Project Estimation

Project Steps

First cloned the repository and gotten familiar with the C++ environment as outlined in C++ Setup.

Project Tasks:

Completed the following scenarios:
Step 1: Sensor Noise
Step 2: Attitude Estimation
Step 3: Prediction Step
Step 4: Magnetometer Update
Step 5: Closed Loop + GPS Update
Step 6: Adding Your Controller
And most of the missing parts of the EKF implemented and tested.

1: Sensor noise

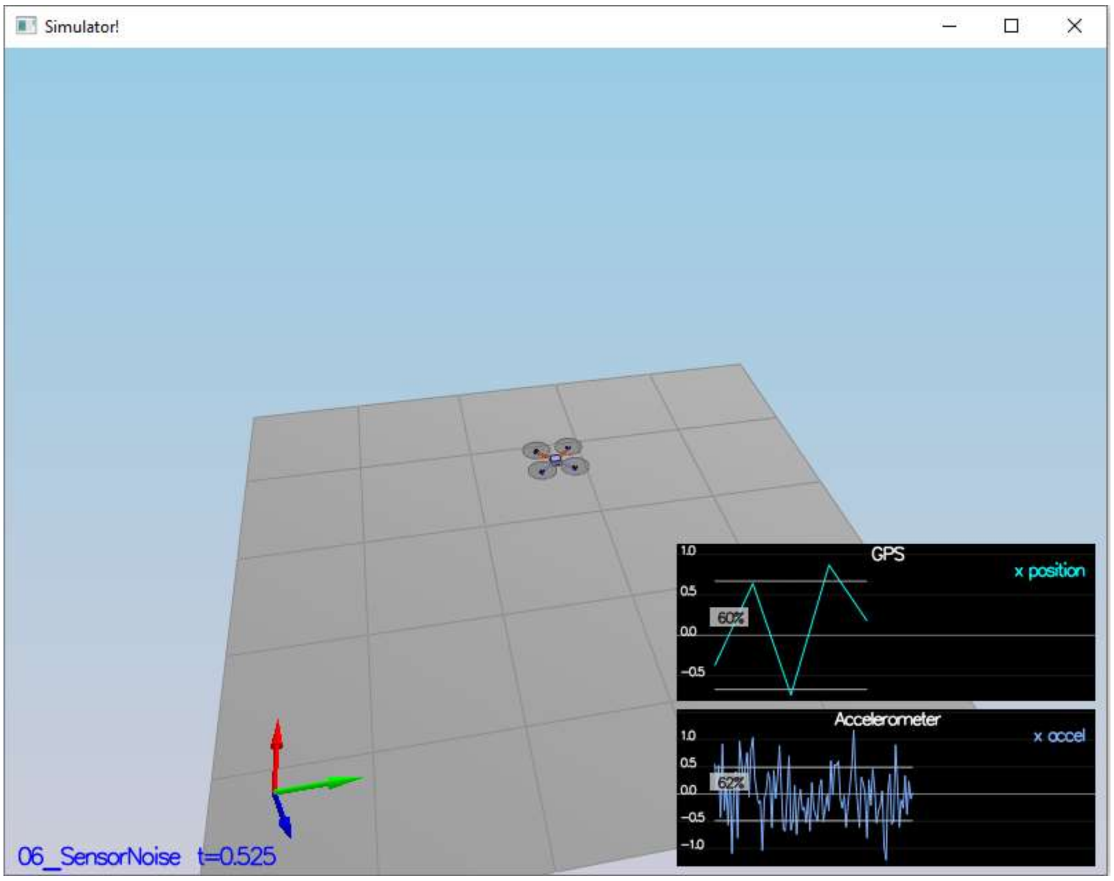
Task: *Determine the standard deviation of the measurement noise of both GPS X data and Accelerometer X data.*

In this scenario 06_SensorNoise. The simulator will generate two files with IMU and GPS measurements. The collected data were saved in /src/data/Accelerometer.X.Data.txt and /src/data/GPS.X.Data.txt. The task is to process those files and calculate the standard deviation(sigma) for those sensors.

