Model Predictive Control (ME-425)

Project Report

Students:
Cédric Portmann - 257168,
Luca Rezzonico - 274269,
Javier Balta - 247476

Professor:
Prof. Colin Jones

Lausanne, January 8, 2021

Contents

Part 1: System Dynamics

The model of the quadcopter is non-linear and can be written in standard form:

$$\begin{aligned}\dot{\mathbf{x}} &= f(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= g(\mathbf{x}, \mathbf{u})\end{aligned}$$

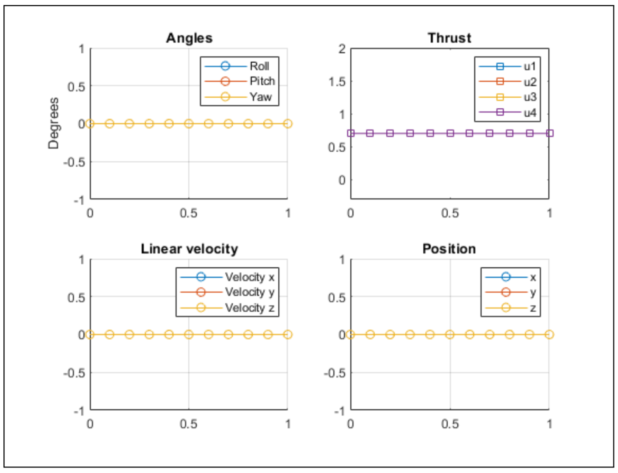
Where the states and the inputs are:

$$\mathbf{x} = \begin{bmatrix} \dot{\alpha} & \dot{\beta} & \dot{\gamma} & \alpha & \beta & \gamma & \dot{x} & \dot{y} & \dot{z} & x & y & z \end{bmatrix}^T$$
$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T$$

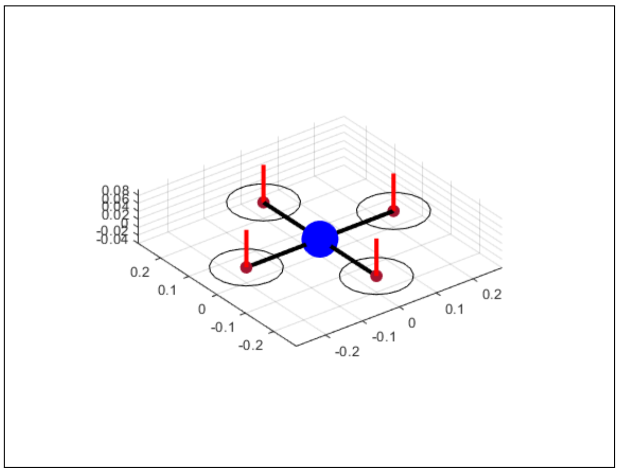
Part 2: Linearization and Diagonalization

2.1 Deliverable 2.1

The non-linear model of the quadcopter is linearized around the equilibrium point defined by the states x_s and the inputs u_s obtained using the steady-state condition $\dot{\mathbf{x}}_s = f(\mathbf{x}_s, \mathbf{u}_s) = 0$. This approximation is valid only near the equilibrium. There is an infinite number of equilibrium points in space, one at each coordinate (x, y, z). This condition is satisfied when the thrust of the four rotors exactly compensates for the gravitational pull on the quadcopter and all body moments are zero. Figure 2.1 shows an example of such a steady-state.



(a) Graphs showing the angle, velocity, and position of the quadcopter along the three axes, as well as the thrust inputs to the four rotors



(b) Quadcopter stablized at the origin

Figure 2.1: Quadcopter at a steady-state

A change of variables results in a more intuitive input:

$$\mathbf{v} = \begin{bmatrix} F \\ M_\alpha \\ M_\beta \\ M_\gamma \end{bmatrix} = T \cdot \mathbf{u}$$

The linearized model can be written as:

$$\dot{\mathbf{x}} = A \cdot \mathbf{x} + B \cdot \mathbf{u} = A \cdot \mathbf{x} + B \cdot T^{-1} \cdot \mathbf{v}$$

With the matrices:

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

$$B \cdot T^{-1} = \begin{bmatrix} 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.06667 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.125 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The linear dynamic equations can be rewritten as:

$$\ddot{\alpha} = \frac{1}{I_\alpha} \cdot M_\alpha,\ \ddot{\beta} = \frac{1}{I_\beta} \cdot M_\beta,\ \ddot{\gamma} = \frac{1}{I_\gamma} \cdot M_\gamma$$
$$\ddot{x} = g \cdot \beta,\ \ddot{y} = -g \cdot \alpha,\ \ddot{z} = \frac{1}{m} \cdot F$$

Based on these equations, it is apparent that there is a link between the state variables β and x and another link between α and y. This results in four uncoupled systems, with each input controlling one system independently.

After the change of variables, the steady-states of the quadcopter are reached when:

$$\mathbf{v}_s = \begin{bmatrix} mg & 0 & 0 & 0 \end{bmatrix}^T$$
$$\mathbf{x}_s = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & y & z \end{bmatrix}^T$$

Since the systems are decoupled, this would also be the case for other steady-state conditions as long as the axis of the quadcopter is parallel to the gravity vector.

Part 3: Design MPC Controllers for Each Sub-System

3.1 Deliverable 3.1

The method for ensuring recursive constraint satisfaction consists of solving two subproblems. The first part involves solving a normal constraint problem for the states up to $k = N$. For the second part, concerning $k > N$, an unconstrained LQR controller is solved, guaranteeing that it is invariant. The LQR controller was implemented using Matlab's **dlqr** function.

For the optimizer, the constraints were added to an array which is passed to the YALMIP optimizer. The terminal cost was computed using the infinite horizon set P given by the previously calculated **dlqr** function.

The tuning parameters Q, R and N were determined experimentally. After trial and error, it was determined that for N the value 30 was sufficient, since anything less than 25 made the MPC problem infeasible. Increasing the horizon N to more than 30 had no significant effect on the performance.

