This is the FCND Building a Controller Project Markdown

- <u>GenerateMotorCommands Function:-</u>

  -Here I've assigned the tau_X, tau_y & tau_z values

```
    float tau_x = momentCmd[0];
    float tau_y = momentCmd[1];
    float tau_z = momentCmd[2];
```

  -defining 4 equations for the 4 motors thrust variables

```
    float eq1 = collThrustCmd;            // f1 + f2 + f3 + f4
    float eq2 = tau_x / (L / sqrtf(2.f)); // f1 - f2 - f3 + f4
    float eq3 = tau_y / (L / sqrtf(2.f)); // f1 + f2 - f3 - f4
    float eq4 = -tau_z / kappa;           // f1 - f2 + f3 - f4
```

  -solving the equations to get the thrust value for each motor, also limiting these values to the motor thrust limits, and eventually assigning these values to the commanded desired thrust cmd.desiredThrustsN

```
    float f1 = (eq1 + eq2 + eq3 + eq4) / 4.f;
    float f4 = (eq1 + eq2 - f1 * 2.f) / 2.f;
    float f3 = (-eq2 - eq3 + 2.f * f1) / 2.f;
    float f2 = eq1 - f1 - f3 - f4;

    f1 = CONSTRAIN(f1, minMotorThrust, maxMotorThrust);
    f2 = CONSTRAIN(f2, minMotorThrust, maxMotorThrust);
    f3 = CONSTRAIN(f3, minMotorThrust, maxMotorThrust);
    f4 = CONSTRAIN(f4, minMotorThrust, maxMotorThrust);

    cmd.desiredThrustsN[0] = f1;
    cmd.desiredThrustsN[1] = f2;
    cmd.desiredThrustsN[2] = f4;
    cmd.desiredThrustsN[3] = f3;
```

- <u>BodyRateControl Function:-</u>

  -getting u_bar_pqr and multiply it to the rotational moment of inertia to get the desired commanded moments while controlling it through the KpPQR gains

```

    V3F tau_xyz = (Ixx, Iyy, Izz);
    momentCmd = tau_xyz*(kpPQR*(pqrCmd - pqr));

```

- <u>RollPitchControl Function:-</u>

  -Here I've assigned the desired vertical acceleration C, and made a condition to make sure that the returned collective thrust is always positive.

  -Then limiting the roll & pitch values. After that we get the controlled commanded Pitch & Roll values through KpBank (KpPitch & KpRoll) gains and assigning them to pqrCmd vector.

```
    float c = -collThrustCmd / mass;
    float pCmd;
    float qCmd;

    if (collThrustCmd > 0.0) {
        float b_x_c = CONSTRAIN(accelCmd.x / c, -maxTiltAngle, maxTiltAngle);
        float b_y_c = CONSTRAIN(accelCmd.y / c, -maxTiltAngle, maxTiltAngle);

        pCmd = (1 / R(2, 2)) * (R(1, 0) * kpBank * (b_x_c - R(0, 2))
            - R(0, 0) * kpBank * (b_y_c - R(1, 2)));

        qCmd = (1 / R(2, 2)) * (R(1, 1) * kpBank * (b_x_c - R(0, 2))
            - R(0, 1) * kpBank * (b_y_c - R(1, 2)));
    }
    else {
        cout << "Negative thrust command" << "\n";
        pCmd = 0.0f;
        qCmd = 0.0f;
    }

    pqrCmd.x = pCmd;
    pqrCmd.y = qCmd;
```

- <u>AltitudeControl Function:-</u>

  -In this function we get the controlled u_1_bar through KpZ & KiZ gains as it's a nonlinear system, KpZ_dot gain, and the feed forward parameter.

  -After getting the desired vertical acceleration value we limit it, then converting it to the desired collective thrust value (the negative here due to the NED coordination).

```
    float b_z = R(2, 2);
    integratedAltitudeError += (posZCmd - posZ) * dt;
    float u_1_bar = kpPosZ * (posZCmd - posZ) + kpVelZ * (velZCmd - velZ) + accelZCmd + KiPosZ * integratedAltitudeError;
    float c = (u_1_bar - 9.81f) / b_z;
    c = CONSTRAIN(c, -maxDescentRate / dt, maxAscentRate / dt);
    thrust = -c * mass;
```

- <u>LateralPositionControl Function:-</u>

  -Here we control the position of the drone in the X & Y Directions (NE) through the commanded & the actual X & Y position and velocity, the feed forward X & Y acceleration parameters, and the KpX & KpX_dot & KpY & KpY_dot gains.