Set the values for MeasuredStdDev_GPSPosXY and MeasuredStdDev_AccelXY. Top of config/6_Sensornoise.txt.

While running the simulator standard deviation correctly capturing approx 68% of the respective measurements which is within +/- 1 sigma bound for a Gaussian noise model.

```
Simulation #189 (../config/06_SensorNoise.txt)
PASS: ABS(Quad.GPS.X-Quad.Pos.X) was less than
MeasuredStdDev_GPSPosXY for 68% of the time
PASS: ABS(Quad.IMU.AX-0.000000) was less than
MeasuredStdDev_AccelXY for 68% of the time
```



2: Attitude Estimation
First of all, include information from IMU to the state estimation.

In this scenario 07_AttitudeEstimation. Sensor used is the IMU and noise levels are set to 0 in config/07_AttitudeEstimation.txt

The top graph is showing errors in each of the estimated Euler angles. The bottom shows the true Euler angles and the estimates. Observe that there's quite a bit of error in attitude estimation.

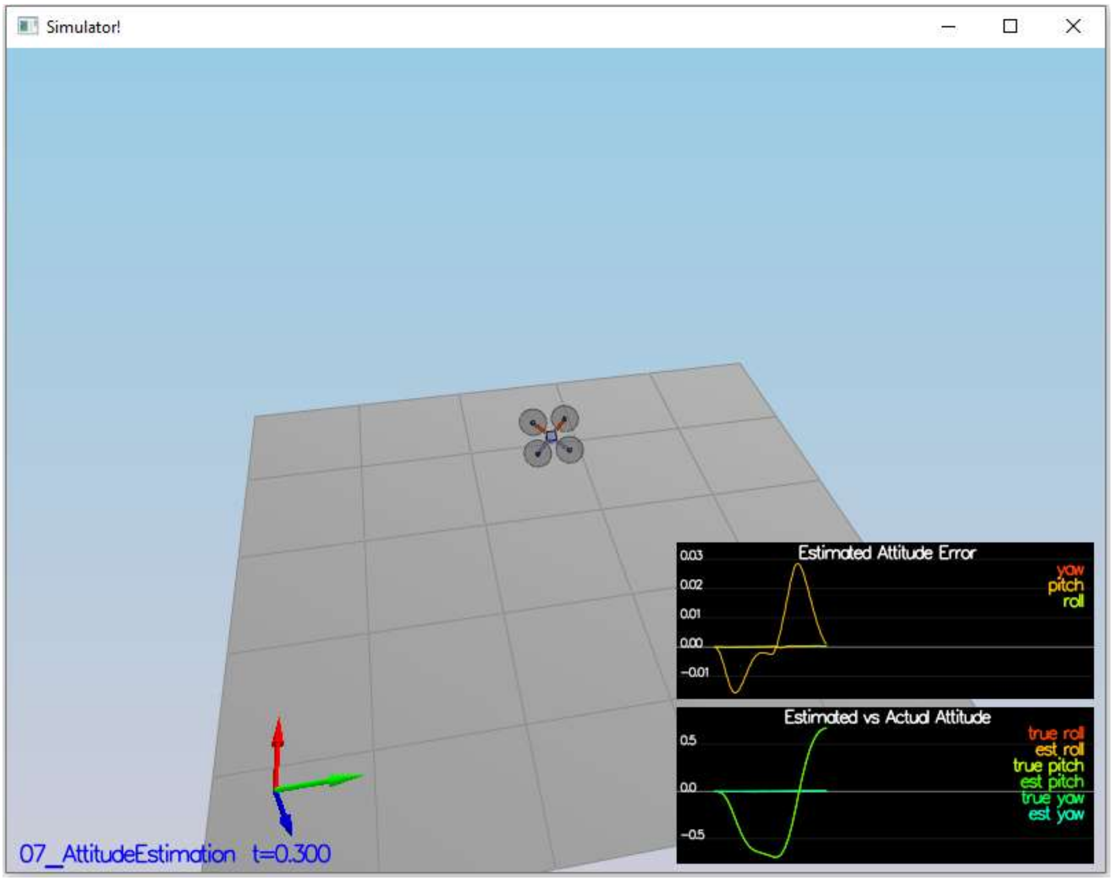
In QuadEstimatorEKF.cpp, the function UpdateFromIMU() contains a complementary filter-type attitude filter. To get better result a non-linear was required. Implemented a better rate gyro attitude integration scheme in the UpdateFromIMU() function. Refer UpdateFromIMU() function implementation in QuadEstimatorEKF.cpp *Line (74).*

