Final Report

ME-425 Model Predictive Control

Group 3:

Gaspard LEROY 287178

Jean LESUR 284531

Filip SLEZAK 286557

Date : January 29, 2022

Contents

1 Deliverables **1**

1.1 Deliverable 2.1 1

1.2 Deliverable 3.1 3

1.3 Deliverable 3.2 7

1.4 Deliverable 4.1 9

1.5 Deliverable 5.1 11

1.6 Deliverable 6.1 14

1 Deliverables

1.1 Deliverable 2.1

- Explain why this occurs, and whether this would occur at other steady-state conditions. Hint : think about what properties x_s and u_s must have for $\dot{x_s} = f(x_s, u_s) = 0$

  The steady state equation is $(I - A)x - Bu = 0$ except here B is not full rank and this equation has multiple solutions. All those solutions are linear combinations of the speeds in roll, pitch, yaw and z. Through the transformation matrix T, one can affect the other 8 parameters of the state if given sufficient information over time. Therefore it makes sense to only design 4 controllers.

  The linearized model gives us the following B matrix.

$$B_{linearized} = \begin{pmatrix} 0 & 0.56 & 0 & -0.56 \\ -0.56 & 0 & 0.56 & 0 \\ 0.7333 & -0.7333 & 0.7333 & -0.7333 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3.5 & 3.5 & 3.5 & 3.5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

  We can clearly see that some inputs are coupled through this matrix since some lines have more than one non-zero coefficient.

The B matrix of the transformed system is given by :

$$B_{transformed} = \begin{pmatrix} 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.06667 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.125 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

With the transformation, the inputs become decoupled, allowing us to build 4 independent controllers.

This transformation is performed with the help of the matrix T which is given below.

$$T = \begin{pmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{pmatrix}$$

The linearization is performed at a steady-state : that is the new matrices A,B,C and D are built using partial derivation of the functions f and h evaluated at points $\bar{x}$ and $\bar{u}$. Given other steady-states, this transformation would also work. Actually, there are many other steady-states : the $\bar{u}$ should still be the same while $\bar{x}$ may change. In other words, the 4 motors should compensate for the weight at all time but the settling position (x,y,z) can change.

1.2 Deliverable 3.1

- Explanation of design procedures that ensure recursive constraint satisfaction.

  To ensure recursive constraint satisfaction, we introduced terminal cost and constraints. The terminal constraint forces the state to be in the terminal invariant set. The terminal cost approximates the cost of the the tail from N to ∞.

- Explanation of choice of tuning parameters (Q, R, N, terminal components).

  First of all, we want the settling time to be around 8 seconds, this is 40 time steps with a sampling time of $0.2s$.

  Since the controllers for x and y are very similar and they have a similar expected behavior, we choose the same Q and R matrices for both controllers. Those matrices are given below.

$$Q_{x,y} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_{x,y} = 1$$

  The coefficient $Q_{2,2}$ is set to a high value in order to force the quadcopter to reduce pitch/roll angle to achieve a stable end pose. Recall that the state for the x controller is $\begin{bmatrix} \dot{\beta} & \beta & \dot{x} & x \end{bmatrix}$ and the state for the y controller is $\begin{bmatrix} \dot{\alpha} & \alpha & \dot{y} & y \end{bmatrix}$.

  For the z and yaw controllers, we choose the following matrices.

$$Q_{z,yaw} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$R_{z,yaw} = 1$$

  Note that we set the coefficients through trial and errors since it does not exist any method to get "good" coefficients directly. Good coefficients mean a settling time shorter than 8

seconds (40 time steps). Also, the R matrices are not set with high valued coefficients because we don't have rate constraints (and do not want to restrict the amount of u energy used). That's why we choose the default R = 1 for all 4 controllers.

To choose the horizon length, we tried different values until we found a nice behavior. Actually we want the horizon to be short enough, to avoid over computation and such that we can compute the MPC controller in real-time but not too short or we could loose stability or feasibility. In all the 4 controllers, we set the horizon to N = 10.

Finally, we simulated the infinite horizon with the terminal cost of a LQR controller, since it makes a good terminal controller. The cost is chosen as $x_N^T P x_N$ where P is the solution to the discrete time algebraic Riccati equation and x_N is the final state. The terminal constraints make sure that the state remains in the terminal set $\mathcal{X}_f$ once the state enters it.

- Plot of terminal invariant set for each of the dimensions, and explanation of how they were designed and tuning parameters used.

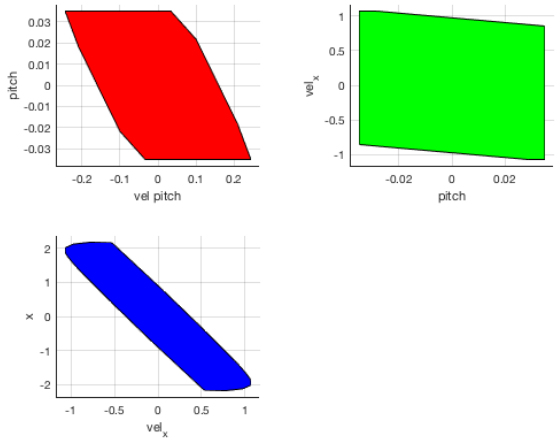
  The plots in Figure 1 present the terminal invariant set for each of the controllers. Note that when the dimension of the terminal set is higher than 2, we projected it.

