Quadcopter MPC controller

Model Predictive Control Project

Chevalley Arthur: (SCIPER: 283178)
Mermoud Paco: (SCIPER: 290469)
Pereira Portela Tifanny: (SCIPER: 289149)

Autumn 2020

Contents

1 System Dynamics

In this project, we will develop a MPC controller to fly a quadcopter. In order to do so, we need to define the quadcopter dynamics. The quadcopter is a six DOF system (3 rotations + 3 positions) with four inputs (u_i, with $i = 1, 2, 3, 4$), which are the thrusts of the four rotors. The input vector can modelled as:

$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T \tag{Eq. 1.1}$$

We can describe the system with the following state vector $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} \dot{\theta} & \theta & \dot{\rho} & \rho \end{bmatrix}^T \tag{Eq. 1.2}$$

Where

$$\theta = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix}, \rho = \begin{bmatrix} x & y & z \end{bmatrix} \tag{Eq. 1.3}$$

With θ representing the orientation of the quadcopter as [roll, pitch, yaw] and ρ represents the quadcopter's center of mass position. Hence, the full state vector $\mathbf{x} \in M_{12,1}(\mathbb{R})$ can be written as:

$$\mathbf{x} = \begin{bmatrix} \dot{\alpha} & \dot{\beta} & \dot{\gamma} & \alpha & \beta & \gamma & \dot{x} & \dot{y} & \dot{z} & x & y & z \end{bmatrix}^T \tag{Eq. 1.4}$$

Regarding the inputs of the model, each of the four rotors produces a force ($F_i = k_F u_i$) and a moment ($M_i = k_M u_i$). From these relations, the net body force (F) and body moments ($M_\alpha, M_\beta, M_\gamma$) of the quadcopter can be determined as a function of the input vector $\mathbf{u}$ as follows:

$$\mathbf{v} = \begin{bmatrix} F \\ M_\alpha \\ M_\beta \\ M_\gamma \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \mathbf{u} = T\mathbf{u} \tag{Eq. 1.5}$$

The angular dynamics are defined as:

$$\omega := \dot{\theta} = \begin{bmatrix} \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix}^T \tag{Eq. 1.6}$$

$$\dot{\omega} = I^{-1}(-\omega \times I \cdot \omega + \begin{bmatrix} M_\alpha \\ M_\beta \\ M_\gamma \end{bmatrix}) \tag{Eq. 1.7}$$

The linear dynamics are given by:

$$v := \dot{\rho} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T \tag{Eq. 1.8}$$

$$\dot{v} = -m \cdot g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + F \cdot R(\alpha, \beta, \gamma) \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{Eq. 1.9}$$

Where $R(\alpha, \beta, \gamma)$ is the rotation matrix :

$$R(\alpha, \beta, \gamma) = R_\alpha \cdot R_\beta \cdot R_\gamma$$
$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{pmatrix} \cdot \begin{pmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{pmatrix} \cdot \begin{pmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{(Eq. 1.10)}$$

2 Linearization and Diagonalization

In the first part of the project, we will control a linearized version of the quadcopter. In order to do so, we first need to find a state and input pair $(\overline{x}, \overline{u})$ such that:

$$\dot{x} = f(\overline{x}, \overline{u}) = \vec{0} \quad \text{(Eq. 2.1)}$$

i.e

$$\begin{bmatrix} \ddot{\alpha} & \ddot{\beta} & \ddot{\gamma} & \dot{\alpha} & \dot{\beta} & \dot{\gamma} & \ddot{x} & \ddot{y} & \ddot{z} & \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \quad \text{(Eq. 2.2)}$$

From Eq. 2.2, Eq. 1.6 & Eq. 1.8, we obtain that both ω & v are equal to $\vec{0}$. Hence, from Eq. 1.7, we get:

$$I^{-1} \begin{pmatrix} M_\alpha \\ M_\beta \\ M_\gamma \end{pmatrix} = 0 \Rightarrow M_\alpha, M_\beta, M_\gamma, = 0 \quad \text{(Eq. 2.3)}$$

Then, using theses results in Eq. 1.5, we can find:

$$M_\alpha = k_F \cdot L \cdot (u_2 - u_4) = 0 \Rightarrow u_2 = u_4 \quad \text{(Eq. 2.4)}$$
$$M_\beta = k_F \cdot L \cdot (u_3 - u_1) = 0 \Rightarrow u_1 = u_3 \quad \text{(Eq. 2.5)}$$
$$M_\gamma = k_M \cdot (u_1 + u_3 - u_2 - u_4) = 0 \Rightarrow 2u_1 = 2u_2 \quad \text{(Eq. 2.6)}$$
$$\Leftrightarrow u_1 = u_2 = u_3 = u_4 \equiv u \quad \text{(Eq. 2.7)}$$

Therefore,

$$F = k_F(u_1 + u_2 + u_3 + u_4) = 4k_F u \Leftrightarrow u = \frac{F}{4k_F} \quad \text{(Eq. 2.8)}$$

Then, with Eq. 2.2, Eq. 2.8 and Eq. 1.9, one can obtain the following relation at the steady state:

$$\dot{v} = \begin{bmatrix} Fsin(\beta) \\ -Fsin(\alpha)cos(\beta) \\ -mg + Fcos(\beta)cos(\alpha) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \Leftrightarrow \quad \text{(Eq. 2.9)}$$

$$\left| \begin{array}{l} sin(\beta) = 0 \Leftrightarrow \beta = k\pi, \;\; k \in \mathbb{Z} \\ sin(\alpha)cos(\beta) = 0 \Leftrightarrow \alpha = k\pi, \;\; k \in \mathbb{Z} \\ F = \frac{mg}{cos(\alpha)cos(\beta)} = 4k_F u \Leftrightarrow u = \frac{mg}{4k_F cos(\alpha)cos(\beta)} \end{array} \right. \quad \text{(Eq. 2.10)}$$

Since the inputs u_i need to be positive to satisfy the constraints ($0 \leq u_i \leq 1.5$). The roll and pitch angles (α and β) can be either both equal to zero or both equal to π. However, in the case of linear MPC controllers, we have constraints on α and β: $-0.035[rad] \leq \alpha,\ \beta \leq 0.035[rad]$. Hence, the roll and pitch angles (α and β) are both equal to zero.

Furthermore, from Eq. 2.2 we know that the yaw angle γ and the positions x, y, z are constants (respectively $c_i,\ i = 1, 2, 3, 4$), since their derivatives are set to 0.