First, we need to find the roll, pitch and yaw derivates using the following equation from the control lectures:

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

Once we have the derivates, we can multiply them by dt to approximate the integral.

I reduced the attitude errors to get within 0.1 rad for each of the Euler angles, as shown in the screenshot below.
Simulation #214 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for at least 3.000000 seconds attitude example



3: Prediction Step

Task: *Implement all of the elements of the prediction step for the estimator.*

In QuadEstimatorEKF.cpp, I implemented the state prediction step in the PredictState() functon, a correct calculation of the Rgb prime matrix, and a proper update of the state covariance. The acceleration should be accounted for as a command in the calculation of gPrime. The covariance update should follow the classic EKF update equation.

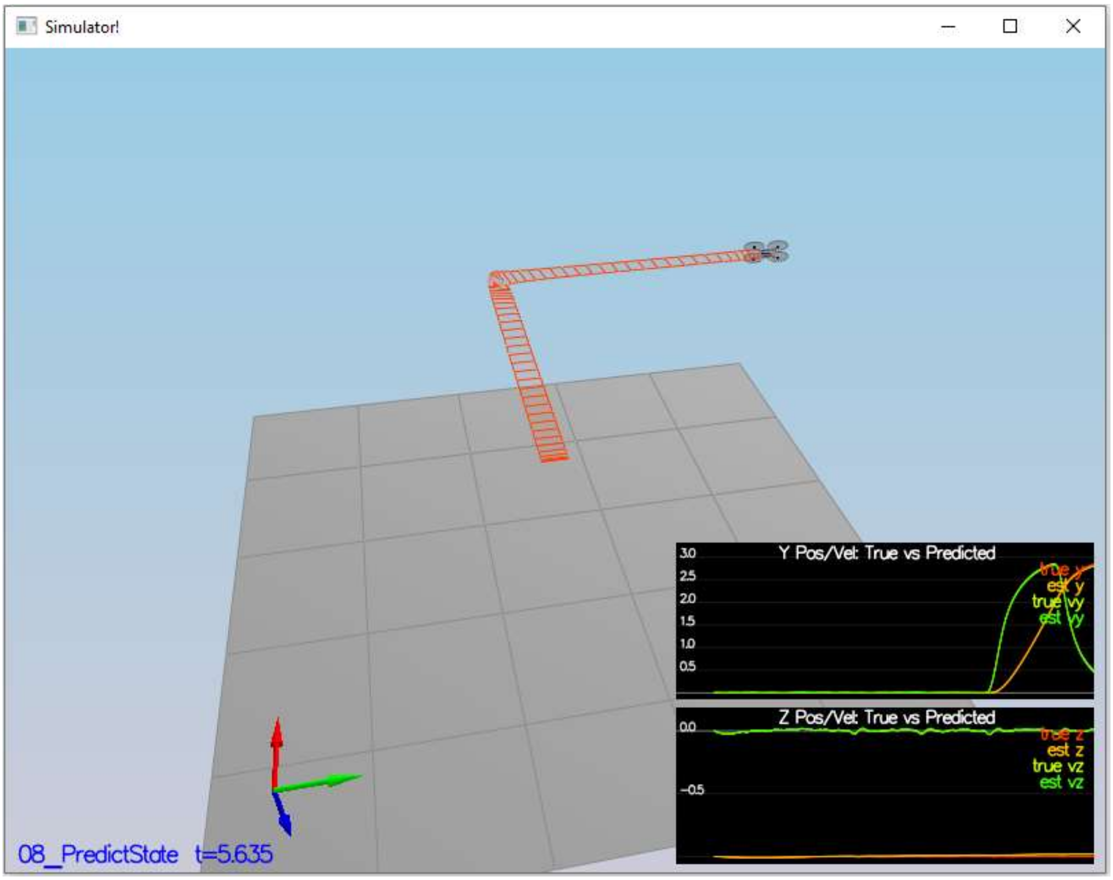
From [1] Algorithm 2, the prediction function has the following form:

```
$function\ PREDICT(\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta_t)$
.    $\bar{\mu}_t = g(u_t, \mu_{t-1})$
.    $G_t = g'(u_t, x_t, \Delta_t)$
.    $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + Q_t$
.    $return\ \bar{\mu}_t, \bar{\Sigma}_t$
```

