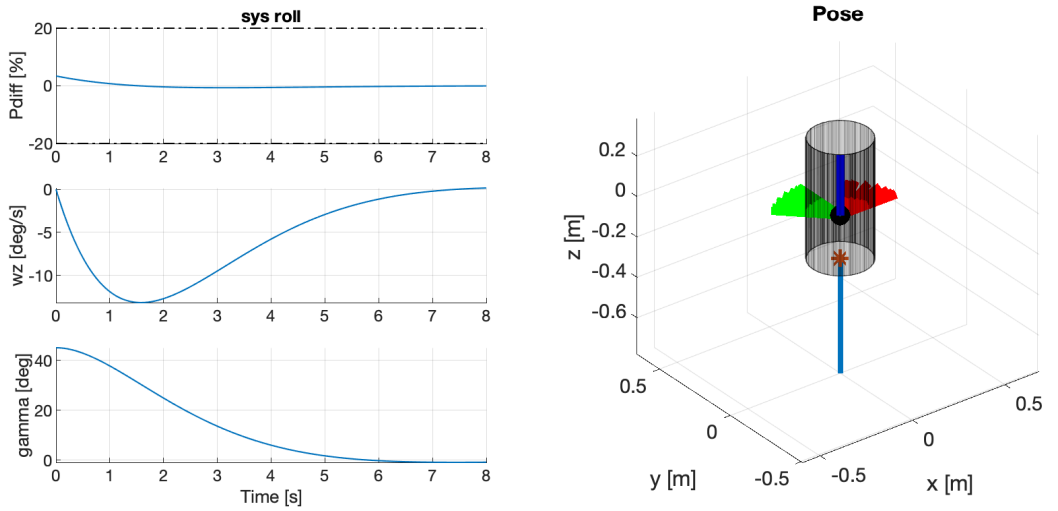


Figure 7: Evolution of states for Roll sub-system

Deliverable 3.2

The goal here is to extend the controllers so they can track constant references.

Design Procedure

Although the terminal set were no longer required, the terminal set constraint was kept in the controllers, it was possible to do so because a shifted version of the terminal invariant set is still invariant. To be able to track step reference, the delta formulation was used. The cost function and the constraints equations are updated.

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

These new deviation variables result in new constraints which we've implemented in a similar fashion than in the previous section. The procedure stays the same except for the delta formulation. The Horizon was kept at 5 seconds, using a longer horizon is not necessary as the step command is simple.

Tuning Parameters

The tuning procedure always started by choosing Q as the identity and R=1. The coefficients on each states constraints and input constraint were then tuned one by one to reach a satisfying trade off between feasible inputs and settling time.

The X and Y direction use the same control (symmetrically), the controllers are therefore similar. The weight on $Q_{4,4}$ was increased to put a bigger weight on command satisfaction. We noticed that the control resulted in high frequency oscillations for the input. This was addressed by greatly increasing the R coefficient penalizing the input sequence. It helped getting rid of the vibrations. Also, the constraints on the angular velocities $Q_{1,1}$ were increased as it helped smoothing the input curve. The control results against a step reference is shown on 8 for the X controller and 9 for the Y controller.

The tuning of the z and roll controllers was less challenging as the inputs P_{avg} and P_{dif} did not suffer from the same vibrations issues. It was therefore possible to increase $Q_{2,2}$ to make the controllers more aggressive without impacting feasibility and keeping reasonable inputs. The control results against a step reference is shown on 10 for the Z controller and 11 for the roll controller.

Results

The plots below show the response of the four individual sub-systems with tuned controller to the following references:

- sys x: $x_{ref} = -5$ [m]
- sys y: $y_{ref} = -5$ [m]
- sys z: $z_{ref} = -5$ [m]
- sys roll: $\gamma_{ref} = 0.785$ [rad]

