Project

Quadcopter

Model Predictive Control
ME: 425

Olivier Charrez: 271550
Matthieu Jacobs: 331404
Johannes van Randenborgh: 331071

January 8, 2021

Contents

1 Introduction

This project is based on a MATLAB simulation to design an MPC controller for a quad-copter. A framework with the drone model and functions for plotting and simulating are given. This project handles the implementation and design of the controllers.

Multiple concepts of MPC are examined to have an overall understanding. The first part consist of analyzing the system of the quad-copter and its decomposition into independent subsystems. Linear controllers are then designed for each subsystem. A terminal set is used to guarantee recursive feasibility.

The influence of a disturbance is also studied. A constant bias, which can be interpreted as an error in the weight is set on the drone. The disturbance makes the previous controller unsuitable because it lacks the estimation of disturbance. The new designed controller has an estimator to adapt the trajectory accordingly. This estimator is only added to the subsystem in the z-direction as the disturbance is only applied here.

The last part consists of a nonlinear controller taking the whole (non-linearized) model. This is expected to give better performance. Soft constraints are used to have a smoother motion.

The project has several original MATLAB scripts, one controller for the whole system and one for each independent subsystem, called MPC_Control. Some lines need to be added inside these scripts to have the desired controller. The script Quad is the global simulation using the controllers and output a 3D plot to visualize the results of the motion. All the other scripts have been added by us for the tasks. There is one script for each studied case, they are like the main file where everything runs. The rest of the files are functions that are executed inside the controller or the deliverable for specific cases.

The project uses external libraries to speed up the computation, it's not mandatory to use them but the solver and optimizer of MATLAB is quite slow compared to these libraries (Gurobi and Casadi as optimizer, YALMIP and MPT3 as solver).

2 Deliverable 2.1

The transformation is based on the following equations. The four motors with lift and the moments around the body axis are connected via the transformation matrix.

$$x^+ = Ax + Bu \quad with \quad v = Tu = \begin{bmatrix} F \\ M_\alpha \\ M_\beta \\ M_\gamma \end{bmatrix} \tag{1}$$

$$x^+ = Ax + BT^{-1}v \tag{2}$$

After applying the transformation to the system, we can observe that only the B matrix has changed. It is recognizable, that for both matrices the input has no direct influence on the location and orientation of the quadcopter.

```
B =
                  u1       u2       u3       u4
 vel_roll          0     0.56        0    -0.56
 vel_pitch     -0.56        0     0.56        0
 vel_yaw      0.7333  -0.7333   0.7333  -0.7333
 roll              0        0        0        0
 pitch             0        0        0        0
 yaw               0        0        0        0
 vel_x             0        0        0        0
 vel_y             0        0        0        0
 vel_z           3.5      3.5      3.5      3.5
 x                 0        0        0        0
 y                 0        0        0        0
 z                 0        0        0        0

B_transformed =
                       F    M_alpha     M_beta    M_gamma
 vel_roll    8.971e-19          0.1  -1.636e-35  -3.324e-18
 vel_pitch   1.705e-18    3.37e-18          0.1  -3.324e-18
 vel_yaw    -3.354e-18  -6.476e-18  -6.807e-18     0.06667
 roll                0           0           0           0
 pitch               0           0           0           0
 yaw                 0           0           0           0
 vel_x               0           0           0           0
 vel_y               0           0           0           0
 vel_z           0.125  -1.561e-17   1.009e-17           0
 x                   0           0           0           0
 y                   0           0           0           0
 z                   0           0           0           0
```

The new inputs are lift and the three moments of each axis. The vector is based on the body coordinate system. If values below $O(E-16)$ are neglected you can see that the new inputs have only one impact on one state. The lift of the motors only influences the velocity in z-direction. The moments influence only the angular velocity of the equivalent axis. This special structure of the transformed matrix indicates a possible decomposition of the whole (input-)system into sub-systems.

Moreover, the A matrix has the same structure (as the modified B matrix), so that one state input has only impact on one state output. Thus, the (dynamic-)system can be decomposed as well. With a closer look it becomes obvious that changing the angular rolling velocity of one axis only needs inputs of motor u2 and u4 (roll-moment), for the pitch-velocity u1 and u3 (pitch-moment), for yaw- and z-velocity all of them. Now, each sub-system has got a specific and unique input, we can decouple the whole system into four sub-systems. The x-axis is dependent on the pitch, the y-axis is dependent on the roll and the z-axis is dependent on the overall lift F.

```
[sys_x, sys_y, sys_z, sys_yaw] = quad.decompose(sys, xs, us)
```

The decomposition can be further illustrated with the transformation matrix, where k_F is a force constant, k_M a momentum constant and L a distance constant:

$$T = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & k_M \end{bmatrix} \tag{3}$$

The lift of the quadcopter is dependent on all four motors. The first new input, the first row of the transformed matrix, is only dependent on the lift of each motor, alpha momentum related to the roll depends on the second and fourth motor, beta momentum related to the pitch depends on the first and third motor and the fourth momentum gamma related to the yaw depends on every motor. The roll and pitch momentum depends on the distance between the center and the motor and the lift of the motor. The yaw momentum depends on the momentum generated by each motor.

Overall the quad-copter is now represented by four new axis, all independent between themselves and representing the motion properly. The steady state values (0 = f(xs,us)) for the position and the inputs are as follows:

```
xs =

     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
```

```
18      us =
19
20          0.7007
21          0.7007
22          0.7007
23          0.7007
```

xs is filled with zeros because the approximation is made around the initial position, which does not require any rotation or displacement. us is filled with 0.7007 which means that every motor needs the same thrust in order to keep the quadcopter at the same height. us is thus the required loitering thrust.

The steady-state condition is not dependent on the location (x, y, z, yaw) of the quadcopter, so that every random value of x, y, z, and yaw could be at steady-state. All further states, e. g. angular velocity, roll, pitch, velocity (x, y, z), have to be zero in order to keep the quadcopter in steady-state. In conclusion, for other steady-state conditions some values of xs can change but not us because the robot will need to stay stable and "work" against gravity at steady-state. The other new transformed matrices will not change either because they are just a reflection of axis transformation.

3 Deliverable 3.1

In order to achieve recursive satisfaction of the input and angle constraints we need to compute a terminal set that fulfills all input and state constraints for the closed-loop system. The terminal set defines terminal constraints at the end of the horizon that will be computed in this task, assuming a LQR controller as terminal controller.

To find the terminal set we need to consider the state constraints, input constraints and a closed loop gain. The input constraints as well as the state constraints are given and vary from subsystem to subsystem. The state constraints depend on the limits set on the roll or pitch angle. The x controller, for example, has four state variables, velocity pitch, pitch, velocity x and x. The given limit only applies to the second state variable (pitch). Regarding the input constraint, the system has only one input, e. g. M_β. The general input constraint $0 \leq T^{-1}\mathbf{v} + u_s \leq 1.5$ doesn't apply anymore. To have a good terminal set for a closed-loop system we introduce a feedback controller. The most simple and efficient one is a LQR-controller. However, we notice that the computed terminal set is not the maximal terminal set, due to limitations of LQR-controller. To compute the terminal set the intersection of the constraints polyhedron and the preset (states and inputs that end in the constraints polyhedron after one step) are compared. If there are no differences the terminal (invariant) set is found. From Figure 3 it can be seen that the terminal sets are rather small. Therefore the horizon is chosen large enough to ensure the MPC problem is feasible.