- Plot for each dimension starting at two meters from the origin (for x, y and z) or stationary at 45 degrees for yaw.

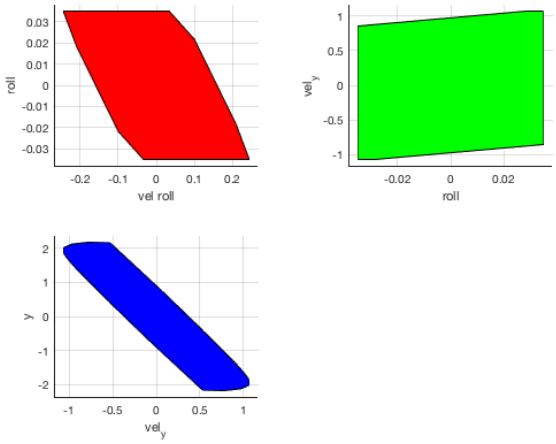
  You will find the plots in Figure 2. The dashed lines in the sub-figures (a) and (b) represent the constraints on pitch and roll respectively (between -0.035 and+0.035 rad).

- m-code for the four controllers, and code to produce the plots in the previous step.

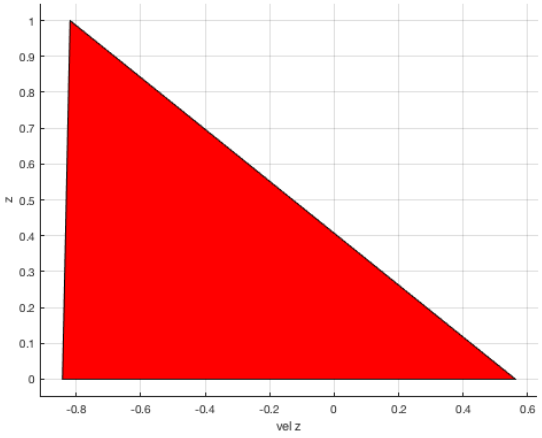
  You will find the m-code files in the zipped folder **Deliverable_3_1.zip**



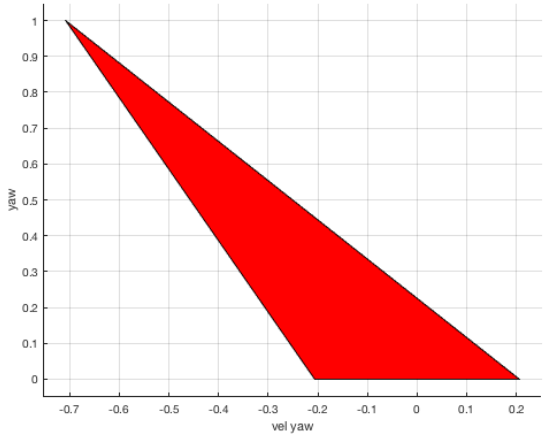
(a) Invariant set for the X controller.



(b) Invariant set for the Y controller.

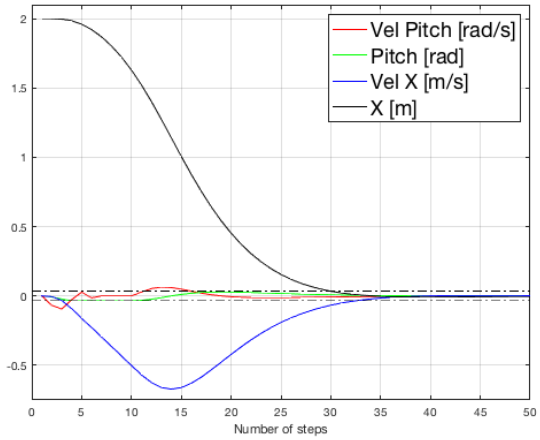


(c) Invariant set for the Z controller.

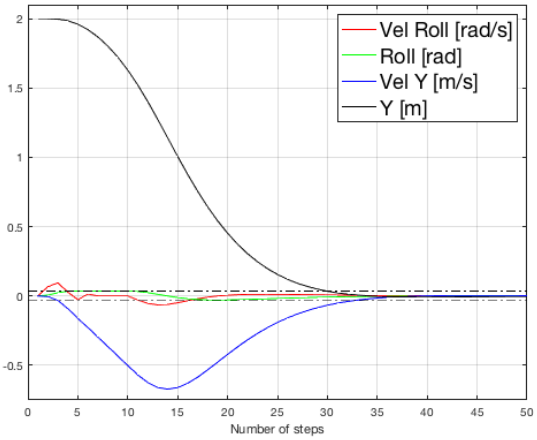


(d) Invariant set for the YAW controller.