For the two subsystems sys_x and sys_y we argue for the choice of Q and R:

The state cost matrix Q, has an influence on the parameters $\dot{\beta}, \beta, \dot{x}$ and x for the subsystem sys_x and on $\dot{\alpha}, \alpha, \dot{y}$ and y for the subsystem sys_y. Its components were chosen as follows:

$$Q = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix}$$

No weights were added to the off-diagonal components as the quadcopter behavior appeared correct without. Penalties were more important for a deviation of the states x and y than for the roll and pitch angles α and β. Less importance was given to the derivatives of these states, since the problem statement demands optimal position tracking.

The coefficient for the input cost matrix R for the two subsystems sys_x and sys_y was chosen $R = 0.01$, because it is assumed that the moments M_β and M_α of the quadcopter can be easily applied.

For the two subsystems sys_{yaw} and sys_z, an argument can be made for the choice of Q and R:

The state cost matrix Q, has an influence on the parameters $\dot{\gamma}$ and γ for the subsystem sys_{yaw} and on $\dot{z}$ and z for the subsystem sys_z. It's components were chosen as follows:

$$Q = \begin{bmatrix} 0.5 & 0 \\ 0 & 5 \end{bmatrix}$$

More penalty was applied for a deviation of the states γ and z than for their derivatives, as the position was desired to be tracked correctly. The penalty for z was chosen to be less important than for the states x and y, in order not to disturb the correct positioning of the quadcopter laterally.

The coefficient for the input cost matrix R for the two subsystems sys_{yaw} and sys_z was chosen $R = 0.1$, because the moment M_γ and the force F of the quadcopter can be applied a bit less easily than the moments M_β and M_α.

The following eight graphs show the terminal sets for the x and y subsystems (projected on their states), as well as the z and yaw subsystems:

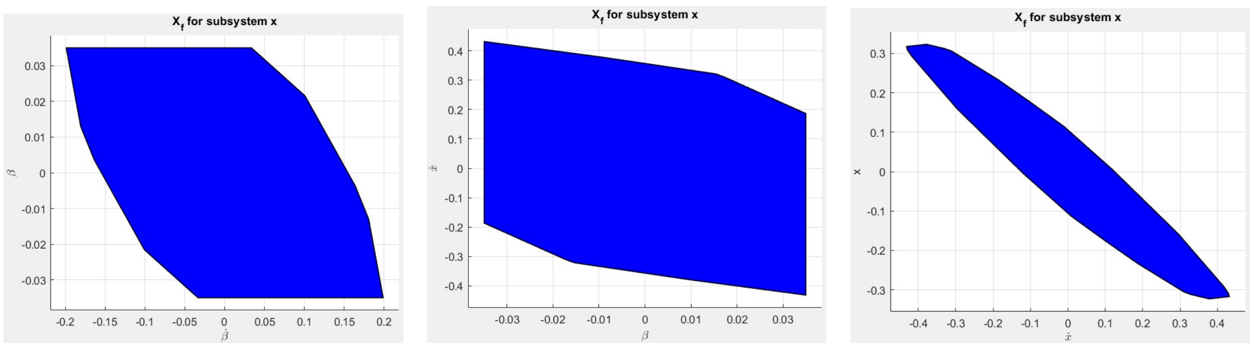


Figure 3.1: Terminal sets for sub-system x

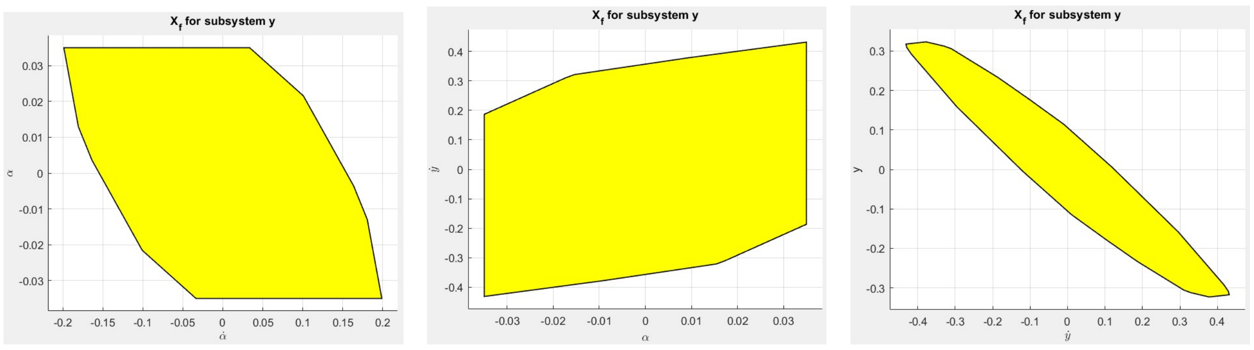


Figure 3.2: Terminal sets for sub-system y

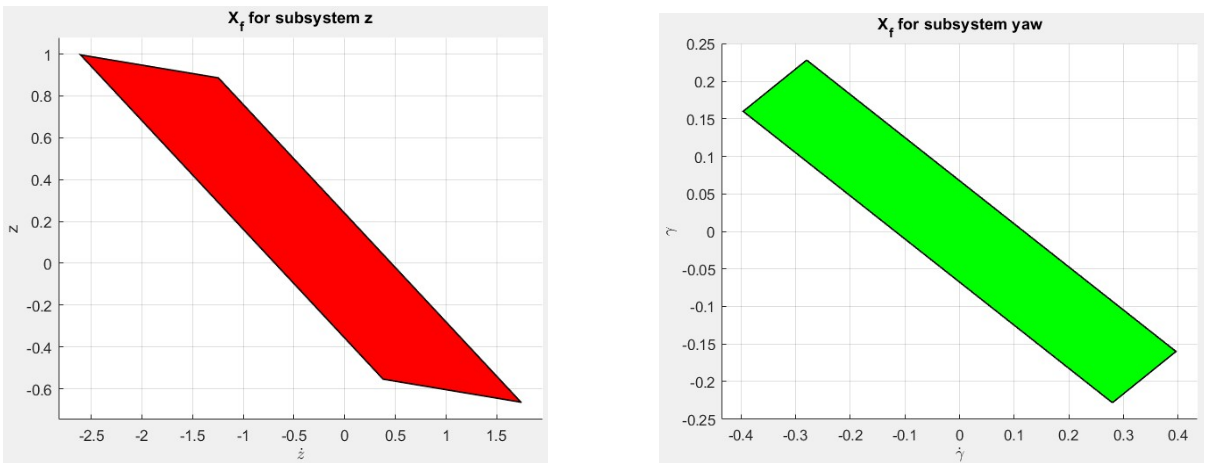


Figure 3.3: Terminal sets for sub-systems z and yaw

-

The following four graphs show the evolution of the x, y, z and yaw states. As can be seen, all stabilize before the eighth second:

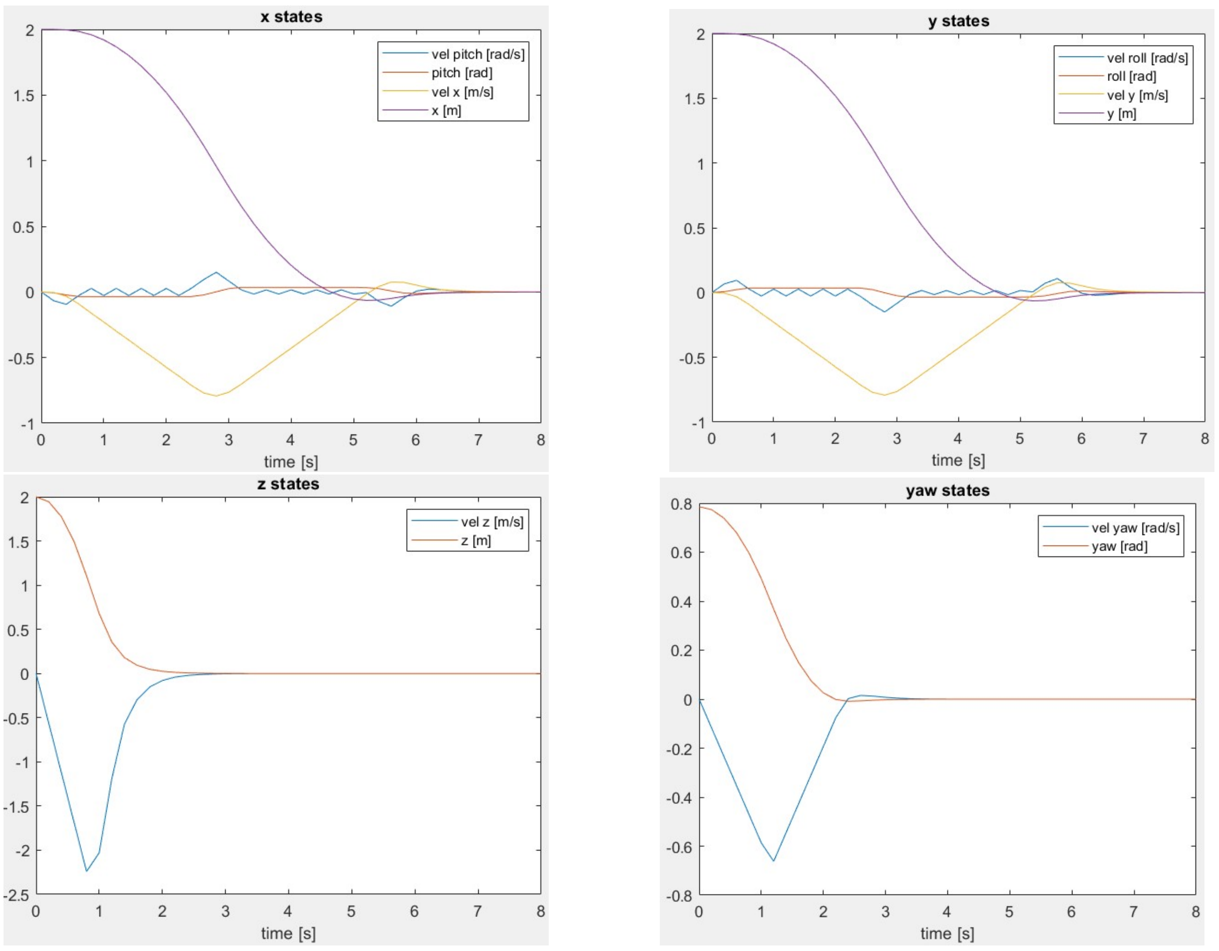


Figure 3.4: Recovery of states x, y, z and yaw after deviation

3.2 Deliverable 3.2

For this part, the same values for the parameters as in section were taken. The following four graphs show the evolution of the x, y, z and yaw states. The terminal set has been dropped, as recommended in the problem definition, while the terminal weight is still implemented.