Regarding the steady state input $\overline{u}$ and Eq. 2.7, we can extract from Eq. 2.10, that:

$$u_1 = u_2 = u_3 = u_4 = \frac{mg}{4k_F} \quad \text{(Eq. 2.11)}$$

To summarize, the state and input pair $(\overline{x}, \overline{u})$ satisfying Eq. 2.1 are :

$$\overline{x} = \begin{bmatrix} \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \\ \alpha \\ \beta \\ \gamma \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ c_1 \\ 0 \\ 0 \\ 0 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} \quad \text{and} \quad \overline{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \frac{mg}{4k_F} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{(Eq. 2.12)}$$

In order to simplify the notation, note that in the rest of this report, we will use the constants (I, k_F, k_L, L, m, g), and steady state and input pair values used in the MATLAB code, where the constants have be chosen as: $\frac{mg}{4k_F} = \frac{8 \cdot 9.81}{4 \cdot 28} = 0.7007$ and $c_1 = c_2 = c_3 = c_4 = \alpha = \beta = 0$, so that :

$$\overline{x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \text{ and } \overline{u} = 0.7007 \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T \quad \text{(Eq. 2.13)}$$

Now that we determined the steady state and input, we need to calculate the jacobian matrices ($A = \frac{\partial}{\partial \mathbf{x}} f(\overline{x}(t), \overline{u}(t))$ and $B = \frac{\partial}{\partial \mathbf{u}} f(\overline{x}(t), \overline{u}(t))$) representing the partial derivatives of $f(\mathbf{x}, \mathbf{u})$ with respect to the input $\mathbf{u}$ and state $\mathbf{x}$ evaluated at $(\overline{x}, \overline{u})$:

$$A = \begin{pmatrix} 0 & 0.5\dot{\gamma} & 0.5\dot{\beta} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.5\dot{\gamma} & 0 & -0.5\dot{\alpha} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{F}{m}cos(\beta) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{F}{m}cos(\alpha)cos(\beta) & \frac{F}{m}sin(\alpha)sin(\beta) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{F}{m}sin(\alpha)cos(\beta) & -\frac{F}{m}sin(\beta)cos(\alpha) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}\Bigg|_{(\overline{x},\overline{u})}$$

$$B = \begin{pmatrix} 0 & 0.1k_FL & 0 & -0.1k_FL \\ -0.1k_FL & 0 & 0.1k_FL & 0 \\ \frac{2}{30}k_M & -\frac{2}{30}k_M & \frac{2}{30}k_M & -\frac{2}{30}k_M \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{k_Fsin(\beta)}{m} & \frac{k_Fsin(\beta)}{m} & \frac{k_Fsin(\beta)}{m} & \frac{k_Fsin(\beta)}{m} \\ -\frac{k_Fsin(\alpha)cos(\beta)}{m} & -\frac{k_Fsin(\alpha)cos(\beta)}{m} & -\frac{k_Fsin(\alpha)cos(\beta)}{m} & -\frac{k_Fsin(\alpha)cos(\beta)}{m} \\ \frac{k_Fcos(\alpha)cos(\beta)}{m} & \frac{k_Fcos(\alpha)cos(\beta)}{m} & \frac{k_Fcos(\alpha)cos(\beta)}{m} & \frac{k_Fcos(\alpha)cos(\beta)}{m} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\Bigg|_{(\overline{x},\overline{u})}$$

By evaluating A and B at $(\overline{x}, \overline{u})$ one can easily simplfy them. In order to keep a readable report, this simplification will not be shown here. With these two matrices, the linearized model can be expressed as (note that this model is only valid in the vicinity of $(\overline{x}, \overline{u})$):

$$\dot{\mathbf{x}} = A(\mathbf{x} - \overline{\mathbf{x}}) + B(\mathbf{u} - \overline{\mathbf{u}}) \qquad \text{(Eq. 2.14)}$$

In order to break the system into four independent and non-interacting systems, we will use the change of variables introduced in Eq. 1.5: $\mathbf{v} = T\mathbf{u}$. Hence, the linearized system introduced in Eq. 2.14 can be written as:

$$\dot{\mathbf{x}} = A(\mathbf{x} - \overline{\mathbf{x}}) + BT^{-1}(\mathbf{v} - \overline{\mathbf{v}}) \equiv A(\mathbf{x} - \overline{\mathbf{x}}) + B'(\mathbf{v} - \overline{\mathbf{v}}) \qquad \text{(Eq. 2.15)}$$

Using the numerical values and the steady state and input $(\overline{x}, \overline{u})$ given in the MAT-

LAB code, we can compute the matrix $B' = BT^{-1}$:

$$B' = BT^{-1} = \begin{pmatrix} 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & \frac{1}{15} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{8} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{(Eq. 2.16)}$$

As we can see both A and B' matrices are sparse and from them we can extract two independant double integrators (one for γ and one for z) and two pairs of coupled double integrators (one for α and y, and one for β and x). These four subsystems are independent and non-interacting, and that's exactly what we aimed for.

In addition, if one think about the problem from a physics point of view, one can have an intuition about these four subsystems. Indeed, by thinking about the physical behaviour of the quadcopter for small angles, we can understand why this separation occurs. At small angles, the pitch velocity and acceleration ($\dot{\beta}, \ddot{\beta}$), and the velocity and acceleration along the x axis ($\dot{x}, \ddot{x}$) can only be modified by four states: $\dot{\beta}, \beta, \dot{x}, x$. This is quite intuitive, because changing the pitch angle, by applying a body moment M_β, necessarily induces a variation of position, velocity and acceleration along the x axis. This means that the pitch angle and the position along the x axis, as well as their derivatives, are tied. Furthermore, it seems quite intuitive that for small angles, the pitch angle and the position along the x axis do not influence the z-position, y-position, roll angle and yaw angle, as well as their derivatives, which means that the $\dot{\beta}, \beta, \dot{x}, x$ states are independent from all the other states. This gives us our first subsystem.

The same reasoning can be done replacing the pitch angle by the roll angle, the x-position by the y-position, and the body moment M_β by the body moment M_α. Doing this explains the second subsystem.

The third subsystem is even more straight forward, as the altitude and the rate of ascent do not depend on any angles nor the x and y positions, as well as their derivatives. Indeed, the velocity and acceleration along the z axis depends only on the z-position and velocity, and the net body force F.

Finally, seeing that the yaw angle does not have any impact on the pose or speed of the quadcopter, except the yaw angle and rate, it's quite easy to explain the fourth subsystem. Indeed making the quadcopter turn around the z-axis, by applying a net body moment M_γ, only changes the yaw angle or rate which gives us the last subsystem.
Those four subsystems are given bellow.

