Final Project Report

MCEN 5228: Linear Systems

Daniel Torres

Fall 2019

1. For this project I chose the multirotor aircraft model. The state variables of the system represented by:

$$\mathbf{x} = [x \;\; \dot{x} \;\; y \;\; \dot{y} \;\; z \;\; \dot{z}] \tag{1}$$

where each of the quadcopter's x, y and z inertial positions and their corresponding velocities $\dot{x}, \dot{y}$ and $\dot{z}$ are represented. The output of the system is:

$$\mathbf{y} = [x \;\; y \;\; z] \tag{2}$$

representing the x, y and z inertial positions of the quadcopter. The matrices of the state space model are as follows:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -0.0104 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -0.0104 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -0.0208 \end{bmatrix} \tag{3}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -0.04167 & 0 & 0.04167 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -0.04167 & 0 & 0.04167 \\ 0 & 0 & 0 & 0 \\ 0.4 & 0.4 & 0.4 & 0.4 \end{bmatrix} \tag{4}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{5}$$

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{6}$$

And the following input vector **u**:

$$\mathbf{u} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \tag{7}$$

represents the four inputs to the system which in this case is the four different motors of the quadcopter.

Using the following state space equations:

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} \tag{8}$$
$$\mathbf{y} = \mathbf{Cx} + \mathbf{Du} \tag{9}$$

We get the following state space realization by plugging in the matrices listed above:

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -0.0104 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -0.0104 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -0.0208 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ -0.04167 & 0 & 0.04167 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -0.04167 & 0 & 0.04167 \\ 0 & 0 & 0 & 0 \\ 0.4 & 0.4 & 0.4 & 0.4 \end{bmatrix} \mathbf{u}$$

$$\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{u}$$

The purpose of this system is to fly in a controlled manner. The most important aspects of the system to model are the quadcopter's dynamics which are given by state matrix **A**, the input matrix **B**, and the output matrix **C**. In this case the **D** matrix is just a 3x4 zeros matrix. These are important to model because from these matrices we can determine how controllable our system is and if we can create an observer for the system which is important information to know if we we're trying to do things like compete in say something like the DARPA Urban Challenge.

2. The open loop system poles are as follows where the name of the vector here matches the name of the vector in the Matlab code:

$$\mathbf{poles\_ol} = [0\ \ -0.0104\ \ 0\ \ -0.0104\ \ 0\ \ -0.0208]$$

These are the eigenvalues of the **A** matrix listed above. Because the poles are all either zero or negative then this system is marginally stable. I simulated the open loop system's response to an input of one on all the thrusters, called $\boldsymbol{u}$ in the code, for a duration of 5 seconds with an initial state where the name of the vector here matches the name of the vector in the Matlab code:

$$\mathbf{x0} = [1\ \ 0\ \ -2\ \ 0\ \ 0\ \ 0]$$

It can be seen from Figure 1 below that the quadcopter's z-position increases continuously for the duration of the simulation:

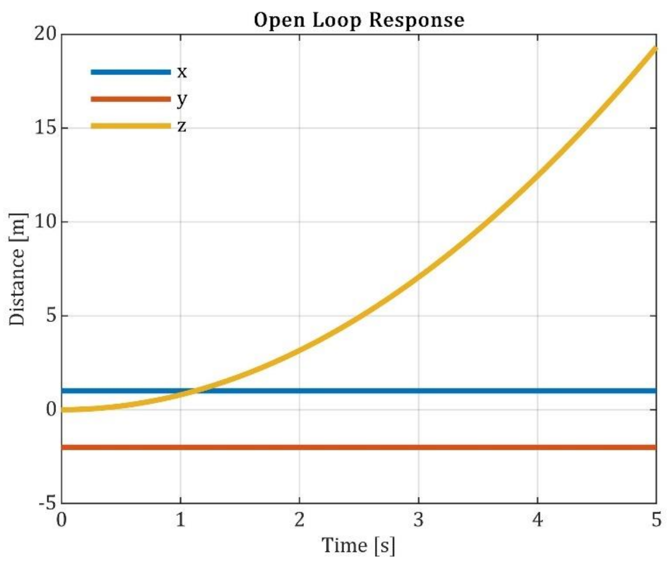

Figure 1: Quad rotor's open-loop response to input vector with initial state vector

This response makes sense because even though the system isn't unstable, applying a continuous thrust will cause the quadcopter's thrusters to apply a continuous thrust force upward that would continue to elevate the quadrotor's and increase the quad rotor's z-position.