Now, that the invariant set is computed, the robot motion/costs need to be handled. For every step in the horizon N, the state and input constraints are added as well as the system dynamic constraints. The objectives are also computed, where the cost and system values are used. The cost matrix Q used for the distance correction is shown in the following equations, where each diagonal element is the constraint for each state. The R matrix is the cost matrix used for inputs. These are defined and tuned for each subsystem separately.

$$\begin{aligned} Q_x &= diag(0.5, 1, 0.5, 1) \\ Q_y &= diag(0.5, 1, 0.5, 1) \\ Q_z &= diag(1, 10) \\ Q_{yaw} &= diag(1, 10) \end{aligned} \tag{4}$$

$$\begin{aligned} R_x &= 0.1 \cdot eye(2) \\ R_y &= 0.1 \cdot eye(2) \\ R_z &= 0.1 \cdot eye(2) \\ R_{yaw} &= 0.001 \end{aligned} \tag{5}$$

The optimal values of Q and R matrix are determined by considering the physical meaning of every state. As it can be seen in Equation 4, the velocities are considered as lower costs, followed by the angles and position. This makes sense because we want to track or regulate the drone's position, not the velocity. The velocity

costs can't be chosen too low, as this would lead to large overshoot. This effect is illustrated in Figure 5. Regulating the drone to z=0, which can be interpreted as the ground would then lead to the drone crashing. The positions offset is regarded as high costs. The input for the quadcopter is interpreted as little costs, since the quadcopter is not in e. g. a "low battery"-mode (see Equation 5). Overshoot is nearly extinguished manually with bigger velocity costs after the computation of Q and R matrix with raised velocity costs. The horizon is set to 30, to ensure the MPC problem is always feasible, i.e. also when a large reference step is applied. The terminal costs are taken equal to the weight of the terminal LQR controller. This ensures recursive stability, as the cost function will be a Lyapunov function, meaning it decreases monotonically until it reaches the origin.

With the given Q and R matrix we simulate the behavior of the quadcopter for case 1 and case 2 of the task in Figures 1 and 2. The second figure clearly shows that at steady state the velocity will be equal to zero in all directions, confirming the format of the steady state values for linearization. For case 1 the quadcopter starts with an offset of 2 meters in each direction. In case two the quadcopter starts without offset with a yaw-angle of 45.

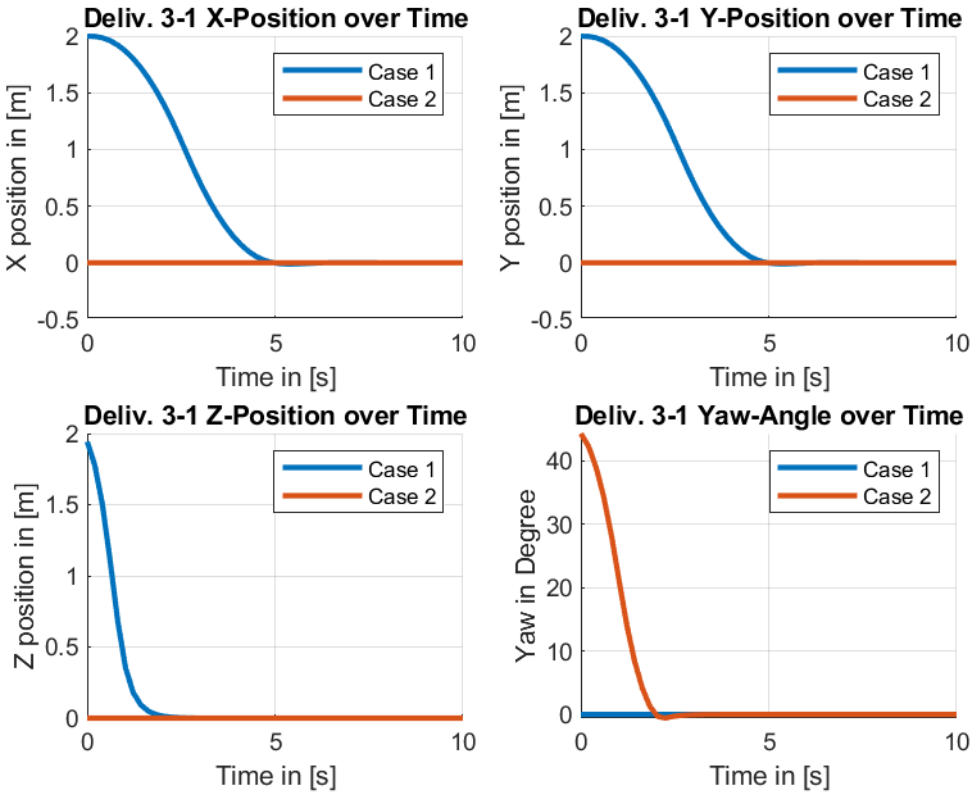

Figure 1: Settling Behavior of Quadcopter for Case 1 and 2: Position

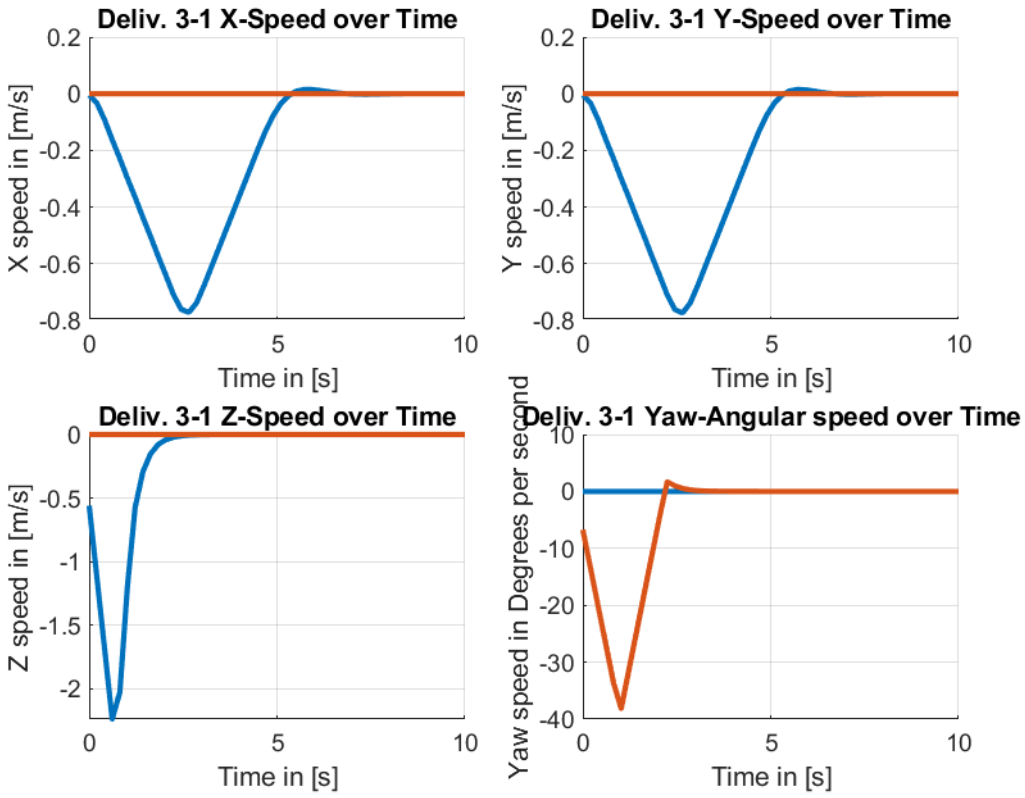

Figure 2: Settling Behavior of Quadcopter for Case 1 and 2: Speed

In Figure 3 we show the invariant sets for every sub-system based on the terminal LQR controller. The figure just beneath (Figure 4) shows the evolution from the initial constraints to the terminal constraint through the iterative methods of comparing the preset with the terminal set. This clearly shows the sets get much smaller than the originals due to the tight constraints on the angles.

