École Polytechnique Fédérale de Lausanne

ME-425: Model Predicitve Control

Designing an MPC for a quadcopter

GROUP 50

Authors:	*SCIPER:*
Dan Cisier	247896
Nicolas Congouleris	250711
Elie Graham	245927

Professor:
Colin Jones

January 11, 2020

Contents

1 Introduction

The aim of this project is to develop an MPC controller to fly a quadcopter. Since its dynamics are complex, the model is first simplified by trimming and linearizing it, thus creating four independent systems where one input controls one output. The validity of the approximate model is ensured by constraining the roll and pitch angles. Taking this and the limited thrust of the motors into account, a recursively feasible and stabilizing MPC controller is designed for each of these systems. The controllers are then extended to be able to track constant references, as well as perform offset-free tracking along the vertical axis. Finally, a non-linear MPC is designed for the non-linearized system.

2 Linearization and Diagonalization

2.1 Deliverable 2.1

The linearized system is defined as:

$$\boldsymbol{x} = \begin{bmatrix} \dot{\alpha} & \dot{\beta} & \dot{\gamma} & \alpha & \beta & \gamma & \dot{x} & \dot{y} & \dot{z} & x & y & z \end{bmatrix}^T = \begin{bmatrix} \dot{\boldsymbol{\theta}} & \boldsymbol{\theta} & \dot{\boldsymbol{p}} & \boldsymbol{p} \end{bmatrix} \tag{1}$$

with :

$$\boldsymbol{\theta} = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix} \quad \text{and} \quad \boldsymbol{p} = \begin{bmatrix} x & y & z \end{bmatrix} \tag{2}$$

$$\begin{aligned} \dot{\boldsymbol{x}} &= A\boldsymbol{x} + B\boldsymbol{u} \\ \boldsymbol{y} &= C\boldsymbol{x} + D\boldsymbol{u} \end{aligned} \tag{3}$$

and is obtained with the following MATLAB code:

```
quad = Quad();
[xs,us] = quad.trim();
sys = quad.linearize(xs, us);
```

The resulting sys structure contains matrices (4) to (7)

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9.81 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -9.81 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

$$B = \begin{bmatrix} 0 & 0.56 & 0 & -0.56 \\ -0.56 & 0 & 0.56 & 0 \\ 0.733 & -0.733 & 0.733 & -0.733 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3.5 & 3.5 & 3.5 & 3.5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{7}$$

We then take the linearized system and apply the change of variable (8), which allows us to express $\dot{\boldsymbol{x}}$ as a function of $\boldsymbol{x}$ and of the moments M_α, M_β, M_γ, and net force F produced by the motors.

$$\boldsymbol{v} = \begin{bmatrix} F \\ M_\alpha \\ M_\beta \\ M_\gamma \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & k_F L \\ -k_F L & 0 & -k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \boldsymbol{u} = T\boldsymbol{u} \tag{8}$$

$$\begin{aligned} \dot{\boldsymbol{x}} &= A\boldsymbol{x} + B\boldsymbol{u} = A\boldsymbol{x} + BT^{-1}\boldsymbol{v} \\ \boldsymbol{y} &= C\boldsymbol{x} + D\boldsymbol{u} = C\boldsymbol{x} + DT^{-1}\boldsymbol{v} \end{aligned}$$

This changes matrix B to:

$$\bar{B} = BT^{-1} \begin{bmatrix} 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.0667 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.125 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{9}$$

By looking at this $\overline{B}$ matrix, we can clearly see that we obtain 4 independent systems, each input controlling one, and only one output: M_α, M_β, M_γ and F control respectively $\dot{\alpha}$, $\dot{\beta}$, $\dot{\gamma}$ and $\dot{z}$. This transformation would work for any steady-state target because the linearization of B will always give the same result and so, $\overline{B}$ will always give these four independent systems for all steady-state target.

3 Design MPC Controllers for Each Sub-System

3.1 Deliverable 3.1

We designed our terminal invariant set using YALMIP, implementing the algorithm in the "Introduction to Constrained Systems" lecture, slide 29. First an optimal LQR controller K and optimal weight Qf are computed for the unconstrained system and K is used to define the closed loop system. The resulting set is intersected with its pre-set in a loop until it stabilises, giving us the maximal invariant set Xf. Then the MPC controller is computed for the horizon N, by making sure the state and input constraints are always satisfied until step N-1. Stability and feasibility are ensured by adding the terminal controller and cost at step N. For example, the x controller regulates $\dot{\beta}$, β, $\dot{x}$ and x, of which only β is constrained: $|\beta| \leq 0.035$ rad. We show the corresponding `MATLAB` code:

```
M = [1; -1]; m = [0.3; 0.3];
F = [0 1 0 0 ; 0 -1 0 0 ]; f = [0.035; 0.035];

R = 15;
Q = diag([10,10,10,10]);
[K, Qf, ~] = dlqr(mpc.A, mpc.B, Q, R);
K = -K;

Xf = polytope([F; M*K], [f; m]);
Acl = [mpc.A + mpc.B*K];
while 1
    prevXf = Xf;
    [T,t] = double(Xf);
    preXf = polytope(T*Acl,t);
    Xf = intersect(Xf, preXf);
    if isequal(prevXf, Xf)
        break
    end
end
[Ff,ff] = double(Xf);