In the C++ code implementation, the PredictState() function is given by the transition function g(), passing to it the control parameters u_t, the current state x_t, and the sensor time delta_t

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

I used attitude.Rotate_BtoI() *Refer to line 189 in QuadEstimatorEKF.cpp* to convert the true acceleration from body frame to the global frame.
The estimator state track the actual state, with only reasonably slow drift, as shown in the picture below:



We use the gyro acceleration to feed in the Jacobian given from [1] eq (51) as:

$$g^{'}(x_t, u_t, \Delta_t) = \begin{bmatrix} 1 & 0 & 0 & \Delta_t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta_t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta_t & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & R^{'}_{bg}[0:]u_t[0:3]\Delta_t \\ 0 & 0 & 0 & 0 & 1 & 0 & R^{'}_{bg}[1:]u_t[0:3]\Delta_t \\ 0 & 0 & 0 & 0 & 0 & 1 & R^{'}_{bg}[2:]u_t[0:3]\Delta_t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In QuadEstimatorEKF.cpp, I calculate the partial derivative of the body-to-global rotation matrix in the function GetRbgPrime() base on the equation.

Refer to line 201,273 in QuadEstimatorEKF.cpp

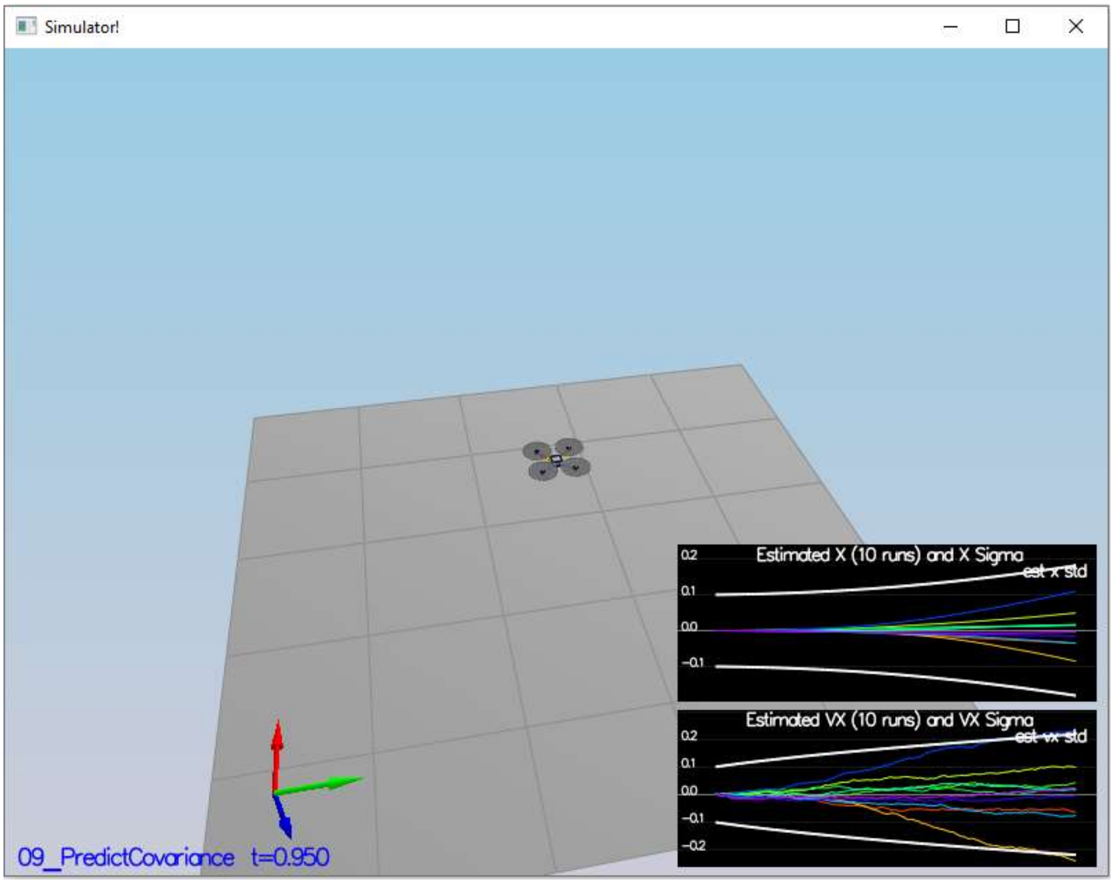
$$R^{'}_{bg} = \begin{bmatrix} -cos\theta sin\psi & -sin\phi sin\theta sin\psi - cos\phi sin\psi & -cos\phi sin\theta sin\psi + sin\phi cos\psi \\ cos\theta cos\psi & sin\phi sin\theta cos\psi - cos\phi sin\psi & cos\phi sin\theta cos\psi + sin\phi sin\psi \\ 0 & 0 & 0 \end{bmatrix}$$

