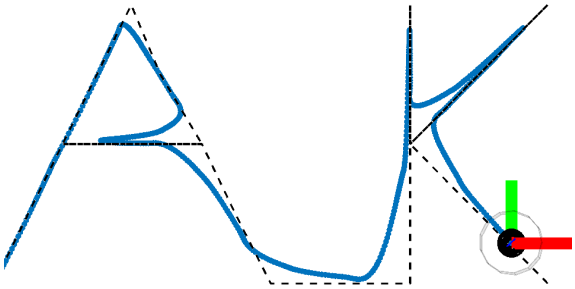
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

GROUP



MODEL PREDICTIVE CONTROL - MINI-PROJECT REPORT

Gilles Regamey, Mathieu Schertenleib, Owen Simon
296642, 313318, 330807

12th January 2024

Chapter 2

Linearization

2.1 Separation into independent subsystems from physical / mechanical perspective.

It is possible to have a mechanical intuition for the behaviour of the drone when we think about the system around a steady state linearization point u_s. Our controller will be giving an input $u = u_{\text{MPC}} + u_s \Rightarrow u_{\text{MPC}} = u - u_s$. The most important part in understanding how to divide the problem into multiple parts is to see that each input acts mainly onto one subsystem around the linearization point

- **Roll**
  We can control roll with only the difference in acceleration and speed of the propellers and using their inertia and air friction to produce a variable torque around Z_b. This difference does not interfere with the thrust produced or the position around x_s and u_s. sys_roll is easily separable into its own subsystem, as it only depends on the input of P_{diff}. Indeed, we see that the biggest factor which influences γ is P_{diff} which outweighs all other factors. This enables us to establish a chain of P_{diff} influencing ω_z, in turn influencing γ creating its own independent system. Note that when the rocket roll is bigger than around 15°, the linear model begins to fall apart as variations in d_1 and d_2 both act on the horizontal speed and position.

- **Altitude control**
  Similarly, we know that changes in thrust will only affect the speed in the Z_w / Z_b direction if all other inputs are at u_s and the rocket is not tilted (around x_s). So $P_a vg$ has the biggest effect on the lift as long as the assumption that our rocket remains upright holds. Hence, we can create a chain of influence, P_{avg} acts on v_z, changing z.

- **sys_x**
  The separation is possible if we assume that our system works around an equilibrium point and that the P_{avg} applied to the system does not diverge too much (i.e. the amplitude of the thrust has little effect on the subsystem). If we pivot δ_2, the thrust will apply a torque on the rocket, influencing ω_y, β, v_x and finally x. Furthermore, it is important to note that the linearization is only possible for very small angles close to the equilibrium pose.

- **sys_y**
  The same reasoning can be applied to sys_y, with δ_1 only influencing ω_x, α, v_y and y.

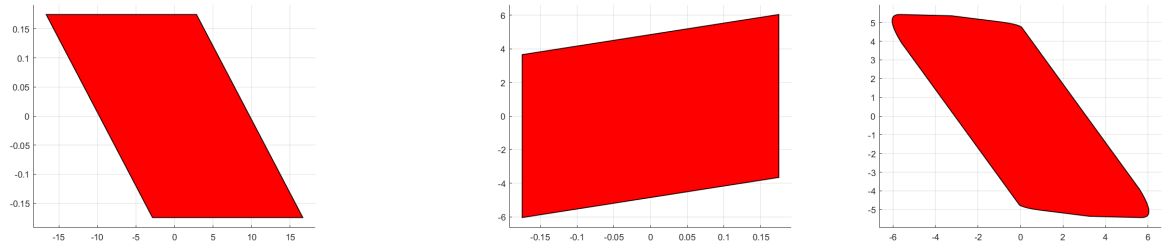
Chapter 3

MPC Controllers for Each Sub-System

3.1 Design of linear MPC

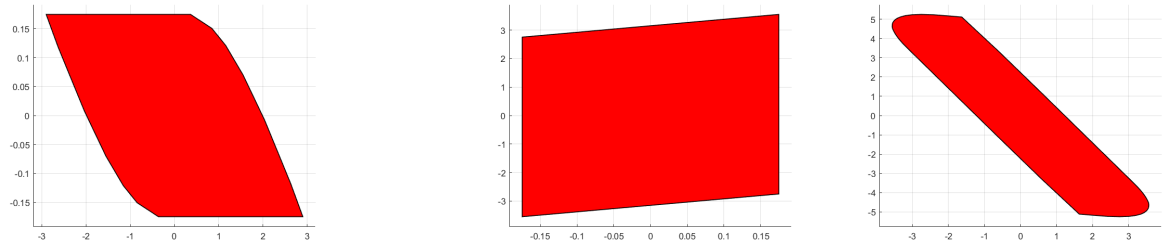
We imposed the constraints on the state and control variables at every time step. We also used these constraints to build the final set whilst adding an extra constraint that the last step should be in the final set. This hence ensures that our constraints are always satisfied.

In order to tune the parameters we chose to start from the parameters given in different exercise sets. We then proceeded to adjust $\frac{\bar{R}}{Q}$, increasing the ratio when we want to get to the right position with more intensity and decrease it when we want to go there more smoothly.



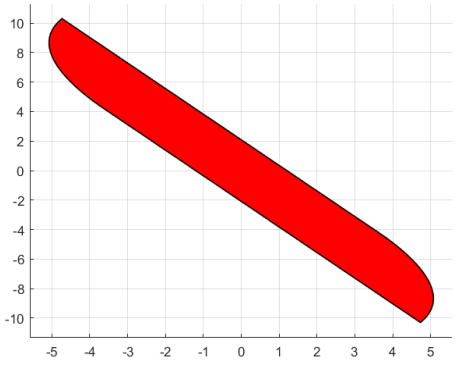
(a) Over axes ω_y and β (b) Over axes β and v_x (c) Over axes v_x and x

