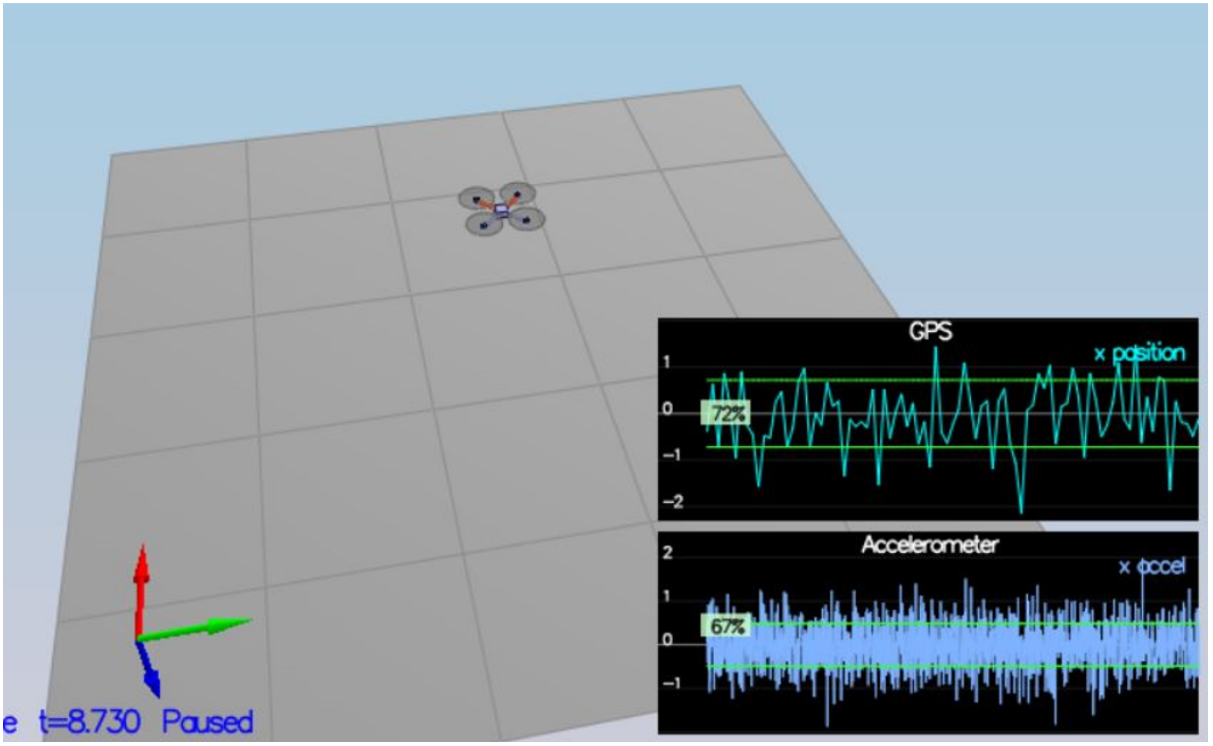
Project 4: Building an Estimator

This project aims to develop an estimator to be used by the drone controller to successfully fly a desired flight path using realistic sensors. IMU, Magnetometer and GPS measurements were used for data fusion. EKF was used for state updates.

Project Steps

Step 1- Sensor Noise: Process the logged files to figure out the standard deviation of the GPS X signal and the IMU Accelerometer X signal.

