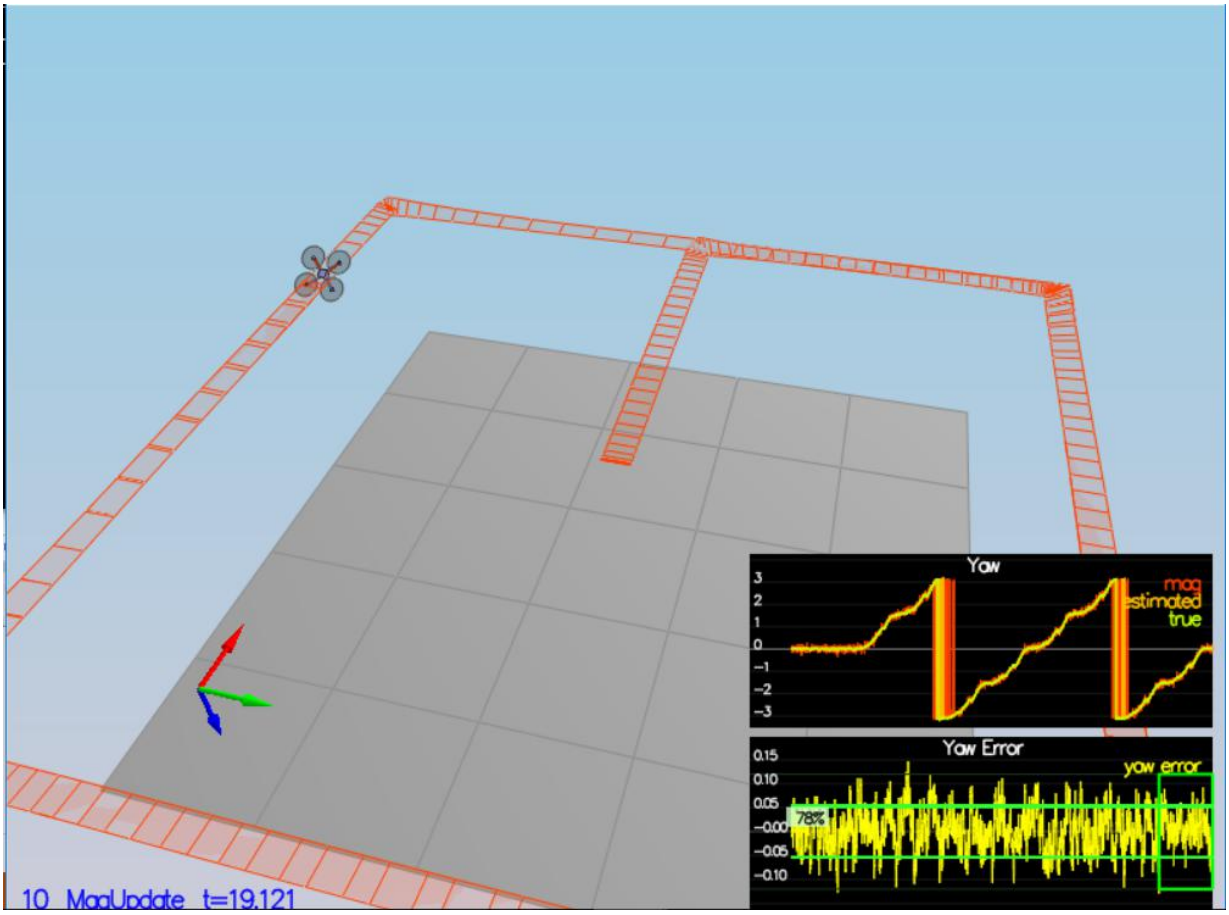
August 4, 2018 BY SWAROOPKS

Building an Estimator



The fourth and final project is about building a estimation project on top of controller project to get close to th reality with existence of noise . In this project , we are using Extended kalman filter by fusing noisy GPS , IMU and compass (magnetometer) to estimate current drone position , velocity and yaw . EKF filter is implemented using c++ with basic code provided by Udacity.

