MODEL PREDICTIVE CONTROL

PROJECT

CONTROLLING A QUADCOPTER WITH MPC

Pablo Guilland, Urbain Lesbros, Philipp Spiess

Group 61

Scipers: 246550, 240678, 253998

January 12, 2020

Contents

1 System Dynamics

The system is defined by a state vector $\mathbf{x}$ containing 12 states describing the angular speeds and angles in the mobile frame (attached to the Quad), and velocities and positions in the fixed frame. It has as well 4 inputs regrouped in a vector $\mathbf{u}$. They represent the motor thrusts from which can be derived, throughout linear projection, the forces and moments applied on the Quad: F, M_α, M_β and M_γ.

This allows to represent the system as a state space continuous time equation. This equation captures the non-linear dynamics of the system. It has the following form:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

We will see that this equation can be simplified into 4 linear sub-systems under certain conditions in order to utilize Model Predictive Control techniques to make it follow a path.

2 Linearization and Diagonalization

Firstly, we need to linearize the system into a single linear state-space system such that,

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

This is only possible for steady-state when $0 = f(x_s, u_s)$, meaning that all the accelerations and velocities are zero, thus the robot is motionless. The computation of pairs (x_s, u_s) is done by calling the function `quad.trim()`. The main system is then decomposed into 4 independent sub-systems thanks to a change of variable such that $\mathbf{u} = T^{-1}\mathbf{v}$. The vector $\mathbf{v} = [F, M_\alpha, M_\beta, M_\gamma]^{\mathrm{T}}$ becomes the new input vector and each of its components is an input of one of the sub- systems.

2.1 Transformation of the system

The fist step to understand how the linearization and transformation works, is to analyze the matrices A, B and the steady-state pair returned by `quad.trim()`. The returned steady-state is $\mathbf{x_s} = \vec{0}$, $\mathbf{u_s} = [0.7,\ 0.7,\ 0.7,\ 0.7]^{\mathrm{T}}$. It leads to $f(\mathbf{x_s}, \mathbf{u_s}) = 0$ because the position being the null vector, it means that robot is horizontal so that the motors only have to compensate for the gravitational force by having all the same thrust of 0.7. Intuitively, the robot can have any Cartesian position x, y, z and yaw angle γ in order to be in steady-state, while u_s remains the same. This is because they don't affect the acceleration nor velocity components and it can be confirmed by looking at the matrix A that contains only zeros in columns 6, 10, 11 and 12, thus the locations of x, y, z and yaw in state vector $\mathbf{x}$. Also, having a non-zero pitch angle would produce a non-zero projected force F along x-axis, thus an acceleration that would be very hard to compensate for.

The transformation is needed to get 4 independent systems because, despite being independent, the 4 non-zero rows of the matrix B don't represent a canonical base of the sub-space $\mathbb{R}^4$. It involves than when applying an input $\mathbf{u} = [1,\ 0,\ 0,\ 0]^T$, it will influence same dimensions than $\mathbf{u} = [0,\ 0,\ 1,\ 0]^T$. This observation are easily deductible by looking at B.

$$B = \begin{bmatrix} 0 & 0.56 & 0 & 0.56 \\ -0.56 & 0 & 0.56 & 0 \\ 0.73 & -0.73 & 0.73 & 0.73 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3.5 & 3.5 & 3.5 & 3.5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B' = BT^{-1} = \begin{bmatrix} 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.067 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.125 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The change of variable allows to rewrite the new matrix BT^{-1} as B'. This matrix has only one component per row and per dimension in the non-zero rows, therefore we can now divide the main system into 4 non-interacting systems. Indeed, each dimension of the input $\mathbf{v}$ won't interact with the others when producing the output. The base isn't canonical though, otherwise all the non-zero values should be equal to 1, but it is sufficient for achieving our goal. The transformed matrix B' will be refered as B in the rest of the report.

3 MPC Controllers for each sub-system

In this part, a recursively feasible stabilizing MPC controller that can track step references is designed for each sub-systems x, y, z, yaw. The sub-systems are discretized with a sampling period $T_s = 0.2$ seconds in order to design the MPC controllers.

3.1 MPC Regulators

The implemented MPC formulation uses the quadratic performance measure with constraints on states x_i and inputs u_i and the terminal state x_N. The dynamics of the system are linear and discrete for the sake of the model, although the robot is a physical continuous system. It means that the MPC model given below is just an approximation useful to compute the first control input in the horizon, i.e. u_0^*, at each iteration of the simulation.

$$\begin{aligned} &\min_{\mathbf{u}} \ \textstyle\sum_{i=0}^{N-1} x_i^T Q x_i + u_i^T R u_i + x_N^T Q_f x_N \\ &\text{s.t.} \quad x_i \in \mathbb{X} \qquad i \in \{1,\dots,N-1\} \\ &\text{s.t.} \quad u_i \in \mathbb{U} \qquad i \in \{0,\dots,N-1\} \\ &x_N \in \mathscr{X}_f \\ &x_{i+1} = Ax_i + Bu_i \\ &x_0 = x \end{aligned}$$

Recursive constraint satisfaction is achieved by assuming feasible initial states and guaranteeing feasible states along the closed-loop trajectory. Concretely, before $i = N$, the constraints are checked for every prediction in the finite horizon. Then for the prediction at $i = N$ (producing terminal weight), it is the terminal set constraint that is checked by the solver. This process is repeated at each iteration of the

closed-loop trajectory. The solver uses the implementation of the equations above to return the first input in the computed control sequence u^*, therefore it works like a black-box.

