To do:

1. Order an optical sensor, such as the QRE1113. Check that it’s compatible to the Raspberry Pi (RPi).
2. Wire the optical sensor to the RPi on the drone (probably using I2C or UART) connection. Mount it on one arm of the drone facing the motor. (Work with Pat and/or Will to make this go quickly and smoothly).
3. Apply a square of reflective tape to the motor.
4. Design a mount to hold the done in place with no rotation while you conduct your motor tests. You should leave the propeller on the motor, but this will create lift, so you’ll have to secure it.
5. Adjust the sample code in the rotor speed measurement documentation. Make it so that you can manually input a rotor speed command, and the optical sensor will automatically record rpm data.
6. Check the data to ensure it’s working as expected. Now you have a functional testbed for the motor.
7. Implement a dynamic model of the motor in MATLAB using the estimated parameters in the documentation.
8. Design a few input trajectories (PWM signals or commanded rotor speeds), such as a step increase, sin waves, sawtooth, etc.
9. Collected data with these input trajectories on both your numerical model and the physical experiment.
10. Compare the data. If there are significant discrepancies, figure out how you need to adjust your model until they behave roughly similarly. Now you have a validated numerical model.
11. Apply Frank or Amy’s algorithm to analyze the gain of the simulated motor using triangulations. Now you have an estimated bound on the gain of the motor.
12. Apply a Weiner process (random walk) input to the motor to test the gain experimentally. Do this several times with different lengths of inputs. If you find that the experimentally calculated gain is smaller than the simulated bound, then we have some validation of your method. **This is one very good paper.**
13. Using the network dissipativity documentation, plug in the gain bound and solve for the controller QSR parameters. Now you have a constraint on your controller. If your controller satisfies this QSR property, then the network should be input-output stable.
14. The existing controller is proportional-derivative (PD) with respect to roll, pitch, and yaw. But we want a controller that takes in the rate of change of roll, pitch, and yaw, as well as the rotor speeds squared (see Figure 3 of the network decomposition documentation). The paper by Mahony (see References) shows a hierarchical controller for dealing with these measurements (Figure 5). If you can find the dissipativity of this type of controller, or you can convert it to a state space, use this controller. Otherwise, you may need to choose a different controller structure or a different network decomposition. (For instance, you could just use an LQG controller with all of the given measurements, or you could set up the network decomposition with one controller for torques and a second for rotor speeds to better reflect Mahony’s).
15. Whatever you decide to use in 14, convert to state space and use the KYP Lemma to constrain your controller parameters.
16. Run a simulation of the full drone with the motors. Randomly pick controllers that satisfy the KYP Lemma. Hopefully, you will see that they all stabilize the attitude. Randomly pick controllers that don’t satisfy the KYP Lemma. Hopefully you’ll see that many of them do not stabilize the attitude. You’ve now validated the control design method in simulation.
17. Now run the same tests on the mounted drone. Hopefully you’ll see that the results hold similarly. Now you’ve validated the control design method in experiments. **This is a second very good paper.**
18. In the distant future, we would then extend the design to deal with adverse conditions like delays and networks of other drones. And we’d try to do tests with the drone flying freely instead of mounted. And we’d stabilize x, y, z as well instead of just roll, pitch, yaw. But that’s distant future stuff. **This is motivation for future work, fodder for grants, and justification for the current work.**