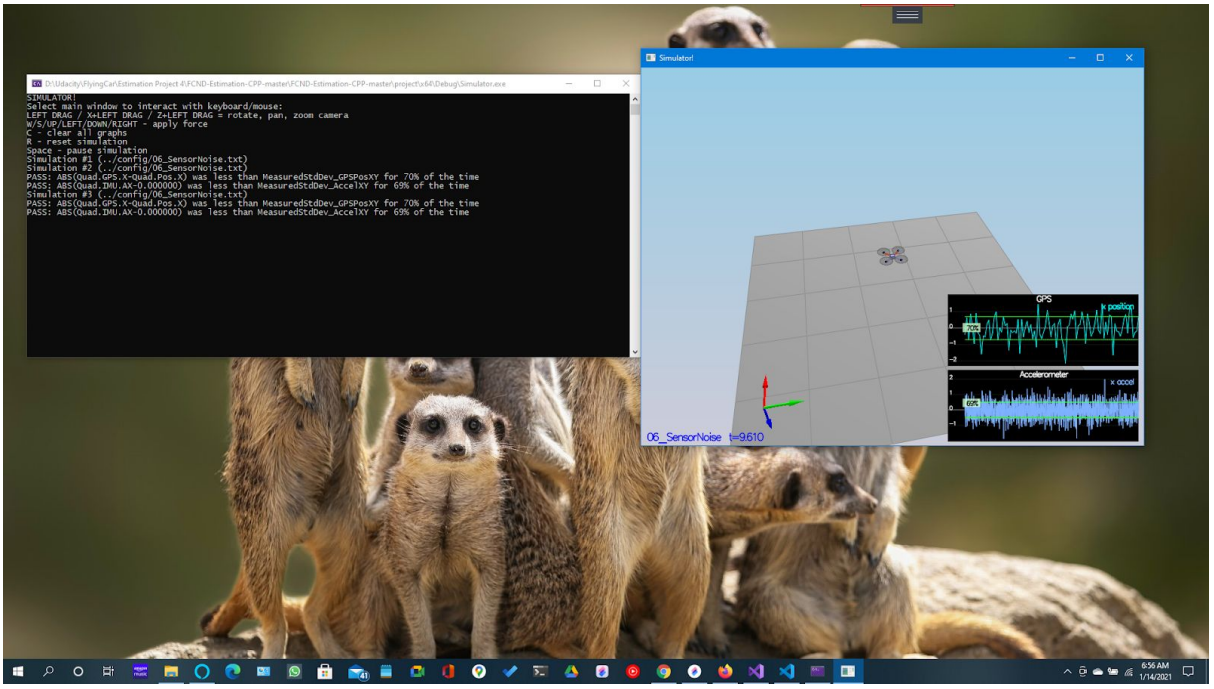
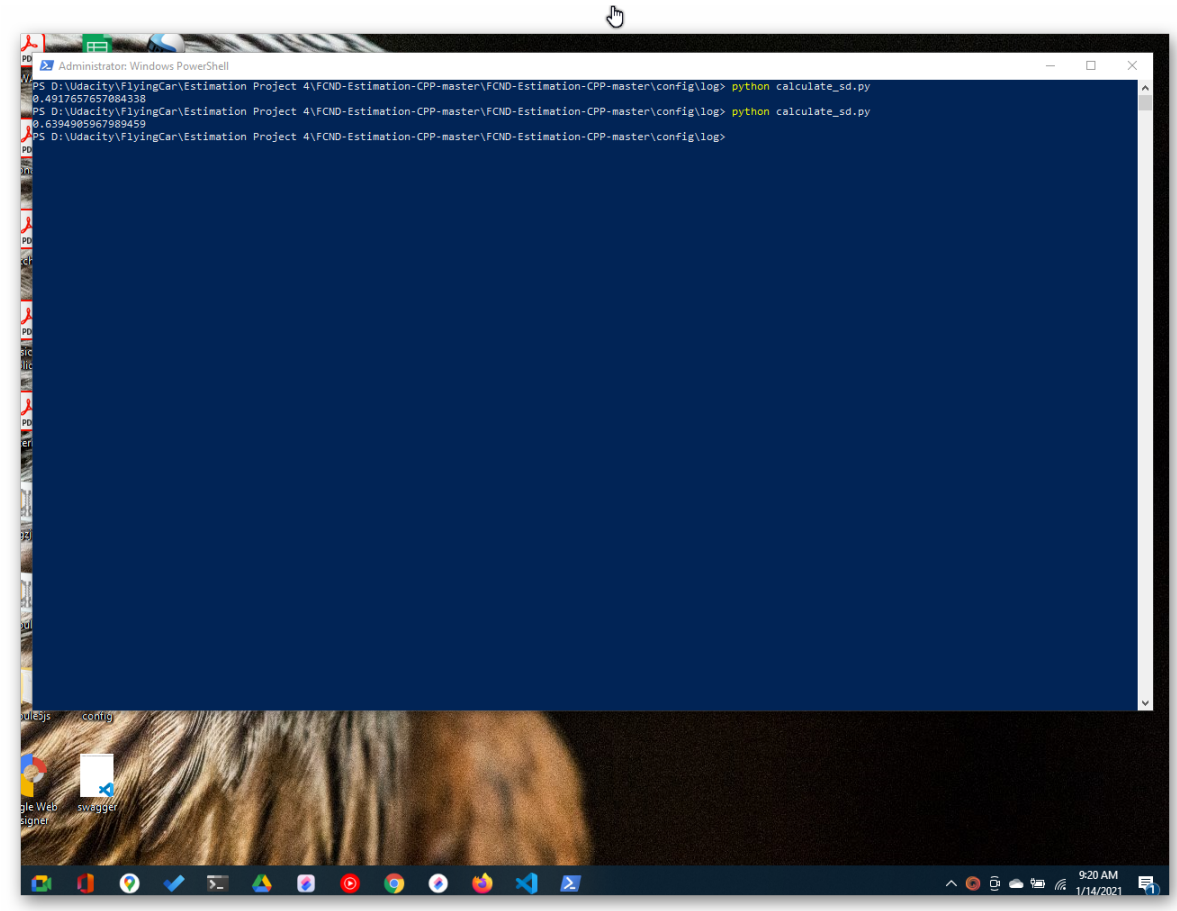
Estimation Project Readme