```

```
22
23 %CONSTRAINTS AND OBJECTIVE
24 con = [];
25 obj = 0;
26
27 con = con + (x(:,2) == mpc.A*x(:,1) + mpc.B*u(:,1));
28 con = con + (M*u(:, 1) <= m);
29 obj = obj + x(:, 1)'*Q*x(:, 1) + u(:,1)'*R*u(:,1);
30
31 for i = 2:N-1
32   con = con + (x(:, i+1) == mpc.A*x(:, i) + mpc.B*u(:, i));
33   con = con + (F*x(:, i) <= f) + (M*u(:, i) <= m);
34   obj = obj + x(:, i)'*Q*x(:, i) + u(:, i)'*R*u(:, i);
35 end
36
37 con = con + (Ff*x(:,N) <= ff);
38 obj = obj + x(:,N)'*Qf*x(:,N);
```

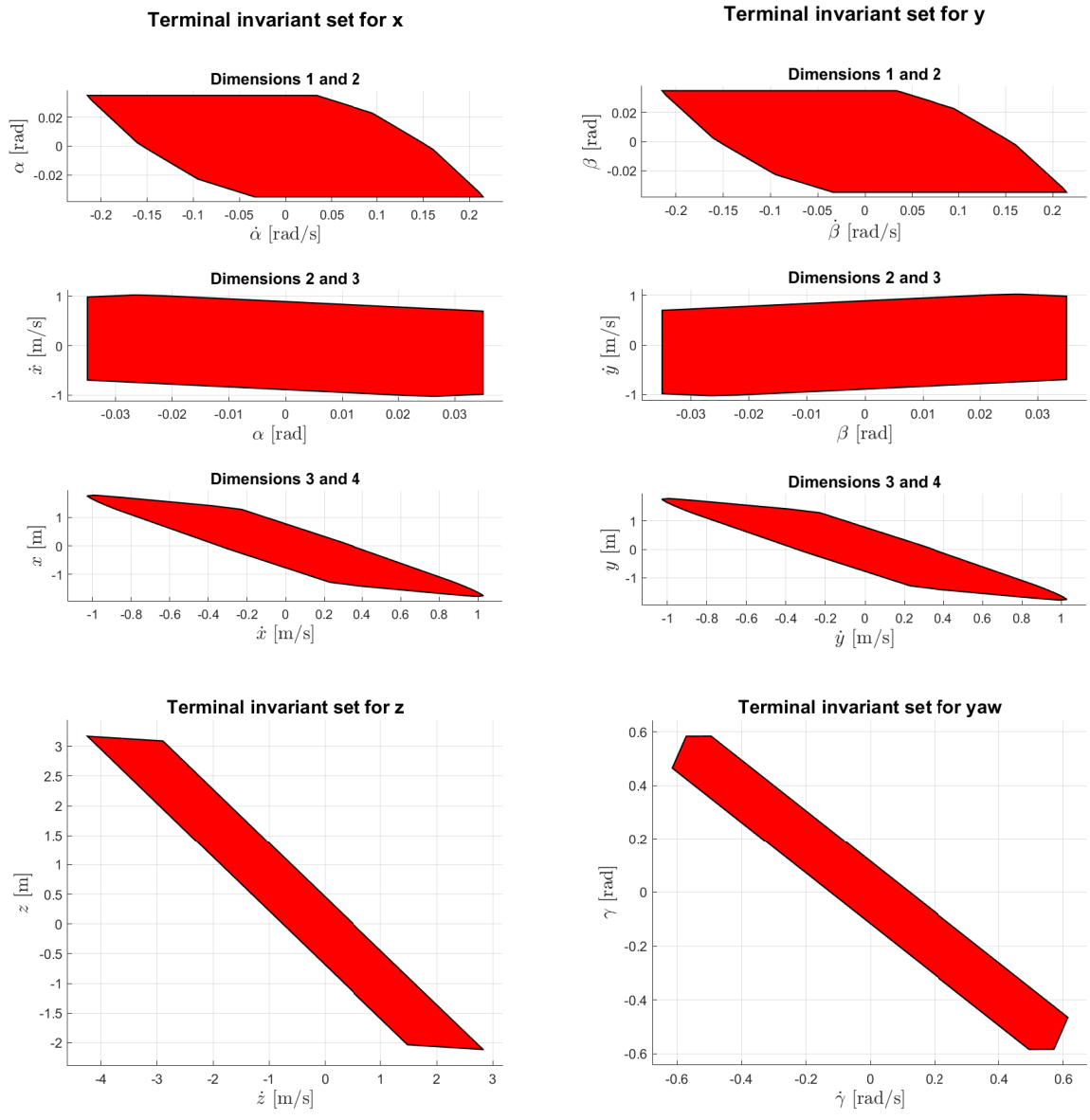


Figure 1: Terminal invariant sets for all four controllers

The resulting invariant sets shown in Figure 1 confirm that α and β never go beyond ± 0.035 rad. The remaining state variables, while unconstrained, are restricted to values ensuring the satisfaction of state and input constraints.

The first parameter we tuned was N, following this procedure:

- start with a low N value (in our case 10)
- increase N, until the feasible set can be calculated
- add a security margin to obtain the final N value

By using this procedure on the x controller, we obtained a N value of 20. For simplicity, we just use this value for all our controller as this value does not seem too big from a computation point of view.

Afterwards, we tuned Q and R, starting with $R = 1$ one and matrix $Q = I$, which we adjusted by looking at the limit of the terminal invariant set and the settling time. The x and y controllers, being identical, share the same values $R = 15$ and $Q = 10 \cdot I$, and the z and yaw controllers share $R = 1$ and $Q = diag([10;\ 20])$ due to their similarity.

