Explain the Project

The goal of this project was to design a cascade PID controller (proportional–integral–derivative controller) very similar to the one illustrated in figure 1. These controllers allow for automatic correction to a control function, similar in function to an airplane's autopilot or (more simply) an automobile's cruise control.

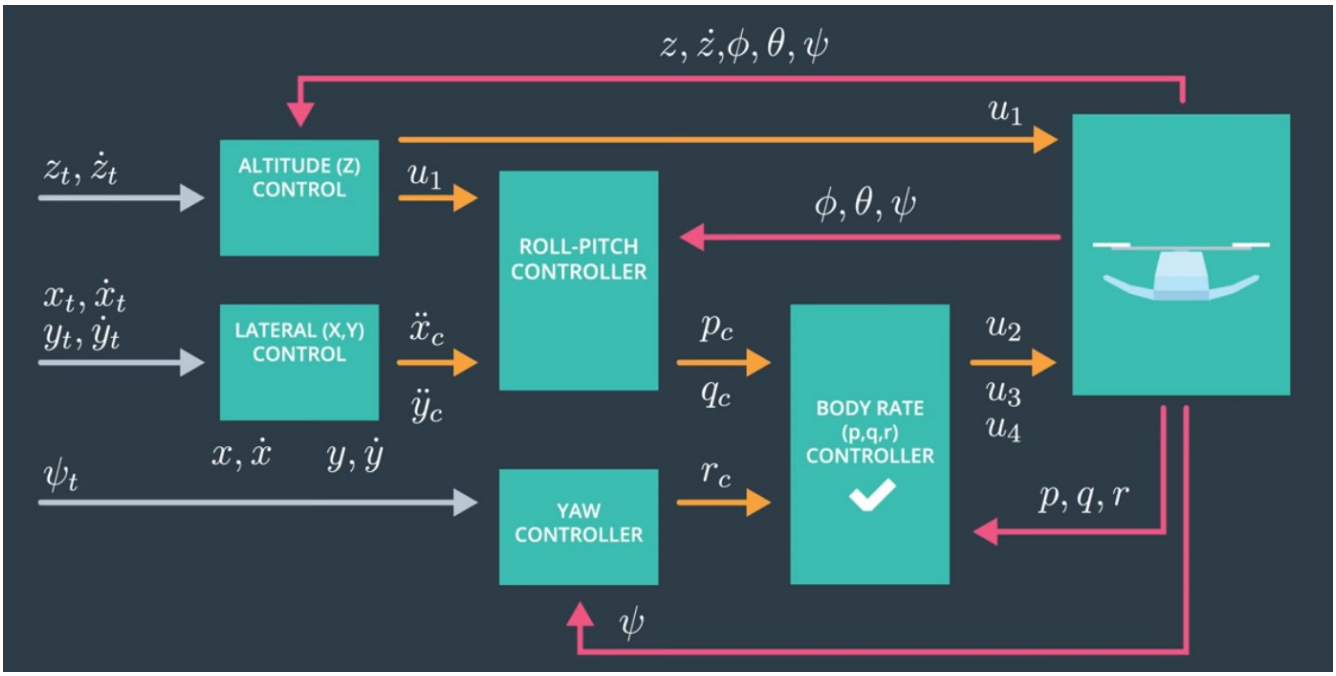

Figure 1: Example PID Diagram

Implemented Controller

A lot of the code for the controller was just converting the given python solution to C++ (*AltitudeControl()* was more involved).

The controller was implemented from the outside in, as our instructors indicated was optimal, however the first part then implemented was the *GenerateMotorCommands()* function (lines 58 – 96) because this needed to be implemented before the *BodyRateControl()* function, so that we could actually test the output. This involved solving a series of linear equations given in lecture for the individual thrusts on the motors.

Following that we could begin designing the rest of the controller, starting with the *BodyRateControl()* function (lines 99 – 124). This function calculates the error between *pqrCmd* and the actual *pqr*, multiplying it by the moments of inertia and a defined rate *kpPQR*. This returns the desired moments across our three axes.

The third thing we needed was the *RollPitchControl()* function (lines 127 – 164). This function takes the acceleration of the drone, the commanded thrust, and the drone's attitude to determine rates of rotation needed about the x and y axes (the roll and pitch).

With *BodyRateControl()* and *RollPitchControl()* functions designed the *kpPQR* and *kpBank* parameters were tuned in Scenario 2 until controller passed testing with *t_set* values of 0.245 and 0.155 for *quad.roll* and *quad.omega.x* repectively.

The next thing that was designed was the *LateralPositionControl()* function (lines 220 – 271). It takes the commanded position and velocity, current position and velocity, and a commanded feed-forward acceleration to determine the desired horizontal acceleration.

This was followed by the *YawControl()* function (lines 274 – 307). This function takes a commanded yaw, and the current yaw returning a desired yaw rate to achieve the commanded yaw.

After *LateralPositionControl()* and *YawControl()* were designed the parameters *kpPosXY*, *kpVelXY*, and *kpYaw* were tuned in Scenario 3 until they passed the test. The controller passed the test with a *t_set* value of 0.645 for both *quad1.pos.x* and *quad2.pos.x*. Unfortunately, the simulation does not show the *t_set* value *quad2.yaw*, but the error was less 0.1 for at least 1 second according to the terminal. Note that there is a a typo in the included project README which states that we needed to tune *kpPosZ* twice, instead of *kpPosXY* and *kpPosZ*.

The final part designed was the *AltitudeControl()* function (lines 172 – 217). While this function was initially just a rehash of the python solution code, it had to be modified to include the integrated altitude error (the "I" part of a PID-Controller), which was accomplished by multiplying the positional error (*posZ_error)* by the functional parameter *dt* and adding it the *IntegratedAltitudeError* variable every time the function is run (should be every loop of the main loop). This function returns thrusts needed to maintain altitude of the quad-copter in simulation.

With *AltitudeControl()* designed, the final two control parameters kpVelZ and KiPosZ were tuned in Scenario 4, until the drone passed the testing (positional errors less than 0.1 for at least 1.5 seconds in all three iterations of the drone). Of special note here is how high the gain was on the parameters here. The assignment instructions informed us that the initial control parameters were all 2x to 4x less than they needed to be, however the values for *kpPosXY*, *kpPosZ*, and *KiPosZ* all needed to be turned up 20x to 40x the original values in order to pass Scenario 4 in my implementation. *kpPosXY* and *KiPosZ* could be 2x to 4x the original and pass Scenario 3, but would not pass here (the new values assigned for Scenario 4 still work for Scenario 3).

Flight Evaluation

My drone is able to fly the roughly able to fly the figure eight trajectory. Altitude looks to be the problem child, but the overall shape of the figure eight is there.