Project Name:- Estimation Project (FCND-Estimation-CPP)
Submitted by:- Bishwajit Kumar

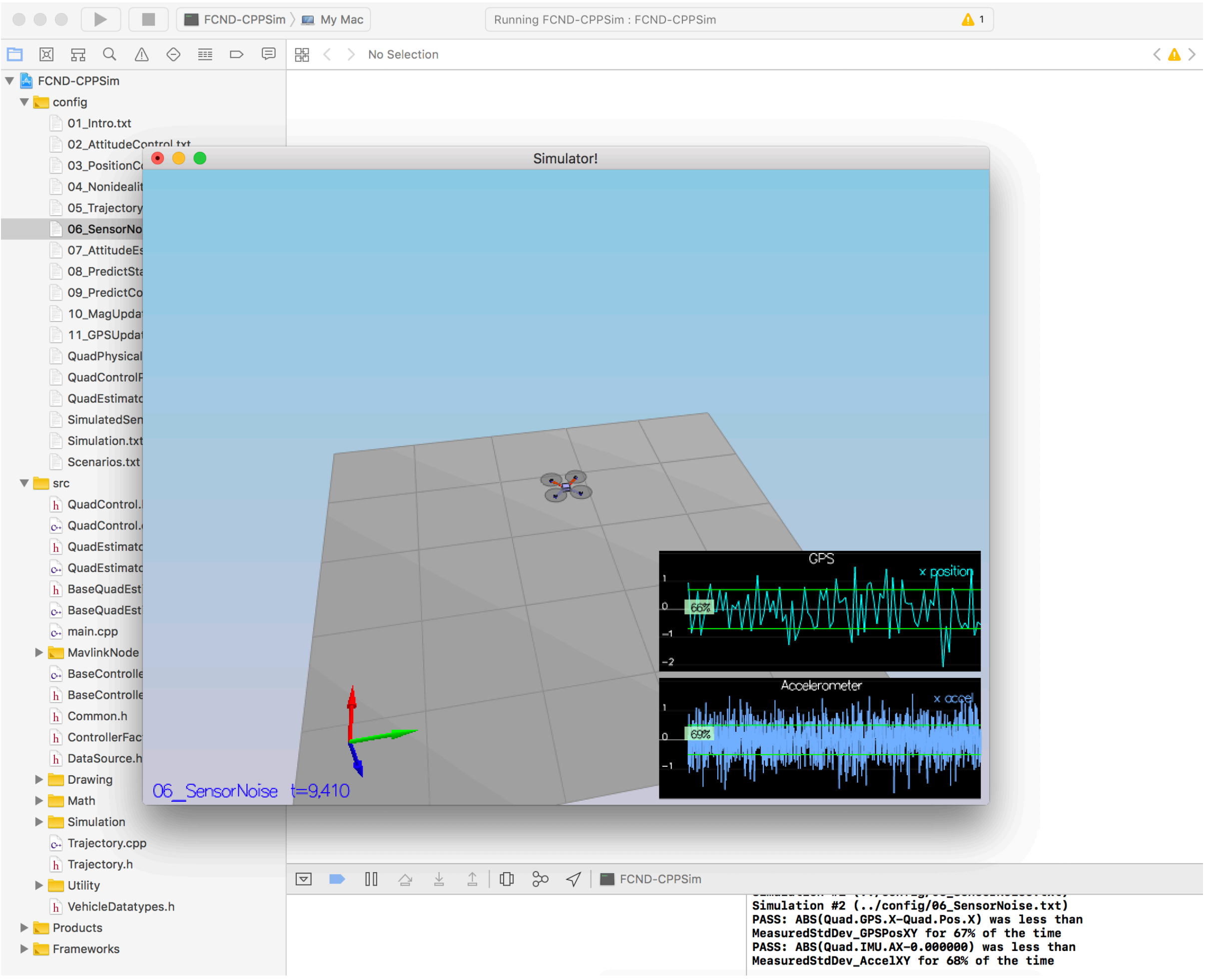
This project is to develop estimation portion of controller used in CPP simulator provided by Udacity.
This project uses custom controller developed in project 3 and extends it to include estimation part.