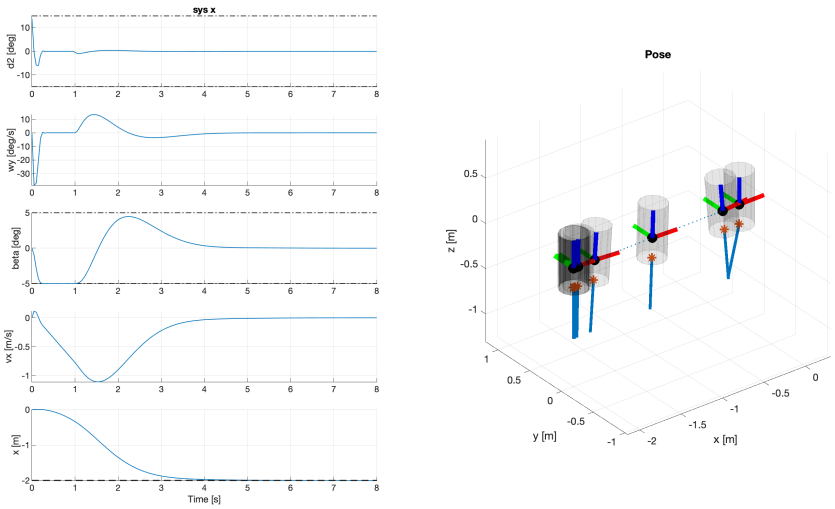


Figure 8: sys x response to step reference

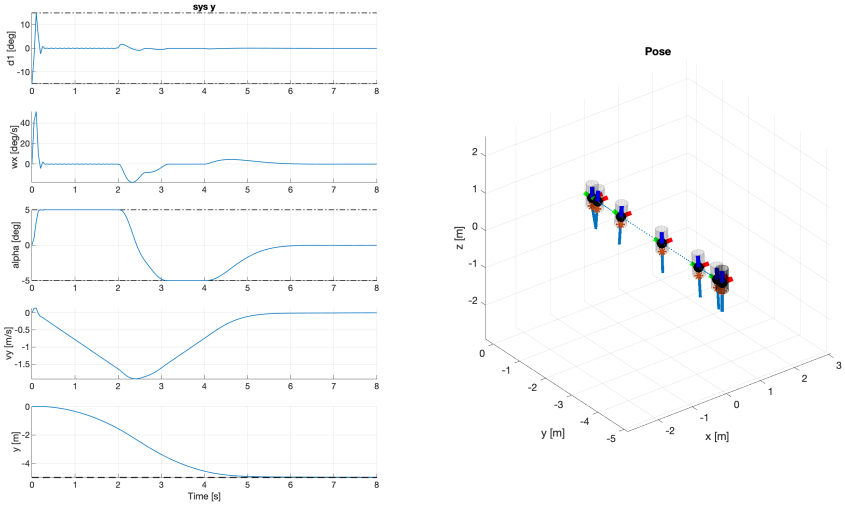


Figure 9: sys y response to step reference

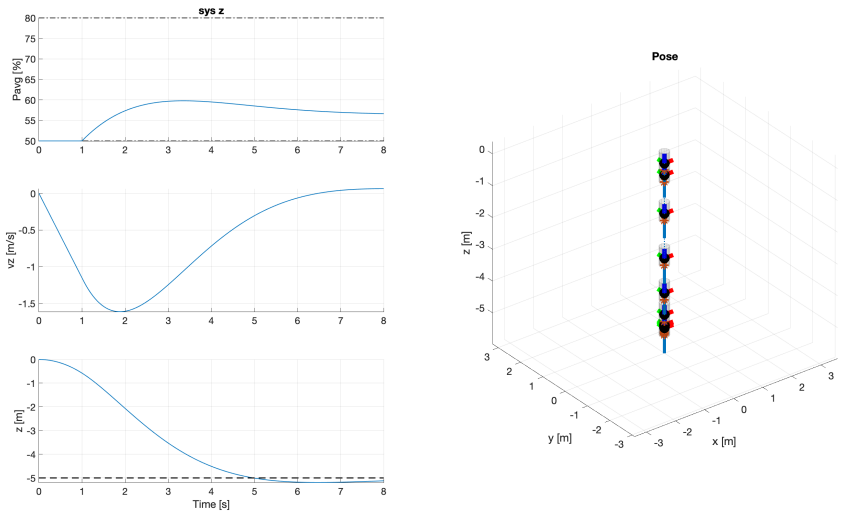


Figure 10: sys z response to step reference

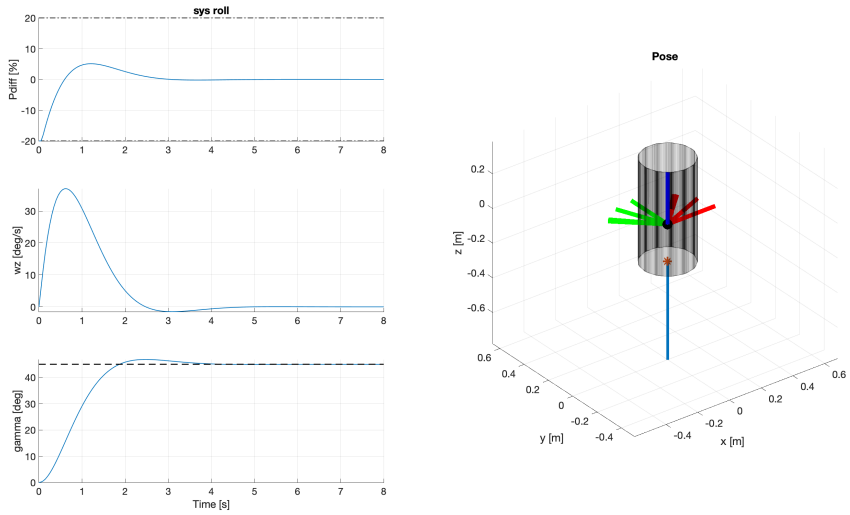


Figure 11: sys roll response to step reference

We can observe for the four sub-systems that the tuning is such that the settling time is faster than 8 [s], (between 3 and 5 seconds depending on the sub-system) without having brutal inputs nor oscillations. We want to avoid oscillations of the input to ensure the closed loop system stability.

Deliverable 4.1

In this section, we will use our controllers to have the nonlinear version of the rocket track a given path.

Design Procedure and Tuning Adjustments

In the previous part, we were using the linear model of the rocket, we are now testing the results of the controller on real dynamics. The full nonlinear system is simulated with a merge of the four controllers.

Running the linear controllers in the nonlinear simulation results in a model mismatch for the predictive controllers. As a result, the states may now violate the constraints, the controller needs to be tuned with extra care to account for infeasibility.