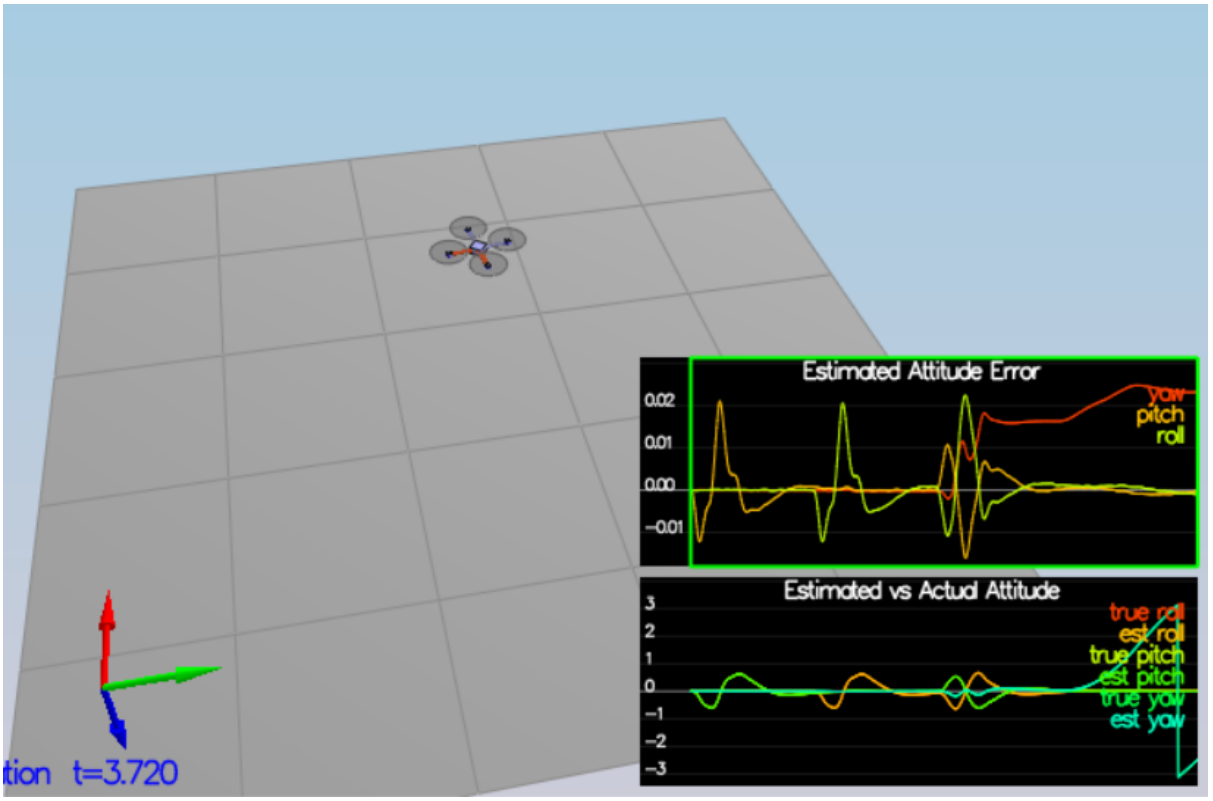
In order to get the gps and accelerometer data Scenario-6 was selected and *scenario6_calculate_sensor_std.py* code is written under the *src* folder. This code processed the recorded flight data. As a result, gps standard deviation was obtained as 0.72, accelerometer x standard deviation was obtained as 0.49. These values were updated in the 06_SensorNoise.txt. Simulation output after this update is given below.



Step 2- Attitude Estimation: In QuadEstimatorEKF.cpp, you will see the function UpdateFromIMU() contains a complementary filter-type attitude filter. To reduce the errors in the estimated attitude (Euler Angles), implement a better rate gyro attitude integration scheme. You should be able to reduce the attitude errors to get within 0.1 rad for each of the Euler angles.

A rotation matrix based on current Euler estimates was built, and multiplied with gyro measurements to obtain Euler rates. Then Euler rates integrated with Euler estimates. The rotation matrix formula and 07_AttitudeEstimation scenario result are given below.

