Building a Drone Estimator in C++
and Subsequently Tuning a Controller
Udacity Flying Car NanoDegree
Mark R. Helms
July 8th, 2018

Introduction

This project uses an Extended Kalman Filter (EKF) to estimate the state of a drone in a simulator, which is placed under various scenarios in order to test the filter logic and culminates in a trajectory-following test using realistic (i.e. noisy) sensors. In order to pass each scenario, certain thresholds must be met. The scenarios include: sensor noise calibration, attitude estimation via a non-linear complementary filter, EKF state prediction, and EKF model updates for magnetometer and GPS readings. The code is written in non-STL C++. The simulator, code skeleton and starter code in some functions were provided by Udacity.

Step 1: Sensor Noise

In this part of the project, a series of noisy GPS points (x-positions) and accelerometer data (x-axis) are saved by the simulator in a comma-separated-values (CSV) file. The only thing to do is calculate the standard deviation of the given data. This can be accomplished by first calculating the mean of the data and then using it in the formula for the standard deviation as shown in the following equations:

$$\bar{x} = \frac{1}{N}\sum^{N} x_n \qquad\qquad \sigma_x = \frac{1}{N}\sum^{N} (x_n - \bar{x})^2$$

Using those equations, the mean for both GPS and accelerometer were found to be zero and the standard deviations were 0.6823 and 0.4904, respectively.

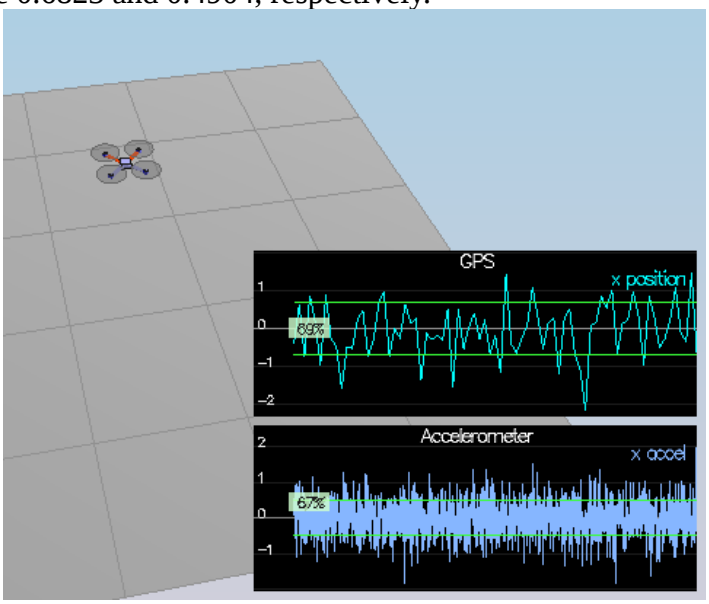

Figure 1. A successful run for scenario 06.

Step 2: Attitude Estimation

The drone uses the gyroscope body rate measurements to track quick changes in its attitude estimate (roll and pitch estimates) and accelerometer measurements to eliminate drift over time. Already given in the code is a small-angle approximation, complementary filter-type, attitude filter. This means that it is assumed that the drone is close to a hover, with the true roll and pitch approximately equal to zero, and that the gyroscope and accelerometer measurements are complementary (they can be combined for a superior result). The filters involved are the low-pass filter for the accelerometer data and the high-pass filter for the gyroscope data. The equation for a complementary filter-type attitude filter is as shown:

$$\hat{\theta} = \frac{\tau}{(\tau + T_s)}\left(\hat{\theta}_{t-1} + T_s z_{t,\dot{\theta}}\right) + \frac{T_s}{(\tau + T_s)} z_{t,\theta}$$

where theta represents either roll or pitch and z represents a measurement vector. The first half of the equation uses a measurement of the change in theta whereas the second represents a direct measurement of theta. Tau is a time constant used to tweak the filters and Ts is the interval between measurements.
In order for this filter to work, the assumption must hold that pitch and roll are approximately zero. The workaround implemented for dealing with all angles uses a non-linear conversion, whereby the prior estimates for pitch and roll are left in the world frame and used to convert the gyroscope measurements into the world frame. This results in only one rotation matrix operation being required. The rotation matrix is from the Euler Forward Method and is as shown:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \frac{\sin(\phi)}{\cos(\theta)} & \frac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

where p, q and r are body roll, pitch and yaw rates and phi, theta, and psi are the roll, pitch and yaw in world-frame.