Subsystem x:

$$\begin{cases} \begin{pmatrix} \ddot{\beta} \\ \dot{\beta} \\ \ddot{x} \\ \dot{x} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 9.81 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \dot{\beta} \\ \beta \\ \dot{x} \\ x \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \cdot M_\beta \\ y = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \dot{\beta} \\ \beta \\ \dot{x} \\ x \end{pmatrix} + \begin{pmatrix} 0 \end{pmatrix} \cdot M_\beta = x \end{cases} \quad \text{(Eq. 2.17)}$$

Subsystem y:

$$\begin{cases} \begin{pmatrix} \ddot{\alpha} \\ \dot{\alpha} \\ \ddot{y} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -9.81 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \dot{\alpha} \\ \alpha \\ \dot{y} \\ y \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \cdot M_\alpha \\ y = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \dot{\alpha} \\ \alpha \\ \dot{y} \\ y \end{pmatrix} + \begin{pmatrix} 0 \end{pmatrix} \cdot M_\alpha = y \end{cases} \quad \text{(Eq. 2.18)}$$

Subsystem z:

$$\begin{cases} \begin{pmatrix} \ddot{z} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \dot{z} \\ z \end{pmatrix} + \begin{pmatrix} \frac{1}{8} \\ 0 \end{pmatrix} \cdot F \\ y = \begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \dot{z} \\ z \end{pmatrix} + \begin{pmatrix} 0 \end{pmatrix} \cdot F = z \end{cases} \quad \text{(Eq. 2.19)}$$

Subsystem yaw:

$$\begin{cases} \begin{pmatrix} \ddot{\gamma} \\ \dot{\gamma} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \dot{\gamma} \\ \gamma \end{pmatrix} + \begin{pmatrix} \frac{1}{15} \\ 0 \end{pmatrix} \cdot M_\gamma \\ y = \begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \dot{\gamma} \\ \gamma \end{pmatrix} + \begin{pmatrix} 0 \end{pmatrix} \cdot M_\gamma = \gamma \end{cases} \quad \text{(Eq. 2.20)}$$

Note that the aforementioned B matrices of each subsystem we derived by hand are equal to the ones given in MATLAB if we use the MATLAB command "[sys_x, sys_y, sys_z, sys_yaw] = quad.decompose(sys_transformed, xs, us)" and not the MATLAB command [sys_x, sys_y, sys_z, sys_yaw] = quad.decompose(sys, xs, us) which is the recommended command to use. This happens, because the inverse of the matrix T is scaled in the *decompose()* function (but then this scaling is undone when the constraints are given in Part 3). Hence, we need to use the recommended command for the simulation to work properly.

3 MPC Controllers for Each Sub-System

In this chapter, we will first design MPC regulators and then MPC tracking controllers for each subsystem.

3.1 MPC regulators

The aim of this section is to design a recursively feasible, stabilizing MPC regulator for each of the four subsystems. In order to do so, we split each infinite horizon problem into two subproblems. The first one is a finite horizon problem (up to time $k = N$), where the state and input constraints are active. The second one is an infinite horizon problem (for $k = N$ to ∞), where the state and input constraints are no longer active.

3.1.1 Infinite horizon problem

To solve this problem, we use the optimal unconstrained LQR control law $u = K_{LQR}x$ to bound the tail of the infinite horizon cost from $k = N$ to ∞. We also define the terminal cost to be the optimal LQR cost: $V_f = x_N^T P x_N$, where P is the solution of the discrete-time algebraic Riccati equation:

$$P = Q + A^t P A^T - A^T P B (R + B^T P B)^{-1} B^T P A \qquad \text{(Eq. 3.1)}$$

From which, we can calculate the LQR control gain:

$$K_{LQR} = -(R + B^T P B)^{-1} B^T P A \qquad \text{(Eq. 3.2)}$$

Then, we can calculate the terminal set X_f using the LQR control law. We choose X_f to be the maximum invariant set for the closed-loop system $x^+ = (A+BK_{LQR}) \cdot x$ subject to $X_f \subseteq \mathbb{X}$ and $KX_f \subseteq \mathbb{U}$. This choice of terminal set ensures recursive constraint satisfaction.

Furthermore, the stage cost is a positive definite function (the Q_{LQR} and R_{LQR} matrices will be chosen to be positive definite). Finally, we know that this terminal cost V_f is a continuous Lyapunov function for the closed-loop system :
$x^+ = (A + BK_{LQR}) \cdot x$ in the terminal set X_f.

Hence, this choice of terminal components also ensures stability.

3.1.2 Finite horizon problem

Every subsystem has a finite horizon problem described as follows:

$$\min_{u} = \sum_{i=0}^{N-1} {x_i}^T Q x_i + {u_i}^T R u_i + {x_N}^T P x_N \quad \text{(Eq. 3.3)}$$

$$\text{s.t} \quad x_{i+1} = A_j x_i + B_j u_i \quad \text{(Eq. 3.4)}$$

$$F x_i \leq f$$

$$G u_i \leq g \quad \text{(Eq. 3.5)}$$

$$x_N \in X_f$$

Where:

- A_j and B_j matrices represent the system dynamics linearized around a steady state and input pair $(\overline{x}, \overline{u})$, with j = 1,2,3,4 corresponding to one of the four subsystems (x,y,z or yaw)
- F and f matrices represent the states constraints
- G and g matrices represent the input constraints
- R and Q matrices represent the cost weights

3.1.3 Constraints implementation

As we found four independent systems, there will be four sets of objectives and constraints. First, let's define these constraints.
The inputs constraints are: (the body moments are expressed in $[kg \cdot m^2 \cdot s^{-1}]$ and the force in [N])

$$-0.3 \leq M_\alpha \leq 0.3 \quad \text{(Eq. 3.6)}$$

$$-0.3 \leq M_\beta \leq 0.3 \quad \text{(Eq. 3.7)}$$

$$-0.2 \leq F \leq 0.3 \quad \text{(Eq. 3.8)}$$

$$-0.2 \leq M_\gamma \leq 0.2 \quad \text{(Eq. 3.9)}$$

The states constraints are: (the angles are expressed in $[rad]$)

$$-0.035 \leq \alpha \leq 0.035 \quad \text{(Eq. 3.10)}$$

$$-0.035 \leq \beta \leq 0.035 \quad \text{(Eq. 3.11)}$$

From the two equations above we can see that, only the subsystems x and y are impacted by the state constraints as they are the only ones being influenced by α or β. Those constraints are set in order to do the small angle approximation which allows us to decompose the system in four subsystems.
The table 3.1.1, in section 3.1.5, shows the different matrices, representing the input and state's polytopic constraints, used to compute the four MPC controllers.