Implement Estimator

1. Determine the standard deviation of the measurement noise of both GPS X data and Accelerometer X data.
   Solution:
   The standard deviation of GPS (MeasuredStdDev_GPSPosXY) and Accelerometer (MeasuredStdDev_AccelXY) data were determined via recorded data and the same data is saved in Graph-logs-saved folder along with code for python program used to calculate the same in calculate_sd.py by changing the data files Graph1.txt and Graph2.txt one after another.

   MeasuredStdDev_GPSPosXY = 0.6394905967989459 (Graph1.txt)
   MeasuredStdDev_AccelXY = 0.4917657657084338 (Graph2.txt)





2. Implement a better rate gyro attitude integration scheme in the `UpdateFromIMU()` function.
   Solution:
   In the UpdateFromIMU method of QuadEstimatorEKF.cpp

```
float phi = rollEst;
  float theta = pitchEst;
  Mat3x3F rotational = Mat3x3F::Zeros();

  rotational(0, ) = 1;
  rotational(0, 1) = sin(phi) * tan(theta);
  rotational(0, 2) = cos(phi) * tan(theta);
  rotational(1, 2) = -sin(phi);
  rotational(2, 1) = sin(phi) / cos(theta);
  rotational(2, 2) = cos(phi) / cos(theta);

  V3F angle_dot = rotational * gyro;
  float predictedRoll = rollEst + dtIMU * angle_dot.x;
  float predictedPitch = pitchEst + dtIMU * angle_dot.y;
  ekfState(6) = ekfState(6) + dtIMU * angle_dot.z;

// normalize yaw to -pi .. pi
```

```
  if (ekfState(6) > F_PI) ekfState(6) -= 2.f*F_PI;
  if (ekfState(6) < -F_PI) ekfState(6) += 2.f*F_PI;

  // CALCULATE UPDATE
  accelRoll = atan2f(accel.y, accel.z);
  accelPitch = atan2f(-accel.x, 9.81f);

  // FUSE INTEGRATION AND UPDATE
  rollEst = attitudeTau / (attitudeTau + dtIMU) * (predictedRoll)+dtIMU / (attitudeTau +
dtIMU) * accelRoll;
  pitchEst = attitudeTau / (attitudeTau + dtIMU) * (predictedPitch)+dtIMU /
(attitudeTau + dtIMU) * accelPitch;

  lastGyro = gyro;
```