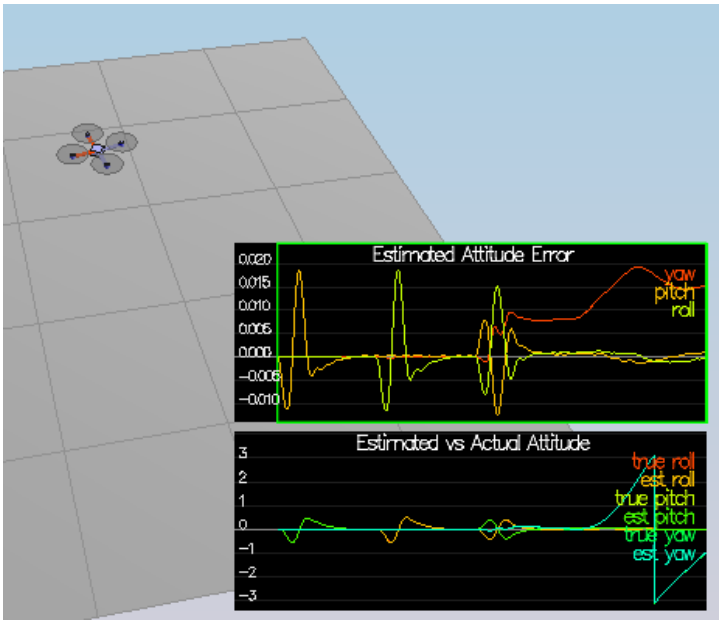

Figure 2. A successful run for scenario 07.

Step 3: Prediction Step

The prediction step for the EKF uses a state-transition matrix. This matrix relates the next state after a specific time interval to the current state and a controls input. The state of the model is given by:

$$x_t = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \psi \end{bmatrix}^T$$

where x, y and z are global x, y and z coordinates and psi is the heading, relative to north. The controls input of the model is given by:

$$u_t = \begin{bmatrix} \ddot{x}^b & \ddot{y}^b & \ddot{z}^b & \dot{\psi} \end{bmatrix}^T$$

which contains accelerations relative to the body frame in the x, y and z directions and a yaw rate.

In order to progress the state, the state velocities are added to the state positions. Then, the commanded body-frame accelerations are converted to world-frame accelerations and added to the velocities. Finally, the commanded change in yaw rate is added to the state yaw rate. The rotation is implemented with the rotation matrix Rbg and the overall state transition is implemented with the equation shown below. The new state is equal to g(old state, controls, change in time).

$$R_{bg} = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}$$

$$g(x_t, u_t, \Delta t) = \begin{bmatrix} x_{t,x} + x_{t,\dot{x}}\Delta t \\ x_{t,y} + x_{t,\dot{y}}\Delta t \\ x_{t,z} + x_{t,\dot{z}}\Delta t \\ x_{t,\dot{x}} \\ x_{t,\dot{y}} \\ x_{t,\dot{z}} - g\Delta t \\ x_{t,\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ R_{bg}[0:] & & & 0 \\ R_{bg}[1:] & & & 0 \\ R_{bg}[2:] & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} u_t \Delta t$$

