AE353: Design Problem 03

T. Bretl

October 26, 2018

1 Goal

The code provided in `DesignProblem03` simulates an unpowered glider that is similar to one used for experiments on fixed-wing perching [1, 2]. This glider has one control surface, an elevator. An actuator allows you to specify the angular rate of this 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 a forward speed of about six meters per second.

2 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:

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

Experimental results show that this flat-plate model is a good approximation for the glider and that it remains accurate at high angles of attack and even post-stall [2]. The function f in (1) depends on a number of parameters (e.g., mass, moment of inertia, surface area of the wing). You can access symbolic descriptions of this function either within your controller code (`Controller.m`) as `parameters.symEOM.f`, or in a separate piece of code that you write for the purpose of control design. In particular, the first time you run `DesignProblem03`, it will create a file that you can load to access the equations of motion. Here is an example:

```
% 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.)
```

3 Tasks

3.1 Analysis

The focus of your analysis this time will be on data collection. The initial conditions in the simulation are random. So, for a given control design, the flight distance will vary (perhaps significantly). It is not enough to look at the results of one flight—you will have to look at the results of many flights. At minimum, for each design that you consider, you should do the following:

- Collect data from at least 1000 flights.
- Compute the minimum, maximum, median, mean, and standard deviation of the flight distance (e.g., `min`, `max`, `median`, `mean`, and `std` in MATLAB).
- Plot a histogram of the flight distance (e.g., `hist` in MATLAB).

Remember that `DesignProblem03` provides options to save data and to turn the graphics off, both of which are very useful for data collection. Here is an example of how to use these options to collect and analyze data from a large number of flights.

```
% Number of flights
nFlights = 1e3;
% Loop over each flight
for i=1:nFlights
    % Run simulation without graphics and save data
    DesignProblem03('Controller','datafile','data.mat','display',false);
    % Load data
    load('data.mat');
    % Get t and x
    t = processdata.t
    x = processdata.x
    % Do analysis...
    %     (your code here)
end
```

3.2 Presentation

The focus of your presentation this time will be on equations. For example, you should be sure to do the following:

- Number any equation to which you will refer, and do the reference with `\eqref`.
- Use standard notation for operations like multiplication (e.g., Ax for "A times x" and not $A \times x$ or $A * x$).
- Place square brackets around matrices. For example,

```
\begin{bmatrix} \ddot{x} \\ \ddot{z} \\ \ddot{\theta} \end{bmatrix}
```

  produces the standard notation

$$\begin{bmatrix} \ddot{x} \\ \ddot{z} \\ \ddot{\theta} \end{bmatrix},$$

  while

```
\begin{matrix} \ddot{x} \\ \ddot{z} \\ \ddot{\theta} \end{matrix}
```

  produces the non-standard and confusing notation

$$\begin{matrix} \ddot{x} \\ \ddot{z} \\ \ddot{\theta} \end{matrix}.$$

- Align equations properly. For example,

```
\begin{align*}
c_{L} &= 2\sin\alpha \cos\alpha \\
c_{D} &= 2\sin^{2}\alpha
\end{align*}
```

  produces the nice-looking result

$$\begin{aligned} c_L &= 2 \sin \alpha \cos \alpha \\ c_D &= 2 \sin^2 \alpha \end{aligned}$$

  while

```
\begin{equation*}
c_{L} = 2\sin\alpha \cos\alpha
\end{equation*}
\begin{equation*}
c_{D} = 2\sin^{2}\alpha
\end{equation*}
```

produces the messy-looking result

$$c_L = 2 \sin \alpha \cos \alpha$$

$$c_D = 2 \sin^2 \alpha.$$

- Use the correct font for functions like cosine (`$\cos\alpha$` produces a nice-looking $\cos \alpha$, while `$cos\alpha$` produces a messy-looking $cos\alpha$).

Please be responsive to feedback on your draft reports—this feedback will focus on helping you improve your presentation of equations.

3.3 Process

Your report will include the following four parts:

- Define one requirement and one verification.
- Derive a model (i.e., linearize (1) and express your result in state-space form).
- Design a controller and an observer (remembering to check, first, that your linearized model is both controllable and observable).
- Implement and test your control design in simulation. Follow the instructions you wrote to verify that your requirement has been satisfied. Characterize the performance of your control design by looking at aggregate results, as described in Section 3.1.

As usual, you are encouraged to go beyond these requirements. For example, it is likely that you will want to iterate on your design, in order to maximize flight distance. In this case, it would be helpful to describe the process that you used to improve your design.

4 Deliverables

You must submit two things by 11:59PM on Friday, November 9:

- Code. This code will be written in MATLAB, using the template `Controller.m`. Please specify how to run your code (e.g., `DesignProblem03('Controller')`).
- Report. This report will be written in LaTeX. You will submit both a PDF document and the LaTeX source files that you used to produce this document. The PDF document must be exactly four pages, and is expected to include a method of approach and a description of results. *Your report must conclude with a section titled "Acknowledgements" that lists the colleagues with whom you worked (if any) and describes the nature of your collaboration (discussion, sharing code, etc.).*

You must also meet intermediate deadlines, the details of which will be posted to reddit. *Late submissions of either draft reports or the final report will not be accepted.*

5 Evaluation

Your work will be evaluated based on on completion of the requirements (40%), on correctness of your code (10%), and on presentation of results in your report (50%). Our evaluation of your technical approach will place special emphasis on data collection and analysis. Our evaluation of your report will place special emphasis on your presentation of equations—we will not look at style, grammar, or any other aspect of the text other than equations, as long as there is no barrier to understanding your work.

References

[1] J. W. Roberts, R. Cory, and R. Tedrake, "On the controllability of fixed-wing perching," in *American Control Conference*, Jun. 2009, pp. 2018–2023. [Online]. Available: http://dx.doi.org/10.1109/ACC.2009.5160526

[2] J. Moore, R. Cory, and R. Tedrake, "Robust post-stall perching with a simple fixed-wing glider using lqr-trees," *Bioinspiration & Biomimetics*, vol. 9, no. 2, p. 025013, 2014. [Online]. Available: http://stacks.iop.org/1748-3190/9/i=2/a=025013