BUILDING A CONTROLLER

1. **Writeup**
   Rubric explanation
2. **Implement Controller**
   - Body rate control in C++
   - Roll pitch control in C++
   - Altitude controller in C++
   - Lateral position control in C++
   - Yaw control in C++
   - Calculating the motor commands given commanded thrust and moments in C++
3. **Flight Evaluation**
   Ensure that in each scenario the drone looks stable and performs the required task.

Writeup:
You're reading it!

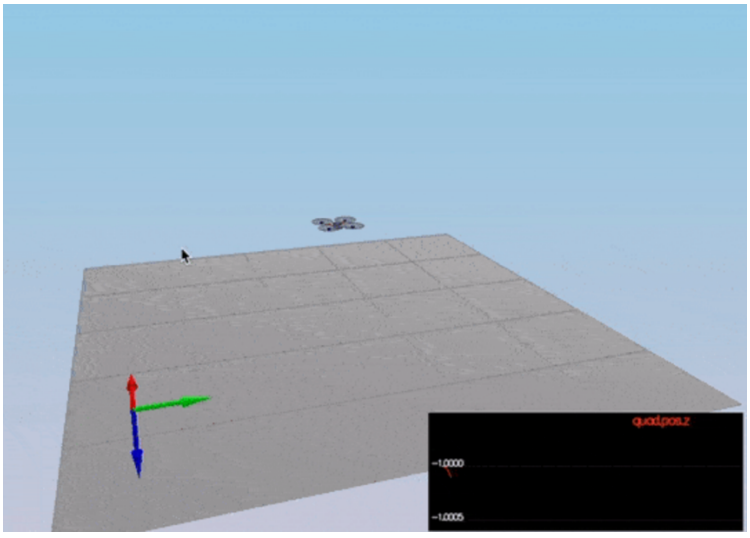
Implement Controller
Parameter tuning in a more complex environment on simulator, given in C++ is definitely more challenging than previous representations in Python, dealing with real limits could affect seriously the flight performance and things could go wrong.

Udacity provides a startup code with placeholders along with the simulator implementation, in the readme file on this repo, you can find detailed information on how to properly run the project and how to approach the coding solution for the applied concepts.

Coding takes place on config directory, and mainly in `QuadControl.cpp` and `QuadControlParams.txt` files. Those files contain the configuration for the controller. Once simulator is running, params tuning can be done and refresh for next execution.

In order to comply and pass the metrics, `QuadControl.cpp file contains the controller only with the parameters tuned.

Scenario 1: Intro | Adjust drone's mass



Scenario 2: Body rate and roll/pitch control

a.GenerateMotorCommands method: (`QuadControl.cpp` file lines 56-92)

$$F_1 + F_4 - F_2 - F_3 = \tau_x / l \quad = t_1 \tag{1}$$

$$F_1 + F_2 - F_3 - F_4 = \tau_y / l \quad = t_2 \tag{2}$$

$$F_1 - F_2 + F_3 - F_4 = -\tau_z / \kappa \quad = t_3 \tag{3}$$

$$F_1 + F_2 + F_3 + F_4 = F_t \quad = t_4 \tag{4}$$

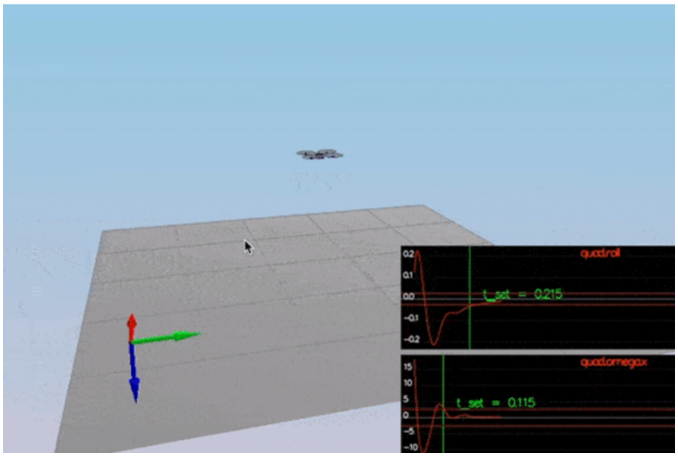
Where Fi are motor's thrust, ti are the moments for each direction, k is the drag/thrust ratio and l are the drone arm length over square root of two. (Ref. Udacity Lessons)

b.BodyRateControl method: (`QuadControl.cpp` file lines 94-118)

It applies a P Controller and the inertia. `KpPQR` parameters are tuned to stop drone from flipping, having an initial thrust for altitude where `**thrust = mass * CONST_GRAVITY**`.

c.RollPitchControl method: (`QuadControl.cpp` file lines 124-169)

It applies rotation matrix from body-frame accelerations and world frame accelerations.



$$\dot{b_c^x} = k_p(b_c^x - b_a^x) \tag{1}$$

$$\dot{b_c^y} = k_p(b_c^y - b_a^y) \tag{2}$$

$$b_a^x = R_{13} \tag{3}$$

$$b_a^y = R_{23} \tag{4}$$

$$\begin{pmatrix} p_c \\ q_c \end{pmatrix} = \frac{1}{R_{33}} \begin{pmatrix} R_{21} & -R_{11} \\ R_{22} & -R_{12} \end{pmatrix} \times \begin{pmatrix} \dot{b_c^x} \\ \dot{b_c^y} \end{pmatrix} \tag{1}$$

Scenario 3: Position/velocity and yaw angle control

d.AltitudeControl method: (`QuadControl.cpp` file lines 171-216)

Controls the acceleration, so the thrust needs to control de altitude.

