Controls Project

James Tyree 5/1/18

Implemented body rate control in python and C++

Body Rate was a simple P controller, comparing commanded rates to actual (or estimated) rates. In order to test this controller, commands were manually dummied in and the quad was verified to follow them. Without the other controllers operating, they would result in a crash, but that was expected.

Implement roll pitch control in python and C++

Roll pitch control was the most complex to implement. Lateral acceleration commands had to be combined with net thrust to yield roll and pitch rates that feed into the body rate controller. This was a proportional-only controller.

Lateral acceleration commands could be dummied in to test the controller. A constant acceleration command resulted in a constant roll or pitch angle. This could easily be observed in simulation with fixed commands before moving on to higher-level controls.

Implement altitude control

The altitude controller takes a desired z position and generates the proper thrust to hold it. This is a full PID controller. Integral is necessary to account for mass variations that the controller may not be aware of. The drone’s attitude is taken into account when calculating thrust. Roll and pitch reduce the z component of thrust as the quad rotates.

Implement lateral position control

Lateral position is a PD controller. Derivative is needed to have the drone start slowing down before it reaches the target position. Without it, the drone shoots past the targets and fails the tests! When derivative is too high, the drone gets “twitchy” where it over commands and then quickly compensates.

Implement yaw control

Yaw control is a simple proportional controller. Over controlling yaw can use up control authority that is usually better applied to roll, pitch, and altitude control. Before implementing proper angle wrap-around, the controller would occasionally rotate the quad in the wrong (longer) direction to get to a new target. Code was added to check for this and normalize the yaw angles.

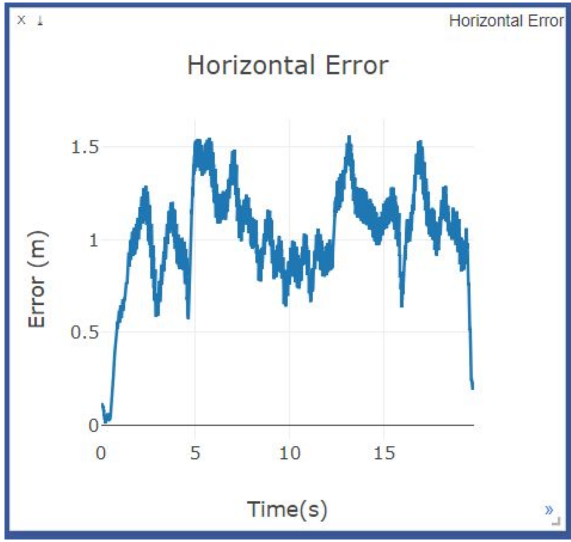
Implement calculating the motor commands given commanded thrust and moments

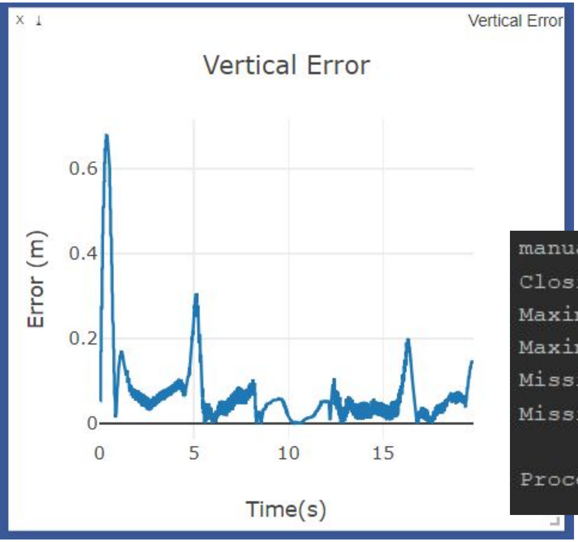
This was a matter of solving 4 equations of 4 variables to get from pqr and collective thrust to four motor thrust commands. Students on the Slack channel pointed out that Wolfram Alpha does a great job of solving equations like these!

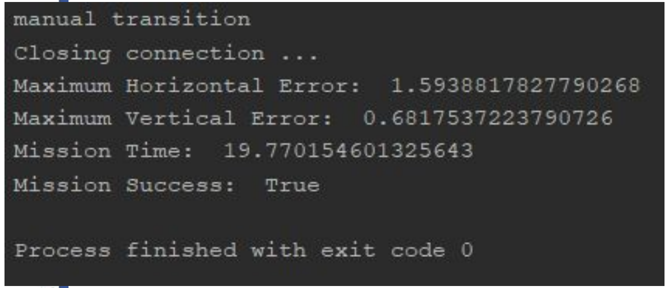
After getting the 4 motor commands, they are constrained between nominal real world values. When flying, you don't want to allow the motors to stop because they have inertia and take time to respond. Real motors also have limited thrust.

Python Criteria

It’s a little twitchy, but it passes all the criteria. I focused more of my tuning time on the C++ project.



