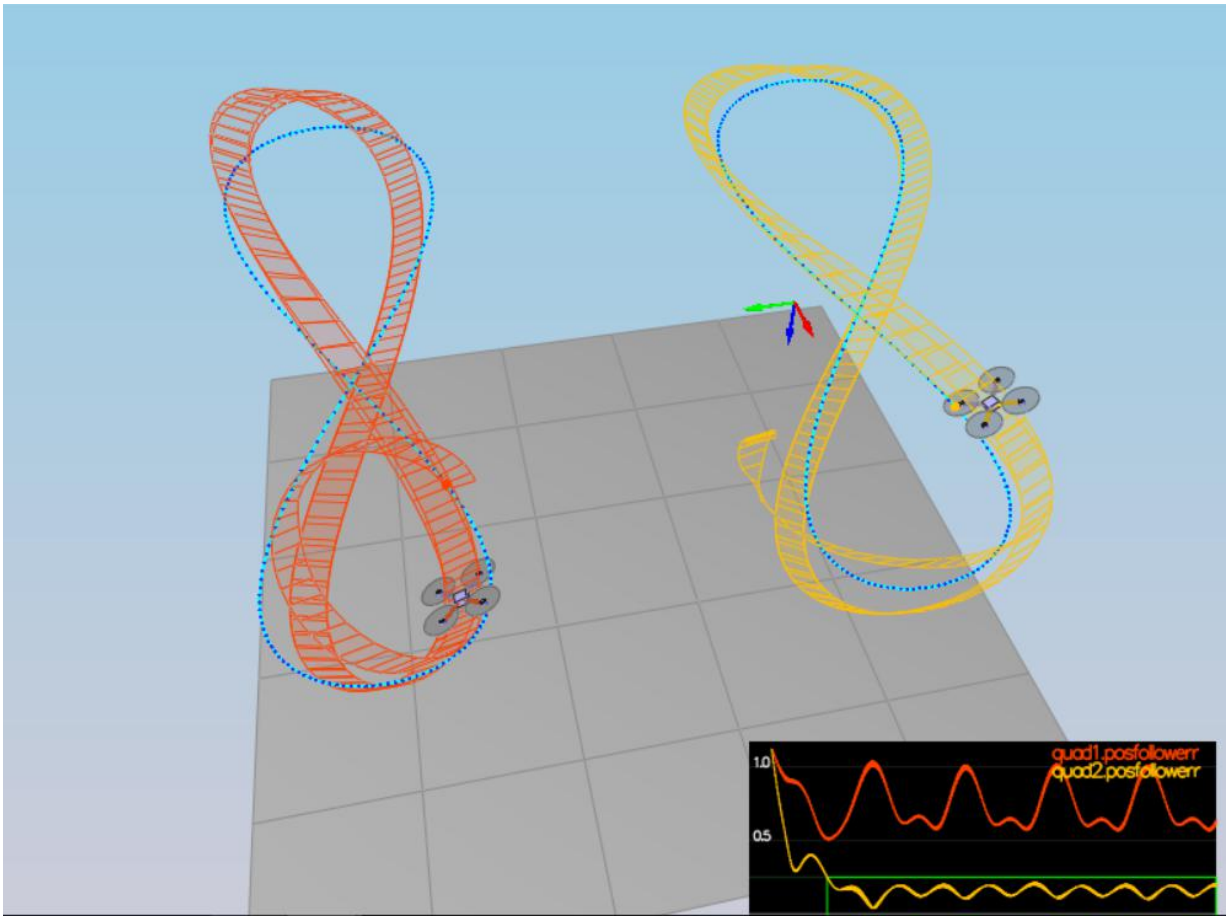
July 24, 2018 BY SWAROOPKS

Control of 3D Quadrotor



The third project is about implementing individual control loops in C++. This code will control a drone in an entirely new simulator. This project is broken into a series of "scenarios". These assist with tuning of your controller, the simulator also contains real time performance evaluation.

Prerequisites for project (windows)

Regardless of your development platform, the first step is to download or clone the **C++ simulator repository**.

```
git clone https://github.com/udacity/FCND-Controls-CPP.git
```

Once you have the code for the simulator, you will need to install the necessary compiler and IDE necessary for running the simulator.