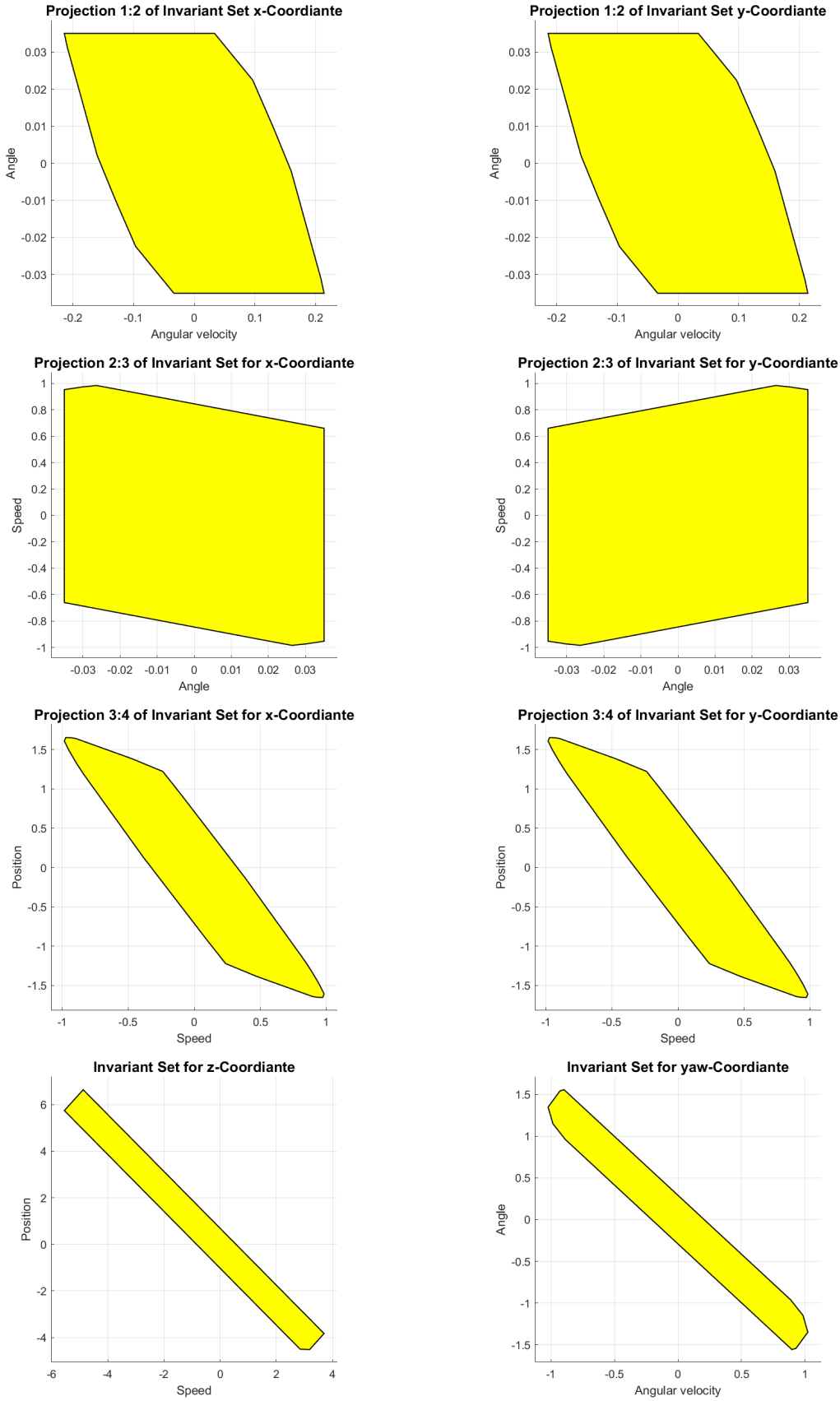

Figure 3: Terminal Sets of each Subsystem)