We choose N to be 40 which corresponds to the settling time divided by the the period $T_s = 0.2$ s. This value makes sense because thereby, the prediction in open-loop horizon converges in the desired settling time.

We fix Q to $10 \cdot I_{n_x}$ (identity matrix). Then, R is tuned for 3 different values in logarithmic scale. Increasing R smooths the input u, being a force or a torque, as showed in Figure 1. It means that the input energy is minimized. Therefore, we decide to keep $R = 100$ as a suitable value for all the controllers. The trajectory with respect to angle and position isn't influenced by this parameter.

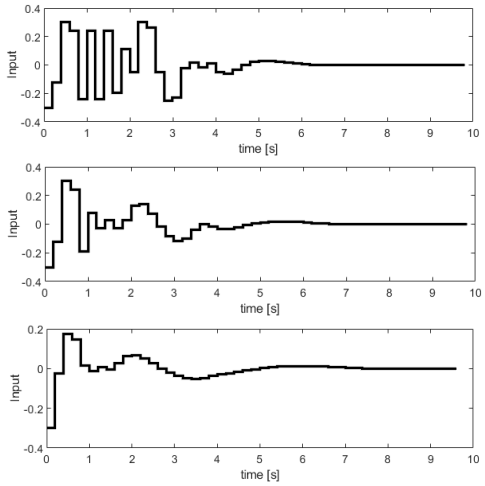
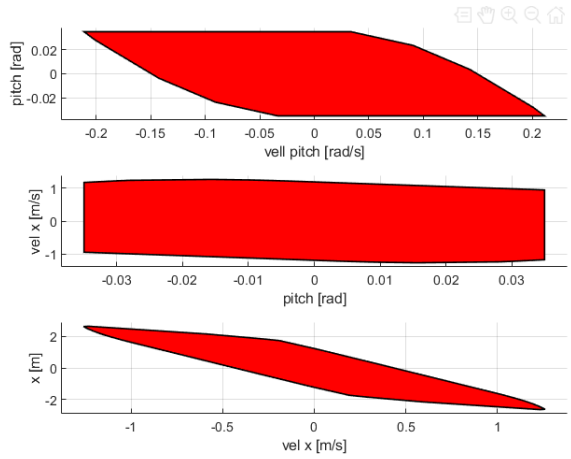
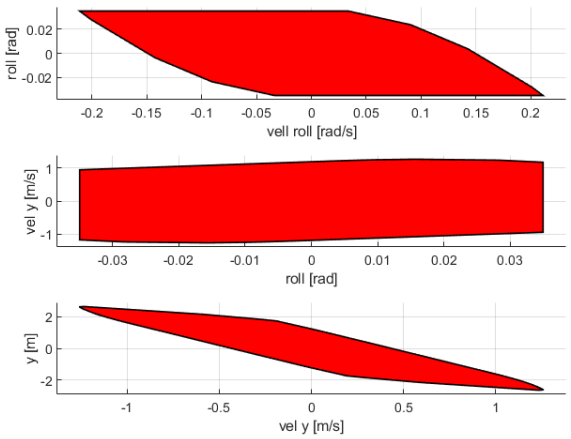


Figure 1: Input of subsystem x for R=1 (top), R=10 (middle) and R=100 (bottom)

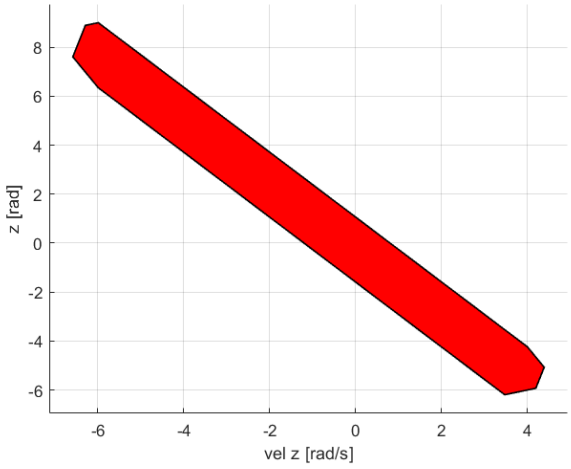
Figure 2 shows the projection of the polytopes of the terminal invariant sets starting at horizon N. The terminal sets were obtained by computing the optimal (unconstrained) LQR controller for the tail of the "fake" infinite horizon for $N \to \infty$. The optimal K_{LQR} is then used to compute the set of constraints serving as the input of the maximal invariant set algorithm implemented right after. The terminal Q_f is extracted as well from the optimal LQR solution. The computation of the terminal sets depends on Q and R in our implementation, only the terminal cost depends on N. The terminal set size increases as R. The horizon length N is big enough to allow the use of this infinite horizon approximation.