Project solution: -

Step 1: Sensor Noise

a. Ran the simulator after importing project in XCode
b. Selected 06_NoisySensors project which recorded data in **config/log/Graph1.txt** (GPS X data) and **config/log/Graph2.txt** (Accelerometer X data).

c. Processed the data and calculated standard deviation of the GPS X signal and the IMU Accelerometer X signal in file 06_Noisysensors_Calculate_GPS_Noise.xlsx and 06_Noisysensors_Accelerometer Noise.xlsx respectively
d. Modified config/6_Sensornoise.txt for MeasuredStdDev_GPSPosXY= 0.7 and MeasuredStdDev_AccelXY= 0.5
e. Ran the simulator and ABS(Quad.GPS.X-Quad.Pos.X) was less than MeasuredStdDev_GPSPosXY for 67% of the time And ABS(Quad.IMU.AX-0.000000) was less than MeasuredStdDev_AccelXY for 68% of the time.



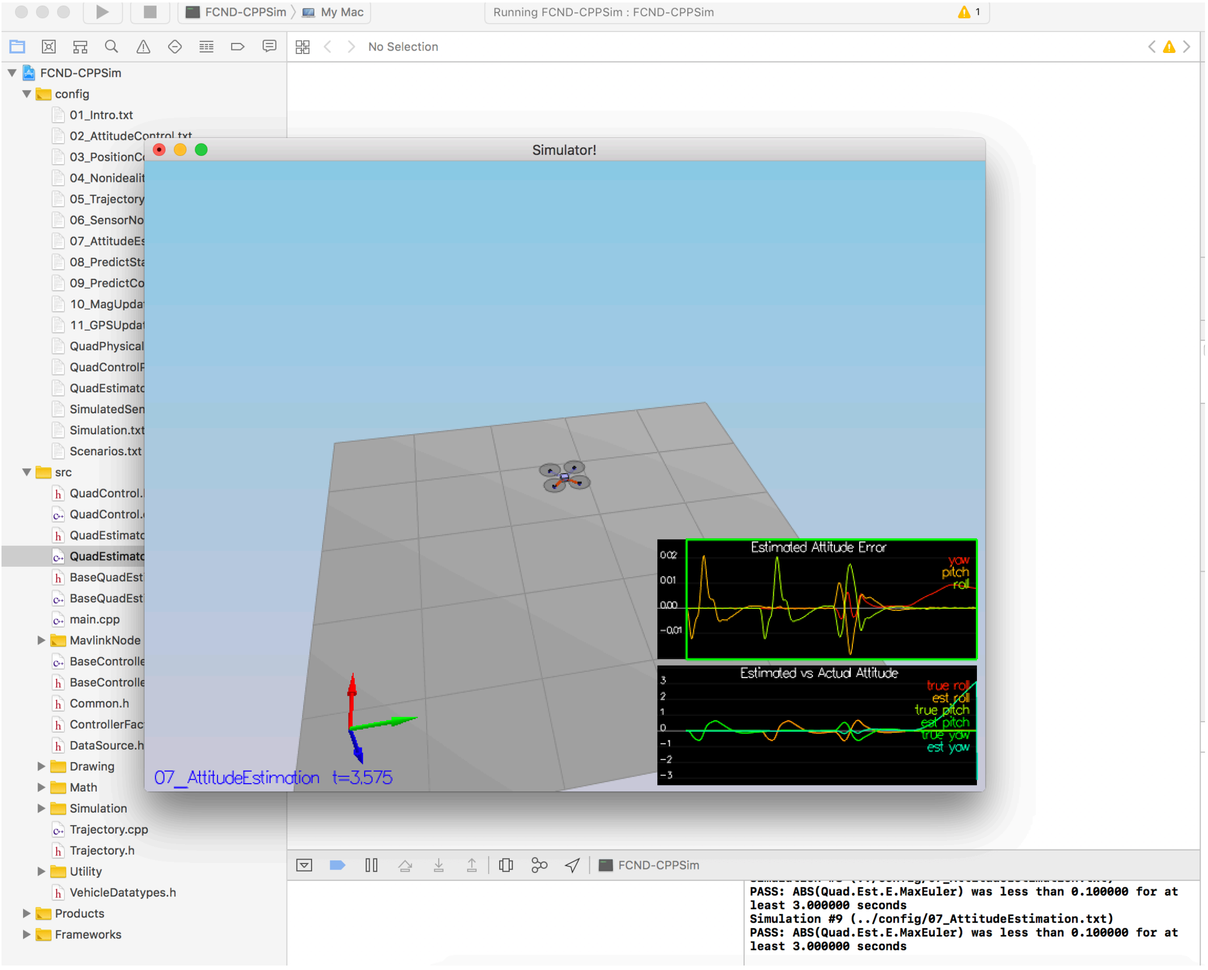
Step 2: Attitude estimation

a. Updated UpdateFromIMU() method where I created a Quartenion to retrieve drone estimated roll,pitch and yaw.