3. Implement all of the elements of the prediction step for the estimator.
   Solution:
   a) Predicate State:
      In the PredictState method of QuadEstimatorEKF.cpp

```
  predictedState(0) = curState(0) + dt * curState(3);
  predictedState(1) = curState(1) + dt * curState(4);
  predictedState(2) = curState(2) + dt * curState(5);

  V3F acc_w = attitude.Rotate_BtoI(accel);

  predictedState(3) = curState(3) + dt * acc_w.x;
  predictedState(4) = curState(4) + dt * acc_w.y;
  predictedState(5) = curState(5) + dt * acc_w.z - dt * CONST_GRAVITY;
```

   b) GetRgbPrime:
      In the GetRbgPrime method of QuadEstimatorEKF.cpp

```
   float cosTheta = cos(pitch);
   float sinTheta = sin(pitch);
   float cosPhi = cos(roll);
   float sinPhi = sin(roll);
   float sinPsi = sin(yaw);
   float cosPsi = cos(yaw);

   RbgPrime(0, 0) = -cosTheta * sinPsi;
   RbgPrime(0, 1) = -sinPhi * sinTheta * sinPsi - cosTheta * cosPsi;
   RbgPrime(0, 2) = -cosPhi * sinTheta * sinPsi + sinPhi * cosPsi;
   RbgPrime(1, 0) = cosTheta * cosPsi;
   RbgPrime(1, 1) = sinPhi * sinTheta * cosPsi - cosPhi * sinPsi;
   RbgPrime(1, 2) = cosPhi * sinTheta * cosPsi + sinPhi * sinPsi;
```

c) Predict:
In the Predict method of QuadEstimatorEKF.cpp

```
gPrime(0, 3) = dt;
gPrime(1, 4) = dt;
gPrime(2, 5) = dt;
gPrime(3, 6) = (RbgPrime(0) * accel).sum() * dt;
gPrime(4, 6) = (RbgPrime(1) * accel).sum() * dt;
gPrime(5, 6) = (RbgPrime(2) * accel).sum() * dt;
ekfCov = gPrime * ekfCov * gPrime.transpose() + Q;
```