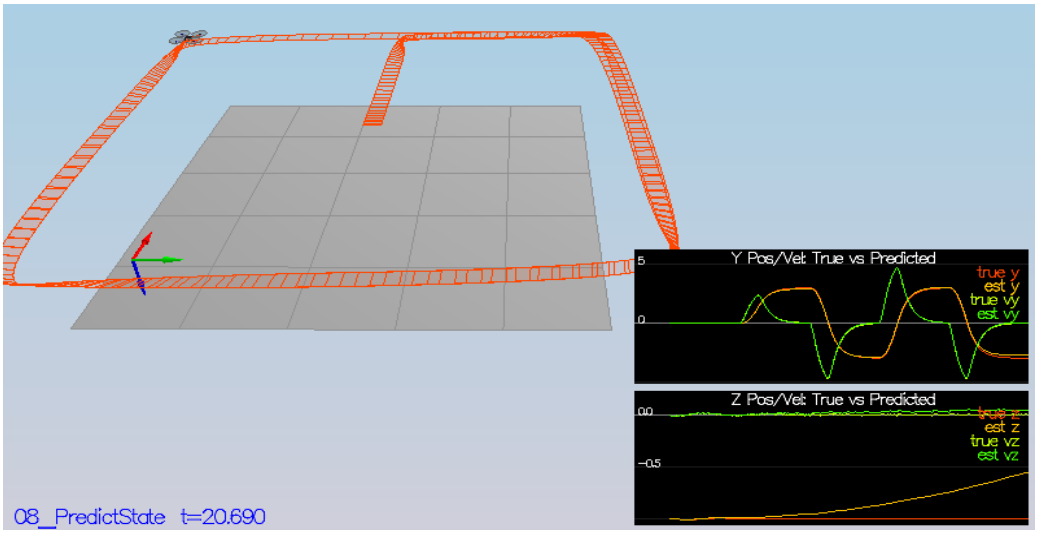

Figure 3. A successful run for scenario 08.

The new covariance must still be calculated. In order to do this, since we are using an EKF, we have to calculate the Jacobian of the g(x,u,t) function. This is given in the class notes as:

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

$$g'(x_t, u_t, \Delta t) = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & R'_{bg}[0:]u_t[0:3]\Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 & R'_{bg}[1:]u_t[0:3]\Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 & R'_{bg}[2:]u_t[0:3]\Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Given these functions, we can calculate the new covariance matrix by in effect performing the transition-type function for the previous noise estimate and adding in the noise estimate of our model noise. This can be achieved with the following equation:

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + Q_t$$

where sigma is the covariance estimate, Gt is the function g'(x,u,dt) and Qt is the additive model covariance noise.

The final part of this step of the project is to calibrate the process noise levels using a special arrangement, whereby ten drones are flown simultaneously, each with different errors. The goal is to choose Qposx,y and Qvelx,y standard deviations in order to contain the visible errors. The determined levels were 0.01 and 0.22, respectively. Note that most of the error in the process model stems from the rotation matrix which depends on accurate attitude knowledge.