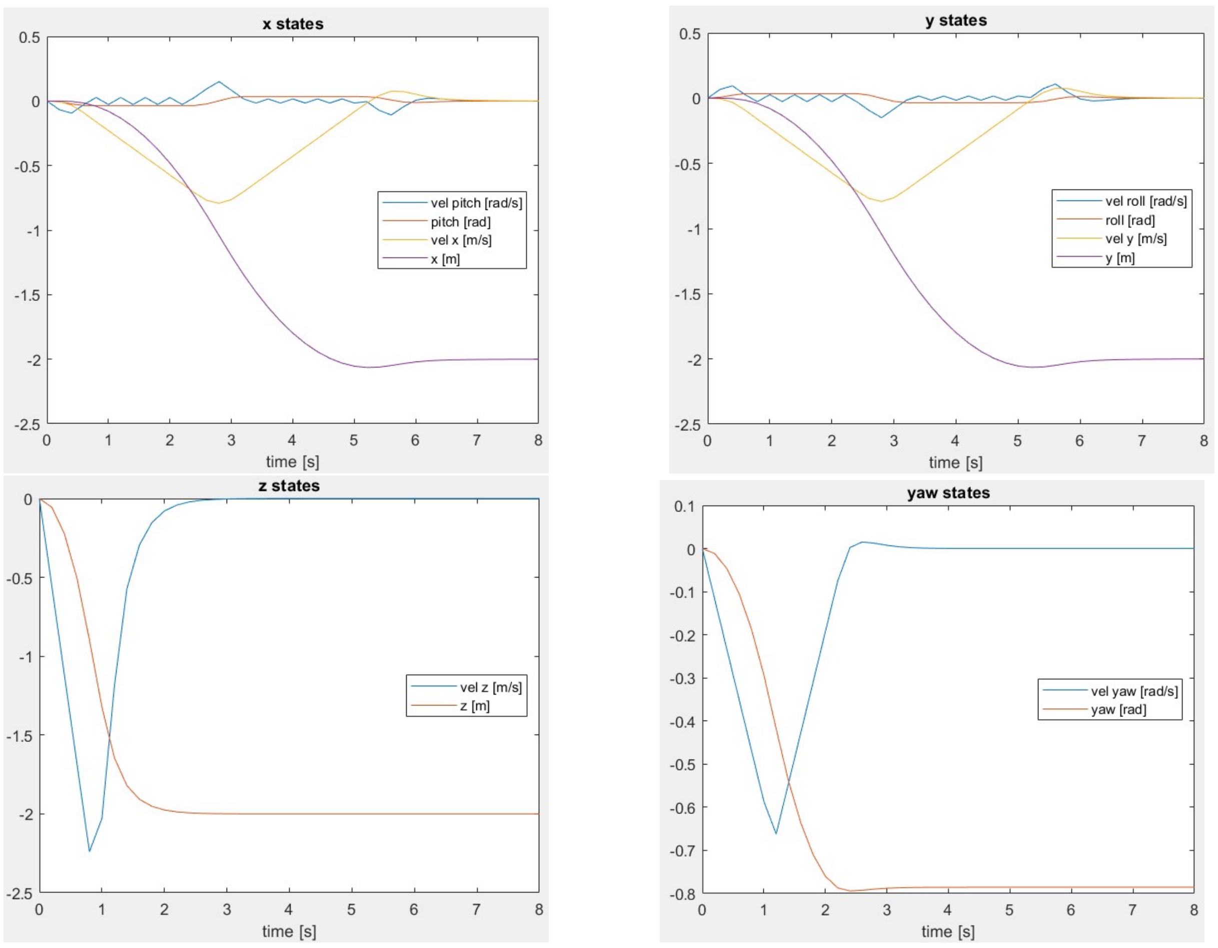


Figure 3.5: Recovery of states x, y, z and yaw after deviation

Part 4: Simulation with Nonlinear Quadcopter

4.1 Deliverable 4.1

The following two figures show the required path and the tracking of the quadcopter in three different perspectives.

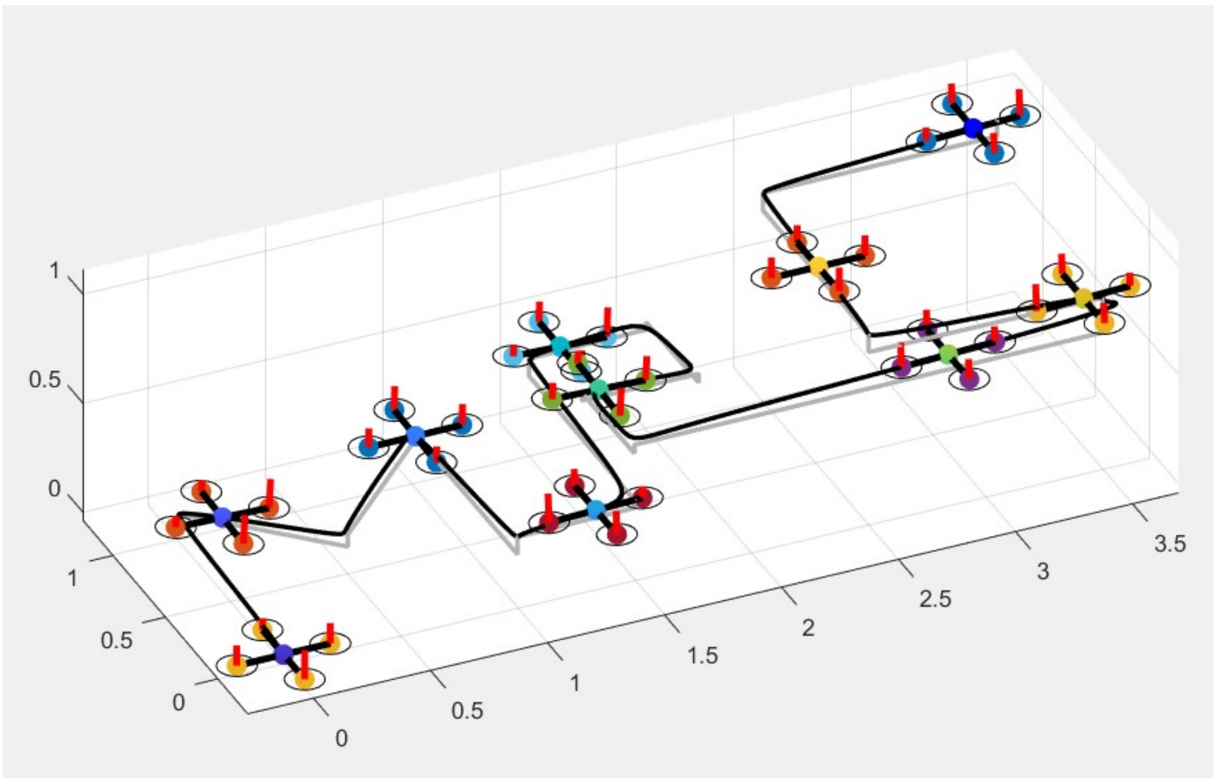


Figure 4.1: Quadcopter trajectory and tracking path from Matlab simulation, 3D view

