VCSFA/MARS

8/10/17

Jake Singh

***PID Simulation in MATLAB***

**How I made the simulation**

As a part of my capstone projects, I wanted to explore the idea of simulating a PID control system on the pad. Being able to simulate of our EREGs would be helpful for engineers to get familiar with PID behavior and develop a plan before a new tuning session so as to minimize time and resources spent on trial-and-error tuning.

***Identifying the system***

Simulating a dynamic system such as the GN2 pressure lines would be difficult to do relying just on physical principles and physics equations. A pressure system is also subject to many nonlinearities, and even a small change in volume in any part of the regulator, valve, or piping would alter any pressure calculations. Having experimented with the Controls/System Identification toolbox in MATLAB, I decided that this software could be useful in identifying a mathematical input-output model of the EREG system. System identification would be the first step in simulation. Our modeling will be data-driven rather than equation-driven.

***Data collection***

To collect sufficient data for making a model, I wanted to capture as much of the system dynamics as possible. This would simply mean changing the setpoint up and down several times across the full range of 0-100 psi and waiting to reach steady state after each change. Around the time I needed to collect data, Joel happened to have paperwork written for working on the ER5531 EREG in the GN2 system, so he allowed me to go out to the pad with him and collect data while we changed the setpoint. On the first day of data collection, we had a sample rate of 0.5 seconds and collected time, setpoint (input), compensated external sensor, and exterior feedback (output). We also recorded the P, I, D values the EREG was tuned to at the time. The data was sufficient, and I was using a step response from 0-100 psi to do the modeling.

After messing around with this data for a while, I decided that I wanted to try to get better and more complete data. We went out for a second day of data collection and this time used the highest frequency sample rate available (0.1 samples/second) and additionally collected internal pressure sensor data. We used a longer sampling window and collected a response to 20 psi increments up to 100 psi, a decrement back to 0 psi, and a final step from 0 to 100 psi. This data set is what was used to make the first MATLAB program.

***System Identification in MATLAB***

The System Identification (SysID) toolbox can automatically generate a transfer function (or other types of models, like state-space) for a system given time domain input and output data. The data can be a step response, pulse response, or completely arbitrary. Because of the imprecise nature of the data output from the EREG (a setpoint change is not precise or instant and the setpoint value actually oscillates with a very small amount of noise), I decided the arbitrary data option would be simplest and most accurate. To create a model, one can either open the SysID app *or* the PID Tune app in MATLAB. Because the SysID and Controls toolboxes are tightly integrated, one can do the necessary system identification all through the PID tune app. I actually find it to be easier to do through the PID Tune app because for some reason, this route allows for the input and output signals to be timeseries objects, while the other does not. This way you can also choose between a few different combinations of poles and zeros and quickly check each setup for accuracy. Opening the SysID app lets you specify the type of model and the number of poles and zeros at the beginning, but then you must start over if, say, you picked two zeros and saw that the fit accuracy was too low.

To make the model, I first converted the raw .delim file to an excel spreadsheet and checked for accuracy by making a few quick plots. I then imported the excel file into MATLAB and created column vectors for each measured quantity. Finally, I made timeseries objects for input and output using the time, and input and output signals. To use the timeseries for modeling purposes, I had to open each one and apply the “uniform time vector” option (the raw data isn’t recorded at a perfectly uniform rate).

***Making the Model***

To make the actual transfer function model, I had to decide how to approach two problems: 1) what to choose as the input signal and 2) how to deal with the fact that I did not have open-loop data (I obviously would not be able to record data on the pressure system without the EREG connected and running). The first problem was a matter of deciding what made up the “plant”. Physically, the real measurable input is probably the internal (pilot) pressure which actually opens the valve to the desired position and thus creates the required downstream pressure. However, after several attempts to make a working transfer function in this way, I decided to use the setpoint as the input. I think this is actually the best way to make the model because the setpoint is the true user input to the system, the PID settings correspond to the setpoint signal, and the setpoint leads to a voltage applied to the internal pressure input/exhaust vents. The internal pressure is actually also controlled by a mini “internal PID loop”, but this loop has been blackboxed due to its small impact and the fact that the internal pressure is essentially directly proportional to the downstream pressure feedback.

The second problem was figuring out how to create the transfer function for the actual plant, not just the plant connected to a controller. To deal with this, I decided to set the transfer function generated from the data (which ended up having just a gain and one pole) equal to the equation for the transfer function of a plant controlled by a PID controller. The equation is:

Gcontroller\*Gplant/(1+ Gcontroller\*Gplant)

Also, the equation for Gcontroller is:

KDs+KP+KIs-1= Gcontroller

We know the values for KD, KP, and KI because we recorded them during the data collection session. They corresponded to 150, 1500, and 200 respectively. Leaving only Gplant unsolved, I solved for it to reach a model of the plant itself. I then recombined this new equation with Gcontroller (except in variable form this time) to match the form of the original plant-controller equation. After simplification, I now had an overall plant-controller transfer function containing the variables KD, KP, and KI. This meant I could programmatically adjust them and see how the step response of the transfer function was affected.

**How to improve and develop the simulation program**

***The GUI***

The existing GUI is solid for now, but can definitely be changed or improved as needed. The GUI is linked to the main MDRT review window and is controlled by a file called PIDSimulator. The file opens and populates a figure in the same way that filterFDTool does. All the functions and callbacks are contained in the same file. The GUI layout was designed in GUIDE but then coded out rather than auto-generated. The GUI will have to be updated as issues are fixed (see issues on GitHub). For example, the business logic was easy to write since I only had one system loaded into the program. The logic will need to be re-written once more systems are uploaded.

***Modeling***

The methodology for modeling the system can of course be subject to debate. I believe the SysID toolbox method is a good one (the generated transfer function models I used always had a fit accuracy of around 95%) because of the difficulty of modeling a pressure system purely from physical equations and relationships. Had the system been actuated electromechanically and not by muscle pressure, then perhaps using equations would be easier. My lingering question is whether the generated transfer function can so easily be simplified to KDs+KP+KIs-1= Gcontroller and Gplant. I spoke to a TESCOM technical representative on the phone and asked him if he had any insight on the problem, but he just told me that the PID gains aren’t meant to correspond to any physical parameter and that he could not offer any other insight on the problem. I think that the values are maybe just based on the setpoint vs. error feedback.

When I used the same methodology described for creating a model but using pilot pressure as the input, I could not get a stable plant model with any combination of poles and zeros that I tried. When I would decrease the proportional gain, I actually had increased overshoot in the simulated step response.

As the summer is ending, I didn’t have time to go back out to the pad and test the simulator against data using different PID values for the EREG. This would probably be the best way to test the simulator going forward.

***Simulation Controls***

The program currently allows the user to interactively change the PID values and see the resulting step response. The step response is currently scaled based on the first and second setpoints, so the magnitude will change but the time will not. This was just a visual aid meant to mirror the ERTune program and how the setpoint is actually adjusted in the field. I also threw in the option to perform an impulse response, but there is probably no practical need for this with our systems.