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$

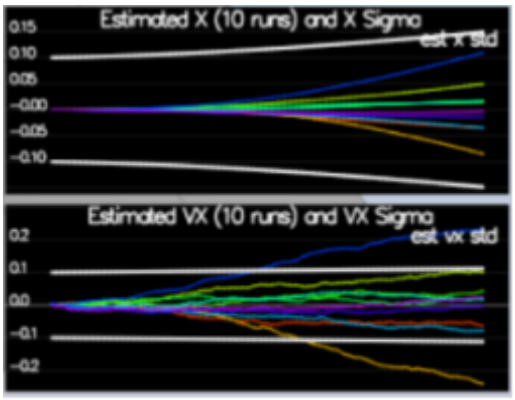


Step 3- Prediction Step: In QuadEstimatorEKF.cpp, implement the state prediction step in the PredictState() functon.Calculate the partial derivative of the body-to-global rotation matrix in the function GetRbgPrime(). Once you have that function implement, implement the rest of the prediction step (predict the state covariance forward) in Predict(). Run your covariance prediction and tune the QPosXYStd and the QVelXYStd process parameters in QuadEstimatorEKF.txt to try to capture the magnitude of the error you see

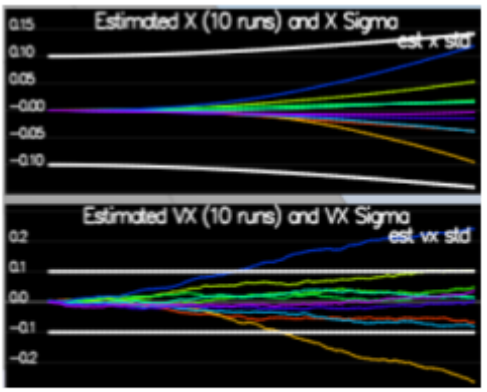
Time update was implemented in PredictState.Rbg is the rotation matrix that rotates from the body frame to the global frame.Rbg prime is the partial derivative of the Rbg rotation matrix with respect to yaw. As a reference Equation 52 in Estimation for Quadrotors was used.

$$R'_{bg} = \begin{bmatrix} -\cos\theta\sin\psi & -\sin\phi\sin\theta\sin\psi - \cos\phi\cos\psi & -cos\phi\sin\theta\sin\psi + \sin\phi\cos\psi \\ \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ 0 & 0 & 0 \end{bmatrix}$$

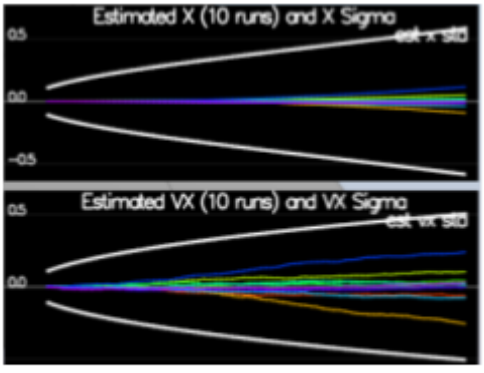
Various QPosXYStd and QVelXYSt were tried as given below.



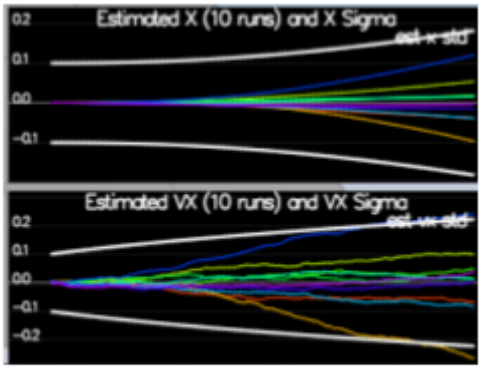
QPosXYStd = .05
QVelXYStd = .05



QPosXYStd = .01
QVelXYStd = .01



QPosXYStd = .5
QVelXYStd = .5



QPosXYStd = .005
QVelXYStd = .2

When covariance values are too high, sigma values do not reflect real estimation errors.