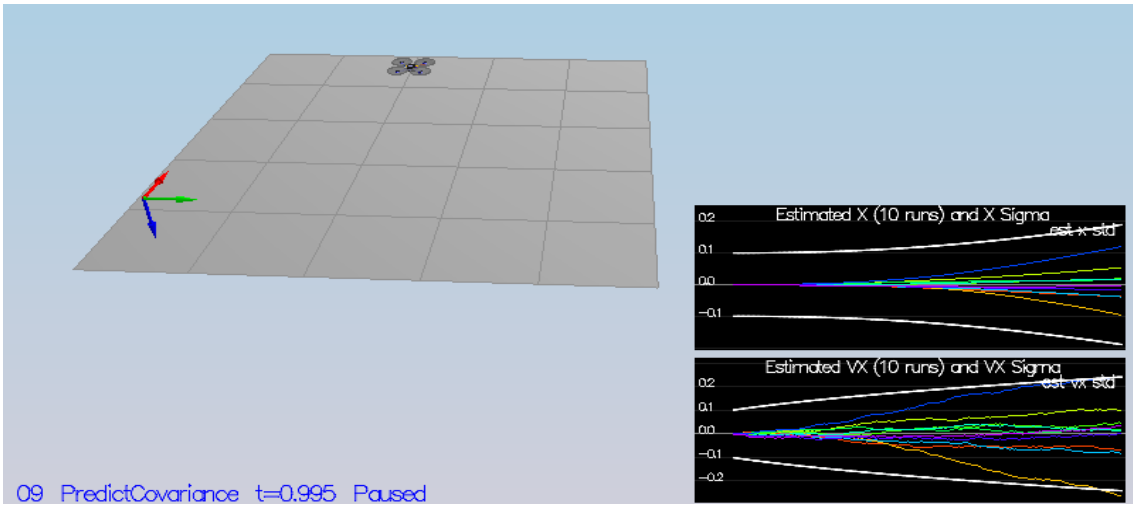

Figure 4. A successful run for scenario 09.

Step 4 and 5: Magnetometer Update and GPS Position Update

The update step for the EKF is broken into two parts in this project. One for the world-frame position and velocity and one for the yaw (heading). Measurements for all of these variables are directly available from the GPS and magnetometer. Because these variables match our state variables, there is absolutely no conversion necessary. The measurement models and partial derivatives are expressed as:

GPS

$$z_t = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T$$

$$h(x_t) = \begin{bmatrix} x_{t,x} & x_{t,y} & x_{t,z} & x_{t,\dot{x}} & x_{t,\dot{y}} & x_{t,\dot{z}} \end{bmatrix}^T$$

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

Magnetometer

$$z_t = [\psi]$$

$$h(x_t) = [x_{t,\psi}]$$

$$h'(x_t) = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]$$

As can be seen, position and velocity are updated directly by the GPS and yaw is updated directly by the magnetometer. Care must be taken to make sure that the magnetometer reading is in the same 2*pi range as the yaw estimate in order to avoid going the long way around the circle to correct for an error.

These matrices and vectors are passed along to EKF code implemented by Udacity which performs the Update step of the EKF. This involves calculating the Kalman gain and then adding the gained version of the error to the most recent estimate, along with updating the covariance estimate, reducing it by a factor approximately equal to one (or the identity matrix) minus the Kalman gain.

$$H_t = h'(\bar{\mu}_t)$$

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + R_t)^{-1}$$

$$\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$$

$$\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$$

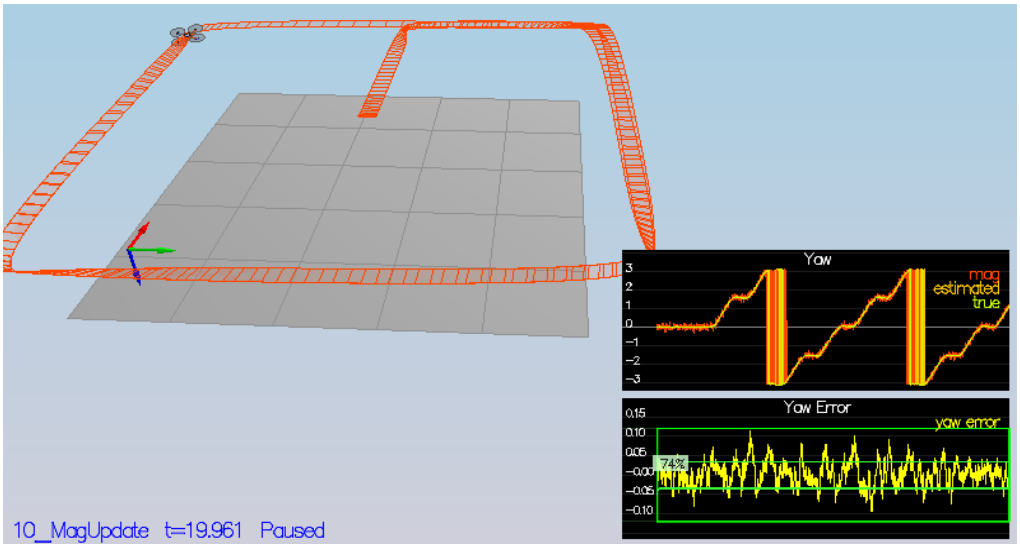

Figure 5. A successful run for scenario 10.