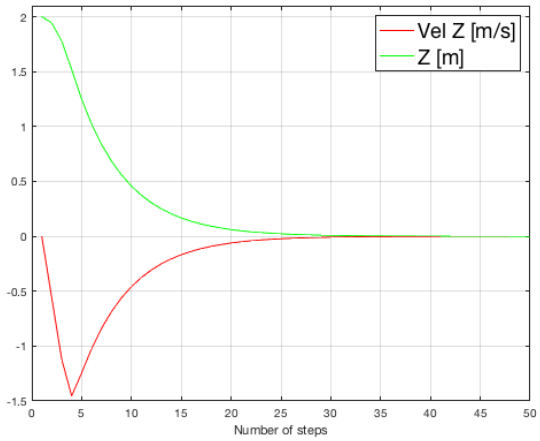
Figure 1: Plots of invariant sets for the 4 controllers.



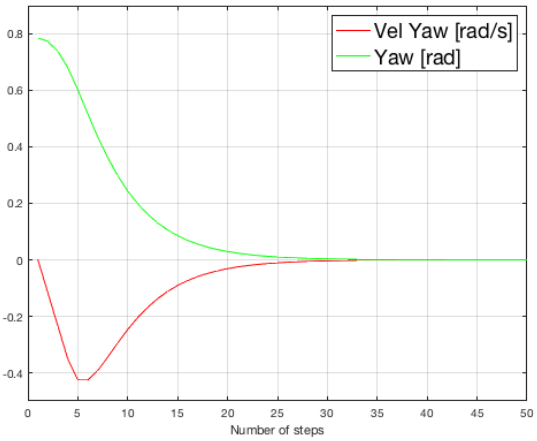
(a) Trajectory for the state of controller X. The dashed line is the pitch constraint.



(b) Trajectory for the state of controller Y. The dashed line is the roll constraint.



(c) Trajectory for the state of controller Z.



(d) Trajectory for the state of controller YAW.

Figure 2: Trajectories for the state of the 4 different controllers.

1.3 Deliverable 3.2

- Explanation of design procedures and choice of tuning parameters.

  We used YALMIP to compute x_s and u_s corresponding to reference r. The problem we want to solve is the following.

$$min\ u_s^T R_s u_s$$
$$s.t \begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ r \end{bmatrix}$$

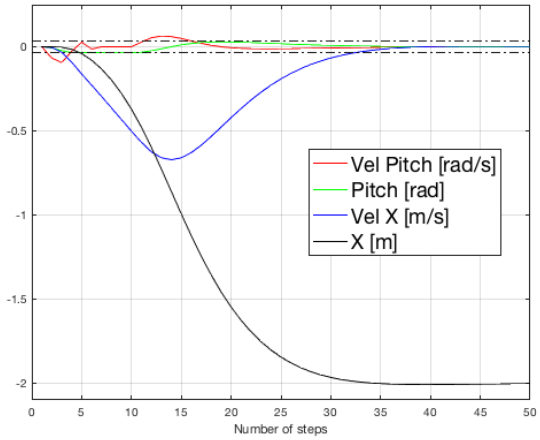
  To achieve tracking, we used the delta formulation. We removed the terminal invariant set as proposed in the instructions.

- Plot for each dimension starting at the origin and tracking a reference to -2 meters from the origin (for x, y and z) or stationary to 45 degrees for yaw.

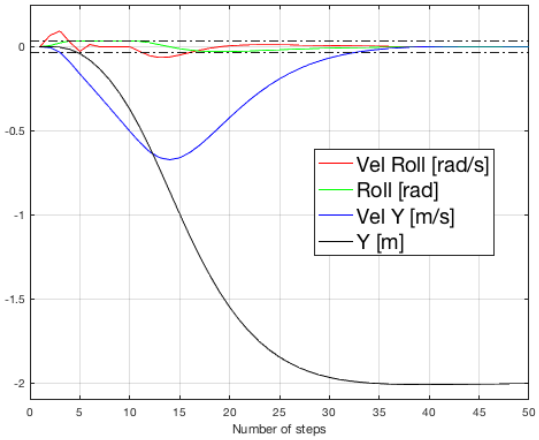
  The plots are given in Figure 3. They show that our controllers get to the reference in about 35 time steps (7 seconds), which is faster than the asked settling time. The black dashed lines on the plots represent the constraints on pitch and roll (between -0.035 and +0.035 rad).

- m-code for the four controllers, and code to produce the plots in the previous step.

