AE353: Design Problem 02

T. Bretl

February 20, 2019

1 Goal

The code provided in `DesignProblem02` simulates a "gravity-assisted underactuated robot arm" that is similar to one proposed for wing-box assembly in aircraft manufacturing [?]. This robot arm has two joints, but only one motor. The motor applies torque about the first joint. The second joint spins freely. Optical encoders measure joint angles and (after some processing—taking a finite difference, for example) joint velocities. The goal is to make the tip of the robot arm move to a sequence of points in space. Equivalently, the goal is to make the second joint angle (the one about which you cannot apply a torque) track a piecewise-constant reference trajectory.

2 Model

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

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + N(q,\dot{q}) = \tau, \tag{1}$$

where

$$q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$

is a matrix of joint angles,

$$\dot{q} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

is a matrix of joint velocities,

$$\tau = \begin{bmatrix} \tau_1 \\ 0 \end{bmatrix}$$

is a matrix of applied torques, and

$$M(q) \qquad C(q,\dot{q}) \qquad N(q,\dot{q})$$

are matrix-valued functions of q and/or $\dot{q}$. These functions depend on a number of parameters (e.g., mass and moment of inertia). You can access symbolic descriptions of these functions either within your controller code (`Controller.m`) as `parameters.symEOM`, or in a separate piece of code that you write for the purpose of control design. In particular, the first time you run `DesignProblem02`, it will create a file that you can load to access the equations of motion. Figure 1 shows an example of how to do so.

```
% Load the equations of motion.
load('DesignProblem02_EOMs.mat');
% Parse the equations of motion.
M = symEOM.M;
C = symEOM.C;
N = symEOM.N;
tau = symEOM.tau;
% Define symbolic variables that appear in the equations of motion.
syms q1 q2 v1 v2 tau1 real
% Proceed to linearize or do whatever else you like...
%       (see: help sym/jacobian, help sym/diff, help sym/subs, etc.)
```

Figure 1: Example code to get a symbolic description of the equations of motion in MATLAB.

3 Tasks

You will focus this time on requirements and verifications.

3.1 Requirements

A *requirement* is a property that the system you are designing must have in order to solve your problem (i.e., a thing that needs to get done). A good requirement is quantifiable—it involves a number that must be within a certain range in order to solve your design problem. A good requirement is also both relevant (it *must* be satisfied—it is not optional) and detailed (it can be read and understood by anyone). Here is an example of a requirement that says what needs to get done but that most engineers would consider unacceptable:

> The joint angle shall track a reference trajectory.

This requirement is not detailed. One way to improve it would be to say which joint angle, to say what reference trajectory, and to say what it means to track this trajectory:

> The second joint angle shall remain close to a constant reference value.

This requirement is not quantifiable. One way to improve it would be to say how close:

> The second joint angle shall remain within $\pm 1^\circ$ of a constant reference value.

Most engineers would argue that this requirement still needs improvement. How long must the joint angle remain close to the reference value? Is there an upper bound on the time it must take for the joint angle to get close to the reference value, assuming it starts at some other value? Must this requirement be satisfied for any choice of reference value, or just values within a specific range? These are examples of reasonable questions that might be asked by an engineer reviewing the requirement. Your task is to define *one* requirement—that is quantifiable, relevant, and detailed—that makes clear what the system you are designing must do in order for your goal to be achieved.

3.2 Verifications

A *verification* is a test that you will perform to make sure that the system you are designing meets a given requirement. A good verification is based on a measurement—it checks that a quantity is in the range specified by the requirement. A good verification also has a set of instructions for how to make the measurement (an experimental protocol) and for how to interpret the results (methods of data analysis and visualization that provide evidence the requirement has been met). Consider the requirement from Section 3.1 (which, as we have said, still needs improvement):

> The second joint angle shall remain within $\pm 1^\circ$ of a constant reference value.

Here is a verification of this requirement that most engineers would consider unacceptable:

> The system will be tested in simulation.

This verification is not based on a measurement. Here is a better version that *is* based on a measurement:

> The error between the second joint angle and the constant reference value will be found in simulation.

This verification does not include a set of instructions for how to make the measurement or for how to interpret the results. Here is a better version that *does* include a set of instructions:

> The MATLAB function `DesignProblem02('Controller','datafile','data.mat')` will be be used to simulate the system. The data generated by this function will be imported into a MATLAB program for analysis. The error at each time step will be found by taking the difference between the second joint angle and the reference value. The maximum absolute value of error over all time steps will be reported—if it is less than 1°, the requirement is met.

