AE353: Design Problem 03

Nico Alba

April 7, 2017

1 Design Verifications and Requirements

The code provided in `DesignProblem03` simulates an unpowered glider that is similar to one used for experiments on fixed-wing perching. This glider has one control surface, an elevator. Sensors allow you to measure both the pitch angle of the glider and the relative angle of the elevator. The goal is to make the glider fly as long a distance as possible, if released at a height of about two meters with an unknown forward velocity.

We cannot actually control the elevator angle. However, we *can* control the elevator angular rate, ϕ. I will try to design an observer that will make my glider fly an average of at least 20 ft. after 1000 simulations. We will check this by making sure that the mean and median of all the simulations are above 20.

- Glide an average of over 20 ft. After 1,000 trials, sum the final distances and divide by 1,000 to get the average.
- Standard deviation below 5 (An arbitrary selection)
- Apply reference Tracking

2 Derive a Model

The motion of the glider is governed by ordinary differential equations with the form

$$\begin{bmatrix} \ddot{x} \\ \ddot{z} \\ \ddot{\theta} \end{bmatrix} = f(\theta, \phi, \dot{x}, \dot{z}, \dot{\theta}, \dot{\phi}) \tag{1}$$

where θ is the pitch angle, ϕ is the elevator angle, $\dot{x}$ is the forward speed, $\dot{z}$ is the vertical speed, $\dot{\theta}$ is the pitch angular rate, and $\dot{\phi}$ is the elevator angular rate (which an actuator allows you to specify). You might be interested to know that these equations were derived by applying a flat-plate model of lift c_L and drag c_D as a function of angle of attack α, for both the wing and elevator:

$$c_L = 2 \sin \alpha \cos \alpha \qquad\qquad c_D = 2 \sin^2 \alpha$$

We want to maximize: c_L/c_D:

$$\frac{c_L}{c_D} = \frac{2\sin\alpha\cos\alpha}{2\sin\alpha\sin\alpha} = \frac{1}{\tan\alpha} \tag{2}$$

The maximum, infinity, can be achieved when α is equal to 0. So we should shoot for an equilibrium elevator angle of 0.

```
% Load the equations of motion.
load('DesignProblem03_EOMs.mat');
% Parse the equations of motion.
f = symEOM.f;
% Define symbolic variables that appear in the equations of motion.
syms theta phi xdot zdot thetadot phidot
% Proceed to linearize or do whatever else you like...
%     (see: help sym/jacobian, help sym/diff, help sym/subs, etc.)
```

The first step of our standard approach to control design is to linearize the nonlinear equations of motion about an equilibrium point. In order to do so, we need to find an equilibrium point—in other words, find a solution to

$$0 = f(\theta, \phi, \dot{x}, \dot{z}, 0, 0).$$

It may be hard to solve this equation by hand. The code that solves this equation numerically in MATLAB using the function `fsolve` has been provided:

```
% INPUTS:
%     f is a symbolic description of the nonlinear EOMs
%     (theta, phi, xdot, zdot) is a guess at the equilibrium point
% OUTPUTS:
%     (theta, phi, xdot, zdot) is an equilibrium point near the guess
function [theta,phi,xdot,zdot] = ...
      GetEquilibriumPoint(f,theta,phi,xdot,zdot)
% Initial guess
x0 = [0;0;6;0];
% Symbolic states
syms theta phi xdot zdot thetadot real
% Symbolic inputs
syms phidot real
% Symbolic EOMs with thetadot=0 and phidot=0
g = subs(f,[thetadot,phidot],[0,0]);
% Numeric EOMs
vars = [theta;phi;xdot;zdot];
g = matlabFunction(g,'Vars',{vars});
% Find solution
xe = fsolve(g,x0);
% Parse solution
theta = xe(1);
```

```
23 phi = xe(2);
24 xdot = xe(3);
25 zdot = xe(4);
26 end
```