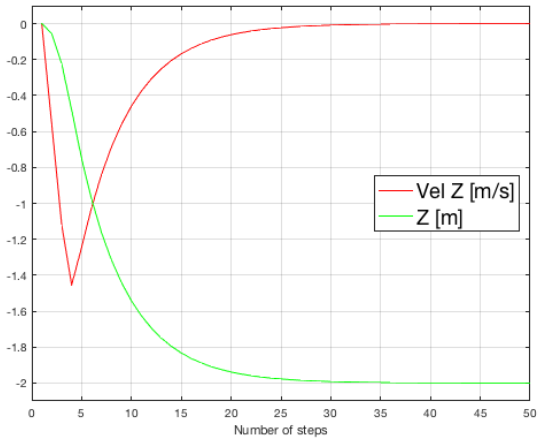
  You will find the m-code files in the zipped folder **Deliverable_3_2.zip**



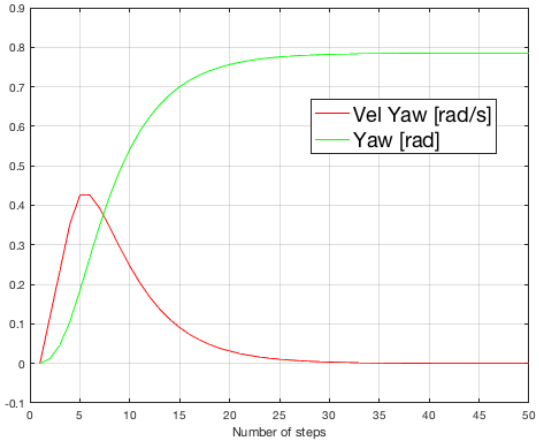
(a) Trajectory for the state of controller X. The dashed line is the pitch constraint.



(b) Trajectory for the state of controller Y. The dashed line is the roll constraint.



(c) Trajectory for the state of controller Z.



(d) Trajectory for the state of controller YAW.

Figure 3: Trajectories for the state of the 4 different controllers when tracking a reference.

1.4 Deliverable 4.1

- A plot of your controller successfully tracking the path using quad.plot(sim).

  Using the controllers, directly taken from deliverable 3, the performance is not quite satisfactory so we tune the controller a little further by augmenting the weights on position x,y,z to 10 (in matrix Q). This ensures a much better tracking of the desired path while maintaining the horizon N=10.

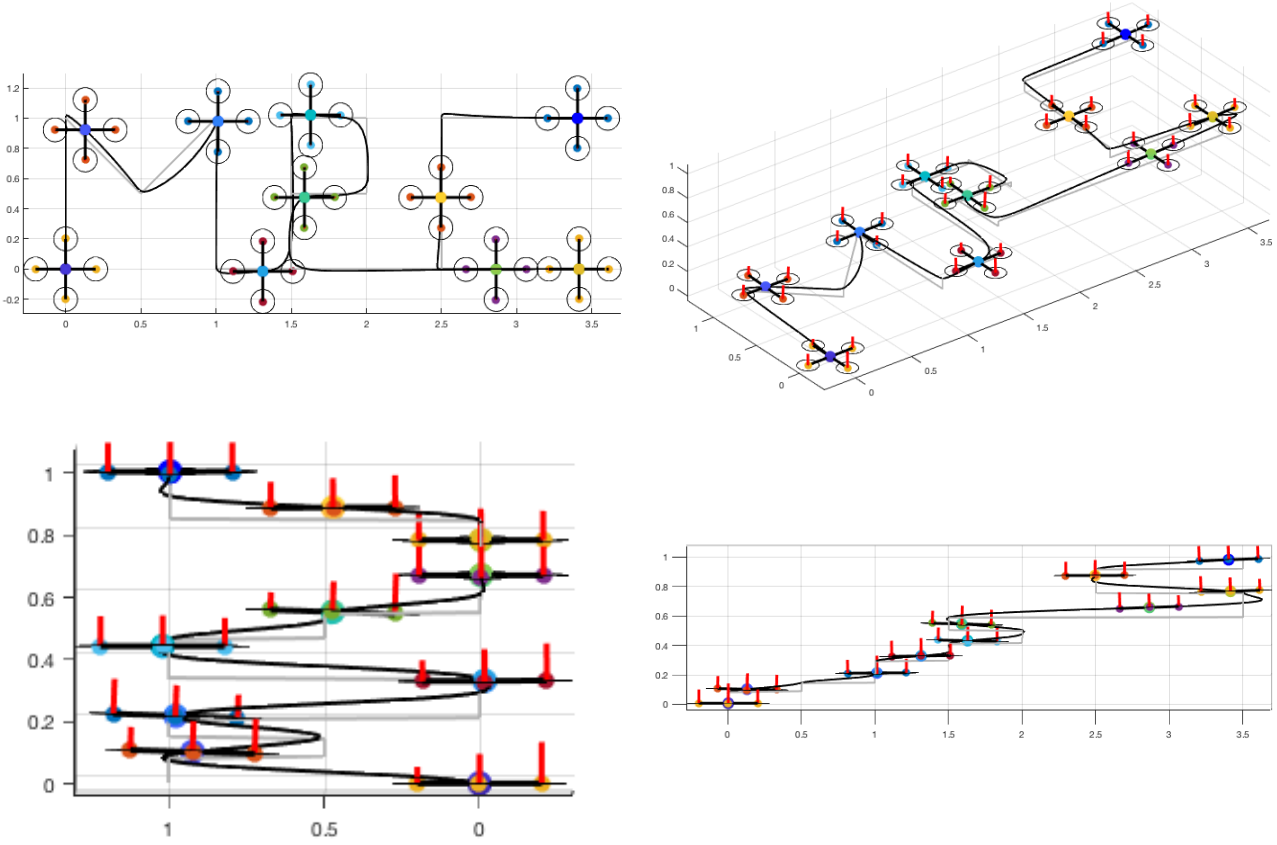


Figure 4: Plots of the controllers successfully tracking the path.