Figure 3.1: Presentation of the X_f for the MPC controlling x

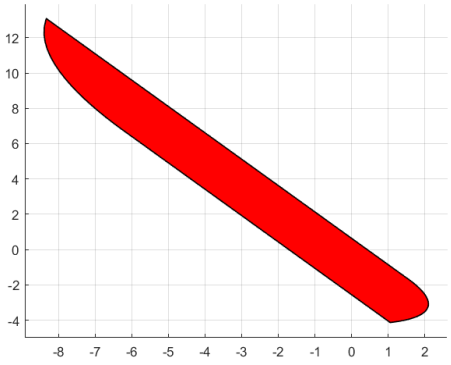


(a) Over axes ω_x and α (b) Over axes α and v_y (c) Over axes v_y and y

Figure 3.2: Presentation of the X_f for the MPC controlling y

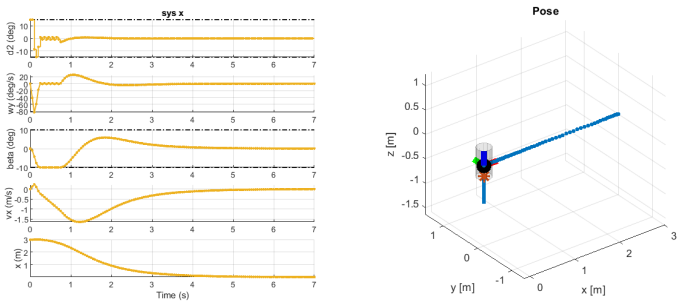


(a) Presentation of the X_f for the MPC controlling rolling as a function of v_z and z