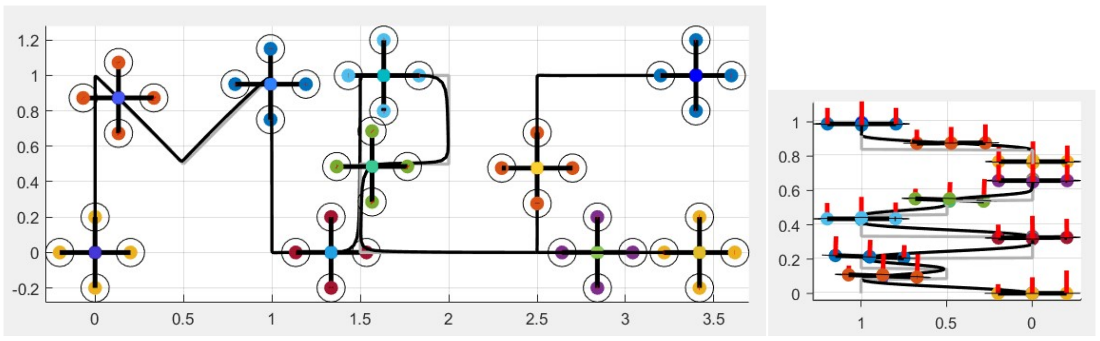


Figure 4.2: Quadcopter trajectory and tracking path from Matlab simulation, top and side views

Figure 4.3 shows the evolution of the parameters angle, thrust, linear velocity and position when tracking the trajectory with the linearized model. It is worth noting that the angles and thrusts experience some saturation, due to constraints on the pitch and roll angles. There is also a slight offset for the z-state of the position, which is to be expected with this model.

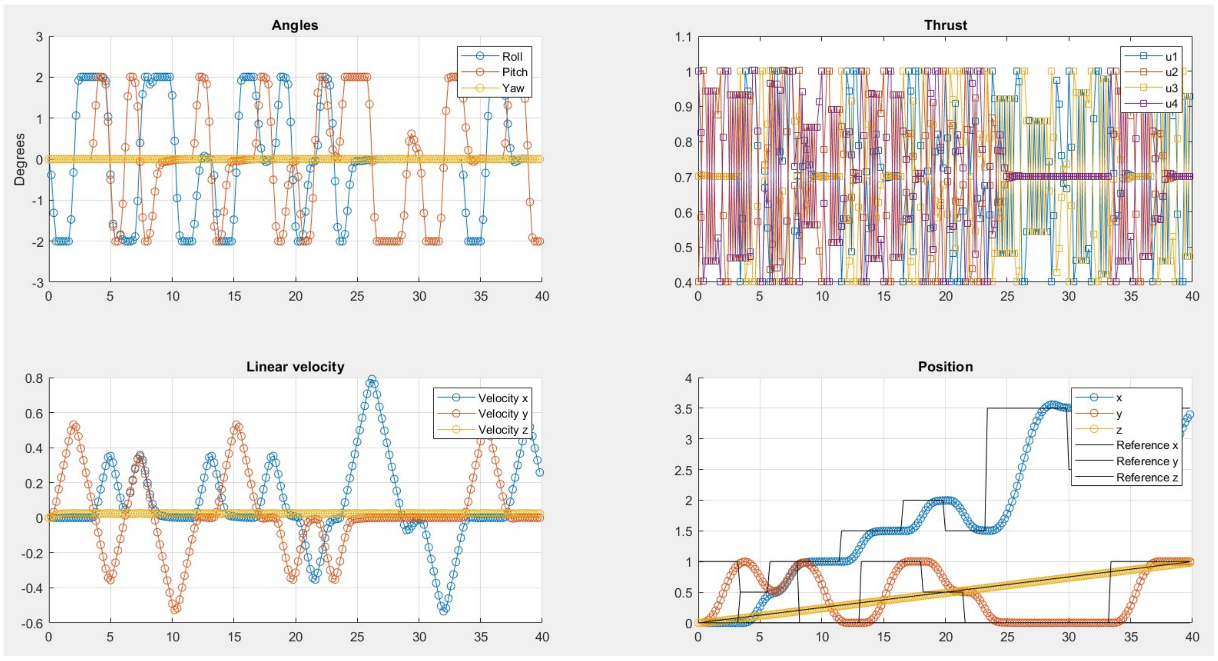


Figure 4.3: Parameters (angles, thrust, linear velocity and position) during trajectory tracking with the linearized model

Part 5: Offset-Free Tracking

5.1 Deliverable 5.1

This part requires compensation of a noise component. Therefore, an adjustment of the model is required resulting in the following:

$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{d}_{k+1} \end{bmatrix} = \begin{bmatrix} A & B_d \\ 0 & I \end{bmatrix} \begin{bmatrix} \hat{x}_k \\ \hat{d}_k \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u_k + \begin{bmatrix} L_x \\ L_d \end{bmatrix} (C\hat{x}_k + C_d\hat{d}_k - y_k)$$

With its compact form:

$$\begin{aligned} \bar{x}^+ &= \bar{A}x + \bar{B}u \\ y &= \bar{C}\bar{x} \end{aligned}$$

The L matrix was computed using Matlab's **place** function, with the poles $\begin{bmatrix} 0 & -0.001i & 0.001i \end{bmatrix}$. The poles were determined experimentally, and showed satisfying results. The idea behind the placement was to place them the closest to the origin as possible in order to have fast poles and thus a valid estimation quickly. In this part, an observer was used to estimate the offset and the state of the system and thus there is no longer a guarantee that the constraints are satisfied. Consequently, the terminal set has been dropped as recommended in the problem definition, while the terminal weight remains implemented.

Figure 5.1 shows the evolution of the angles, thrust, linear velocity and position when tracking the path with the offset-free z-controller. The same behavior of saturation as in section 4.1 can be observed. However, using the noise-compensating model the z-state for the position quickly stabilizes to the desired trajectory after an initial dip.

