Project 1c: Detect UAV Controller Changes

The purpose of this project is to build upon projects 1a and 1b. You will design a build a detector capable of detecting if the UAV controller was changed in flight.

Step 1: Design a set of “normal” UAV parameters. Construct this by saving the default *.parm* file. Ensure that you can complete a normal flight (without crashing) with these parameters. Modify this file by setting the *RLL2SRV* and *PTCH2SRV* integral and derivate parameters to 0 (*RLL2SRV\_I,RLL2SRV\_D,PTCH2SRV\_I,PTCH1SRV\_D)*. Make sure you do not modify the *TCONT* and proportional gain components. Save this file with a new file name “*noid.parm*” (no integral/derivative). With these new parameters from *noid.parm*, complete a long flight (Tip: Use –S flag in your command to speed up the simulation). You can find more information on the parameters here: <http://ardupilot.org/plane/docs/parameters.html>

Step 2: Analyze the UAV roll controller in the figure below, modify the diagram to set the integral and derivative parts of the controller to 0. Build on your MATLAB code from Project 1b to estimate the K\_P parameter, by solving the following equation (using LSM):

The aircraft will have several servo motors, which are responsible for moving the UAV’s control surfaces. Generally servo\_1 corresponds with the plane’s ailerons (but check!!) and servo\_2 to the elevators. Keep in mind, this estimated a\_lsm value will not be exactly equal to *K\_P* (proportional part of the roll PID controller)due to non-linearity, scalar multiplication, and variable airspeed.

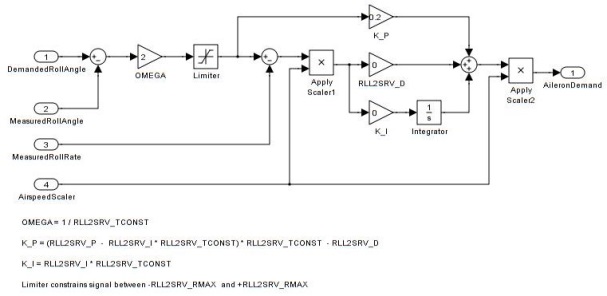


Figure 1. ArduPlane Roll Controller

Step 3: This step will require some trial and error, repeating flights until the desired output is reached. Copy your param file used in step 1 “*noid.parm*” file into a new file called “*changed.parm*”. Adjust the *RLL2SRV\_P* value by decreasing it significantly. Adjust the *RLL2SRV\_TCONST* by increasing it significantly. Re-run your flight and ensure that you do not crash with these new parameters (verify in your mission planner ground control station).

Step 4: Using this flight from *changed.parm*, recalculate your *a\_lsm*.

Step 5: Once your estimated a\_lsm (K\_P) values for steps 2 and 4 differ significantly you may continue. I’ve found that Step 3 *a\_lsm* can be 1/2 of Step 2, but this will vary depending on the parameters you have chosen. If the estimated a\_lsm values do not differ significantly, repeat steps 3 and 4.

Step 6: Design an RLSM procedure to estimate the 'a' value from the equation in Step 2, except now call is *a\_rlsm*. Now run your captured flight data through the RLSM procedure, plotting your estimated *a\_rlsm* for both the “normal” flight, parameters from step 1 (“noid.parm”) and the “changed” (“changed.parm”) flight, parameters from step 3.

Step 7: Capture a new flight, in which you start with the “normal” parms, change to the “changed” parms about 15 minutes into the flight, and then change back to the “normal” parms 15 min later. (Tip: Use –S flag in your Linux command to speed it up.) This might take some trial and error to ensure that your flight is long enough. You can always make your flights longer, in general the longer the flight, the easier it will be to determine parameter differences.

Step 8: Determine a normalcy profile based on normal flight, using the mean and variance of your output from Step 6. Make sure to only take data from the part of the flight once your *a\_rlsm* value has converged (stopped changing). You can always initialize your *a\_rlsm* value to your *a\_lsm* value from step 2 instead of initializing it to 0 for quicker convergence. You will need to “play” with different forgetting factors in the RLSM procedure. Measurements older than *τ*=1/1−*(forgetting\_factor)* typically carry a weight that is less than about 0.3.

Step 9: Insert a detector into the RLSM for loop, determining if something is “wrong” with the controller. Detected “wrong” parameters could be indicative of a UAV control system hack. Feed your captured data from Step 7 into this RLSM loop (now with anomaly detector) and alert when the controller changes parameters, and when it has changed back.

Submit the following:

1. 2 parameter files: Normal flight and changed flight
2. 3 pairs of flight files (.tlog & csv): Normal flight, changed flight, and normal-to-changed-to-normal flight
3. 3 RLSM plots: Normal flight, changed flight, and normal-to-changed-to-normal flight
4. 2 calculated LSM values: Normal flight and changed flight
5. MATLAB Code. The code should be heavily commented, clearly showing your detector
6. Any write-up documentation you think your project requires to be understood