Flying Car Nanodegree – Estimation

Writeup

Implement Estimator:

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

- Execute the scenario once to collect GPS and Accelerometer data.
- Find the standard deviation of the data from the obtained logs.
- The found values are :

```
MeasuredStdDev_GPSPosXY = 0.6749861224119309
MeasuredStdDev_AccelXY = 0.4757094158613545
```

Python code to find standard deviation

```
import numpy as np

GPS_x   = np.loadtxt('Graph1.txt',delimiter=',',dtype='Float64',skiprows=1)[:,1]

Accel_x = np.loadtxt('Graph2.txt',delimiter=',',dtype='Float64',skiprows=1)[:,1]

print("Std. Dev. of GPS X data : ", np.std(GPS_x))

print("Std. Dev. Accel X data : ",np.std(Accel_x))
```

Implement a better rate gyro attitude integration scheme in the UpdateFromIMU() function:

- In this function the Complimentary filter is improved by integrating he body rates.
- The function `FromEuler123_RPY()` is used to find the quaternion from the estimated roll, pitch and yaw.
- Then the gryo and IMU measurements are integrated with the body rates using the function `IntegrateBodyRate()`.
- This is implemented in the function `UpdateFromIMU()`

Implement all of the elements of the prediction step for the estimator:

Find the covariance and new state in the prediction part as 2 step process. The 2 process are implemented in the function `Predict()`.

Step 1:

- The prediction is based on the current acceleration and body rates measurement using Dead Reckoning method.

- The state transition uses the time difference dt, along with the current state and its velocity to predict the new state. Similarly, for the velocity with acceleration as inputs.
- As the acceleration is in body frame, it will be converted to the inertial frame using the function `Rotate_BtoI()`.
- As the yaw is integrated already in IMU update, it was not integrated again here.
- This is implemented in the function **`PredictState()`**.

Step 2:

- The partial derivative of the RBG matrix is calculated using the below mentioned formula. This is implemented in the function **`GetRbgPrime()`**
- The derivative of the g is calculated using the below mentioned formula.
- Finally, with the obtained g', we can estimate the new covariance from the current covariance using the below mentioned algorithms.
- The **`QPosXYStd`** and the **`QVelXYStd`** parameters can be tuned to reduce the errors.

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

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

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

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

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

Implement the magnetometer update:

- The filter's performance can be improved by estimating the drone's heading using magnetometer measurements.
- The difference between the current estimated yaw and the measured yaw is calculated and the difference is normalized.
- Also, update the tuneable parameter `QYawStd`, to capture the magnitude of the drift during estimation.
- This operations are implemented in the function `UpdateFromMag()`.

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

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

Implement the GPS update:

- Use the ideal estimator and realistic sensors by changing the parameters in `config/11_GPSUpdate.txt`
- The GPS update is integrated into the estimator
- The observation state, measurement model h, and its derivative h' are calculated as defined below.
- The steps are implemented in the function `UpdateFromGPS()`.

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

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

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

Flight Evaluation

Meet the performance criteria of each step.:

- The performance criteria of each step were met successfully.
- Images and output of all the scenarios are added below.

De-tune your controller to successfully fly the final desired box trajectory with your estimator and realistic sensors.