Compared to the previous part, the reference now change every seconds with variable step magnitude. To be able to correctly follow the path, we tuned the controllers to be more aggressive in order to reach the command as quickly as possible. This was done by increasing the states in the Q matrix. However, pushing the weights too much led to infeasible states, the tuning was done by optimizing the trade-off between feasibility and performance.

The x and y controllers tuning procedures were similar, the maximum difference between two consecutive references was not very high, so it was not necessary to increase the state constraints a lot. In the contrary, tuning the z controller was more challenging as the z reference showed some aggressive steps. The roll controller was a little bit different. The constraints were less easily violated which both allowed pushing the weights on the state matrix and decreasing R.

Results and Plots

Below are the inputs for trajectory following for respectively δ_1 and δ_2, P_{avg} and P_{diff}

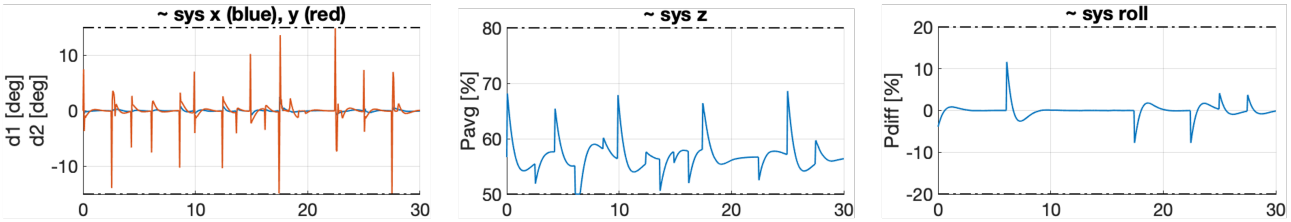


Figure 12: Inputs for roll

The plots below show the states evolution for subsystems x and y (ω_x, α, v_x and x are in blue)

