CS 524 Task 3: Autopilot PID Controller

Description

The goal of this task is to build, tune, and evaluate a basic proportional-integral-derivative (PID) control system from a software engineering perspective. It operates within our familiar framework of modeling, simulation, visualization, and analysis. The notional application is to control an airplane such that it maintains an altitude. However, the aeronautical background does not play any role beyond establishing the context. This approach could equally apply to balancing server loads or any other CS-specific need.

Model

The implementation of this airplane model is already provided in class `Agent` so that everyone is modeling the same problem the same way. Do not change this code. It is available just to help your understanding.

The static definition (i.e., what you cannot directly alter at runtime) of the airplane is a point source in a two-dimensional side-view world defined by unitless *x* and *y* coordinates that increase right and upward, respectively, from the origin (0,0). The airplane also has a pitch, which is where it is pointing above or below the reference line; positive is pitch up. It also has a unitless speed. There is no gravity, and the notion of lift is captured in `Agent`. When the airplane it instructed to update (via a call to `update_()`), it goes from its current position to its new position based on its pitch and speed. Between this definition of data and control, the airplane can move predominantly to the right over time at any speed within the two-dimensional world. However, you do not have direct access to either aspect.

The dynamic definition (i.e., what you can directly alter at runtime) is limited to the elevator of the airplane, which changes its pitch. You do have direct access to it. You can deflect it plus or minus by a certain amount (on the order of five degrees maximum). The current pitch of the airplane changes by this amount upon each update. There are two modes: *instantaneous* immediately deflects the elevator to the specified angle, whereas *noninstantaneous* incrementally changes to it by a fixed step (in `PITCH_DELTA_DELTA`) per update. These modes loosely correspond to linear and nonlinear behavior, respectively.

The objective is to start from an initial position and track a horizontal altitude line that extends infinitely to the right at *y*=0. Your controller needs to continuously change the deflection of the elevator to attempt to intercept and hold the line as well as possible under different conditions.

Simulation

The simulation is a simple framework for executing the model. It is provided in class `Driver`, which you may change any way you want. It loops over an arbitrary number of steps, and once per step, it updates the airplane. The experiments are described below. There should be only one implementation of the PID code, which is configured by passing in appropriate tuning values for its P, I, and D parameters (and whatever else you need to set up the problem). It should be possible to rerun each experiment without changing the code beyond these values. Terminate execution when your solution reaches a representative point; e.g., is tracking the line consistently or unrecoverably out of control. It is acceptable to pass in the number of steps to run, which you may determine by eye.

The driver code already includes a primitive "bang bang" controller and a stub for where your PID controller should go. You will need to add support code for the data logging and analysis.

Visualization

Use Gnuplot to demonstrate how your model behaves within the simulation. Lecture 30 and later show a variety of outputs.

Write your output to a text file (you may add file I/O to automate this) with two columns for *x* and *y*, respectively.

Set the scale to be uniform in both directions. Substitute an appropriate value for your execution.

```
set xrange [-100:100]
set yrange [-100:100]
```

Generate your output with `plot '`*`yourfilename`*`' with lines`

Analysis

In addition to the visualization, which is mostly a qualitative analysis, we are also doing some quantitative analysis. You need to report (1) the average and (2) the standard deviation of the error from the first to the last step, and (3) the step count to achieve stability (or unrecoverable loss of control). You also need to report (4) the approximate length of the track line (as defined in (3)), and (5) the area under the curve.

At the end, provide some kind of comparative graph that shows all of these parameters together for all the experiments. Multiple graphs are ok, but try to minimize the number. Keep in mind to compare apples to apples.

Experiments

For each experiment, submit the values of your parameters in a meaningful way, along with a screenshot of the visualization (including the scale on the axes), and the five analytical measures above. Provide a brief discussion of what happened, why you think it happened (for better or worse), and if it did not work, then what you might try to improve it.

There are three dimensions to combine:

A. *Approach angle*: {shallow, intermediate, steep}, which respectively correspond to 10, 27, and 45 degrees off altitude (bottom left of origin). The specific (*x*,*y*) starting positions are respectively about (–49,–9), (–45,–23), and (–35,–35), which are all 50 distance units away from the intended intercept point at the origin.

B. *Speed*: {slow, medium, fast}, which respectively correspond 1, 2, and 3 for the `speed` parameter in the constructor.

C. *Deflection type*: {instantaneous, noninstantaneous} for the `isDeflectionInstantaneous` parameter in the constructor.

Submit a professional report with your findings, as well as your source code for `Driver` and whatever else you needed. Order the experiments by nesting them as they appear above; i.e., shallow+slow+instantaneous, shallow+slow+noninstantaneous, shallow+medium+instantaneous, shallow+medium+noninstantaneous, shallow+fast+instantaneous, shallow+fast+noninstantaneous, and so on.