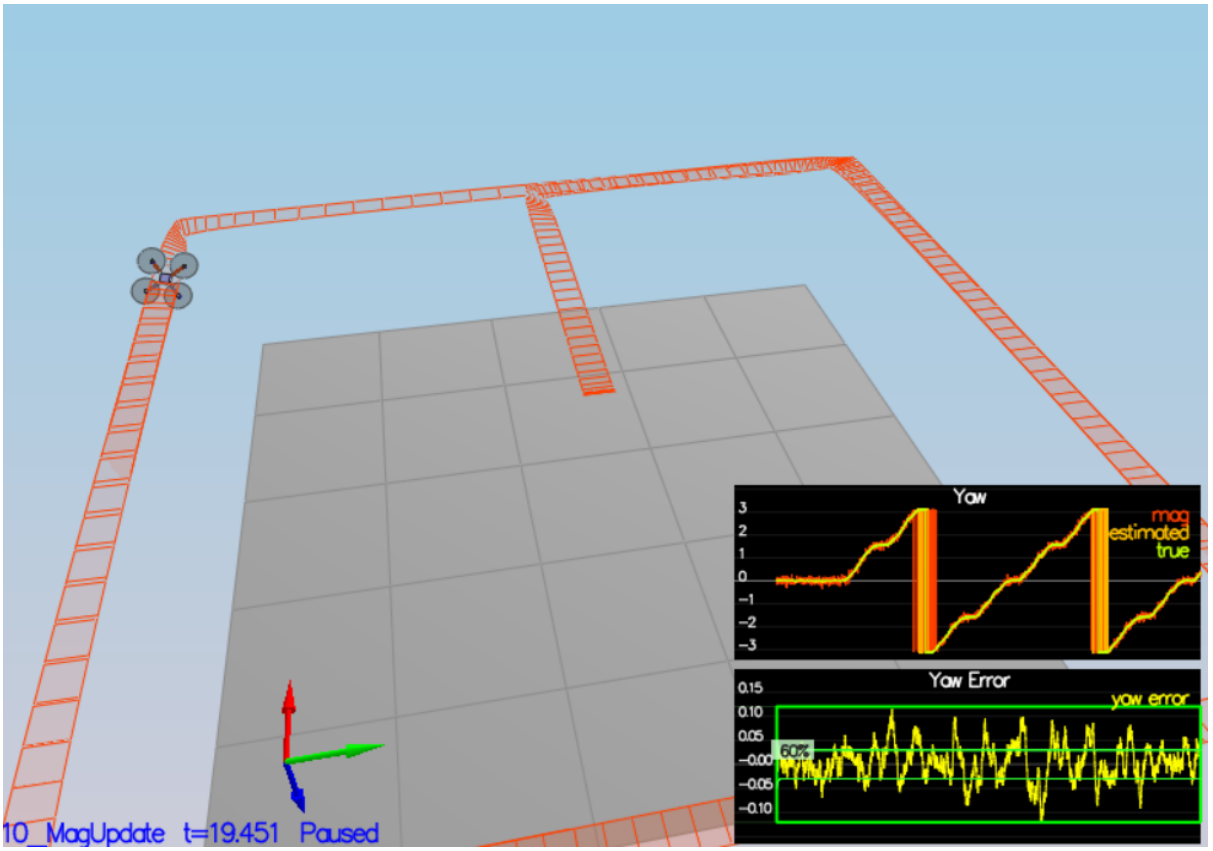
Step 4-Magnetometer Update: Tune the parameter QYawStd for the QuadEstimatorEKF. Implement magnetometer update in the function UpdateFromMag()

QYawStd was set as 0.09. Since magnetometer directly measures the yaw angle, Jacobian matrix is very straightforward as given below. From Estimation for Quadrotors equations 56-58 were used as given below.

