Robotic Control Theory and Systems Coursework 2 – Yujie Wang

Question 1

1) How to Run the Code.

To run the code, simply navigate to the folder "q1" and run the file Sim_Quadcopter.m.

You can test the three different simulation scenarios, by commenting and uncommenting the code from line 62 to line 64 in Quadcopter.m, which is shown in Figure 1.

```
61
62          % Setting Input [Comment & Uncomment the following code to simulate different scenarios]
63          obj.input = [0.735; 0.735; 0.735; 0.735]; %<=======euilibrium
64          %obj.input = [0; 0; 0; 0]; %<=======free fall
            %obj.input = [0.98; 0.49; 0.98; 0.49]; %<=======float&rotate
```

Figure 1: Code that defines the input of three simulation scenarios.

- For the first simulation scenario, which is when the quadcopter is in equilibrium, the inputs are calculated by equating the trust and the gravitational force, while keeping the four inputs the same.

$$\begin{cases} \gamma_1 + \gamma_2 + \gamma_3 + \gamma_4 = mg \\ \gamma_1 = \gamma_2 = \gamma_3 = \gamma_4 \end{cases}$$

$$\gamma_1 = \gamma_2 = \gamma_3 = \gamma_4 = 0.735\, rad^2/s^2$$

- For the second simulation scenario, which is free fall, simply the inputs to zeros.
- For the third simulation scenario, the inputs are calculated so that, each two diagonal propellers have the same angular velocity, but each two adjacent propellers do not, while keeping the trust equal to the gravitational force. For example:

$$\begin{cases} \gamma_1 + \gamma_2 + \gamma_3 + \gamma_4 = mg \\ 0.5\gamma_1 = \gamma_2 = 0.5\gamma_3 = \gamma_4 \end{cases}$$

$$\gamma_1 = \gamma_3 = 0.98 rad^2/s^2, \qquad \gamma_1 = \gamma_3 = 0.49 rad^2/s^2$$

2) Present & Analyse Test Results

- The results for the first scenario, which is when the quadcopter is in equilibrium, are shown in Figure 2. The diagram shows that the four elements of quadcopter's state stay the same during the simulation.
- The results for the second scenario, which is free fall, are shown in Figure 3. It shows that the linear velocity and position increase due to gravity in the negative z-direction. Note that because of the air frictional force increasing with velocity, the linear velocity will converge to a constant value and the change of position will become linear.
- The results of the third scenario, which is exhibiting change in rotation while keeping floating, are shown in Figure 4. It shows that the position and the linear velocity of the quadcopter stay the same while the angular position of the quadcopter keep changing.

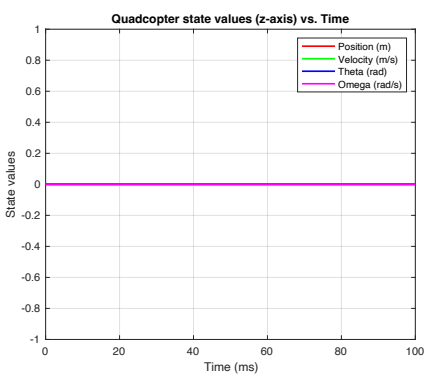


Figure 2: Results for simulation scenario 1

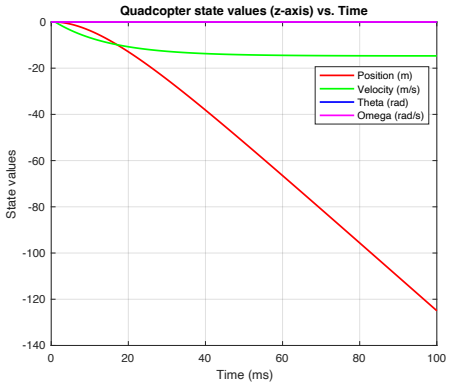


Figure 3: Results for simulation scenario 2

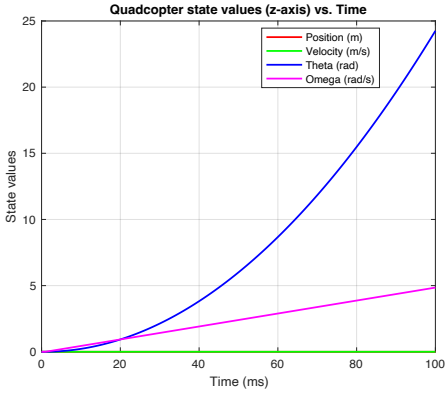


Figure 4: Results for simulation scenario 3

Question 2

1) How to Run the Code.