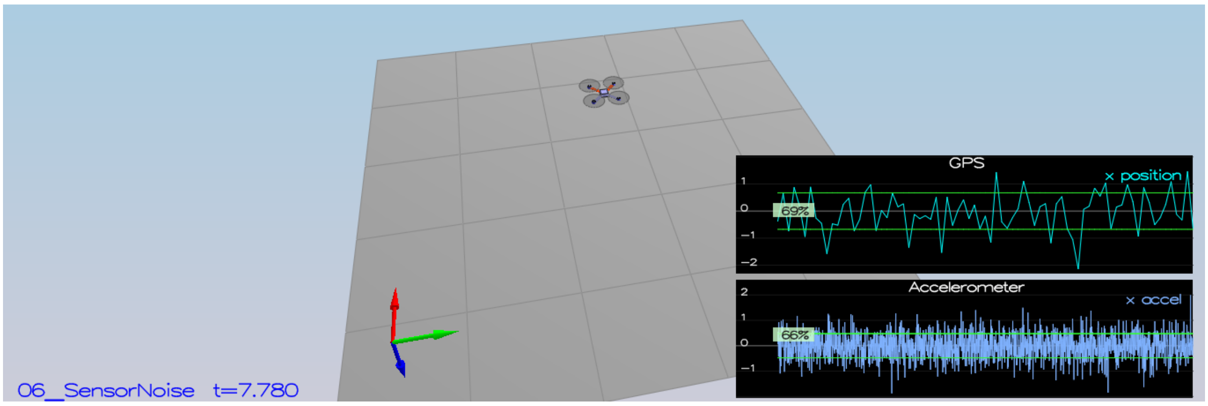
- The controller parameters from the earlier project was adapted for the current estimation project. The values of the positional control gains and velocity control gains are reduced.
- Also, the usage of ideal estimator is removed and realistic sensors were used using the below code.

```
Quad.UseIdealEstimator = 0

#SimIMU.AccelStd = 0,0,0
#SimIMU.GyroStd = 0,0,0
```

OUTPUTS:

Scenario 6:



PASS: ABS(Quad.GPS.X-Quad.Pos.X) was less than MeasuredStdDev_GPSPosXY for 68% of the time

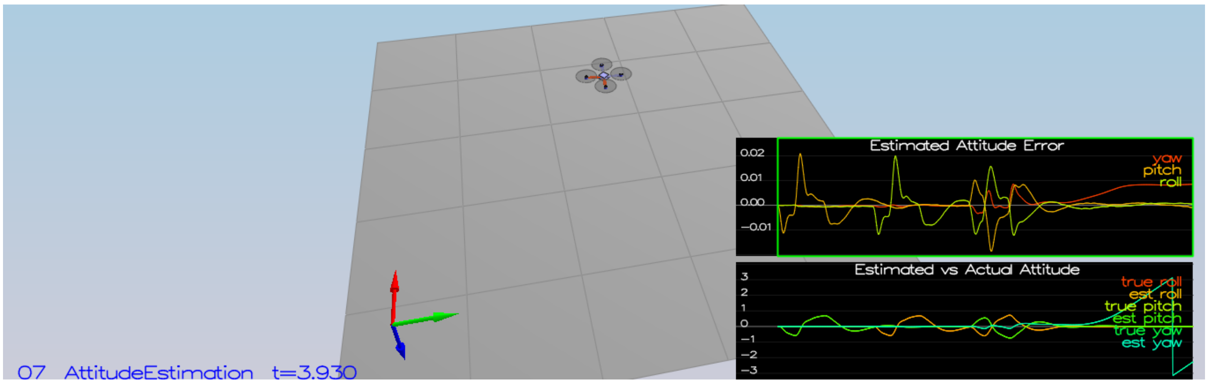
PASS: ABS(Quad.IMU.AX-0.000000) was less than MeasuredStdDev_AccelXY for 67% of the time

Simulation #16 (../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 67% of the time