By specifying the equilibrium points in the MATLAB function `GetEquilibriumPoint` as

$$\begin{bmatrix} \theta_e \\ \phi_e \\ \dot{x}_e \\ \dot{z}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} 0.004613 \\ -0.06604 \\ 6.097 \\ -0.5488 \\ 0 \end{bmatrix}$$

**Note that dotθ_e was set specified as 0 by us, because we want it to be 0 at steady state.*

Next, we must add θ and ϕ to our equations of motion, f, with one line: `f = [thetadot ; phidot; f]`. by taking the Jacobian of this matrix with respect to our states and inputs, we can solve our A and B matrices.

$$A = \frac{df}{d(state)} \tag{3}$$

and

$$B = \frac{df}{d(input)} \tag{4}$$

Our MATLAB code give this to us in symbolic variables. So we must substitute in our equilibrium values.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ -8.77 & 0.80 & -0.015 & -0.17 & 0.1 \\ 120.6 & 24.2 & 1.43 & -19.9 & 1.1 \\ -56.4 & -80.64 & 0.83 & 9.18 & -5.39 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \\ 0.0122 \\ 0.198 \\ -0.658 \end{bmatrix}$$

For C, we have the option of choosing one or two variables from the sensors: ϕ and θ. I chose to use both. So my C matrix is:

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

As always,

$$D = \begin{bmatrix} 0 \end{bmatrix}$$

Finally, this gives us the equations

$$\begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \ddot{x} \\ \ddot{z}_e \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ -8.77 & 0.80 & -0.015 & -0.17 & 0.1 \\ 120.6 & 24.2 & 1.43 & -19.9 & 1.1 \\ -56.4 & -80.64 & 0.83 & 9.18 & -5.39 \end{bmatrix} \begin{bmatrix} \theta \\ \phi \\ \dot{x} \\ \dot{z} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0.0122 \\ 0.198 \\ -0.658 \end{bmatrix} \begin{bmatrix} \dot{\phi} \end{bmatrix} \tag{5}$$

and

$$y = \begin{bmatrix} \theta \\ \phi \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \phi \\ \dot{x} \\ \dot{z} \\ \dot{\theta} \end{bmatrix} \tag{6}$$

To check for controllability, `length(A)= rank(ctrb(A,B))=5`, which indeed, it does. The controllability matrix W, is defined as:

$$W = \texttt{ctrb(A,B)} = \begin{bmatrix} 0 & -0.6584 & -75.28 & 622.6 & -4163 \\ 1 & 0 & 0 & 0 & 0 \\ 0.01215 & 0.7023 & -5.031 & 815.3 & -6319 \\ 0.1975 & 19.56 & -550.5 & 2559 & 2.072e+04 \\ -0.6584 & -75.28 & 622.6 & -4163 & 1.145e+04 \end{bmatrix} \tag{7}$$

$$5 = \texttt{rank(W)= length(A)}$$

However, the system is asymptotically unstable! Meaning that not all of the eigenvalues of A, `eig(A)`, are real and negative. Since not all of the eigenvalues have negative real parts, then the glider will more then likely nose-dive, or stall. This tells us that we must design a controller for our system in order to improve the glide distance.

$$\texttt{eig(A)} = \begin{bmatrix} -15.17 \\ -10.28 \\ 0.07 + 1.36i \\ 0.07 - 1.36i \\ 0 \end{bmatrix}$$

Now, we check to see if our system is observable. To so, we can use the command `obsv(A,C)` and compare its rank to the length of C.

$$O = \texttt{obsv(A,C)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ -56.4 & -80.65 & 0.826 & 9.18 & -5.39 \\ 0 & 0 & 0 & 0 & 0 \\ 1403.1 & 657.1 & 8.63 & -232.2 & -17.2 \\ 0 & 0 & 0 & 0 & 0 \\ -27102 & -4223 & -345.7 & 4462 & 1241 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{8}$$

`rank(O)= length(C)= 5`

3 Design a Controller

We cannot design a controller in the same way as the last 2 projects. That is because we do not know what the actual state of the system is. We only have the sensor data y, which we can use to estimate x.