Prerequisites for project (windows)

This project will continue to use the C++ development environment you set up in the Controls C++ project.Clone the repository

```
git clone https://github.com/udacity/FCND-Estimation-CPP.git
```

Import the code into your IDE like done in the Controls C++ project

We should now be able to compile and run the estimation simulator just as you did in the controls project

Project description

The steps of this project are following :

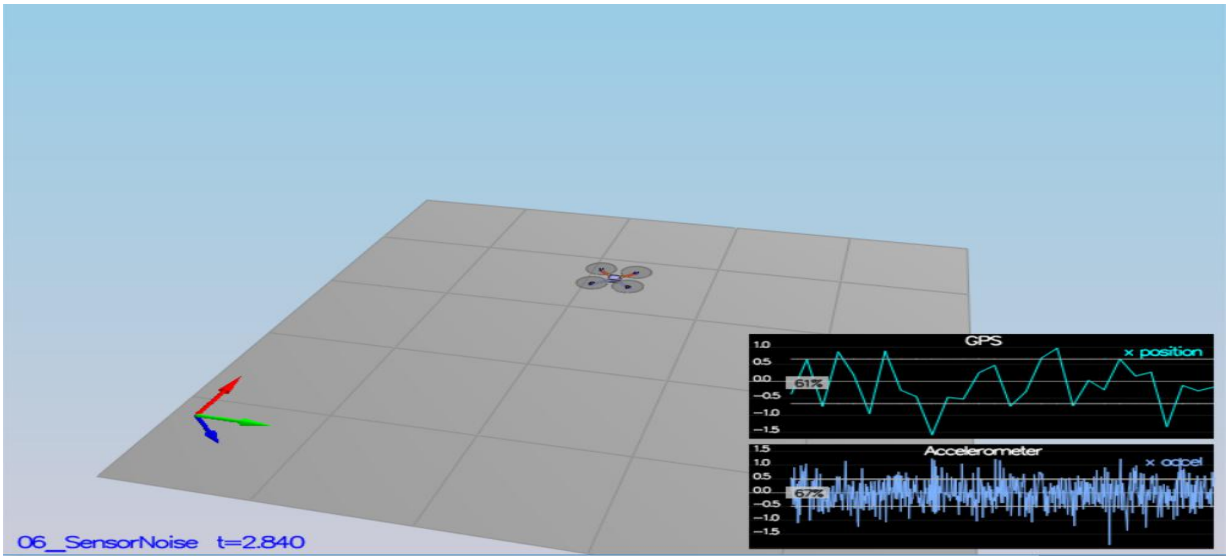
- **Sensor Noise :** Collect simulated noisy sensor data and determine standard deviation of the quad's sensor.
- **Attitude Estimation:** Implement a better integration method that uses the current attitude estimate (rollEst, pitchEst and ekfState(6)) to integrate the body rates into new Euler angles. The only thing we need to do is to integrate `pqr` from the gyroscope into the estimated pitch and roll
- **Prediction step :** Implementation of prediction filter based on the acceleration measurement by using Dead Reckoning method.
- **Magnetometer update:** the information from the magnetometer is added to improve filter's performance in estimating the vehicle's heading.
- **Closed loop & GPS Update :** GPS update is implemented into estimator.
- **Adding Your Controller :** Tuning Control parameter to overall improve performance in all scenarios

Sensor Noise:

In this step , collecting some simulated noisy sensor data (GPS and IMU measurements) and estimate the standard deviation of those sensors. I wrote a small python code to parse data from text file of respective sensors to calculate standard deviation. Later these measurement are updated to parameter file `config/6_Sensornoise.txt`.

```
MeasuredStdDev_GPSPosXY = 0.6569747495144732

MeasuredStdDev_AccelXY = 0.4909264198233001
```



Results :

```
Simulation #66 (../config/06_SensorNoise.txt)
PASS: ABS(Quad.GPS.X-Quad.Pos.X) was less than MeasuredStdDev_GPSPosXY for 66% of the time
PASS: ABS(Quad.IMU.AX-0.000000) was less than MeasuredStdDev_AccelXY for 68% of the time
```