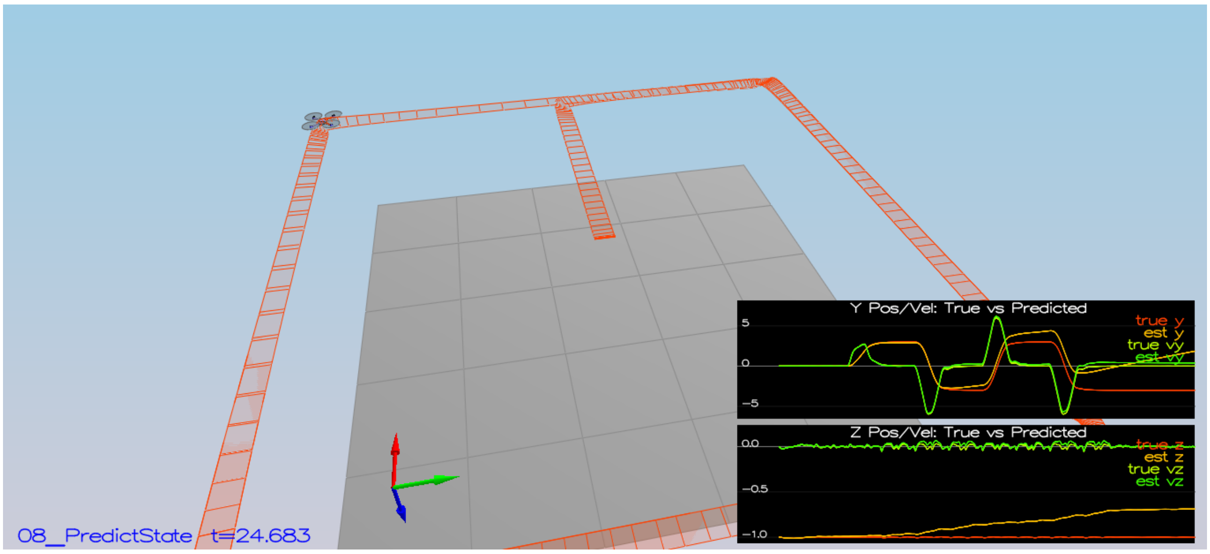
Scenario 7:



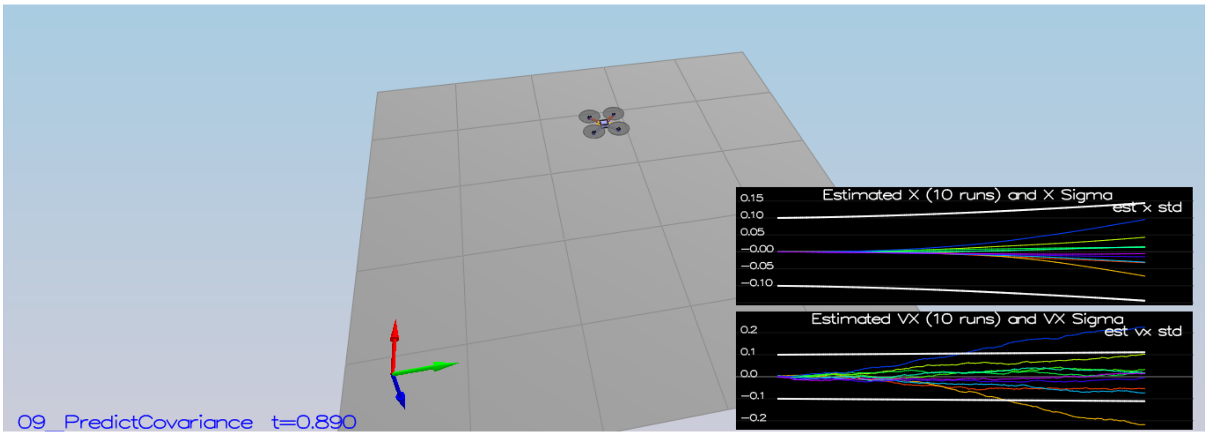
Simulation #2 (../config/07_AttitudeEstimation.txt)

PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for at least 3.000000 seconds