To run the code, simply navigate to folder "q2" and run the file "Sim_Quadcopter.m". You can simulate the linear model in different scenarios (floating, free fall, floating while rotation) by commenting and uncommenting the code between line 80 and 82 in the file "Quadcopter.m". You can also modify the equilibrium state and equilibrium input by altering the parameters stored in matrix "equilibrium_x" and "equilibrium_u", which is located between line 173 to 178 in the file "Quadcopter.m".

2) Derivation of the linearised model.

The derivation of the linearised model is written in the function linearisation(obj), located between line 138 to 198 in the file "Quadcopter.m". Firstly, we define 28 symbolic variables (line 140 to 142), where 12 of them represent the state variables, 12 of them represent the dynamics variables and 4 of them represent the input variables. Secondly, use the dynamics functions in Q1 to derive the symbolic expressions for each dynamic variables, only with state variables and input variables (line 160 to 163). Thirdly, using jacobian() command to calculate the Jacobian matrices, "**Aj**" and "**Bj**"(line 168 and 169). Finally, replace the state variables in the Jacobian matrices "Aj" and "Bj" with the equilibrium state values and equilibrium input values, and therefore drive the state matrix "**A**" and input-to-state matrix "**B**" (line 180 to 183).

The equilibrium state is when all 12 state values equal to 0, and the equilibrium input is when the four input for four propellers are all 0.735 rad^2/s^2 (quadcopter is floating).

3) Presenting comparative tests and analysing results

There are two non-equilibrium scenarios, free fall and floating while rotating. Table 1 and Table 2 compares the non-linear model, linear small error model and linear big error model in the two non-equilibrium scenarios respectively.

The linear small error model is derived by adding a small error = 0.05 to every equilibrium state element during linearisation, while the linear big error model is derived by adding a big error = 0.5 to every equilibrium state element during linearisation.
The linear model with zero error has extremely the same behaviour as the non-linear model, thus the linear model is not compared in this report.

Scenario 2: Free Fall

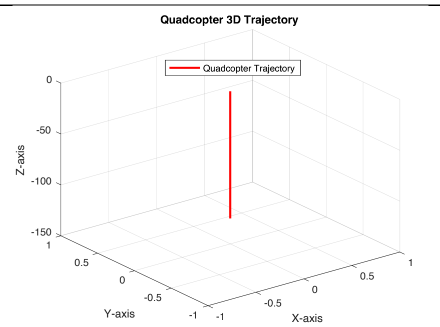
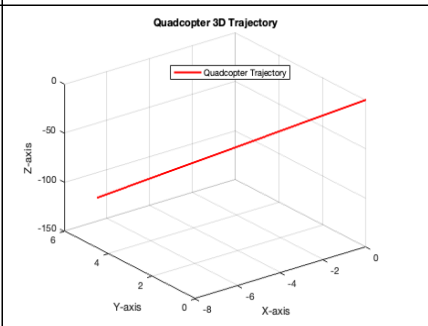
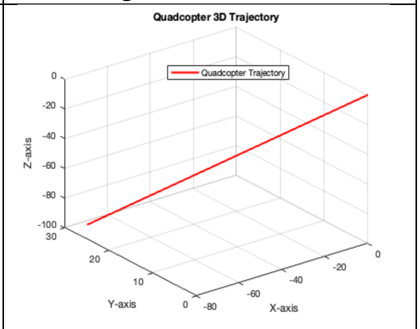
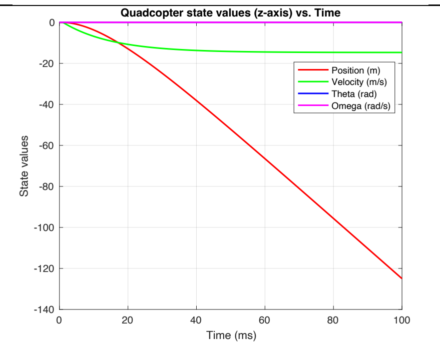
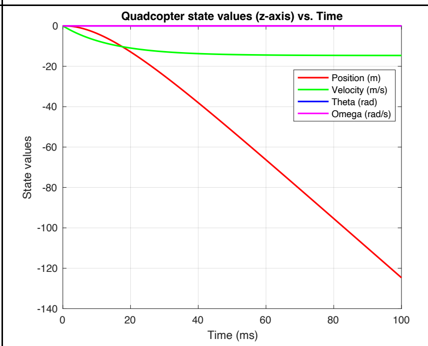
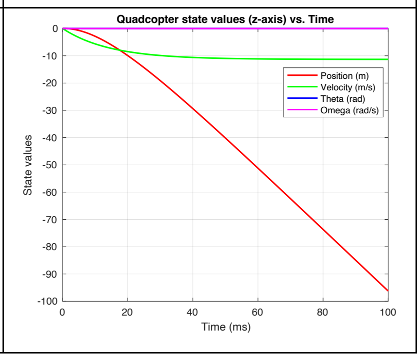
	Non-linear Model	Linear Small Error Model	Linear Big Error Model
Trajectory			
State Values (z-axis)			