$$z_t = \begin{bmatrix} \psi \end{bmatrix}$$

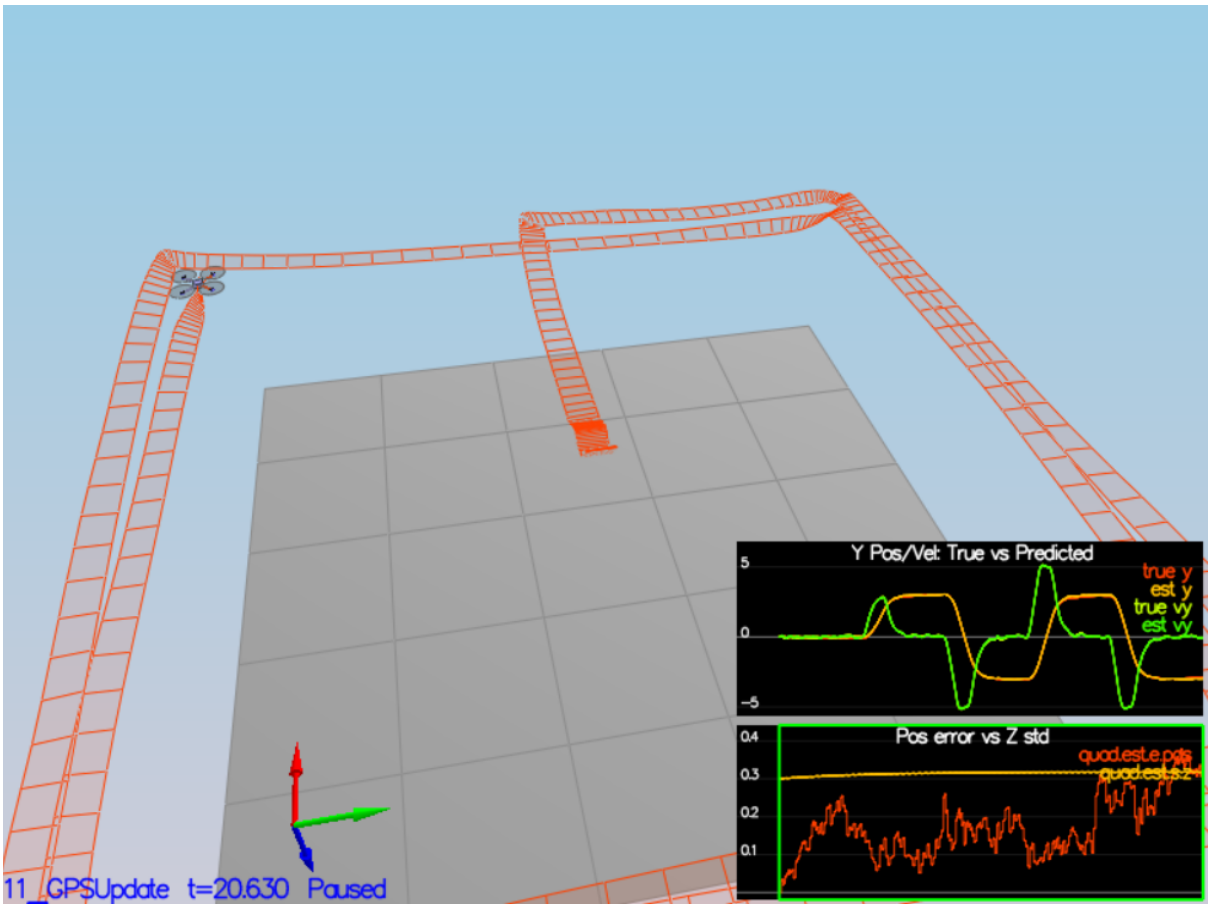
$$h(x_t) = \begin{bmatrix} x_{t,\psi} \end{bmatrix}$$