Then I implemented the rest of the prediction step (predict the state covariance forward) in Predict() base on the the Jacobian of g to perform a linear approximation of the system:

$$g'(x_t, u_t, \Delta t) = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \frac{\partial}{\partial x_{t,\psi}}\left(x_{t,\dot{x}} + R_{bg}[0:]u_t[0:3]\Delta t\right) \\ 0 & 0 & 0 & 0 & 1 & 0 & \frac{\partial}{\partial x_{t,\psi}}\left(x_{t,\dot{y}} + R_{bg}[1:]u_t[0:3]\Delta t\right) \\ 0 & 0 & 0 & 0 & 0 & 1 & \frac{\partial}{\partial x_{t,\psi}}\left(x_{t,\dot{z}} + R_{bg}[2:]u_t[0:3]\Delta t\right) \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (50)$$

$$= \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & R'_{bg}[0:]u_t[0:3]\Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 & R'_{bg}[1:]u_t[0:3]\Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 & R'_{bg}[2:]u_t[0:3]\Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (51)$$

The dotted line is growing showing sigma growing over time due to the prediction step. Following is a scenario picture:


Simulator!
Estimated X (10 runs) and X Sigma
est x std
0.2
0.1
0.0
-0.1
Estimated VX (10 runs) and VX Sigma
est vx std
0.2
0.1
0.0
-0.1
-0.2
09_PredictCovariance t=0.950

4: Magnetometer Update
Task: *Implement all of the elements of the prediction step for the estimator.*

Implemented magnetometer update in the function UpdateFromMag() base on the paper Estimation for Quadrotors, 7.3.2 Magnetometer.