  -Then we limit the desired acceleration in the X & Y directions before returning them.

```
    accelCmd.x += kpPosXY * (posCmd.x - pos.x) + kpVelXY * (velCmd.x - vel.x) + accelCmdFF.x;
    accelCmd.y += kpPosXY * (posCmd.y - pos.y) + kpVelXY * (velCmd.y - vel.y) + accelCmdFF.y;
    accelCmd.x = CONSTRAIN(accelCmd.x, -maxAccelXY, maxAccelXY);
    accelCmd.y = CONSTRAIN(accelCmd.y, -maxAccelXY, maxAccelXY);
```

- <u>YawControl Function:-</u>

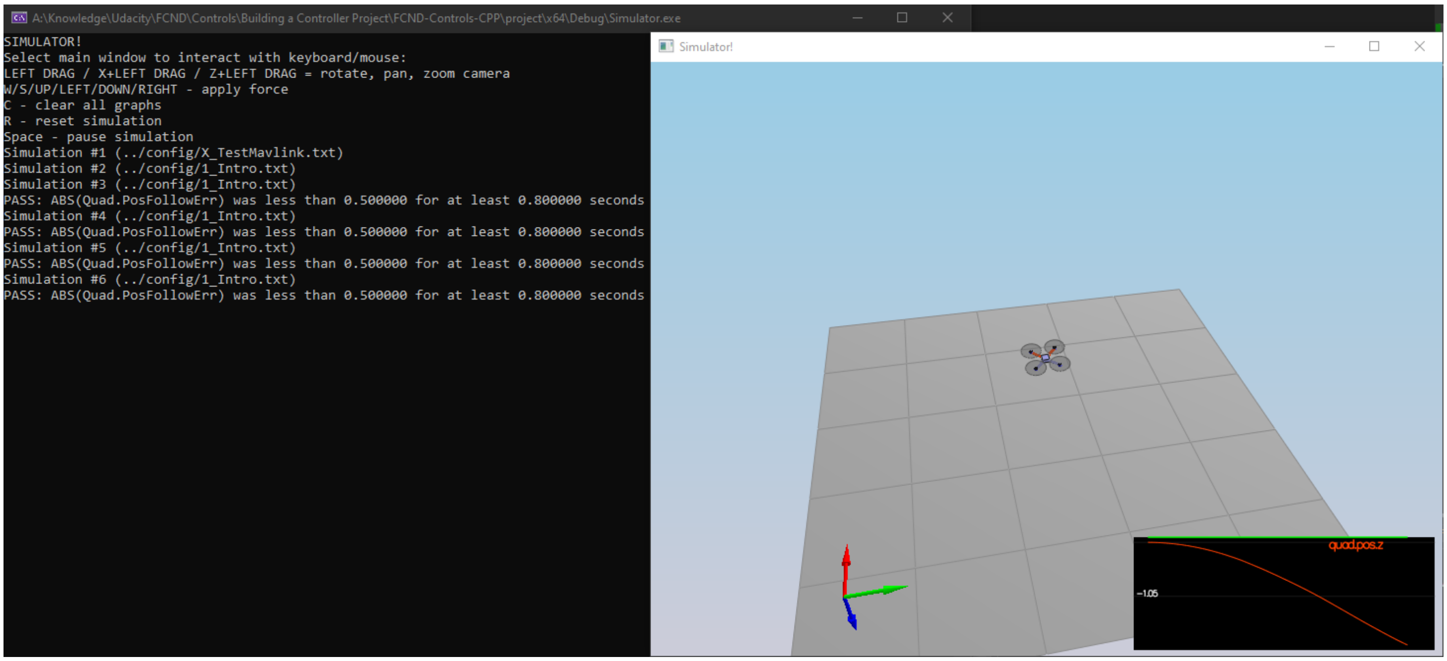
  -Here we control the yaw rate through the commanded & actual yaw rate and KpYaw gain after making sure that the yawCmd is always between 0~2Pi

```
    yawCmd = fmodf(yawCmd, 2 * 3.14159f);
    yawRateCmd = kpYaw * (yawCmd - yaw);
```