3.1.4 Choice of tuning parameters

The cost matrix on the inputs R and on the states Q for the finite horizon problem, the cost matrix on the inputs R_{LQR} and on the states Q_{LQR} for the infinite horizon problem, and the horizon length N are parameters to be set experimentally.
For the deliverable 3.1, the only specification was for the quadcopter to regulate in around eight seconds starting from the initial state $[x\ y\ z\ \gamma]^T = [2\ 2\ 2\ \frac{\pi}{4}]^T$. Many possibles values for R, Q and N were conform to this specification. Hence, we decided to choose them according to the following criteria:

- Regarding the horizon length, in general, we want a short horizon N, so that we can deploy the MPC control action sufficiently quickly. However, we know that the terminal constraint (induced by the terminal set) reduces the region of attraction. Hence, we need to choose the horizon N to a sufficiently large value to increase this region. So, there is a trade-off between the speed of the control action's deployment and the size of the region of attraction. Since in this project, we don't have any constraints on the speed of the control action's deployment, we set N to a sufficiently large value to increase the region of attraction and make the problem feasible.
- Regarding the choice of the Q matrix, we set it high enough, so that the state would track the reference. Regarding the choice of the R matrix, we set it high enough to get a smooth curve for the evolution of the input.
- Regarding the choice of the Q_{LQR} and R_{LQR} matrices, we chose them to have the biggest terminal invariant set X_f.

3.1.5 Matrices used throughout the 3.1 & 3.2 deliverable

System	G	g	F	f
sys_x	$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$	$\begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0.035 \\ 0.035 \end{bmatrix}$
sys_y	$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$	$\begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0.035 \\ 0.035 \end{bmatrix}$
sys_z	$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$	$\begin{bmatrix} 0.3 \\ 0.2 \end{bmatrix}$	-	-
sys_yaw	$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$	$\begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix}$	-	-

Table 3.1.1: Table of the different matrices representing the constraints

System	Q	Q_{LQR}	R	R_{LQR}	N
sys_x	$\begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	8	8	20
sys_y	$\begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	8	8	20
sys_z	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	9	8	12
sys_yaw	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	9	8	12

Table 3.1.2: Table of the chosen horizon length and cost matrices

Note that the Q_{LQR} and R_{LQR} matrices are only used to compute the terminal set in the deliverable 3.1.

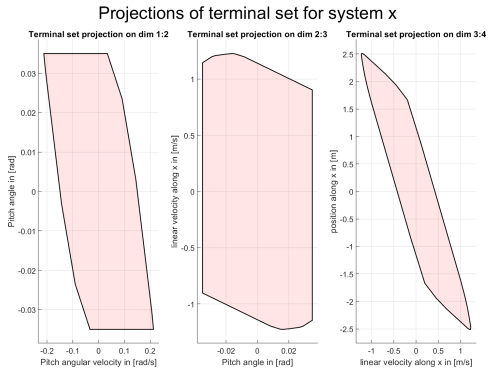
3.1.6 Invariant terminal set

In order to compute the terminal invariant set, we followed the pseudo code given in the figure 3.1.1. This code computes the intersection of the current set with its preset, until they are equal.

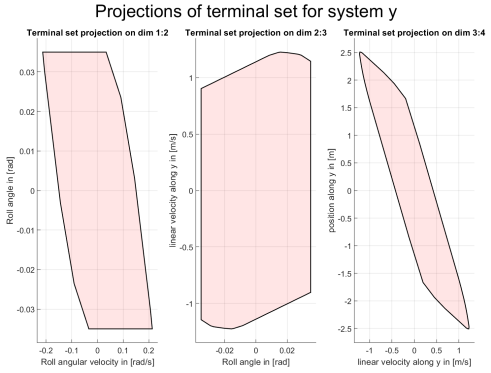
$\Omega_0 \leftarrow \mathbb{X}$
loop
  $\Omega_{i+1} \leftarrow \text{pre}(\Omega_i) \cap \Omega_i$
  if $\Omega_{i+1} = \Omega_i$ **then**
    return $\mathcal{O}_\infty = \Omega_i$
  end if
end loop

Figure 3.1.1: Pseudo code to compute the terminal invariant set

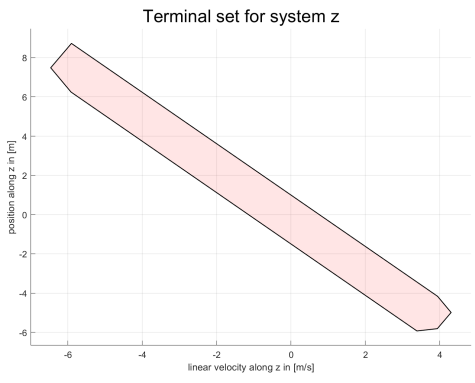
In figure 3.1.2, the four subsystems' terminal invariant sets are shown. As the subsystem x and y are 4-dimensional polyhedrons, we projected them into 3 set of axis to have their 2-dimensional representation. Since, the terminal sets of the subsystems z and yaw are 2-dimensional, they could be directly plotted.



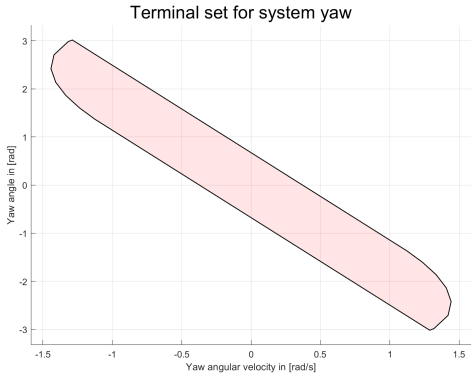
(a) subsystem x



(b) subsystem y



(c) subsystem z

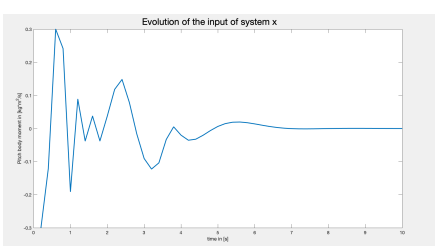


(d) subsystem yaw

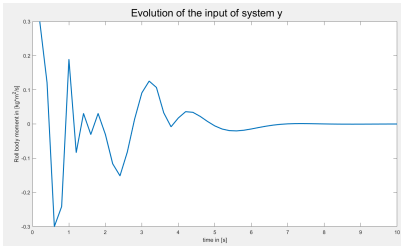
Figure 3.1.2: Terminal sets for each subsystem

3.1.7 Regulation plots

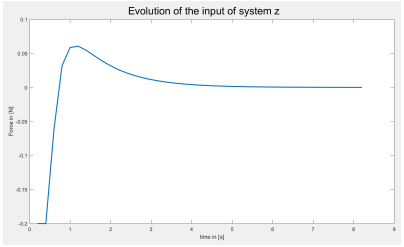
In this section, we plot the regulation performance of each subsystem starting from the initial state $[x \ y \ z \ \gamma]^T = [2 \ 2 \ 2 \ \frac{\pi}{4}]^T$. As mentioned previously, in order to meet the project specifications, the states must reach the origin in less than eights seconds. In addition, the inputs and the states must satisfy the constraints. By looking at the following plots (figure 3.1.3 and 3.1.4), we can confirm that these specifications are met.



