You will write a Behavior class and a Pilot class that invokes Behavior instances. Each Behavior instance tries to achieve a specific flying goal (given to the Behavior constructor). One important such goal is straight and level flight at a constant air speed. This is what we discussed Friday in class as a recovery method for failed more aggressive actions. But we would like to achieve other goals and pursue multiple goals at the same time. The Pilot class manages a set of behaviors. When given an additional behavior, it is added to the active set if its precondition is satisfied (your choice what this means but be sure to explain and defend it in your write-up). At each time point Pilot evaluates the active behaviors, combines their recommended control manipulations (your choice again as above), and manipulates the plane’s controls. Behaviors that succeed or fail should, of course, be removed from the active list. The Pilot instance should terminate when no active behaviors remain.

Build your plane with FGI (config=<as you like>, collect=4). It will generate a flight file with a name such as Flight_1474238866.flt which contains every 4th point of your flight.

The classes will have the following profiles:

class Behavior:

    def __init__(self, fgi, goal):
        Where fgi is an already-flying instance of an FGI and goal consists of two lists: (pitch goal, roll goal, kias goal) and tolerances for each of these. For example, goal of ((1.5, 20.0, 100.0), (0.1, -1.0, 1.0)) specifies a pitch between 1.4 and 1.6 degrees, an air speed within 1 knot of 100 and any value for the roll angle whatsoever.

    def pc(self, state):
        state is also two lists: first, the result of fgi.look() and second, the current (elevator, aileron, throttle) settings. It returns a number [0.0, 1.0]. Zero means that the behavior cannot achieve its goal from this state; one means that the behavior will certainly succeed. Numbers between 0 and 1 indicate a measure of confidence in success with 0.5 indicating no information.

    def reinit(self, warmP=False):
        This resets the behavior so that it can be invoked again to achieve its goal. If warmP is False then it is a cold re-initialization and the accumulated integral errors should be zeroed, otherwise they should remain unchanged.

    def behave(self):
        This is called to repeatedly calculate how the controls should be set to accomplish the goal. It returns a string and a list. The string is one of ‘success’ ‘failure’ ‘continue’ indicating whether the behavior has finished. The list contains three sub lists each with a number and a Boolean. They specify recommended elevator, aileron and throttle control positions and whether or not they are used by the behavior. For example, returning ‘continue’, ((0.2, True), (0.5, False), (0.9, True)) indicates the behavior has not yet finished but expects to succeed and would like the elevator set to 0.2, doesn’t care how the aileron is set, and wants throttle set to 0.9.

class Pilot(threading.Thread):

    def stop(self):
        Terminate the controller and the FGI instance.

    def report(self):
        Print the current status of the controller and its running behaviors.

    def run(self):
        For the thread execution.

    def addBeh(self, beh, warmP=False):
        This tries to add Behavior instance beh: test its preconditions and if satisfied, initialize with the argument warmP and add to the active set.
You are to exercise a behavior instance encapsulating 3 PID controllers which performs emergency recovery to straight and level flight at 100 kts from the least restrictive preconditions that you can manage.

Once again all team members should contribute as equally as possible and must understand all of the design decisions, code, and evaluations. For this MP you are assigned to the following teams:

Team 1: btyao2, jieyin3, adnigam2, lu72

Team 2: vndrbch2, fxiang2, aepstei3, zwu22, chen317

Team 3: xyuan12, mzeng5, hzhan107, apkizer2, qding5

Hand in:

1) Your documented code

2) A paper a) explaining and defending the various design decisions you have made, b) characterizing and explaining the failure areas of your recovery behavior, c) an honest discussion of the accuracy of your preconditions.

3) A set of graphs and/or plots showing the behavior in action.

Extra Credit:

Implement sufficient additional behaviors that could support landing the plane. These behaviors may encapsulate one or more PIDs, other simpler behaviors, or anything you like. In an accompanying paper describe the landing procedure and how it would be accomplished by some executive function using your Pilot’s addBeh function. You may assume that you have already extended your Pilot to notify the executive of significant events. Hand in your documented code and a paper.

Extra Extra Credit:

Extend your Pilot and write the executive landing function. It takes as input the already-flying plane as an FGI instance, a field elevation, a runway’s orientation and the latitude and longitude of the runway’s threshold (its beginning point). After touch down (with closed throttle) judicious use of the differential brakes may be helpful. This may be accomplished using fgi.setFGProp with '/controls/gear/brake-right' and '/controls/gear/brake-left'. A perfect landing would stop pointing down the runway on the center line close to the threshold and with no damage. Hand in documented code, a paper, and evidence of landing outcomes.