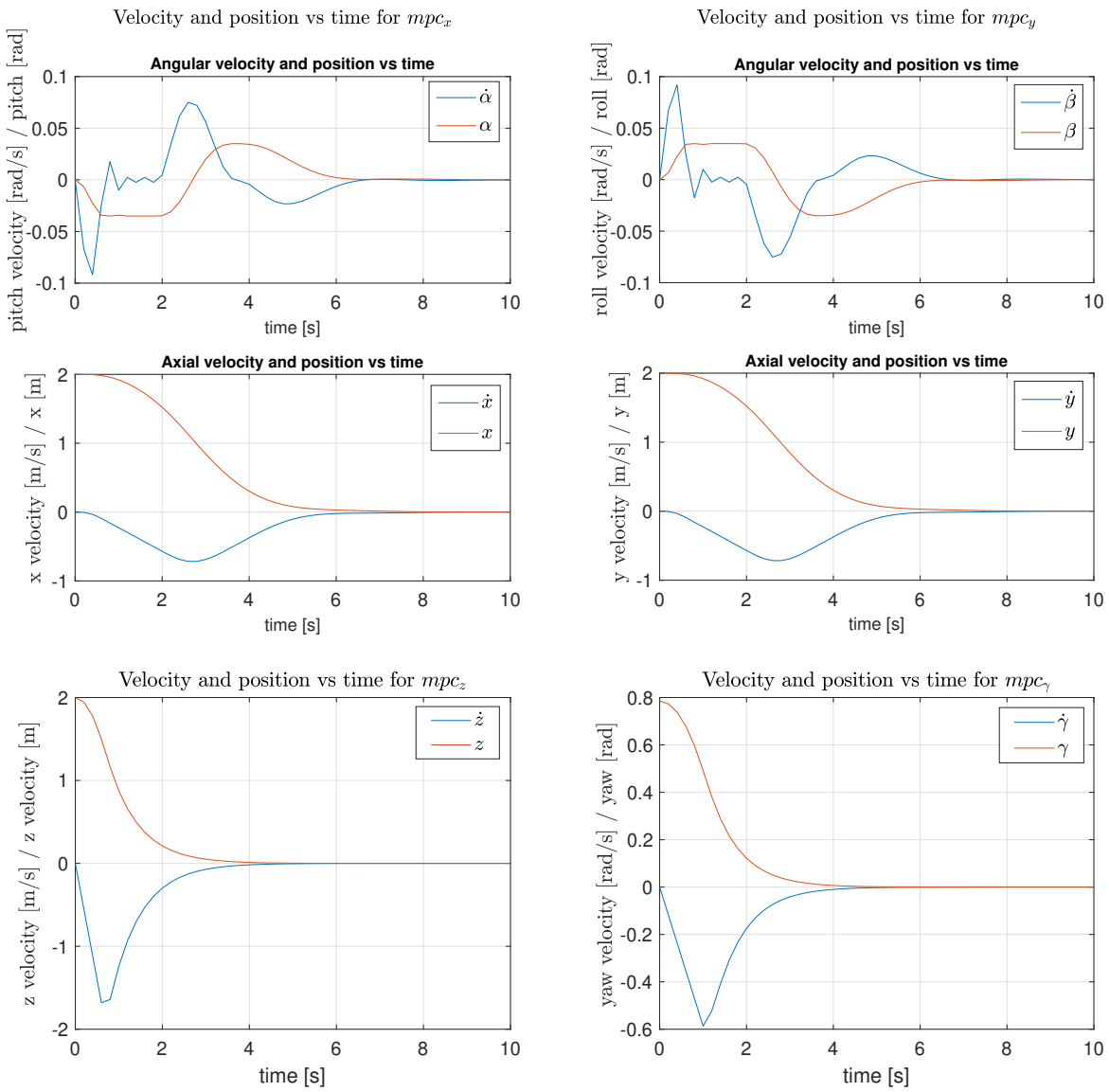


Figure 2: Plots showing the quadcopter coming back to the origin

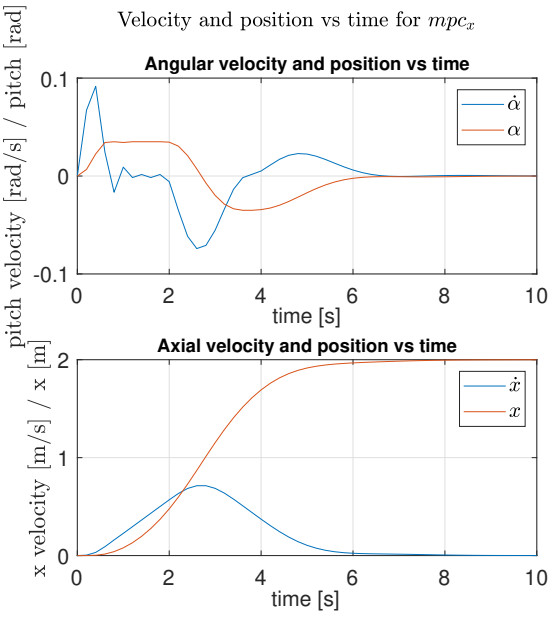
Figure 2 confirms that the controllers behave similarly in pair, and that they are able to bring the quadcopter back to the origin in under 8 s with no overshoot. Note that the x and y controllers also need to compensate for the slight variations of angle they generate. We also observe that the z and yaw controllers need less time to go back to the origin point than the x and y controllers. This is due to the former being simpler from a constraint point of view, as they only have constraints on one input and none over the outputs.