4. Implement the magnetometer update.
Solution: In the UpdateFromMag method of QuadEstimatorEKF.cpp

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

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

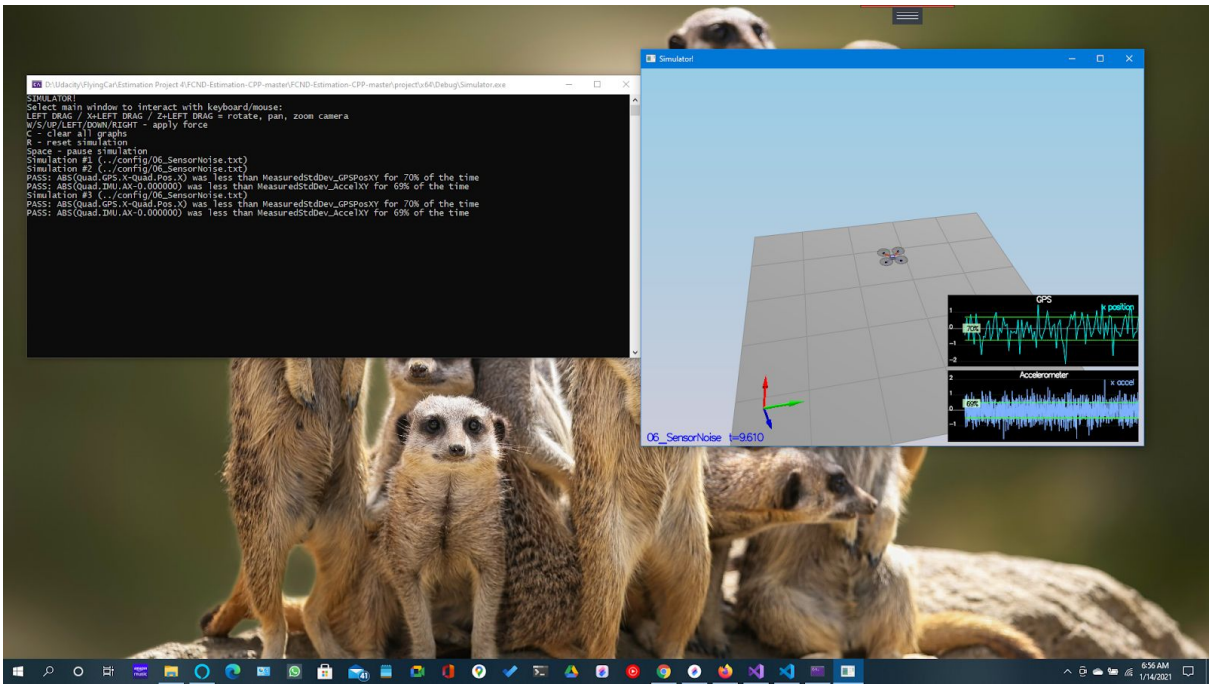
5. Implement the GPS update.
Solution: In the UpdateGPS method of QuadEstimatorEKF.cpp

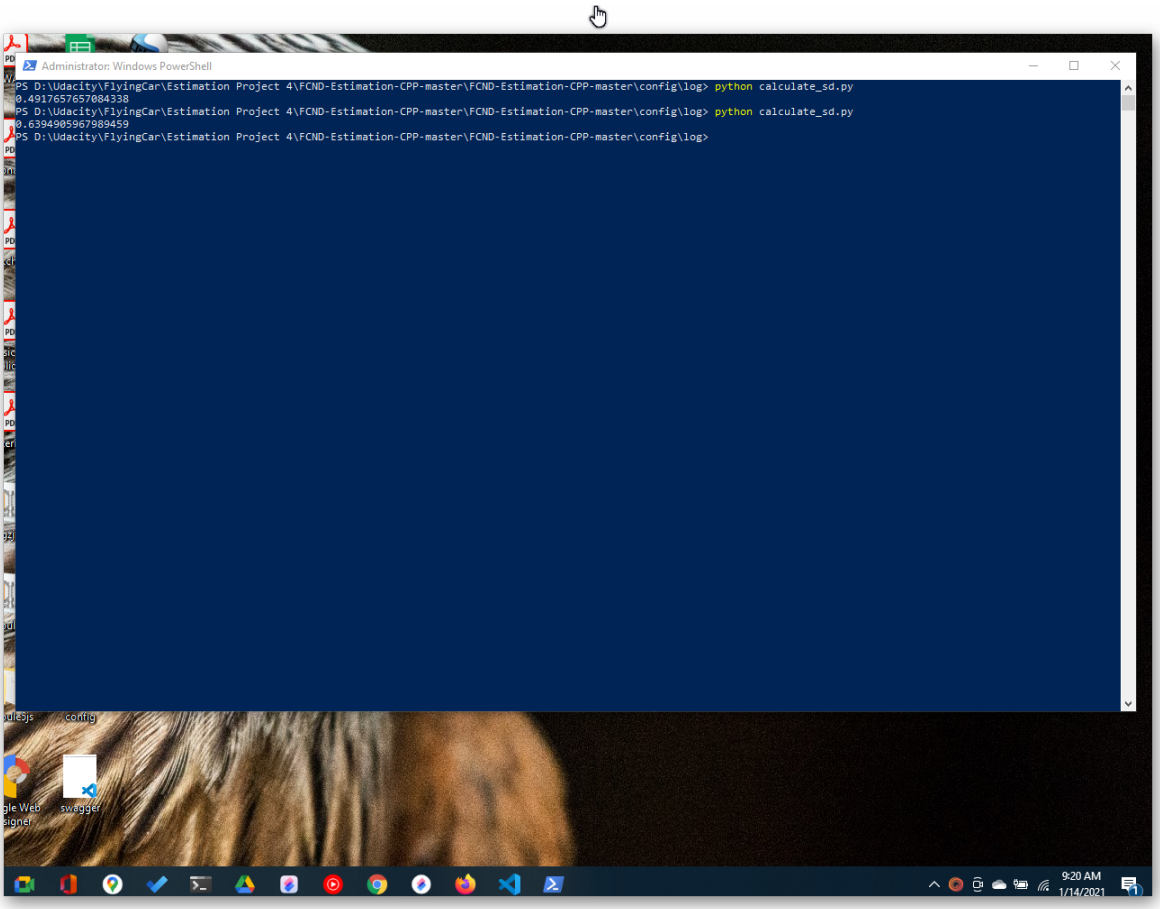
```
zFromX(0) = ekfState(0);
zFromX(1) = ekfState(1);
zFromX(2) = ekfState(2);
zFromX(3) = ekfState(3);
zFromX(4) = ekfState(4);
zFromX(5) = ekfState(5);

for (int i = 0; i < 6; i++) {
   hPrime(i, i) = 1;
}
```