For Windows, the recommended IDE is Visual Studio. Here are the steps required for getting the project up and running using Visual Studio.

Download and install **Visual Studio**

Select Open Project / Solution and open `<simulator>/Simulator.sln`

From the Project menu, select the Retarget solution option and select the Windows SDK that is installed on your computer (this should have been installed when installing Visual Studio or upon opening of the project).

To compile and run the project / simulator, simply click on the green play button at the top of the screen. When you run the simulator, you should see a single quadcopter, falling down.

Project description

This project originally had two parts. In the first part we asked students to implement a controller in Python. We've since removed that portion of the project, but you may find the**T solution implementation**helpful to consult as a reference.

Make sure you have cloned the repository and gotten familiar with the C++ environment as outlined in **C++ Setup**.

Complete each of the scenarios outlined in the **C++ project readme**. This will involve implementing and tuning controllers incrementally:

Intro (scenario 1)

Body rate and roll/pitch control (scenario 2)

Position/velocity and yaw angle control (scenario 3)

Non-idealities and robustness (scenario 4)

For this project, you will be building a controller in C++. You will be implementing and tuning this controller in several steps.

You may find it helpful to consult the Python controller code as a reference when you build out this controller in C++.

Notes on Parameter Tuning

Comparison to Python: Note that the vehicle you'll be controlling in this portion of the project has different parameters than the vehicle that's controlled by the Python code linked to above. **The tuning parameters that work for the Python controller will not work for this controller**

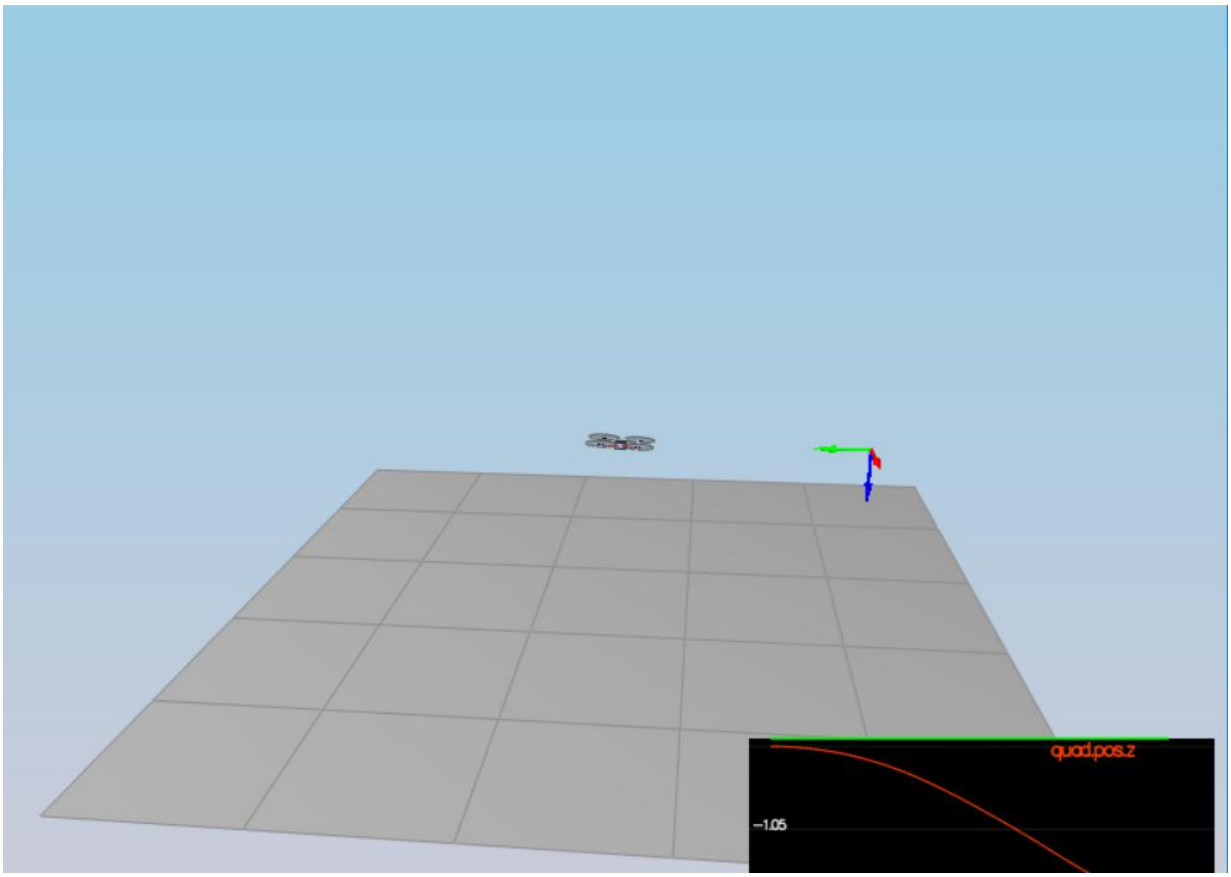
Parameter Ranges: You can find the vehicle's control parameters in a file called QuadControlParams.txt. The default values for these parameters are all too small by a factor of somewhere between about 2X and 4X. So if a parameter has a starting value of 12, it will likely have a value somewhere between 24 and 48 once it's properly tuned.