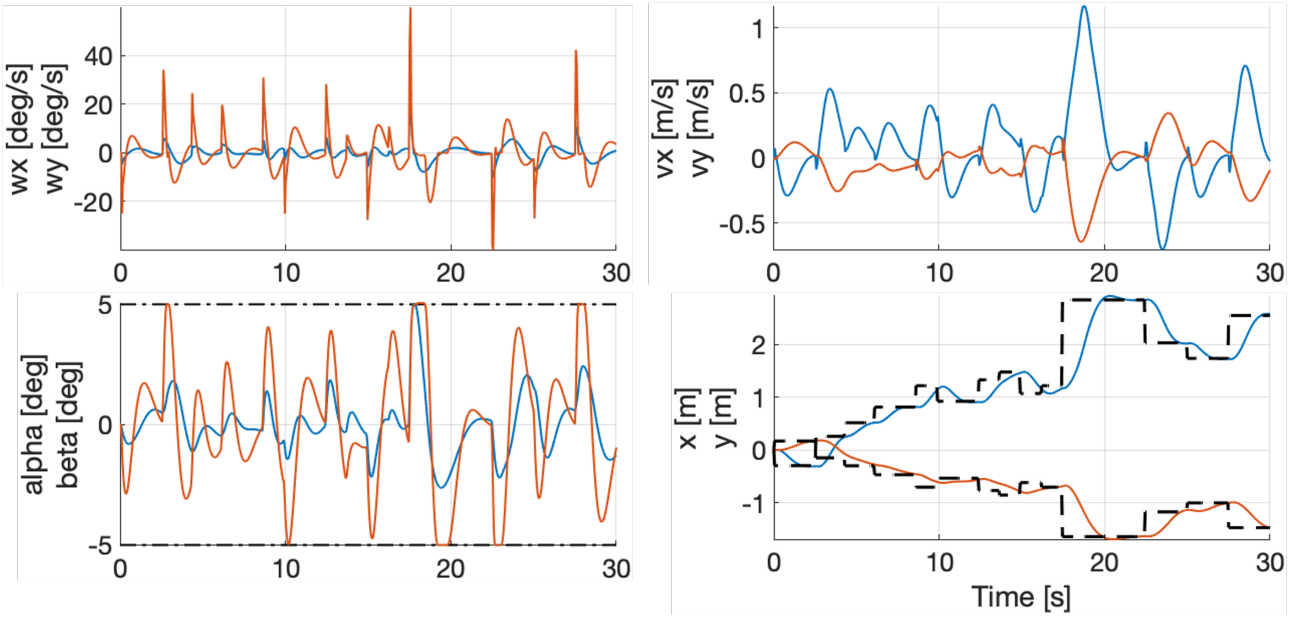


Figure 13: States and tracking for x and y

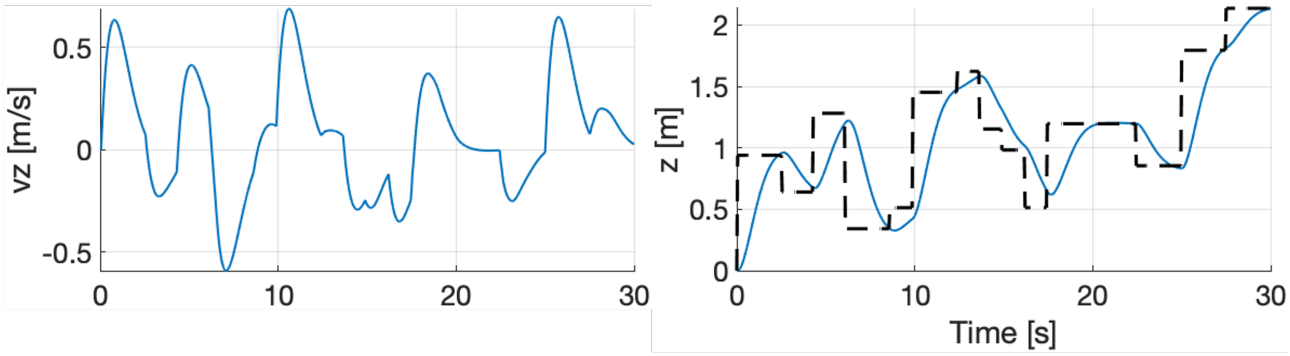


Figure 14: States and tracking for z

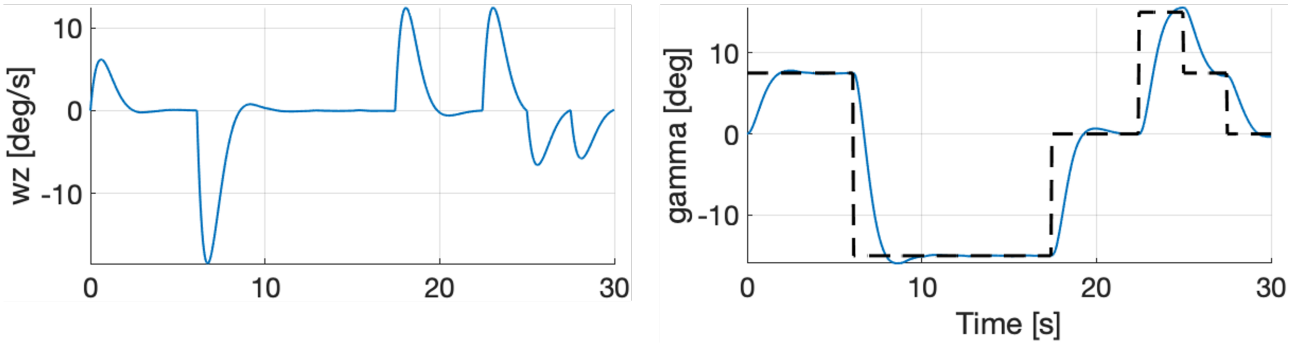


Figure 15: States and tracking for γ (roll)