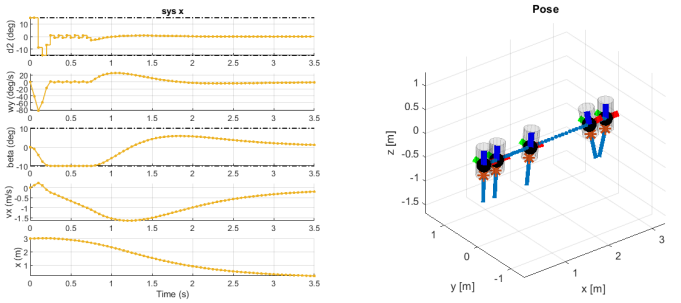


(b) Presentation of the X_f for the MPC controlling z with regards to v_z and z

Figure 3.3: Presentation of the X_f for the MPC controlling z and γ

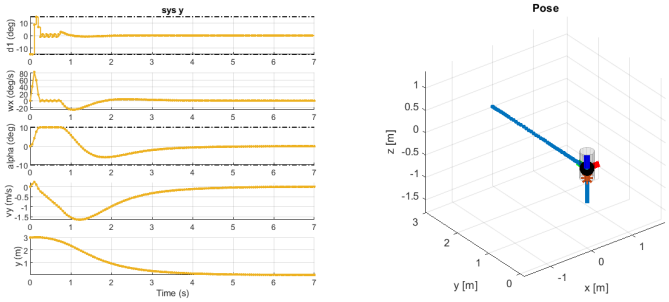


(a) Closed loop

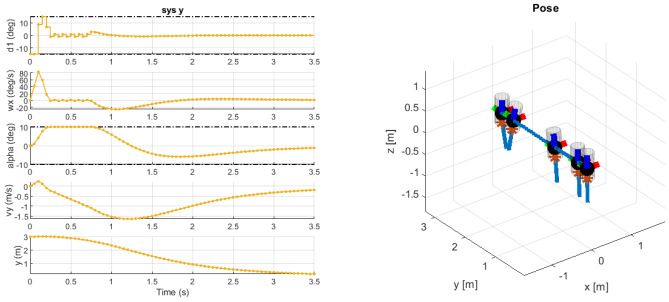


(b) Open loop

Figure 3.4: Linear open and closed loop for x

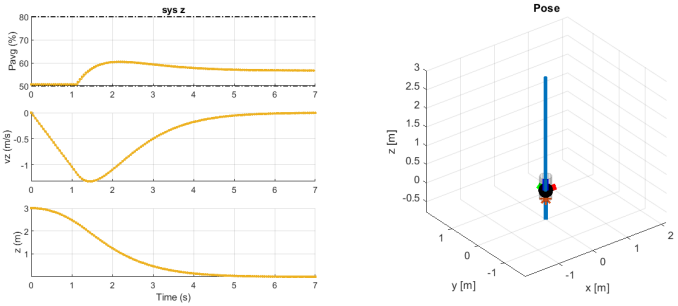


(a) Closed loop

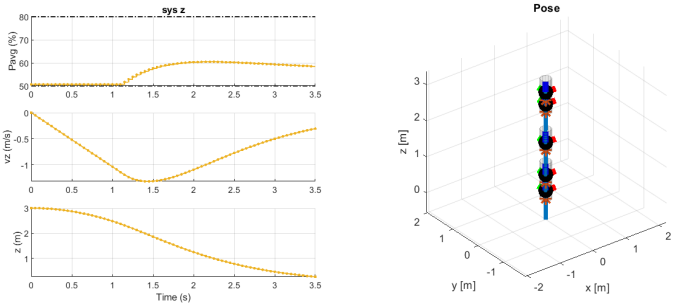


(b) Open loop

Figure 3.5: Linear open and closed loop for y

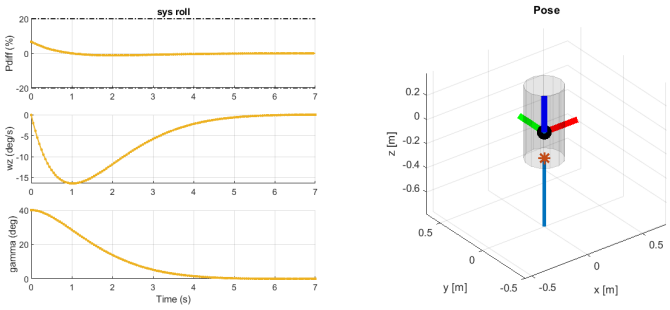


(a) Closed loop

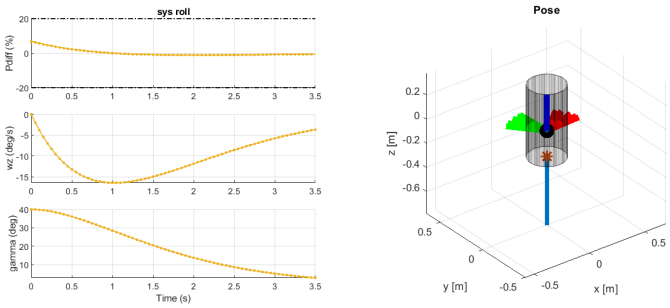


(b) Open loop

Figure 3.6: Linear open and closed loop for z



(a) Closed loop

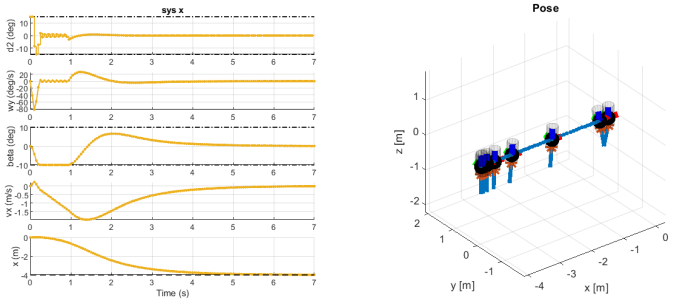


(b) Open loop

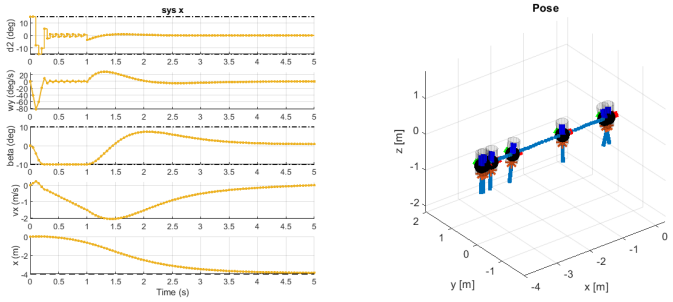
Figure 3.7: Linear open and closed loop for γ

3.2 Tracking a reference point

In order to get our system to converge to a given tracking point we shifted our problem by the xs and us necessary to track our reference point. For this part we also decided to drop the final set as it would require us to calculate the control invariant set to scale it, a very costly operation.