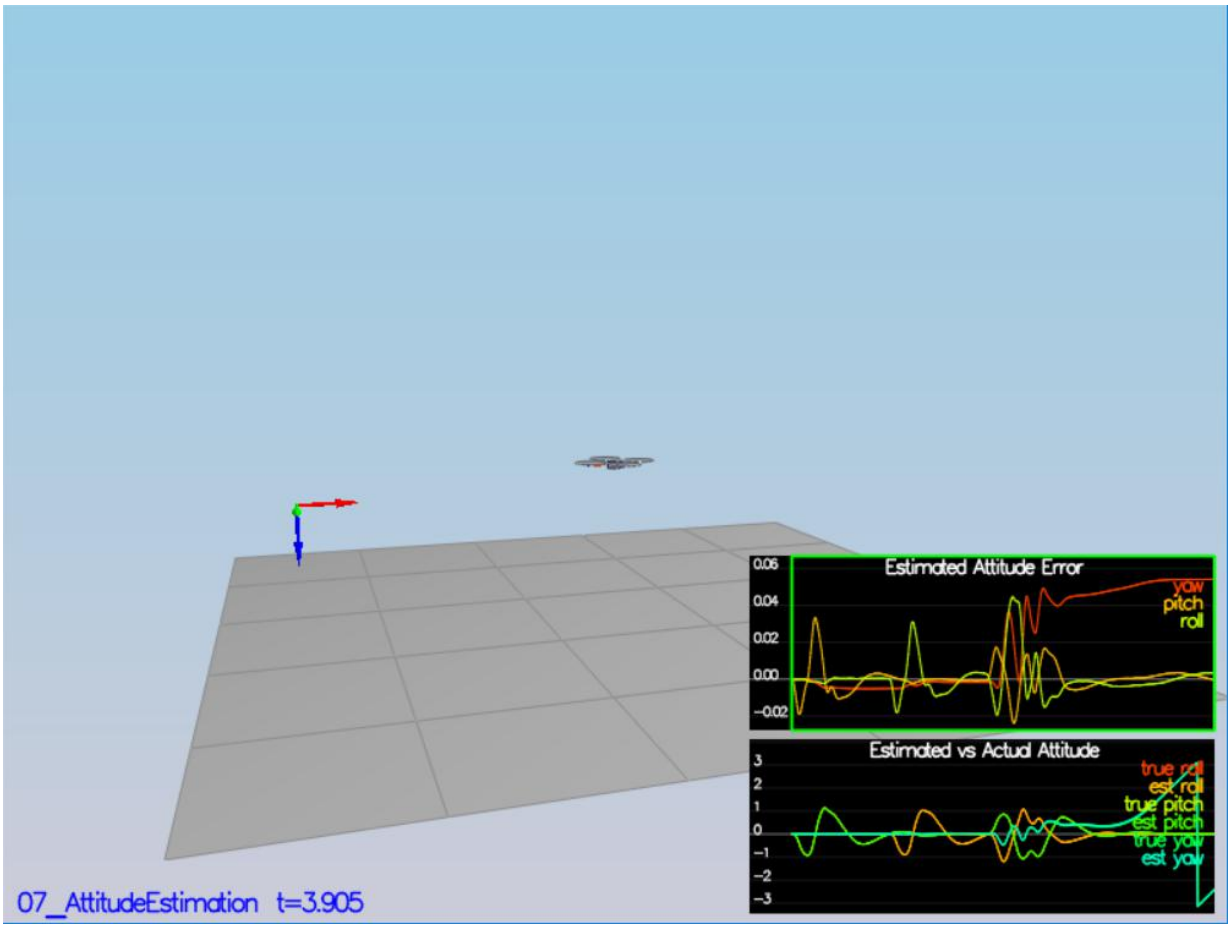
Attitude Estimation:

In this step , the complementary filter-type attitude filter is improved by integrating body rate `p,q,r` which obtained from rate gyro into the estimated pitch and roll angle.

