Tips for Drone Control PID project

1) First make sure your PID control functions are correct. A good way to do that is to try some manual PID tuning first. This will also help you understand your system better. To do this, you can temporarily set the TWIDDLE flag to False in DualRotor_TestSuite.py and try setting manual values for thrust_params and roll_params in run_test() method. Remember to turn TWIDDLE flag back True after this. A good resource for a manual tuning method is here: https://youtu.be/uXnDwojRb1g

2) Make sure that you understand how Twiddle should behave for this project. Some manual tuning as described in tip #1 above will help with that. There is a subtle difference between this project and the example of steering a car in the lectures. In particular, when you steer a car during twiddle, you will affect it immediately even if the steering angle is very small, and you will receive an immediate feedback which you can use to determine the direction of your search. In the case of a drone, it requires a minimum value of Thrust (and hence tau_p) to take off. Before that value is reached, you won't get any useful feedback because the error is going to stay the same. You need to think a bit about where that might cause a problem in the twiddle implementation from lectures. One thing that will help you is to turn on the VISUALIZE_TWIDDLE flag in DualRotor_TestSuite.py. You can pause at each iterations of twiddle and examine your parameter values (tau), and see if they are moving in the expected direction based on how the error changed (or did not change). For example, if you see that twiddle keeps effectively reducing your thrust even though your drone did not move enough, then that could indicate that you need to check your implementation. You can also turn on DEBUG_TWIDDLE in DualRotor_TestSuite.py. This will print out values on your console that you can inspect.

3) Twiddle is susceptible to local minima. This means that based on different starting values of your gain parameters or hyperparameters (tau, dp, dp change factor), twiddle may end up with different ending values. One thing you could do is to try different initial values for your parameters or hyperparameters. But first try stepping through your code as described in tip #2 (with initial from lectures) and make sure your Twiddle implementation is doing what you expect it to do. Once you are comfortable with your implementation's behavior and you are confident that you just need to expand your search, then you can try the following:

    a) Grid Search: Basically put your twiddle loop within another loop that tries different initial values.

    b) Manual tuning. You can temporarily set the TWIDDLE flag to False in DualRotor_TestSuite.py and try setting manual values for thrust_params and roll_params in run_test() method. Once you arrive at some relatively good values manually, you can turn TWIDDLE flag back on and use those values as initial values in your find_parameters_XXX() function in drone_pid.py. A good resource for manual tuning method is here: https://youtu.be/uXnDwojRb1g

4) For find_parameters_with_int(), where you also have to find an Integral gain, you can also try Grid Search (see 3a) with a few different integral values. Or you can manually come up with an initial Integral value and let Twiddle fine tune it.

5) For find_parameters_with_roll(), you can try two strategies:

    a) The first approach is simultaneous tuning. Here you simply extend the Twiddle algorithm to tune 6 parameters instead of 3. You might need to make a choice and experiment with what sequence works best. For example, place all 3 parameters for Thrust first, followed by parameters for Roll. Or place P for Thrust and Roll, followed by D for both, followed by I for both.

    b) Sequential tuning: Run the Twiddle loop for thrust params first, holding roll params constant. Then hold thrust params constant and run Twiddle for roll. Repeat until error improvement is less than some threshold.

6) Stopping criteria: Depending on your implementation, you could find that the simple stopping criteria for Twiddle from lectures (i.e. sum(dp) < tol) may not be enough. This could be indicated by your program running for a very long time even though your error has decreased to an acceptable value, or timing out in gradescope. You may want to use additional stopping criteria if this is the case. Here are a couple of suggestions, although there could be others:

a) Keep a track of the best_error across iterations of your loop, and stop if the improvement in the best_error is less than a threshold over some number of iterations.

b) Stop if the error is below an acceptable threshold.

7) Error: In the lectures, when you twiddle the error that your run() method returned was the total Cross Track Error for the whole path or some form of it. In this project, the run_callback() method returns multiple values. You have to use them to create an error value that you will minimize across iterations of Twiddle. At a minimum you will need to use the hover_error. Sometimes that is enough, but depending on your implementation, you may find that if you just reduce the hover_error, you may exceed the max velocity and/or max oscillations. So you can convert the velocities and oscillations returned from run_callback() into an error measure. For example, consider velocity. The return values contain a maximum velocity your drone should stay within. You are also given the drone's max velocity at any point in its flight. You can convert this info into an error value. So if you come up with gain values which make the drone exceed the max velocity, your error should be higher than the values that keep it within the max velocity. Furthermore, the error should increase the more you exceed the max allowed velocity. The case of oscillations is similar. For an example of how to convert the returned values into an error, see DualRotor_TestSuite.py > Part_1_a_TestCase > get_score() method.

Also, depending on your Twiddle implementation, if you are using all 3 errors (hover, velocity and oscillation), you might notice that your hover_error might want to pull your gains in one direction, but your velocity error is pulling them in the opposite direction. Same could happen between hover and oscillation error, or velocity and oscillation errors. If you are running into this issue, you will notice this especially in the initial iterations of Twiddle. In this case, you may need to use some weighted combination of the 3 errors.

Overall, start with a simple strategy, then add to it if needed.