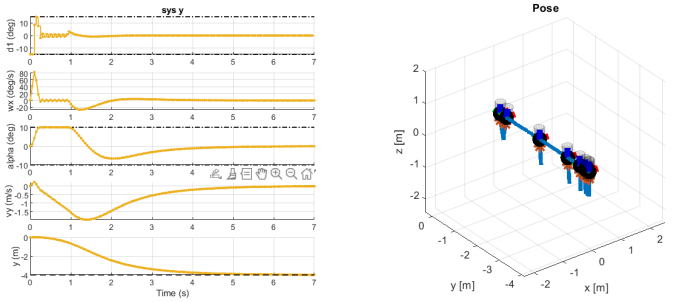


(a) Closed loop

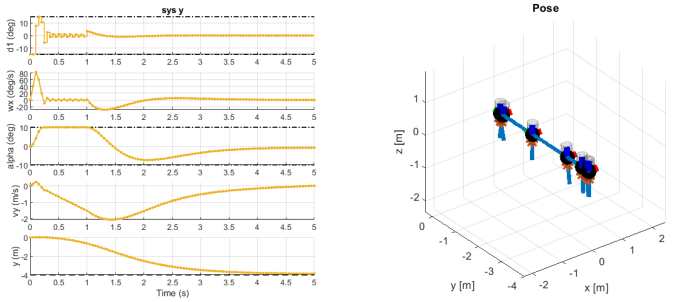


(b) Open loop

Figure 3.8: Linear open and closed loop with tracking for x

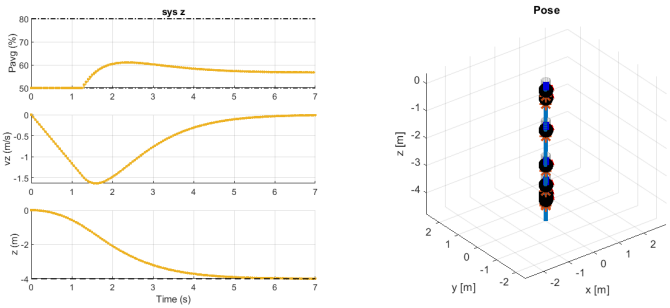


(a) Closed loop

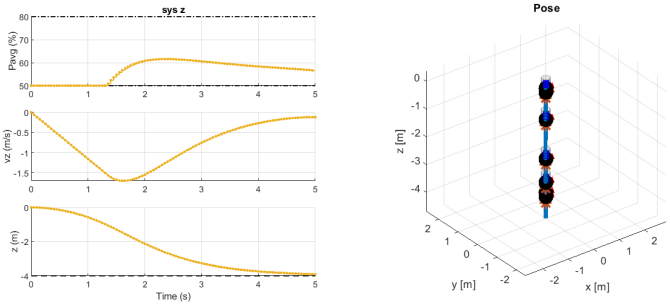


(b) Open loop

Figure 3.9: Linear open and closed loop with tracking for y

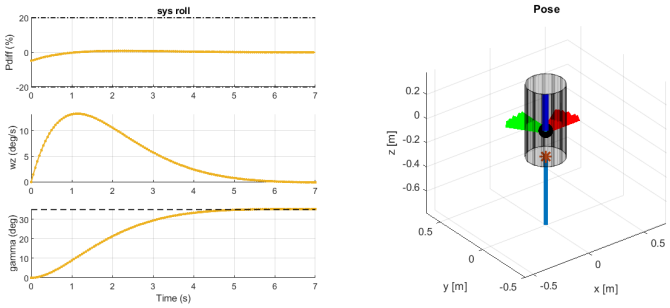


(a) Closed loop

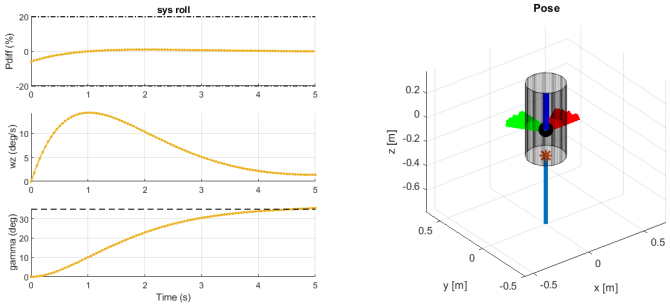


(b) Open loop

Figure 3.10: Linear open and closed loop with tracking for z



(a) Closed loop



(b) Open loop

Figure 3.11: Linear open and closed loop with tracking for γ

Chapter 4

Simulation with Nonlinear Rocket

4.1 Linear controller design

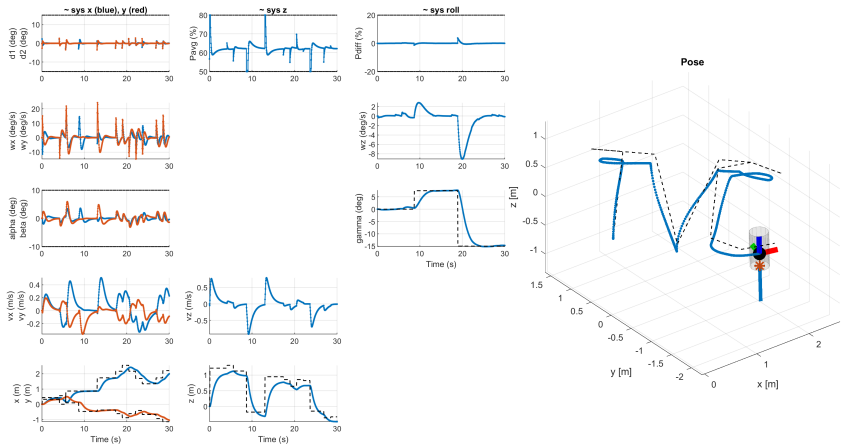


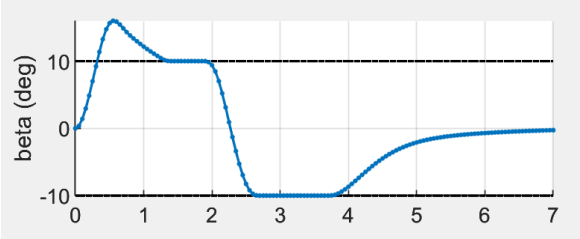
Figure 4.1: TVC reference tracking in non-linear simulation

4.1.1 Z offset

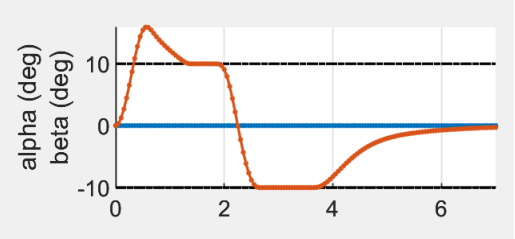
As seen in Figure 4.1, the tracking performance is quite good for x, y and roll but suffers a consequent offset in altitude control not seen in the linear model simulations even tho the mass of the rocket hasn't been tempered with. We think that this is due because nothing in the model is taking motor and propeller inertia and efficiency into account. In addition, thrust vectoring reduces vertical lift a little and isn't modeled because of the linearization and the separation of the controllers. With higher gains on z we can reduce this near constant altitude mismatch at the cost of having higher accelerations in z than in x and y, leading to a less linear path to the target.

4.1.2 Slack variable

Adding soft state constraints was necessary to tune a more aggressive controller in x and y and keep good feasibility while keeping the terminal set. The only constraints to which we can add slack variables are the angles α and β because they only limit the linear model mismatch, meaning it can be violated for a short period of time without leading to unrecoverable states in the non linear simulation. In order to stay in the state constraint range if possible, we had to implement quadratic and linear penalty in the optimization problem. The trade-off between amount and time of constraint violation was determined by trial and error in the non-linear simulation. Figure 4.2a and 4.2b shows the effect of soft constraining β in the x controller for linear and non-linear simulation. We see that in order to go to the reference position [10 0 0 0]' with a horizon of 5 seconds, β has to violate the constraint for almost one second before complying, with a peak at almost 20 deg of pitch. The non-linear simulation confirms that this short violation wasn't a problem.



(a) linear simulation



(b) non-linear simulation

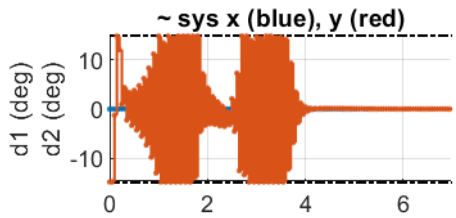
Figure 4.2: Effect of soft state constraint

4.1.3 Additional rate constraint

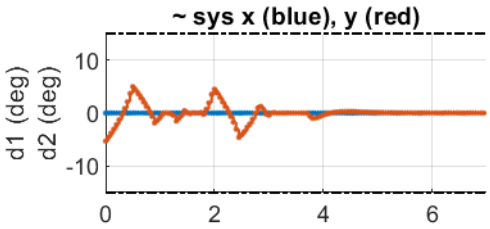
While trying to make aggressive controllers with the help of soft state constraint, high amplitude oscillation behaviour on d_1 and d_2 commands became more and more apparent (Fig 4.3a). Even though the non linear simulation smoothed these hard commands, they are unwanted for realistic deployment and are taken care of with an additional roughly estimated rate constraint.

$$|U_i - U_{i-1}| \leq \phi \qquad \phi = T_s * \left(\frac{\delta d}{\delta t}\right)_{max} \qquad i \in 2, \ldots, N-1$$

We used 20 deg/s as our limit, this is conservative for strong servo motors but not for hydraulic actuators often found on real rocket hopper gimbals. The result is significantly better (Fig 4.3b) and does not reduces performance in the 10 meter in 7 seconds with 5 seconds of horizon test.



(a) without rate constraints



(b) with 20 deg/s rate constraints

Figure 4.3: command oscillations on d_2

Chapter 5

Offset-Free Tracking

5.1 Constant disturbance rejection

5.1.1 Augmented system observer design

We kept the same values for $Q = 100 * I$ and $R = I$ as these seemed to create an objective function that converged well even in the nonlinear simulation. The more relevant tuning parameters were the poles to tune our L matrix dynamics for fast disturbance rejection. Our objective was to try to minimize them enough to have faster dynamics than the controller in order to have a system that would converge as fast as possible whilst remaining stable.

pole number	p_1	p_2	p_3
value	0.03	0.02	0.01

Most of the estimation is done behind the scenes leaving us with the sole task of taking d into account in the constraints in `setup_steady_state_target` and `setup_controller`.