By using below mentioned equation , an instantaneous change in euler angles (World frame) is obtained from turn rate in body frame ,

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

This step is followed by integrating Euler Rate into the estimated pitch and roll angle.



Results of scenario 2:

```
Simulation #91 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for at least 3.000000 seconds
```

Prediction step :

Predict the current state forward by time dt using current accelerations and body rates as input.
Use attitude.Rotate_BtoI(<V3F>) to rotate a vector from body frame to inertial frame

Calculating the partial derivative of the body-to-global rotation matrix in the function `GetRbgPrime()` by using equation,

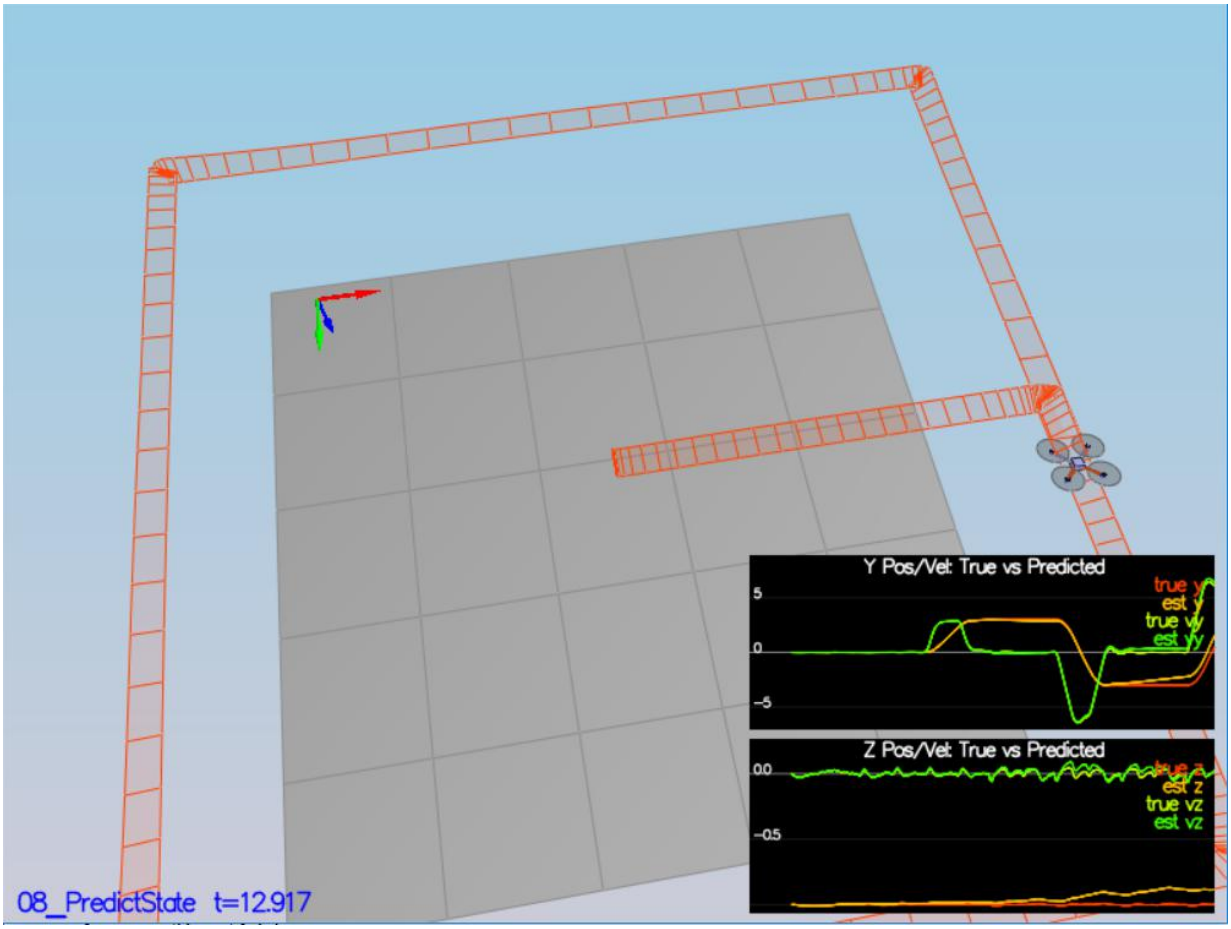
$$R_{bg} = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi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

Covariance Prediction

Obtaining `Jacobian Matrix` by using `GetRbgPrime()` and after iplementing prediction step (predict the state covariance forward) to update the covariance matrix cov according to the EKF equation.