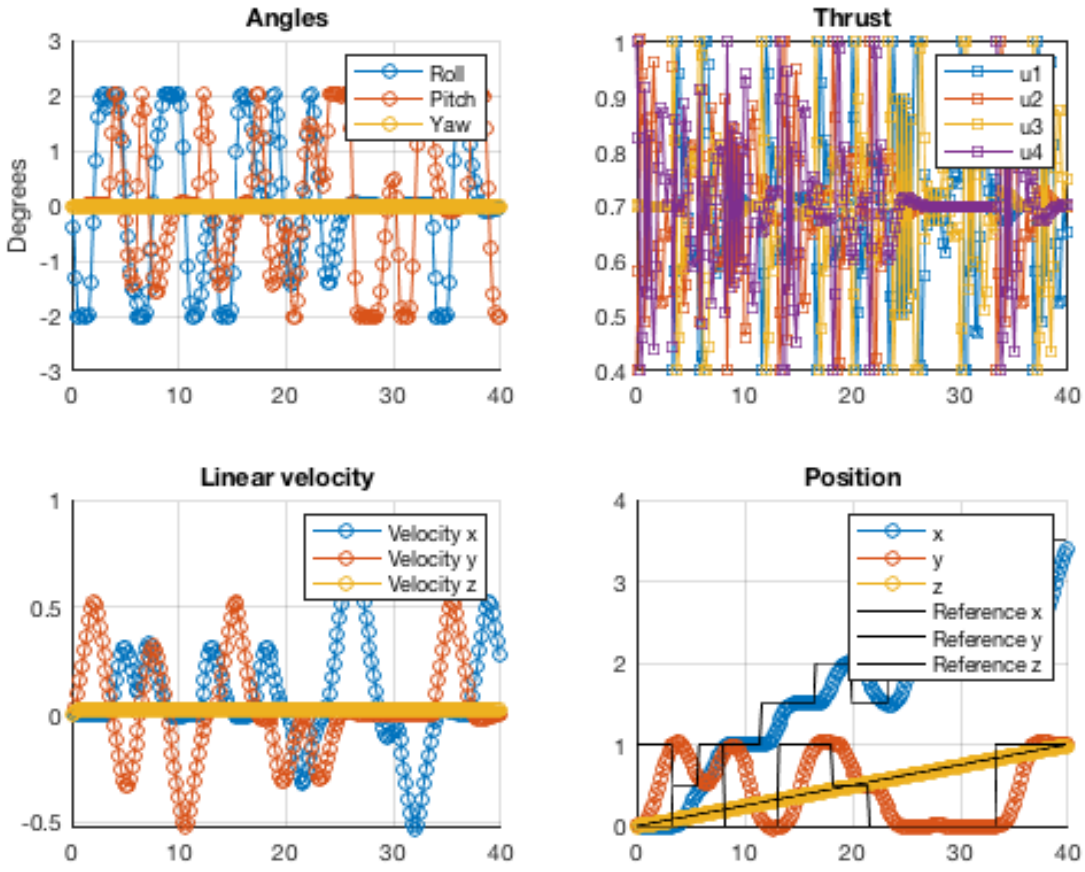


Figure 5: Plots of angles, thrust, linear velocity and position for the controllers when successfully tracking the path.

1.5 Deliverable 5.1

- Explanation of your design procedure and choice of tuning parameters.

  To implement offset-free tracking in presence of constant disturbance, we augmented the system as in the lecture. We get :

$$\begin{bmatrix} x_{k+1} - \hat{x}_{k+1} \\ d_{k+1} - \hat{d}_{k+1} \end{bmatrix} = \left(\begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} + \begin{bmatrix} L_x \\ L_d \end{bmatrix} \begin{bmatrix} C & C_d \end{bmatrix} \right) \begin{bmatrix} x_k - \hat{x}_k \\ d_k - \hat{d}_k \end{bmatrix}$$

  With the help of the matlab command 'place' it is easy to find a good observer matrix L. We decided to put the desired poles of the transfer function to be as close as possible to the center of the unit circle, to achieve fast convergence. Ideally, we would like to make them 0 to have a dead-beat observer but 'place' command does not permit it. Hence, we put them at $\begin{bmatrix} 0.001 & 0.002 & 0.003 \end{bmatrix}$ because the command does not allow for pole multiplicity.