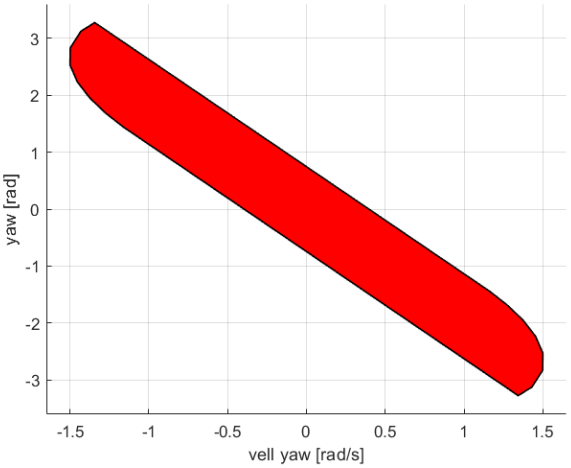
(a) Subsystem x



(b) Subsystem y



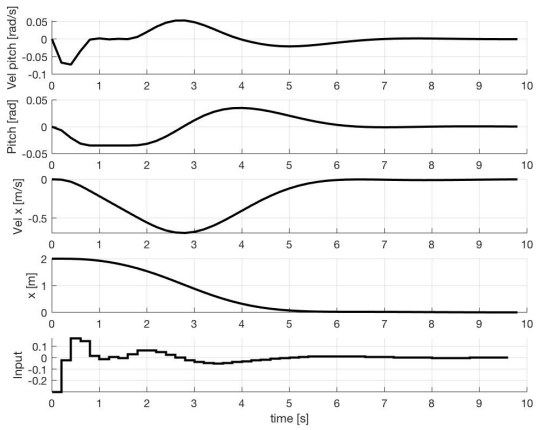
(c) Subsystem z



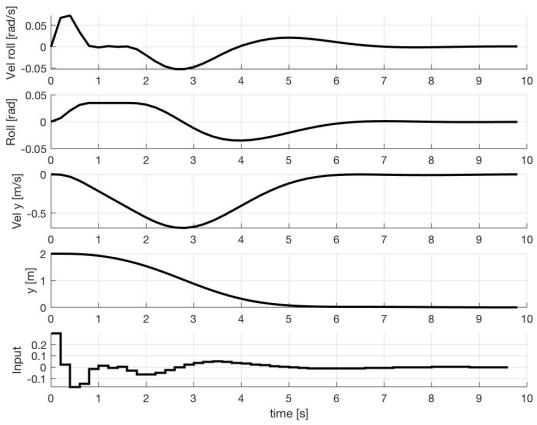
(d) Subsystem yaw

Figure 2: Plots of the invariant sets for each sub-system controller. Chosen projected dimensions are 1-2 plus 2-3, 3-4 for controller x and y

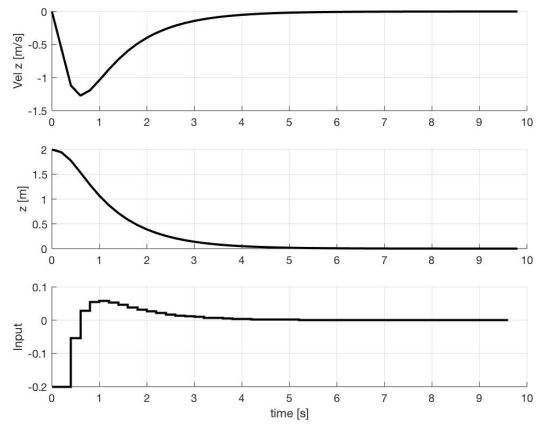
Figure 3 shows that each model allows the system to stabilize to origin with the above parameters starting 2 meters away (for x, y, z) or with an initial angle of 45 degrees (for yaw). As expected, we observe the convergence of each sub-system towards the origin for every dimension.