Parameter Ratios: In this one-page document you can find a derivation of the ratio of velocity proportional gain to position proportional gain for a critically damped double integrator system. The ratio of kpV / kpP should be 4.

Intro (Scenario 1);

In this scenario, we adjust the mass of the drone in **/cpp/config/QuadControlParams.txt** until it hovers for a bit:

When the scenario is passing the test, you should see this line on the standard output:



Results of scenario 1

```
Simulation #1891 (../config/1_Intro.txt)
PASS: ABS(Quad.PosFollowErr) was less than 0.500000 for at least 0.800000 seconds
```

Body rate and roll/pitch control (scenario 2)

First, you will implement the body rate and roll / pitch control. For the simulation, you will use Scenario 2. In this scenario, you will see a quad above the origin. It is created with a small initial rotation speed about its roll axis. Your controller will need to stabilize the rotational motion and bring the vehicle back to level attitude.

To accomplish this, you will:

Implement body rate control

- implement the code in the function `GenerateMotorCommands()`
- implement the code in the function `BodyRateControl()`
- Tune `kpPQR` in `QuadControlParams.txt` to get the vehicle to stop spinning quickly but not overshoot

If successful, you should see the rotation of the vehicle about roll (omega.x) get controlled to 0 while other rates remain zero. Note that the vehicle will keep flying off quite quickly, since the angle is not yet being controlled back to 0. Also note that some overshoot will happen due to motor dynamics!.

If you come back to this step after the next step, you can try tuning just the body rate omega (without the outside angle controller) by setting QuadControlParams.kpBank = 0.

Implement roll / pitch control We won't be worrying about yaw just yet.

- implement the code in the function RollPitchControl()
- Tune kpBank in QuadControlParams.txt to minimize settling time but avoid too much overshoot

The roll-pitch controller is a P controller responsible for commanding the roll and pitch rates in the body frame. First, it sets the desired rate of change of the given matrix elements using a P controller.