  Once the L matrix is computed, we can observe the state and then estimate the disturbance in order to compensate it. We then change the way to compute the steady-state target to take into account this disturbance estimate as shown during the lecture.

$$min\ u_s^T R_s u_s$$
$$s.t \begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} B \cdot \hat{d_{est}} \\ r \end{bmatrix}$$

- Plot showing that the z-controller achieves offset-free tracking.

  You will find the plots in Figures 6 and 7. We see that the observer compensates for the bias since the path is well tracked, just as in deliverable 4. Also, the amount of thrust used when compensating is higher which is not a surprise.

- m-code for the four controllers, and code to produce the plots in the previous step.

  You will find the m-code files in the zipped folder **Deliverable_5_1.zip**

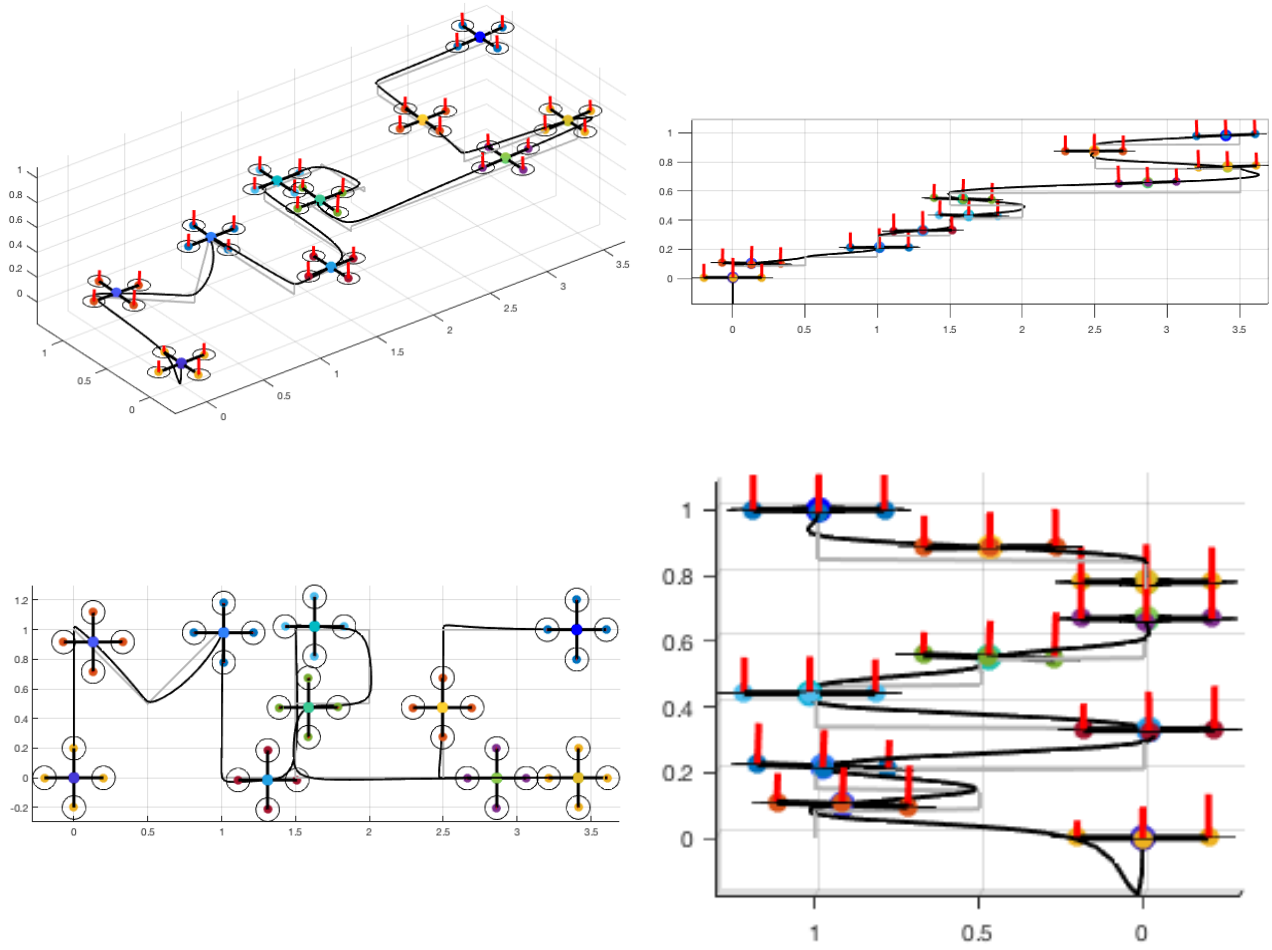


Figure 6: Plots of the controllers successfully tracking the path with BIAS = -0.1.

