Flying Car Nanodegree – Build a Controller

Writeup

Implemented Controller:

Implemented body rate control in C++:

- A **proportional** controller is implemented as a body rate control
- The proportional error is calculated by the difference between the desired body rates and the estimated body rates in rad/s
- The product of this error along with the Moment of inertias across all 3 directions and the proportional gain (`kpPQR`) gives the final moment for the body rate control.
- This was handled in the function **`BodyRateControl`**`()`.

Implement roll pitch control in C++:

- The roll and pitch angle rates are calculated using roll-pitch controller.
- Convert the collective thrust in newton to acceleration by scaling thrust with 1/mass.
- Estimate the pitch and roll rates from the obtained desired lateral acceleration and constrain it between the min and max tilt angles using the acceleration obtained from total thrust.
- Find the deviation of the estimated angles with the rotation matrix entries.
- Product with kpBank gain parameter helps in finding the roll and pitch angles by Rotation and transformation of the obtained error in X and y direction using the Rotation matrix.
- The pitch and roll angles are set to 0; if total thrust is negative.
- This was implemented in the function **`RollPitchControl`**`()`

Implement altitude controller in C++:

- In order to successfully implement scenario 3 and 4, a **PID controller** is implemented as an altitude controller.
- Find Positional (Proportional), velocity (Derivative) and steady state error (Integral) errors using the targets and estimates values (position and velocity)
- Scale each error with its corresponding gain parameters (`kpPosZ, kpVelZ, KiPosZ`).
- Get the sum of P, I and D terms along with the feed forward vertical acceleration to get the acceleration value. Reduce the impact of the gravity.
- Constraint the obtained acceleration between maximum ascent and decent rate.
- Scaling the acceleration with mass of vehicle yields the necessary thrust
- This was developed inside the method **`AltitudeControl`**`()`.

Implement lateral position control in C++:

- Lateral position control helps in estimating the horizontal acceleration needed for the drone.
- Find the positional and velocity error (between actual and desired).
- Scaling the errors with the respective gains `kpPosXY`, `kpVelXY` `kpVelXY` helps in finding the acceleration along with the addition to feed forward acceleration.
- Make the z component of acceleration to 0.
- Constraint the acceleration and the velocity to be within the +- max limits.
- This process was implemented in the function **`LateralPositionControl`**`()`.

Implement yaw control in C++:

- Calculate the yaw rate for the vehicle using a Proportional controller
- Unwrap the yaw in radian angle measurement to range (-2pi to 2pi)
- Find the deviation between actual and desired yaw rates.
- Wrap the error again within the range.
- Scale the error with the proportional gain `kpYaw`.
- This was performed in the function **`YawControl`**`()`

Implement calculating the motor commands given commanded thrust and moments in C++:

- The moments and thrust are converted into 4 different thrust forces for each motor.
- The moment along the x and y direction is scaled by inverse of arm length / sqrt(2).
- The moment in the z direction is scaled by inverse of drag to thrust ratio (kappa)
- The thrust for each of the motors namely front left, right; rear left, right using the moments and the total thrust as implemented in the function **`GenerateMotorCommands`**`()` based on the formulas provided in the lecture.