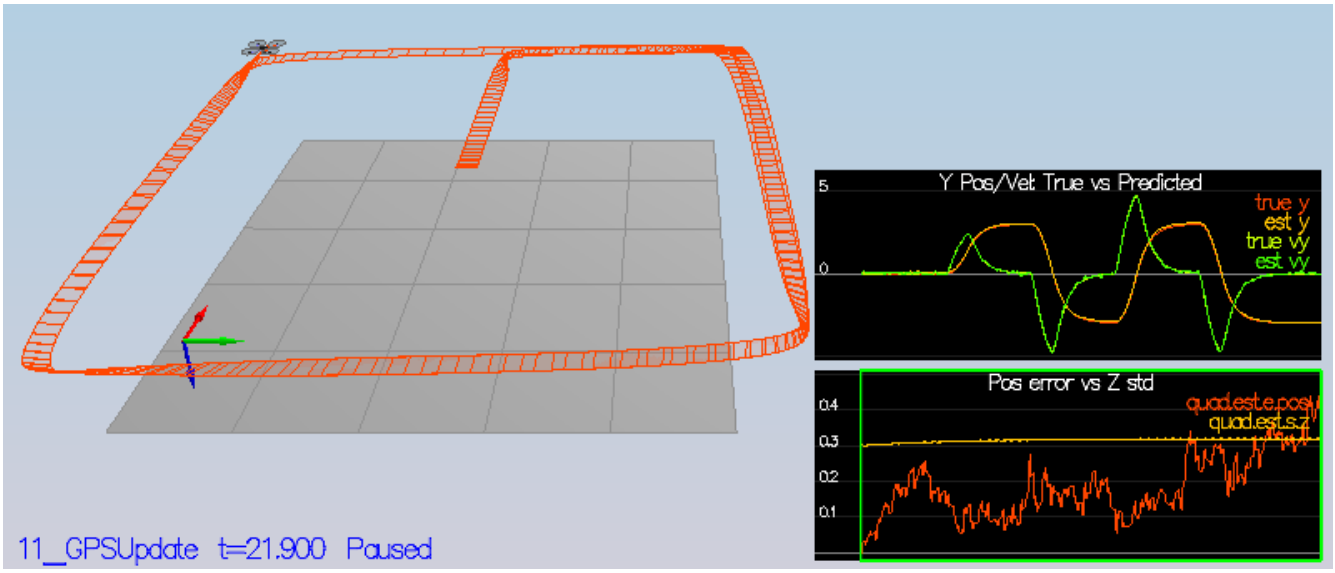

Figure 6. A successful run for scenario 11 – with ideal estimator and ideal sensors.

Two scenarios were run. One tested the magnetometer updates and the other tested the GPS positioning updates. The output from those runs are shown as Figures 5 and 6.

Step 6: Integrating the Controller from the Previous Project

The final step in this project is copying in the quad controller code from the previous project and retuning the proportional-integral-derivative (PID) controller parameters to allow for successful flight. Generally, this involved "loosening" the tuning, by making the correction strengths milder. Otherwise, the error introduced in the actual positions and attitude estimates may cause the drone to oscillate along many axes at once! The parameters which needed to be modified the most were the yaw parameters and the parameters for x,y position (and velocity).