3.2 Deliverable 3.2

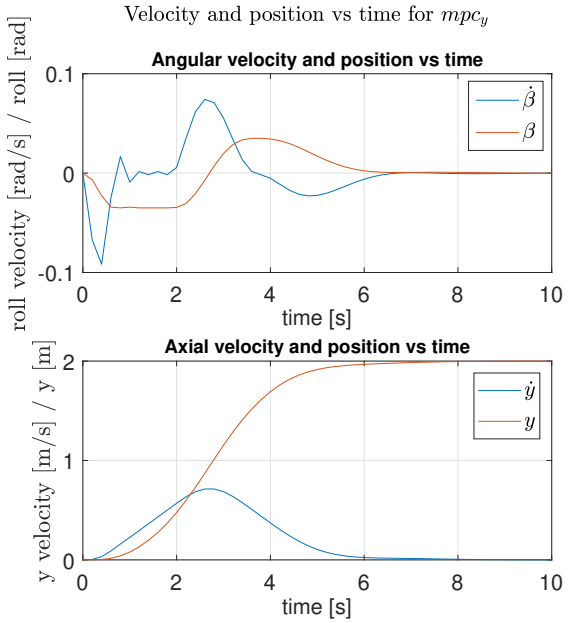
Terminal sets were dropped and the tracking was designed using the formulas given in the "Practical MPC" lecture, slides 42 and 43:

$$\begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ ref \end{bmatrix} \tag{10}$$

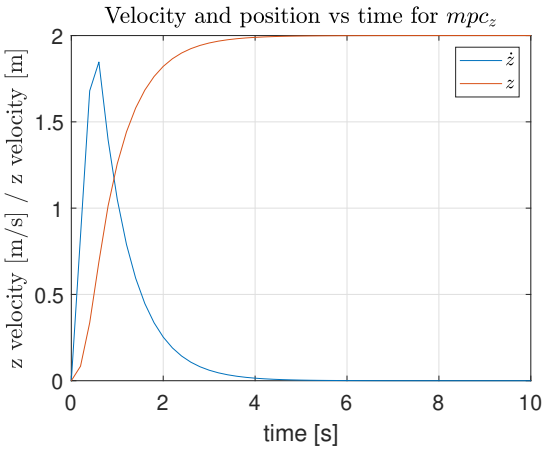
Figure 3 shows that the time needed to go to a reference point from the origin is the same as going from the origin to this reference. So, the quadcopter is also able to travel to a point 2 m away from the origin, or rotate by 45° in under 8 s.



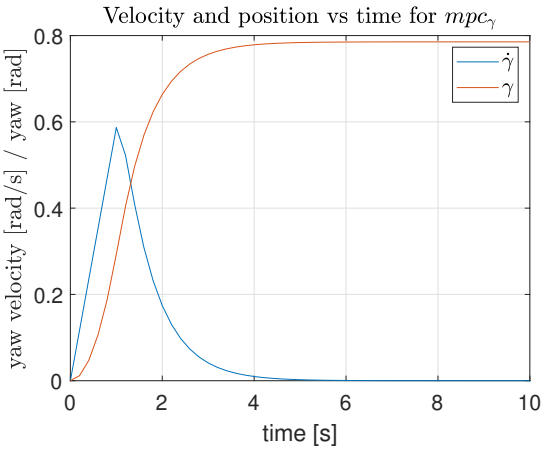
(a) x = 2 m



(b) y = 2 m



(c) z = 2 m



(d) $\gamma = \frac{\pi}{4}$ rad

Figure 3: Plots showing the quadcopter going from the origin to a desired position

4 Simulation with Nonlinear Quadcopter

4.1 Deliverable 4.1

Figure 4 shows the use of the controller in a 3D simulation. The reference track is shown in grey. We see that the drone is able to follow in general the track, only deviating for sharp changes of movement, thus smoothing corners.

The references followed by each controller are displayed on the "Position" plot, along with the corresponding displacement of the quadcopter. We see that ramp references are perfectly followed and step references are tracked without error, albeit with delay.

On the "Angles" graph, we can see the constraints in action : if we observe the pitch line in red, we see around 25 s that the maximum angle is limited to 2°. Interestingly, the yaw stays 0 througout the entire simulation, which makes sense since rotating along the z axis is not required to track the path.

The "Thrust" plot shows that the inputs never go beyond 1, despite the motors accepting thrusts up to 1.5. The tight constraints on roll and pitch make it difficult to fully exploit the capabilities of the motors.

