MODEL PREDICTIVE CONTROL

ME-425

GROUP 66

MINI-PROJECT REPORT

February 29, 2020

CUI Mingbo (293330)

ZHOU Xiao (294916)

WU Yi-Shiun (294105)

Contents

1 DELIVERABLE 2.1

- **Explain why this occurs, and whether this would occur at other steady-state conditions.**

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} = A\boldsymbol{x} + BT^{-1}\boldsymbol{v} \tag{1}$$

Before we do the transform, the column vector of matrix B is correlated, which means that four control signals will influence each other. The initial **B** relates 4 motor thrust with multiple state values, and there are inter-correlation among the state values, making them not decoupled. The mapping from **u** to new control input **v** by matrix **T** realizes the transformation from control of motor thrusts to force and moments. According to the equation above, the transformation leads to a new matrix $B_v = B \cdot T^{-1}$. When keep the matrices **A**, **C** and **D** unchanged, the transformation can correlate one moment or forth to single state variable. For example, pitch moment M_β will only affect $\dot{\beta}$ but not other directions.

To analyze mathematically, non-zero values in rolls of **B** correspond to those in **T**. The transformation calculates a new matrix through a permutation matrix with scaling. For example

$$T = H \cdot B_{4\times4} \tag{2}$$

$$H^{-1} = B_{4\times4} \cdot T^{-1} \tag{3}$$

$$H = \begin{bmatrix} 0 & 0 & 0 & a \\ b & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & d & 0 \end{bmatrix} \tag{4}$$

Thus, the new matrix B_i is the permutation of a diagonal matrix without zero values in rolls. Therefore, after transform, the control signals are decoupled. In other word, the column vector of the matrix B is independent from each other.

This may occur where there are sufficiently large number of steady state conditions upon which the system can be linearized.

2 DELIVERABLE 3.1

- **Explanation of design procedure that ensures recursive constraint satisfaction**
  Constraints for the output:

$$|\alpha| \leq 2^\circ = 0.035rad \tag{5}$$
$$|\beta| \leq 2^\circ = 0.035rad \tag{6}$$

  Constraints for the input:

$$0 \leq T^{-1}\boldsymbol{v} + u_s \leq 1.5 \tag{7}$$

  which can be pre-computed and reformulated as:

$$-0.3 \leq M_\alpha \leq 0.3 \tag{8}$$
$$-0.3 \leq M_\beta \leq 0.3 \tag{9}$$
$$-0.2 \leq M_\gamma \leq 0.2 \tag{10}$$
$$-0.2 \leq F \leq 0.3 \tag{11}$$

  For state x, there are output and input constraints, we denote the transform matrix as F_x and the control matrix as F_u.

  Then we can design the constraints of our controller upon that. For example, in the control design of x ($\dot{\beta}$, β, $\dot{x}$ and x) after split into 4 independent control systems, the constraints of x correspond the constraint in pitch. That is the second element in state x is in the range [-0.035, 0.035]. Also the control in u_x that represents pitch should be limited into [-0.3, 0.3].

  To ensure the recursive constraint satisfaction with the horizon, the state and input should satisfy the following constraint recursively when i increases from 0 to N-1:

$$x_{i+1} = Ax_i + Bu_i \tag{12}$$

$$F_x = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, f_x = \begin{bmatrix} 0.035 \\ 0.035 \end{bmatrix} \tag{13}$$

$$F_x \cdot x_i \leq f_x \tag{14}$$

$$F_{u,x} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, f_{u,x} = \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix} \tag{15}$$

$$F_{u,x} \cdot u_i \leq f_{u,x} \tag{16}$$

The objective cost function $J(x)$ will be accumulated with the increase of i:

$$J(x) = J(x) + x_i^{\mathrm{T}} Q x_i + u_i^{\mathrm{T}} R u_i \tag{17}$$

When i increases to N, we will add the terminal constraint and terminal cost:

$$F_f \cdot x_N \leq f_f \tag{18}$$

$$J(x) = J(x) + x_N^{\mathrm{T}} Q_f x_N \tag{19}$$

Here the F_f, Q_f and f_f are obtained from the maximum invariant set for terminal invariant controller, and this will be introduced in later part of Deliverable 3.1.

The control design for **y**, **z** and **yaw** are similar. The only difference in the desgin procedure is the constraint definition. In system **y**, the roll state and moment should satisfy the constraints stated in the beginning of the section, so that $F_y, f_y, F_{u,y}, f_{u,y}$ are defined as:

$$F_y = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, f_y = \begin{bmatrix} 0.035 \\ 0.035 \end{bmatrix} \tag{20}$$

$$F_{u,y} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, f_{u,y} = \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix} \tag{21}$$

For system **z** and **yaw**, there are not specific constraints for state according to the project requirement, and constraints for thrust and moment are defined as follows:

$$F_{u,z} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, f_{u,z} = \begin{bmatrix} 0.3 \\ 0.2 \end{bmatrix} \tag{22}$$

$$F_{u,yaw} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, f_{u,yaw} = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix} \tag{23}$$

- **Explanation of choice of tuning parameters. (e.g., Q, R, N, terminal components)**
  N will influence the stability, feasibility and invariance. With small N, the solution is infeasible. By increasing N to 10, we could get a feasible solution but the settling time will be exceed 8 seconds in x and y systems. To fulfill the requirements for settling time limit, finally we set N to 20 (for all 4 systems); Q, R is relevant to the cost computation, here a identity matrix for Q and 1 for R is good enough to our model (for 4 systems). Initially we tried to set Q to diag(1,5,1,5) in **x** and **y** system, and diag(1,5) in **z** and **yaw** system, since intuitively we want to put large weight on position instead of velocity (e.g. pitch and x in **x** system) , and that's the state we care about most. However, the performance of different sets of tuned parameters don't show a large difference. In the later deliverables, we will mainly use I for the 2 matrices. The terminal Q_f in each system is obtained by LQR system by dlqr function in Matlab, which will be derived in later part of terminal invariant set.