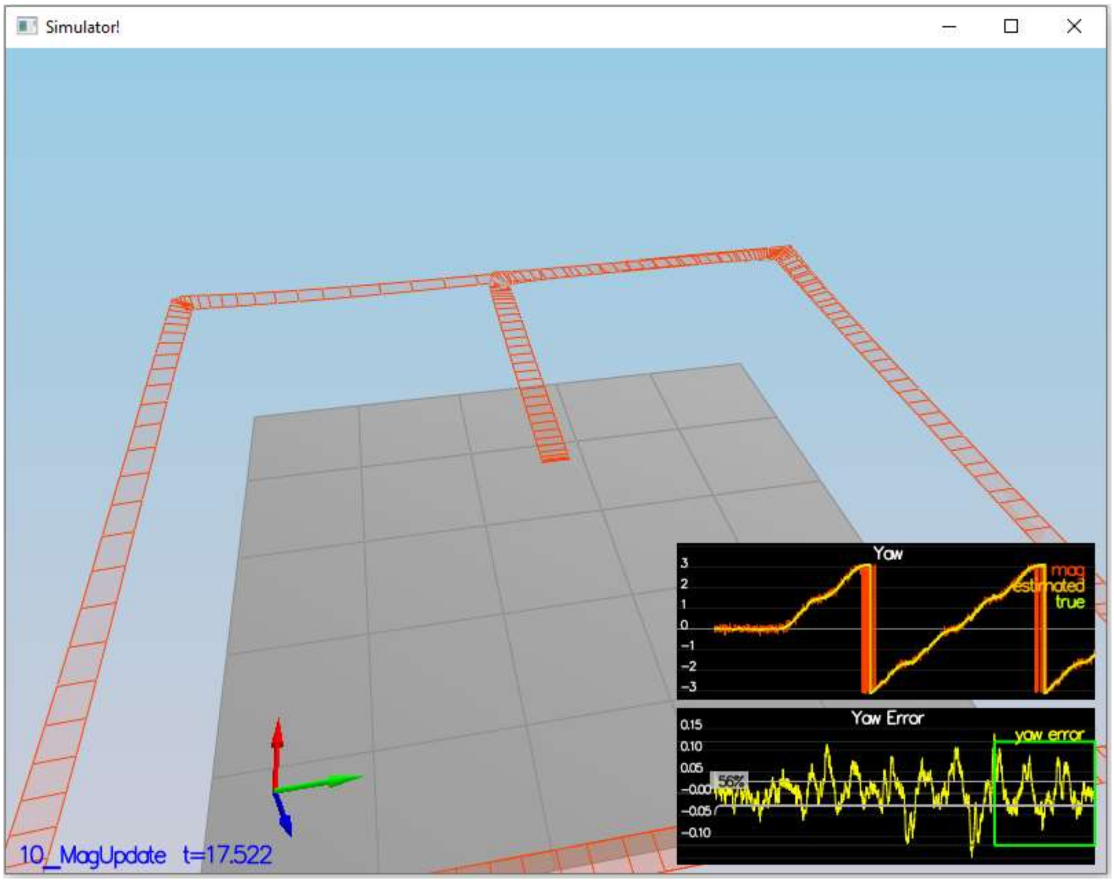
Refer to line 334 in QuadEstimatorEKF.cpp

First I Tune the parameter QYawStd to 0.08 in QuadEstimatorEKF.txt and it approximately captures the magnitude of the drift, as in picture below.

I successfully obtained an estimated standard deviation that accurately captures the error and maintain an error of less than 0.1rad in heading for at least 10 seconds of the simulation.

```
Simulation #642 (../config/10_MagUpdate.txt)
PASS: ABS(Quad.Est.E.Yaw) was less than 0.120000 for at least 10.000000
seconds
PASS: ABS(Quad.Est.E.Yaw-0.000000) was less than Quad.Est.S.Yaw for
59% of the time
```

You can see there the yaw error decrease, and the sigma remained stable. Here is a scenario picture:

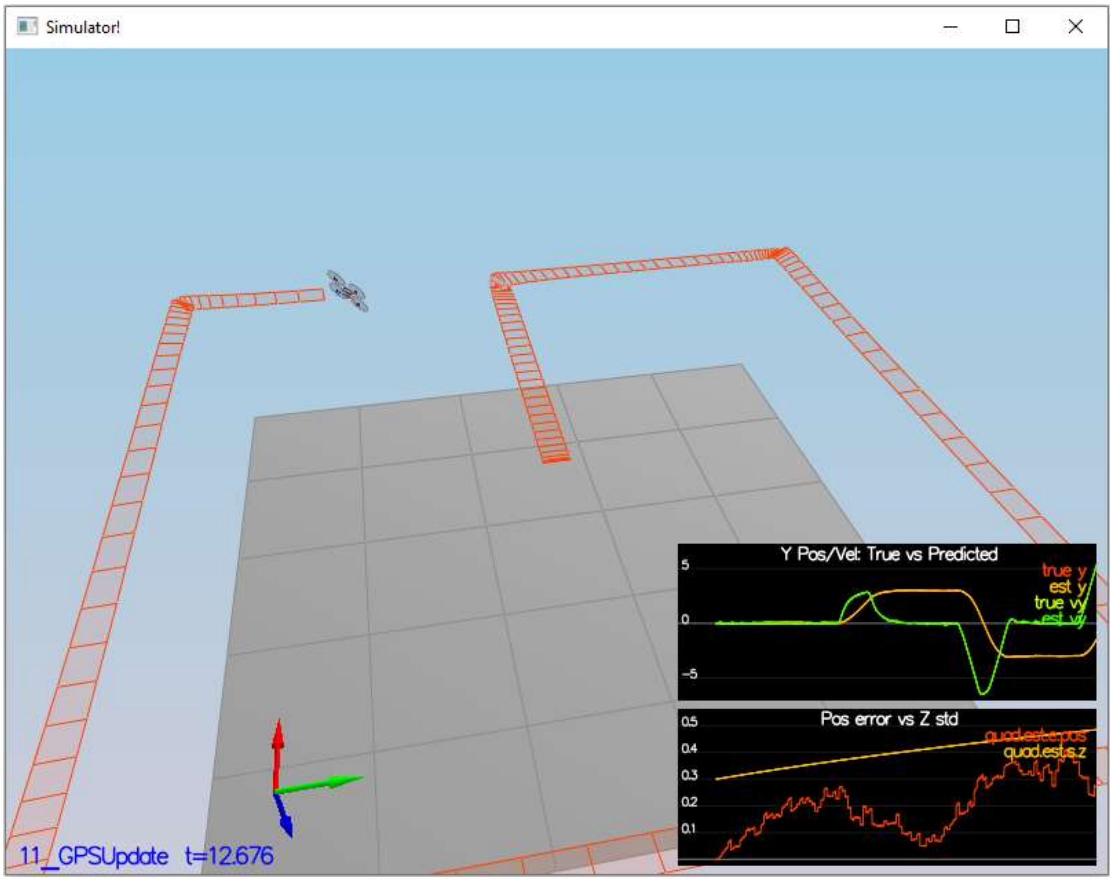


5: Implement the GPS update
Task: *The estimator should correctly incorporate the GPS information to update the current state estimate.*

In the task of GPS Update. I implemented the EKF GPS Update in the function UpdateFromGPS() base on the paper section 7.3.1 of Estimation for Quadrotors for a refresher on the GPS update.

Refer to lines 296 to 332 in QuadEstimatorEKF.cpp

We can see there the position error, and sigma decreased. After it is implemented, received this data and here is a scenario picture:



When the scenario is passing the test, got his line on the standard output:

```
Simulation #668 (../config/11_GPSUpdate.txt)
PASS: ABS(Quad.Est.E.Pos) was less than 1.000000 for at least 20.000000 seconds
```

GPS update requires to pass the proper parameters to the EKF's Update function, this time the parameters are given from equations (53), (54), and (55).

$$GPS\ Measurement:\ z_t = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

$$Measurement\ Model:\ h(x_t) = zFromX = \begin{bmatrix} x_{t,x} \\ x_{t,y} \\ x_{t,x} \\ x_{t,\dot{x}} \\ x_{t,\dot{y}} \\ x_{t,\dot{z}} \end{bmatrix}$$