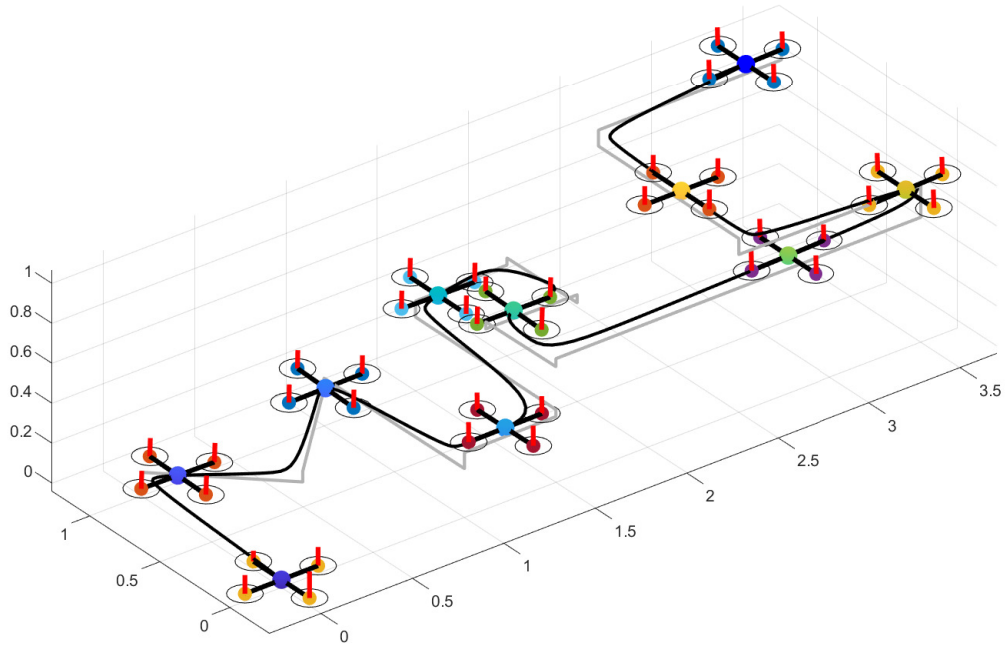


Figure 4: Plot of the controllers tracking the path

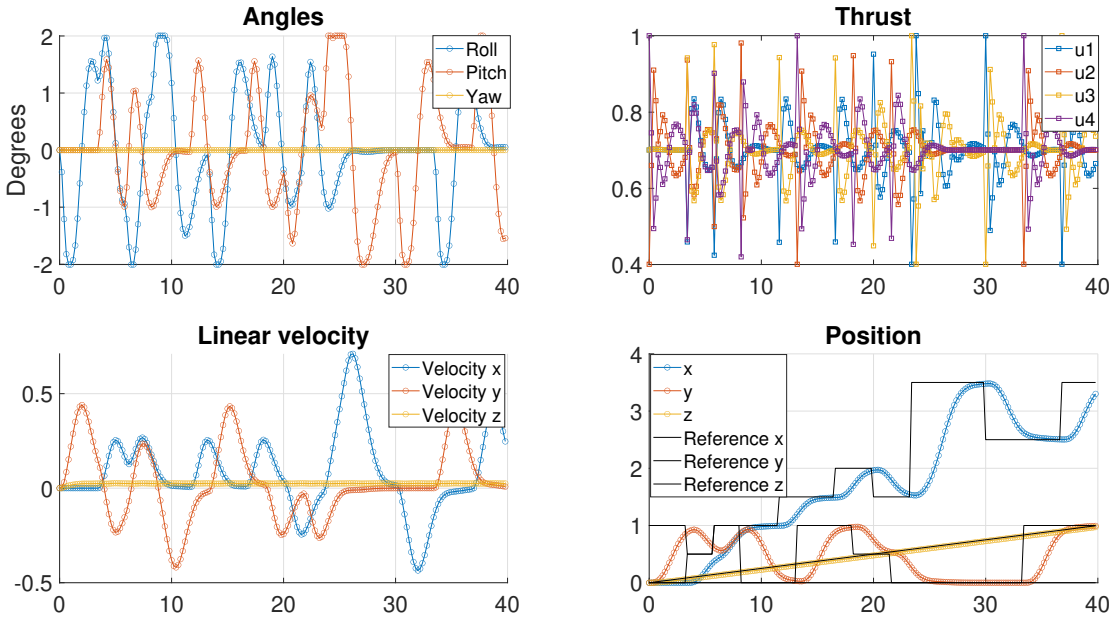


Figure 5: Plots of angles, thrust, linear velocity and position through time

5 Offset-Free Tracking

5.1 Deliverable 5.1

For the observer design of the z controller, we used the formulas given in the "Practical MPC" class, slide 53:

$$\begin{bmatrix} \hat{x}_{i+1} \\ \hat{d}_{i+1} \end{bmatrix} = \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} \hat{x}_i \\ \hat{d}_i \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k + \begin{bmatrix} L_x \\ L_d \end{bmatrix} (C\hat{x}_i + C_d\hat{d}_i - y_i) \tag{11}$$

From this formulas we obtain $\overline{A}, \overline{B}$ and $\overline{C}$.

$$\overline{A} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix}, \quad \overline{B} = \begin{bmatrix} B \\ 0 \end{bmatrix} \quad and \quad \overline{C} = \begin{bmatrix} C \\ C_d \end{bmatrix} = \begin{bmatrix} C \\ 1 \end{bmatrix} \tag{12}$$

For the estimator gain L, we used the pole placement method, starting with slow poles (close to the unity circle) and progressively reduced them. Our goal was to reduce the drop at the begining and to fully compensate a 0.1 offset bias in under 5 s. Poles were always picked inside the unity circle, in order to guarantee stability.

Figure 6 shows a 3D simulation with a -0.1 bias on the z axis and $L = [0.1, 0.2, 0.3]$. The quadcopter drops by 0.2 m at the origin before going back to following the same path as in Figure 4, as the observer fully compensates the bias.

The effect of the input bias is displayed in Figure 7, where the initial drop is clearly visible on the "Linear velocity" and "Position" graphs. The maximal thrust of the motors is slightly greater than in Figure 5, in order to compensate the bias.