b. Integrated the Quaternion to and combined it with accelerometer data to create final attitude estimate.

```
Quaternion<float> droneAttitude = Quaternion<float>::FromEuler123_RPY(rollEst,
pitchEst, ekfState(6));
    V3D bodyRates = V3D(gyro.x, gyro.y, gyro.z);
    Quaternion<float> predictedAttitude =
droneAttitude.IntegrateBodyRate(bodyRates, dtIMU);

    float predictedPitch = predictedAttitude.Pitch();
    float predictedRoll = predictedAttitude.Roll();
    ekfState(6) = predictedAttitude.Yaw();
```



Step 3: Prediction step

a. Implemented state prediction step which rotates the acceleration vector from body to inertial frame.
b. After rotating it updates the state vector using dt to advance velocity and position

```
V3F accel_w = attitude.Rotate_BtoI(accel);
    VectorXf trueState(QUAD_EKF_NUM_STATES);
    predictedState(0) = curState(0) + dt * curState(3);
    predictedState(1) = curState(1) + dt * curState(4);
    predictedState(2) = curState(2) + dt * curState(5);
    predictedState(3) = curState(3) + dt * accel_w.x;
    predictedState(4) = curState(4) + dt * accel_w.y;
    predictedState(5) = curState(5) + (accel_w.z - 9.81f) * dt;
    predictedState(6) = curState(6);
```

c. For scenario 09_PredictionCov calculated partial derivative of the body-to-global rotation matrix in the function GetRbgPrime()

```
RbgPrime(0, 0) = -cos(pitch) * sin(yaw);
```

```
    RbgPrime(0, 1) = -sin(roll) * sin(pitch) * sin(yaw) - cos(roll) * cos(yaw);

    RbgPrime(0, 2) = -cos(roll) * sin(pitch) * sin(yaw) + sin(roll) * cos(yaw);

    RbgPrime(1, 0) = cos(pitch) * cos(yaw);

    RbgPrime(1, 1) = sin(roll) * sin(pitch) * cos(yaw) - cos(roll) * sin(yaw);

    RbgPrime(1, 2) = cos(roll) * sin(pitch) * cos(yaw) + sin(roll) * sin(yaw);
```

d. Implemented predict() method to predict state covariance.