Suitable PID controller factors found were as follows:

Variable	kp	ki	kd
position x / y	8	-	6
altitude z	10	50	20
world pitch / roll	17	-	-
world yaw	5	-	-
body pitch / roll	65		
body yaw	8		

Scenario 11, which tests the GPS update was successfully run, however the controls were a bit sloppy. This is likely due to the effects of noise and performs indicates that further tuning of the PID controllers could produce cleaner results. One other item of note was that it appeared that the Z position would continue to drift very slightly, however this was also the case with the ideal estimator and sensors.

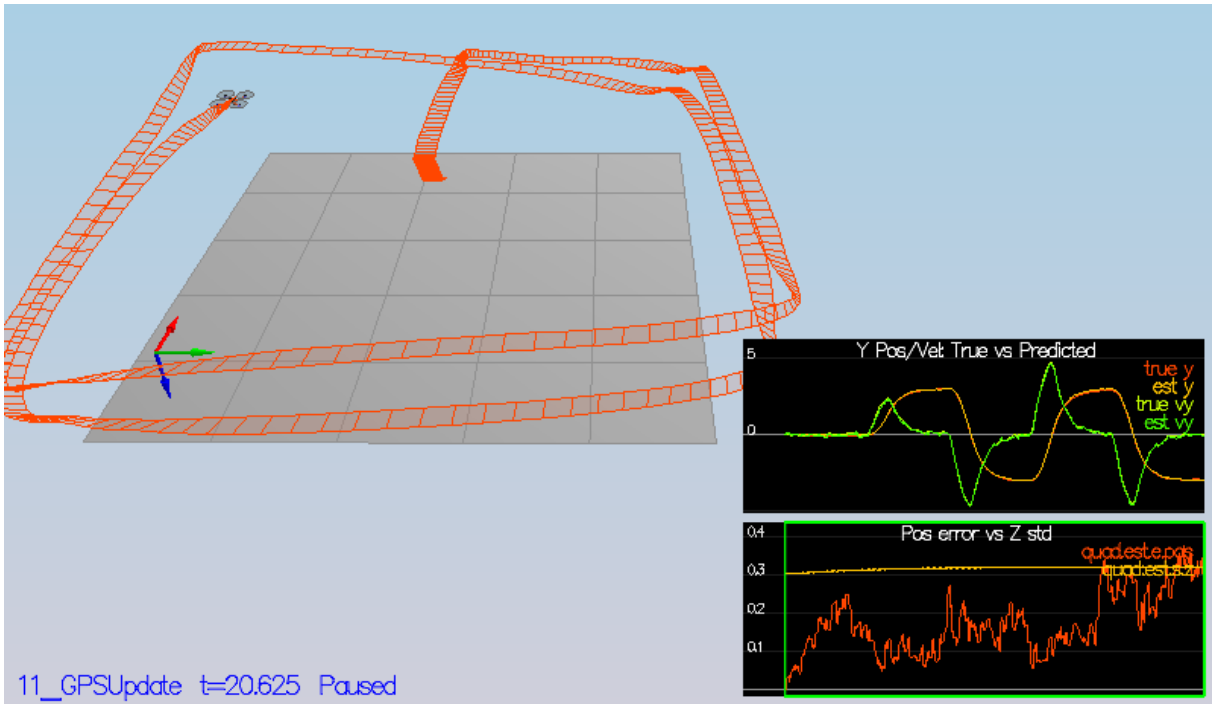

Figure 7. A successful run for scenario 11 – with the EKF estimator and noisy sensors.

Code Sections

Job	Code Function (QuadEstimatorEKF.cpp)
Non-linear attitude estimate update - advancement of yaw in the state	UpdateFromIMU
EKF State Prediction - body rate conversion to world frame - advancement of position and velocity in state	PredictState
Generation of Jacobian for position/velocity state update, Rbg.	GetRbgPrime
EKF Prediction of state (indirectly) EKF Prediction of covariance matrix.	Predict
EKF Update using GPS measurements for position and velocity. (preparation only)	UpdateFromGPS
EKF Update using magnetometer reading for yaw. (preparation only)	UpdateFromMag
EKF Update (written by Udacity)	Update