5.1.2 Model mismatch impact and correction

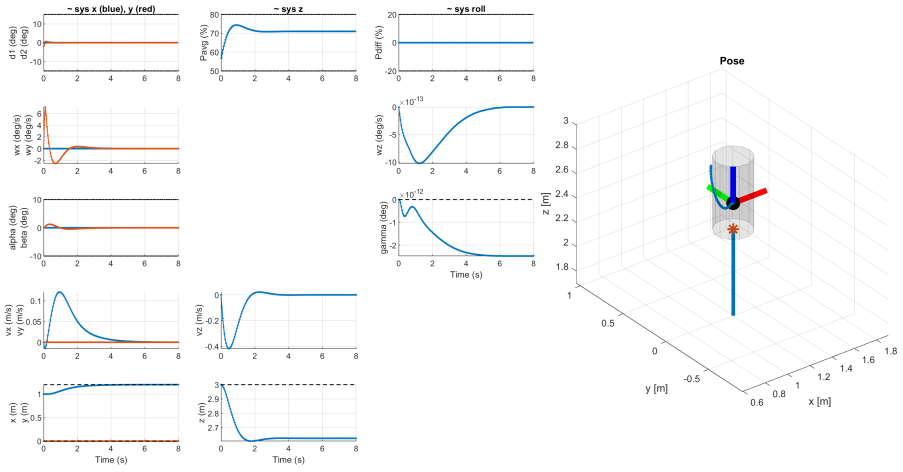


Figure 5.1: Impact on original controller

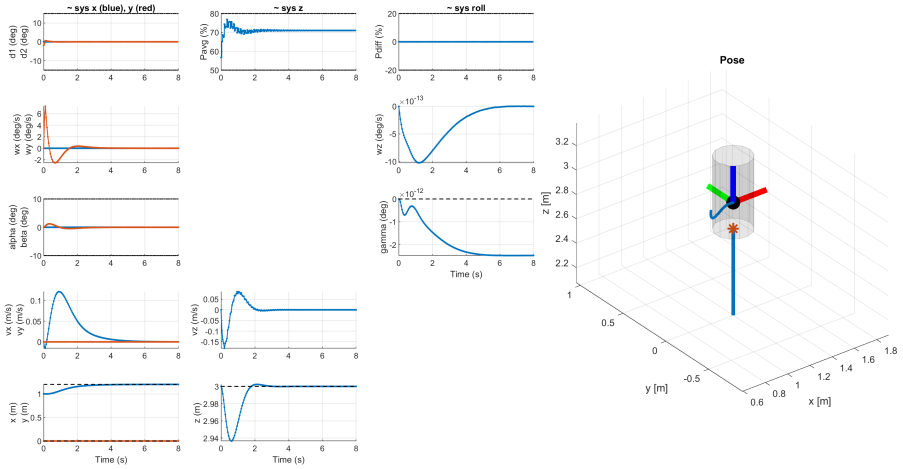


Figure 5.2: effect of disturbance correction

5.2 Dynamic disturbance rejection

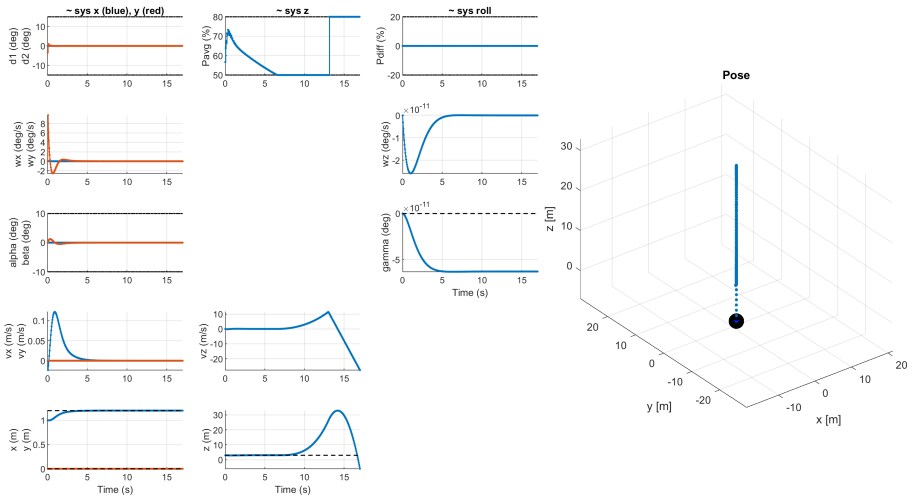


Figure 5.3: Thrust-dependent mass dynamics

5.2.1 Tracking offset and better tracking

The disturbance estimator is trying to model a constant disturbance and so the estimation is always falling behind the changing disturbance. An observer with really fast dynamics could minimize the offset but could not bring it to zero. To reject varying disturbances, they would have to be accounted in the model and considered as an additional parameter in the altitude control system.

$$x_{i+1} = Ax_i + Bu_i + B_d d_i \tag{5.1}$$

with $x_i = \begin{bmatrix} \dot{z} & z \end{bmatrix}^T$, $u_i = P_{avg}$ and $d_i = \begin{bmatrix} d & \dot{d} \end{bmatrix}^T$.

The augmented system becomes a bit more involved. We take $z = \begin{bmatrix} \hat{x}^T & \hat{d} & \dot{\hat{d}} \end{bmatrix}^T$ and construct the

augmented system to be:

$$z_{i+1} = \bar{A}z + \bar{B}u \qquad y = \bar{C}z \tag{5.2}$$

with

$$\bar{A} = \begin{bmatrix} A & B_d \\ 0 & A_d \end{bmatrix} \qquad \bar{B} = \begin{bmatrix} B \\ 0 \\ r \end{bmatrix} \qquad \bar{C} = \begin{bmatrix} C \\ 0 \\ 0 \end{bmatrix} \qquad r \in \mathbb{R} \tag{5.3}$$

and

$$A_d = \begin{bmatrix} 1 & Ts \\ 0 & 0 \end{bmatrix} \qquad \bar{B}_d = \begin{bmatrix} B & 0 \end{bmatrix} \tag{5.4}$$

Note that $\dot{d}$ is not observable because we know that the mass change is somewhat proportional to the input P_avg with `rocket.mass_rate`, abbreviated by r in $\bar{B}$. Since $\dot{d}$ does not impact the states directly, the steady state optimizer doesn't have to be changed.