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

6: Adding Your Controller

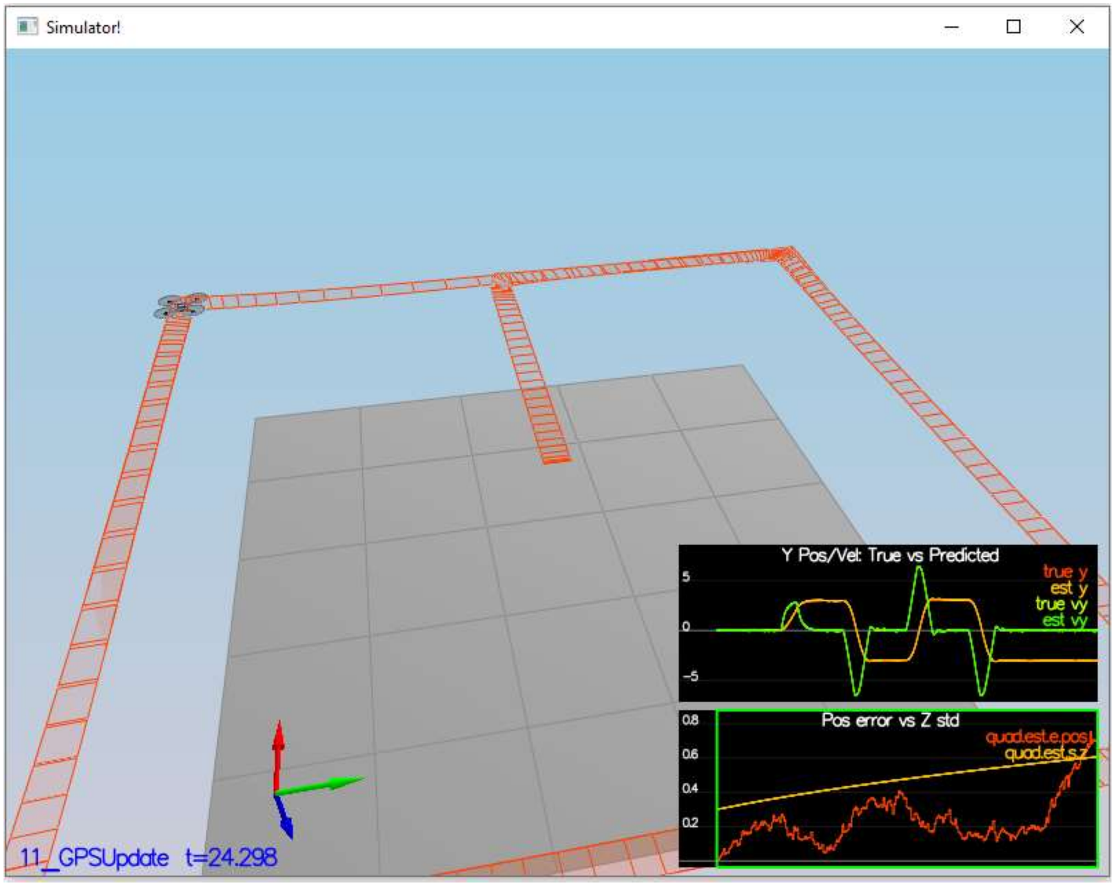
Task: For each step of the project, the final estimator should be able to successfully meet the performance criteria with the controller provided. The estimator's parameters should be properly adjusted to satisfy each of the performance criteria elements.

I replaced QuadController.cpp with the controller from my control project. And also replaced QuadControlParams.txt with the control parameters from my last project.
completed the entire simulation cycle with estimated position error of < 1m.* without tuning the the control parameters.

Run scenario 11_GPSUpdate again, got the following result:

Simulation #674 (../config/11_GPSUpdate.txt)

PASS: ABS(Quad.Est.E.Pos) was less than 1.000000 for at least 20.000000 secondsHere is a scenario picture:


Simulator!
Y Pos/Vel: True vs Predicted
true y
est y
true vy
est vy
5
0
-5
Pos error vs Z std
quad.est.e.pos.
quad.est.s.z
0.8
0.6
0.4
0.2
11_GPSUpdate t=24.298