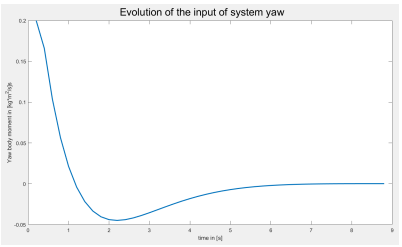
(a) subsystem x



(b) subsystem y

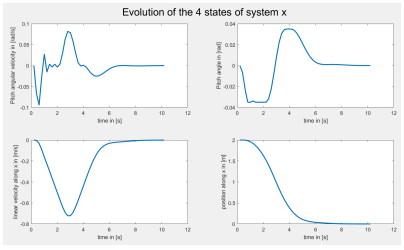


(c) subsystem z

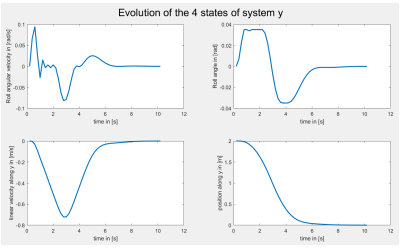


(d) subsystem yaw

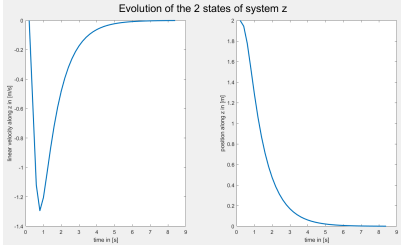
Figure 3.1.3: Evolution of the inputs of each subsystem over time



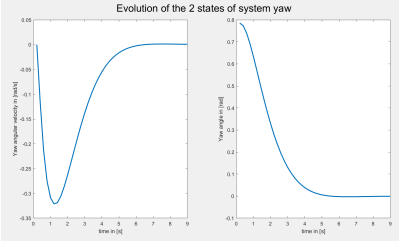
(a) subsystem x



(b) subsystem y



(c) subsystem z



(d) subsystem yaw

Figure 3.1.4: Evolution of the states of each subsystem over time

3.2 MPC tracking controllers

In this section, we extend our four controllers so that they can track constant references, while maintaining recursive feasibility.

We know that, if the initial state lies in a sufficiently small subset of the feasible set and if the horizon length is sufficiently large, then we can remove the terminal set while maintaining stability. By making these assumptions and as it is recommended in the project description, we will drop the terminal set for the rest of the project (until the deliverable 5.1, where we drop the terminal set, but for another reason).

3.2.1 Steady-state target problem

First, in order to design a tracking controller for each subsystem, we need to solve the steady-state target problem.

Assuming that the target problem is feasible, we need to compute the feasible steady-state state (x_s) and input (u_s) corresponding to the reference position r:

$$\min_{u_s} u_s^T R_s u_s \tag{Eq. 3.1}$$

$$\text{s.t.} \quad \begin{bmatrix} I - A_j & -B_j \\ C_j & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ r \end{bmatrix} \tag{Eq. 3.2}$$

$$Fx_s \leq f \tag{Eq. 3.3}$$

$$Gu_s \leq g \tag{Eq. 3.4}$$

Where:

- r is the reference to be tracked
- R_s the cost matrix on the input, set to $R_s = 1$ by default

As the constraints appearing in this problem are the same as the ones used in the MPC regulation problem presented previously, the matrices representing the input and state's polytopic constraints, will be the same as the ones in table 3.1.1.

3.2.2 MPC problem for tracking

Now that we determined the steady-state target (x_s, u_s), we can use them to define the MPC problem for tracking :

$$\min_{u} \sum_{i=0}^{N} (x_i - x_s)^T Q (x_i - x_s) + (u_i - u_s)^T R (u_i - u_s) \tag{Eq. 3.5}$$

$$\text{s.t} \;\; x_{i+1} = A_j x_i + B_j u_i \tag{Eq. 3.6}$$

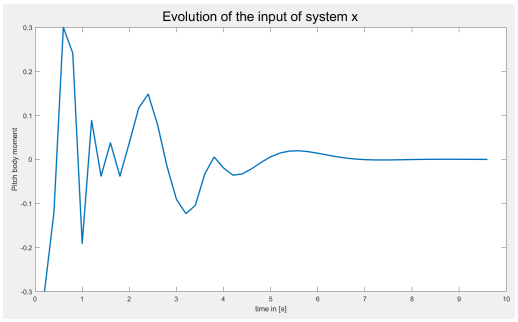
$$Fx_i \leq f \tag{Eq. 3.7}$$

$$Gu_i \leq g \tag{Eq. 3.8}$$

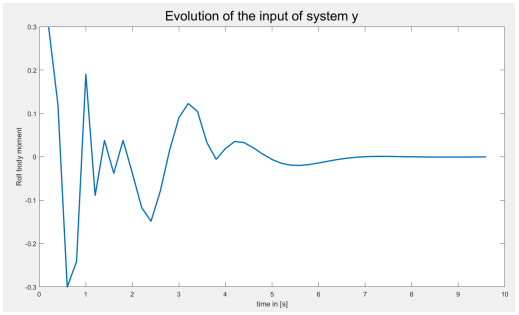
Regarding the choice of the cost matrices Q and R and the horizon length N, we followed the same procedure as the one described in section 3.1.4. It turns out that the optimal (in the sense of our criteria) set of Q, R and N are exactly the same as the ones presented in table 3.1.2.

3.2.3 Tracking reference plots

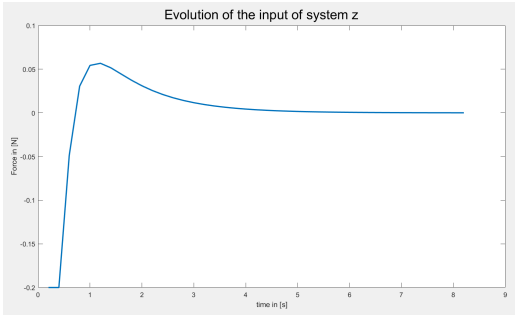
In this section, we plot the tracking performance of each subsystem starting from the initial state $[x\ y\ z\ \gamma]^T = [0\ 0\ 0\ 0]^T$ and aiming to reach the reference state $[x\ y\ z\ \gamma]^T = [-2\ \ -2\ \ -2\ \ \frac{\pi}{4}]^T$ (while the other 8 states regulate). In order to meet the project specifications, the states must reach the reference in less than eight seconds. In addition, the inputs and the states must satisfy the constraints. By looking at the following plots, we can confirm that these specifications are met.



