Building an Estimator

1. Tasks

1.1. Sensor Noise

A script is written to generate the standard deviation of the measurement noise.

The following is the code:

```
gvenkat@PT7910:~/FCND/FCND-Estimation-CPP$ more stdd.py
from statistics import stdev
filename = "config/log/Graph1.txt"

gps_x_data = []
acc_x_data = []

with open(filename , 'r') as f:
    first_line = f.readline()
    for line in f:
        gps_x_data.append(float(line.split(',')[1]))

filename = "config/log/Graph2.txt"
with open(filename , 'r') as f:
    first_line = f.readline()
    for line in f:
        acc_x_data.append(float(line.split(',')[1]))

print(len(gps_x_data))
print(gps_x_data)
```

```
print(len(acc_x_data))

print(acc_x_data)


print(stdev(gps_x_data))

print(stdev(acc_x_data))
```

The values are plugged in 06_Sensor_Noise.txt as below.

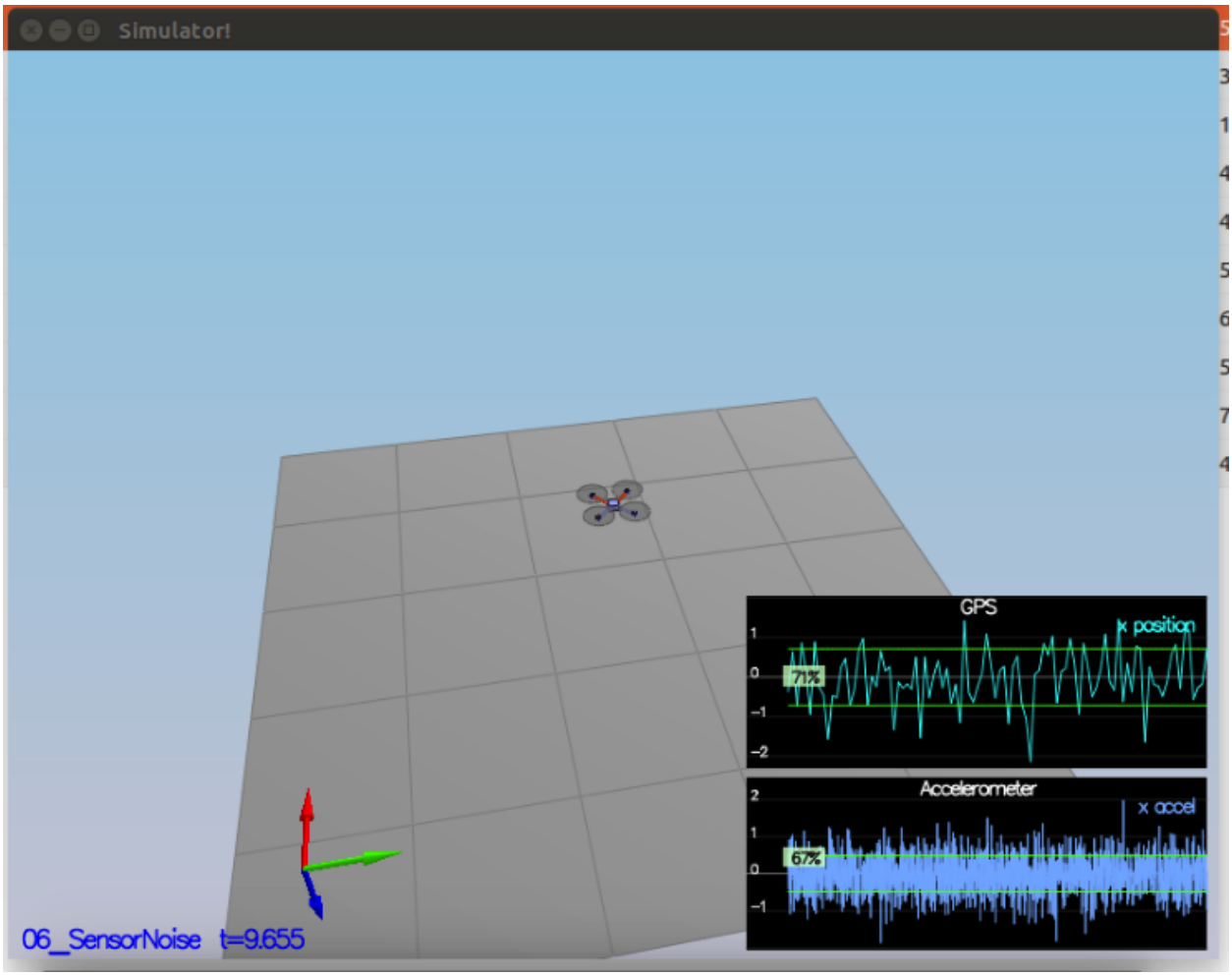
```
### STUDENT SECTION

MeasuredStdDev_GPSPosXY = 0.712796

MeasuredStdDev_AccelXY = 0.488375

### END STUDENT SECTION
```

