Coursework 2

Michael Adlerstein

December 19, 2022

1 part 1

The drone starts at a height of 5 meters. Between 0 and 2 seconds it hovers stationary, between 2 and 4 seconds the propeller inputs increase by 15% and finally from 4 to 8 seconds, the third propeller shuts down causing the drone to fall and rotate due to the active propellers tilting the drone. The code sets a timer from the start of the simulation adding the time in increments of 0.02. At every time increment the dynamics of the drone are updated according to the inputs set. from 0 to 2 seconds the inputs are set at equilibrium ($\frac{mg}{4}$) for every propeller. From 2 to 4 seconds ($\frac{1.5mg}{4}$) and from 4 to 8 seconds the third propeller is set to 0.

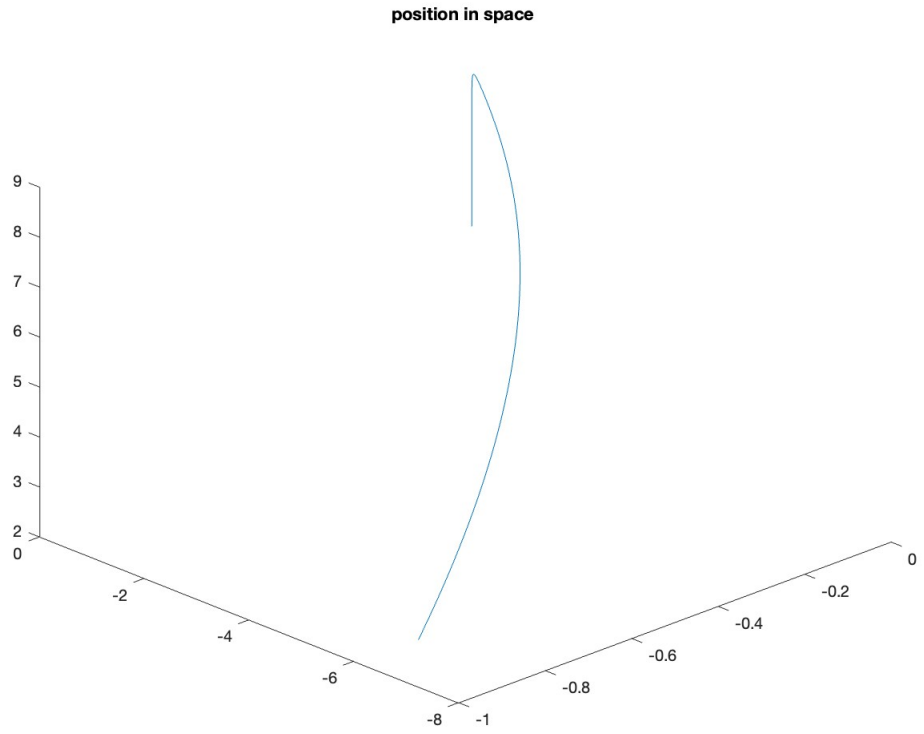

Figure 1: positions simulation

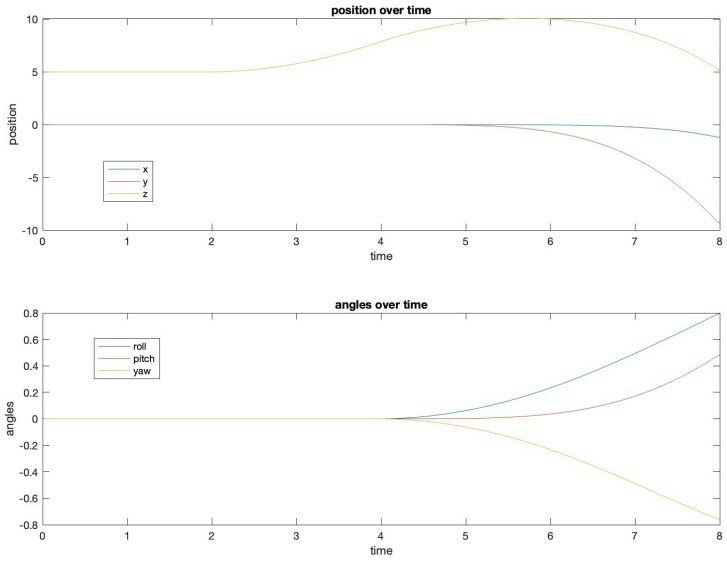

Figure 2: positions simulation

2 part 2

2.1 2a

for this section run the matlab script part2a.m
the model of the drone dynamics was linearised by using symbolic variables throughout the calculations of the states. Once the states were computed the jacobian was computed between the derivative states and the states of the systems for A and the derivative states and the inputs for B.
(A_J = jacobian(xd , x) , B_j = jacobian(xd ,u)). Once the jacobians were obtained the equilibrium point was chosen at hover position where angles, omegas and accelerations are all set to 0, since the drone cannot be at equilibrium until this requirement is met. This is achieved when the force on each propeller is equal to

$$input_i = \frac{mg}{4} \tag{1}$$

the drone class includes the physics parameters of the drone such as acceleration and angular accelerations. In the update function , the code calls the other instances to simulate the drone at different time steps. Since the system is discrete, the states are updated in the following form.

$$x_{t+1} = \dot{x_t} \times \delta t + x_{t-1} \tag{2}$$

2.2 2b

the linearised dynamics of the drone are similar to the non-linear dynamics, the main difference occurs when one of the propellers turns off and the drone start to drop. In the non linear dynamics the drone position changes both in the x and y axis as the angle of the drone causes the propellers to deviate from the 0 axis. In the linearised dynamics the drone falls whilst maintaining no deviation.

the angular dynamics are similar for the roll and yaw but differ in pitch, since the linearised model maintains a constant pitch.