To Find K,

$$K = lqr(A, B, Q_c, R_c) \tag{9}$$

where

$$Q_c = 1.8 \begin{bmatrix} 200 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 140 \end{bmatrix}, R_c = 0.01$$

Similarly,

$$L = lqr(A', C', R_o, Q_o) \tag{10}$$

where

$$Q_o = 0.01 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, R_o = 0.1 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{bmatrix}$$

These are the values that I tuned to after much experimentation. This gives us:

$$K = \begin{bmatrix} -855 & 86.7 & 41.8 & -10.1 & -163.6 \end{bmatrix}$$

$$L = \begin{bmatrix} 6.1 & -1.8 \\ -1.8 & -2.6 \\ -1.6 & 5.4 \\ 28.7 & -4.9 \\ 14.98 & -15.5 \end{bmatrix}$$

After checking for stability, we find that `eig(A−BK)` has negative and real part eigenvalues, so the closed loop system is asymptotically stable!

$$\texttt{eig(A−BK)} = \begin{bmatrix} -97.8 + 60.4i \\ -97.8 - 60.4i \\ -0.52 \\ -5.06 \\ -17.13 \end{bmatrix}$$

Checking for controllability, Finally, we can implement our controller. We start by specifying

our initial guesses of the state.

$$\hat{x} = \begin{bmatrix} \hat{\theta} \\ \hat{\phi} \\ \hat{\dot{x}} \\ \hat{\dot{z}} \\ \hat{\dot{\theta}} \end{bmatrix} = \begin{bmatrix} sensors.theta \\ sensors.phi \\ 6 \\ 0 \\ 0 \end{bmatrix}$$

For reference tracking, we choose a kRef such that

$$kRef = \texttt{1/(-C*inv(A-B*K)*B)} = \begin{bmatrix} 0 & 2061.7 \end{bmatrix} \tag{11}$$

I chose a reference

$$r = \begin{bmatrix} \frac{senors.theta}{10} \\ 0 \end{bmatrix}$$

Such that anytime we have a positive elevation angle of attack, we control $\dot{\phi}$ to lower it back to equilibrium, 0. We implement this in our controller with:

```
function [actuators,data] = runControlSystem(sensors,references,parameters,data)

data.r = [.1*sensors.theta;0];
h=1/100; % Time-Step
y= [sensors.theta; sensors.phi]- data.C*data.xe;
u= -data.K*data.xhat +data.kRef*data.r;
data.xhat=data.xhat...
         +h*(data.A*data.xhat+data.B*u-data.L*(data.C*data.xhat-y));

actuators.phidot = u;
end
```

Figure 1: Controller Code

4 Implement and Test Your Controller in Simulation

To test the controller, I run it 1,000 times, take the final distances, and find the average, median, and standard deviations. To record the data and plot the path-line of each simulation, I run a simple `for loop`:

```
for i=1:n
    DesignProblem03('Controller','datafile','data.mat','display',false);
    load('data.mat');
    x(i)= processdata.x(end);
    t(i)= processdata.t(end);
    figure(1)
    plot(processdata.x,processdata.z)
    grid on
    hold on
end
```