Table 1: Comparison between non-linear and linear model in Scenario 2

Scenario 3: Floating while Rotating

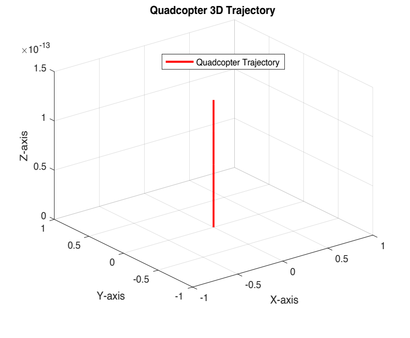
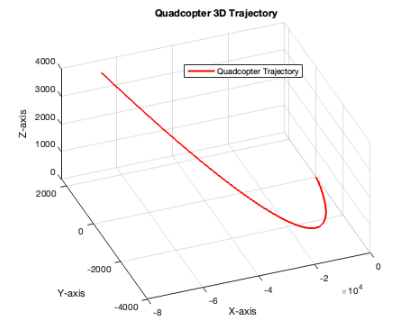
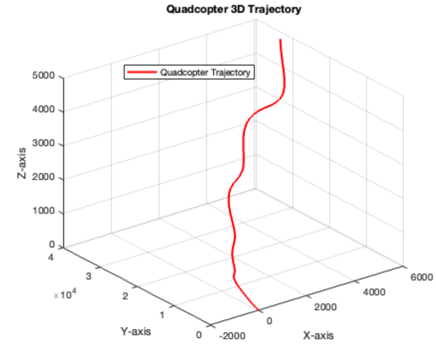
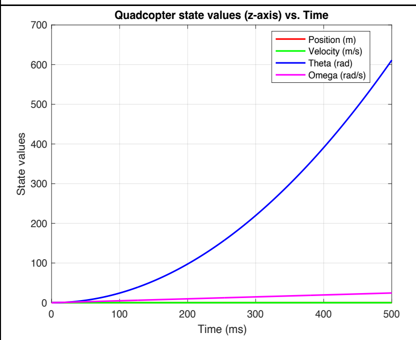
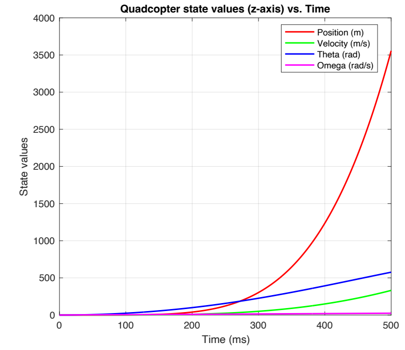
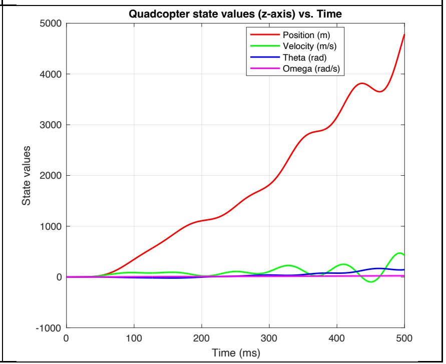
	Non-linear Model	Linear Small Error Model	Linear Big Error Model
Trajectory			
State Values (z-axis)			