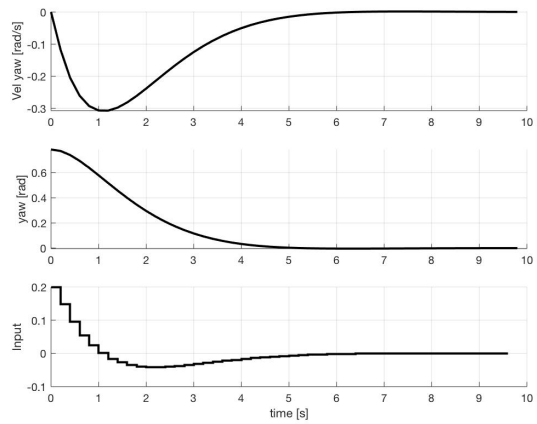
(a) x controller



(b) y controller



(c) z controller



(d) yaw controller

Figure 3: Plots of each controller stabilizing to the origin

3.2 MPC Tracking Controllers

To allow the system to track a reference r, the optimization solver first minimizes a steady state input u_s to reach the steady state corresponding to the reference and then the overall system minimizes the difference between x and x_s (the steady state reference). The extension of the controller hence first involves the computation of a steady state target by minimizing:

$$objective = {u_s}^2$$

given that the x_s and u_s follow these equations:

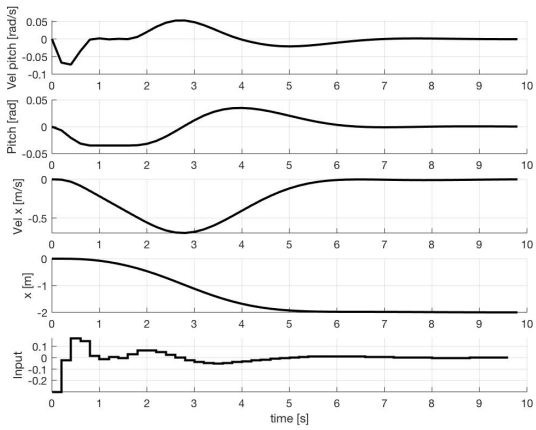
$$\mathbf{x_s}^+ = A\mathbf{x_s} + B\mathbf{u_s}$$

$$\mathbf{r} = C\mathbf{x_s} + D$$

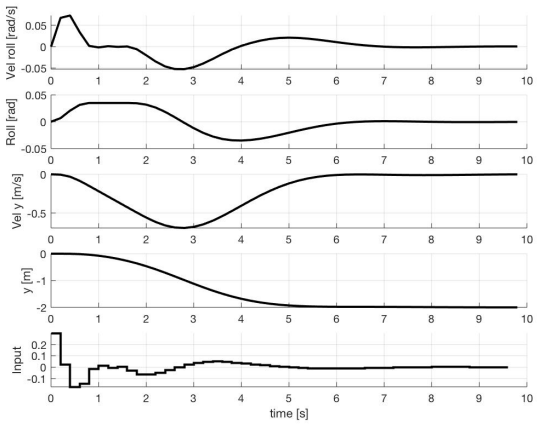
Then, the same procedure as before is done to compute the optimal input to apply to the system but this time the objective function to minimize becomes:

$$\begin{aligned} &\min_{\mathbf{u}} \ \textstyle\sum_{i=0}^{N-1} (x_i - x_s)^T Q (x_i - x_s) + (u_i - u_s)^T R (u_i - u_s) \\ &\quad \text{s.t.} \quad x_i - x_s \in \mathbb{X} \quad i \in \{1, \ldots, N-1\} \\ &\quad \text{s.t.} \quad u_i - u_s \in \mathbb{U} \quad i \in \{0, \ldots, N-1\} \\ &\quad x_{i+1} = Ax_i + Bu_i \\ &\quad x_0 = x \end{aligned}$$

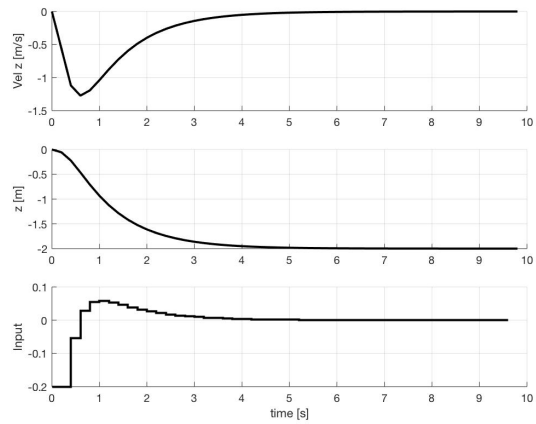
In fact, by dropping the requirement of invariance because there are no constraints on the position [x,y,z] and on the angle yaw which are the dimensions involved in tracking, we do not use a terminal set. We keep the same values for Q, R and N as previously. Figure 4 shows that each controller is able to successfully stabilize to a reference starting from origin.