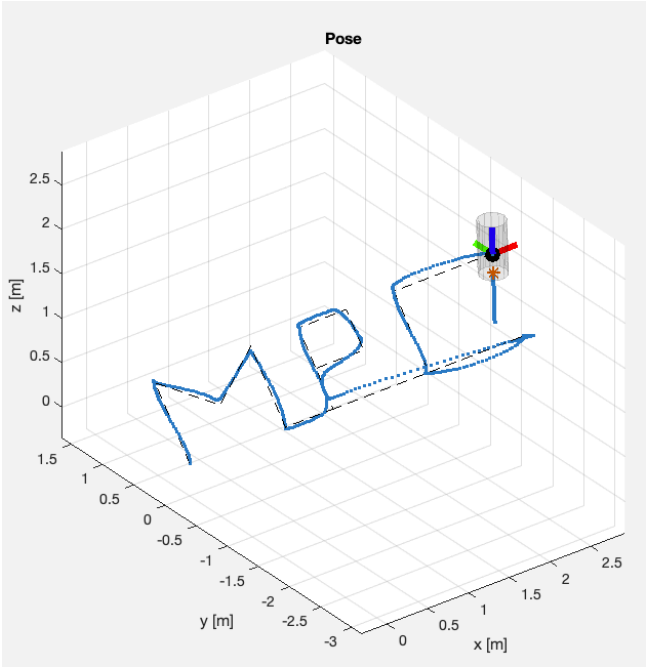


Figure 16: Tracking

As we can see in the results above, this tuning allows good tracking while avoiding brutal oscillations on δ_1 and δ_2.

Deliverable 5.1

This section aims at extending the z-controller to compensate for a change of the rocket's mass.

Design procedure

The model was updated to take an unknown constant disturbance into account in order to provide the controller with the ability to reject that disturbance. An observer was designed to estimate the offset and the state of the system. The same design procedure as in the previous part was used, except that the cost function was updated to take the estimator into account. The estimator was designed so that the state estimate converges towards the correct state and constant input disturbances. The following equation describes the state estimate update :

$$\overline{x_+} = \overline{A} * \overline{x} + \overline{B} * u + L * (\overline{C} * \overline{x} - y)$$

With $\overline{A}$, $\overline{B}$, $\overline{C}$ defined from A, B, C and L is a state feedback matrix computed using pole placement.

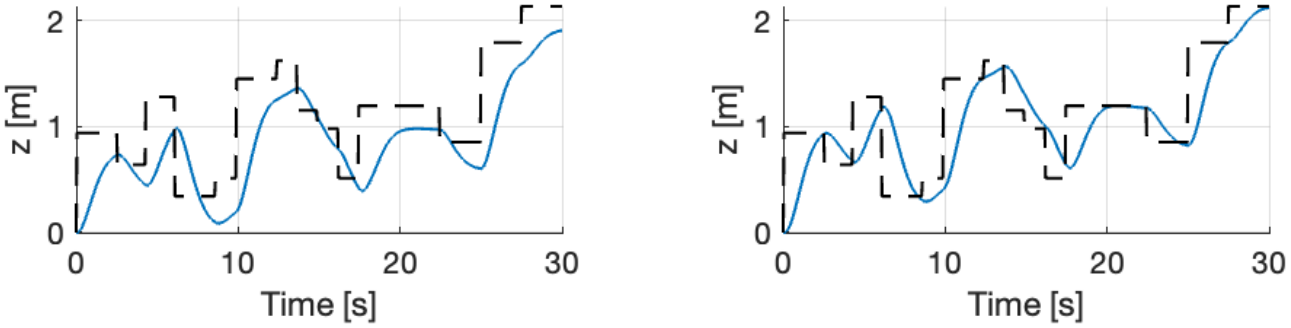
Tuning Parameters

The new controller was tested using the same Q and R matrices as in the deliverable 4.1. and achieved similar results. The weights were not updated as we considered that achieving similar results than the test on the disturbance-free model was enough. The controller is now able to track references with no offset by effectively rejecting the disturbance coming from the mass change. Fig 17b shows that the controller is able to track the reference without offset. The offset-free tracking is shown on Fig 19.

Impact of changing the mass

The behavior of the rocket was observed, keeping the controller used on the disturbance free model to observe how does the mass mismatch affects the system. The system was simulated with the rocket.mass property set to 1.783.

