To: 42 Users
From: Eric Stoneking
Date: Oct 2017
Subject: Getting Started with 42 Flight Software Models

1 Introduction

Developing a control system is a laborious process. 42 has features that enable getting a quick-and-dirty simulation up and running rapidly. This is very useful for visualization and preliminary actuator sizing studies. For more in-depth studies and development of flight algorithms and software, quick-and-dirty needs to be replaced by thorough-and-rigorous. This note walks through the features and processes involved.

2 Global Variables and the FSW Structure

42 makes heavy use of data structures to organize and segregate information. Each spacecraft has an `SC` structure, and within that there is an `FSW` structure whose purpose is to contain all the data known to the spacecraft control flight software. See `42types.h` and `fswtypes.h` for the definitions of these structures.

The `FSW` structure is intended to contain all the data that the flight control software will need. In the early going, it may be expedient to use a global truth variable in a control law (eg. `SC.svn`) if the `FSW` equivalent hasn't been coded yet. 42 allows this — as shipped, `42fsw.c` includes `42.h`, which contains all the global variables. As the project matures from concept study to implementation, however, the flight code needs to be limited to the information that it actually will have in flight. To enforce this, comment out `#include "42.h"` and uncomment `#include "42fsw.h"`.

To assist in rapid prototyping, the as-shipped FSW structure has variables to support a variety of control architectures. The file `42fsw.c` also has a variety of notional control functions; some still run, some are prior exercises left in to be mined for ideas.

3 Flight Software Models

The source file `42fsw.c` is home to the model of the GNC flight software. As a project evolves, the model may grow to become the flight algorithm, or actual flight

code. In the beginning, though, it is vastly simplified, so we call it a model. It can be convenient to swap a model in or out, or to have multiple spacecraft running the same FSW functions. To facilitate that, we have the FSW "tag", `FswTag` in the `SC` structure. This is set in the spacecraft description file (eg. `SC_foo.txt`) as the Flight Software Identifier. The string entered there is converted in the `DecodeString` function to a constant (see `42defines.h` for defined values). This tag controls which function in `42fsw.c` is executed for that spacecraft. In the following paragraphs, we describe the tags you'll encounter first.

3.1 PASSIVE_FSW

Passive FSW does nothing. It exerts no forces or torques. It is most useful for studying natural motion, passively-stabilized spacecraft, and out-of-control (eg. decommissioned) spacecraft. In the `Demo` folder, the Moai, Big Black Monolith, and the three cubesats are using the passive FSW. BBM is demonstrating the natural motion due to an initial angular rate about the intermediate axis of inertia.

3.2 PROTOTYPE_FSW

As its name implies, prototype FSW is most useful for quick starts and very early studies. It uses inputs from the command script file (eg. `Inp_Cmd.txt`) to define a commanded attitude. See `Inp_Cmd.txt` for command templates and examples. The function `PrototypeFSW` then controls the attitude to comply with the command. In the `Demo` folder, the Shuttle and Ion Cruiser are using the prototype FSW. In the InOut folder, `SC_Simple.txt` is using prototype FSW.

3.3 AD_HOC_FSW

This model is intended as the best jumping-off point for customization. As shipped, it is simply a proportional-derivative (PD) linear feedback to align the spacecraft with the inertial N frame, using ideal torquers for actuation. It is up to you to modify the sensor inputs, commanded attitude, control law, and actuator commanding to fit your situation. In the `Demo` folder, Voyager is running ad hoc FSW.

3.4 THREE_AXIS_FSW

This model is an example with a few more realistic features in it. The commanded attitude is aligned with the local vertical-local horizontal (LVLH, or L) frame. Wheels

and magnetic torquers are used for actuation. There is a single-axis solar array gimbal that tracks the Sun. In the `Demo` folder, Aura is running this three-axis FSW.

3.5 And Beyond

As a project matures, you will outgrow any of the above models, and will want to make your own. Here are some things you should know.

To add a new FSW tag, changes need to be made in four places:

1. In your `SC_foo.txt`, introduce your new Flight Software Identifier, `FOO_FSW`.
2. In `42defines.h`, find the list of FSW Tags, and add `FOO_FSW` to it, with a unique integer.
3. In `42fsw.c:FlightSoftWare`, add a new case to call your new function, `FooFSW`.
4. In `42init.c:DecodeString`, there is a long `if-elseif` tree. This is how 42 turns strings read from input files into internally-useable constants. Find the lines that interpret the existing FSW tag strings, and add another one to change the string `"FOO_FSW'"` into the constant `FOO_FSW`.

The FSW structure is defined in `fswtypes.h`. As shipped, it is populated with all the global variables needed for all the FSW models in `42fsw.c`. As you customize your controller, you may add or delete variables as needed. If you get to the point that your FSW model becomes flight software, you will probably either gut the FSW structure of everything extraneous or build up a separate, dedicated data structure that serves the same purpose. As you get started, just be aware that the FSW structure has some pre-existing variables for you to grow into, but you shouldn't assume that they're all populated and maintained all the time.

4 Sensor Models

In a simulation, sensors are one interface between the truth model (environment, dynamics) and the FSW model. In general terms, a sensor model takes the truth, perhaps adds some noise or other errors, and repackages the result in the proper format for the FSW model. In 42, the function `42sensors.c:Sensors` has some very simple sensor models provided. For example, the IMU model simply copies the true angular rate of the spacecraft's main body into the FSW data structure. It's up to you to embellish this model to reflect the realities of your sensor hardware to whatever fidelity you require.

5 Actuator Models

Actuators are the other interface between the truth model and the FSW model. An actuator model accepts commands from the FSW model (eg. wheel torque commands, or thruster pulsewidth commands), and determines the forces and torques to apply to the spacecraft dynamical models. Actuator models are called from `42actuators.c:Actuators`. Like the sensor models, these models are very simple, and it's up to you to embellish them to reflect your hardware.

One actuator type requires some explanation. To enable rapid prototyping, 42 includes some ideal actuators, `IdealFrc` and `IdealTrq` as elements in the FSW structure. These actuators have no hardware equivalent. They give you the forces and torques you ask for, no questions asked. This is very useful for early studies. Suppose, for instance, that you want to study an orbital rendezvous and prox ops scenario. You want to exert control forces to perform prox ops, but you don't want to have to figure out your thruster layout or write up your thruster selection logic. `IdealFrc` is the answer. Prototype FSW and ad hoc FSW both use `IdealTrq`. As you add the hardware-based actuators into your control loop, you'll wean yourself off of the ideal actuators.