$$h'(x_t) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$



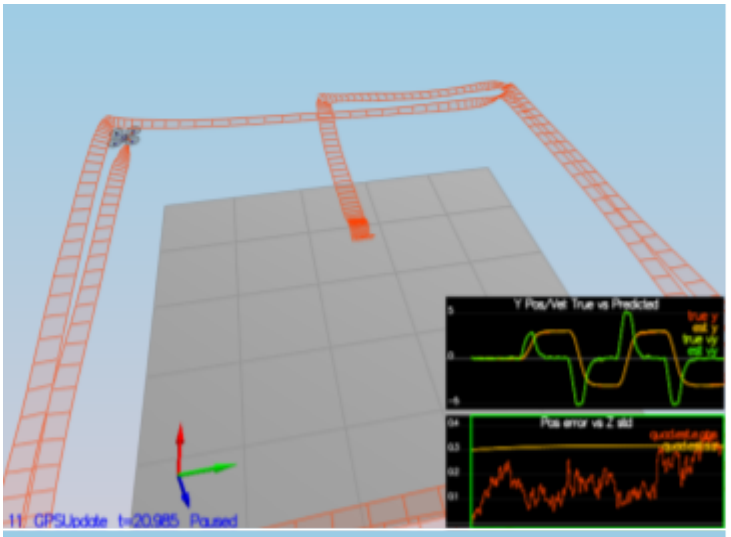
Step 5- Closed Loop + GPS Update: Let's change to using your estimator by setting Quad.UseIdealEstimator to 0. Implement the EKF GPS Update in the function UpdateFromGPS().our objective is to complete the entire simulation cycle with estimated position error of < 1m (you'll see a green box over the bottom graph if you succeed). You may want to try experimenting with the GPS update parameters to try and get better performance.

The necessary changes and UpdateFromGPS had been made. Scenario 11_GPSUpdate plots are given below. As it can be observed, drifts in position error are improved due to GPS update.

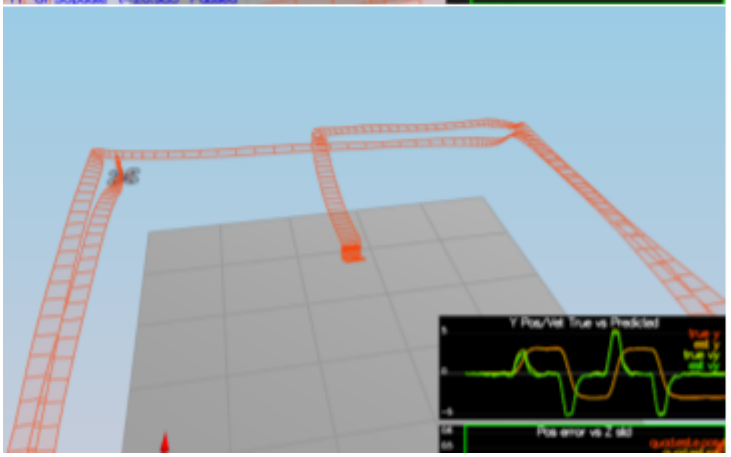


Step 6: Adding Your Controller: Replace QuadController.cpp with the controller you wrote in the last project.Replace QuadControlParams.txt with the control parameters you came up with in the last project.Run scenario 11_GPSUpdate

Simulation results and used controller coefficients are given below. The quadrotor follows the trajectory with estimated states.



Quad.UseIdealEstimator = 0
#SimIMU.AccelStd = 0,0,0
#SimIMU.GyroStd = 0,0,0



Quad.UseIdealEstimator = 0
SimIMU.AccelStd = 0,0,0
SimIMU.GyroStd = 0,0,0

Previous controller gains are provided below. Quad became unstable with these gains. Especially kpPos gains were too high.

# Position control gains
kpPosXY = 30
kpPosZ = 30
KiPosZ = 32

# Velocity control gains
kpVelXY = 12
kpVelZ = 12

# Angle control gains
kpBank = 10
kpYaw = 4

# Angle rate gains
kpPQR = 70, 70, 12

kpPos gains were reduced. Below coefficient were used to obtain a stable controller.

```
# Position control gains
kpPosXY = 2
kpPosZ = 2
KiPosZ = 50

# Velocity control gains
kpVelXY = 8
kpVelZ = 8

# Angle control gains
kpBank = 12
kpYaw = 3

# Angle rate gains
kpPQR = 70, 70, 15
```