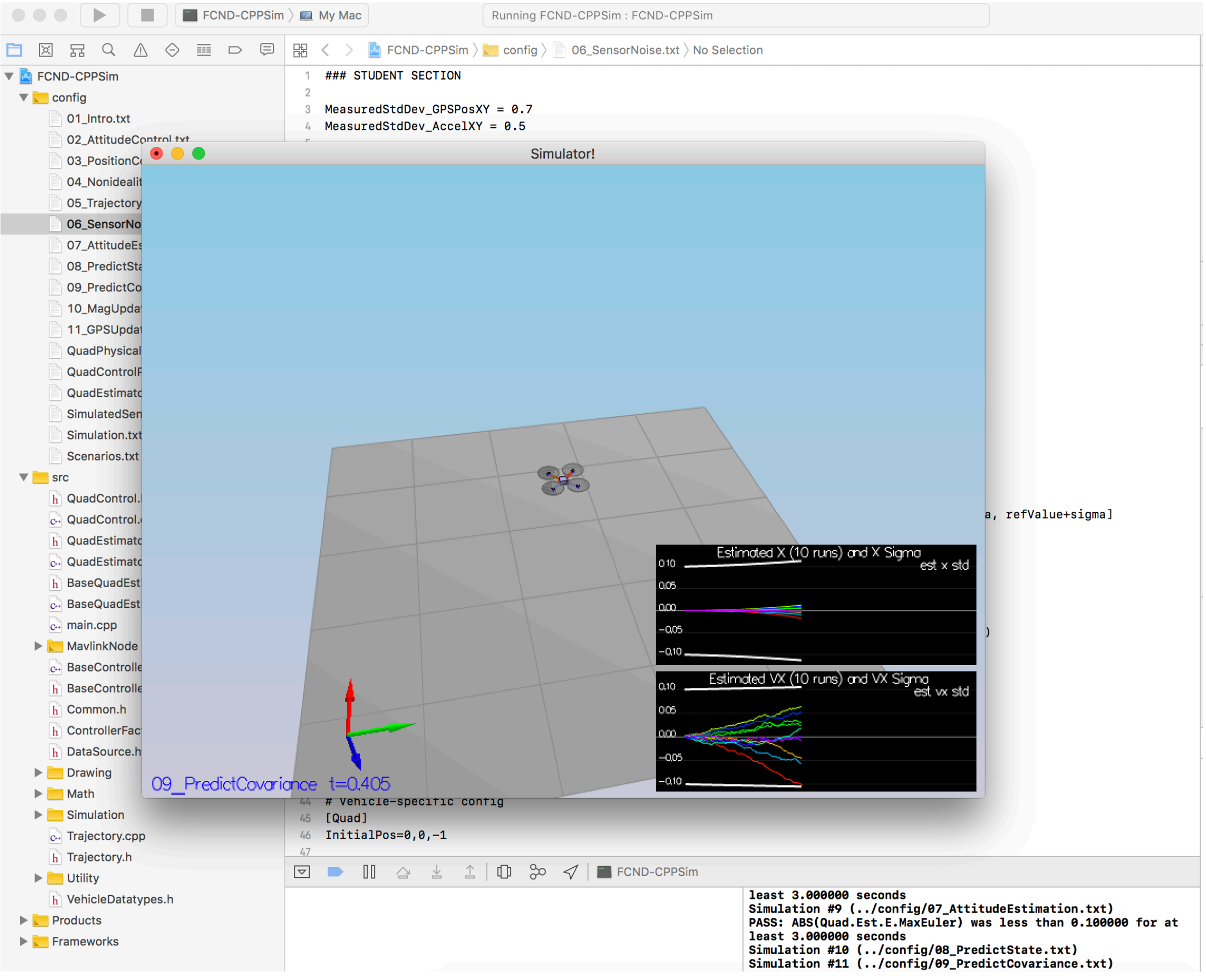
```
gPrime(0, 3) = dt;
    gPrime(1, 4) = dt;
    gPrime(2, 5) = dt;

    gPrime(3, 6) = (RbgPrime(0, 0) * accel.x + RbgPrime(0, 1) * accel.y +
RbgPrime(0, 2) * (accel.z - 9.81f)) * dt;
    gPrime(4, 6) = (RbgPrime(1, 0) * accel.x + RbgPrime(1, 1) * accel.y +
RbgPrime(1, 2) * (accel.z - 9.81f)) * dt;
    gPrime(5, 6) = (RbgPrime(2, 0) * accel.x + RbgPrime(2, 1) * accel.y +
RbgPrime(2, 2) * (accel.z - 9.81f)) * dt;

    MatrixXf gPrimeTranspose(QUAD_EKF_NUM_STATES, QUAD_EKF_NUM_STATES);
    MatrixXf sigG(QUAD_EKF_NUM_STATES, QUAD_EKF_NUM_STATES);

    gPrimeTranspose = gPrime;
    gPrimeTranspose.transposeInPlace();

    sigG = gPrime * (ekfCov * gPrimeTranspose) + Q;
    ekfCov = sigG;
```