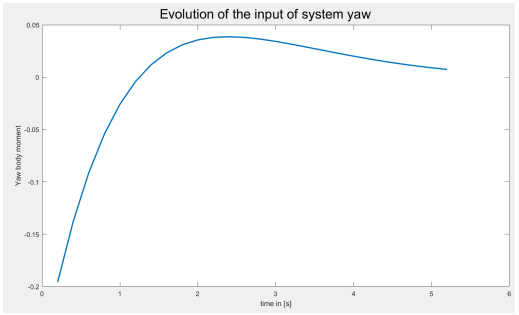
(a) subsystem x



(b) subsystem y

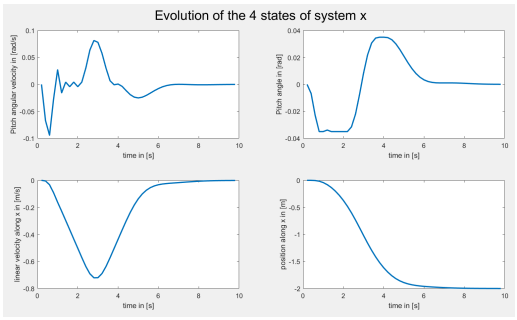


(c) subsystem z

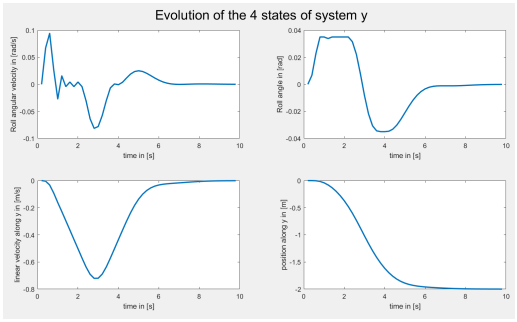


(d) subsystem yaw

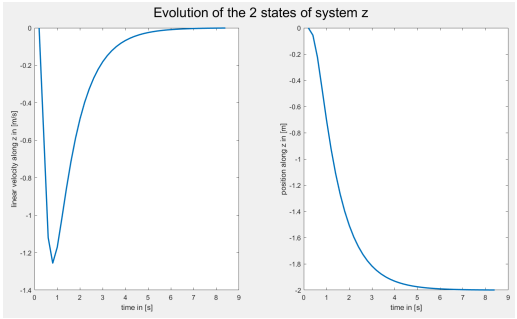
Figure 3.2.1: Evolution of the four subsystems' inputs over time



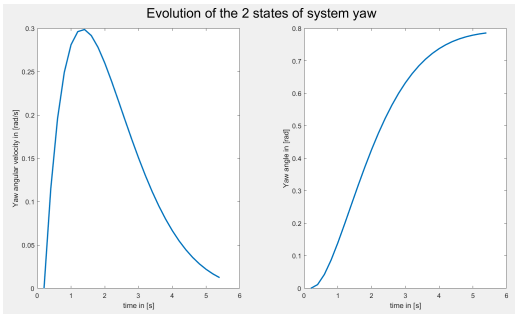
(a) subsystem x



(b) subsystem y



(c) subsystem z

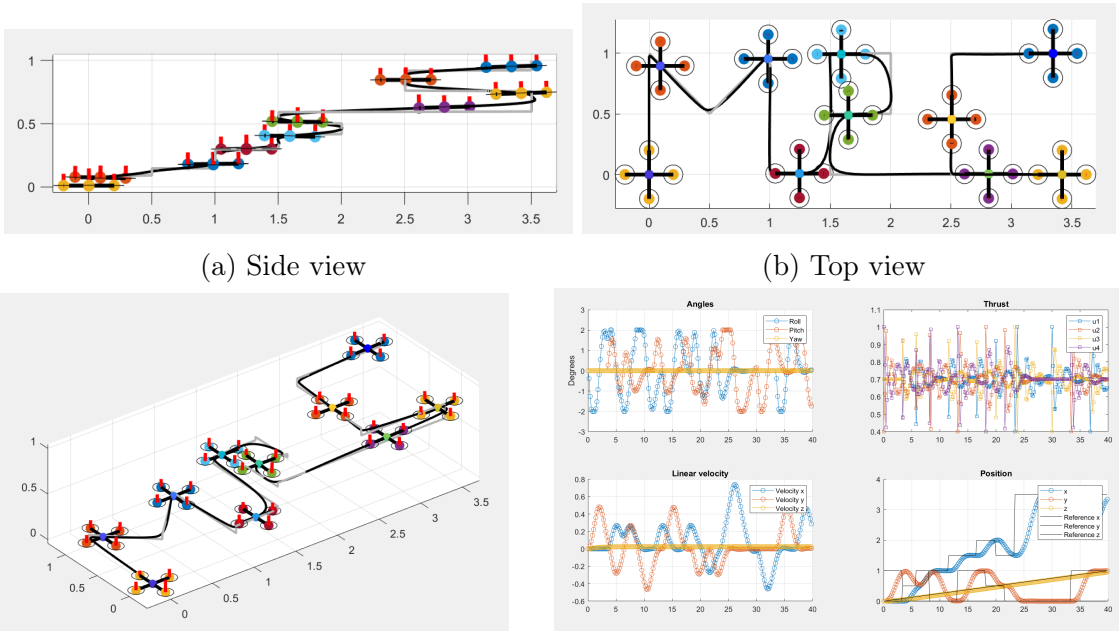


(d) subsystem yaw

Figure 3.2.2: States for reference tracking

4 Simulation with Nonlinear Quadcopter

In this section, we use the four MPC tracking controllers to have the nonlinear quadcopter track a given path. The result of this simulation is shown in figure 4.0.1. As we can see, the thrusts of the four rotors ($u_i,\ \ i = 1, 2, 3, 4$), which are the inputs of the system, satisfy the constraints ($0 \leq u_i \leq 1.5$). Furthermore, we can see that the quadcopter can't follow the path perfectly when the track is highly nonlinear. This phenomenon can clearly be seen on the corners of the "P" letter.



(a) Side view

(b) Top view

(c) Tracking of the path

(d) Inputs & analysis

Figure 4.0.1: Simulation results of the linearize model

The set of optimal (in the sense of our criteria presented in section 3.1.4) cost matrices (Q and R) and the horizon lengths N used in this simulation can be found in the following table:

	sys_x	sys_y	sys_z	sys_yaw
Q	$\begin{bmatrix} 25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$	$\begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix}$	$\begin{bmatrix} 50 & 0 \\ 0 & 20 \end{bmatrix}$	$\begin{bmatrix} 10 & 0 \\ 0 & 40 \end{bmatrix}$
R	20	20	20	20
N	30	30	30	30

Table 4.0.1: Table of the chosen horizon length and cost matrices

5 Offset-Free Tracking

In this section, we assume that the mass of the quadcopter changes, and the aim is to extend the z-controller to compensate for this change. Since we will need to use an observer to estimate the offset $\hat{d}$ and the system's state $\hat{x}$, we can no longer ensure constraint satisfaction and hence, we can drop the terminal set.