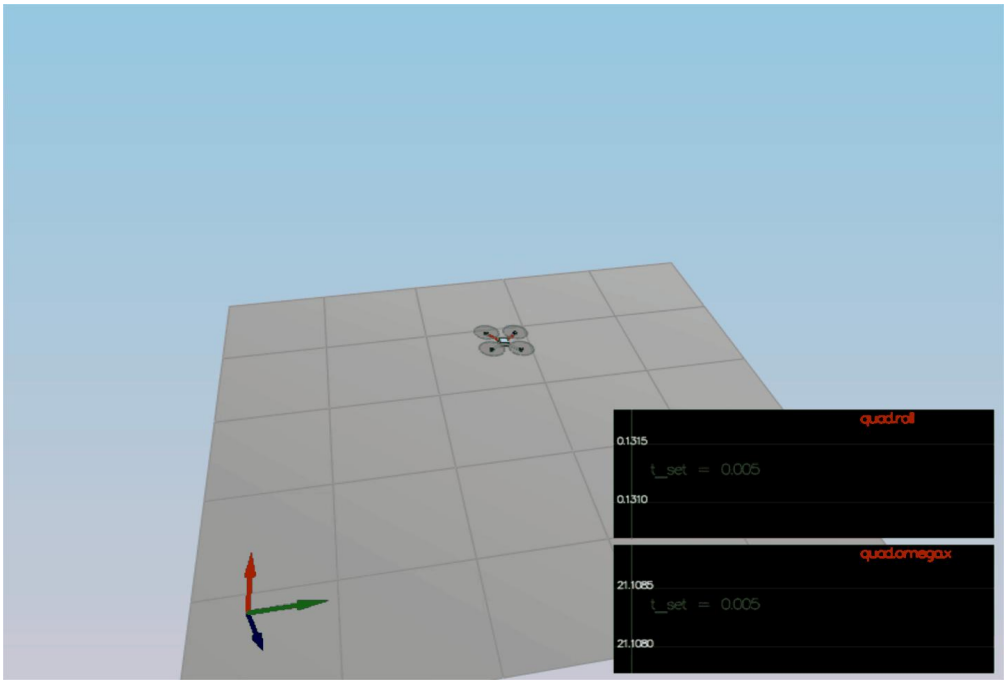
$$\dot{b}_c^x = k_p(b_c^x - b_a^x)$$

$$\dot{b}_c^y = k_p(b_c^y - b_a^y)$$

where $b_a^x = R_{13}$ and $b_a^y = R_{23}$. The given values can be converted into the angular velocities into the body frame by the next matrix multiplication.

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

If successful you should now see the quad level itself (as shown below), though it'll still be flying away slowly since we're not controlling velocity/position! You should also see the vehicle angle (Roll) get controlled to 0.



Result of scenario 2:

```
Simulation #1852 (../config/2_AttitudeControl.txt)
PASS: ABS(Quad.Roll) was less than 0.025000 for at least 0.750000 seconds
PASS: ABS(Quad.Omega.X) was less than 2.500000 for at least 0.750000 seconds
```

Position/velocity and yaw angle control (scenario 3)

Next, you will implement the position, altitude and yaw control for your quad. For the simulation, you will use Scenario 3. This will create 2 identical quads, one offset from its target point (but initialized with yaw = 0) and second offset from target point but yaw = 45 degrees.

- implement the code in the function LateralPositionControl(): This is another PID controller to control acceleration on x and y.
- implement the code in the function AltitudeControl(): This is a PD controller to control the acceleration meaning the thrust needed to control the altitude.
- tune parameters kpPosZ and kiPosZ:
- tune parameters kpVelXY and kpVelZ.

At start , all the parameters were low number and tests scenario was failing . Later , I tried increasing these value by 2 to 3 times . Finally , I settled down with these Numbers

```
# Position control gains
kpPosXY = 30
kpPosZ = 20
KiPosZ = 40

# Velocity control gains
kpVelXY = 12
kpVelZ = 8
```

AltitudeControl:

If successful, the quads should be going to their destination points and tracking error should be going down (as shown below). However, one quad remains rotated in yaw.