Table 2: Comparison between non-linear and linear model in Scenario 3

The comparison in Table 1 shows that in the free fall scenario, the linear models with errors exhibit inaccuracy in terms of position and velocity, especially in the x-direction and y-direction, compared to the non-linear model. The angular position and angular speed are not affected by the error in linearisation as they are zero. The linear model with big error exhibits greater errors than the linear model with small error, and the error tends to increase in the x and y directions as the linearisation error increase.

The comparison in Table 2 shows that there are errors in linear models with errors compared to the non-linear model, in terms of position, velocity m angular position and angular velocity. The position of the quadcopter should be unchanged (almost unchanged) during the simulation, but both two linear models with errors shows great bias in position. The difference of the angular position between the nonlinear and small error models are mild. The big error shows greater errors compared to linear model with small error, especially in terms of positions and velocities, as the big error model's position and velocity become unpredictable after 200 ms.

Question 3

1) How to Run the Code.

To run the code, simply navigate to folder "q3" and run the file "Sim_Quadcopter.m". The total simulation time "TOTAL_TIME" should be at least 132s for completed simulation.

2) How the controller is implemented.

A full state feedback controller is implemented to allowing the quadcopter to pass pre-defined checkpoints. Firstly we discretise the linear model derived from Q2, using c2d(), with a sampling time of 0.1s. Then we can check whether is system is reachable by the function "checkReachability()" defined between line 284 and 295 in Quadcopter.m. If the system is reachable then we can proceed to designing our FSF controller.

A state feedback controller is defined as a linear combination of the errors between state variables and the desired reference r_x (Lecture 8 slides), where the complete system has the following form:

$$x[k+1] = (A - BK)x[k] + BKr_x$$

The value K is determined by the state matrix A, input-to-state matrix B and the poles of the system. The designed K shoud make $A - BK$ equal to the system's eigenvalues. The choice of system's poles should be less than 1 for stability, but also closer to 1 for faster system response. Since there are 12 elements in the system state, there are 12 poles to tune. Tuning 12 poles can be extremely time-consuming, therefore we applied linear quadratic regulator (LQR) to determine the value of K. We assume the simplest case that $\boldsymbol{R} = 1$ and $\boldsymbol{Q} = 1$, and use lqr() to calculate the value of K (line 115).

Once K is determined, we calculate the error e of the system, which is the difference between the current state and the reference state. The input of the system u is then defined as $u = -k * e$. In our quadcopter system, to prevent the quadcopter from sudden falling, the input u should start with its equilibrium values, such that:

$$u(t) = u_{equilibrium} - k * e(t)$$

We apply this full state feedback control method to update the input of the quadcopter, such that the quadcopter will fly in a pre-defined trajectory.

3) Presenting simulation tests and analysing results

Figure 5 shows the quadcopter trajectory. It shows that the quadcopter passed all checkpoints accurately. It also demonstrates the perfect performance of the FSF controller in position control because the trajectory is highly smooth and non-oscillated. Figure 6 shows the inputs of the quadcopter γ_i varying during the whole simulation. It demonstrates that the input to each propeller is strictly limited within [-1.5, +1.5]. Figure 7 presents the quadcopter landing speed (speed from Point 7 to Point 8) with respect to time. It shows that the final landing speed is successfully moved down at 0.09m/s to allow the quadcopter lands safely.

The three results show that the full-state feedback controller was implemented successfully.