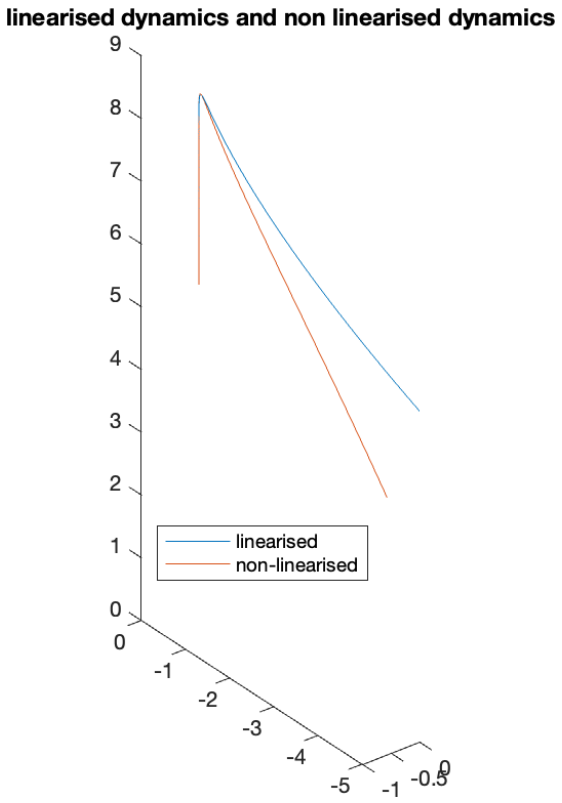

Figure 3: positions of the linearised and non linearised models over 8 seconds

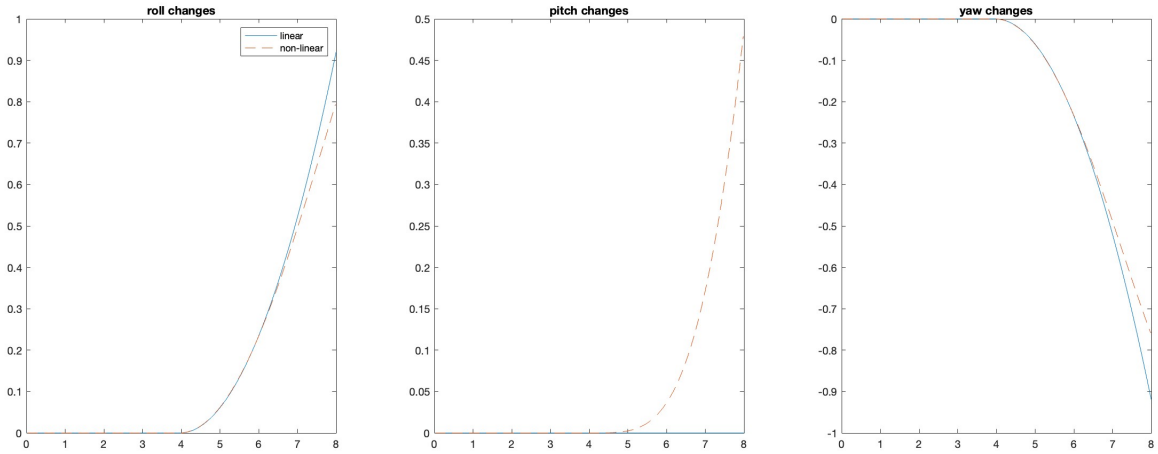

Figure 4: roll pitch and yaw in linearised and non linearised model 8 seconds

3 part3

the code implements a full state feedback in order to set the inputs so that the drone travels to the desired positions. the closed loop equation of the FSF system becomes

$$x[k+1] = (A - BK)x[k] + BKr_x \tag{3}$$

where the gain K is takes into account the position of the system's eigenvalues with the condition that they lie in the unit circle and are not repeated. the values of K are computed with the command place. The circular trajectory was created by creating intermediate points in a circular shape and setting the

points as way points alongside the main targets. Once the drone reaches the checkpoint n within a threshold euclidean distance , the checkpoint n+1 becomes the new target.

```
function FSF(obj, ref_pos)
    %calculate the input given a target point using full state
    %feedback
    obj.K = place(obj.Ad , obj.Bd , obj.eigenvalues);
    %error between reference and current state
    e = obj.x - ref_pos;
    %adjust input
    obj.inputs = obj.equ_inputs - obj.K*e;
    obj.inputs = obj.inputs(:);
end
```

Figure 5: code for the full state feedback

The FSF (full state feedback) function takes two inputs: an "obj" which is an object that represents the drone, and a "ref-pos" which is a reference position for the drone.

The first line inside the function calculates the control gains for the full state feedback control system using the "place" function. The "Ad" and "Bd" matrices are the discrete-time state transition and control input matrices for the drone, and the "eigenvalues" are a set of desired eigenvalues for the closed-loop system. The "place" function calculates the control gains (K) that will place the closed-loop poles of the system at the desired eigenvalues. The second line calculates the error between the current state of the drone (obj.x) and the reference position (ref-pos). The third line adjusts the input to the drone using the control gains and the error. The final line ensures that the inputs are in the correct form for the drone's actuators.

In order to make the drone hover at (5,5,5) for five seconds a condition was added in the update function so to start a separate counter which overwrote the target checkpoint to the current checkpoint for 5 seconds. And a separate condition was added to manually set the velocity once the drone arrives at the final checkpoint.

figure 5 shows the drone starting at position (0,0,0) and reaching all the checkpoints before landing at (5,5,0). At the last checkpoint the velocity is changed so that the drone is able to land slowly at the final target.

from figure 6 it can be seen that the drone changes it's orientation in it's roll , pitch and yaw to achieve the desired position, whilst also trying to maintain an altitude of 5