Flight Evaluation

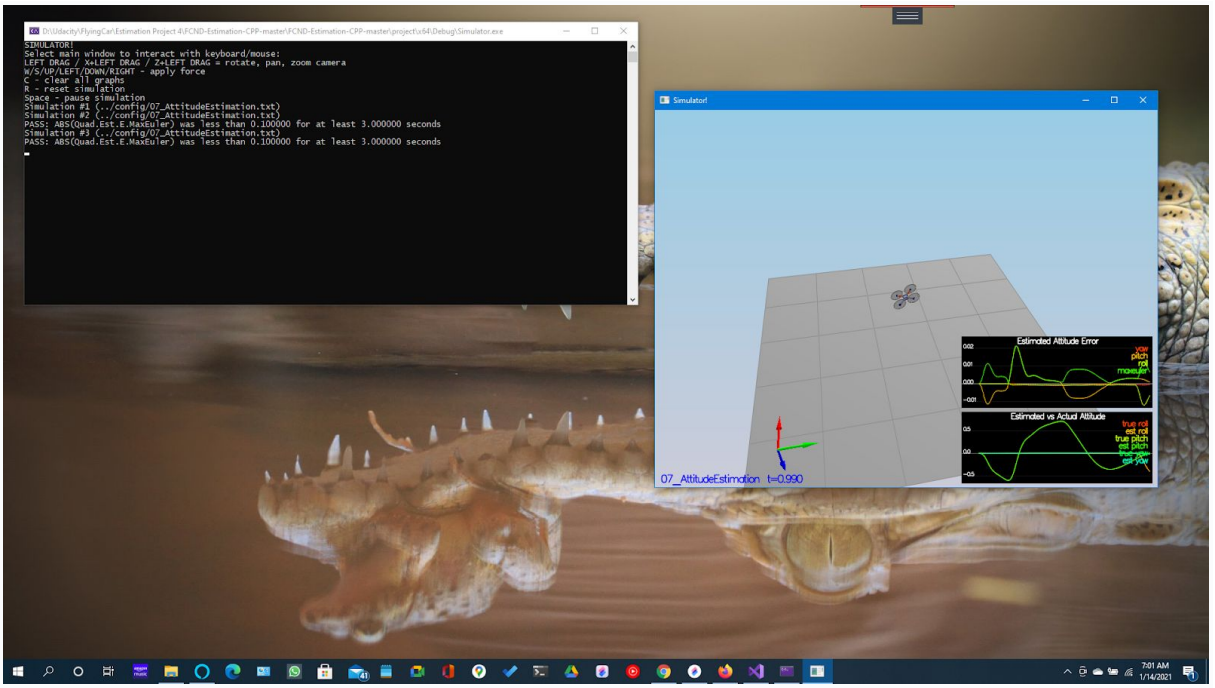
1. Meet the performance criteria of each step.
Solution: The project passes all performance criteria for all given steps.