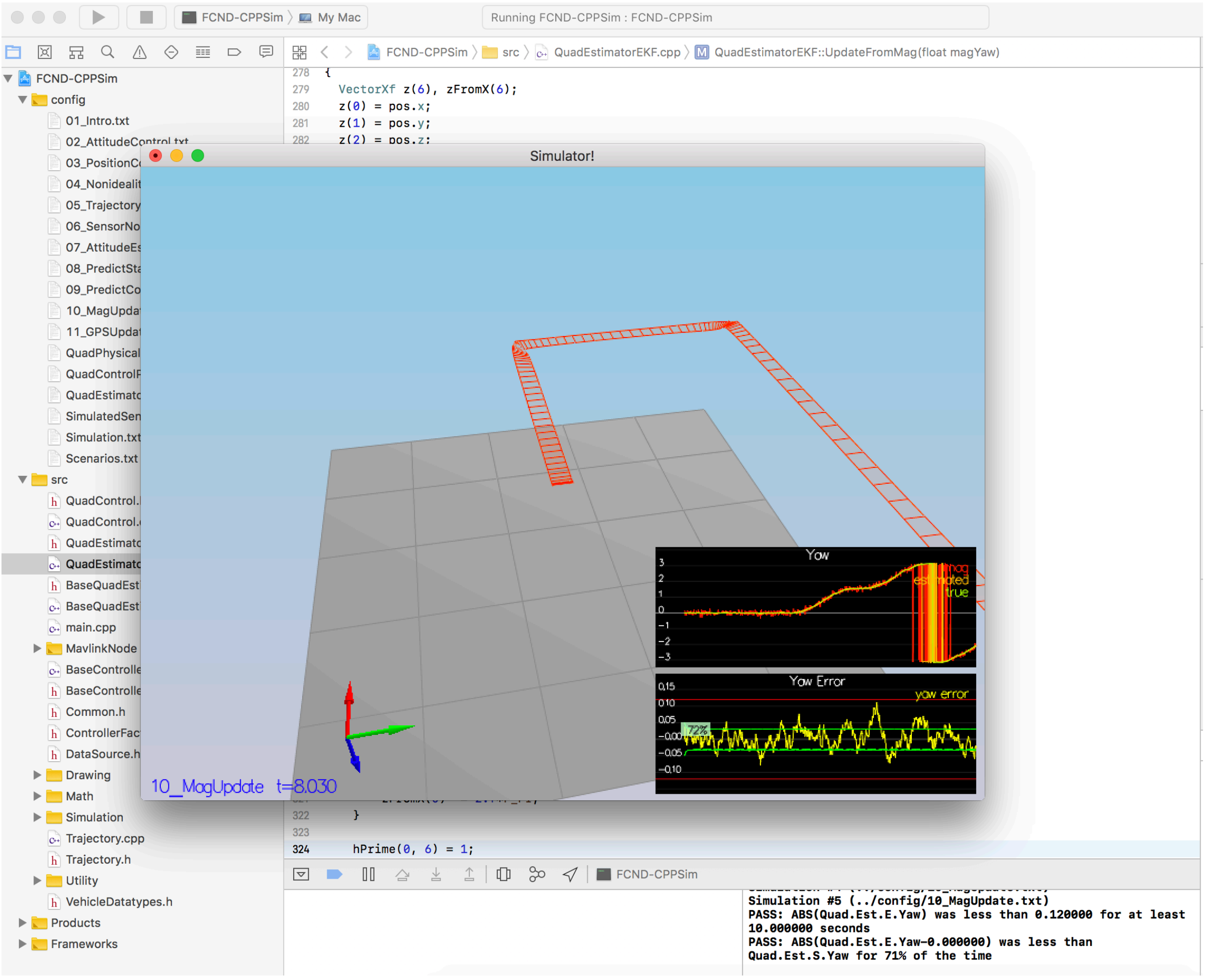


Step 4: Magnetometer update

a. Modified QYawStd = .10 (inside QuadEstimatorEKF.txt)
b. Implemented UpdateFromMag() method to normalize difference between measured and estimated yaw

```
zFromX(0) = ekfState(6);
    float diffYaw = magYaw - zFromX(0);
    if ( diffYaw > F_PI ) {
        zFromX(0) += 2.f*F_PI;
    } else if ( diffYaw < -F_PI ) {
        zFromX(0) -= 2.f*F_PI;
    }

    hPrime(0, 6) = 1;
```



Step 5: Closed Loop + GPS update

Commented below lines in config/11_GPSUpdate.txt to use realistic IMU

```
#SimIMU.AccelStd = 0,0,0
#SimIMU.GyroStd = 0,0,0
```