- **Plot of terminal invariant set for each of the dimensions, and explanation of how they were designed and tuning parameters used**
  To calculate the maximum terminal invariant set, the main idea is intersect the set with its pre-set until convergence. The initial terminal set is designed as a polytope that satisfies:

$$\begin{bmatrix} F_x \\ F_u K \end{bmatrix} \cdot x \leq \begin{bmatrix} f_x \\ f_u \end{bmatrix} \tag{24}$$

where K and Q_f are the control gain from LQR system ($u = Kx$) according to dlqr function in Matlab. K has to reverse its sign. In system **z** and **yaw**, we can drop the F_x and f_x since there are not specified constraints on state values.

In each iteration, X_f can be represented as a polytope:

$$T \cdot x \leq t \tag{25}$$

Then we can calculate the pre-set of X_f in each iteration according to T and t in polytope:

$$T(A + BK)x \leq t \tag{26}$$

The new polytope can represent pre(X_f), and so the new set X_f in each iteration is:

$$X_f = X_f \cap pre(X_f) \tag{27}$$

The iteration ends when $X_f = pre(X_f)$, and the obtained X_f is the terminal invariant set. F_f and f_f will then represent the polytope of computed terminal invariant set, which is used in the constraint design procedure in previous part.

The plot of terminal invariant set for **x**, **y**, **z**, and **yaw** are illustrated in Figure 1, Figure 2, Figure 3, and Figure 4 respectively. From the figures we could observe that our system could converge to the steady state quickly and smoothly.

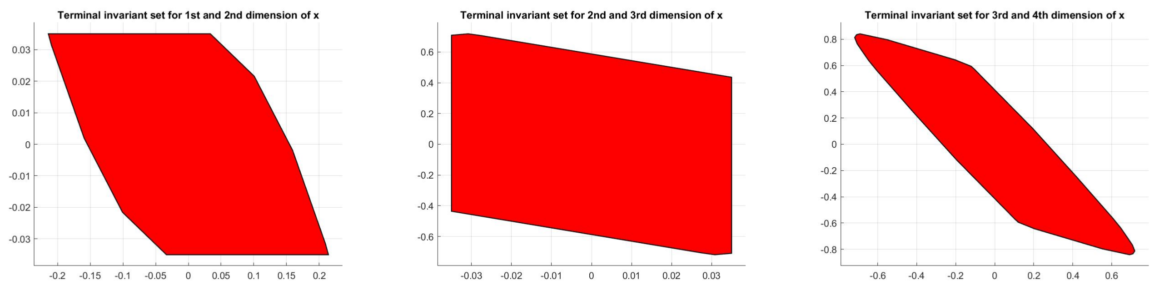


Figure 1: Projection plot of terminal invariant set for dimension **x**

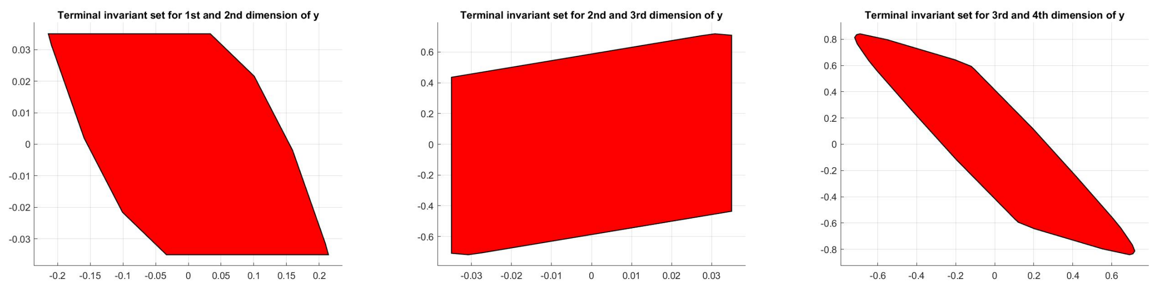


Figure 2: Projection plot of terminal invariant set for dimension **y**

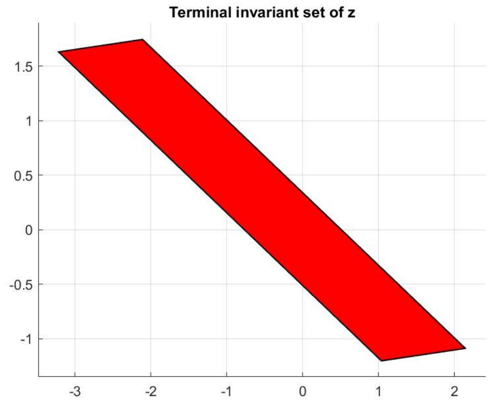


Figure 3: Plot of terminal invariant set for dimension **z**

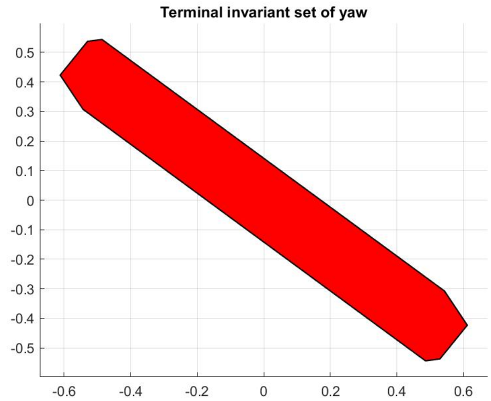


Figure 4: Plot of terminal invariant set for dimension **yaw**

- **Plot for each dimension starting stationary at two meters from the origin (for x, y and z) or stationary at 45 degrees for yaw**
  The plots for system **x**, **y**, **z** and **yaw** are illustrated in Figure 5, Figure 6, Figure 7, and Figure 8 respectively. In addition, the settling time is around 8 seconds when starting stationary at two meters from the origin (for **x**, **y** and **z**) or stationary at **45 degrees** for **yaw**, which fulfills the requirement for deliverable 3.1.