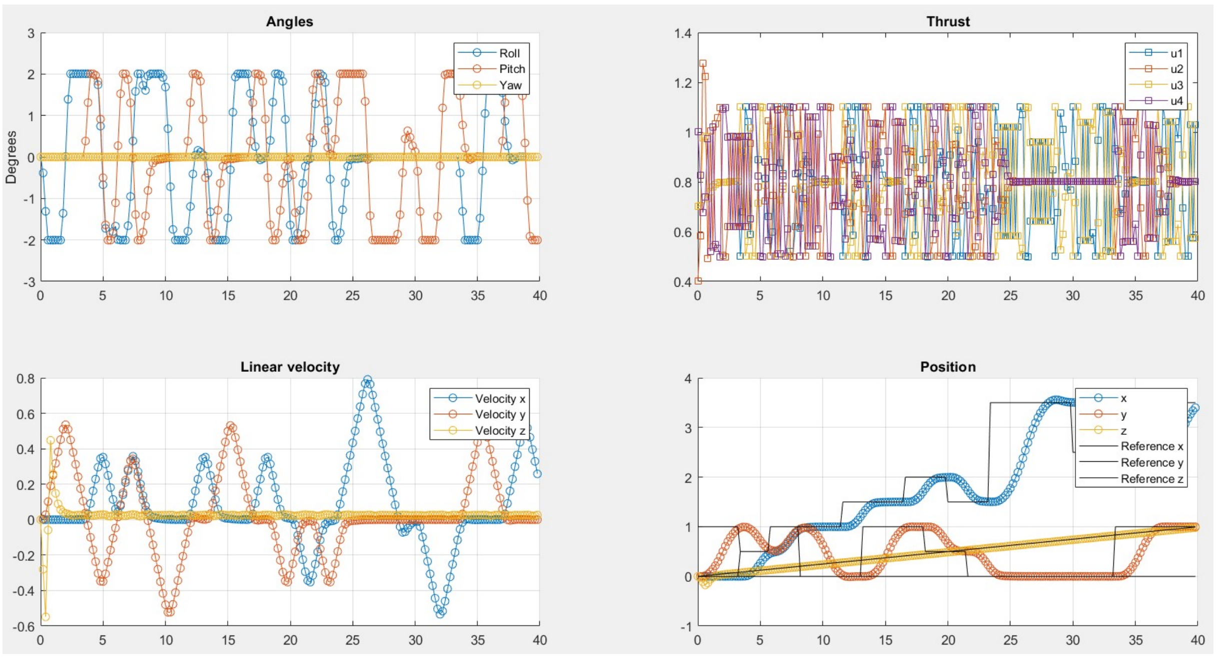


Figure 5.1: Parameters (angles, thrust, linear velocity and position) during trajectory tracking with the linearized model and the offset-free z-controller

The following two figures show the required path and the tracking of the quadcopter in three different perspectives.

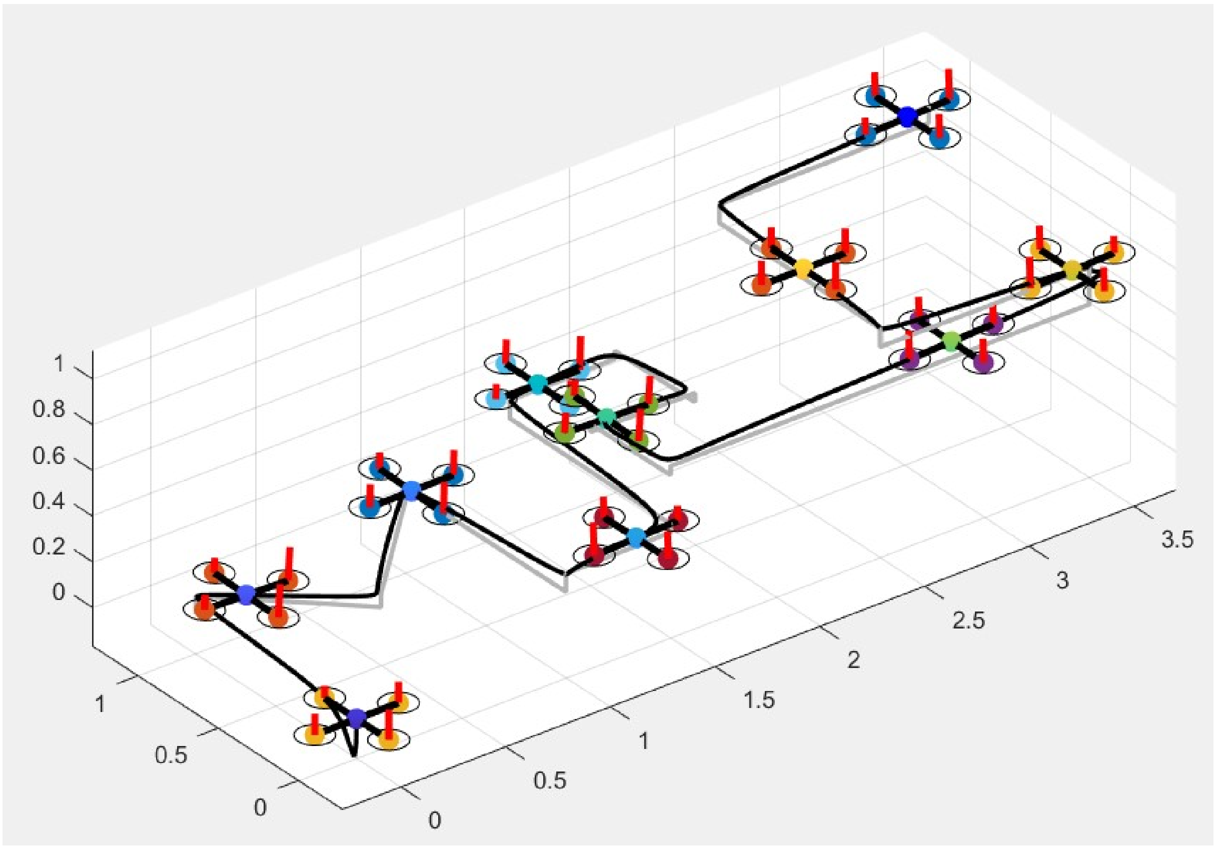


Figure 5.2: Quadcopter trajectory and tracking path from Matlab simulation, 3D view