5.2.2 Trajectory behaviour

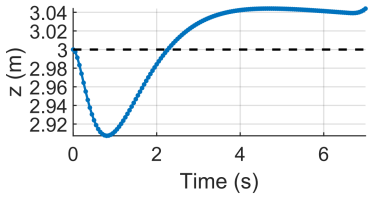


Figure 5.4: Z Trajectory for thrust-dependent mass dynamics

In Figure 5.4 We can see three stages of the trajectory. The first is when the disturbance estimation has not yet converged and is still too low compared to the instant real disturbance, resulting in a drop approximately up to the 1 second mark. After that the controller tries to correct the offset on the position and climbs steadily up to the target. Finally, after 2 second, the disturbance change rate effect on the controller can be seen as the positive offset on z. This is because the disturbance estimation is constantly following the real disturbance but is always falling behind because the dynamics of the disturbance has not been modeled.

Then, after that we can see two more events. after approximately 6.7 seconds, The lower limit on P_{avg} makes it shoot up because the disturbed steady state input is under 50%. after approximately 16 seconds, the motor cease to push because there is no more fuel to propel the rocket and it falls down. We see that the input shoots up to 80% because the disturbance estimation doesn't make sense if the thrust does not follow the input.

Chapter 6

Nonlinear MPC

6.1 Nonlinear MPC

6.1.1 Design procedure

The first step in the design of our NMPC controller is the choice of tuning parameters. Based on good initial results, we set Q to 100 for almost all state variables, and R to 1 for all control variables. For v_z and z we set Q to 400 in order to reduce the tracking error identified in 4.1.1.

In closed-loop, we settle for a horizon length of 2 seconds (corresponding to 40 time steps with $T_s = \frac{1}{20}$), which seems to balance performance and leads to a manageable computation time. We note however that a reduction of the horizon length as far as 0.5 seconds is possible, without compromising too much on performance. Using a short horizon length brings up the usefulness of a terminal weight, which reduces disparities in performance by simulating an infinite horizon. In order to compute the terminal weight, we linearize the system around its steady-state (using `rocket.trim` and `rocket.linearize`). By observing that the linearized dynamics are invariant on the position of the rocket (x, y, z), we can assume that this works relatively well as long as the orientation stays close to zero. The terminal weight is the LQR weight of the discretized linearized model.

The rest of the implementation is a standard reference tracking problem setup, with the main difference being the switch to a Casadi solver, and the control of the full system at once. In the cost function, we penalize the deltas between states/inputs and the corresponding references. Note that the reference input is simply the steady-state input of the system, because the reference state is stationary, and has zero pitch and yaw.

For the system dynamics, we use a Runge-Kutta 4 integrator to minimize integration error, and setup the dynamic equations as a set of equality constraints. The inputs are bounded by their respective limits, and a limit on $|\beta| \leq 75°$ is added to avoid problems near the gymbal lock singularity.

6.1.2 Nonlinear vs linear controller

Observing our NMPC controller in open (fig. 6.1) and closed-loop (fig. 6.2), we can deduce that the tracking performance is quite similar to our previous linear controller. This makes sense since the choice of tuning parameters is similar; any other strong disparity would have been caused by the linear assumption breaking down. We still observe the z offset as explained in 4.1.1. Another reason for the similarity with the linear controllers is the low aggressivity of the system, meaning the α and β angles never need to actually increase past the limits previously used in linear mode. The main difference here is on the roll axis, on which a reference of 50° can be tracked without problem in NMPC (fig. 6.3). If we test the same max reference with our linear controller, it becomes hard to find tuning parameters that actually allow the problem to be solved. Oscillations on the x and y axes appear after the big change in roll at 20 seconds (fig. 6.4). This is because the x and y linear subsystems are far from aligned with the corresponding axes, therefore rendering control impossible.

We note that NMPC would allow for a much more aggressive controller since it is able to cover the whole state space, which might be more visible if the tracked reference did not assume zero speed, pitch and yaw.

Finally, the increase in computation time is a strong drawback when we keep in mind that a controller should work online.