Step 1: Sensor Noise (scenario 06_NoisySensors)

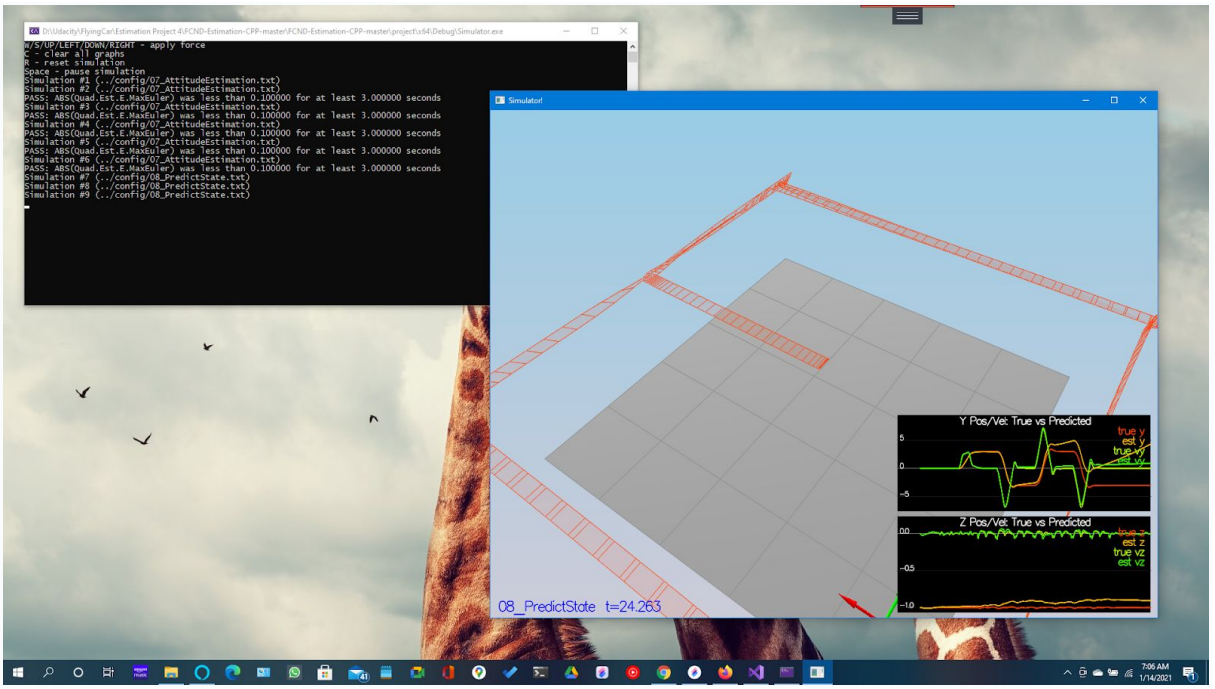




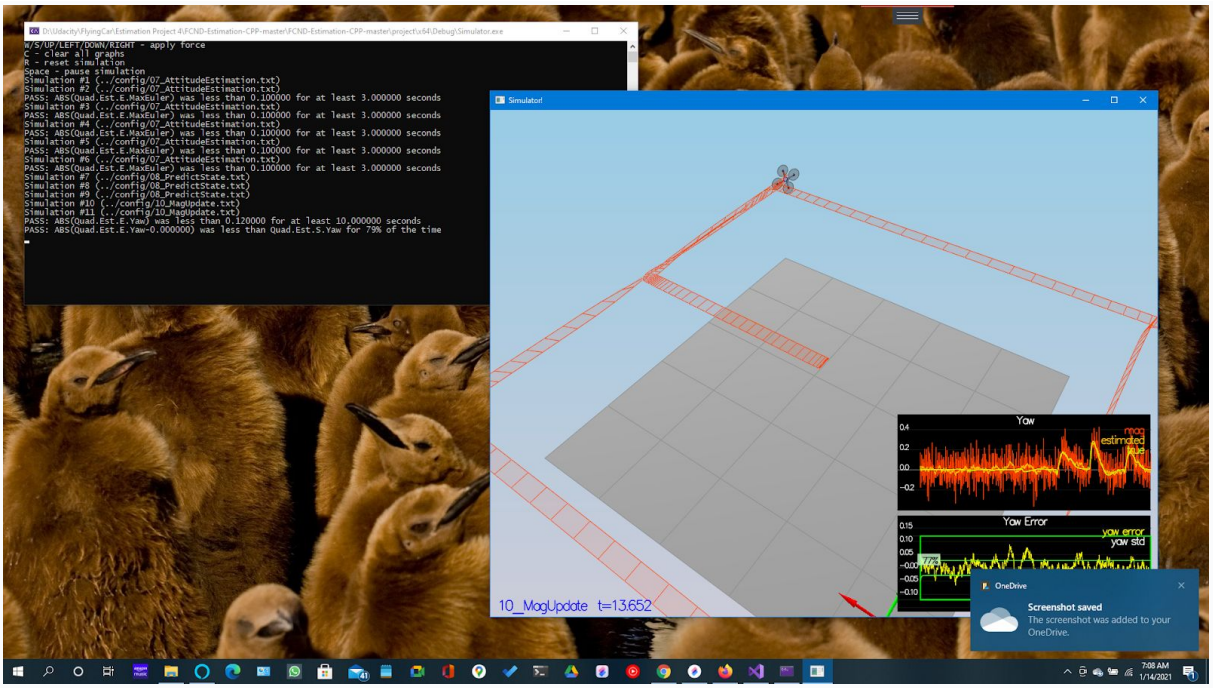