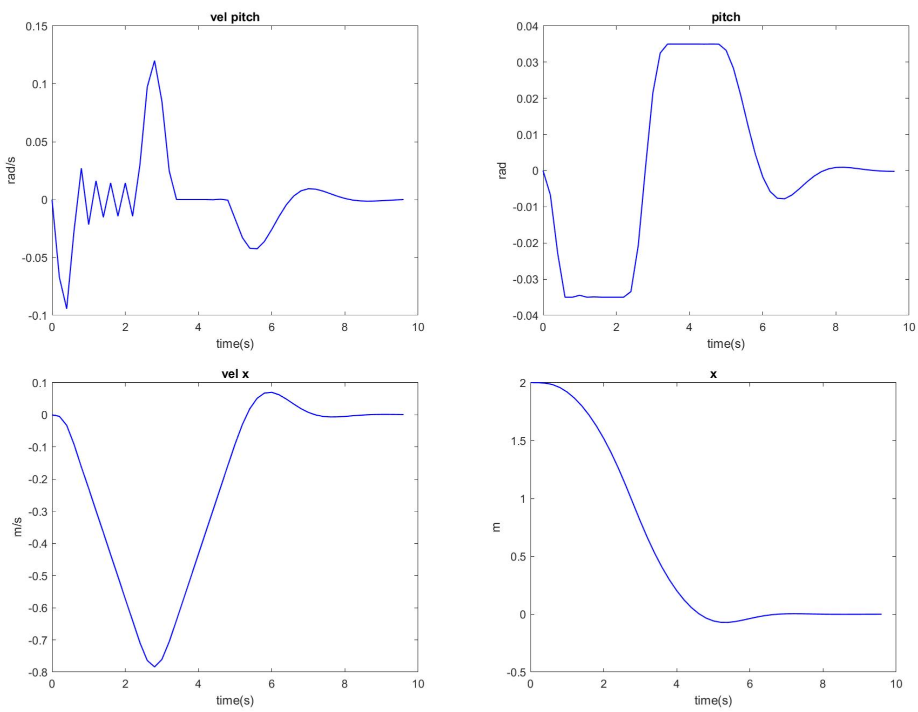


Figure 5: MPC regulating controller for system x

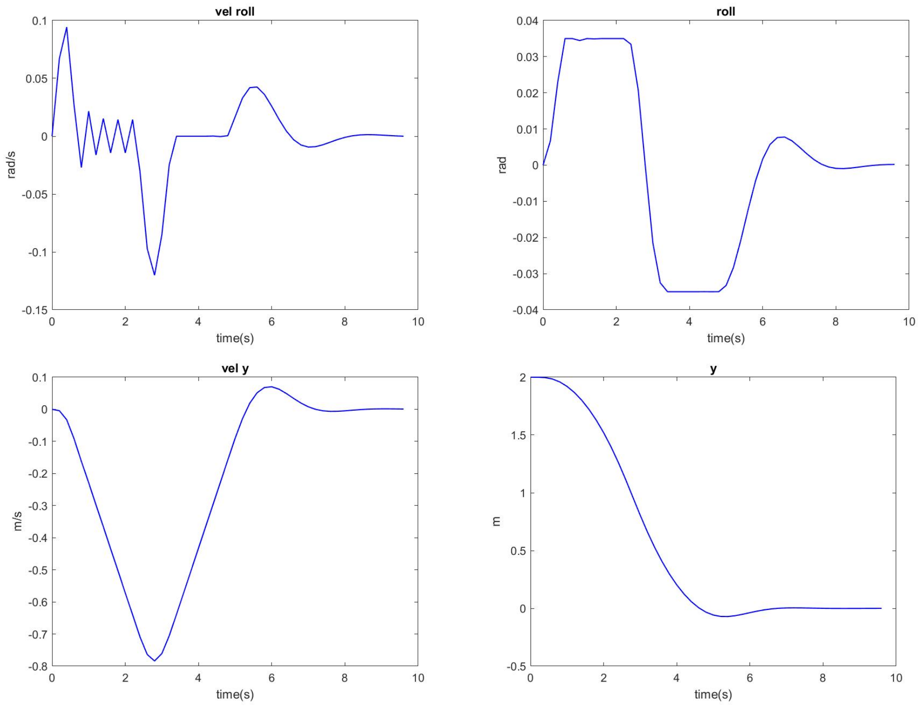


Figure 6: MPC regulating controller for system y

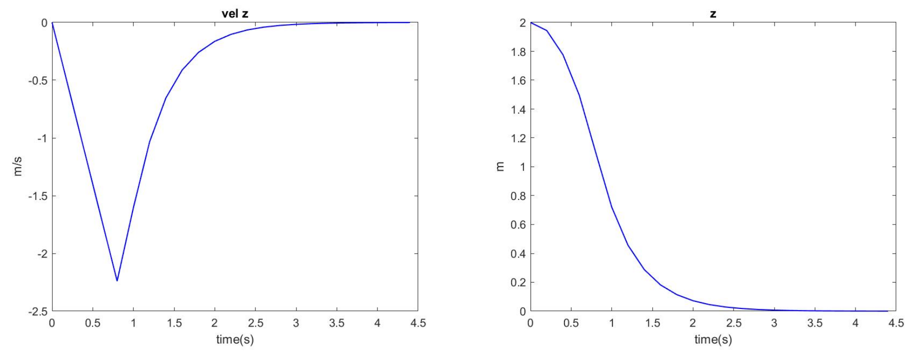


Figure 7: MPC regulating controller for system z

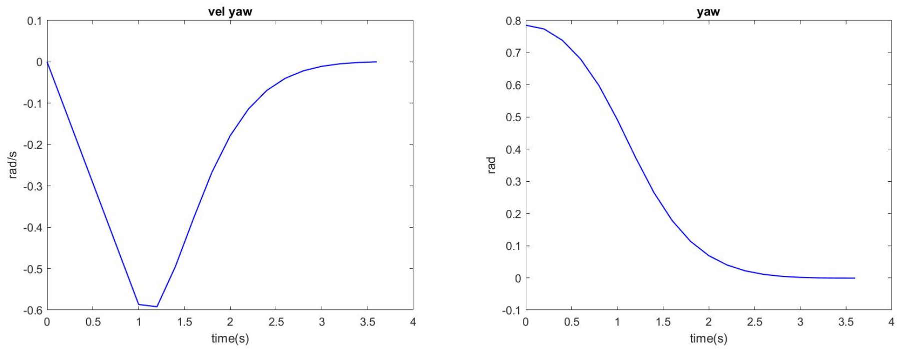


Figure 8: MPC regulating controller for system yaw

3 Deliverable 3.2

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

  The main idea in this design procedure is to introduce and calculate the expected steady state x_s and input u_s according to given reference for tracking. Also, we will take the control in x system for example. The constraint for x_s and u_s are as follows:

$$F_x \cdot x_s \le f_x \tag{28}$$

$$F_u \cdot u_s \le f_u \tag{29}$$

  In system **z** and **yaw**, we can drop the F_x and f_x since there are not specified constraints on state values. Also, the value of x_s and u_s should satisfy the equation:

$$x_s = Ax_s + Bu_s \tag{30}$$

$$r = Cx_s + Du_s \tag{31}$$

  where r is the tracking reference in x direction.

  Then the objective function $V(x_s, u_s)$ is defined in aim to minimize input:

$$V(x_s, u_s) = u_s^{\mathrm{T}} u_s \tag{32}$$

  It should be noted that if the reference cannot be reached, the objective function $V(x_s, u_s)$ should try to minimize the difference between output and reference, which is:

$$V(x_s, u_s) = (r - Cx_s - Du_s)^{\mathrm{T}}(r - Cx_s - Du_s) \tag{33}$$

In our case, we use the first objective function and acquire the expected x_s, u_s by solving the function based on constraints by gurobi solver.

After that, we will change a bit of cost function. In the recursion of i from 0 to N-1, the stage cost will be accumulated as follows:

$$J(x) = J(x) + (x_i - x_s)^{\mathrm{T}} Q (x_i - x_s) + (u_i - u_s)^{\mathrm{T}} R (u_i - u_s) \tag{34}$$

Similarly, the terminal cost will be defined as:

$$J(x) = J(x) + (x_N - x_s)^{\mathrm{T}} Q (x_N - x_s) \tag{35}$$

The constraints of x_i and u_i and tuning parameters including **N**,**Q**,**R** are the same as derived in Deliverable 3.1. Also, the design of **y**, **z** and **yaw** system are almost the same. In this section, we drop the terminal set (especially in x system), otherwise the quadcopter will not follow the latter part of the reference trajectory.

- **Plot for each dimension of the system starting at the origin and tracking a reference to -2 meters from the origin (for x, y and z) and to 45 degrees for yaw**
  The plots for system **x**, **y**, **z** and **yaw** are illustrated in Figure 9, Figure 10, Figure 11, and Figure 12 respectively. From the figures we could observe that our system could track to the reference quickly and smoothly.