The resulting behavior shows an offset in the z reference tracking. This can be explained by the fact that the rocket relies on a model that is no longer accurate, the P_{avg} command does not increase according to the change of mass. Figure 17a shows the offset in the z direction, the tracking of the x and y commands are not affected. Figure 18 illustrates the loss of performance when changing the mass. When following the set-points references, the overall shape of the letters is roughly the same as before as the controllers in the x and y direction don't suffer from the change of mass. However, the path is shifted downwards as the P_{avg} command depends on a model that is no longer an accurate representation of reality.



(a) Without estimator (b) With estimator

Figure 17: Z command tracking

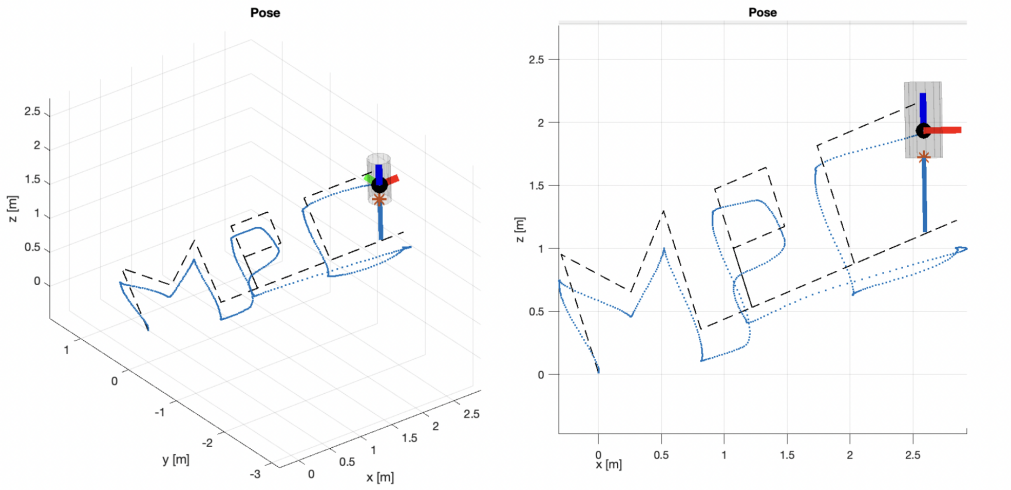


Figure 18: Impact of changing the mass : setpoints tracking without estimator

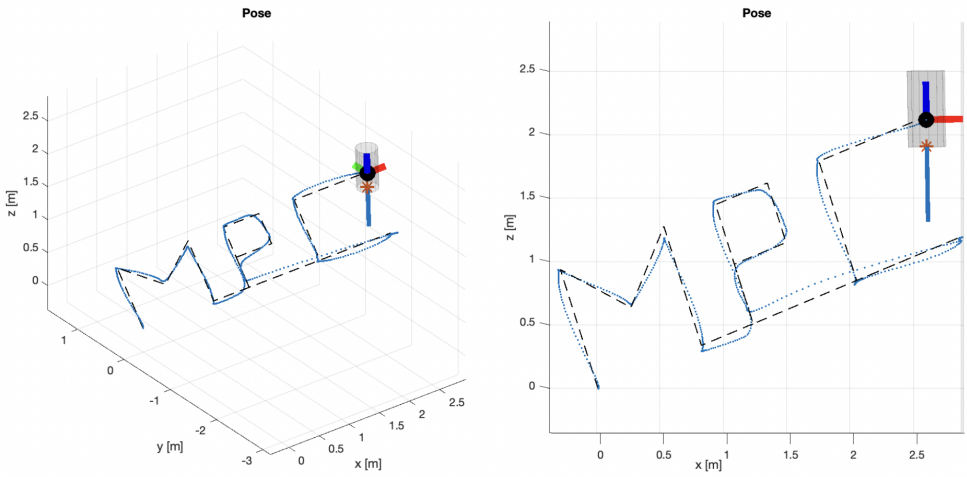


Figure 19: Setpoints tracking with estimator

Deliverable 6.1

Design procedure & Choice of tuning parameters

Finally, we can consider the nonlinear model for the controller design. We must now design a controller taking the full state vector as input $[w_x\ w_y\ w_z\ \alpha\ \beta\ \gamma\ v_x\ v_y\ v_z\ x\ y\ z]$, and outputting the necessary control inputs $[\delta_1\ \delta_2\ P_{avg}\ P_{diff}]$ to track a given reference $[x\ y\ z\ \gamma]$.

Since it is difficult to find closed-form solutions for non linear dynamics we must use a numerical integration technique. We know that the Runge-Kutta 4 method can give an accurate approximation of the non-linear differential equation which describes the rocket's dynamics and this is why we use this algorithm for our controller. The chosen integration time step is the same as the rocket's sample period.