5.1 Steady-state target problem

First, in order to design an offset-free tracking controller for the subsystem z, we need to solve the steady-state target problem.

Assuming that the target problem is feasible, we need to compute the feasible steady-state state (x_s) and input (u_s) corresponding to the reference position r:

$$\min_{u_s} u_s^T R_s u_s \tag{Eq. 5.1}$$

$$\text{s.t.} \quad \begin{bmatrix} I - A_z & -B_z \\ C_z & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} B_d d_s \\ r \end{bmatrix} \tag{Eq. 5.2}$$

$$Gu_s \leq g \tag{Eq. 5.3}$$

Where:

- d_s is the steady state disturbance and the best forecast for d_s is the current estimate: $d_s = \hat{d}$.
- $G = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $g = \begin{bmatrix} 0.3 \\ 0.2 \end{bmatrix}$

5.2 MPC problem for offset-free tracking

Now that we determined the steady-state target (x_s, u_s), we can use them to define the MPC problem for offset-free tracking, where d_i is a constant, unknown disturbance:

$$\min_u \sum_{i=0}^{N} (x_i - x_s)^T Q (x_i - x_s) + (u_i - u_s)^T R (u_i - u_s) \tag{Eq. 5.1}$$

$$\text{s.t} \;\; x_0 = \hat{x} \tag{Eq. 5.2}$$

$$d_{i+1} = d_i = \hat{d} \tag{Eq. 5.3}$$

$$x_{i+1} = A_z x_i + B_z u_i + B_z d_i \tag{Eq. 5.4}$$

$$Gu_i \leq g \tag{Eq. 5.5}$$

Regarding the choice of the cost matrices Q and R and the horizon length N, we followed the same procedure as the one described in section 3.1.4. It turns out that the optimal (in the sense of our criteria) set of Q, R and N are exactly the same as the ones presented in table 4.0.1.

5.3 Estimator dynamics

Based on the augmented model (see Eq. 5.3 and Eq. 5.4), we can design the following state and disturbance estimator:

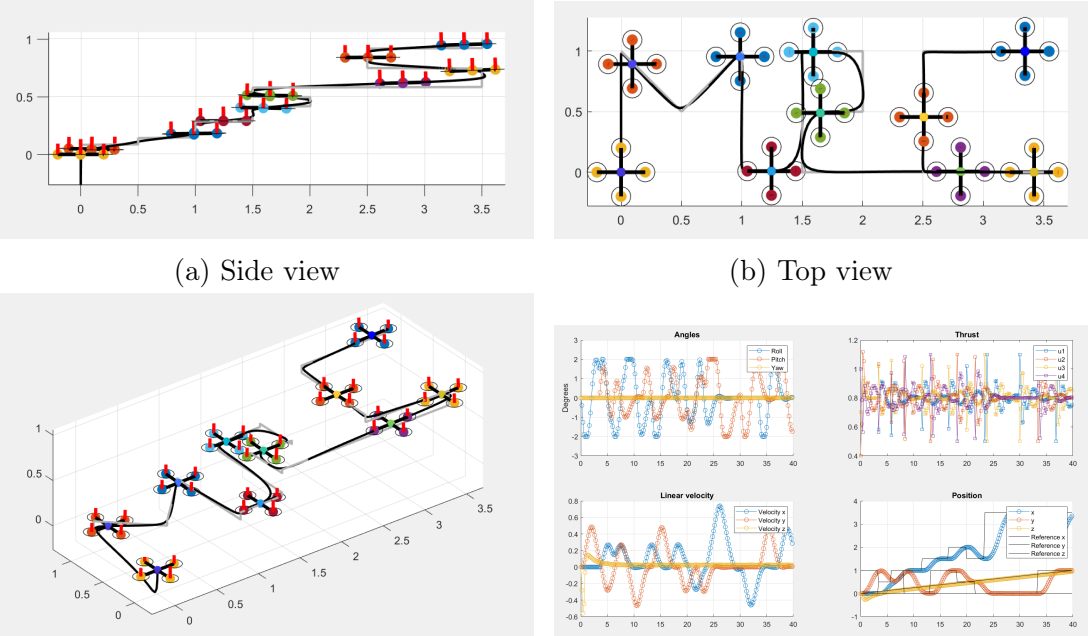
$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{d}_{k+1} \end{bmatrix} = \begin{bmatrix} A_z & B_z \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_k \\ \hat{d}_k \end{bmatrix} + \begin{bmatrix} B_z \\ 0 \end{bmatrix} u_k + \begin{bmatrix} L_x \\ L_d \end{bmatrix} (C\hat{x}_k - y_k) \qquad \text{(Eq. 5.1)}$$

Where the observer gain $\mathrm{L} = [L_x, L_d]^T \in M_{3,1}(\mathbb{R})$ is found by using the MATLAB *place* function. This function requires a set of poles, that we set inside the unit circle in order to have stable error dynamics converging to zero. This set of poles can be freely chosen inside the unit circle, but we carried out multiple tests to choose the best system dynamics. In theory, if we want to be robust against noise, we should choose poles close to the unit circle, but this leads to slower response. In contrast, with poles set closer to the origin, we will better stick to the path, but this will lead to higher inputs and more sensitivity to noise. Since in this project there is no noise, poles can be chosen close to the origin. To determine the poles we started with very small poles values, in order to have fast response and little drop at the start, and we increased them until we found good dynamics. Our final chosen poles are [0.1 0.15 0.2].

5.4 Tracking Plot

This section shows the results of the simulation of the system using our 3 tracking controllers (for system x,y and yaw) and our offset-free tracking controller (for system z), in order to compensate a constant input bias of -0.1.

As we can see on figure 5.4.1, our offset-free tracking controller z is able to reject the disturbance, since the quadcopter can track setpoint references with no offset. Also, the figures 5.4.1a and 5.4.1c show that the quadcopter isn't able to follow perfectly the path at the very beginning of the simulation. This can be explained by the fact that, the first estimations of the observer do not match the real states, but after some time the observer properly works and the drone follows nicely the path.



(a) Side view

(b) Top view

(c) Lateral view

(d) States and inputs

Figure 5.4.1: Offset-free tracking simulation results

6 BONUS: Nonlinear MPC

The implementation of the nonlinear module was made using CASADI. The nonlinear MPC controller (NMPC) take the full state of the quadcopter as input ($x \in M_{12,1}(\mathbb{R})$) and provide four propeller speeds ($u \in M_{4,1}(\mathbb{R})$). The aim of this section is to track a path, which is described by the variable ref.