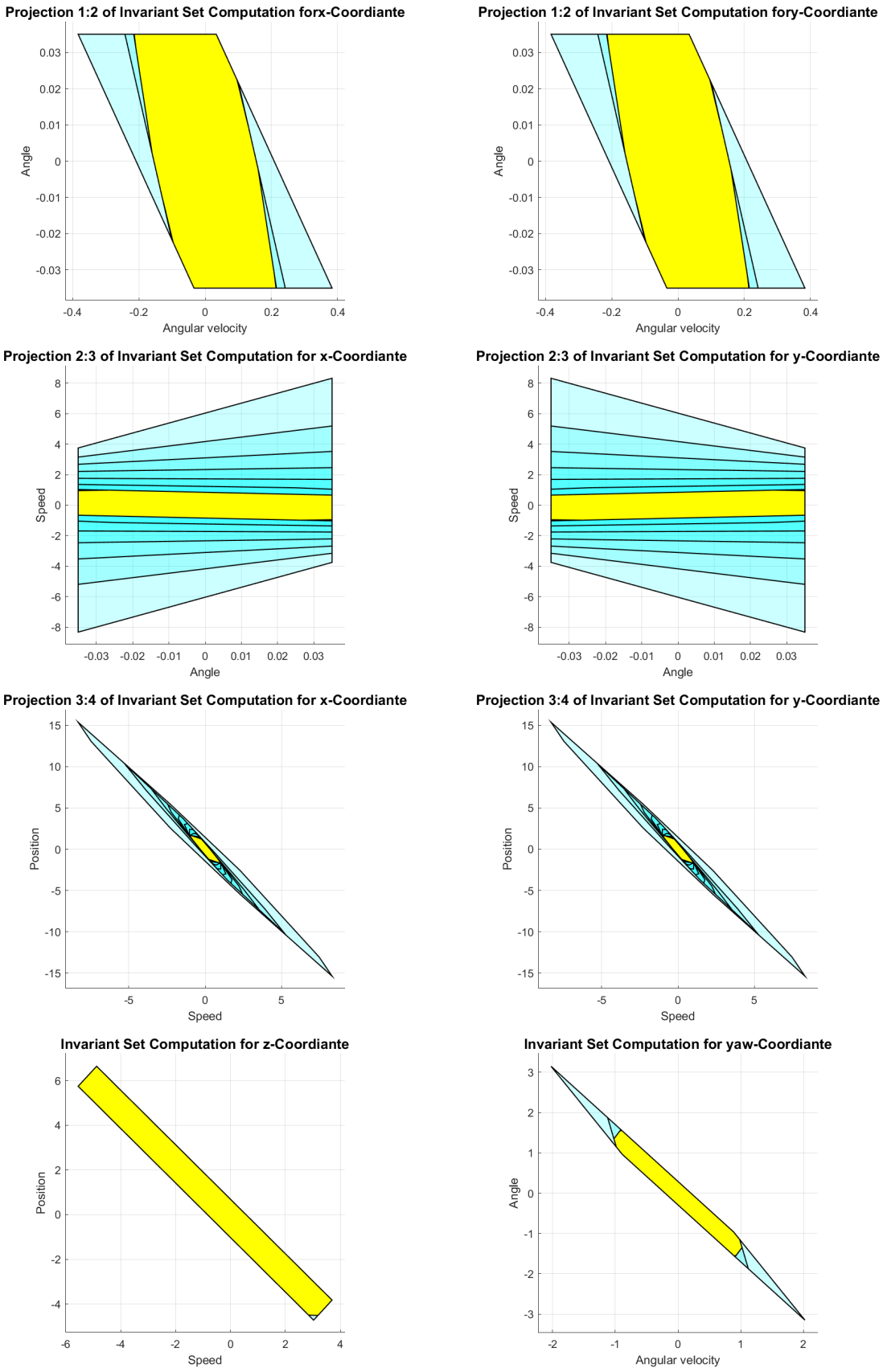

Figure 4: Computation of Terminal sets

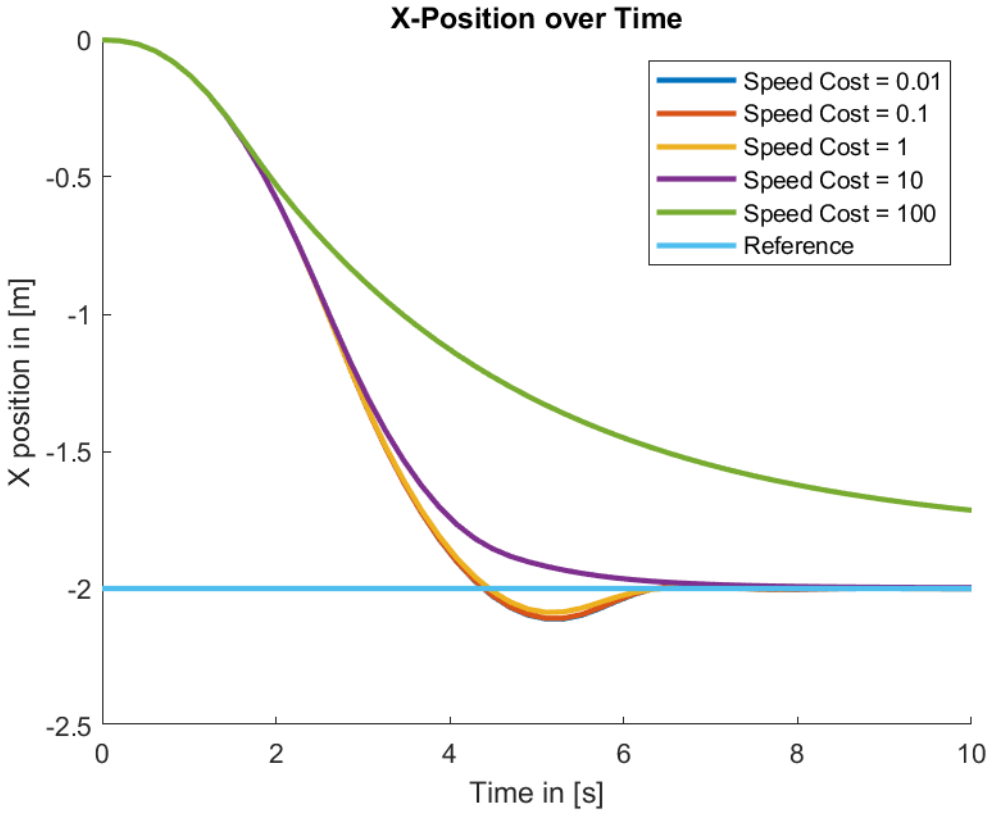

Figure 5: Behavior of Quadcopter in X-Direction for Different speed weights

4 Deliverable 3.2

For reference tracking a reference is introduced to the controller (see Equation 6). For this deliverable, the quadcopter starts at the origin and needs to move to the reference. All sub-systems are needed to respond in order to fulfill the task. The delta between the starting position and the reference is the same in comparison with Case 1 and 2 from Chapter 3. Since the subsystems are independent from each other, a combination of Case 1 and 2 does not imply changes in Q and R matrix. Thus, the same Q and R matrix is implemented as shown in Equations 4, 5. The settling behavior to the reference is given in Figure 6. As expected this leads to the exact same behaviour as for deliverable 3.1. The graphs are just shifted to the new reference.

$$x_{ref} = [0, 0, 0, 0, 0, 45°, 0, 0, 0, -2, -2, -2]^T \tag{6}$$

The terminal set from Chapter 3 can still be used for this deliverable since a shifted version of the terminal set along the states x, y, z and yaw is still invariant. The reference is added to the setup_steady_state_target-function, as shown below. To compute the steady state reference the constraints on state and input must still be satisfied. The dynamic constraints are modified so that the next state is equal to the current state, ensuring that this is a steady state and this steady state leads to the output matching the reference. The chosen cost only includes the input. This ensures maximum availability of the drone engines for movement and also makes sense from a dynamic point of view in this case as a minimal input will be achieved when the drone is loitering in a constant position.

```
1 con = [];
2          con = [con,xs == mpc.A*xs+mpc.B*us];
3          con = [con,mpc.C*xs==ref];
4          con = [con,H*xs≤h];
5          con = [con,G*us≤g];
6          obj = us'*us;
```

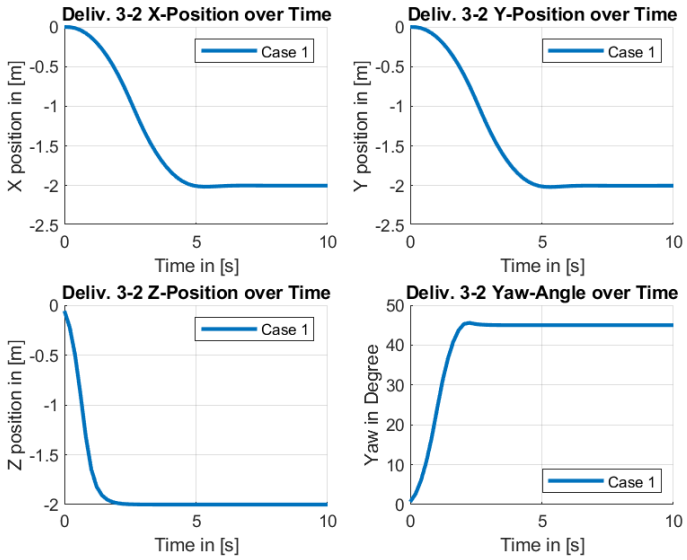

Figure 6: Settling Behaviour of Quadcopter to given Reference

5 Deliverable 4.1

The figures and plots for the simulation in Deliverable 4.1 are given in the following (see Figure 7 and 8).

In the first figure you can see on the left side that in the first seconds the quadcopter rolls negatively and lifts off in positive y-direction. This supports the descriptions in Chapter 2. The max. roll angle is limited by the constraints. The quadcopter starts to slow down early in order to catch the first corner properly. For the first corner the quadcopter slows down to zero velocity, as for all corners / set points of the path. The position plots show that the quadcopter sometimes has trouble reaching the reference in time. There are different reasons for this. First, the control inputs are computed based on the linear model. They are then applied to the nonlinear model, which means there will be a difference between the predicted trajectory and the followed one (even during the first time step). Secondly, the reference path that the quadcopter must follow is more aggressive than the one used to tune it. This means the quadcopter may not reach them in time. Finally this can also be explained by the inertia of the quadcopter and the tight constraints on the angles. The quadcopter cannot move indefinitely fast as the motors have limited power and the quadcopter has a finite mass. The plots showing the angles clearly show that the quadcopter is often doing the maximum possible to reach the desired reference but is limited by the tight constraints. (It is worth mentioning that these are due to the linearization of the drone model and a nonlinear drone would be able to reach greater angles and thus better performance.)

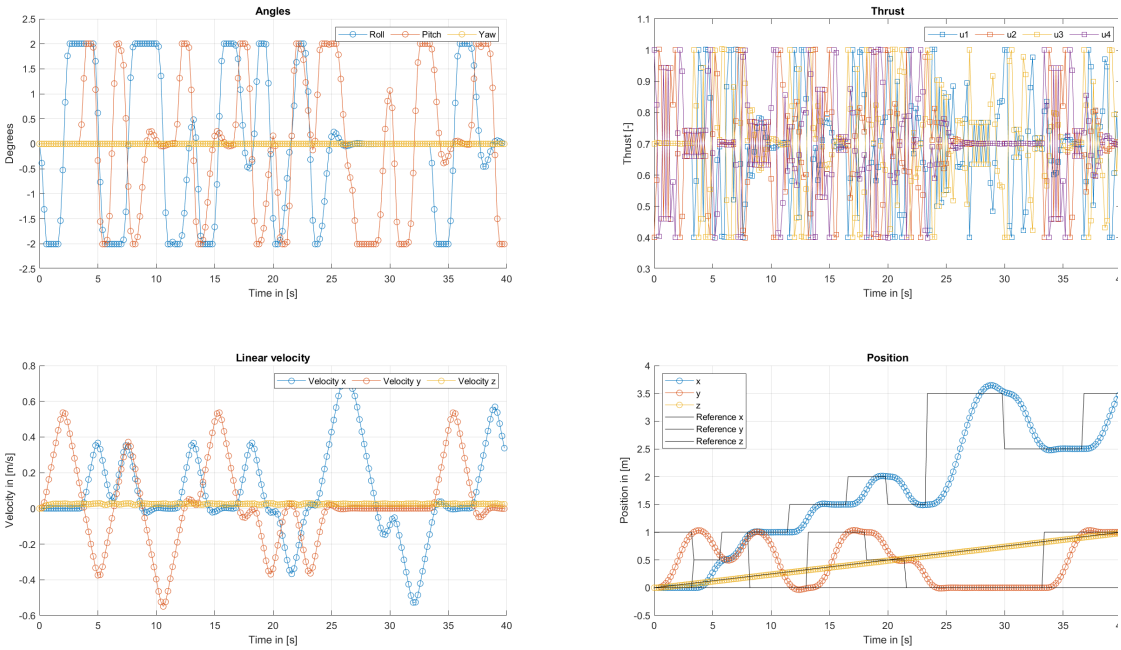

Figure 7: top left: Angles, top right: Thrust of All Motors, bottom left: Linear Velocities in x-, y- and z-Direction, bottom right: Position of Quadcopter and Reference