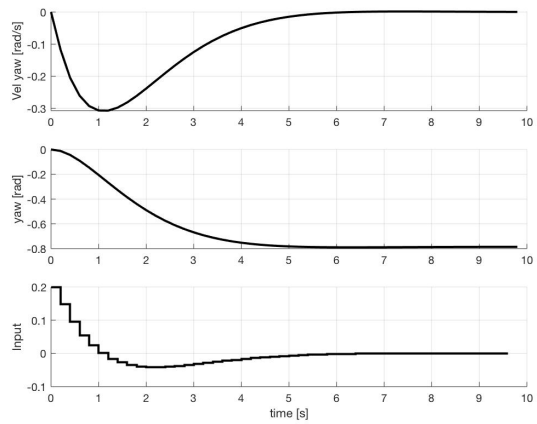
(a) x controller



(b) y controller



(c) z controller



(d) yaw controller

Figure 4: Plots of each controller tracking a reference

4 Simulation with Nonlinear Quadcopter

Merging all 4 controllers from the previous section, we can observe the behavior of the quadcopter by simulating the system tracking a given path of successive references.

4.1 Tracking a given path

Figure 5 shows the quadcopter tracking references over a given path with good precision. Figure 6 shows the angles, the linear velocities, the thrust of each motors and the position given by [x,y,z]. Interestingly, we first notice that the quadcopter does not yaw at all to follow this track. We also notice that the controllers are always able to reach the target but tracking the x and y position involves some latency. The latency depends on the controller outputs that are bounded. The tuning of R (control matrix) can be done in order to penalize less high values u_i, involving stiffer actuation. However, there is a trade-off since the performance or cost of transport will be higher, thus the robot consumes more energy.

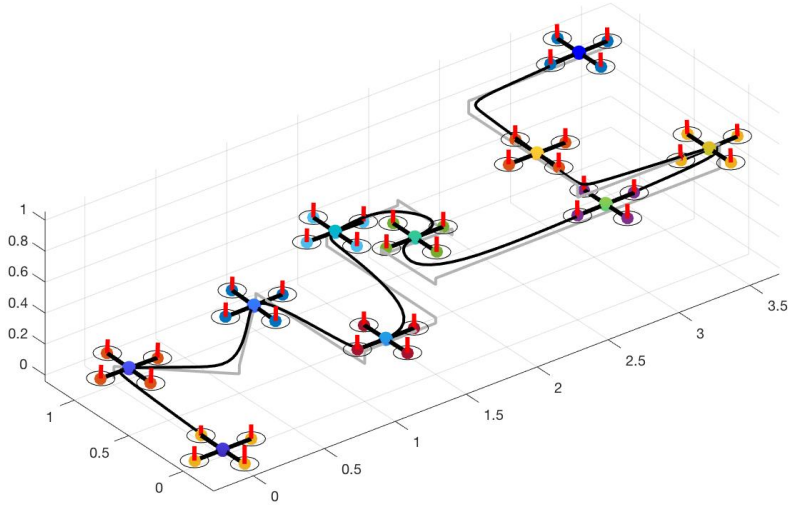


Figure 5: Simluation of the quadcopter following the path

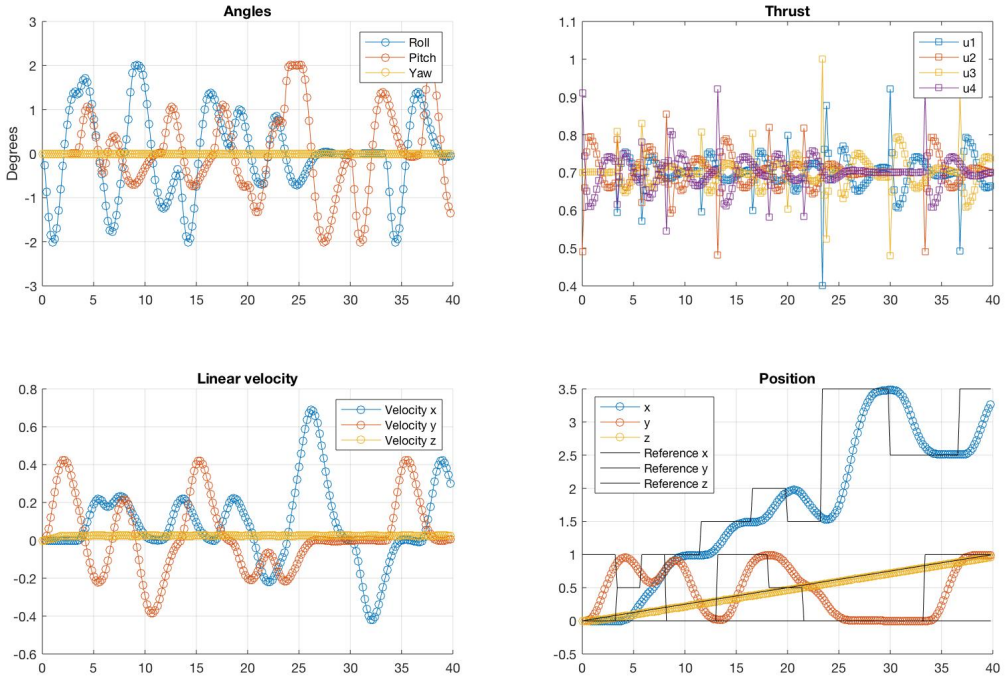


Figure 6: Plots of the angles, the thrust of each motors, the linear velocity and the position in the [x,y,z] dimensions