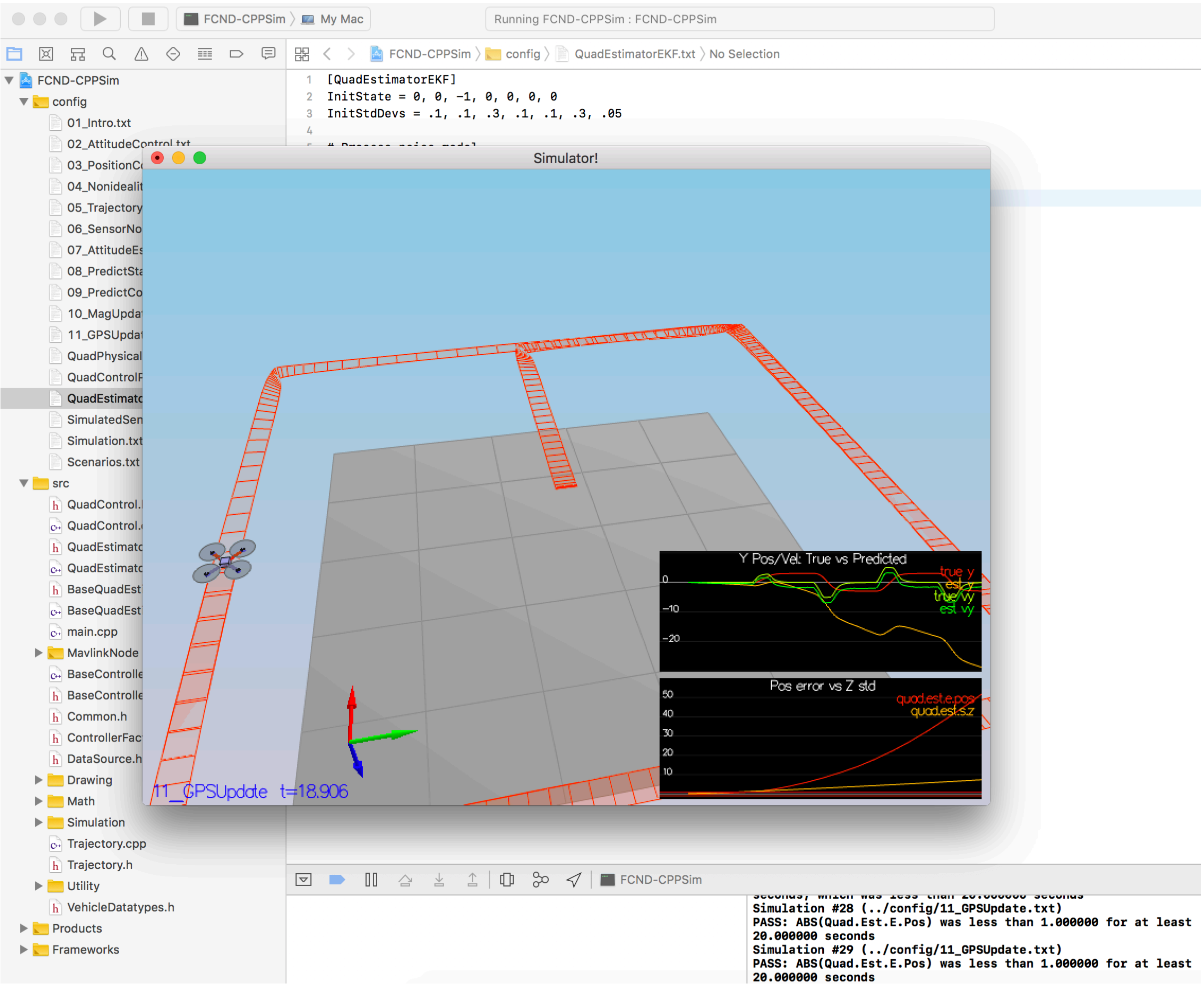
Tuned [QuadEstimatorEkf.txt]()

```
QPosXYStd = .09
QPosZStd = .05
QVelXYStd = .2
```

Implemented `UpdateFromGPS()` method inside [QuadEstimatorEKF.cpp]()

```
hPrime(0, 0) = 1;
    hPrime(1, 1) = 1;
    hPrime(2, 2) = 1;
    hPrime(3, 3) = 1;
    hPrime(4, 4) = 1;
    hPrime(5, 5) = 1;
```

```
zFromX(0) = ekfState(0);
zFromX(1) = ekfState(1);
zFromX(2) = ekfState(2);
zFromX(3) = ekfState(3);
zFromX(4) = ekfState(4);
zFromX(5) = ekfState(5);
```