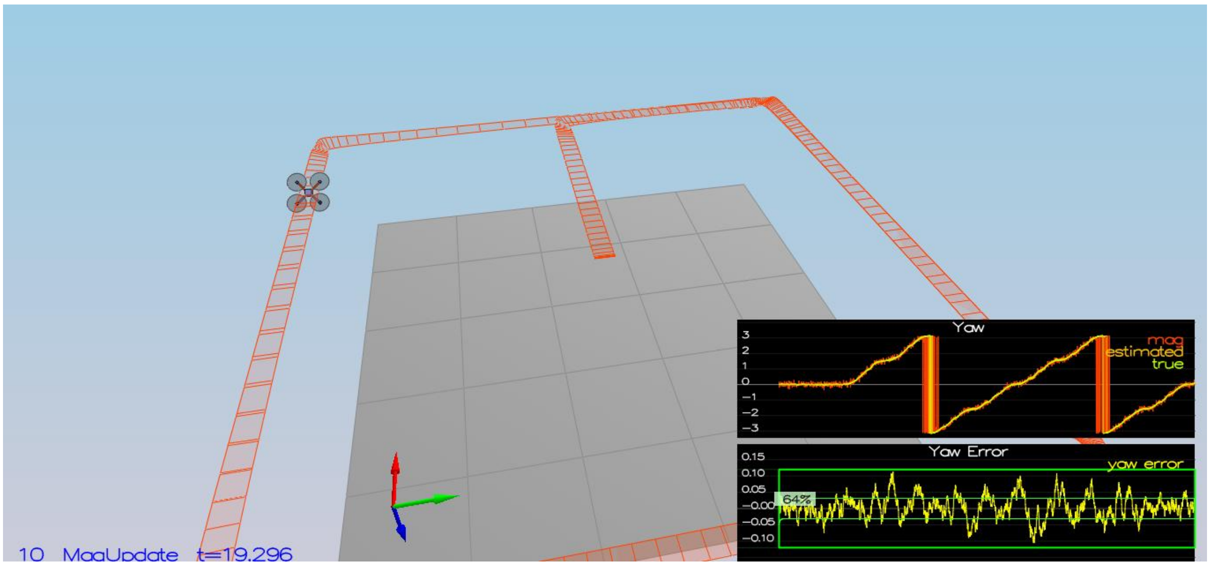
Scenario 8:



Scenario 9:



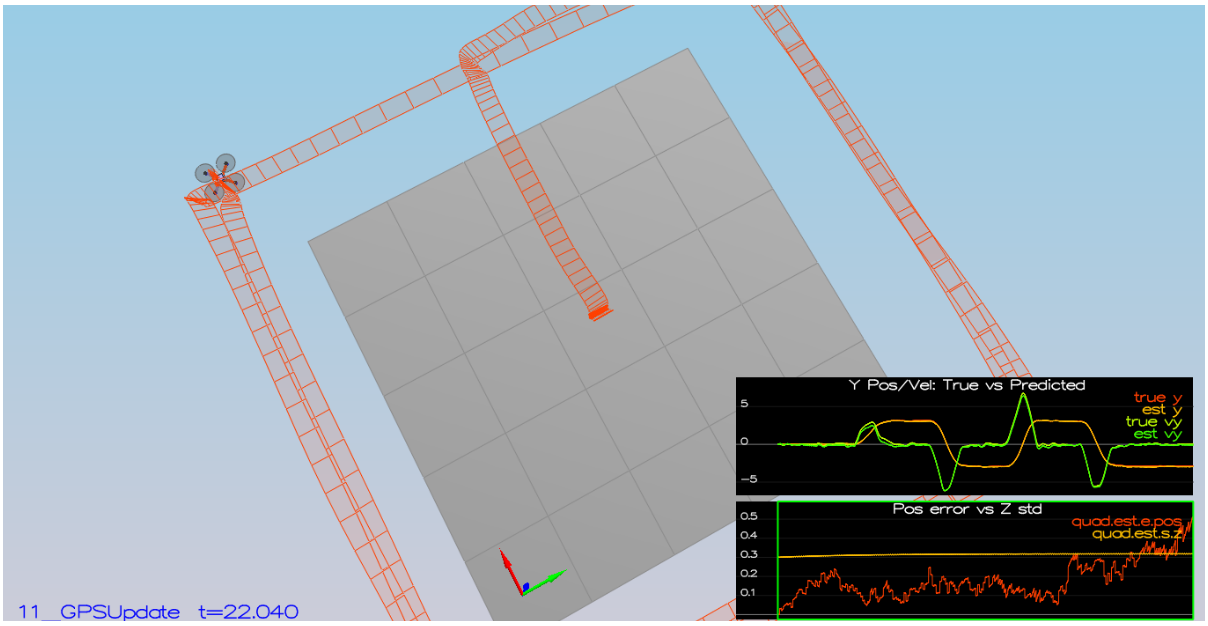
Scenario 10:



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 65% of the time

Scenario 11:



Simulation #2 (../config/11_GPSUpdate.txt)

PASS: ABS(Quad.Est.E.Pos) was less than 1.000000 for at least 20.000000 seconds