6.1 Discretization

In order to compute the nonlinear MPC controller, we first discretize the system $\dot{x} = f(x, u)$ with the Runge-Kutta-4 method. This iterative method uses the Euler method and consists to approximate a function with a sum of four weighted average multiplied by an increment h as shown in Eq. 6.1 and illustrated with figure 6.1.1. Since the inputs u are considered to be constant during a time step (i.e for $t \in [kh, (k+1)h]$, $k \in \mathbb{R}^+$), we only discretize the function around the X parameter.

$$\begin{cases} k_1 = quad.f(X, U) \\ k_2 = quad.f(X + \frac{h \cdot k_1}{2}, U) \\ k_3 = quad.f(X + \frac{h \cdot k_2}{2}, U) \\ k_2 = quad.f(X + h \cdot k_3, U) \\ x_{k+1} = X + h \cdot (\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}) \end{cases} \quad \text{(Eq. 6.1)}$$

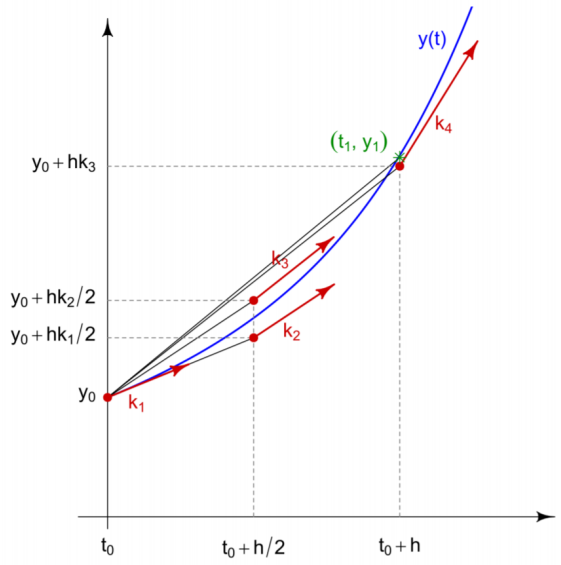


Figure 6.1.1: Illustration of the Runga Kutta 4 method

6.2 NMPC controller

The optimisation problem of the nonlinear MPC controller is the following:

$$\min_{u} \sum_{i=0}^{N} (x_i - ref)^T Q (x_i - ref) + {u_i}^T R u_i \quad \text{(Eq. 6.1)}$$

$$\text{s.t } x_{i+1} = f_{discrete}(x_i, u_i) \quad \text{(Eq. 6.2)}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T \leq u_i \leq \begin{bmatrix} 1.5 & 1.5 & 1.5 & 1.5 \end{bmatrix}^T \quad \text{(Eq. 6.3)}$$

As one can see, the system only have constraints on the inputs. The state constraints were dropped, because we don't need to work in a small angle approximation anymore, since we are dealing with the nonlinear case.

6.3 Choice of tuning parameters

In this section, we explain how we chose the values of the cost matrices (Q and R), the horizon length N and The time step h.

- The time step was chosen not too small, because this would have been a source of problems, as the simulator is applying the inputs every 0.2 seconds. For example, if we had chosen the time step to be 0.1 seconds, the system would think that the simulator would apply the inputs calculated after 0.1 seconds, but in fact, it would be applying the ones calculated after 0.2 seconds. The time step was also chosen not too big, because it would lead in a less accurate discretization. After some simulations, $h = 0.26$ seconds appeared to be the choice giving us the best results.
- To choose the best Q and R matrices, we started our simulations using identity matrices. Then, each diagonal element of the Q matrix was modified to follow the path more accurately. Simulation after simulation, the cost associated to each state, that was not following sufficiently well the path was increased, in order to have a better tracking. Once the Q matrix was found, the R matrix was tweaked to have less inputs constantly jumping from one extreme value to another (i.e from 0 to 1.5).
- For the horizon length, we tested again multiple values. We know that need to have a sufficiently large horizon to insure feasibility but not too large, because it would be computationally too expensive.

The final choices for these parameters are given in the following table:

	NMPC
Q	$\begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 20 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 200 \end{bmatrix}$
R	$\begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$
N	20
h	0.26 [s]

Table 6.3.1: Table of the chosen horizon length and cost matrices

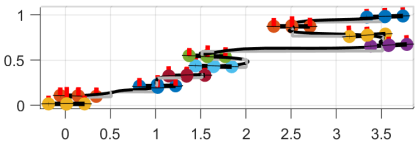
6.4 MATLAB implementation

To implement the nonlinear MPC controller, we used two different approaches. In the first one, we redefined by hand the system dynamics. The second one uses the quad files given with the project.

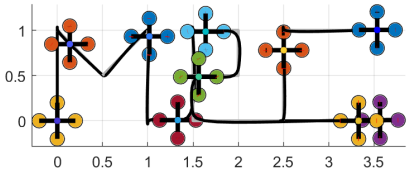
The first approach, which is not using the quad architecture is much quicker than the the second approach and gives the same result! This is why we highly advise to use the first nonlinear controller (see MATLAB code).

6.5 Tracking Plot

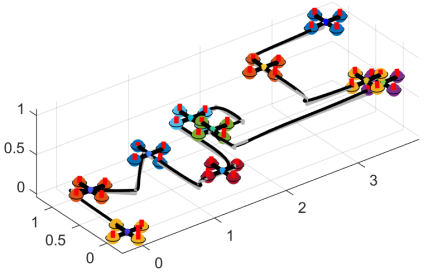
In this section we plot the simulation results of the nonlinear MPC controller following a given path.



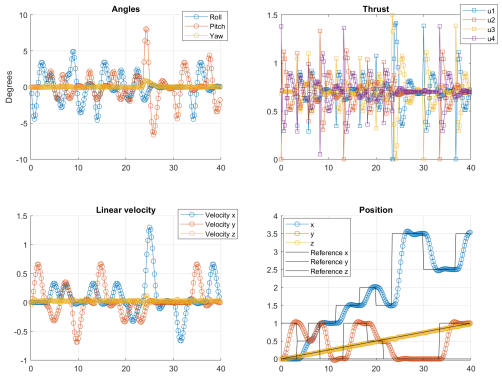
(a) Side view



(b) Top view



(c) Tracking of the path



(d) States & Inputs

Figure 6.5.1: Non-linear MPC simulation results

As we can see, the results of the nonlinear controller are better than the ones we got with the linear controllers of the last sections. Indeed, the quadcopter using the NMPC controller is able to follow the path perfectly even when the track is highly nonlinear (on the corners of the "P" letter for example), which was not the case for the linear controllers.

This amelioration is directly linked to the fact that, the nonlinear controller do not approximate the system with a small angle approximation, which means that the dependencies between each state will appear in the model. Getting rid of the small angle approximation allows the quadcopter to follow much more aggressive trajectories, hence giving better tracking results.

References

[1] Class ME-425, *Model Predictive Control* by professor Colin Jones, EPFL, 2020