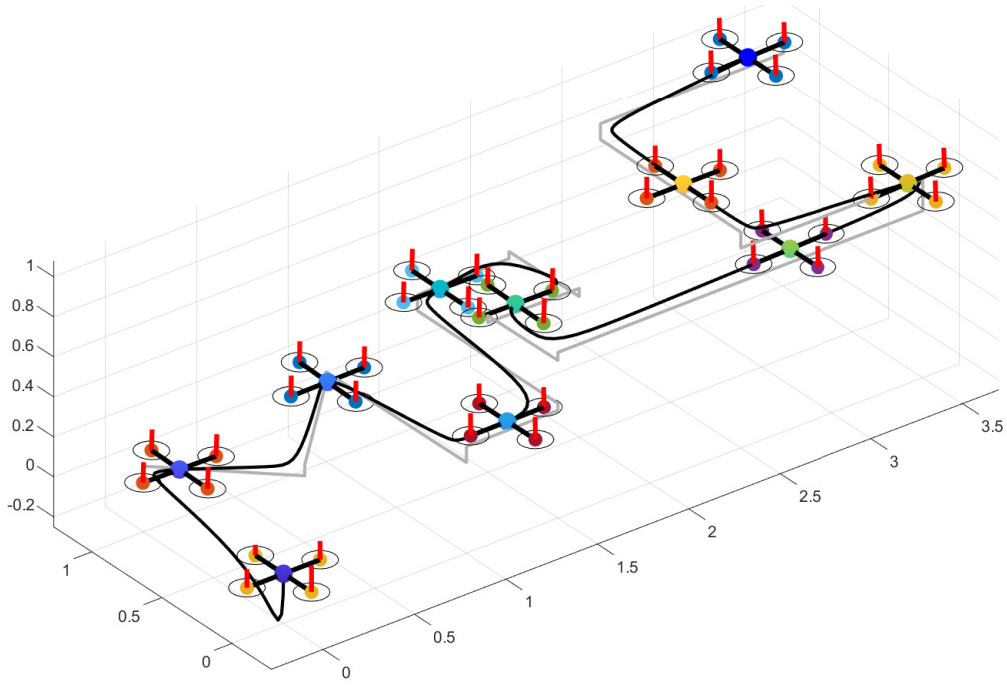


Figure 6: Plot of the controllers tracking the path without offset

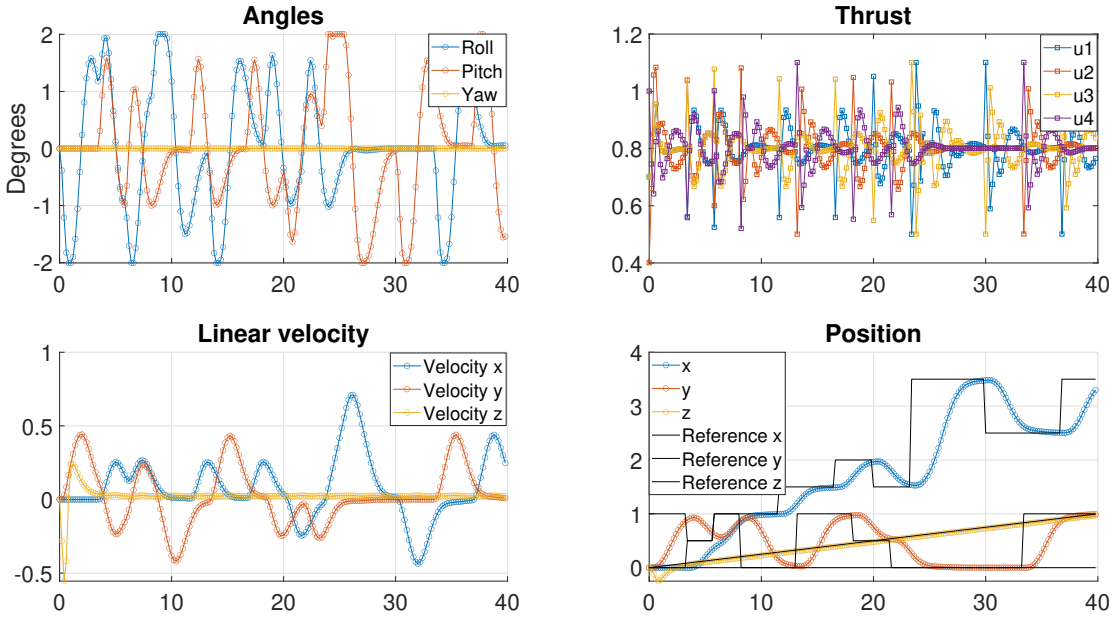
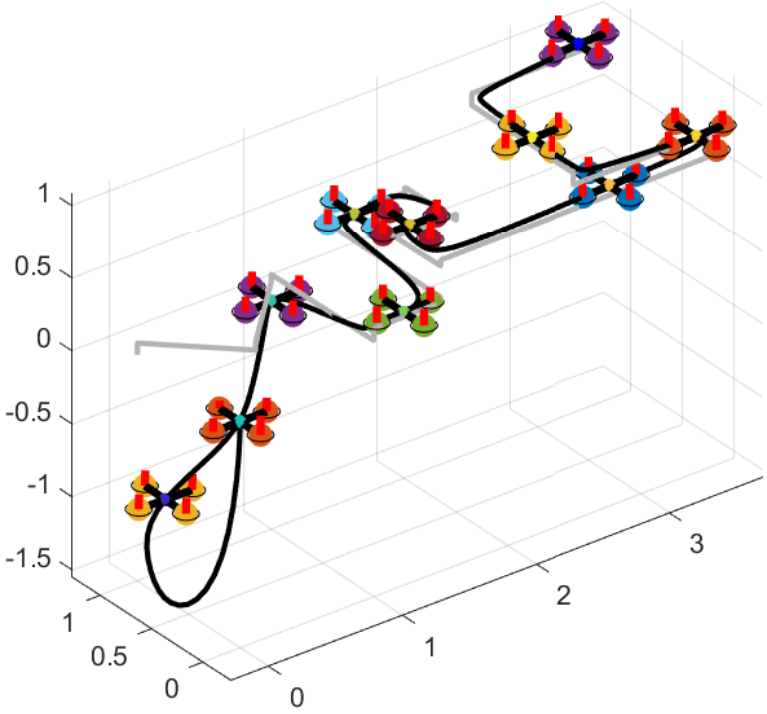