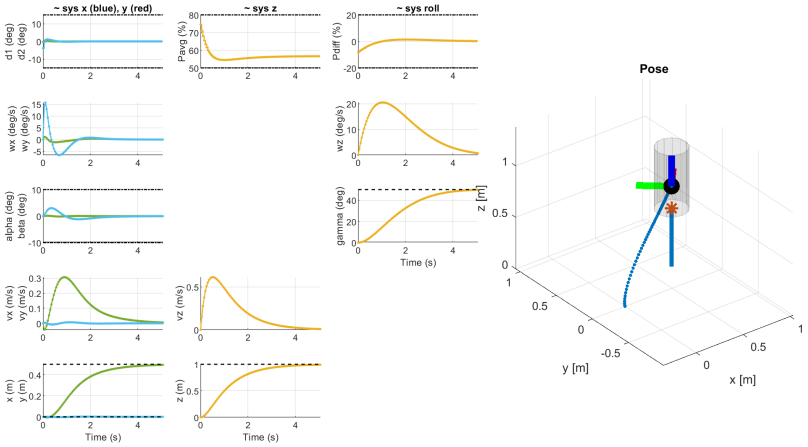


Figure 6.1: NMPC controller in open loop

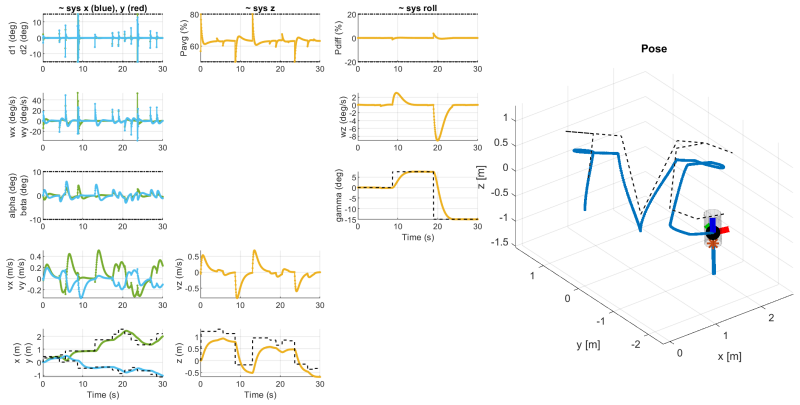


Figure 6.2: NMPC controller in closed loop, with a maximum default roll of 15°

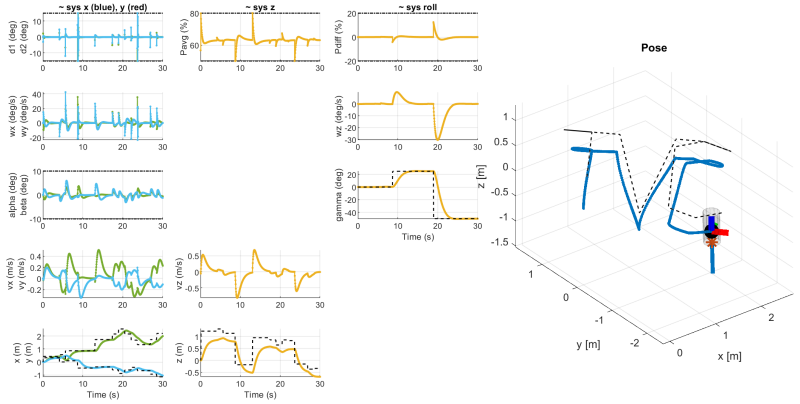


Figure 6.3: NMPC controller in closed loop, with a maximum specified roll of 50°

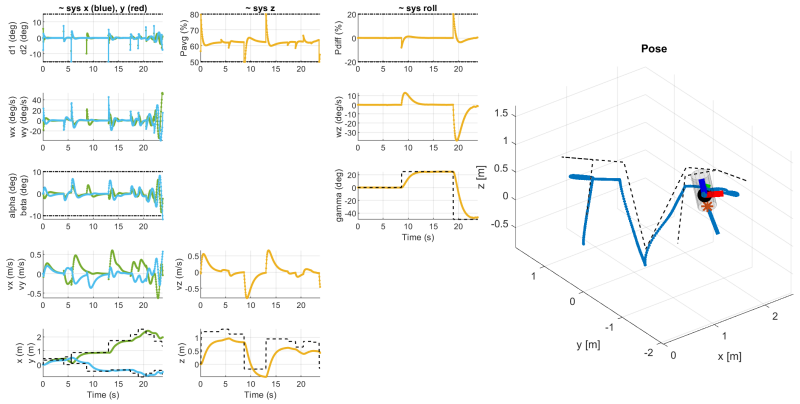


Figure 6.4: Linear MPC controller with a maximum specified roll of 50°

6.2 Nonlinear MPC with delay compensation

6.2.1 Effect of delay

When comparing with a system without delay, we can observe that even a single time step of delay (i.e. 25 milliseconds) is enough to introduce oscillations in the closed-loop-system, especially on δ_1, δ_2, ω_x and ω_y (fig. 6.5). Intuitively, a delayed input u_0^* computed from x_0 can cause the next computed input u_1^* to be of a higher amplitude, because the state x_1 does not correspond to its prediction done during the computation of u_0^*. Depending on the particular set of state and input values, an oscillating feedback loop can appear.

As soon as the delay reaches 2 time steps (fig. 6.6) δ_1, δ_2, ω_x and ω_y enter a persistent oscillating cycle. The positions and angles have visible oscillations, but stay stable and still converge to the right values. From 4 time steps of delay and further, the oscillations get out of hand and increase in amplitude, the closed-loop system becomes unstable, causing the rocket to spin out of control and fall (fig. 6.7 and 6.8).

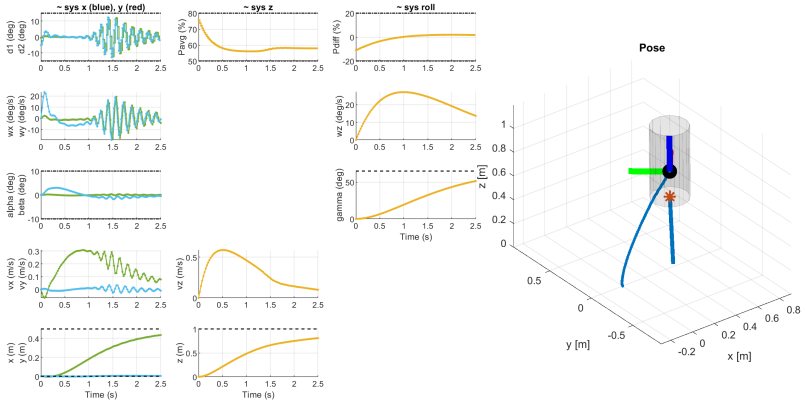


Figure 6.5: NMPC controller with a rocket delay of 1 time step (25 milliseconds)

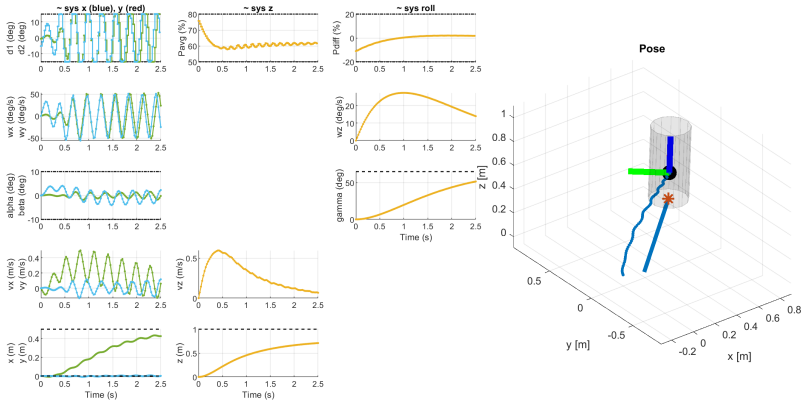


Figure 6.6: NMPC controller with a rocket delay of 2 time steps (50 milliseconds)