Using CASADI which is meant to behave similarly to YALMIP, we also compute the terminal cost Q_f of an LQR controller and add it to our objective function. The constraints and objective are the same as for the linear controllers except for the angular constraints that which due to linearization and which we can now drop. An additional constraint on is added for numerical safety as instructed in the project description.

Pros & Cons of Nonlinear Controller

The linear controller is easier to implement and requires less computational effort therefore it runs faster. It is suited for a trajectory that doesn't involve extreme attitude angles for the linear approximation to remain valid. Another drawback of the linear controller is the high influence of tuning on the feasibility of the problem: since there is a model miss-match (linear controller with non linear dynamics), the controller can push the system close to state constraints which can result in violations.
For the nonlinear controller, performances are better and feasibility issues are less frequent. When tuned properly the results in path-following are very good but the computation is intensive which makes the controller tedious to tune via trial and error and limits the frequency of the controller as well as the length of the horizon.

Performance of the nonlinear controller with default $\gamma_{ref} = 15°$

After setting a reasonable horizon that limits computation time but insures that the rocket does not become myopic and after tuning the Q and R matrices with a similar approach as for the linear controller, we can achieve very good results on the given path using a non linear controller.

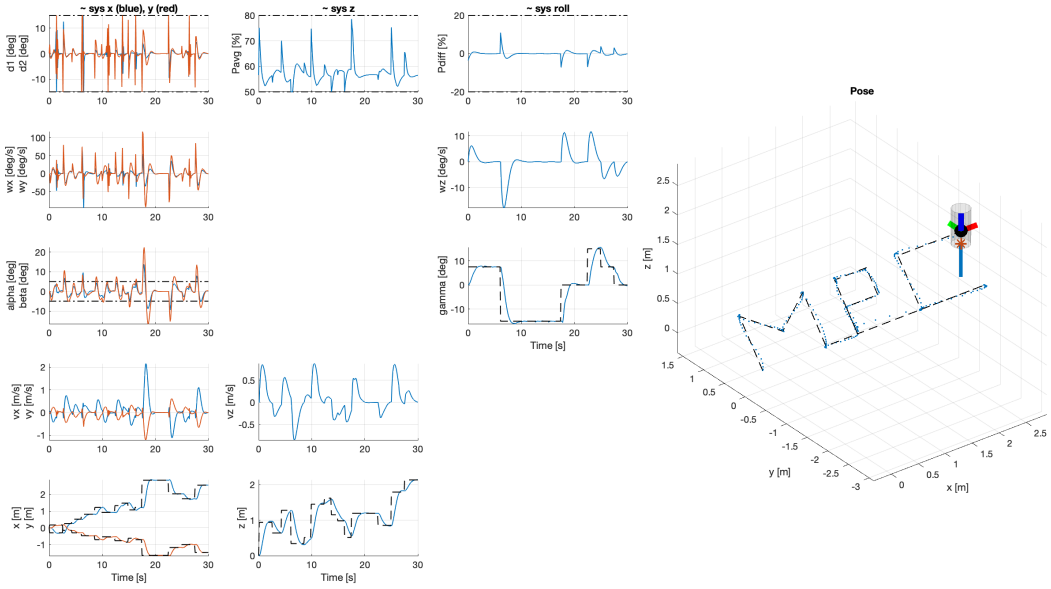


Figure 20: Performance of the nonlinear controller with $\gamma_{ref} = 15°$

Performance of the nonlinear controller with $\gamma_{ref} = 50°$

With a larger new maximum roll reference, the previously tuned controller performs reasonably well and by increasing the weights for w_x and w_y in the Q matrix we have improved it slightly. Nonetheless, the overall tracking is worse than with the previous maximum roll reference but this result is expected as the path is now more difficult to follow.