Step 2: Attitude Estimation (scenario 07_AttitudeEstimation)



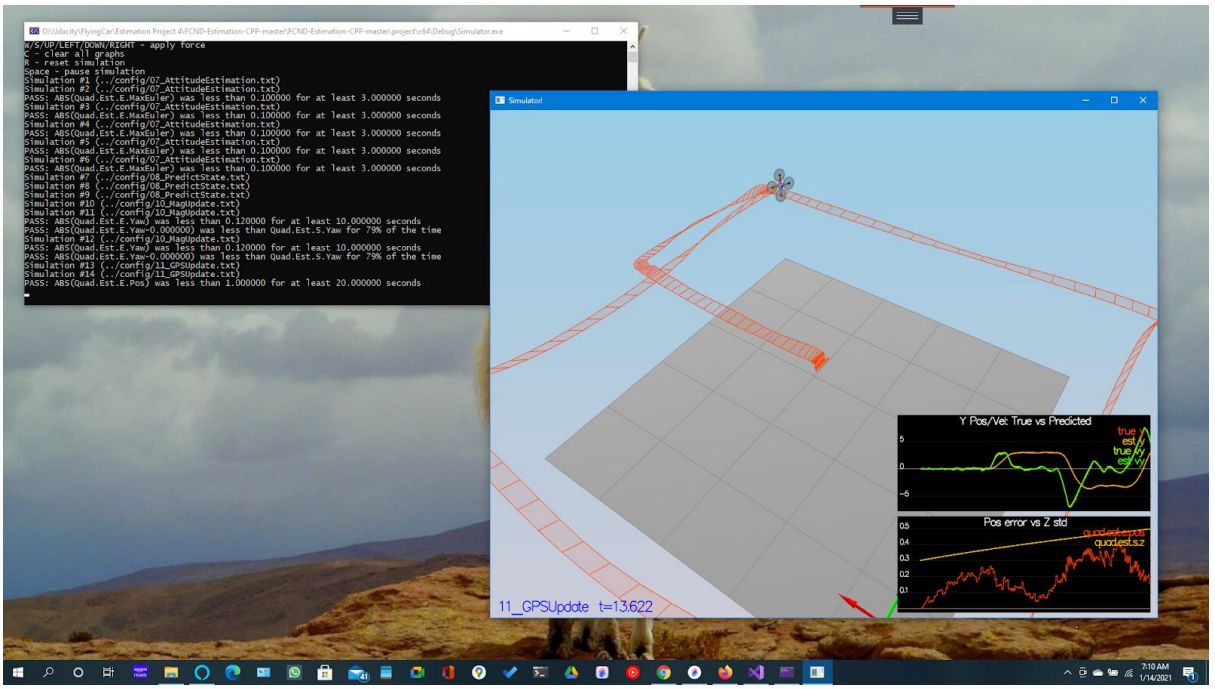
Step 3: Prediction Step (08_PredictState)