Figure 7: Plots of angles, thrust, linear velocity and position through time

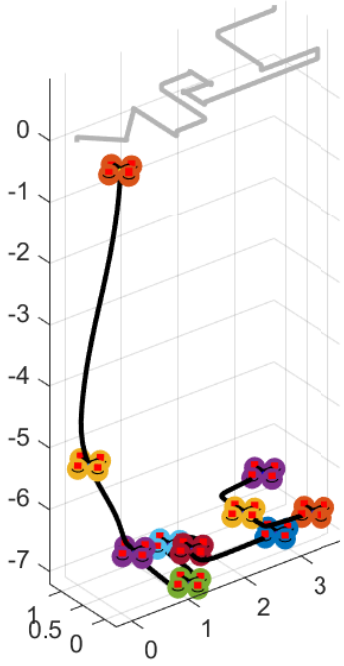
Figure 8 shows the effect of the pole placement on the z controller. We observe the following:

- small poles (also called fast poles) allow faster compensation of the bias.
- if one of the poles is on the unity circle, the controller is unable to compensate the bias on the reference
- if one of the poles is out of the unity circle, the controller is unstable

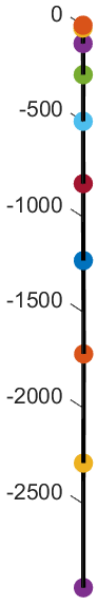
Our controller is able to reach the last reference point for any bias in the [-2.9, 1.9] range. Good tracking, arbitrarily defined as a maximum drop of 0.5 m and full compensation before reaching the first reference point, is attained for biases in the [-2, 1] range.



(a) poles of L : [0.7,0.8,0.9]



(b) poles of L : [0.7,0.8,1]



(c) poles of L : [0.7,0.8,1.1]

Figure 8: Plots showing the effect of L pole on the z controller

6 Nonlinear MPC

6.1 Deliverable 6.1

We used the same structure as in Exercise 6 to implement the non-linear MPC controller. The quadcopter dynamics are discretized using the RK4 function. The hard constraints of our optimisation problem are the thrust, which is limited to 1.5, and the initial state X0. We aim to minimize the difference between the quadcopter position and the reference (in x, y, z and yaw), while using as little thrust from the rotors as possible. We tuned the horizon in a similar fashion to section 3.1, starting with N = 10 and increasing it until the path was satisfactorily tracked with N = 20. We also limited the x-displacement to 3.5 and y-displacement to 1, in order to supress some overshoots. We show the corresponding MATLAB code:

```
N = 20; % MPC horizon [SET THIS VARIABLE]
h=0.2;
%decision variables
X = opti.variable(12,N+1); % state trajectory variables
U = opti.variable(4, N); % control trajectory (thrusts)
X0 = opti.parameter(12,1); % initial state
REF = opti.parameter(4,1); % reference position [x,y,z,yaw]

f = @(x,u) quad.f(x, u);
f_discrete = @(x,u) RK4(x,u,h,f);

param=[0.7 20 0.5 0.3];

%---- objective ---------
opti.minimize(...
  param(1)*(X(10,:)-REF(1))*(X(10,:)-REF(1))' + ... % tracking ref X
  param(1)*(X(11,:)-REF(2))*(X(11,:)-REF(2))' + ... % tracking ref Y
  param(2)*(X(12,:)-REF(3))*(X(12,:)-REF(3))' + ... % tracking ref Z
  param(3)*(X(6,:) - REF(4))*(X(6,:)-REF(4))' + ... % tracking ref Yaw
  param(4)*U(1,:)*U(1,:)' + ... % Minimize input
  param(4)*U(2,:)*U(2,:)' + ... % Minimize input
  param(4)*U(3,:)*U(3,:)' + ... % Minimize input
  param(4)*U(4,:)*U(4,:)');     % Minimize input

for k=1:N % loop over control intervals
  opti.subject_to(X(:,k+1) == f_discrete(X(:,k), U(:,k)));
end

opti.subject_to(0 <= U <= 1.5);
opti.subject_to(X(:,1)==X0);
opti.subject_to(X(10,:)<=3.5);
opti.subject_to(X(11,:)<=1);
opti.subject_to(X(12,:)<=REF(3));
```