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

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



To capture the magnitude of error more precisely, I also tuned the parameters in `QuadEstimatorEKF.txt`

```
QPosXYStd = .01
QVelXYStd = .2
```

prediction equations as shown below,

$$
\begin{aligned}
&\textbf{function } \textsc{Predict}(\mu_{t-1}, \Sigma_{t-1}, u_t, \Delta t) \\
&\quad \bar{\mu}_t = g(u_t, \mu_{t-1}) \\
&\quad G_t = g'(u_t, x_t, \Delta t) \\
&\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + Q_t \\
&\quad \textbf{return } \bar{\mu}_t, \bar{\Sigma}_t
\end{aligned}
$$

```
This step doesn't have any specific measurable criteria being checked.
```

Magnetometer update:

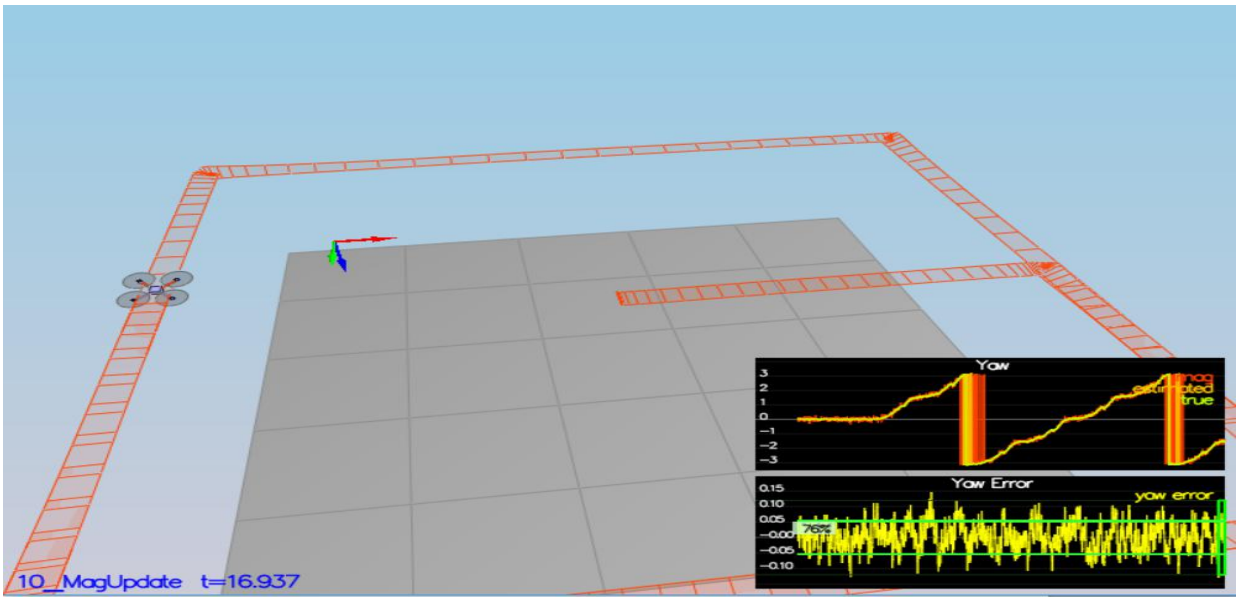
Up until now we've only used the accelerometer and gyro for our state estimation. In this step, We added the information from the magnetometer to improve your filter's performance in estimating the vehicle's heading.

In this step, the information from the magnetometer is added to improve filter's performance in estimating the vehicle's heading.