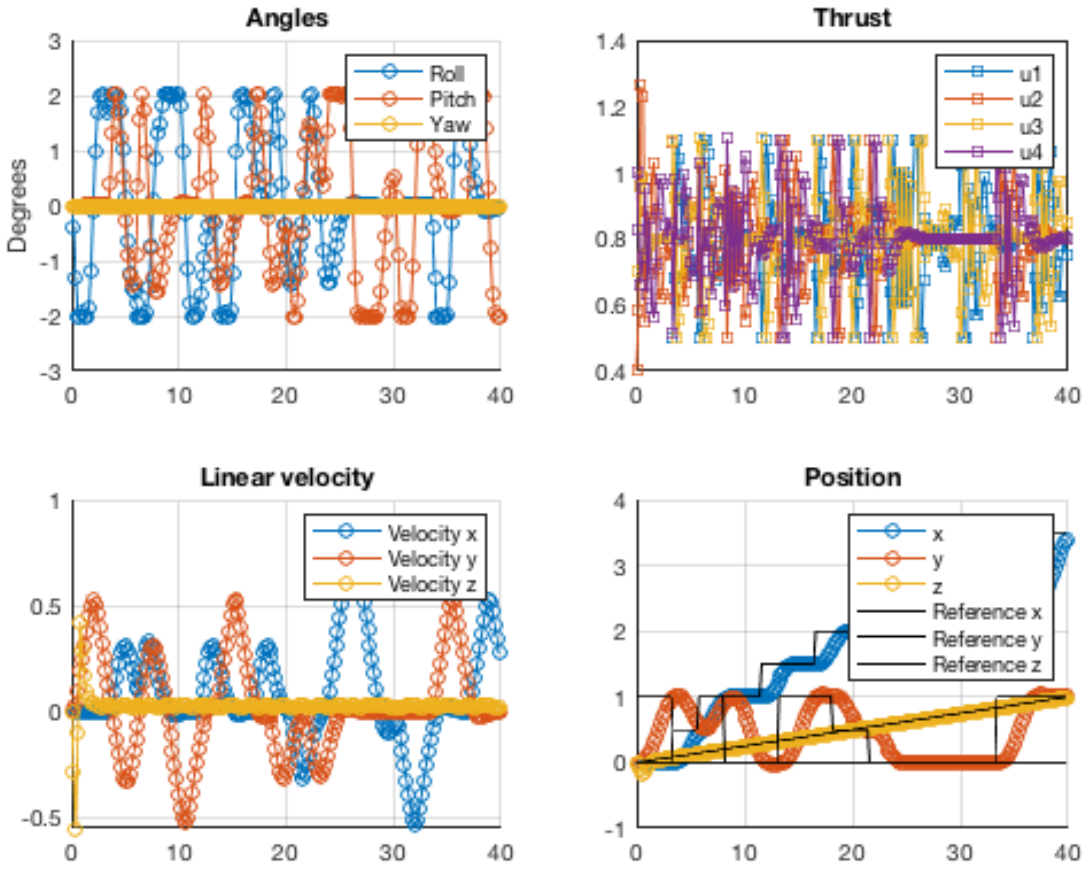


Figure 7: Plots of angles, thrust, linear velocity and position for the controllers when successfully tracking the path in presence of a BIAS.

1.6 Deliverable 6.1

- Explanation of your design procedure and choice of tuning parameters

  The design procedure was to first discretize the continuous function quad.f using the Runge-Kutta 4 (RK4 for short) method for great precision. We then set state and input constraints, respectively $x^+ = f(x, u)$ and $0 \leq u \leq 1.5$. After several iterations we figured it was best to constrain the speeds in x, y, z (to a minimum of 0.5 m/s for the problem to be feasible) and use a greater weight on the position in z. The objective function to minimize considers the distance to the reference point and the input of the motors that reflect energy use. A greater weight is given to the error in position as this is the important factor here. Without the input minimization the drone flies off to infinity or the problem has no solution.

- Explanation of why your nonlinear controller is working better than your linear one.

  The nonlinear controller considers the full system dynamics and only accounts for errors in integration which are negligible when using RK4. The linear controller on the other hand, as its name already tells us, is linear and hence can only be used inside its region of linearization. When leaving this region, the error grows more and more and there is a discrepancy between the resulting path and the planned path.

  The nonlinear quadcopter is working better than the linear one because it can take pitch and roll angles greater than 0.035 radians in magnitude (see "Angles" in Figure 9). This allows it to make harder and faster changes in direction when tracking the path (have a look at the M letter in the sub figure (a) of Figure 8 and compare it with the M letter in Figure 6 (a)). The path is tracked more efficiently.

  Despite the fact that the thrusters are sometimes saturating, the error in position remains small as we can see in the subfigure "Position". When we try to avoid saturation, the tracking of the path is worse (overshoot appears).