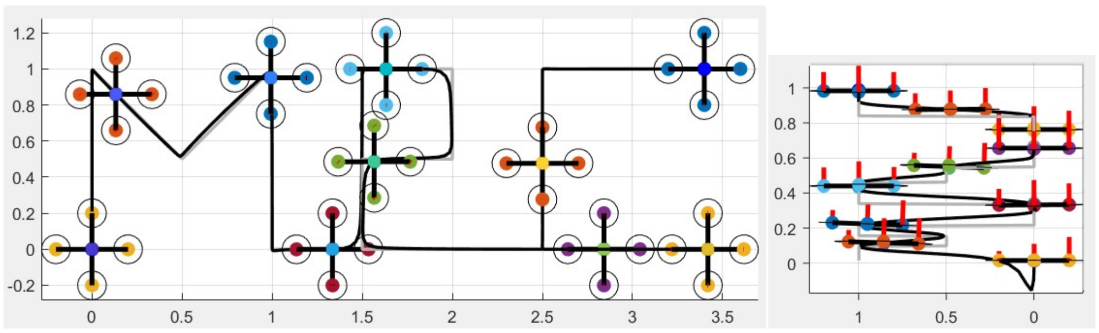


Figure 5.3: Quadcopter trajectory and tracking path from Matlab simulation, top and side views

Part 6: Nonlinear MPC (Bonus)

6.1 Deliverable 6.1

The nonlinear MPC problem was implemented using CASADI, and was based on the problem:

$$J^*(x) = \min_{x,u} \sum_{i=0}^{N-1} (x_i - x_{ref})^T Q (x_i - x_{ref}) + u_i^T R u_i + (x_N - x_{ref})^T Q (x_N - x_{ref})$$

$$x_{i+1} = f_{discrete}(x_i, u_i)$$
$$M \cdot u_i \le m$$
$$x_0 == X_0$$

The nonlinear dynamic of the system was discretized using the Runge-Kutta (4^{th}) method.

The horizon N had to be reduced from 30 to 20. While, generally, predictions are better with a longer horizon, it is not so for reference tracking, which needs to react faster to follow the reference, which is easier with a shorter horizon. In other words, since the important thing is to follow the required path, and not simply get to the end point, having a longer horizon would potentially go for a straighter path to the end point.

Since we don't decompose the model into subsystems, we only have one Q and one R matrix and set them to:

$$Q = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&5&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&5&0&0\\ 0&0&0&0&0&0&0&0&0&0&5&0\\ 0&0&0&0&0&0&0&0&0&0&0&5000 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.0001&0&0&0\\ 0&0.0001&0&0\\ 0&0&0.0001&0\\ 0&0&0&0.0001 \end{bmatrix}$$

In Q, a deviation of state z is penalized more heavily, otherwise the altitude of the quadcopter seems to fit the trajectory poorly. Small penalties are added to the states x, y and yaw, which is sufficient to track the path correctly.

For the value of R the same justification follows as previously, but in this case the inputs are the thrusts given to each rotor. Independently the thrusts can be applied more easily than previously the force on the whole quadcopter, for this reason the values are smaller.

To be able to track the reference position this vector was converted to a reference state vector

$$\begin{bmatrix} x & y & z & yaw \end{bmatrix}^T \to \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & yaw & 0 & 0 & 0 & x & y & z \end{bmatrix}^T$$

No terminal set was implemented for this method and the constraints we had for the linear model can be omitted, since the non-linear model is also valid away from the equilibrium points.

Figure 6.1 shows the evolution of the parameters angle, thrust, linear velocity and position when tracking the trajectory with the nonlinear MPC controller.

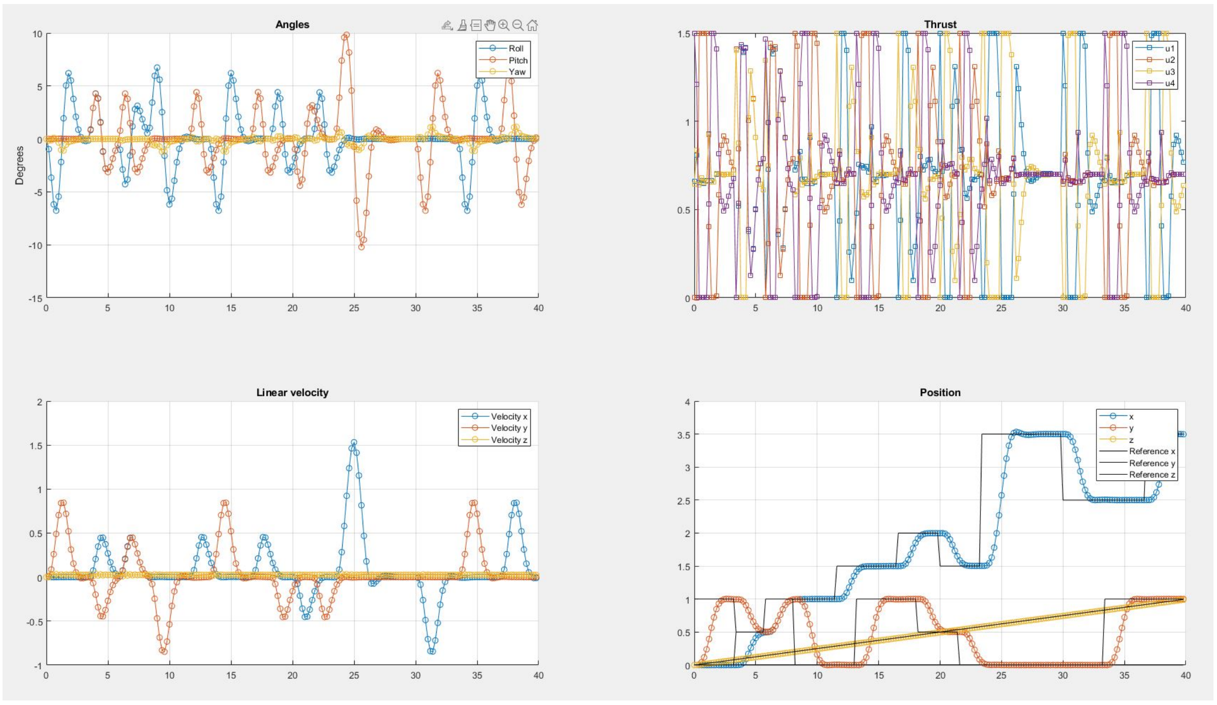


Figure 6.1: Parameters (angles, thrust, linear velocity and position) during trajectory tracking with the nonlinear model

The following two figures show the required path and the tracking of the quadcopter in three different perspectives.