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} + R \begin{pmatrix} 0 \\ 0 \\ c \end{pmatrix} \tag{1}$$

$$b^x = R_{13} \tag{2}$$

$$b^y = R_{23} \tag{3}$$

$$b^z = R_{33} \tag{4}$$

$$\bar{u}_1 = \ddot{z} = cb^z + g \tag{5}$$

$$c = (\bar{u}_1 - g)/b^z \tag{6}$$

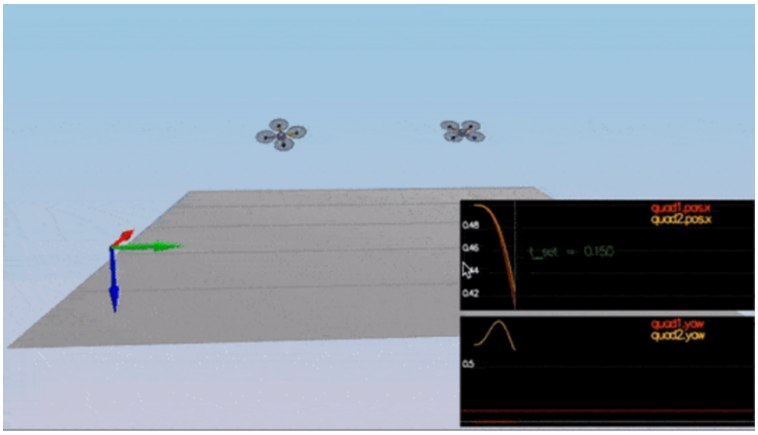
$$\bar{u}_1 = k_{p-z}(z_t - z_a) + k_{d-z}(\dot{z}_t - \dot{z}_a) + \ddot{z}_t \tag{7}$$

e.LateralPositionControl method: (`QuadControl.cpp` file lines 218-275)

Controls acceleration on x and y.

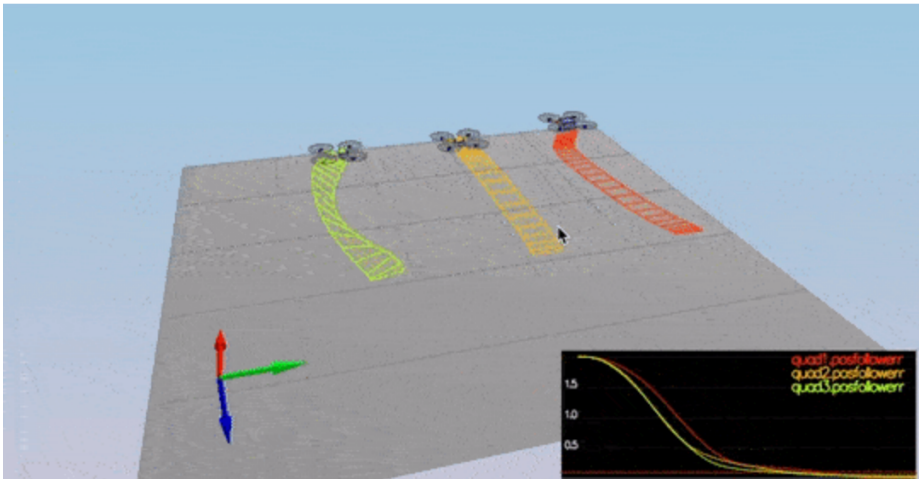
f.YawControl method: (`QuadControl.cpp` file lines 277-311)

P Controller for optimize the yaw to be between [-pi, pi].



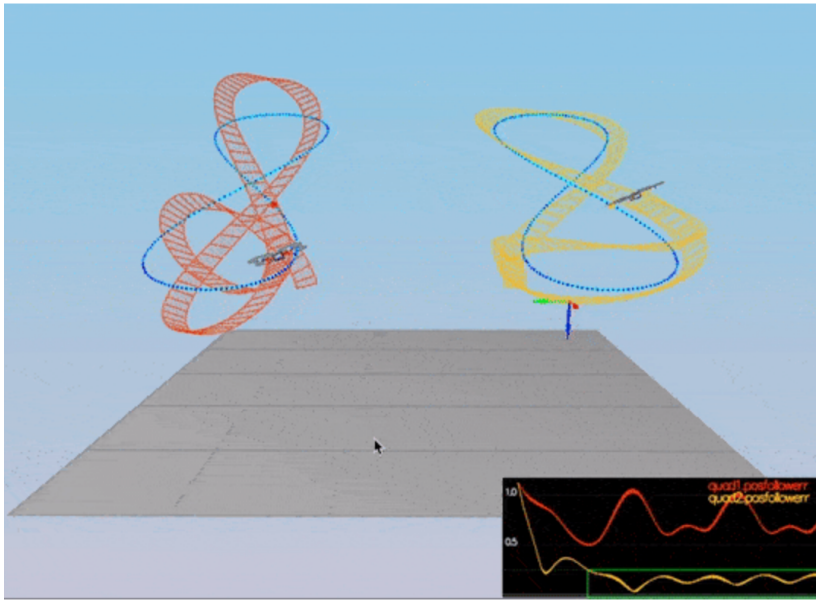
Scenario 4: Non-idealities and robustness

Once we finished tuning our parameters, we need to move forward from PD to PID controller, incorporating an integral part to the altitude controller.



Scenario 5: Tracking trajectories

The drone is required to follow a trajectory. Fine tuning would be required in order to avoid errors.



Flight Evaluation: Succeed!