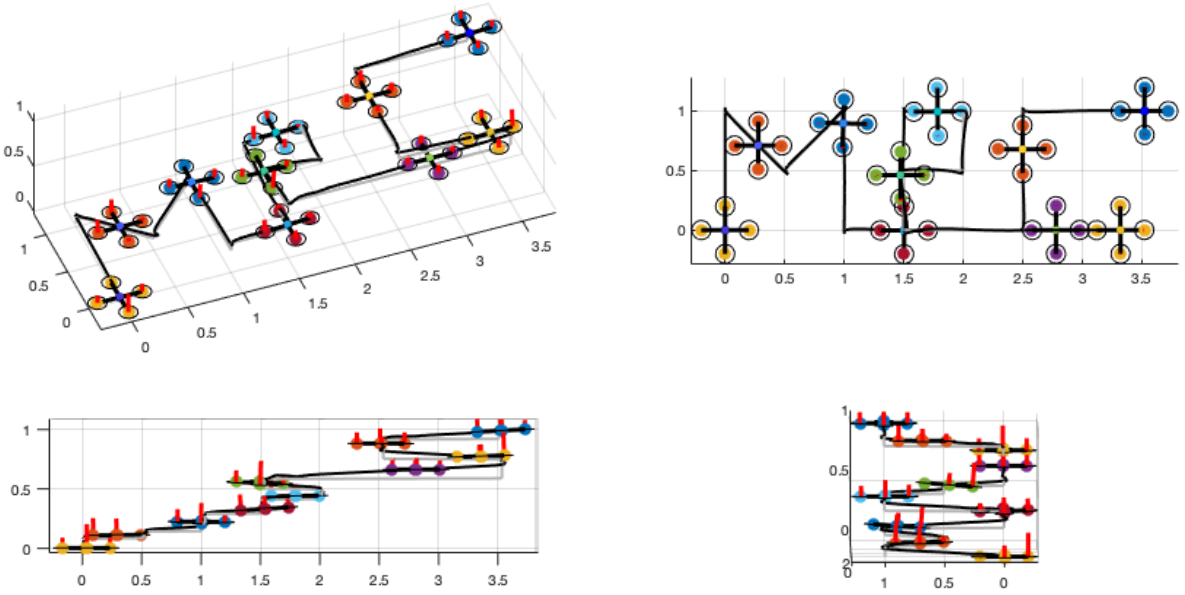


Figure 8: Plots of the drone's position in space from different viewing angles resulting from the use of the NMPC controller to follow the MPC trajectory with a horizon of $N = 10$.

- Plots showing the performance of your controller.

  You will find the plots in Figure 8 and in Figure 9. The plots demonstrate the almost perfect tracking of the path. We have very little overshoots at the edges (see the central edge of the M letter). The change in z position is however perfectly tracked. However, this comes at a greater computational cost and is thus slower, compared to the LMPC controller.

- m-code for your controllers, and code to produce the plots in the previous step

  The nonlinear controller is in ctrl_NMPC.m and the code to run in **Deliverable_6_1.zip**

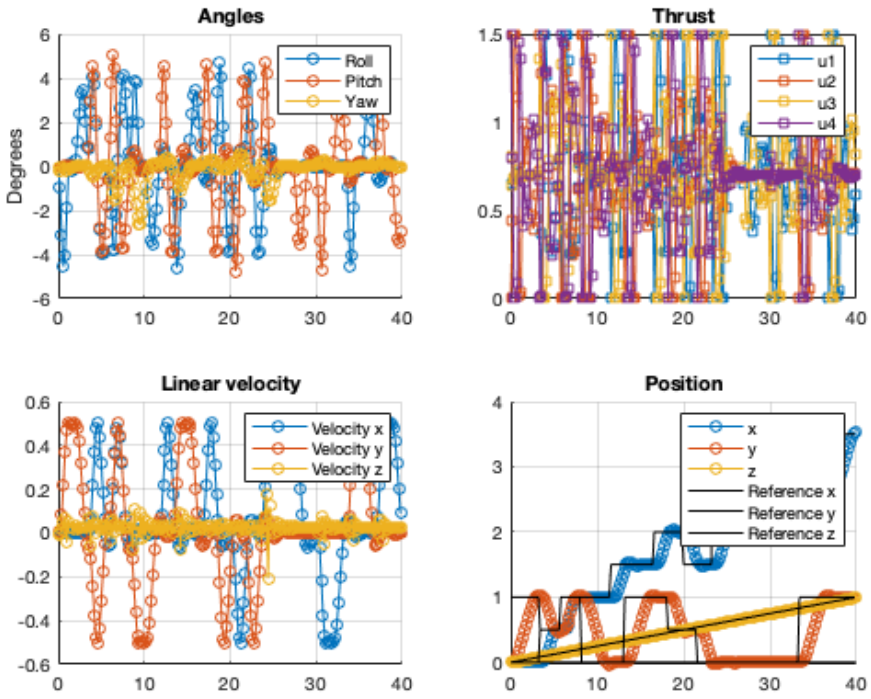


Figure 9: Plots of the state (angle, thrust, linear velocity and position) of the drone when using the NMPC controller to follow the MPC trajectory in Figure 8.