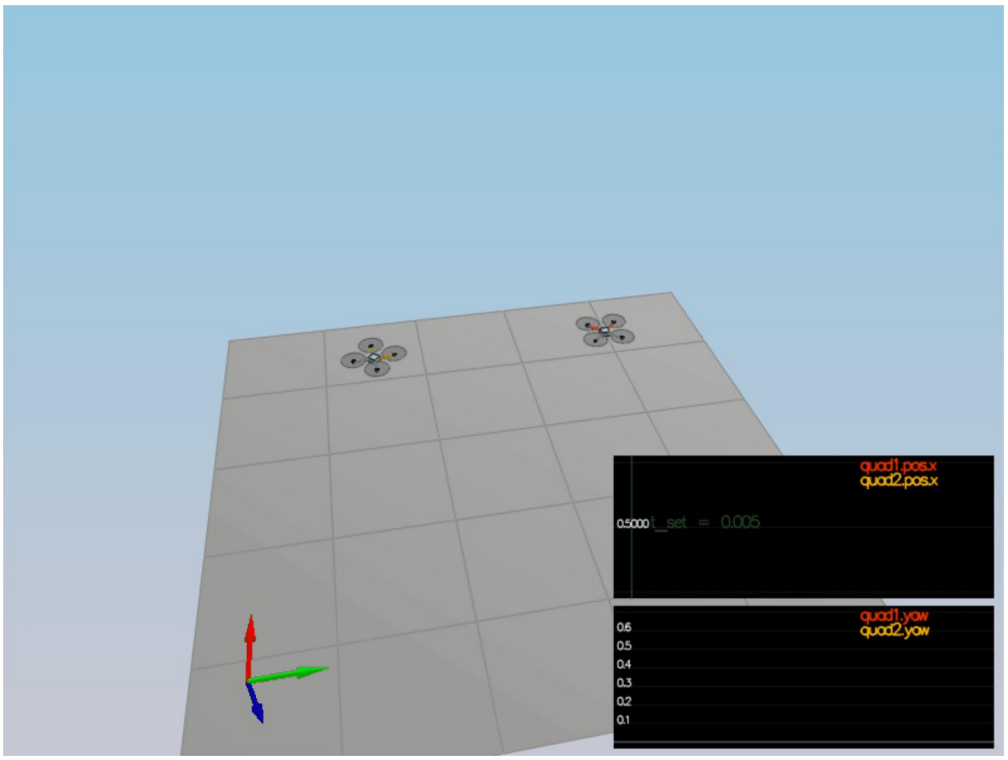
- implement the code in the function YawControl(): This is a simpler case because it is P controller. It is better to optimize the yaw to be between [-pi, pi].
- tune parameters kpYaw and the 3rd (z) component of kpPQR

At start , all the parameters were low number and tests scenario was failing . Later , I tried increasing these value by 2 to 3 times . Finally , I settled down with these Numbers

```
# Angle control gains
kpBank = 10
kpYaw = 2

# Angle rate gains
kpPQR = 70, 70, 5
```

Tune position control for settling time. Don't try to tune yaw control too tightly, as yaw control requires a lot of control authority from a quadcopter and can really affect other degrees of freedom. This is why you often see quadcopters with tilted motors, better yaw authority!



Results of senario

3:

```
Simulation #1969 (../config/3_PositionControl.txt)
PASS: ABS(Quad1.Pos.X) was less than 0.100000 for at least 1.250000 seconds
PASS: ABS(Quad2.Pos.X) was less than 0.100000 for at least 1.250000 seconds
PASS: ABS(Quad2.Yaw) was less than 0.100000 for at least 1.000000 seconds
```

Non-idealities and robustness (scenario 4)