3. Using matrices, **A**, **B**, and **C** listed above along with Matlab's controllability and observability functions, "ctrb" and "obsv" respectively, it was determined that the system is both completely controllable and observable because the rank of the matrices is equal to the number of state variables which is 6 therefore the system is also stabilizable and detectable. This means that the closed loop system's poles can be placed arbitrarily.

4. Because the system is fully controllable, I placed the state feedback controller's poles on the negative real axis as follows where the name of the vector matches the name of the vector in Matlab:

$$\mathbf{poles\_cl} = [-8 \ -8.5 \ -9 \ -9.5 \ -10 \ -10.5]$$

I placed the poles here because I wanted to eliminate overshoot, have a rise time less than 1 second and be able to track the reference input with less than 10% error. Using the "place" function with the **A** and **B** matrices I was able to find the gain matrix **k** to use in the formulation of the closed loop system model named "sys_cl" in the code. Figure 2 shows the results of simulating the closed loop system response using the same input and initial conditions used for the open loop system simulation in part 2:

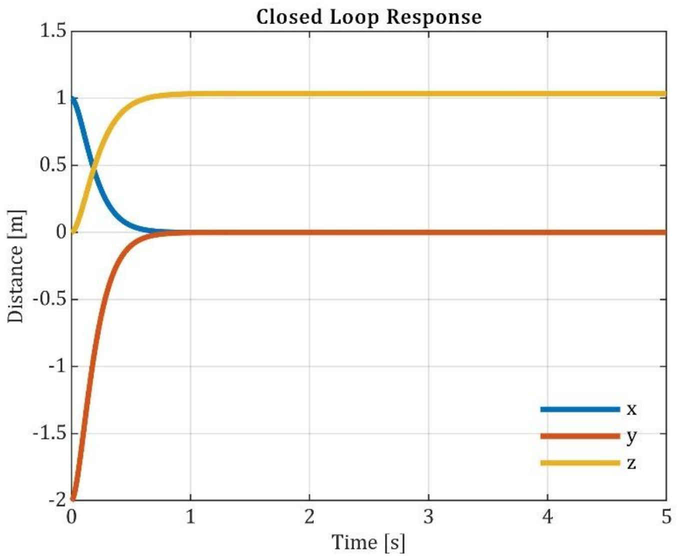

Figure 2: Closed loop system response to same input and initial conditions used for open loop response simulation

It can be seen from Figure 2 that the requirements are met. The final height of the quadcopter is around 1.05 meters which is well within the tolerance I set. The rise time is also much less than the 1 second condition I set. The poles did help with the overshoot and rise time requirements, but they alone didn't help reach the reference tracking tolerance, so I used the precompensation gain **f** (found by trial-and-error) to help achieve that requirement. This gain is also named f in the code.

5. For the observer I chose to make the poles ten times bigger than the closed loop-system poles so that the observer could be one order of magnitude faster than the closed-loop system as suggested in class. They are listed as follows where the name of the vector here matches the name of the vector in Matlab:

$$\mathbf{poles\_obs} = [-80\ \ -85\ \ -90\ \ -95\ \ -100\ \ -105]$$

I used the same "place" function with the $\mathbf{A^T}$ and $\mathbf{C^T}$ to find the $\mathbf{L}$ gains matrix for the observer system. When simulating the closed loop system consisting of the observer and the state variable feedback, named "sys_obs" in the code, I got the following results shown in Figure 3:

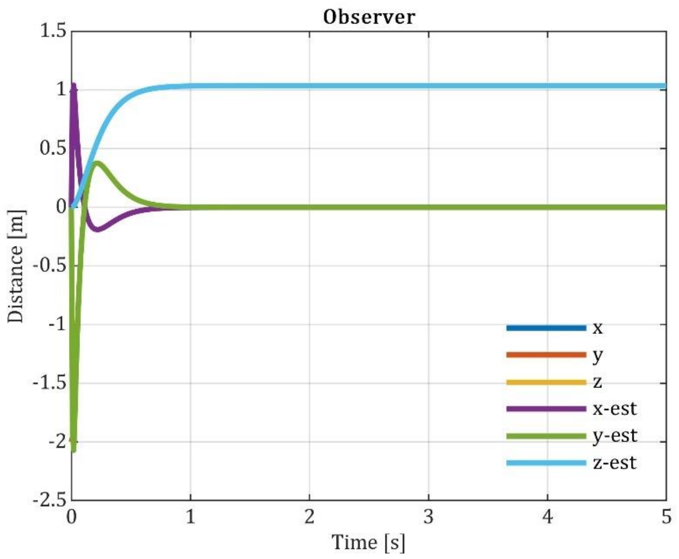

Figure 3: Observer response to input and initial conditions

From Figure 3, the observer estimates quickly converge and begin to track the state variables well once in steady state. When comparing this to the closed loop system simulation from part 4 there's a few differences. The first is that the observer started from the origin rather than the initial condition vector from part 1. There's also a bit of overshoot in the x and y state responses but not in the z state which is fine.