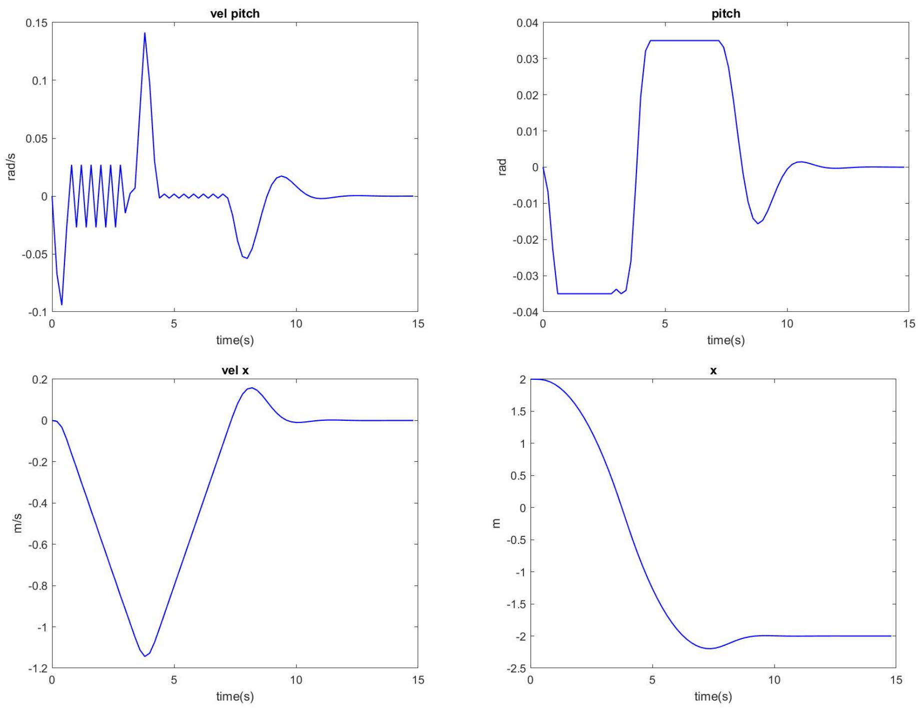


Figure 9: MPC regulating controller for system x

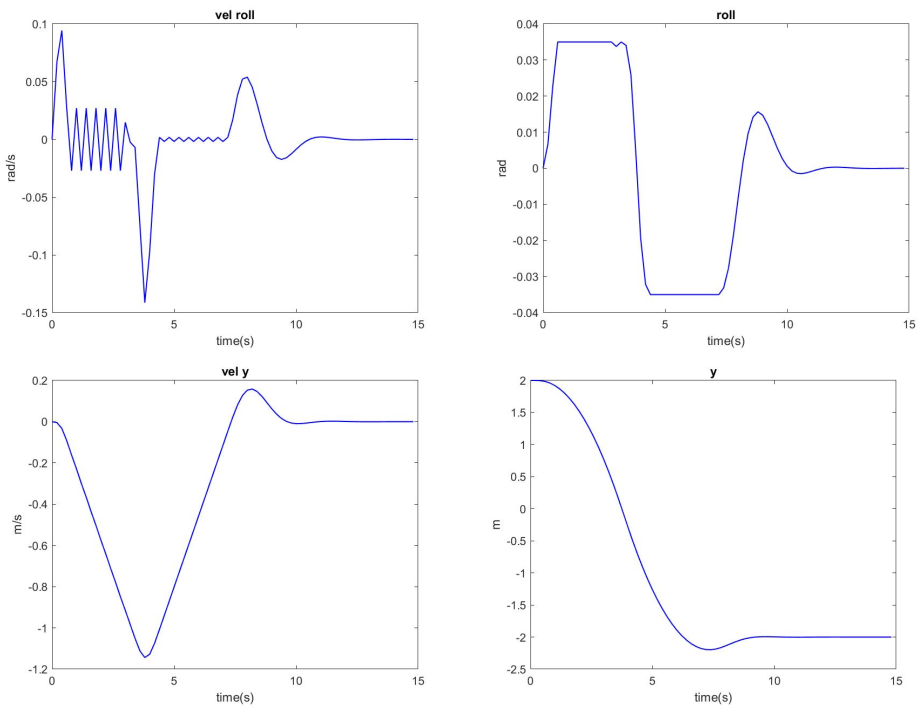


Figure 10: MPC regulating controller for system y

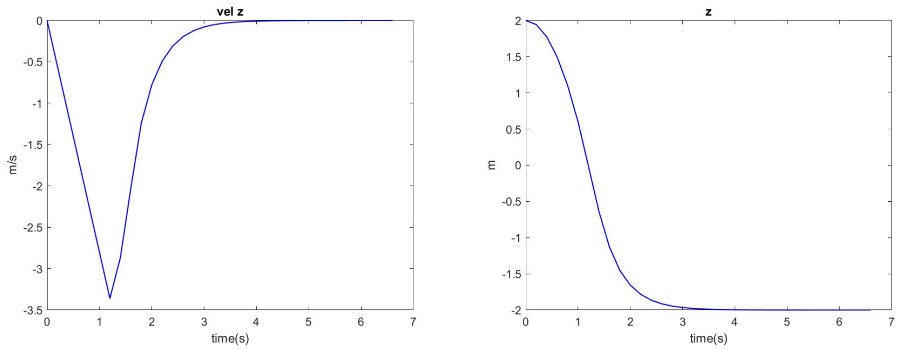


Figure 11: MPC regulating controller for system z

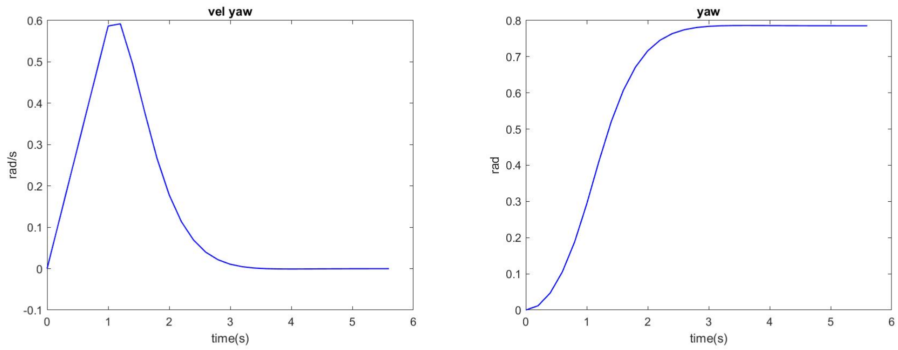
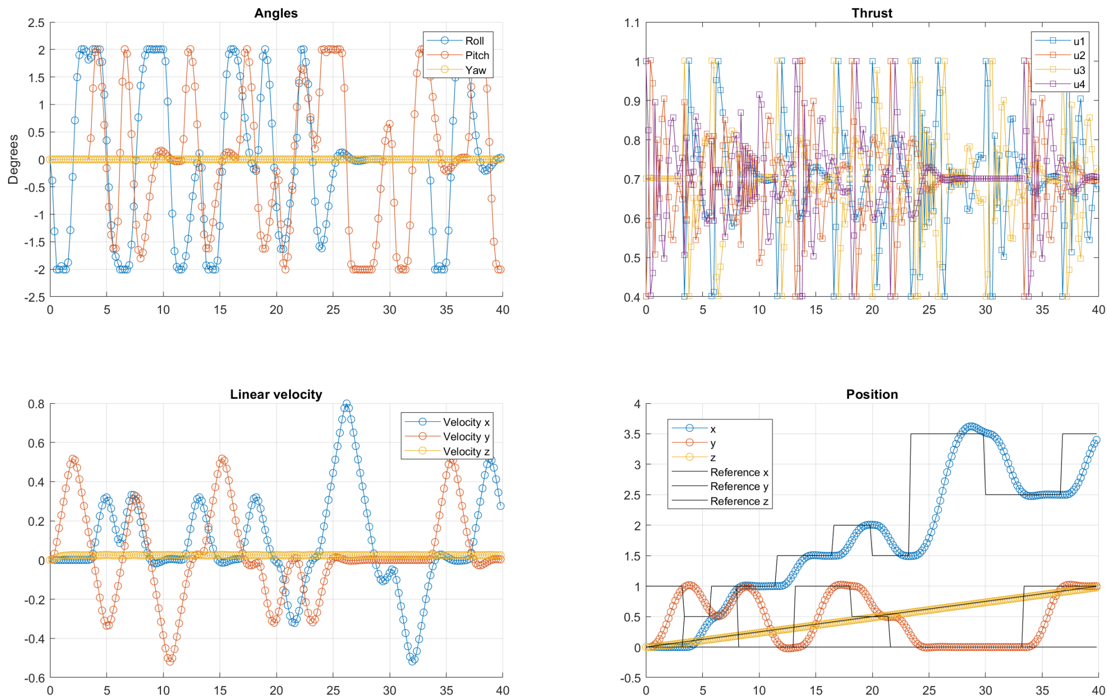