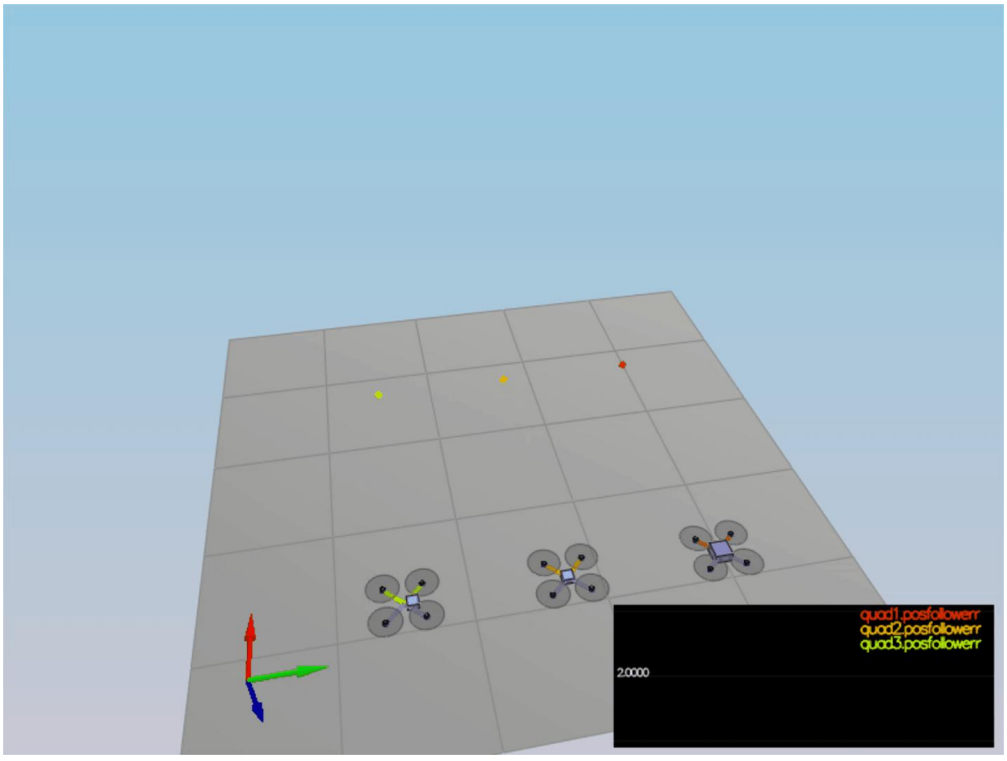
In this part, we will explore some of the non-idealities and robustness of a controller. For this simulation, we will use Scenario 4. This is a configuration with 3 quads that are all are trying to move one meter forward. However, this time, these quads are all a bit different:

- The green quad has its center of mass shifted back
- The orange vehicle is an ideal quad
- The red vehicle is heavier than usual

Run your controller & parameter set from Step 3. Do all the quads seem to be moving OK? If not, try to tweak the controller parameters to work for all 3 (tip: relax the controller).

Edit AltitudeControl() to add basic integral control to help with the different-mass vehicle.

Tune the integral control, and other control parameters until all the quads successfully move properly. Your drones' motion should look like this:



Results of scenario 4:

```
Simulation #1991 (../config/4_Nonidealities.txt)
PASS: ABS(Quad1.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds
PASS: ABS(Quad2.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds
PASS: ABS(Quad3.PosFollowErr) was less than 0.100000 for at least 1.500000 seconds
```

Tracking trajectories

Now that we have all the working parts of a controller, you will put it all together and test it's performance once again on a trajectory. For this simulation, you will use Scenario 5. This scenario has two quadcopters:

- the orange one is following traj/FigureEight.txt
- the other one is following traj/FigureEightFF.txt - for now this is the same trajectory. For those interested in seeing how you might be able to improve the performance of your drone by adjusting how the trajectory is defined, check out **Extra Challenge 1** below!

How well is your drone able to follow the trajectory? It is able to hold to the path fairly well?

