Chapter 1

Lab #7: State Space Compensator Design

Objective

The purpose of this lab is to expose you to state space design techniques. In particular, you will use LQR techniques to design controllers and estimators. Because you are now all largely familiar with LabVIEW, the description of this lab will be kept rather brief. However, keep in mind that this is nevertheless a reasonably long lab—don’t put off the work until the second week!

Note that this lab will be the last lab.

Warning:

As usual, please always run the ECP unit with someone holding the power button, in case something happens to go wrong.
CHAPTER 1. LAB #7: STATE SPACE COMPENSATOR DESIGN

1.1 Pre-Lab Tasks

Read this entire document before starting the lab.

1.2 Tasks

1. The code that you will see will be found in students-simulation.vi, students-hardware.vi, and students-estimator.vi. You will only need to edit the mathscript in students-simulation.vi and students-estimator.vi.

2. Note that you will only change things in MathScript in this lab. Please do not change any of the graphical LabVIEW code except for rewiring the diagram to change full state feedback to estimated state feedback (as seen in Fig 1.2).

3. Lastly, you will be given values for \( c_{1,2,3} \) and \( k_{1,2} \) for each torsional disk system. Also, the hardware gain is now calculated within the code—you don’t need to address it.

1.2.1 Task #1–State Space simulation

Task: Using the three second-order differential equations used to derive your transfer functions, that you have been using in the previous labs, derive the state-space model of the torsional system. It should become clear that the state dynamics (\( \dot{x} = Ax + Bu \)) will be the same for all 3 disks while the \( C \) matrix of the output equation (\( y = Cx \)) will different. Although state-space representation allows for multi-input and multi-output (MIMO) models, we will deal with single-input single-output (SISO) models.

Task: Copy the folder state-space from Desktop/ITLL Modules/ECEN Control Systems to your Z: drive.

Task: In students-simulation.vi, replace the second-order model with your model of the torsional disk system. In students-estimator.vi, change the state feedback gain \( K \) (near “[MARKER 1]”) to be the appropriate length vector (e.g., it should match the dimension of the state model) made up of zeros. On the last line (near “[MARKER 3]”), set “Nbar=1”—this will allow you to track a reference input. (This hack will allow your code to compile, though there will be some errors due to the LQR control and estimation calculations not matching the dimension of \( K \). You will learn by next week what Nbar is and how to compute the controller, estimator, and Nbar without errors.)

Task: Simulate some open loop response (using the top disk as the output) with nontrivial initial condition to verify that your model is functioning properly.
1.2. TASKS

1.2.2 Task #2–Controller Design

Now that you have a model working, you should not need to change the student-simulation.vi file any more. Instead, everything else should occur within the student-estimator.vi file, where both the controller and estimator are designed. First you will design a state-space controller for stabilizing the top disk, then you will design a state-space estimator. For the controller, you should assume you have access to full state feedback—make sure the block diagram reflects this.

Task: Implement a proportional controller and a PD controller in state space form. (I.e., You will have to replace your $K$ of zeros with your own definition of $K$ that implements a PD control law.) Simulate the closed loop response. Is it sensitive to changes in system parameters? Disable the reference input and study your controller’s response to initial conditions only by setting “Nbar=0” in students-estimator.vi.

This is probably a good stopping point for the first week. However, you should have done most of the simulation components of this lab by the next time you meet.

Task: Design a controller using LQR (“lqr()” in MATLAB/MathScript)) that keeps the total control effort for a 1 rad step response less than 2 Nm with a rise time of $t_r < 0.75s$, overshoot $M_p < 25\%$, $e_{ss} < 5\%$, and settle time $t_s < 2s$. To track a reference, you will need to comment out the “[MARKER 3]” line so that Nbar is not disabled.

1.2.3 Task #3–Estimator Design

Task: Rewire the simulation so that it is using estimated state for purposes of control calculation, not the actual state.

Task: Using the top disk as the output, design a state estimator by placing the poles at desired locations using the command “place()”.

Task: Simulate the system using your controller and the estimated state. Is your performance as good? (You will have to comment out the line at “[MARKER 2]”.)

Task: Using the formula for the controller $C(s)$ given controller $K$ and estimator $L$, calculate the phase margin and gain margin of your system?

Task: Does your controller still work when you vary parameters somewhat? If it does, you need to iterate!

1.2.4 Task #4–Hardware Test

Task: If you are confident that your controller is working properly in simulation, verify it in hardware. This can be done using students-hardware.vi. Set the hardware gain for your
plant in the VI’s wiring diagram and run it—it will reference the students-estimator.vi you have already created.

**Task:** What happens if you change the sample rate from $200Hz$ to $500Hz$? to $1000Hz$? Why?

Remember, if you get stuck on some part of the lab, ask your classmates, the TA, or myself.
1.2. TASKS

Your system model should go here. This represents the actual plant. You can vary these parameters while leaving your estimator untouched to explore robustness.

```matlab
A = [ -1; 0; -1];
D = 0;
B = 0; [1 0];
C = [1 0];
sys = ss(A, B, C, D);
```

This icon represents a "SubVI." SubVIs are similar to functions in other programming languages. They let you divide your code into self-contained components. This particular SubVI creates your estimator and controller. By using a SubVI, we are able to use the same code for the simulation and hardware VI. Double-click on the icon to open the SubVI.

The returned values are stored in hidden front panel indicators and then read using local variables. This makes the code a little cleaner since there are no wires running all over the screen.

Figure 1.1: System Model
Figure 1.2: State Space Simulation
1.2. TASKS

Figure 1.3: State Space Estimation and Control