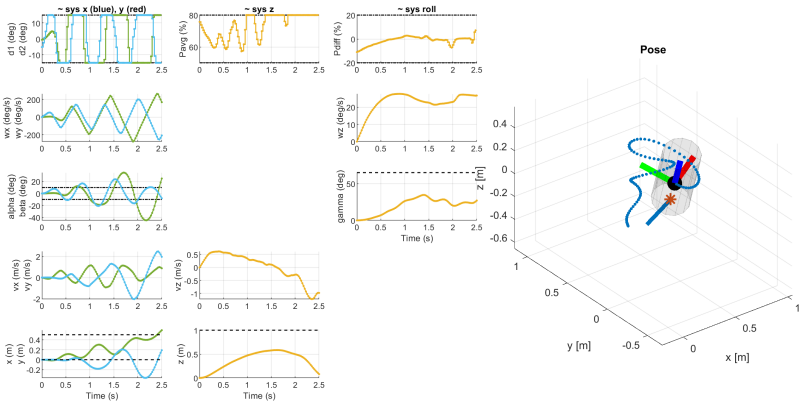


Figure 6.7: NMPC controller with a rocket delay of 4 time steps (100 milliseconds)

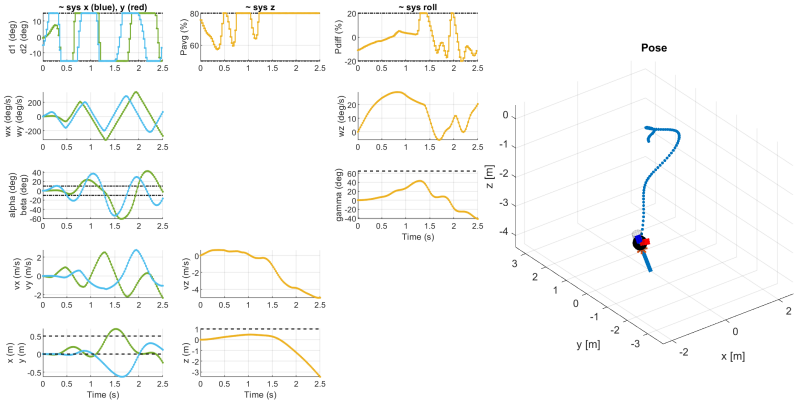


Figure 6.8: NMPC controller with a rocket delay of 5 time steps (125 milliseconds)

6.2.2 Delay compensation

The introduction of delay compensation allows stabilization of the system and tracking of the target. As seen on fig. 6.9, even a partially compensating controller works, while however keeping oscillations on the inputs as observed before. With a fully compensating controller (fig. 6.10), the oscillations disappear. If we try to increase the delay, a positive then negative overshoot of the inputs with respect to the 0-delay case appears, due to the aforementioned lag on the input. Note that some transient oscillations can appear, but this is highly dependent on the tuning parameters and the state of the system. Quite impressively, the system is stabilized even for a delay of 20 time steps (500 milliseconds) (fig. 6.11). We observe however that the initial overshoots become out of hand, and even if they do not destabilize the system, the overall tracking trajectory is poor.

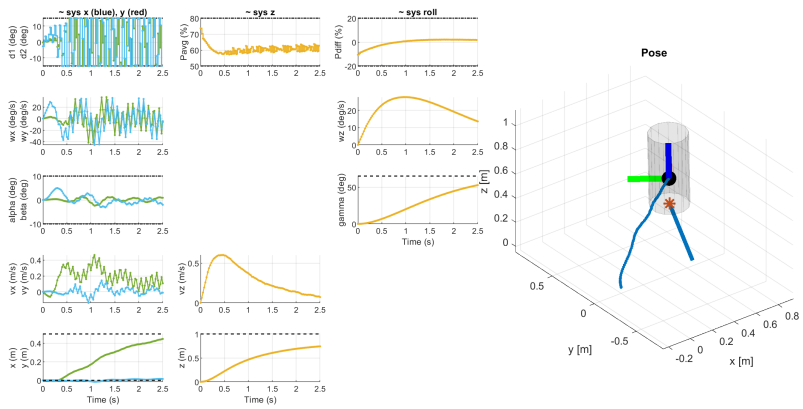


Figure 6.9: NMPC controller with a rocket delay of 5 time steps (125 milliseconds) and a partially compensated delay of 3 time steps (75 milliseconds)

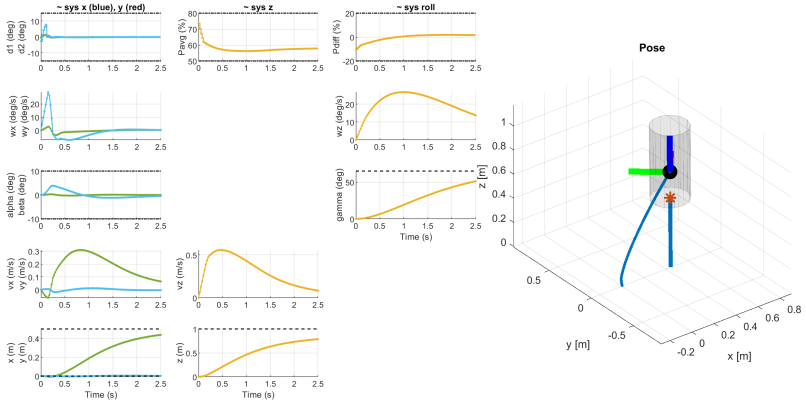


Figure 6.10: NMPC controller with a fully compensated delay of 5 time steps (125 milliseconds)

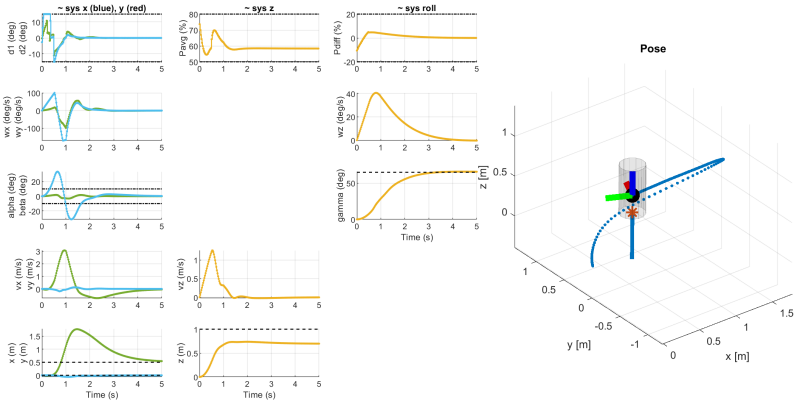


Figure 6.11: NMPC controller with a fully compensated delay of 20 time steps (500 milliseconds)