I included Figure 4 to show what happens in the first half second of the plot above because it's easier to see that the error does converge to zero in this range of time and that we get reliable reference tracking very early on.

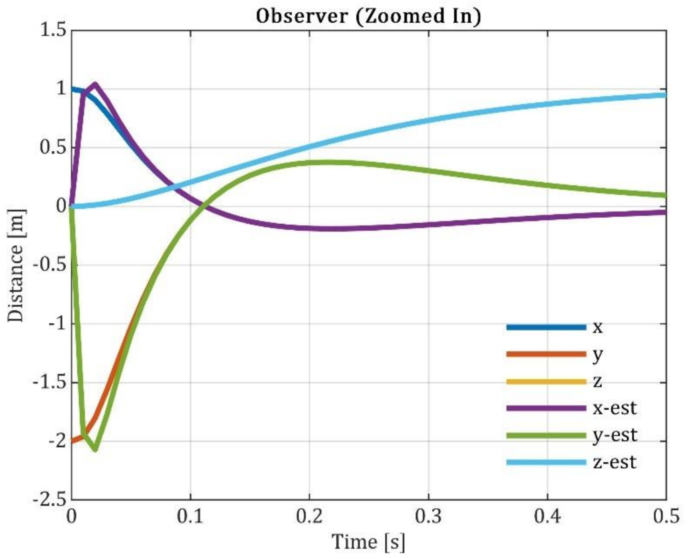

Figure 4: Zoomed-in observer response

Appendix – Matlab code used for project

```
% Daniel Torres
% Final Project - Multirotor Aircraft
% MCEN 5228 - Linear Systems
% Fall 2019

clear all; close all; clc;

% Multirotor aircraft state space realizations

A = [0 1 0 0 0 0;
     0 -0.0104 0 0 0 0;
     0 0 0 1 0 0;
     0 0 0 -0.0104 0 0;
     0 0 0 0 0 1;
     0 0 0 0 0 -0.0208];

B = [0 0 0 0;
     -0.04167 0 0 0.04167;
     0 0 0 0;
     0 -0.04167 0 0.04167;
     0 0 0 0;
     0.4 0.4 0.4 0.4];

C = [1 0 0 0 0 0;
     0 0 1 0 0 0;
     0 0 0 0 1 0];

D = [0 0 0 0;
     0 0 0 0;
     0 0 0 0;
     0 0 0 0];

sys = ss(A,B,C,0); % create state space model object

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Part 2: Determine the system poles

% calculate A matrix eigenvalues
poles_ol = eig(A); % poles of open loop system
fprintf('The system poles are: %f, %f, %f, %f, %f and %f.\n', poles_ol);
disp('The system poles are either zero or negative so it''s marginally stable.');

% simulate input to system without controller
t = 0:0.01:5; % time vector
x0 = [1; 0; -2; 0; 0; 0]; % initial state vector
input_gain = 1; % input gain variable to scale input to thrusters (for testing)
u1 = input_gain*ones(1,length(t)); % input to thruster 1
u2 = input_gain*ones(1,length(t)); % input to thruster 2
u3 = input_gain*ones(1,length(t)); % input to thruster 3
u4 = input_gain*ones(1,length(t)); % input to thruster 4
```

```
u = [u1; u2; u3; u4]; % combine all thruster inputs into one vector
[y_ol,t_ol,x_ol] = lsim(sys,u,t,x0); % simulate open-loop response

% plot open-loop response x, y, z inertial positions
figure(1);
plot(t_ol,y_ol(:,1),'LineWidth',2); % plot open loop x-position
hold on; grid on;
plot(t_ol,y_ol(:,2),'LineWidth',2); % plot open loop y-position
plot(t_ol,y_ol(:,3),'LineWidth',2); % plot open loop z-position
title('Open Loop Response');
xlabel('Time [s]');
ylabel('Distance [m]');
legend('x','y','z','Location','Northwest');
legend boxoff

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Part 3: Determine controllability, observability, stabilizability,
%         and detectability

% determining controllability matrix and rank
Co = ctrb(A,B);
rank_Co = rank(Co);
fprintf('Controllability matrix rank: %d\n', rank_Co);

% determining observability matrix and rank
Ob = obsv(A,C);
rank_Ob = rank(Ob);
fprintf('Observability matrix rank: %d\n', rank_Ob);

% determining stabilizability matrix and rank
stbl = [A B];
rank_stbl = rank(stbl);
fprintf('Stabilizability matrix rank: %d\n', rank_stbl);

% determining detectability matrix and rank
detec = [A; C];
rank_detec = rank(detec);
fprintf('Detectability matrix rank: %d\n', rank_detec);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Part 4: Design a state feedback controller

poles_cl = [-8 -8.5 -9 -9.5 -10 -10.5]; % poles for state-feedback controller