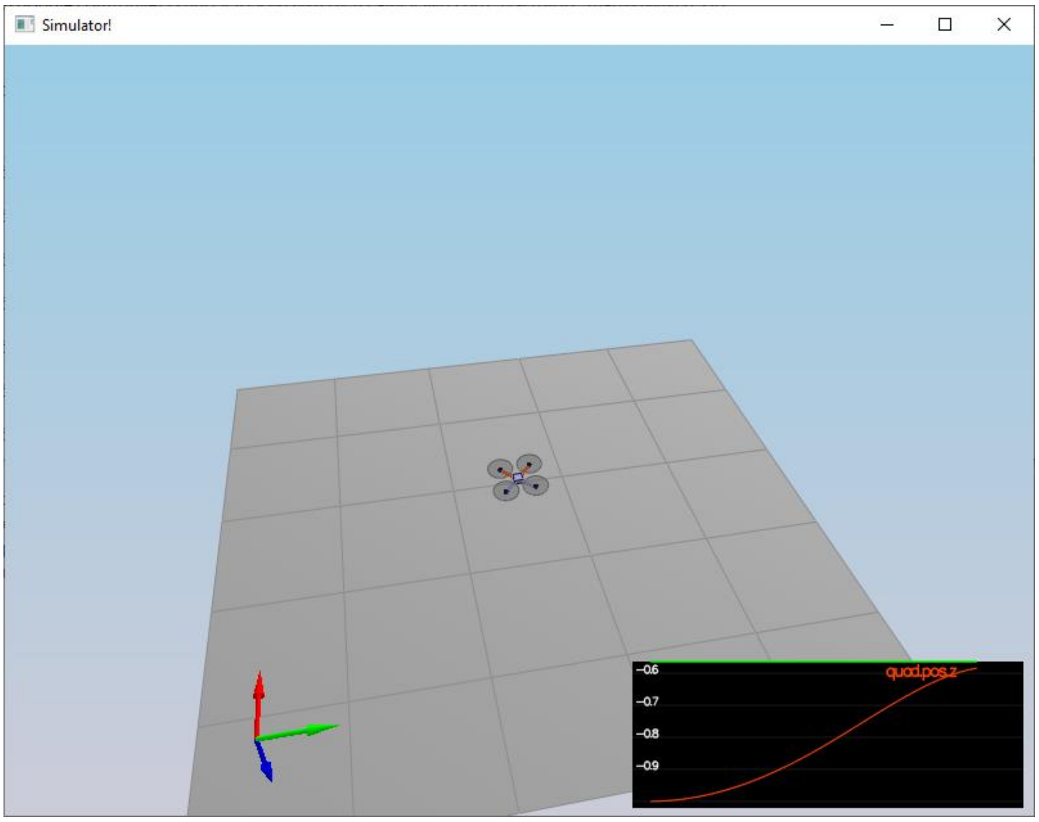
Flight Evaluation

Your C++ controller is successfully able to fly the provided test trajectory and visually passes inspection of the scenarios leading up to the test trajectory:

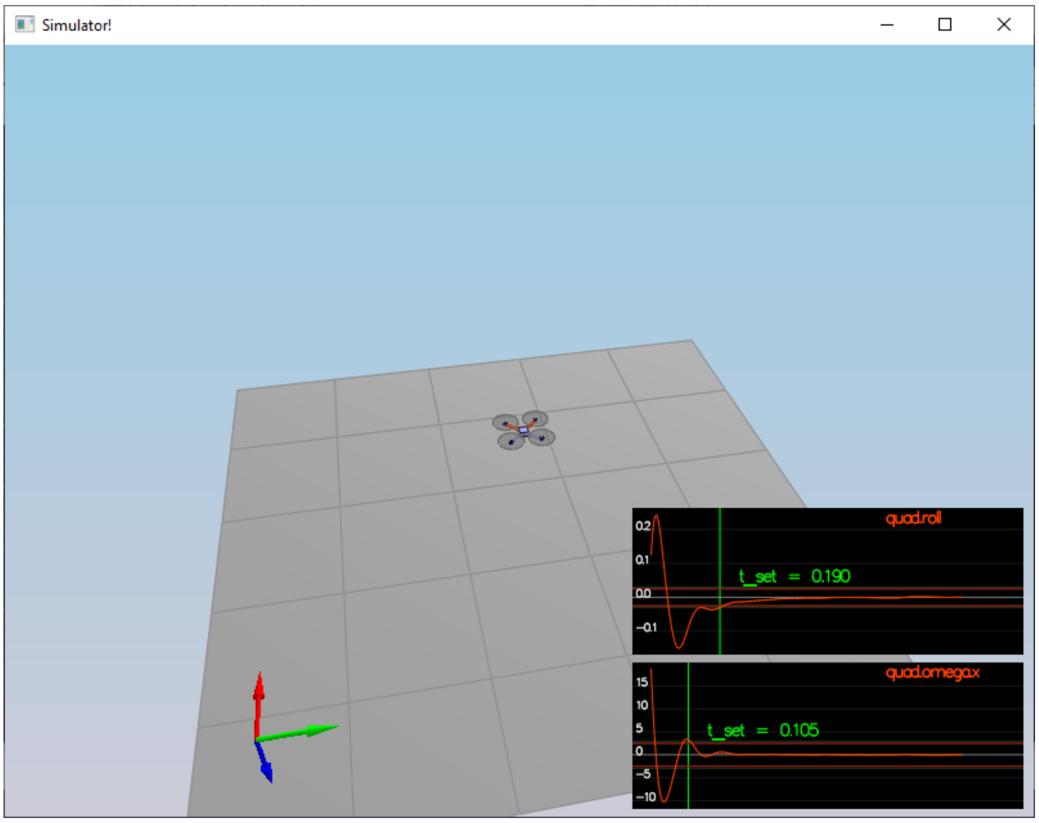
The drone passes all the tests in the provided scenarios (1 to 5) based on the observation in the simulation and command window output.