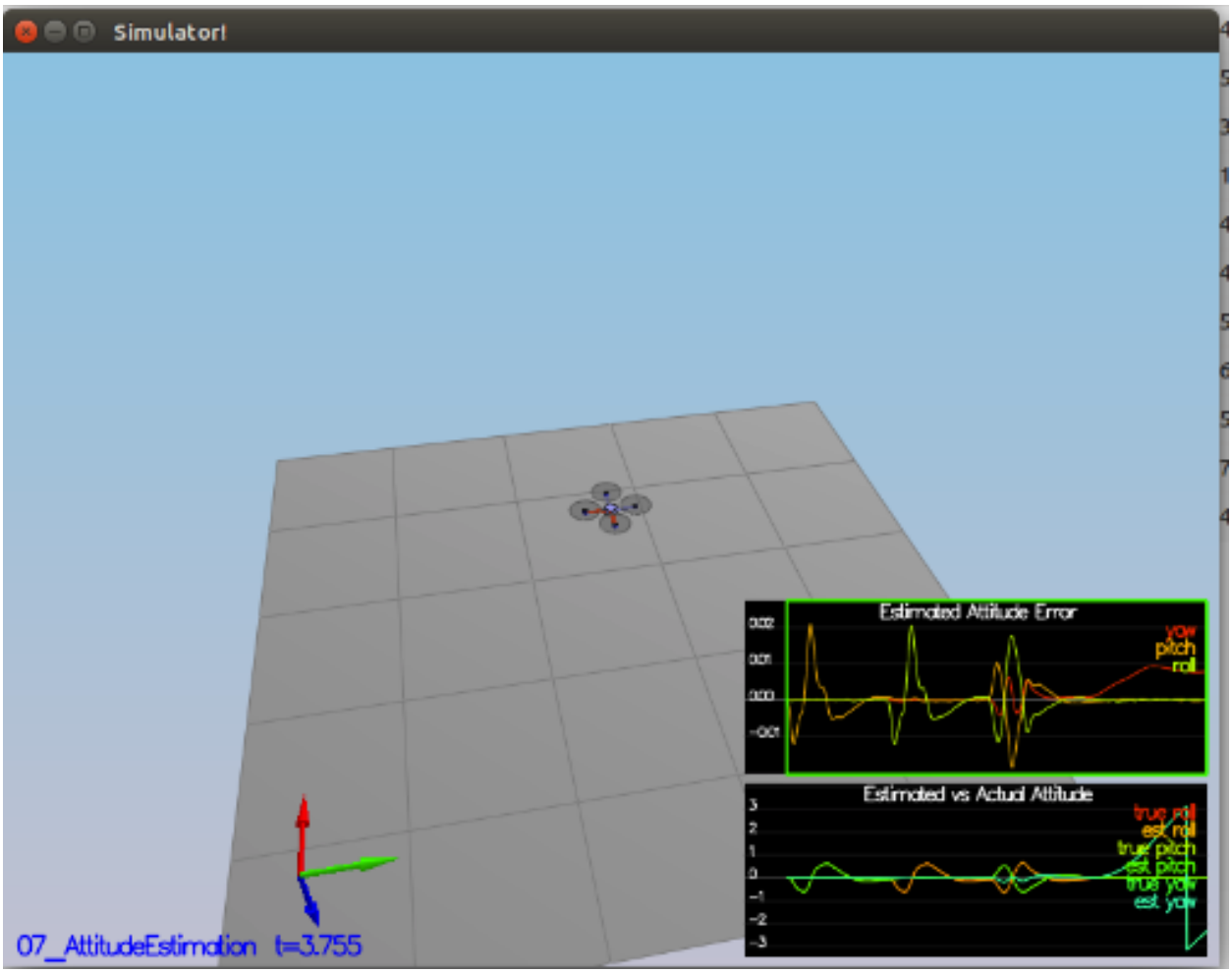
Following these changes, we get approximately 68% of the GPS and accelerometer measurements within +/- 1 sigma. Please take a look at the screen capture below.