5 Offset-Free Tracking

For a quadcopter, a good controller must be able to work given an additional unknown mass added to it. The controller must be able to estimate this mass to compute the steady state values $[x_s, u_s]$. The dynamics of the system in the z-direction thus becomes:

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

and the steady state targets must now be found by again minimizing:

$$objective = {u_s}^2$$

given that the x_s and u_s follow these equations now:

$$\mathbf{x_s}^+ = A\mathbf{x_s} + B\mathbf{u_s} + B\mathbf{d}$$

$$ref = C\mathbf{x_s} + D + \mathbf{d}$$

but the disturance d is unknown at first and needs to be estimated using the following augmented matrices:

$$\overline{B} = \begin{bmatrix} B \\ 0 \end{bmatrix} \quad \overline{A} = \begin{bmatrix} A & B \\ eye(2) & 1 \end{bmatrix} \quad \overline{C} = \begin{bmatrix} C & ones(2,1) \end{bmatrix}$$

L must be satisfying this equation:

$$\overline{\mathbf{x}^+} = \overline{A} * \overline{\mathbf{x}} + \overline{B}\mathbf{u} + L * (\overline{C} * \overline{\mathbf{x}} - ref)$$

and the observer for this discrete-time system must be asymptotically stable meaning that the matrix A-LC has all the eigenvalues inside the unit circle. An adequate solution for L can be found using the following matlab function:

$$L = -place(\overline{A}', \overline{C}', [\lambda_1, \lambda_2, \lambda_3])'$$

The estimation of **d** is then inferred from $\overline{\mathbf{x}}$ at each step and converges to the true value of the disturbance. We define λ_1,λ_2,λ_3 with small values, respectively 0.1, 0.2 and 0.3, to reduce the initial overshoot (see figure 8). For comparison, Figure 9 shows that choosing 0.5, 0.6 and 0.7 would result in a larger overshoots in velocity and in position. The rest of the procedure and the other parameters R, Q and N are the same as in the previous sections.

5.1 Tracking a given path and rejecting the constant disturbance

Figure 7 shows the quadcopter tracking a reference and rejecting a disturbance over a given path. Figure 8 shows the angles, the linear velocities, the thrust of each motors and the position given by [x,y,z]. The position plot of figure 8 let us confirm that the model is able to quickly learn the amount of disturbance to reject it.