The output of the scenarios are attached in the following pages.

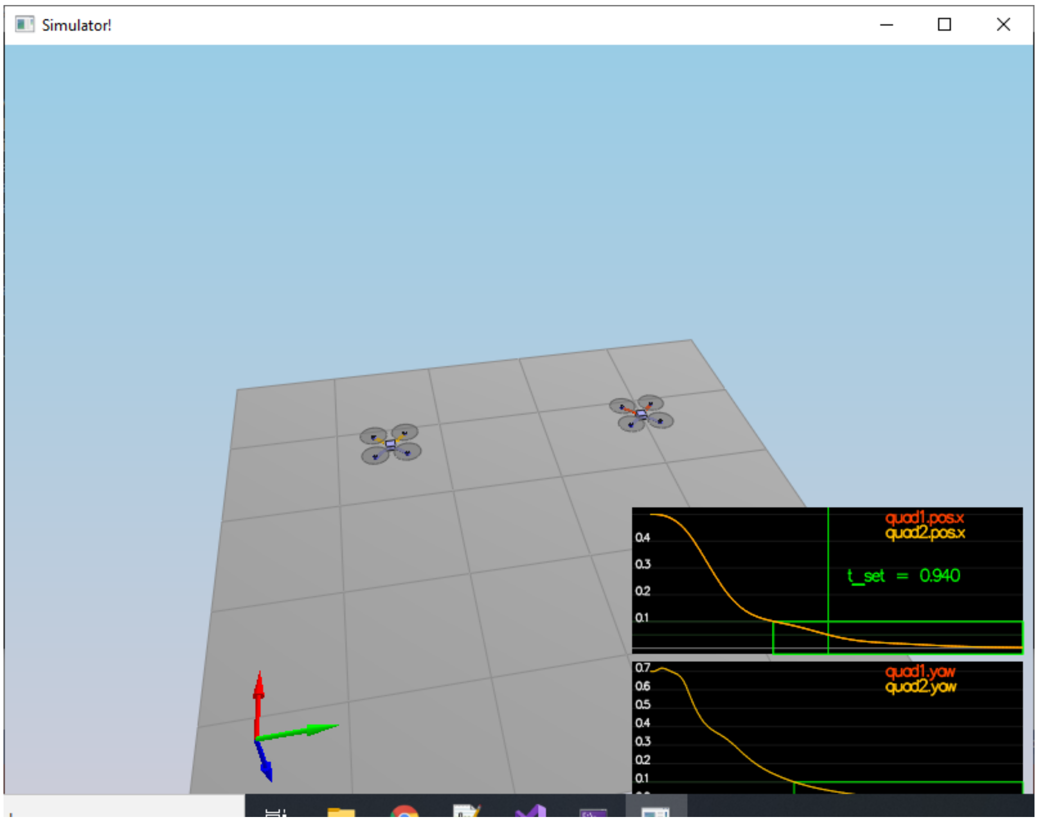
Scenario 1:



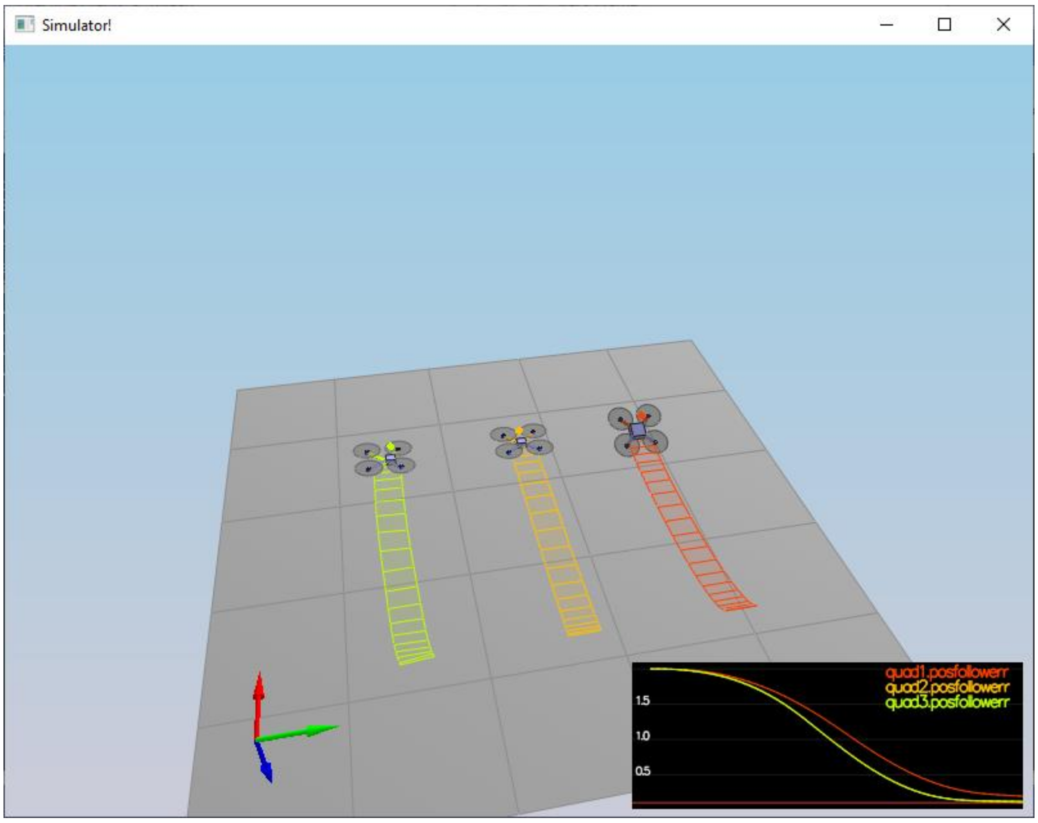
Scenario 2:



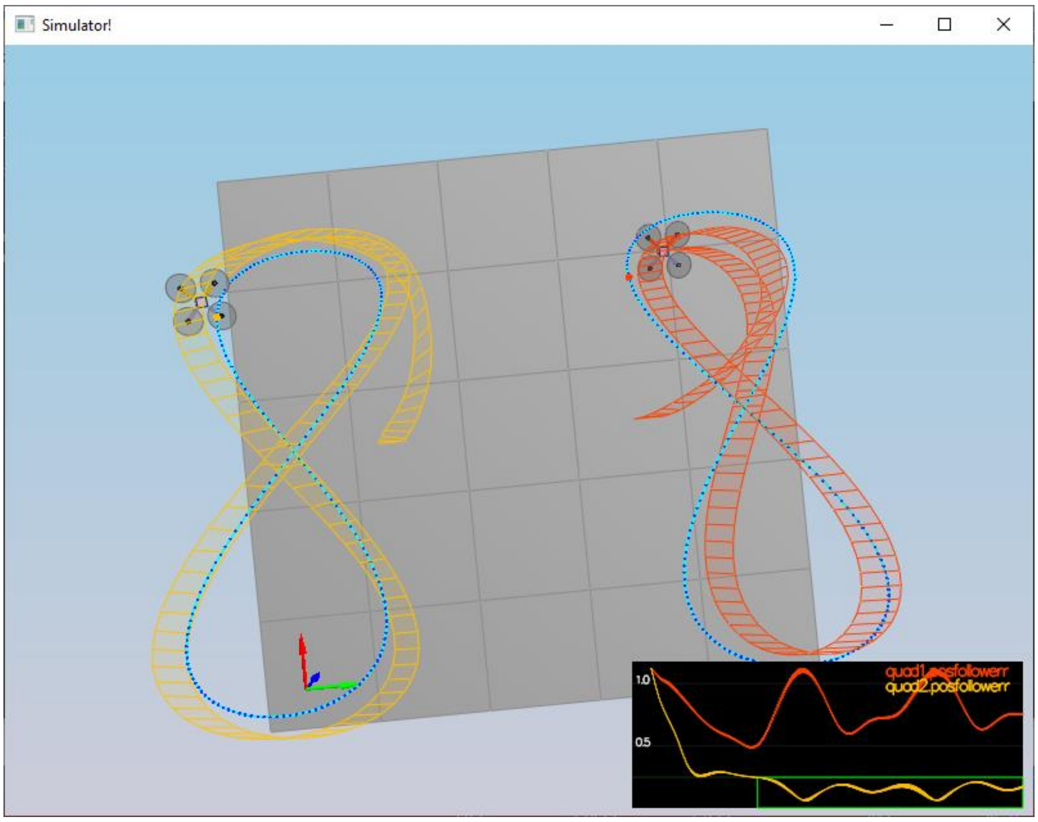
Scenario 3:



Scenario 4:



Scenario 5:


Simulator!
1.0
0.5
quad1.posfollowerr
quad2.posfollowerr