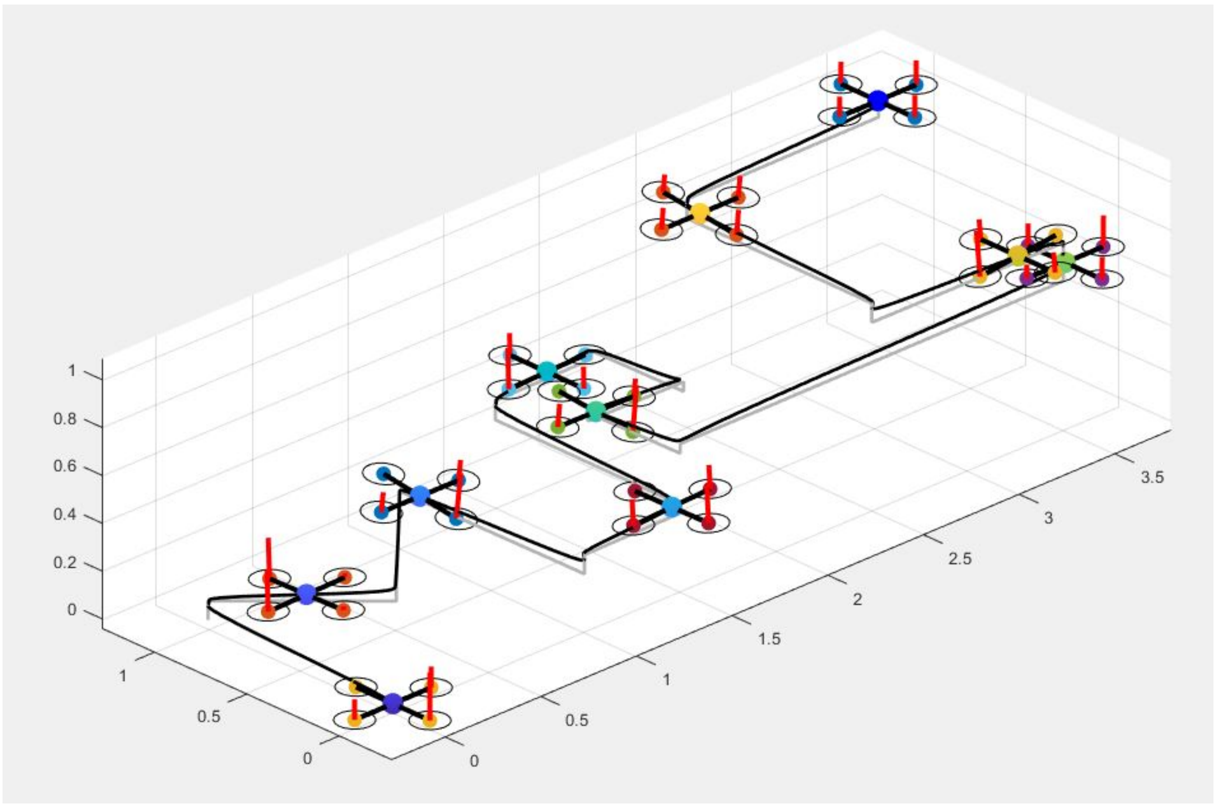


Figure 6.2: Quadcopter trajectory and tracking path from Matlab simulation, 3D view

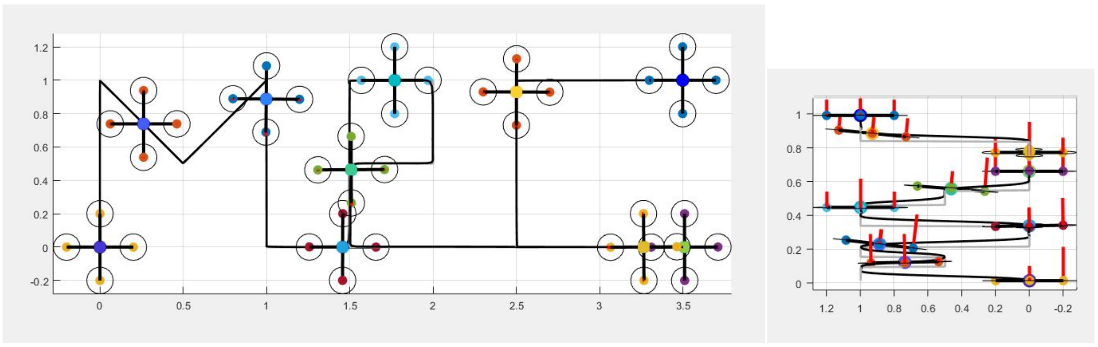


Figure 6.3: Quadcopter trajectory and tracking path from Matlab simulation, top and side views

It can now be observed that the trajectory tracking is almost perfect. In Figure 6.1 the position, converges much faster to the desired values. This result is due to the absence of constraints that are no longer needed. In contrast to Figure 5.1, where the constraints were necessary due to the linearization. As a consequence, a more aggressive correction behavior is possible. The upper left part of the Figure 6.1 shows that the angles often exceed the constraint imposed for the linearization in the previous sections, reaching a maximum value of about 10°.

The advantage is that this method no longer requires linearization of the model. The linearized model used in the beginning was only valid for small roll and pitch angles around the steady states, though now the model is correct for any angle. The disadvantage is that this method requires significantly more computational resources compared to the previous method.