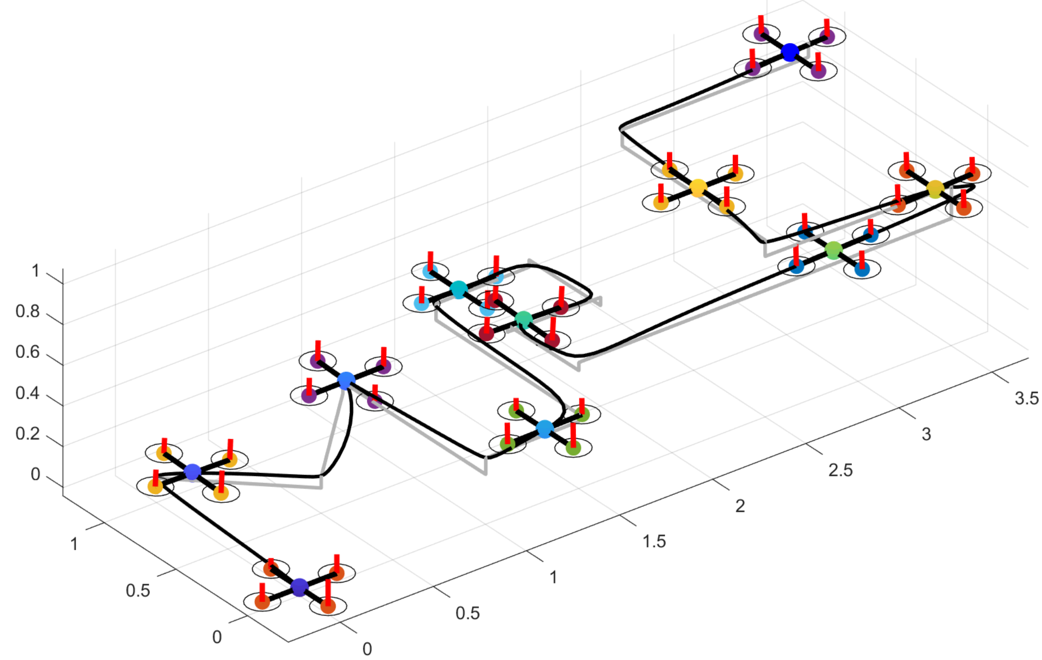
Figure 12: MPC regulating controller for system yaw

4 DELIVERABLE 4.1

- **A plot of your controllers successfully tracking the path using quad.plot(sim)**
  Result is shown in Figure 13. Our model could enable the quad-copter follow the pre-defined trajectory perfectly.



(a) Plots for all states



(b) Illustration of trajectory

Figure 13: Simulation results without disturbance

5 DELIVERABLE 5.1

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

  The notation of system dynamics are the same with previous answers. Since state and disturbances are unknown at time zero, we need to design an observer **L** to estimate them. The matrix **L** could be computed by:

$$L = -place(\hat{A}, \hat{C}^T, F)^T \tag{36}$$

where the function place computes a state-feedback matrix **L** such that the eigenvalues of $A - BL$ are those specified in the vector **F**. In our case, the above mentioned variables are defined here:

$$F = [0.5, 0.6, 0.7] \tag{37}$$

$$\hat{A} = \begin{bmatrix} A & B \\ 0_{MxN} & I_{MxM} \end{bmatrix} \tag{38}$$

$$\hat{B} = \begin{bmatrix} B & 0_{MxM} \end{bmatrix} \tag{39}$$

$$\hat{C} = \begin{bmatrix} C & 0_{1xM} \end{bmatrix} \tag{40}$$

After defining the system, we choose to use YALMIP to compute a steady state for the system which minimizes u^2.

The other part of controller design in system **z** is very similar to that in deliverable 3. The only difference is to consider the disturbance estimate d_{est} when computing next state x_{i+1} and steady state x_s, which can be represented as constraints in Matlab solver.The new state function is defined as:

$$x_{i+1} = Ax_i + Bu_i + Bd_{est} \tag{41}$$

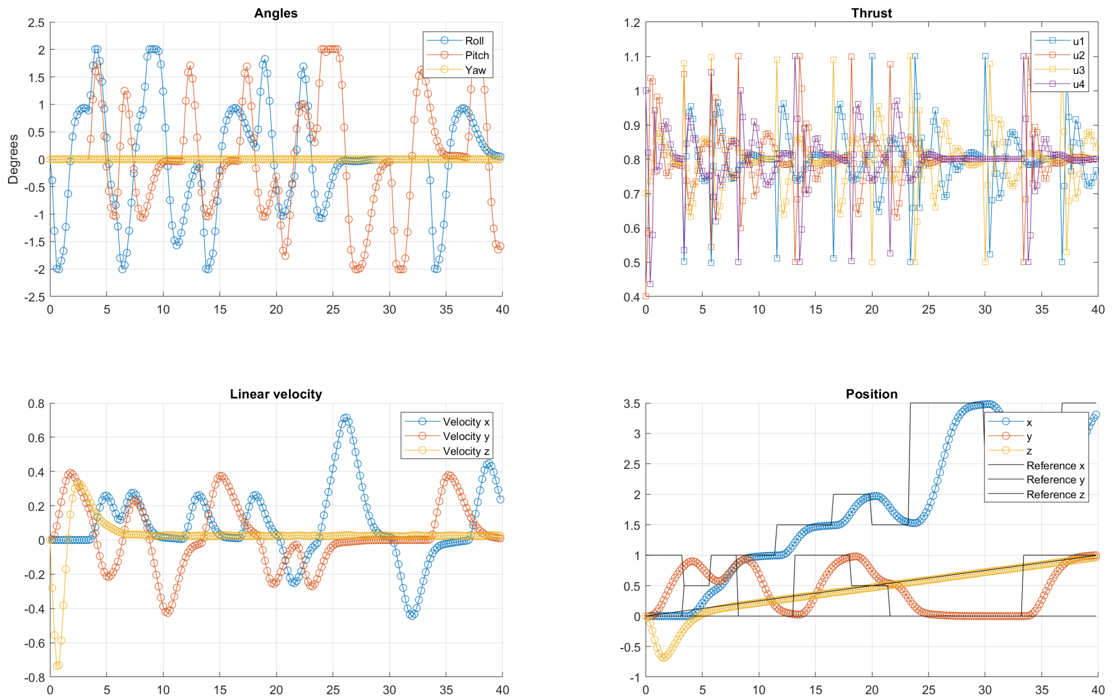
The calculation of steady state will be modified to:

$$x_s = Ax_s + Bu_s + Bd_{est} \tag{42}$$

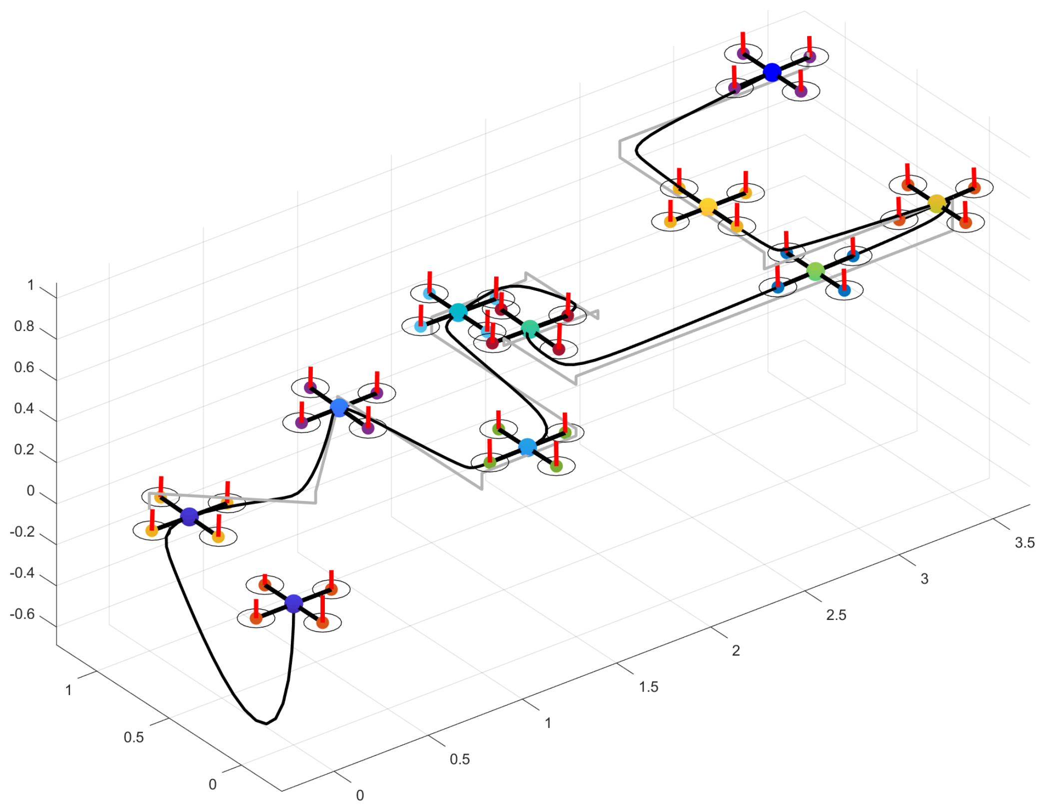
$$r = Cx_s + Du_s \tag{43}$$

Choice of Tuning parameters: There are many values for our model, we choose a set for the horizon and stage costs: **N** = 10, **Q** = I, **R** = 1, and **F** = [0.5,0.6,0.7] which ensures acceptable feasibility and stability in z system. It is worth noting here that **F** with a small norm will speed up the estimation process, but may increase the initial overshoot of the disturbance estimate.

- **Plot showing that the z-controller achieves offset-free tracking**
  Result is shown in Figure 14. It can be observed that despite the violation in the initial time period, the trajectory of **z** can then track the reference in an offset-free fashion.



(a) Plots for all states when injecting disturbance



(b) Illustration of trajectory

Figure 14: Simulation results with disturbance

6 DELIVERABLE 6.1

- **Explanation of your design procedure and choice of tuning parameters**
  Different from the linear MPC, NMPC takes as input a full state **X** and an input control vector **U**. To do the integration, we employ the RK4 as our integration approximation function $f_{discrete}$ with horizon length **N** = 10, and a sample period of $h = 0.22$. Given we are in step k, we update the state **X** for step $k+1$:

$$X(k+1) = f_{discrete}(X(k), U(k)) \tag{44}$$

The constraints for the state here will correspond to the constraint in roll and pitch recursively:

$$-0.035 < X_{i,4} < 0.035, -0.035 < X_{i,5} < 0.035 \tag{45}$$

The recursive constraint for control input is defined as:

$$\begin{bmatrix} -1.5 \\ -1.5 \\ -1.5 \\ -1.5 \end{bmatrix} < U_i < \begin{bmatrix} 1.5 \\ 1.5 \\ 1.5 \\ 1.5 \end{bmatrix} \tag{46}$$

Also, we define steady state and input to implement tracking, X_s and U_s satisfy the following constraints:

$$-0.035 < X_{s,4} < 0.035, -0.035 < X_{s,5} < 0.035 \tag{47}$$

$$\begin{bmatrix} -1.5 \\ -1.5 \\ -1.5 \\ -1.5 \end{bmatrix} < U_s < \begin{bmatrix} 1.5 \\ 1.5 \\ 1.5 \\ 1.5 \end{bmatrix} \tag{48}$$

The relation of X_s and U_s should conform to the equation to realize the steady state property:

$$X_s = f_{discrete}(X_s, U_s) \tag{49}$$

To realize tracking, the values in X_s should match the x,y,z and yaw values in reference:

$$X_{s,10} = r_1, X_{s,11} = r_2, X_{s,12} = r_3, X_{s,6} = r_4 \tag{50}$$

The recursive objective cost function to be minimized is:

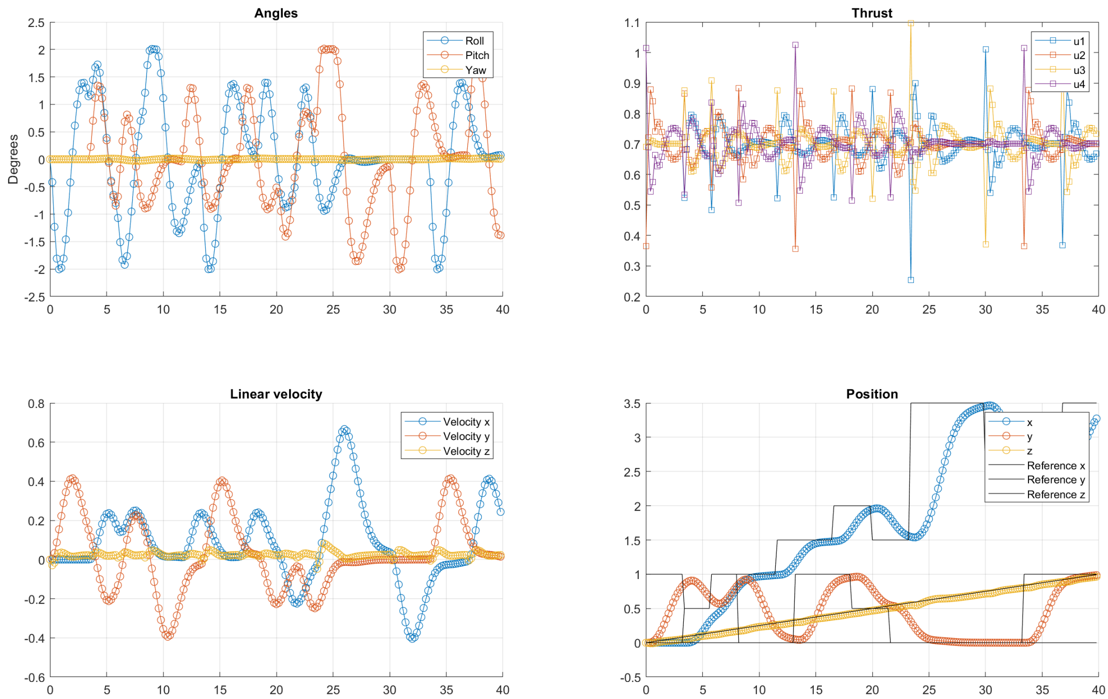
$$J(x) = J(x) + (x_i - x_s)^{\mathrm{T}} Q (x_i - x_s) + (u_i - u_s)^{\mathrm{T}} R (u_i - u_s) \tag{51}$$

Here the **Q** and **R** are both tuned as I. Then the system can be optimized by minimizing our cost function while ensuring feaibility upon the constraints.

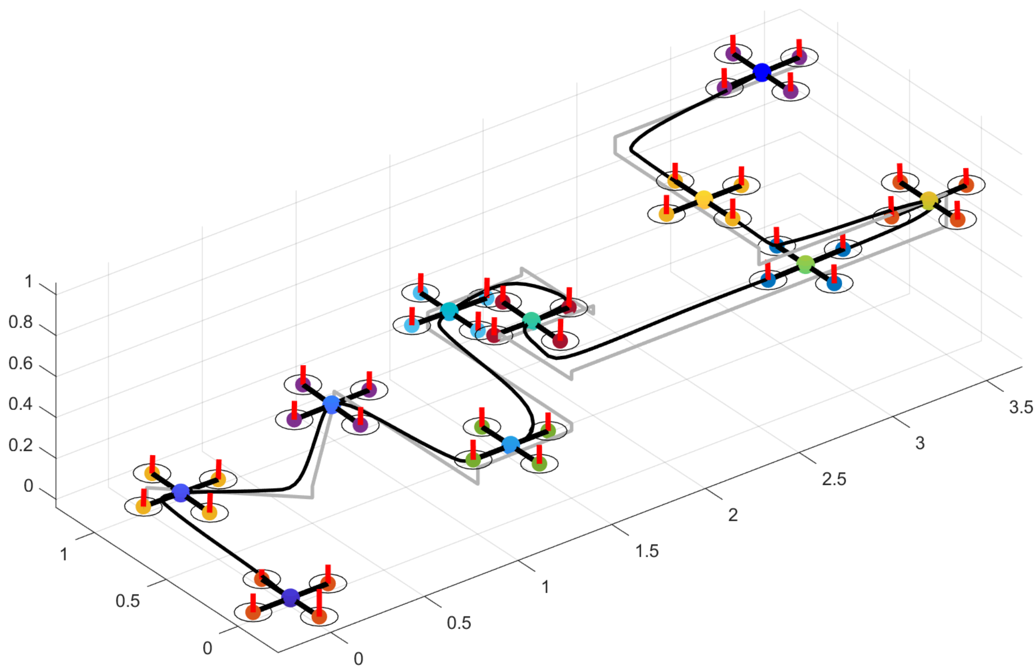
- **Explanation of why your nonlinear controller is working better than your linear one.**
  As indicated from previous questions, linear MPC is employing a model decomposed linearized around some specified condition. By doing that, we could have a very simple implementation because of the decoupling, but it will give rise to a limited control of the quadcopter since then every command will be independent.

  The non-linear MPC controller takes the full state of the quadcopter as input. Without linearization and decomposition, it will exploit the **full system dynamics**, especially for some command coupling with other motions, which will lead to faster response without noticeable overshot.

- **Plots showing the performance of your controller.**
  Results is shown in Figure 15.



(a) Plots for all states of non-linear MPC



(b) Illustration of trajectory of quadcopter with non-linear MPC

Figure 15: Simulation results with non-linear MPC