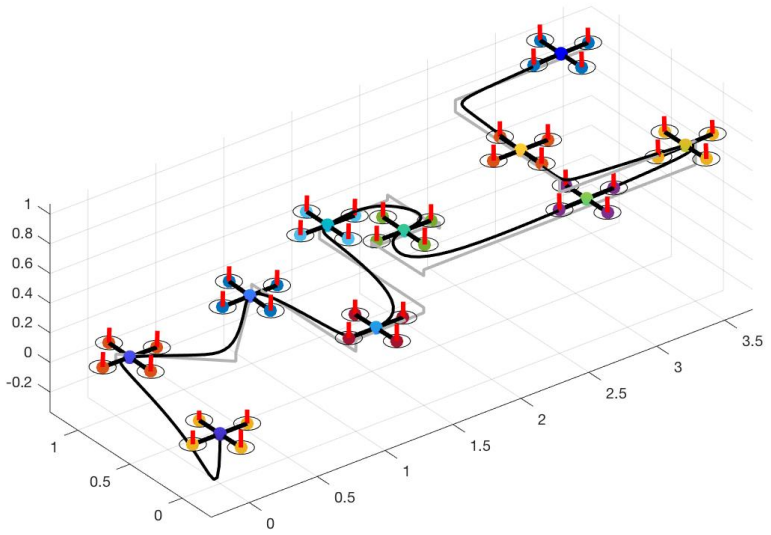


Figure 7: The quadcopter following the path in offset-free tracking

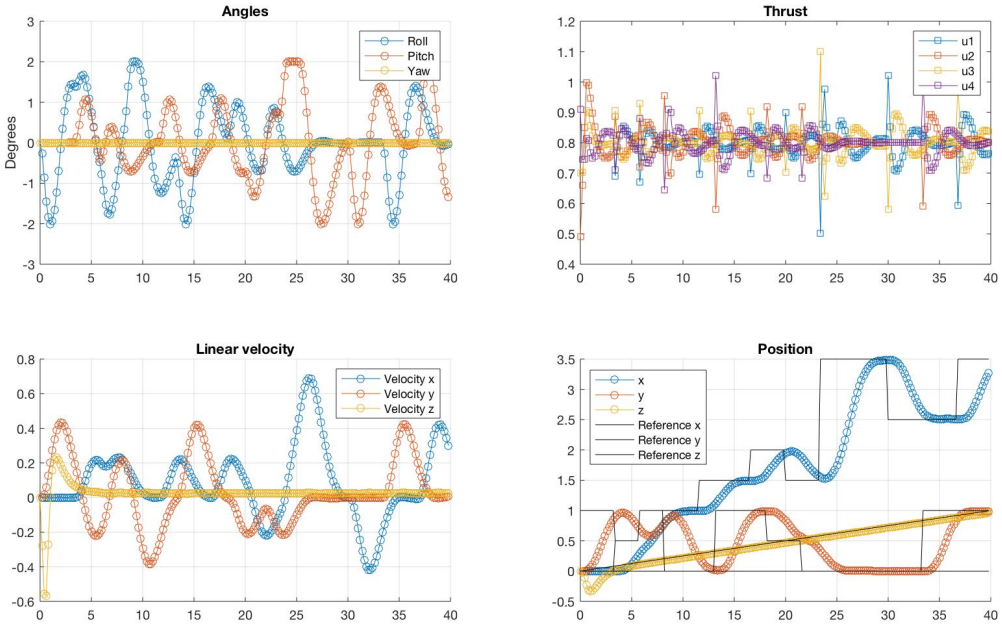


Figure 8: Plots of the angles, the thrust of each motors, the linear velocitiy and the position [x,y,z] in offset free tracking with λ_1,λ_2,λ_3 = 0.1, 0.2, 0.3

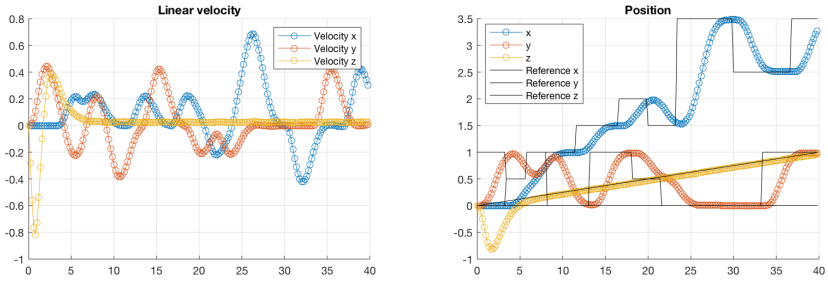


Figure 9: Effects of using larger eigenvalues λ_1,λ_2,λ_3 = 0.5, 0.6, 0.7; Larger overshoots in velocity and position can be observed in the z dimension.

6 Nonlinear MPC

In this section, model predictive control is performed on the whole system using the Matlab non-linear computing library CASADI. The non-linear system involves 12 states with 4 inputs. We use again a horizon N = 40 although we investigated that the minimum horizon required for feasibility is N = 5. The function is non-linear and given by:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

We discretize it using the fourth order runge kutta method so that:

$$\mathbf{x}^{+} = \mathbf{x} + (T_s/6) * (k_1 + 2k_2 + 2k_3 + k_4)$$

with

$$k1 = f(x,u)$$

$$k2 = f(x + T_s/2 * k1, u)$$

$$k3 = f(x + T_s/2 * k2, u)$$

$$k4 = f(x + T_s * k3, u)$$

The weights for velocity, position and the input are set to,

$$\begin{aligned} weight_{vel} &= 5 \\ weight_{pos} &= 10 \\ weight_{input} &= 2 \end{aligned}$$

Thereby, the input energy is less penalized than the state in the objective function and allows to reach faster the path references. In return, the transitions between path segments can lead to overshoots.

6.1 Tracking a given path with NMPC

We can drop the requirements of the other sections on the angles which were due to the linearization. The constraints are thus exclusively on the inputs. Moreover, in this case, the inputs are directly the thrusts, therefore they can reach their maximal value unlike previously where the constraints on **v** were quite restrictive. As a result, the NPMC controller performs better than the merged linear controllers as we can notice in figure 10 which shows the angles, the linear velocities, the thrust of each motors and the position given by [x,y,z]. Figure 11 and 12 show how accurately the quadcopter tracks the given path with the NMPC controller. However, the price to pay using NMPC is computation cost. Some measures would have to be taken to make it work in real time: either reducing N or using explicit MPC.