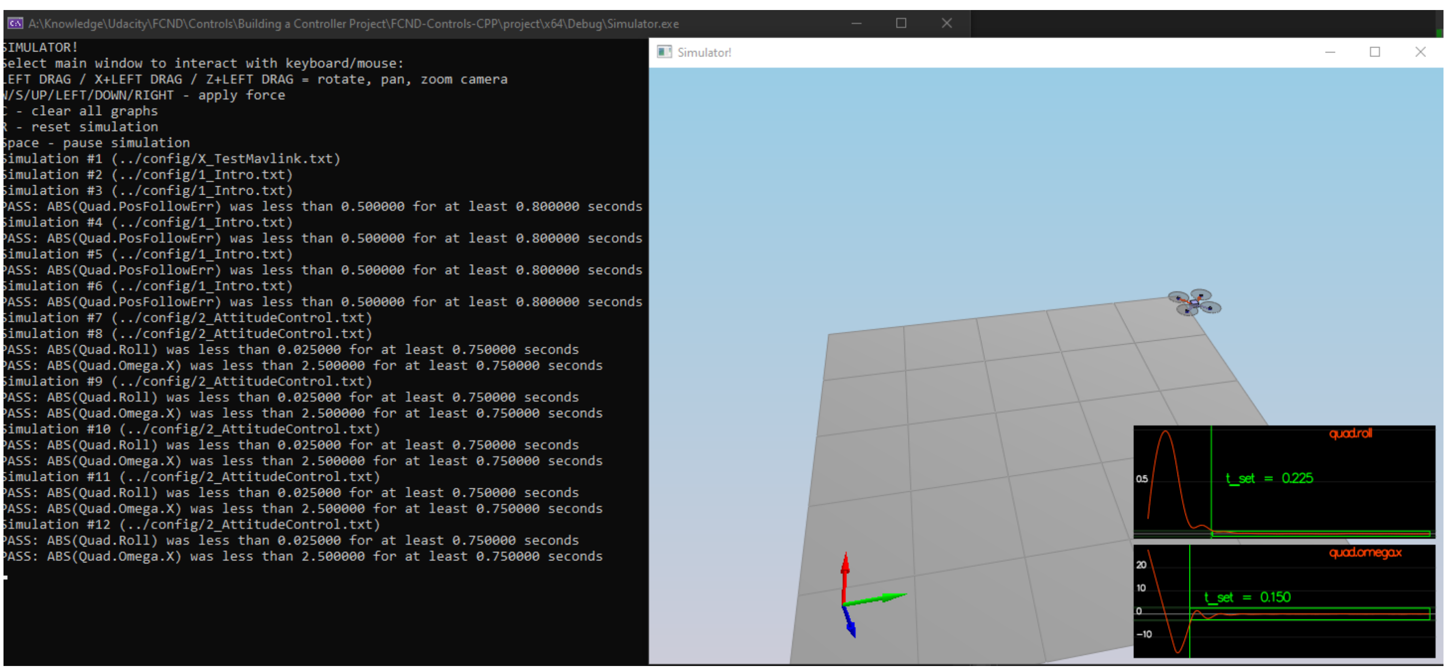
Here we're looking at some screen shots of the simulator:-

The **QuadControlParams** has been modified many times while coding each function and to pass all the scenarios, so this pictures are of the last modification