Figure 2: Storing Data

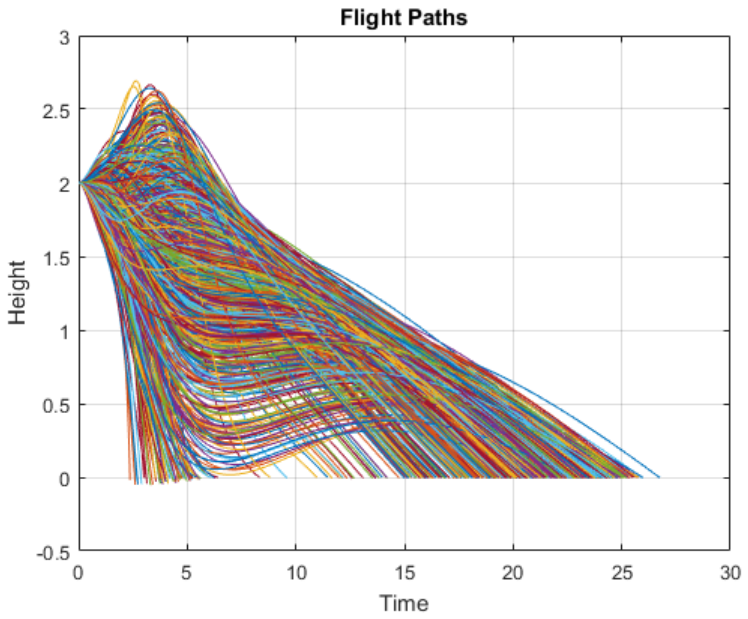

Figure 3: Flight Paths

We then find the maximum, mean, median, and standard deviation.

$$x_{max} = \texttt{max(x)} = 26.63$$

$$\bar{x} = \texttt{mean(x)} = 20.23$$

$$x_{median} = \texttt{median(x)} = 21.59$$

$$\sigma = \texttt{std(x)} = 4.8625$$

Plotting a histogram,

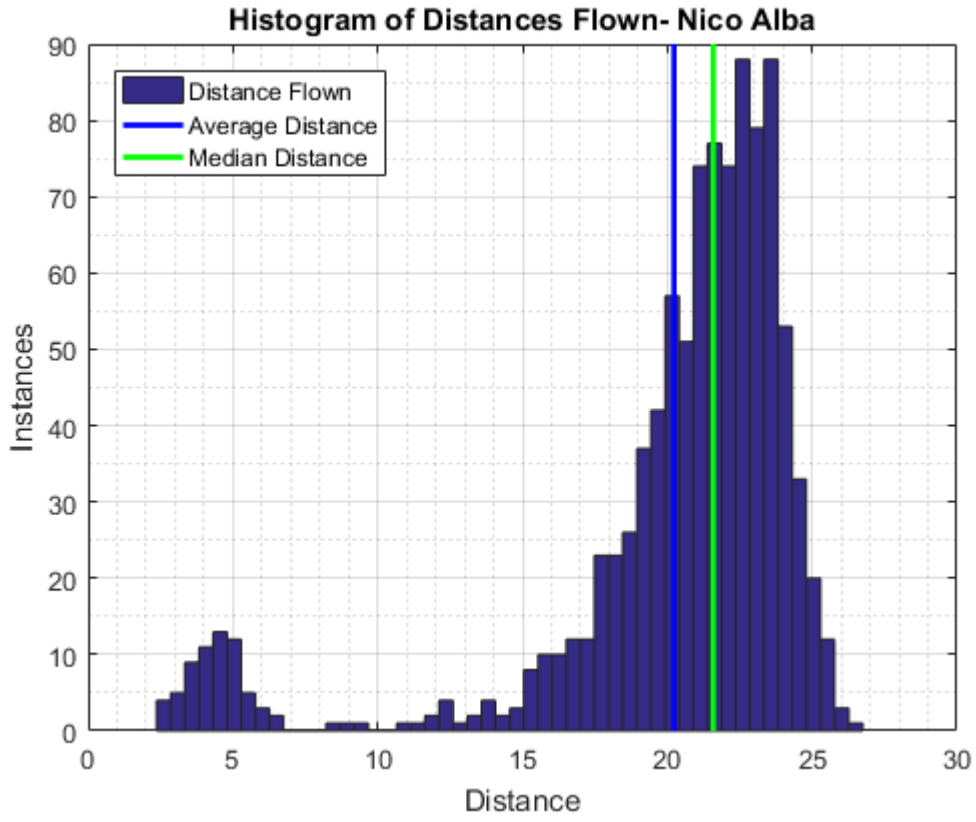

Figure 4: Histogram of Flight Distances

So even though the average and median were both above 20 meters, does this mean that the controller was successful?