Most engineers would argue that this verification still needs improvement. For example, does the simulation generate the same results every time, or is there variation? You know that there is variation because, for example, the initial joint angles and joint velocities are selected at random. A reasonable engineer, then, would question whether or not the results of one simulation would really show that the requirement is met. Many verifications also provide more than just a single number as evidence—for example, they might produce a figure (e.g., a plot of error as a function of time) or some other type of visualization. Your task is to define *one* verification for your requirement that has a measurement and a set of instructions for how to make the measurement and how to interpret the results.

3.3 Process

Your report will include the following four parts:

- Define one requirement and one verification, as described in Sections 3.1-3.2.
- Derive a model. At minimum, you should linearize (1) about some choice of joint angles and joint velocities, expressing your result in state-space form.

- Design a controller. At minimum, you should determine if the open-loop linear system is controllable, determine if the open-loop linear system is asymptotically stable, consider the application of state feedback, determine if the closed-loop linear system is asymptotically stable, and find the steady-state error in reference tracking (both with and without disturbance) of the closed-loop linear system.

- Implement and test your controller in simulation. At minimum, you should follow the instructions you wrote to verify that your requirement has been satisfied.

As usual, you are encouraged to go beyond these requirements.

4 Deliverables

4.1 First draft

You must submit one thing by 11:59PM on Wednesday, February 27, 2019:

- A PDF document that was generated using LaTeX and that includes, at minimum, a draft of your requirement and verification and of your state-space model.

Please submit this draft report at the following URL:

`https://forms.illinois.edu/sec/6782858`

You may submit as many times as you want—only the latest submission will be recorded. So please, submit early and often.

4.2 Second draft

You must submit one thing by 11:59PM on Wednesday, March 6, 2019:

- A PDF document that was generated using LaTeX and that includes, at minimum, a draft of your requirement and verification, of your state-space model, of your control design and analysis, and of your results in simulation.

Please submit this draft report at the following URL:

`https://forms.illinois.edu/sec/6288613`

You may submit as many times as you want—only the latest submission will be recorded. So please, submit early and often.

4.3 Final report

You must submit four things by 11:59PM on Friday, March 8, 2019:

- Code — Controller and Test. The controller code will be written in MATLAB, using the template `Controller.m`. The test code will be written in MATLAB, using the template `Test.m`. If `Test.m` is executed by one of your peers in a folder with your controller code `Controller.m` and the simulation code `DesignProblem02.m`, it will produce results that verify your requirement has been satisfied.

- Report — PDF and Source Files. 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. The source files must be in a ZIP folder and must include all figures, references, etc., in addition to the `.tex` file.

Please submit this final report at the following URL:

https://forms.illinois.edu/sec/1616271

You may submit as many times as you want—only the latest submission will be recorded. So please, submit early and often.

5 Evaluation

Your work will be evaluated based on meeting intermediate deadlines (10%), on peer reviews both of your code (20%) and of your report (60%), and on your peer reviews of other reports (10%). Reviews of your technical approach will place special emphasis on requirements and verifications. Reviews of your report will place special emphasis on figures—reviews will not critique style, grammar, or any other aspect of the text other than figures, as long as there is no barrier to understanding your work.

Peer review will be "double-blind." You won't know who reviewed your report, and your reviewers won't know whose report they reviewed. To enable a double-blind review process, it is **very important** that your final report be completely anonymous. To repeat, **DO NOT INCLUDE YOUR NAME** or anything else that would identify you in any of the materials you submit—not in your PDF, in any of the files in your ZIP folder, or in either of your two MATLAB scripts. If you do include your name, your report will be returned without review, and you will receive a zero score. No exceptions will be made. Other details of peer review will be posted to piazza.

Acknowledgements

The notes on requirements and verifications are based on a conversation with Professor Scott Carney (Electrical and Computer Engineering) and Dr. James Norton (Neuroscience Program). Some of the text used in defining requirements and verifications was taken from lecture notes for ECE398PSC (https://courses.engr.illinois.edu/ece398psc/docs/slides/ece398psc_SP17_Lecture6_postedversion.pdf) and ECE445 (https://courses.engr.illinois.edu/ece445/guidelines/requirements-and-verification.asp). Prof. Bretl would like to thank Prof. Carney and Dr. Norton for their help.