Simulation Methods: Double Integrator Example

In the last post I focused on placing the lead zero for the roll and pitch axes based on the limit imposed by a second double-pole our plant introduces via the motor-propeller, ‘A’ term. I neglected to calculate the proportional gain required for unity-gain crossover at the frequency of maximum phase margin. I also did not design the yaw-axis controller. The first short video completes these steps from last time. The rest of this post will cover some simulation tools and techniques we’ll be using.

Simulation tools and techniques

In the video you can see in the last post I mention the, “satellite attitude” control problem you can find in nearly all control system textbooks. This is basically the same, “plant” model as our roll, pitch, and yaw axes. However, by stripping away our extra poles and parameter values we can first make sense of the simulation tools and methods before applying the tools to our problem.

Satellite Attitude Control: classic double-integrator plant stabilization

Is space absent gravity or any impeding forces lateral thrust a moment-arms distance from the mass center on opposite of a

body is a, “couple”. It produces angular acceleration of the body when the moment-arm from each thruster is the same length from the mass center. The simplified model models the rotational thrusters as the only external torques on the body.

This torque results in angular acceleration that double-integrates to angular position. Hence the term, “double integrator” I suppose, but I don’t know who coined the phrase. In any case, this is a classic, “unstable plant” control problem. Let’s use this simple problem to set-up some simulation tools we will use for the quad-copter control simulations.

Textbook Problem Statement

The following PDF sets-up the basic problem as described in Digital Control of Dynamics Systems by Franklen, Powell, and Workman but you can likely find it in any control systems textbook.

fpw_satellite_attitude

Matlab Simulation Techniques

Matlab software is the go-to tool for control system design and simulation. You can see I use a neat math software tool called Maple as well, but this is fairly expensive and less common. Maple is awesome for creating mathematical, “documents” with in-line calculations as you have seen to this point.

I could probably push Maple to do a bunch more, but then I’d be producing example code that most of you couldn’t use. I’ll be using Matlab quite a bit now, so I can share M-files (the Matlab scripts) with you so you can play with them on your own.

A companion to Matlab is Simulink, which is useful for simulations. I chose the scripted ordinary differential

equation (ODE) solver method in a loop instead. It is less intuitive than Simulink, but will offer more flexibility as we get into the Quadrotor simulations. Under-the-hood Simulink uses the same ODE solver, but the, “graphical programming” can become a burden for all but the simplest simulations.

Video of simulation output

This video shows a space capsule initially pointing Up, but it’s assumed to be in outer space in a zero-gravity environment. When the simulation begins the capsule thrusters (illustrated in red) respond to a step-change in desired capsule orientation of 180-degrees. The simulated control system thrusts the capsule to the new desired orientation.

An actual, “satellite orientation” controller would be much more optimal relative to fuel burn. For example, fuel wouldn’t be wasted allowing the overshoot you see here. The burn trajectory would be pre-determined somewhat like an, ‘S’ to minimize the burn, and the servo method illustrated here would be left to operate in a very small window for final adjustment most likely. Perhaps we will return to this simple model if we explore optimal control topics later.

For now, this video shows what the Matlab code below does. If you run the Matlab M-file you should see what you see in this video, and from there you can modify the code as you learn.

Matlab M-Files

Download the following two files, place them in the same folder, and run the, “satellite” script. You’ll see the above plot output, and you can experiment from there by reading the comments in the script.

Main simulation script: satellite.m

The ODE function: ssodefun.m

Closing Remarks

The goal here was to frame-up some simulation and animation tools using Matlab. I used a simple, single-axis double-integrator, “textbook problem” to get the tools set-up.

From here I can proceed to introduce the multi-axis complexity of the Quadrotor with it’s extra poles and parameters. That’s next!