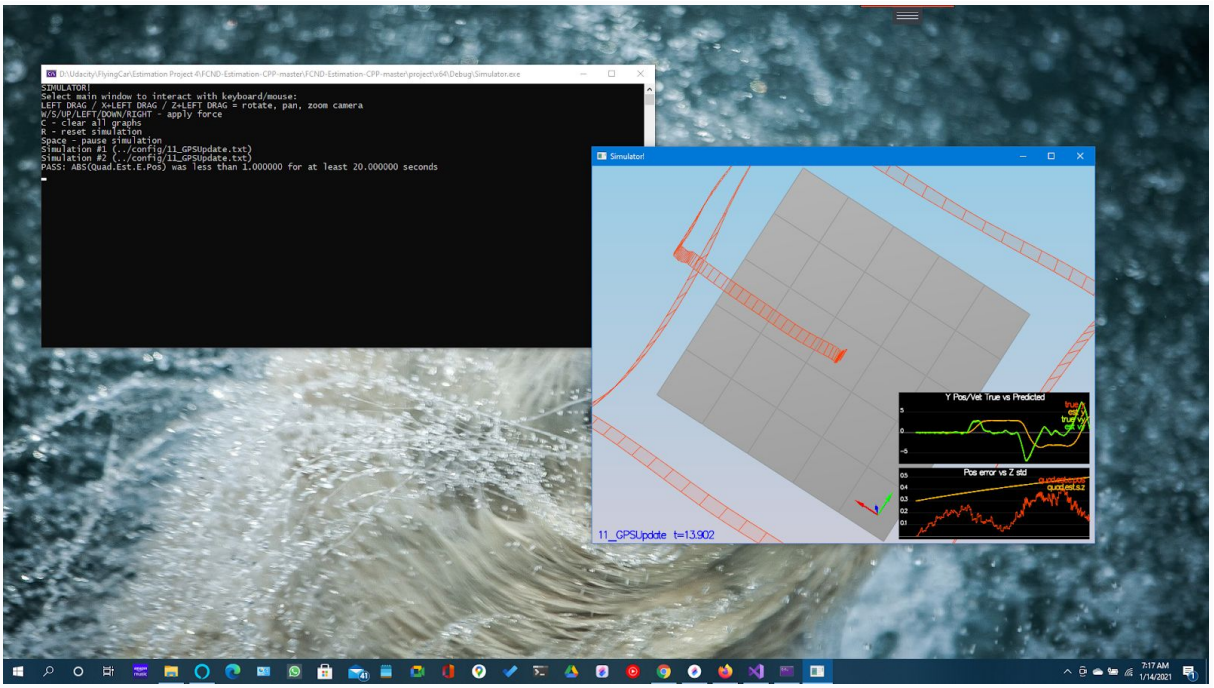
Step 4: Magnetometer Update (10_MagUpdate)



Step 5: Closed Loop + GPS Update (scenario 11_GPSUpdate)



Step 6: Adding Your Controller (scenario 11_GPSUpdate)



2. De-tune your controller to successfully fly the final desired box trajectory with your estimator and realistic sensors.
   Solution: The controller code from the controller from the control project and gain values and managed to pass the performance criteria without any major changes.