Yes! Kind of... Since our standard deviation is 4.86, about 68% of the time, the glider will land in between 15.4 and 25.1 meters.

What would happen to the mean, median, and standard deviation if we were to eliminate all of the instances that the glider took a nose-dive? These are the instances in which the initial guess of the state was so off that our controller could not stabilize in time.

```
j=1;
for i=1:length(x)
    if x(i)>10
        z(i)=x(i);
        x_new(j)=z(i);
        j=j+1;
    else
        z(i)=nan;
    end
end
```

Figure 5: Singling out anything above 10 meters

Now the histogram looks like:

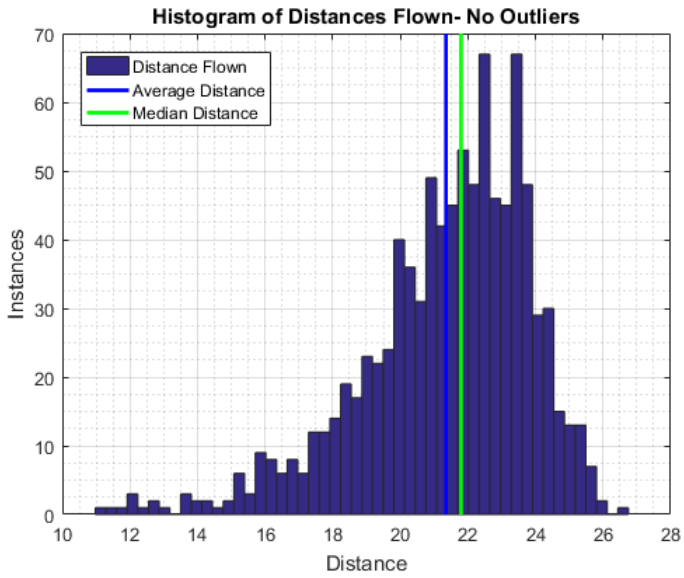

Figure 6: Flight Distances without "Nose-Dives"

The new mean, median, and mode are:

$$x_{new,max} = \texttt{max(x)} = 26.63$$

$$\bar{x}_{new} = 21.35$$

$$x_{new,median} = 21.8$$

$$\sigma_{new} = 2.53$$

With an increase in average and a decrease in standard deviation, our new 68^{th} percentile lies in between 18.8 and 23.88, which is much tighter interval.

5 Discussion of Results

After a lot of tweaking of $Q_c, R_c, Q_o, and R_o$, I was able to get a very consistent system. For the most part I was able to remove most of the "nose-dives" that gave so many people a binomial distribution. But I was getting an average of about 19 meters. After a lot of tuning, I was able to maintain the spirit of a normal distribution to increase the average of the glider distance. However, this meant that my controller could not correct enough if the initial guesses were very off. And because of this, you can see a small amount of instances of a normal distribution of very low values, which *increases* the standard deviation, which is expected.

And though I was able to improve the consistency of glide distances, my controller was never able to reach a distance of at least 30 meters.

6 References and Acknowledgments

I would like to preface that I have no idea how you want this section structured, nor do I care because it is not in the rubric. So this is purposefully going to be a lazy section. I just want to properly recognize the people who have helped me through this Design problem

[1] Tim Bretl: Provided code to get the equations of motions and find equilibrium values.

[2] Isabel Anderson: Helped me linearize the system to find A,B,C,D.

[3] Jacob Heglund: Helped me find my K and L matrices by properly using `lqr` and having the correct Q_o.

[4] Allery Hsu: Gave me ideas for how to tune Q_c, R_c, Q_o, R_o

[5] Nick Zuiker: Help me implement reference tracking to have a smoother trajectory.

[6] Logan Brodhead and Jason Pike: General questions and putting up with my nuisances.