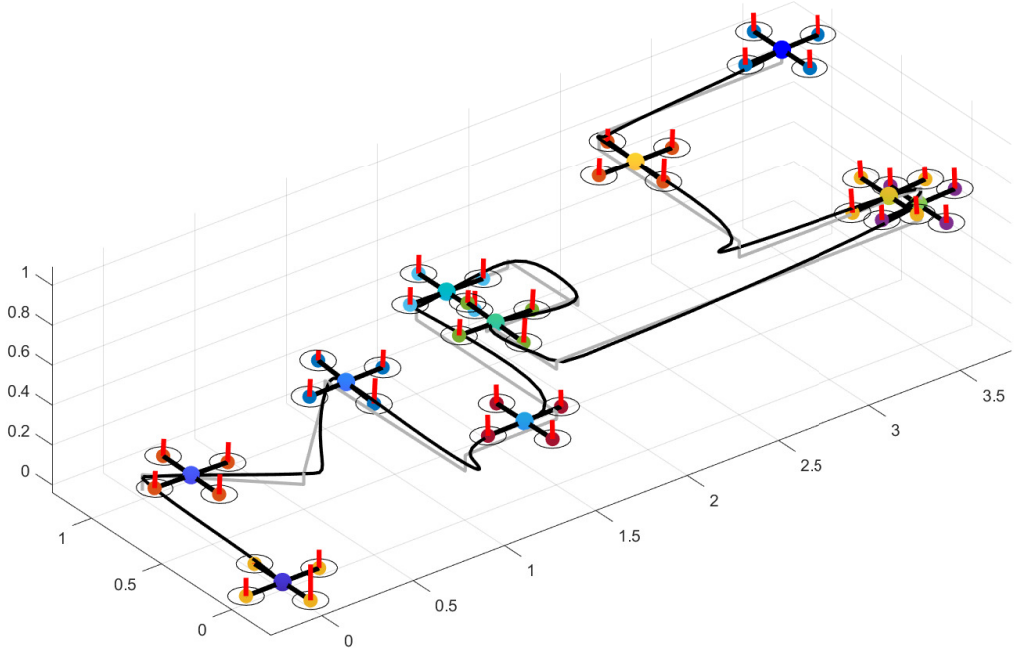


Figure 9: Plot of the NMPC controller tracking the path

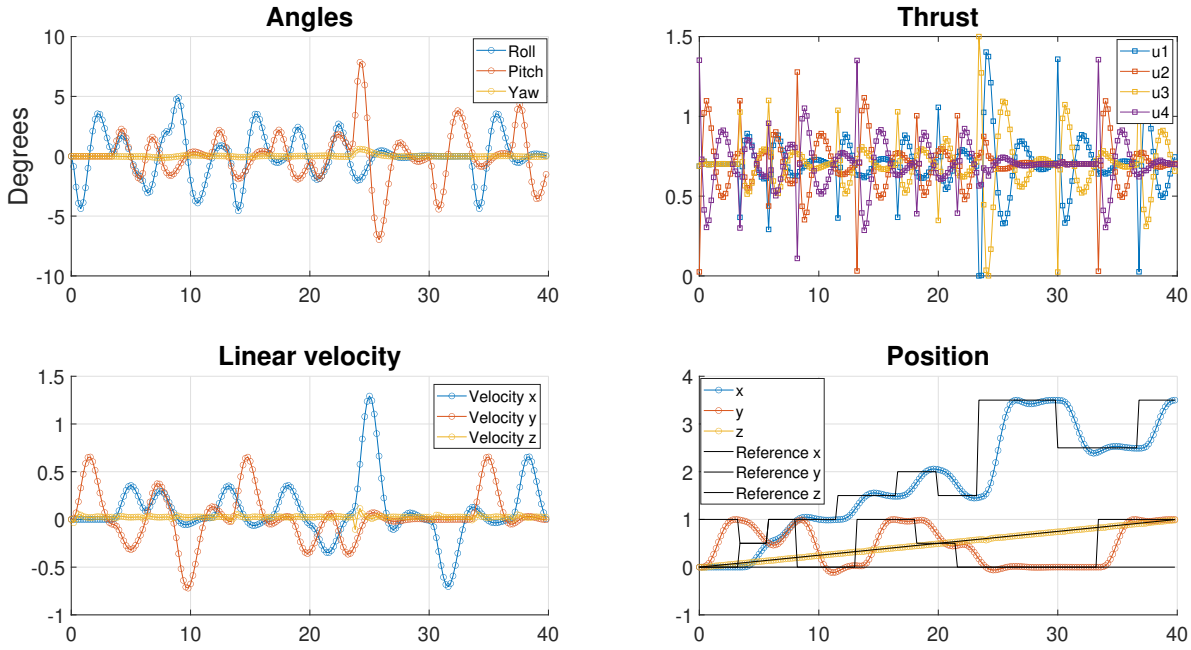


Figure 10: Plots of angles, thrust, linear velocity and position through time

We observe on Figures 9 and 10 that this non-linear regulator can track the reference faster than the split MPC on the linearized system. Indeed, since the system was not linearized, there are no constraints on the roll and pitch angles, allowing to fully exploit the capabilities of the quadcopter. As a result, it can take sharp turns but this produces overshoots that are not easy to temper.

7 Conclusion

The quadcopter successfully tracks a given path, while compensating biases along the vertical axis, for any state-state target. The MPC controllers guarrantee the validity of the linearized model by constraining the roll and pitch angles, while giving admissible thrust commands. We hand-tuned the horizon N, state cost Q and input cost R so that the drone can move by 2 m and rotate by 45° in under 8 s. Since the controllers are similar in pairs (x and y, z and yaw), we only tuned one set of values for each pair. We then extended the controllers to track constant references, using the delta-forumulation, and to compensate constant biases by augmenting the system with an observer. We hand-tuned the estimator gain L with the pole placement method, and decided the controller performed well enough when the quadcopter could reject a -0.1 bias in under 5 s with an initial drop under 0.5 m. As a result, small constant disturbances are rejected, and ramp and step references are tracked without errors. Besides, we observe that the motors are never fully exploited, as the roll and pitch constraints are very tight, making it difficult not to violate them with higher thrusts. Finally, the non-linear MPC regulator shows that the absence of linearisation constraints on the roll and pitch angles allows us to use the full potential of the quadcopter and thus to track the path more agressively, at the cost of some overshoots. Overall the project went well and we were able to obtain a working controller for the task at hand.