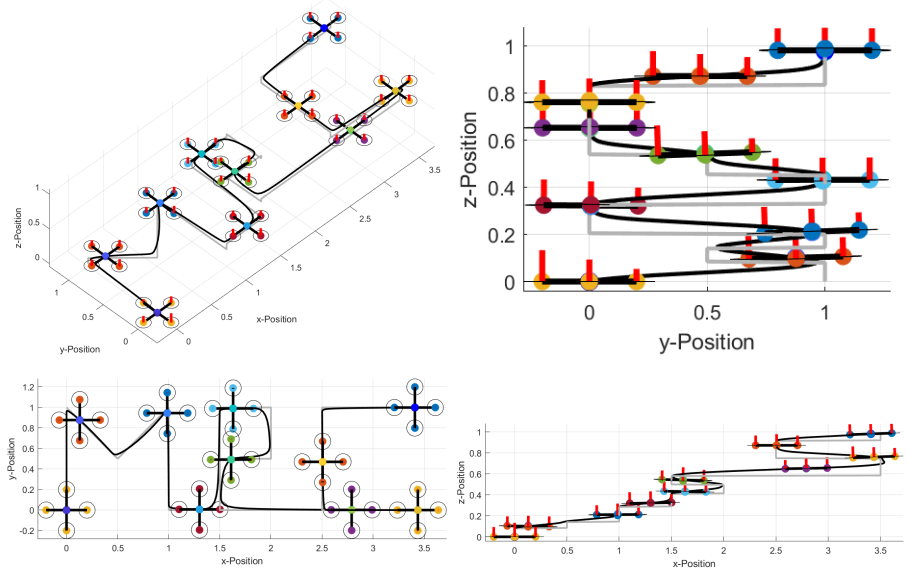

Figure 8: Path of Quadcopter, top left: 3D, top right: yz-Plane , bottom left: yx-Plane, bottom right: xz-Plane

6 Deliverable 5.1

In this chapter, a disturbance on the z-axis (for instance weight) is examined. Therefore offset-free tracking technique must be implemented. To do so, an estimator is needed to estimate the disturbance and compute the correct steady-state. The steady state is now computed by including the estimated disturbance:

```
%In setup_steady_state_target:
 con = [];
 G = [1;-1]; g = [0.3;0.2];
 Aex = [eye(2)-mpc.A, -mpc.B;
        mpc.C, zeros(1,1)];
 con = [con,Aex*[xs;us] == [mpc.B*d_est;ref]];
 con = [con,G*us≤g];
 obj = us'*us;
```

Before making an estimator we first checked that the system is observable, which means that the state of the system is measurable. The following line gives us linear independent eigenvectors, which means that we can indeed use an estimator:

```
Observability = obsv(sys_z.A, sys_z.C) = [0 1; 1 0]
```

The Obvervability-Matrix is observable if it has full rank. Furthermore, the following matrix must have full column rank. This is as well checked in the code.

$$\begin{bmatrix} A - I & B_d \\ C & C_d \end{bmatrix} \tag{7}$$

The new matrices of the system are composed as follows for the disturbance estimation:

$$x^+ = A * x + B * u + B_d * d \quad (B_d = B) \tag{8}$$

$$y = C * x + C_d * d \quad (C_d = 0) \tag{9}$$

$$\bar{A} = \begin{bmatrix} A & B \\ \mathbf{0} & I \end{bmatrix} \bar{B} = \begin{bmatrix} B \\ 0 \end{bmatrix} \bar{C} = \begin{bmatrix} C & C_d \end{bmatrix} \tag{10}$$

In order to extinguish the constant offset due to the disturbance we have set C_d to zero. B_d equals to B to account for the mass disturbance in the system. To find the estimator gain we use the pole placement technique. The poles chosen are small enough so that the estimator will converge rapidly to the real offset.

```
% In setup_estimator:
L = -place(A_bar',C_bar',[0.5,0.6,0.7])';    % slow poles
L = -place(A_bar',C_bar',[0.05,0.06,0.07])';    % fast poles
```

The estimator will be used to compute the estimated position of the quad-copter and the estimated disturbance. The estimated disturbance will change the system dynamics constraints while the estimated position will be used to update the state constraint.

The terminal constraint can no longer be used here as we cannot guarantee constraint satisfaction as long as the disturbance is not correctly estimated. We used the same costs as before:

```
%in setup_controller():
G = [1;-1]; g = [0.3;0.2];
H = [0,0];
h = 0;
Q = diag([0.5;10]); R = 0.1*eye(1);
for i = 1:N-1
    con = [con, mpc.A*x(:,i)+mpc.B*u(i)+mpc.B*d_est ==  ...
        x(:,i+1)]; % System dynamics
    con = [con, G*u(i) ≤ g]; % Input constraints
    obj = obj+(x(:,i)-xs)'*Q*(x(:,i)-xs)+(u(i)-us)'*R*(u(i)-us);
end
obj = obj+x(:,N)'*Q*x(:,N);
```

We actually tried different values for the estimator poles, the faster they are, the less it will fall at the beginning. Below we can observe the impact of having a different gains, specially on the z-position which is quite lower at the beginning with the slower poles.