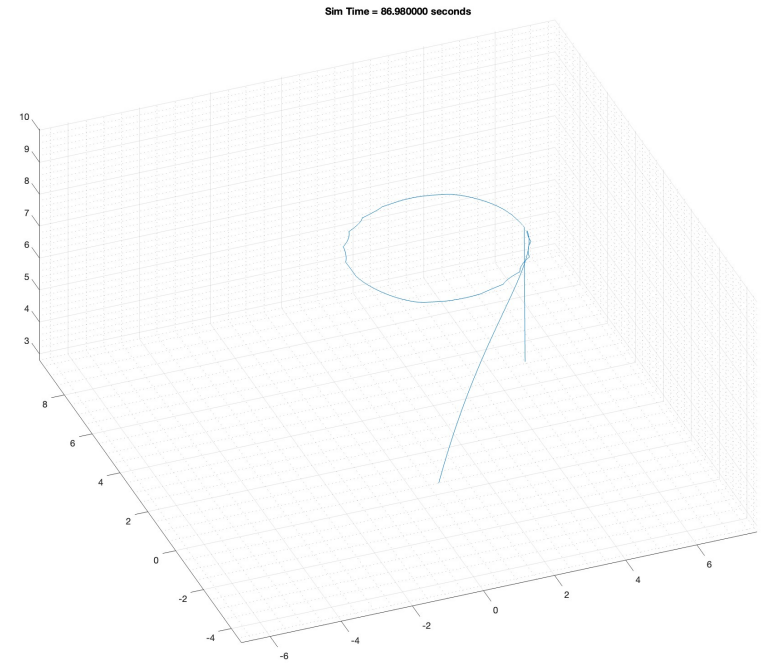

Figure 6: drone trajectory

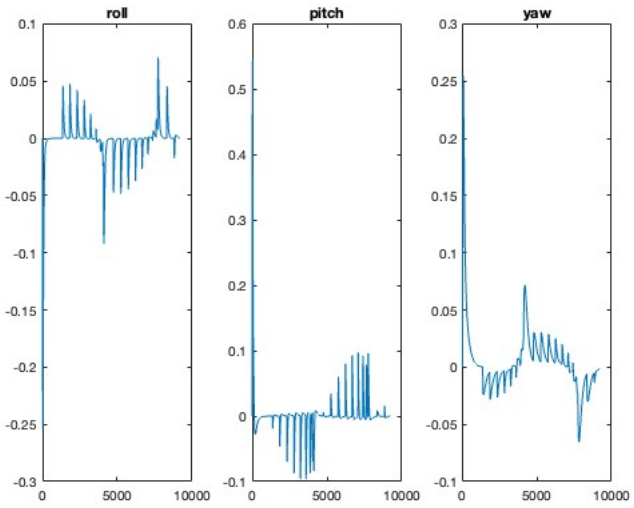

Figure 7: drone trajectory

3.1 3b

The state being measured are $\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$, which are commonly measured by an accelerometer. To simulate sensor noise I added randomly distributed 0 mean Gaussian error to the state measurement and the results are the following.

despite the noise the drone managed to complete a near circular path and land in the desired spot. in order to compensate for the noise one can choose lower values for the eigenvalues in order to achieve a slower and more stable response.

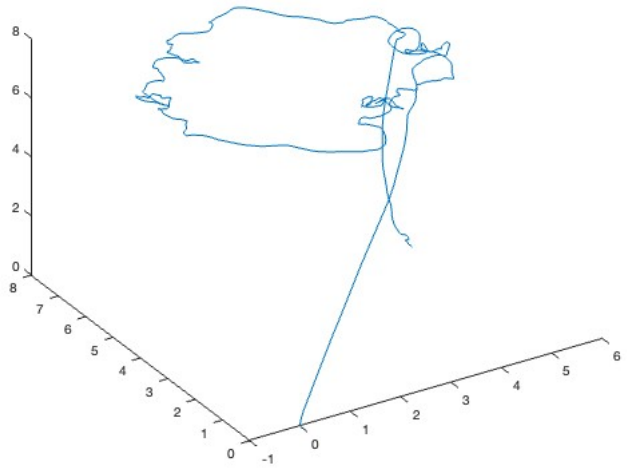

Figure 8: drone angles on different time steps

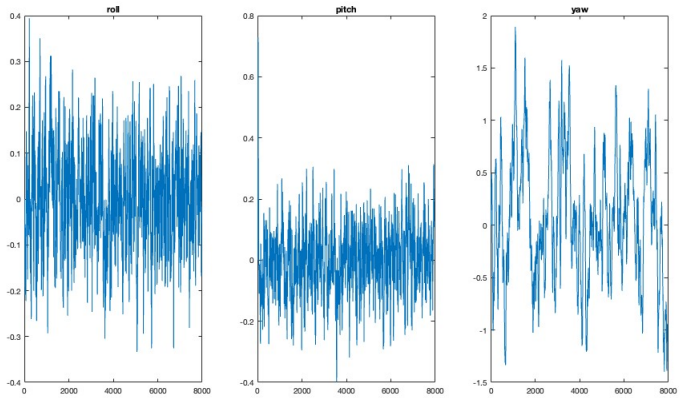

Figure 9: drone angles with noise