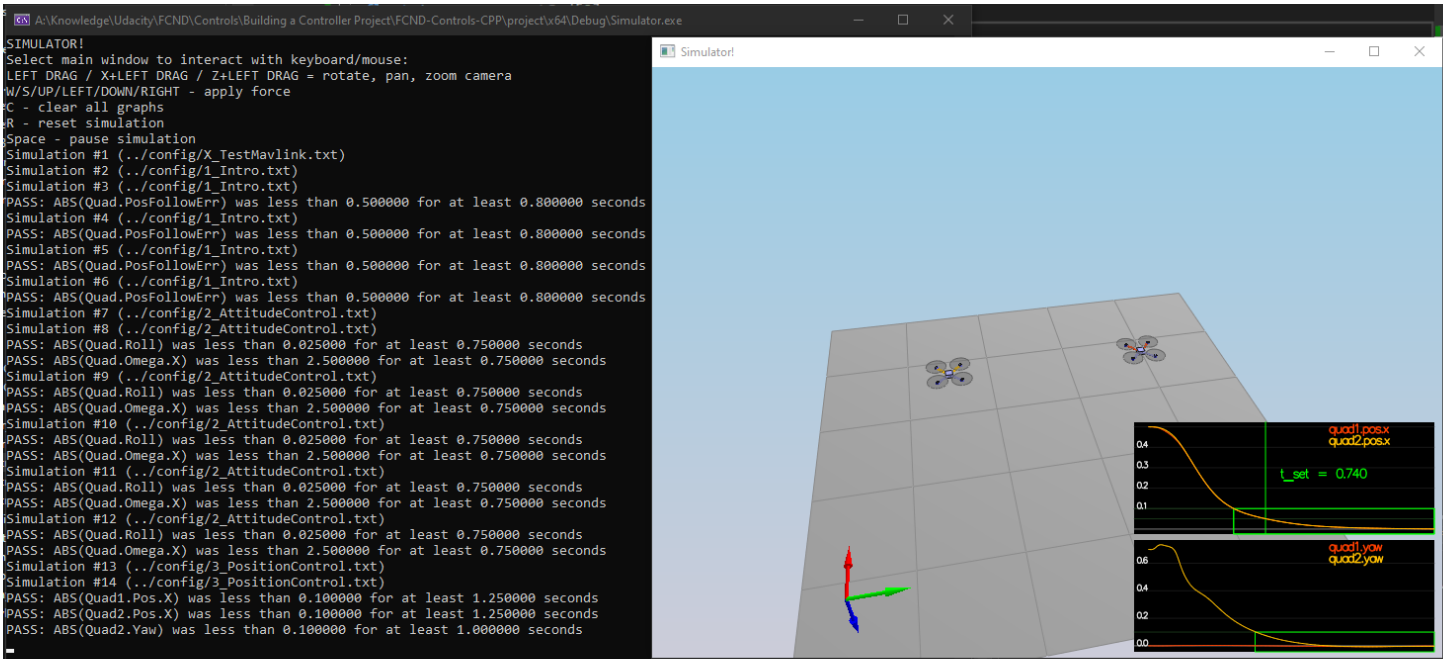
Intro Scenario



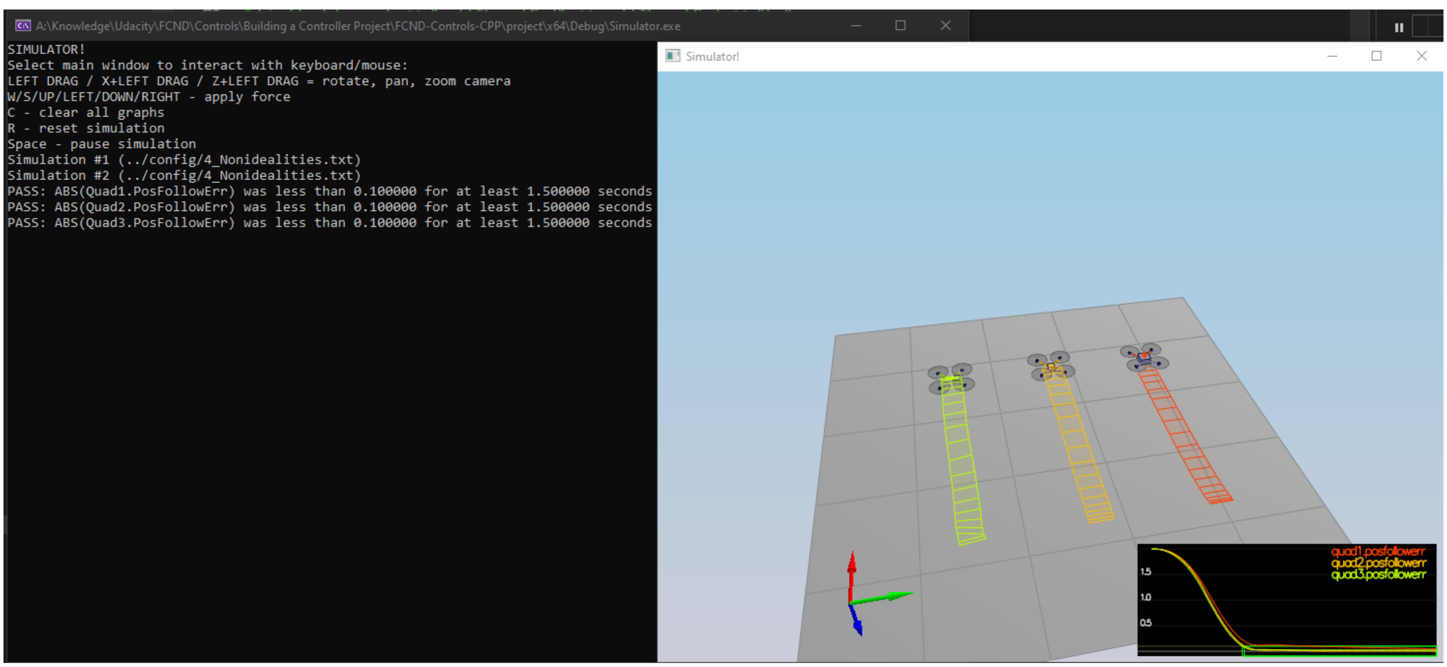
AttitudeControl Scenario