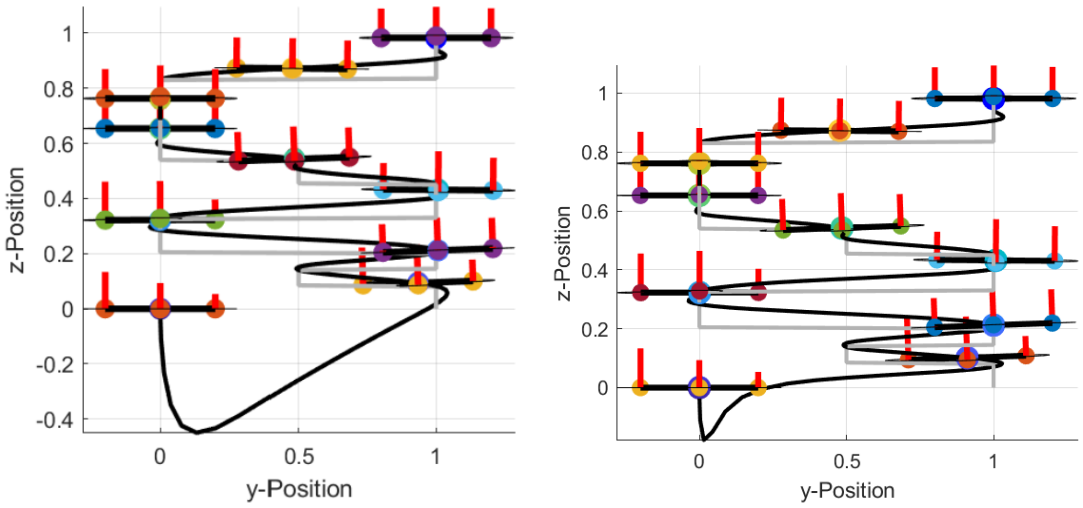

Figure 9: yz-plane with slow (left) and fast (right) poles

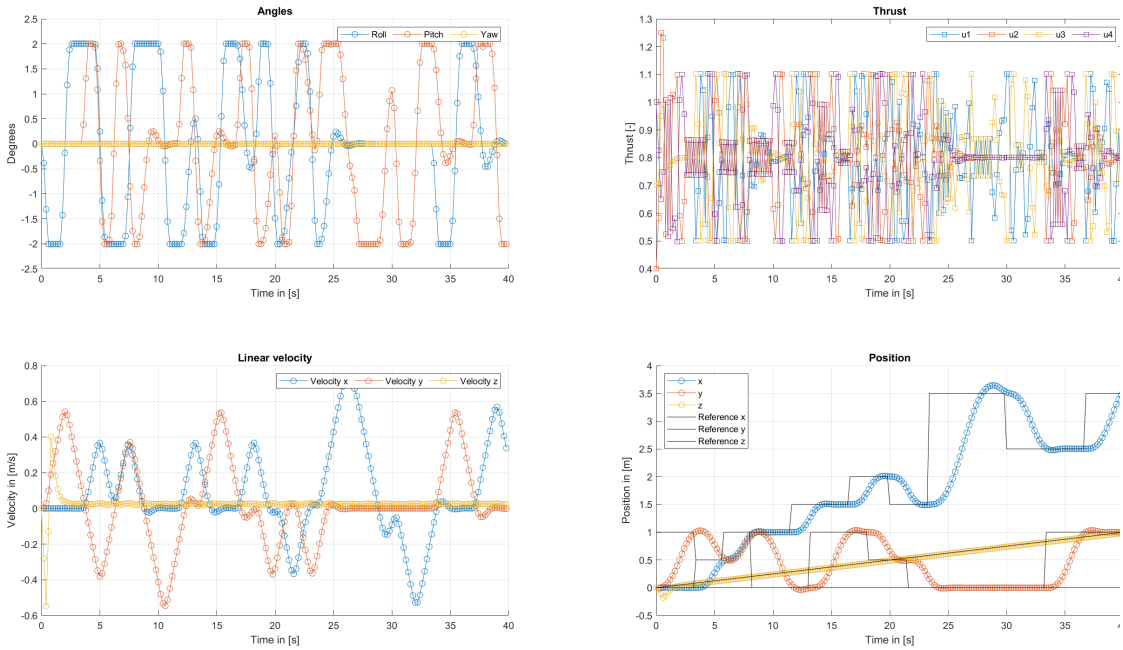

Figure 10: BIAS = -0.1 | top left: Angles, top right: Thrust of All Motors, bottom left: Linear Velocities in x-, y- and z-Direction, bottom right: Position of Quadcopter and Reference | fast poles

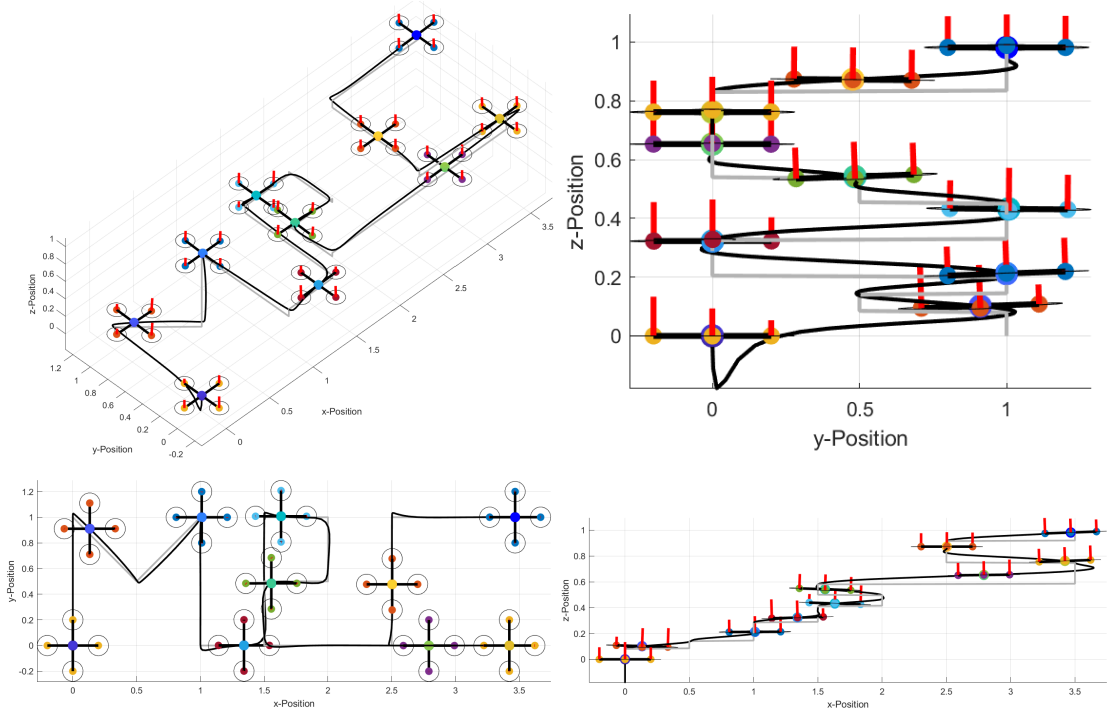

Figure 11: Path of Quadcopter with BIAS = -0.1 | top left: xz-Plane, top right: 3D, bottom left: yx-Plane, bottom right: yz-Plane | fast poles

7 Deliverable 6.1

In the final part, a nonlinear controller is designed for the quadcopter. The advantage of a nonlinear model is that it is a more accurate model of the dynamics of the system, leading to more realistic predicted trajectories. The nonlinear model and its associated controller also take the system as a whole and compute a control action for the entire system. Decoupling the system introduces small errors as for example the motion in the z direction will be influenced by the pitch and roll angles. By keeping the links between all the states, the performance can be improved. Finally, a nonlinear controller allows to relax the constraints on the pitch and roll angles as these constraints were due to the assumptions made during the model linearization. This allows the drone to move faster in x and y direction.

The design of the nonlinear controller is done in the following steps: First the nonlinear model is discretized as we want to apply inputs at specific times, determined by the sampling time. This discretization is done using a fourth order Runge-Kutta scheme that gives a discrete integration of the states between consecutive steps.

In the following, the constraints on the inputs and states are defined. The input constraints are now the global constraints on the four motor inputs. The state constraints stay the same and are only applied on pitch and roll. However, because the angles are less critical for the nonlinear model, these can be modified to be soft constrained (the constraints are no longer required for the linearization, but to ensure the drone will stay stable it is advisable to keep the attitude of the drone limited between certain boundaries). The constraints and costs are now defined as follows. The value of exceeding the constraints is represented by the slack variable E; the cost for the slack variable is reasonably high and determined by W.

```
1
2  for i=1:N
3   opti.subject_to(X(:,i+1) == F(X(:,i),U(:,i))); % Dynamics
4   opti.subject_to(H*X(:,i)≤h + ones(4,1)*E(i)); % State constraints
5   opti.subject_to(G*U(:,i)≤g); % Input constraints
6   opti.subject_to(E ≥ 0); % constraints for slack variable
7  end
8
9  for i=1:N
10      cost = cost+(X(:,i)-XREF)'*Q*(X(:,i)-XREF)+U(:,i)'*R*U(:,i)+ ...
11      E(i)'*W*E(i);
12      % Add soft constraint in optimization
13 end
```

Next, terminal constraints are considered. In general it is complicated to compute terminal constraints for nonlinear MPC problems, but in this specific case the terminal constraints computed for the linear model can be used. This can be understood by looking at the references applied to the drone. In steady state the drone will always be horizontal with the engines providing thrust to counteract gravity, else it would move in some direction. This is exactly the configuration assumed

in the linear system, meaning that a large enough horizon will ensure that the terminal constraint is applied to a state very close to a horizontal steady state position.

The last step in the design of the nonlinear controller is the tuning. As mentioned above the horizon must be large enough to allow the use of terminal constraints. The horizon is set to 30, the same as for the linear controllers. This value was determined experimentally, by trying different horizons and comparing the performance of the quadcopter. The tuning of the state and input weights is done in the same way as for the linear controllers, by considering the physical significance. The input is again chosen quite small, allowing the quadcopter to quickly reach the desired reference. Again the position has the highest cost. It is worth noticing that here the z-coordinate has a higher weight than the others. This ensures the drone doesn't drop too much when moving in a certain direction. The drone for instance drops at approximately 24 seconds while accelerating in x-direction (see Figure 12). This happens because the thrust is no longer vertical, but shifted due to the pitch angle. The cost for the slack variable E is even higher than the state cost of z-coodrinate. We have seen that with lower costs the quadcopter not always tries to follow the path properly and accepts an offset of the path. To diminish this behaviour, we have raised the cost for exceeding the constraints (see Equation 13).

$$\begin{aligned} Q_{angularvelocity} &= 0.1 \cdot eye(3); \\ Q_{angle} &= 1 \cdot eye(3); \\ Q_{velocity} &= eye(3); \\ Q_{position} &= diag(10, 10, 500); \end{aligned} \tag{11}$$

$$R = 0.1 \cdot eye(4) \tag{12}$$

$$W = 1000 \tag{13}$$

The states values and path over time of the soft constrained NMPC problem are shown below.