1.2. Attitude Estimation

The code changes are implemented in QuadEstimatorEKF.cpp:100.110 lines. I used Quaternions and the FromEuler123_RPY as suggested in the hints.

The following is the screen captured which shows the green box



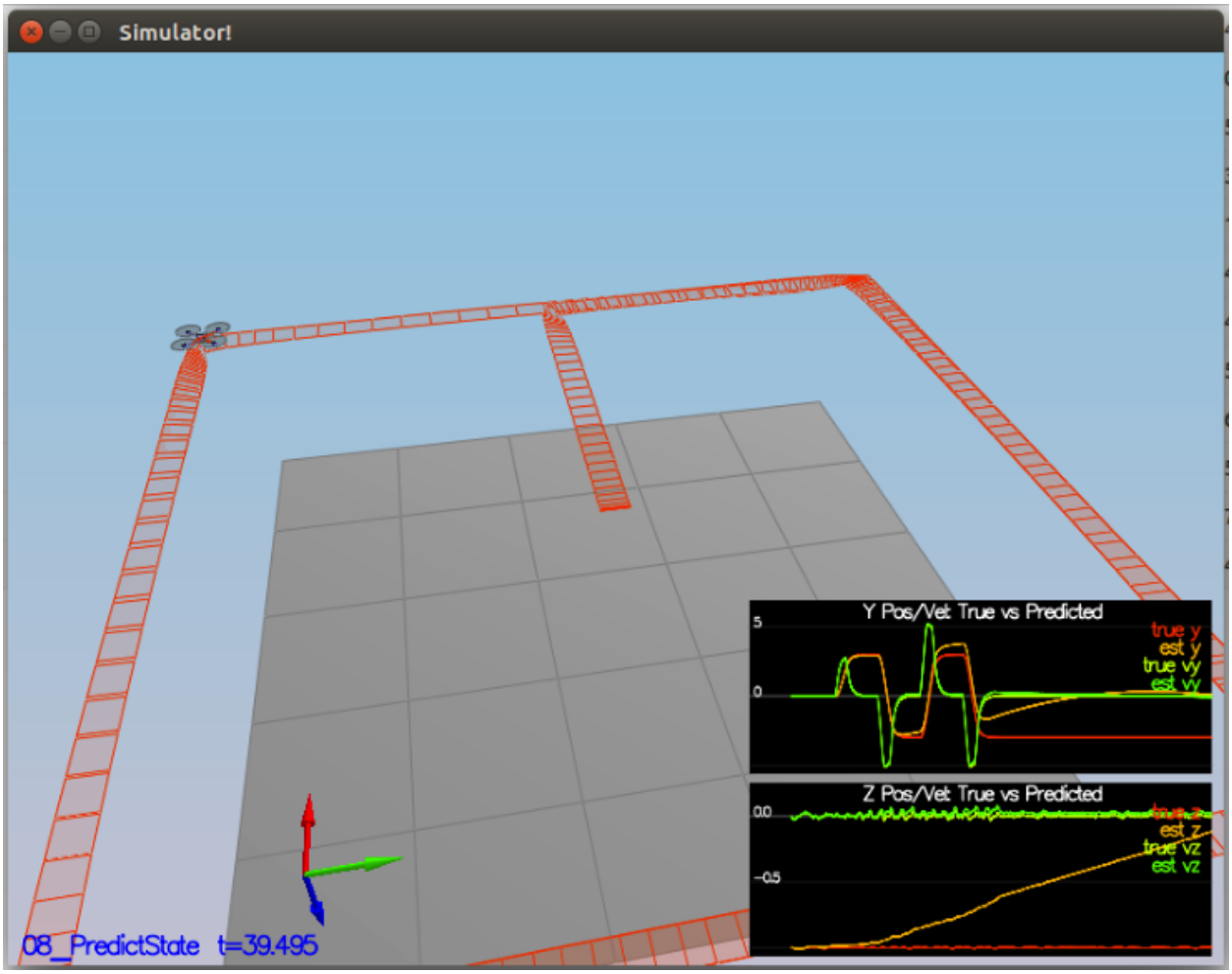
1.3. Prediction Step

1.3.1. Implementing the predict state

This is implemented in QuadEstimatorEKF.cpp:173..181 lines

This is done by integrating the current state.