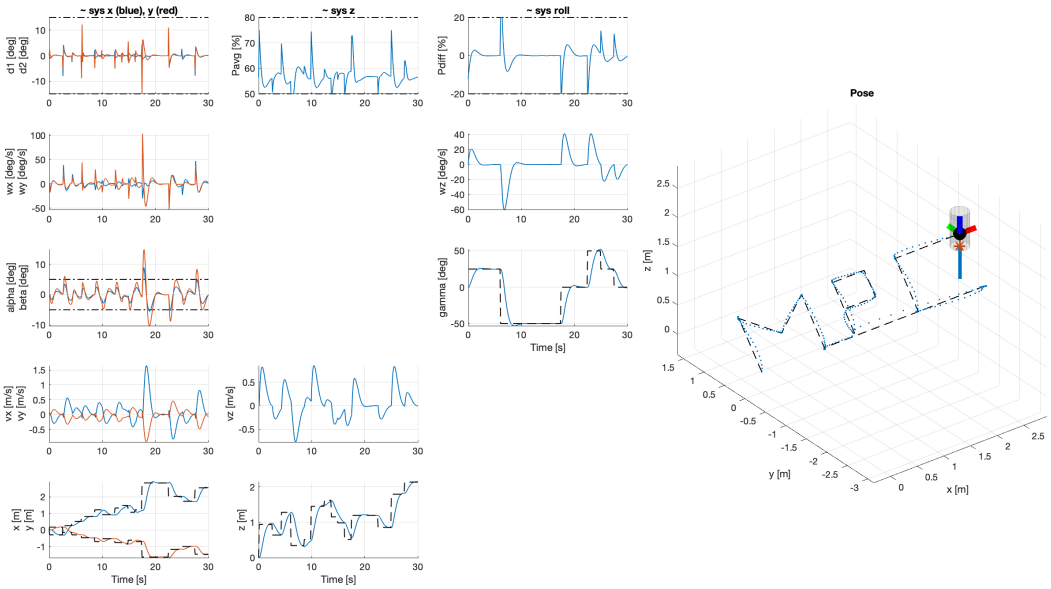


Figure 21: Performance of the nonlinear controller with $\gamma_{ref} = 50°$

Performance of the linear controller with $\gamma_{ref} = 50°$

Finally, we would like to compare the performance of the nonlinear and linear controllers. The first run was done with the tuning of part 4.1 for the linear controller.

Increasing γ_{ref} to 50° induces high angular speeds for ω_x and ω_y , especially when the roll reference reaches 50°. This results in violations of the α and β constraints, preventing the simulation to complete properly. The controller was re-tuned in order for the simulation to be completed. This was done by adding high weights on angular speeds and weights inferior to 1 on inputs. This tuning leads to an inaccurate path following but more tuning often results in infeasibility.

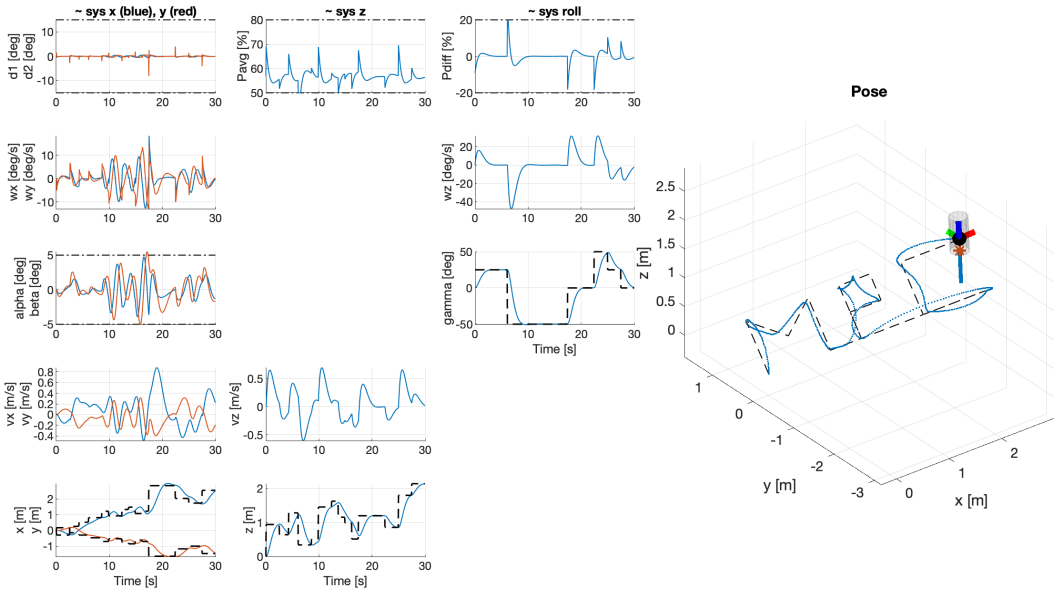


Figure 22: Performance of the linear controller with $\gamma_{ref} = 50°$

Conclusion

On this particular problem, the superiority of the nonlinear controller is demonstrated: some more tuning was required but the controller finally managed good path following. The conclusion would be that both controllers can be useful. It is certain that in terms of tracking performance the nonlinear controller is a better choice but at the cost of high computational efforts.