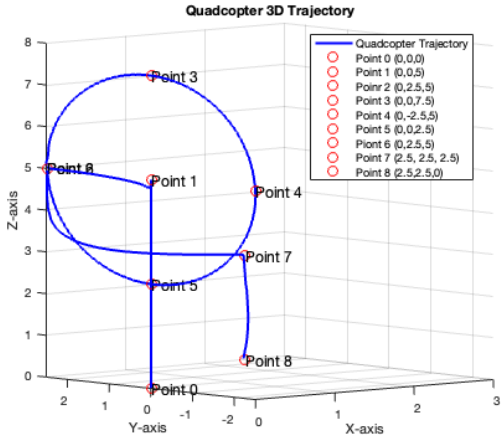


Figure 5: Quadcopter Trajectory

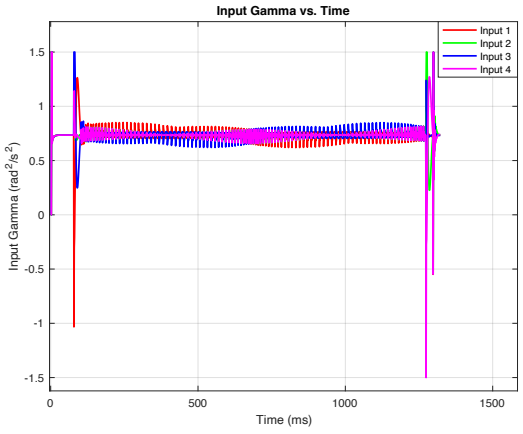


Figure 6: Quadcopter input gamma vs. time

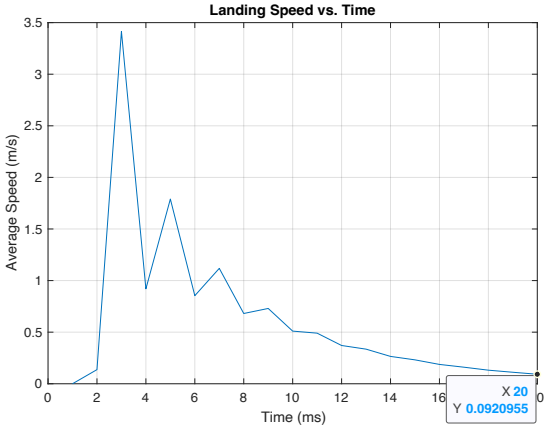


Figure 7: Landing speed vs. time

Question 4

1) How to Run the Code.

To run the code, simply navigate to folder "q4" and run the file "Sim_Quadcopter.m". The total simulation time "TOTAL_TIME" should be more than 300s for completed simulation. You can vary the parameters for wind disturbance simulation by changing the mean value, stand deviation and noise amplitude of the wind gaussian noise, between line 153 and 155, in Quadcopter.m.

2) Adding wind disturbance to quadcopter dynamics.

A function "wind_noise()" is created (line 275 to 283, Quadcopter.m) to generate gaussian noise. When updating the dynamic of the quadcopter, add random gaussian noise to the quadcopter's position, velocity, angular position, and angular velocity (line 345 to 348) to simulate the wind disturbance. The parameters chosen for the wind disturbances simulation is:

- Mean value = 0
- Standard deviation = 0.5
- Noise amplitude = 0.01

3) Presenting simulation tests.

Figure 8 shows the quadcopters' trajectory under wind disturbances. It illustrates that the FSF controller designed in Q4 still have good performance under certain wind disturbances because the quadcopter can still successfully pass all the checkpoints, with relatively small oscillation.

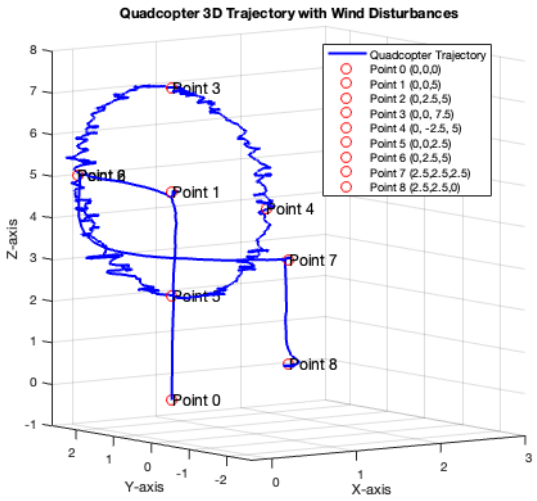


Figure 8: Quadcopter trajectory with wind disturbances