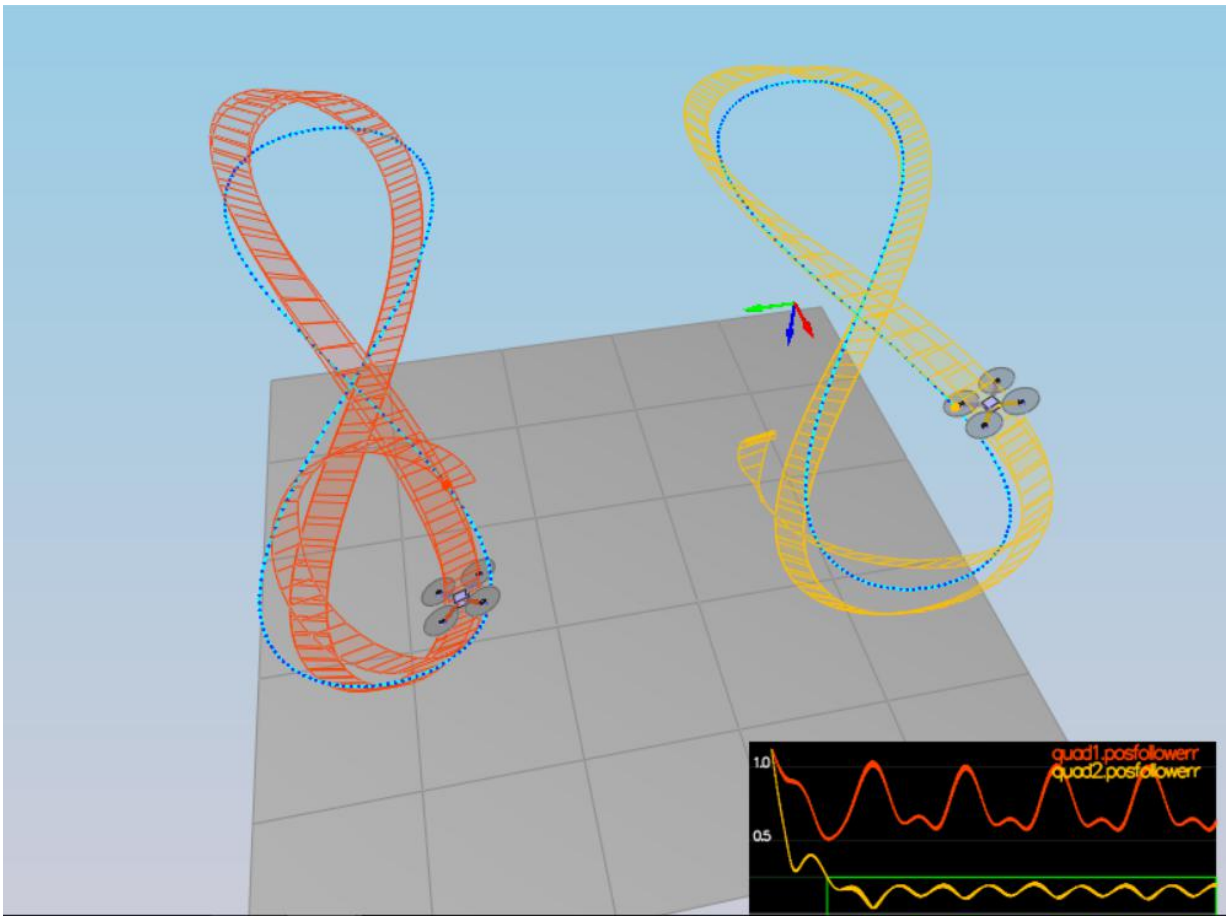
Extra Challenge 1 (Optional)

You will notice that initially these two trajectories are the same. Let's work on improving some performance of the trajectory itself.

Inspect the python script traj/MakePeriodicTrajectory.py. Can you figure out a way to generate a trajectory that has velocity (not just position) information?

Generate a new FigureEightFF.txt that has velocity terms Did the velocity-specified trajectory make a difference? Why?

With the two different trajectories, your drones' motions should look like this:



Extra Challenge 2 (Optional)

For flying a trajectory, is there a way to provide even more information for even better tracking?

How about trying to fly this trajectory as quickly as possible (but within following threshold)!

Evaluation

To assist with tuning of your controller, the simulator contains real time performance evaluation. We have defined a set of performance metrics for each of the scenarios that your controllers must meet for a successful submission.

There are two ways to view the output of the evaluation:

- in the command line, at the end of each simulation loop, a **PASS** or a **FAIL** for each metric being evaluated in that simulation
- on the plots, once your quad meets the metrics, you will see a green box appear on the plot notifying you of a **PASS**

Performance Metrics

The specific performance metrics are as follows:

scenario 2

- roll should less than 0.025 radian of nominal for 0.75 seconds (3/4 of the duration of the loop)
- roll rate should less than 2.5 radian/sec for 0.75 seconds

scenario 3

- X position of both drones should be within 0.1 meters of the target for at least 1.25 seconds
- Quad2 yaw should be within 0.1 of the target for at least 1 second

scenario 4

- position error for all 3 quads should be less than 0.1 meters for at least 1.5 seconds

scenario 5

- position error of the quad should be less than 0.25 meters for at least 3 seconds

Conclusion

Over all , this was good project which pretty much give idea about vehicle dynamics and 3D motion control for drone .