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

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

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



I also tuned the parameters in `QuadEstimatorEKF.txt` for the QuadEstimatorEKF so that it approximately captures the magnitude of the drift

```
QYawStd = .3
```

Results of scenario 4:

```
Simulation #212 (../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 78% of the time
```

Closed loop & GPS Update :

In this step, GPS update is implemented into estimator.

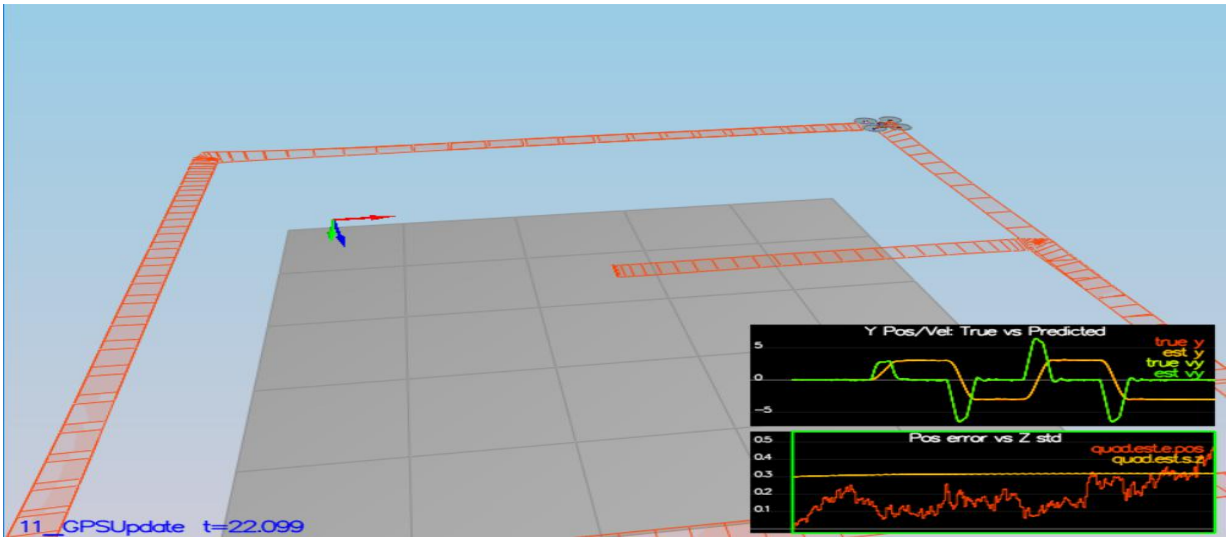
$$z_t = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}$$

Then the measurement model is:

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

Then the partial derivative is the identity matrix, augmented with a vector of zeros for $\frac{\partial}{\partial x_{t,\phi}} h(x_t)$:

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



Results of scenario 5:

```
Simulation #217 (../config/11_GPSUpdate.txt)
PASS: ABS(Quad.Est.E.Pos) was less than 1.000000 for at least 20.000000 seconds
```

Adding Your Controller :

Up to this point, we have been working with a controller that has been relaxed to work with an estimated state instead of a real state. So now, you will see how well your controller performs and de-tune your controller accordingly.

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`. If your controller crashes immediately do not panic. Flying from an estimated state (even with ideal sensors) is very different from flying with ideal pose. You may need to de-tune your controller. Decrease the position and velocity gains (we've seen about 30% detuning being effective) to stabilize it. Your goal is to once again complete the entire simulation cycle with an estimated position error of < 1m.

Success criteria: Your objective is to complete the entire simulation cycle with estimated position error of < 1m.

Final tunned parameters as follows :

```
# Position control gains
kpPosXY = 30
kpPosZ = 20
KiPosZ = 40

# Velocity control gains
kpVelXY = 10.0
kpVelZ = 8.0

# Angle control gains
kpBank = 10
kpYaw = 2

# Angle rate gains
kpPQR = 70, 70, 6
```

Conclusion

Over all , this was good project which pretty much give idea about vehicle dynamics and 3D motion control for real time senarios with noise and error from sensors using Extended kalman filter .