Step 6: Adding your controller

a. Replaced [QuadControlParams.txt](#) with my implementation
b. Replaced [QuadControl.cpp](#)
c. Ran 11_GPSupdate

d. De-tuned my controller to

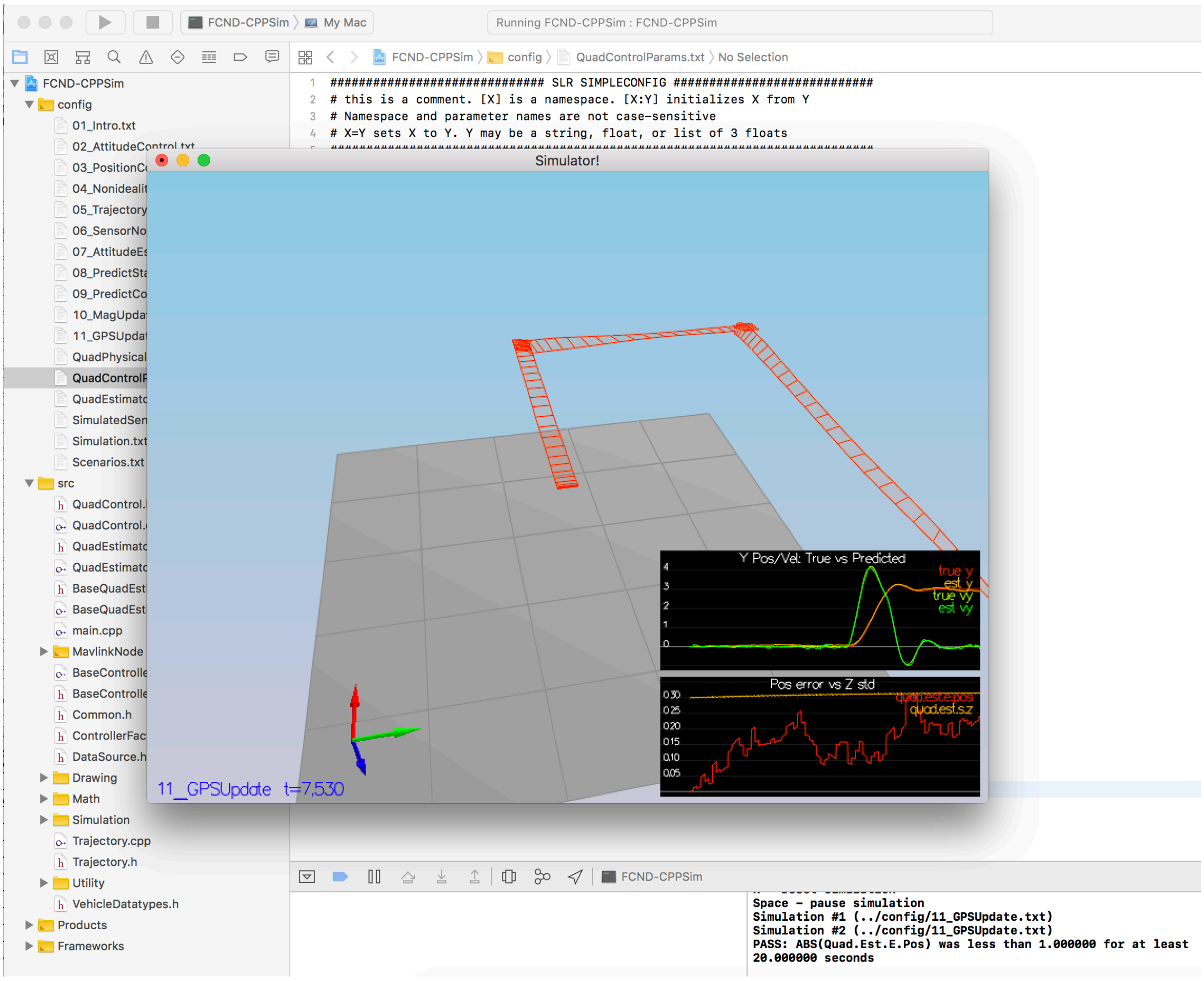
```
# Position control gains
kpPosXY = 5
kpPosZ = 26
KiPosZ = 15

# Velocity control gains
kpVelXY = 5
kpVelZ = 6

# Angle control gains
kpBank = 12
kpYaw = 1

# Angle rate gains
kpPQR = 40, 40, 4
```

e. Successfully passed the final scenario:-



Video of all scenarios [here](#).