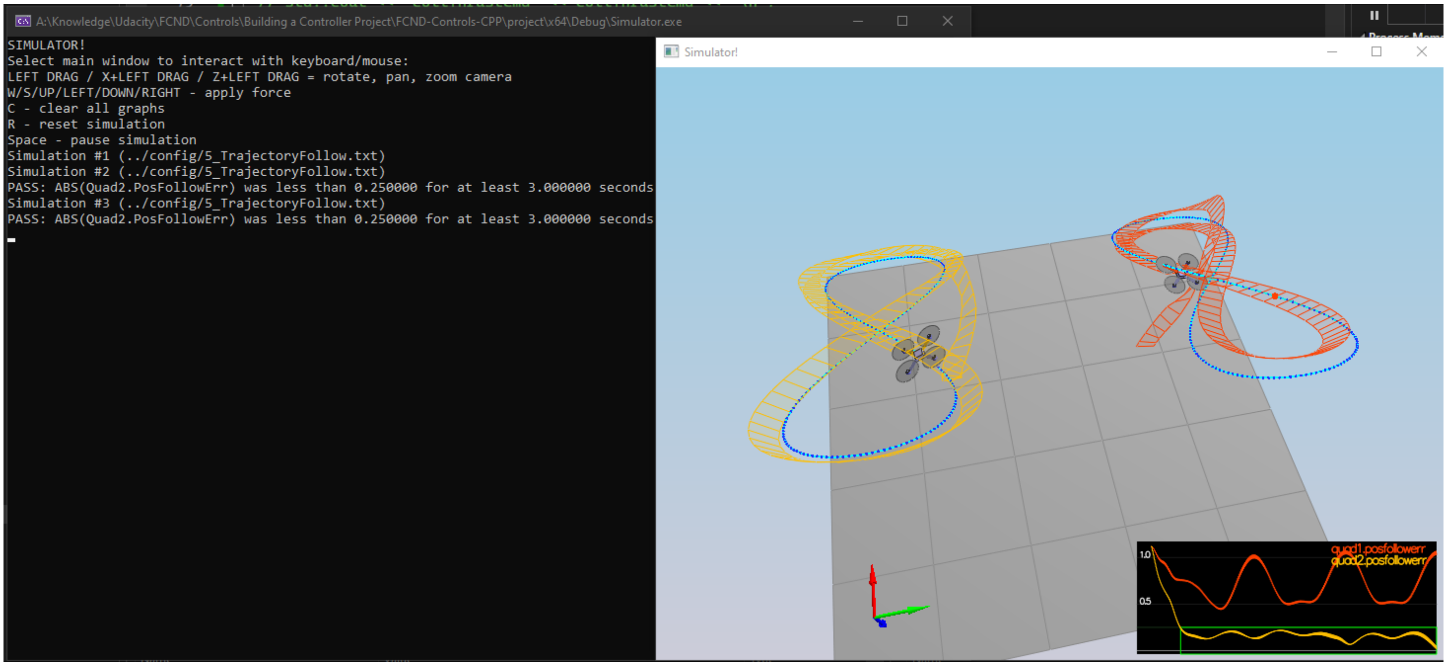
PositionControl Scenario



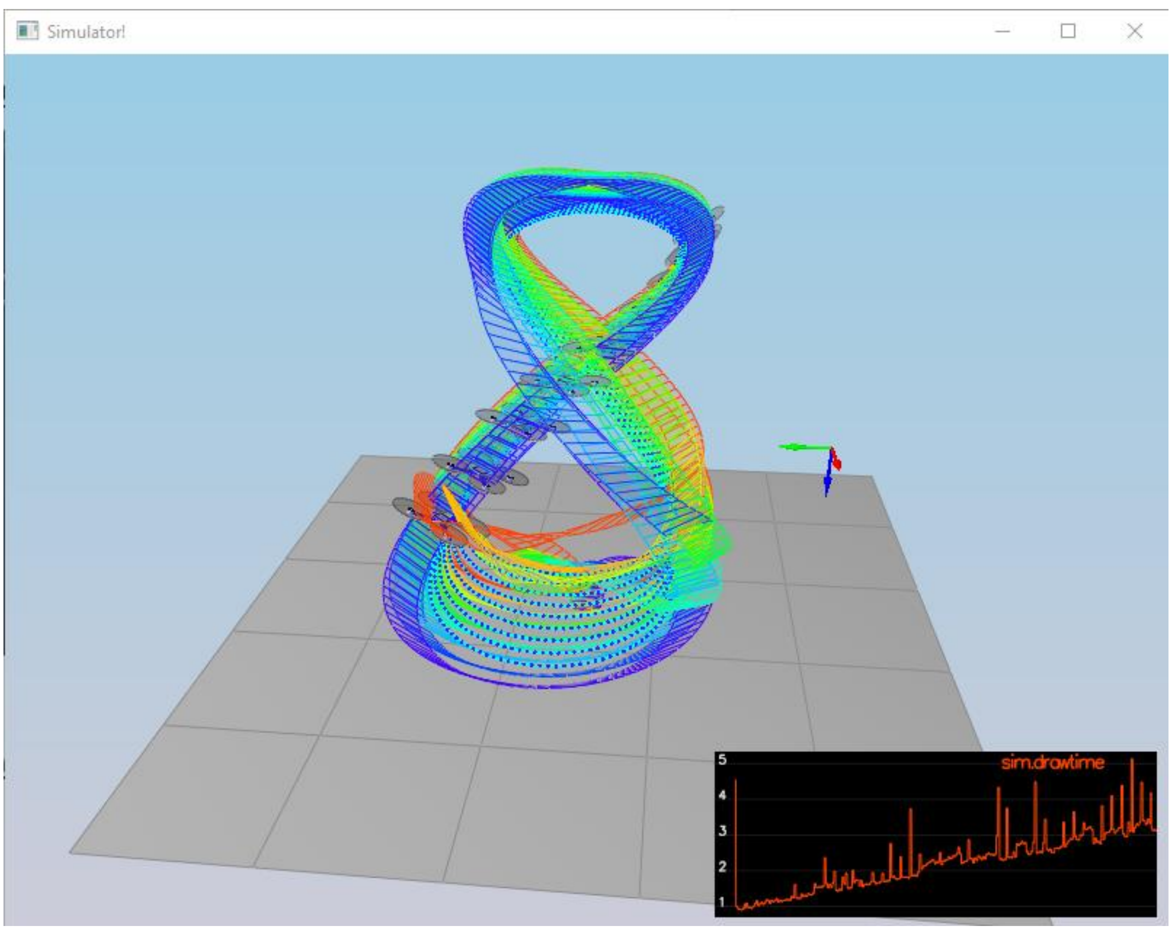
Nonidealities Scenario



<u>**TrajectoryFollow Scenario**</u>



<u>**Here is the scene from the X_TestManyQuads Scenario**</u>



Pretty cool, right? ;)