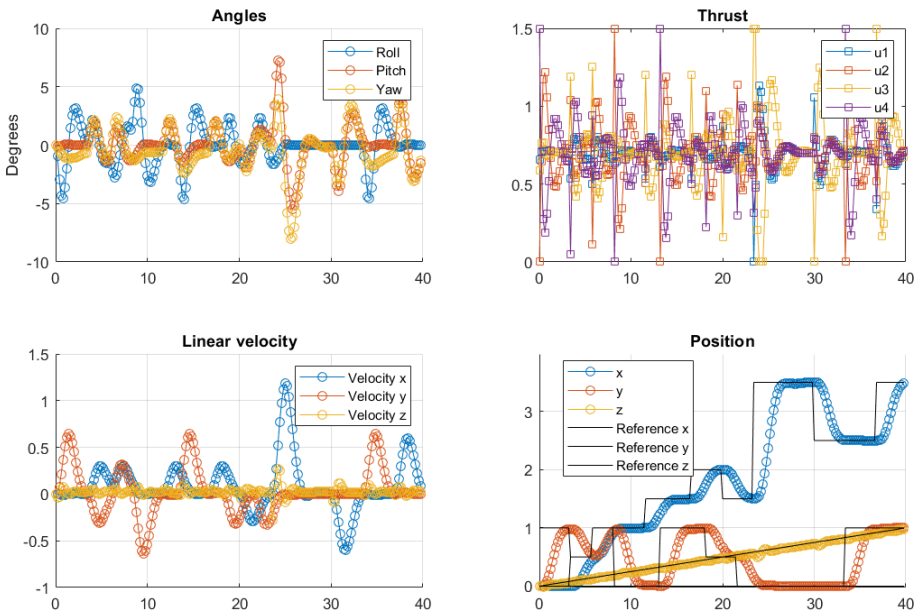


Figure 10: Plots of the angles, the thrust, the linear velocity and the [x,y,z] position in NMPC

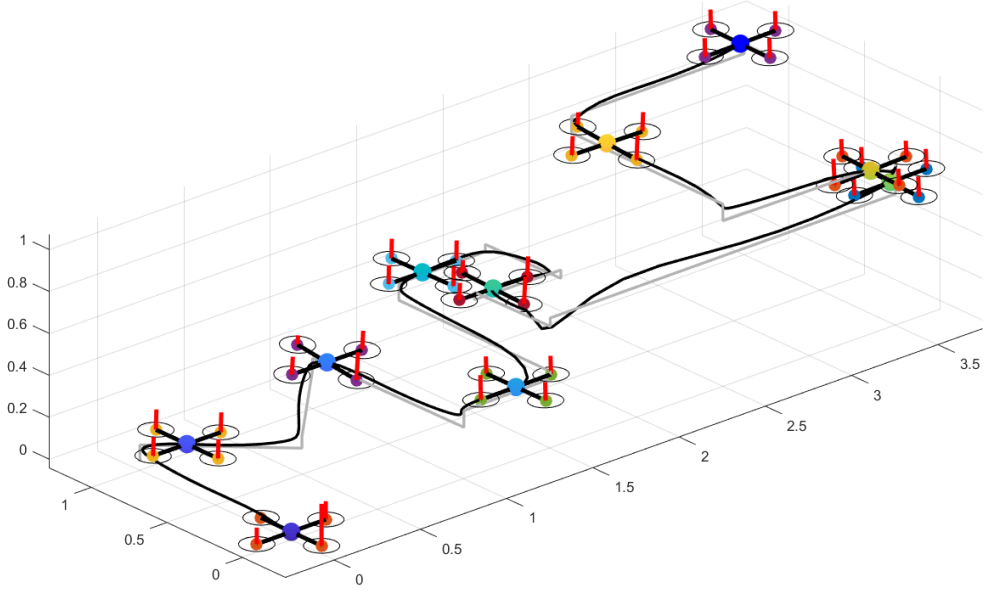


Figure 11: Simulation of the predicted trajectory with the NMPC controller

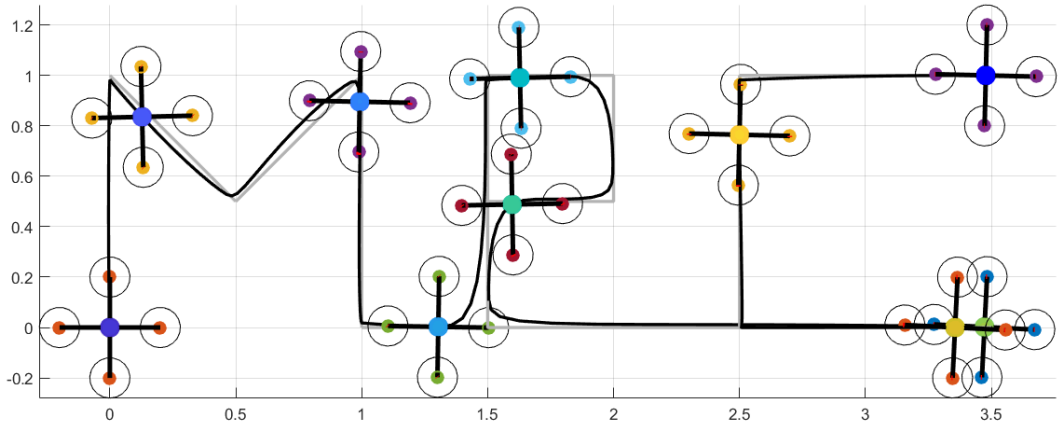


Figure 12: Top view of the simulated path in Figure 11