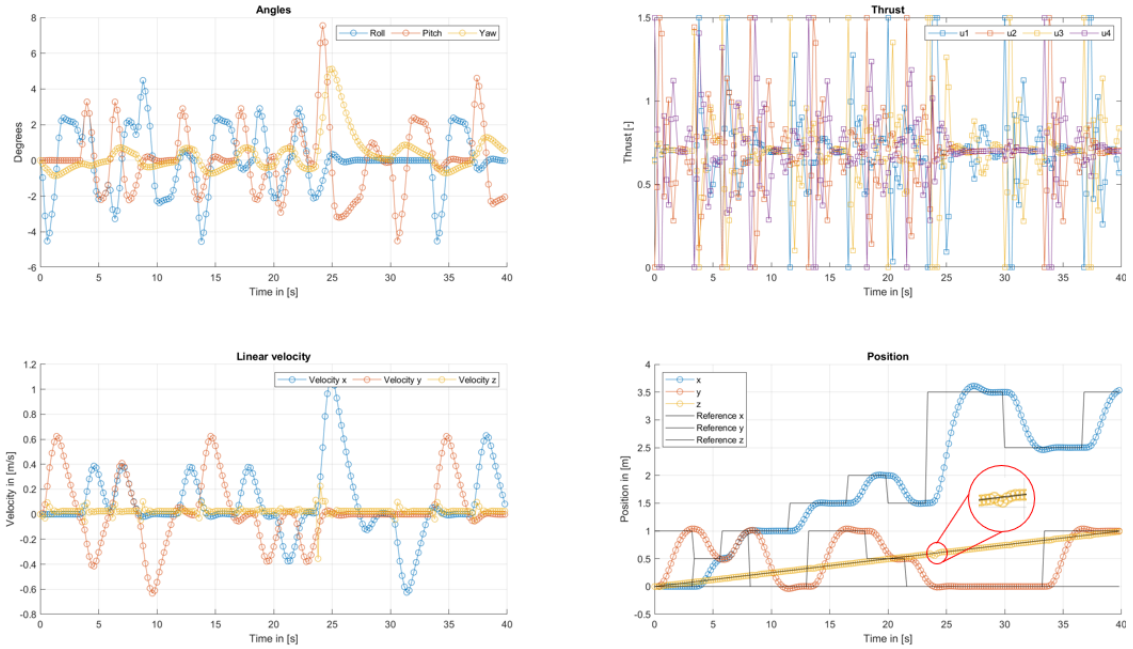

Figure 12: top left: Angles, top right: Thrust of All Motors, bottom left: Linear Velocities in x-, y- and z-Direction, bottom right: Position of Quadcopter and Reference, red circle: Drop of Height of Quadcopter due to fast Acceleration

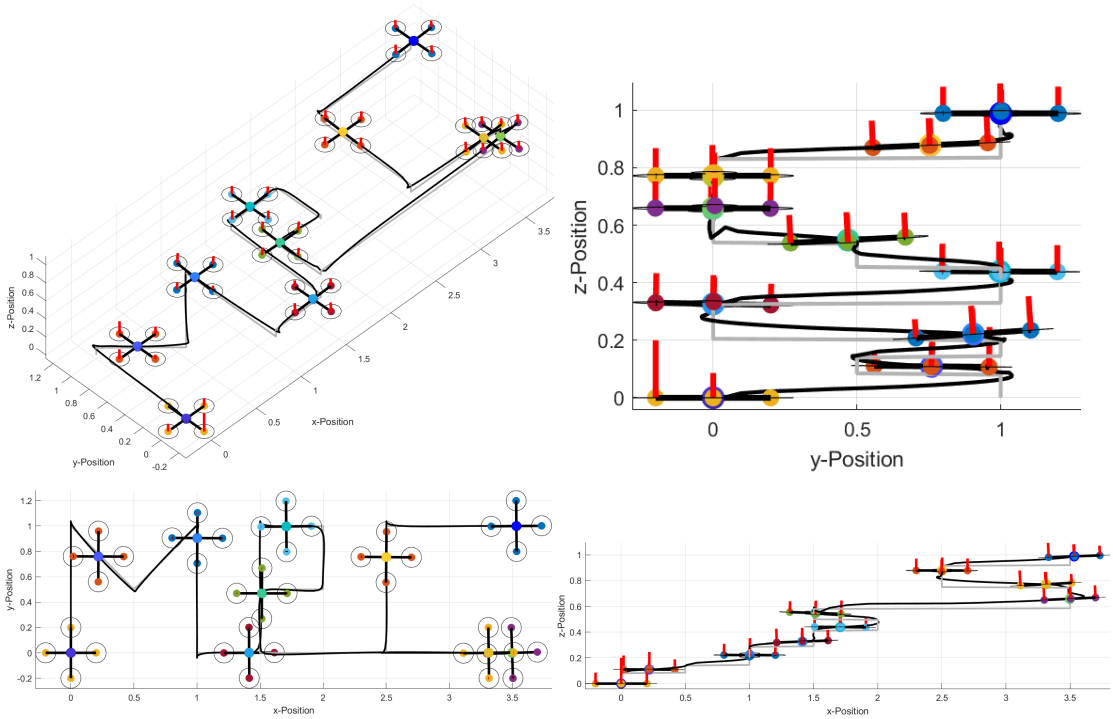

Figure 13: Path of Quadcopter, top left: 3D, top right: yz-Plane , bottom left: yx-Plane, bottom right: xz-Plane

Figure 14 and 15 show the response when the constraints are kept identical (no soft constraints) to the ones for the linear system in comparison with the response of the soft constrained system. This clearly shows the added value of softening the constraints for the nonlinear case. It can be seen that the maximal angle of 2 is exceeded several times, so that the quadcopter can gain more velocity. In Figure 15 it can be see that the quadcopter with the soft constraints is much faster than the quadcopter with normal constraints. Considering the x-position, it can be seen that the soft constrained quadcopter is especially faster for long distance movements. The velocity gain is not outstandingly big in the first 22 seconds of the simulation. Furthermore, for the y-position the soft constrained quadcopter is faster in the first 10 seconds. This lead to the assumption that the soft constrained quadcopter has advantages for long distance movements and fast changes of direction. In general, it creates an impression that the soft constrained quadcopter is more agile.

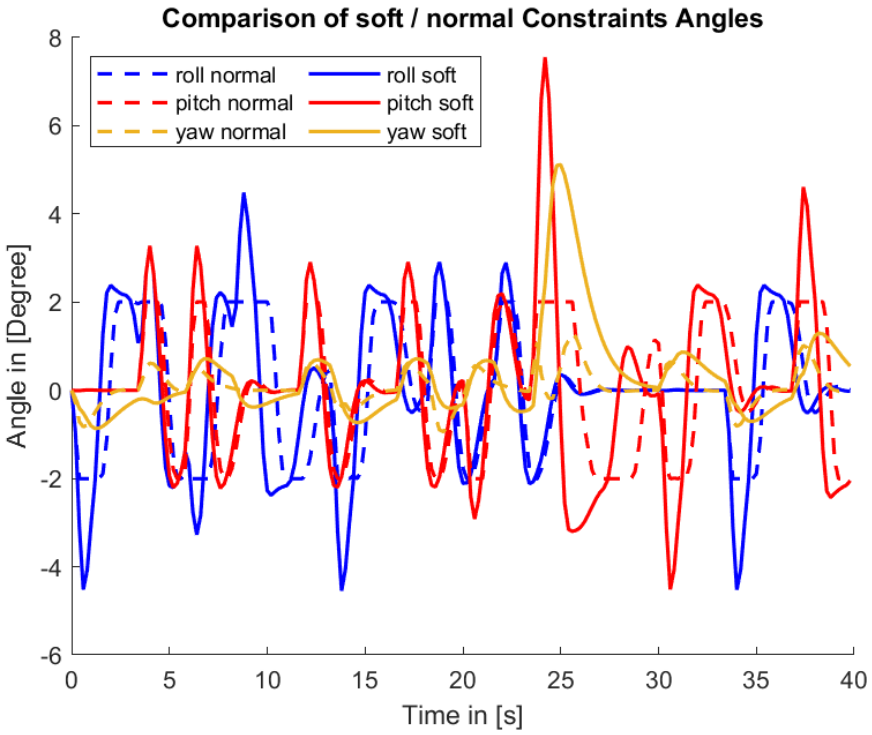

Figure 14: Comparison of the Anlges of the Quadcopter for normal and soft Constraints