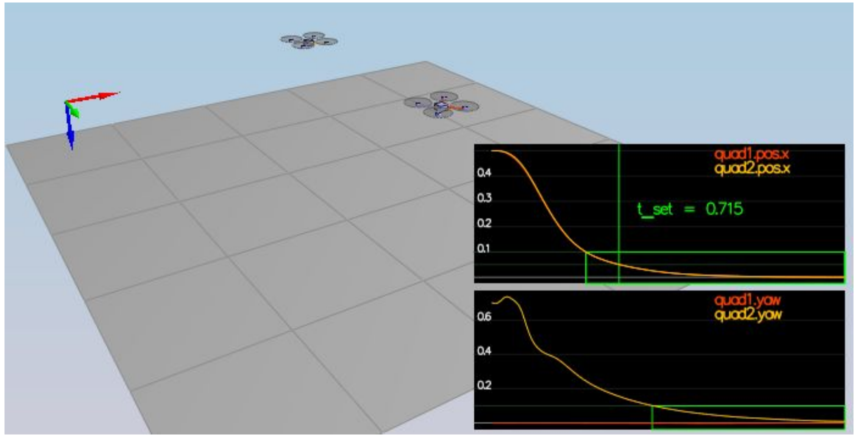


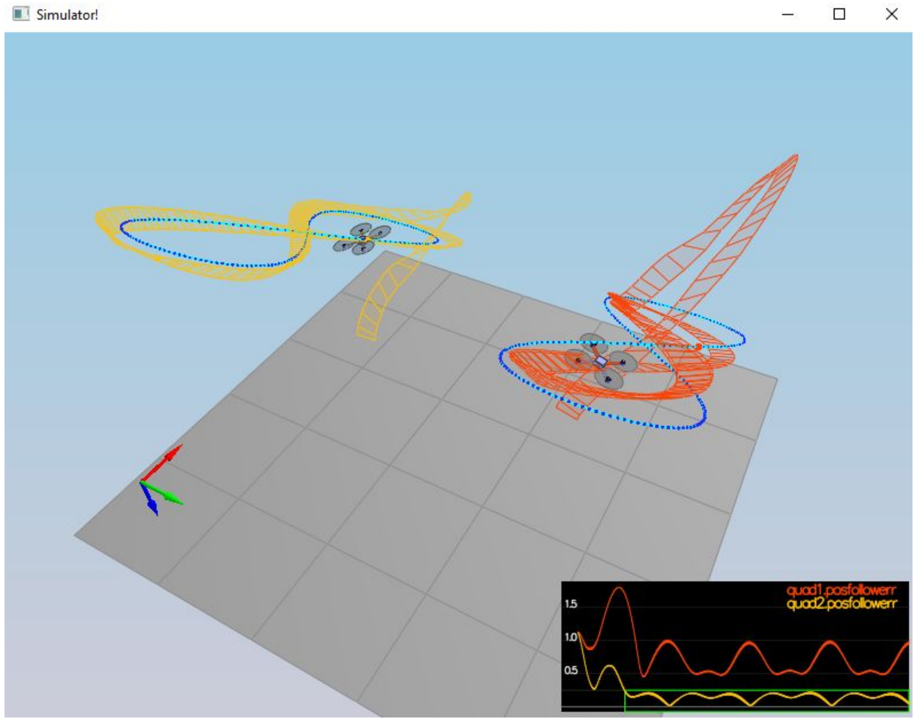


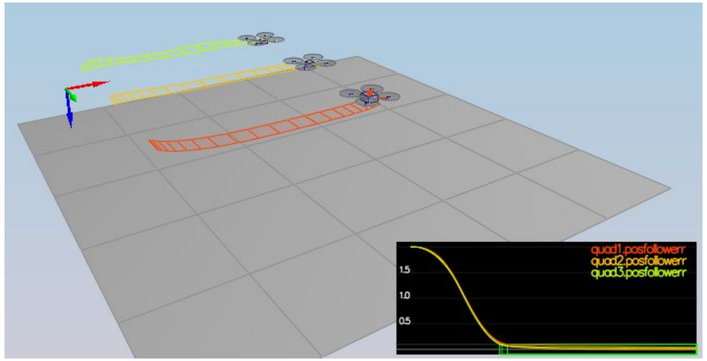
```
manual transition
Closing connection ...
Maximum Horizontal Error:  1.5938817827790268
Maximum Vertical Error:  0.6817537223790726
Mission Time:  19.770154601325643
Mission Success:  True

Process finished with exit code 0
```

C++ Criteria


quad1.pos.x
quad2.pos.x
t_set = 0.715
0.4
0.3
0.2
0.1
quad1.yaw
quad2.yaw
0.6
0.4
0.2


Simulator!
quad1.posfollowerr
quad2.posfollowerr
1.5
1.0
0.5


quad1.posfollowerr
quad2.posfollowerr
quad3.posfollowerr
1.5
1.0
0.5