Once it is run, the below is the screen capture where we see that the estimator track follows tine actual state.



1.3.2. Implementing the GetRbgPrime()

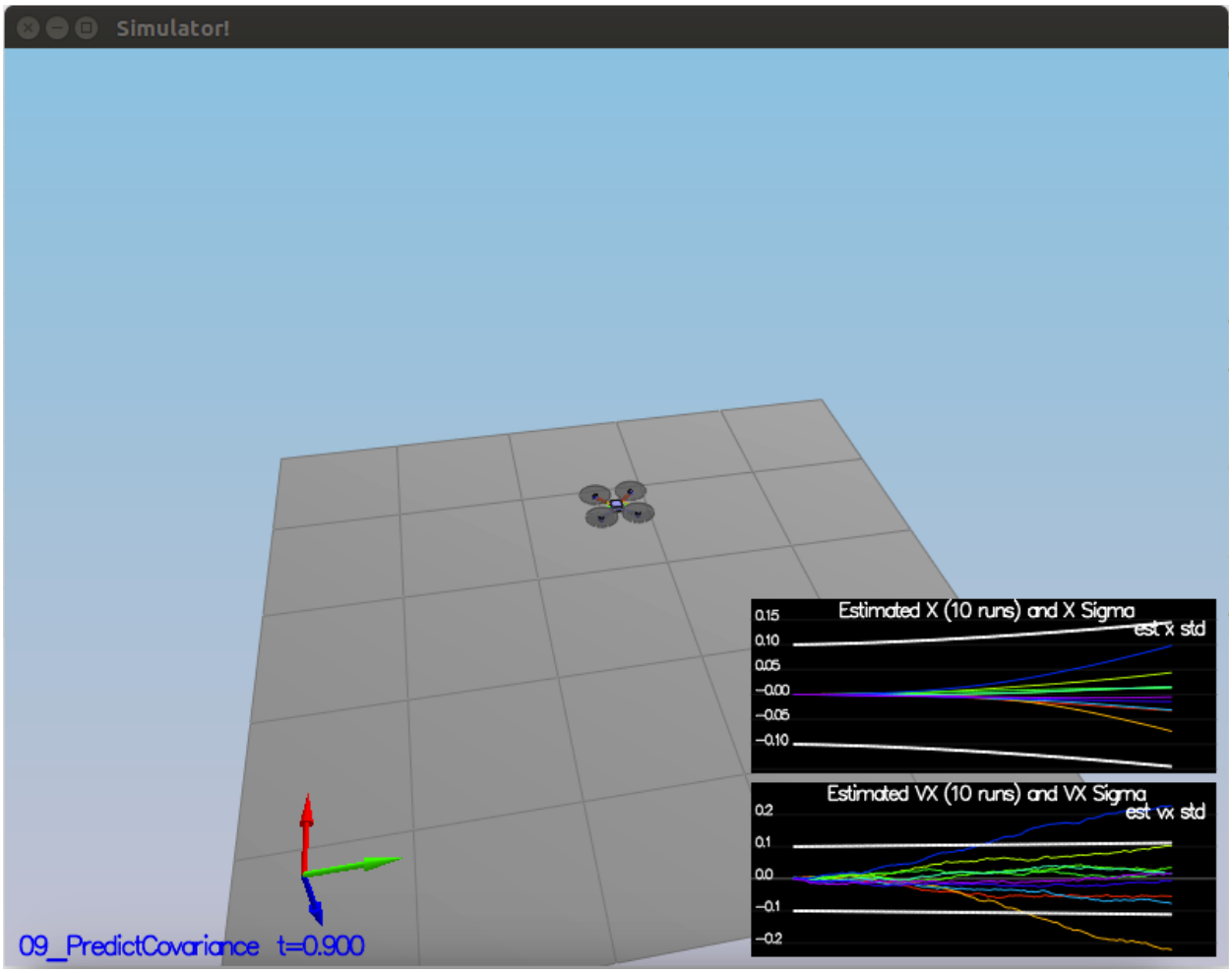
This is done using the equations in section 7.2 of “Estimation for Quadrotors”

The Get_Rbg_Prime is implemented in lines QuadEstimatorEKF.cpp:188..230

1.3.3. Implementing the Predit()

This is implemented in lines QuadEstimatorEKF.cpp:271..279. This is implemented using the algorithm in section 3 in “Estimation for Quadrotors”