k = place(A,B,poles_cl); % calculate k gain matrix to place poles

f = 44; % precompensation gain to eliminate steady state error
A_cl = A-B*k; % closed-loop A matrix
B_cl = B*f; % closed-loop B matrix
sys_cl = ss(A_cl,B_cl,C,0); % create closed-loop state space object
[y_cl,t_cl,x_cl] = lsim(sys_cl,u,t,x0); % simulate closed-loop response

% closed-loop x, y, z positions
```

```
figure(2);
plot(t_cl,y_cl(:,1),'LineWidth',2); % plot closed-loop x-position
hold on; grid on;
plot(t_cl,y_cl(:,2),'LineWidth',2); % plot closed-loop y-position
plot(t_cl,y_cl(:,3),'LineWidth',2); % plot closed-loop z-position
title('Closed Loop Response');
xlabel('Time [s]');
ylabel('Distance [m]');
legend('x','y','z','Location','Southeast');
legend boxoff

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Part 5: Design an observer to reconstruct the state

x0_obs = [x0' x0']; % initial state vector for observer
poles_obs = poles_cl.*10; % place observer poles 10x as far as CL poles
L = place(A',C',poles_obs)'; % find gains matrix L for observer poles
At = [A-B*k B*k; zeros(size(A)) A-L*C]; % A-tilde matrix
Bt = [B*f; zeros(size(B))]; % B-tilde matrix
Ct = [C zeros(size(C))]; % C-tilde matrix
sys_obs = ss(At,Bt,Ct,0); % create observer state space system
[y_obs,t_obs,x_obs] = lsim(sys_obs,u,t,x0_obs); % simulate observer response

figure(3); % plot observer response
states = x_obs(:,1:2:6); % x, y, z states from observer
e = x_obs(:,7:2:12); % error from observer
x_hat = states - e; % calculate estimated states
x_ = states(:,1); % x-state
y_ = states(:,2); % y-state
z_ = states(:,3); % z-state
x_est = x_hat(:,1); % estimated x-state
y_est = x_hat(:,2); % estimated y-state
z_est = x_hat(:,3); % estimated z-state

% plot x, y , z states
plot(t_obs,x_,'LineWidth',2); hold on; grid on;
plot(t_obs,y_,'LineWidth',2);
plot(t_obs,z_,'LineWidth',2);

% plot estimated x, y, z states
plot(t_obs,x_est,'LineWidth',2);
plot(t_obs,y_est,'LineWidth',2);
plot(t_obs,z_est,'LineWidth',2);

title('Observer');
xlabel('Time [s]');
ylabel('Distance [m]');
legend('x','y','z','x-est','y-est','z-est','Location','Southeast');
legend boxoff

figure(4); % plot the first 0.5 seconds of observer to see convergence
% plot x, y , z states
len = 1:51;
```

```
plot(t_obs(len),x_(len),'LineWidth',2); hold on; grid on;
plot(t_obs(len),y_(len),'LineWidth',2);
plot(t_obs(len),z_(len),'LineWidth',2);

% plot estimated x, y, z states
plot(t_obs(len),x_est(len),'LineWidth',2);
plot(t_obs(len),y_est(len),'LineWidth',2);
plot(t_obs(len),z_est(len),'LineWidth',2);

title('Observer (Zoomed In)');
xlabel('Time [s]');
ylabel('Distance [m]');
legend('x','y','z','x-est','y-est','z-est','Location','Southeast');
legend boxoff

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
The system poles are: 0.000000, -0.010400, 0.000000, -0.010400, 0.000000 and -0.020800.
The system poles are either zero or negative so it's marginally stable.
Controllability matrix rank: 6
Observability matrix rank: 6
Stabilizability matrix rank: 6
Detectability matrix rank: 6
```

Published with MATLAB® R2019a