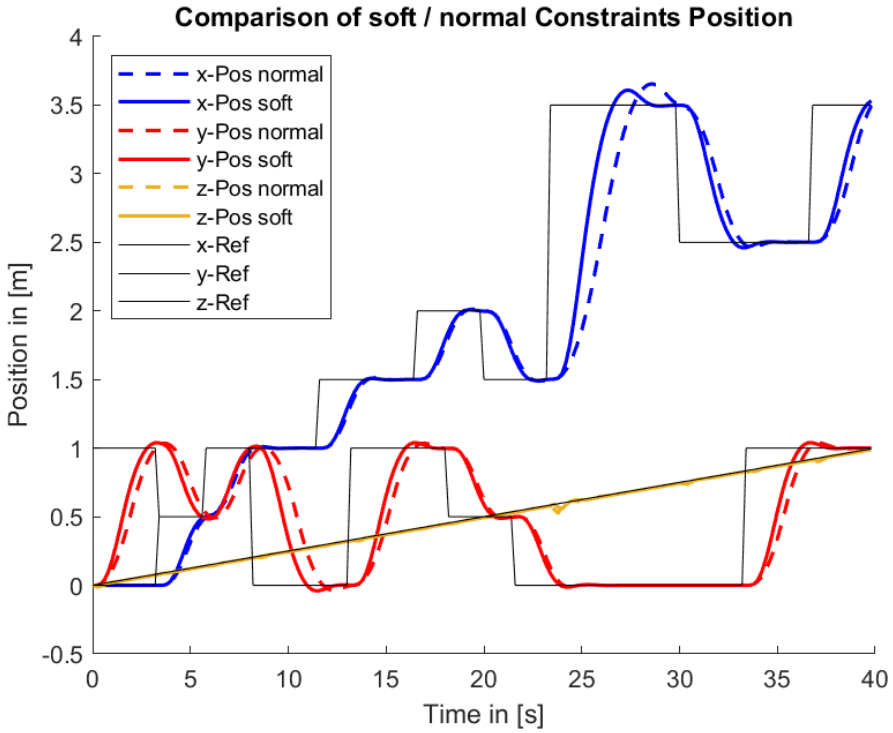

Figure 15: Comparison of the Position of the Quadcopter for soft and normal Constraints

For the non linear MPC-Problem it is as well possible to not implement a terminal set with terminal costs. In the following, we have created a comparison between a soft constrained quadcopter with terminal set / constraints and without (see Figure 16, 17). Both figures show that the differences between the two quadcopters are reasonably small. This might be due to the fact that the horizon (30) is relatively large. In Figure 16 it gets visible that only the yaw angles differ from each other. At a first glance at Figure 17, no differences on the position of the quadcopters can be detected. However, if we zoom in slightly, differences appear. The quadcopter with terminal set has a little more overshoot and is a little faster. In conclusion it can be said, that the two quadcopter have more or less the same behaviour. The differences are reasonably small.

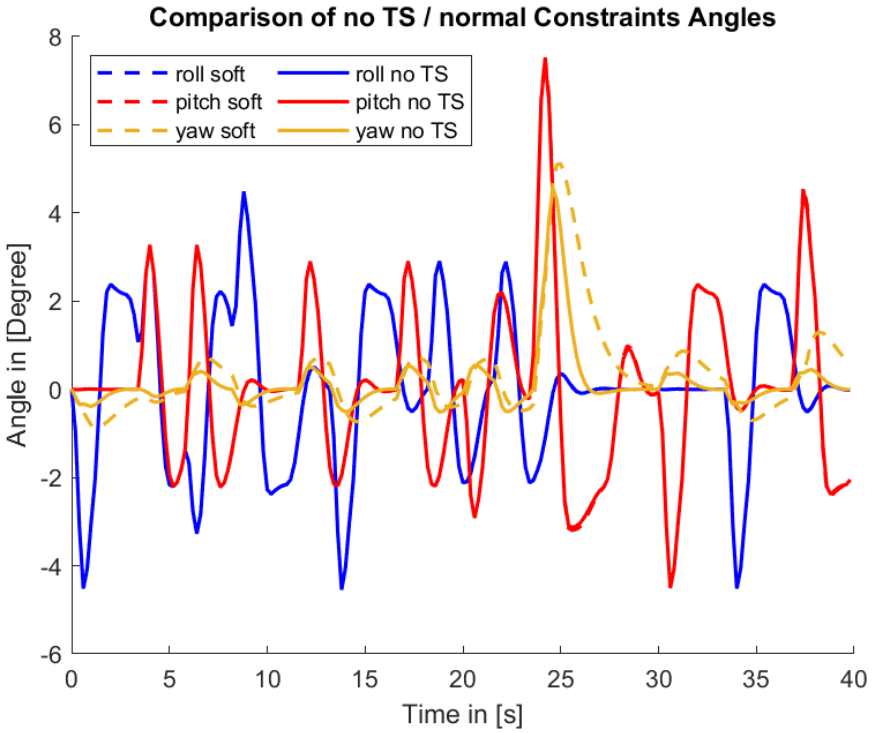

Figure 16: Comparison of the Angles of the Quadcopter with a Terminal Set and without a Terminal Set

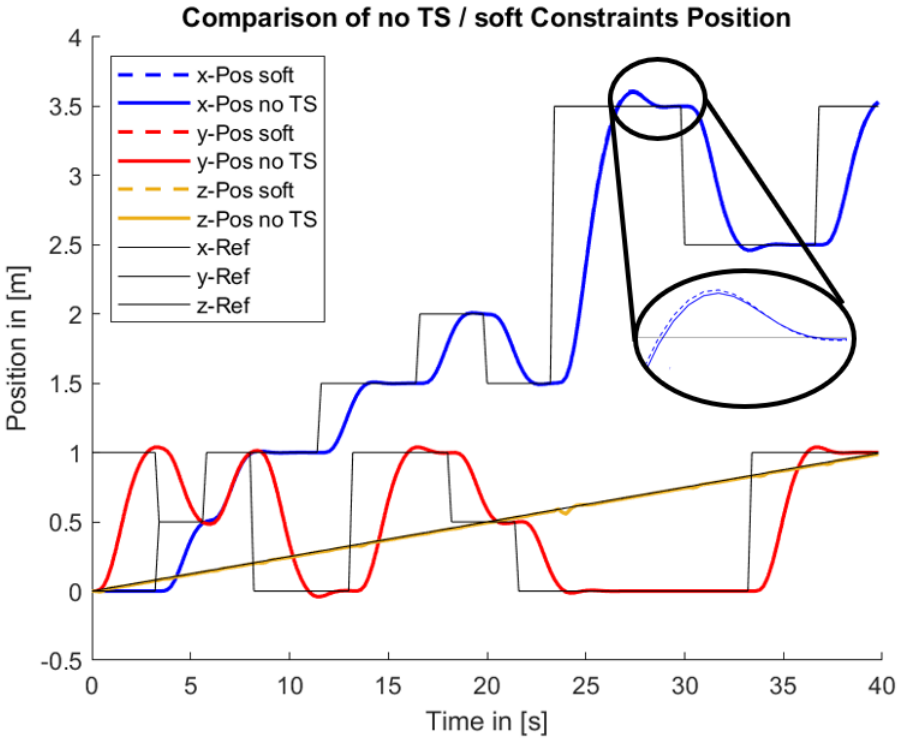

Figure 17: Comparison of the Position of the Quadcopter with a Terminal Set and without a Terminal Set