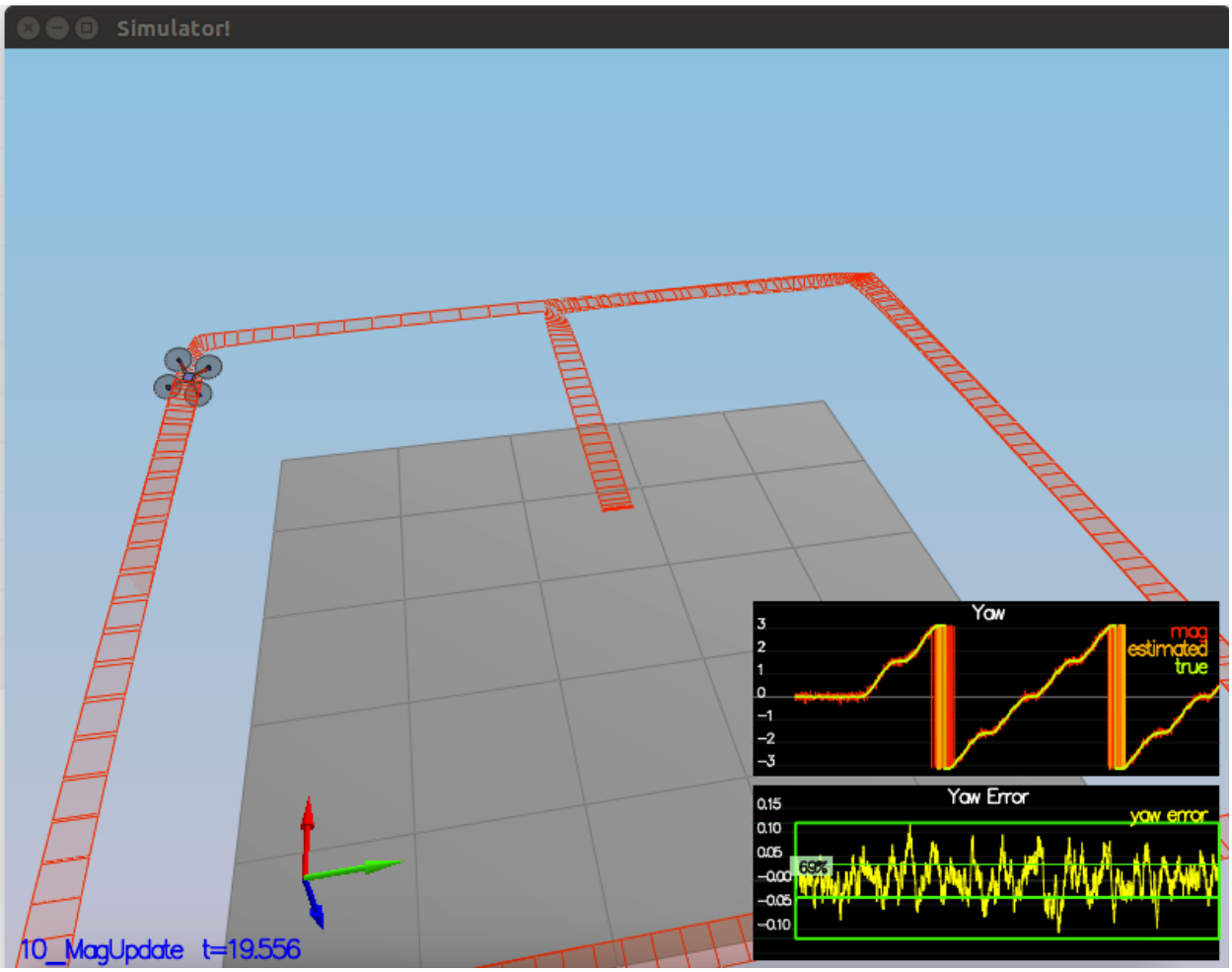
Following the implementation, I ran the scenario 09_PredictCovariance and the following is the observation.



1.4. Magnetometer Update

This is implemented in lines QuadEstimatorEKF.cpp:331..339. the difference is normalized to make sure we are not updating the yaw the long way! Also followed the equations in section 7.3.2 in “Estimation for Quadrotors”.

Once the UpdateFromMag is implemented and ran the scenario, we observe that the yaw error is betweed +/- 0.1 rad



1.5. Closed Loop + GPS Update

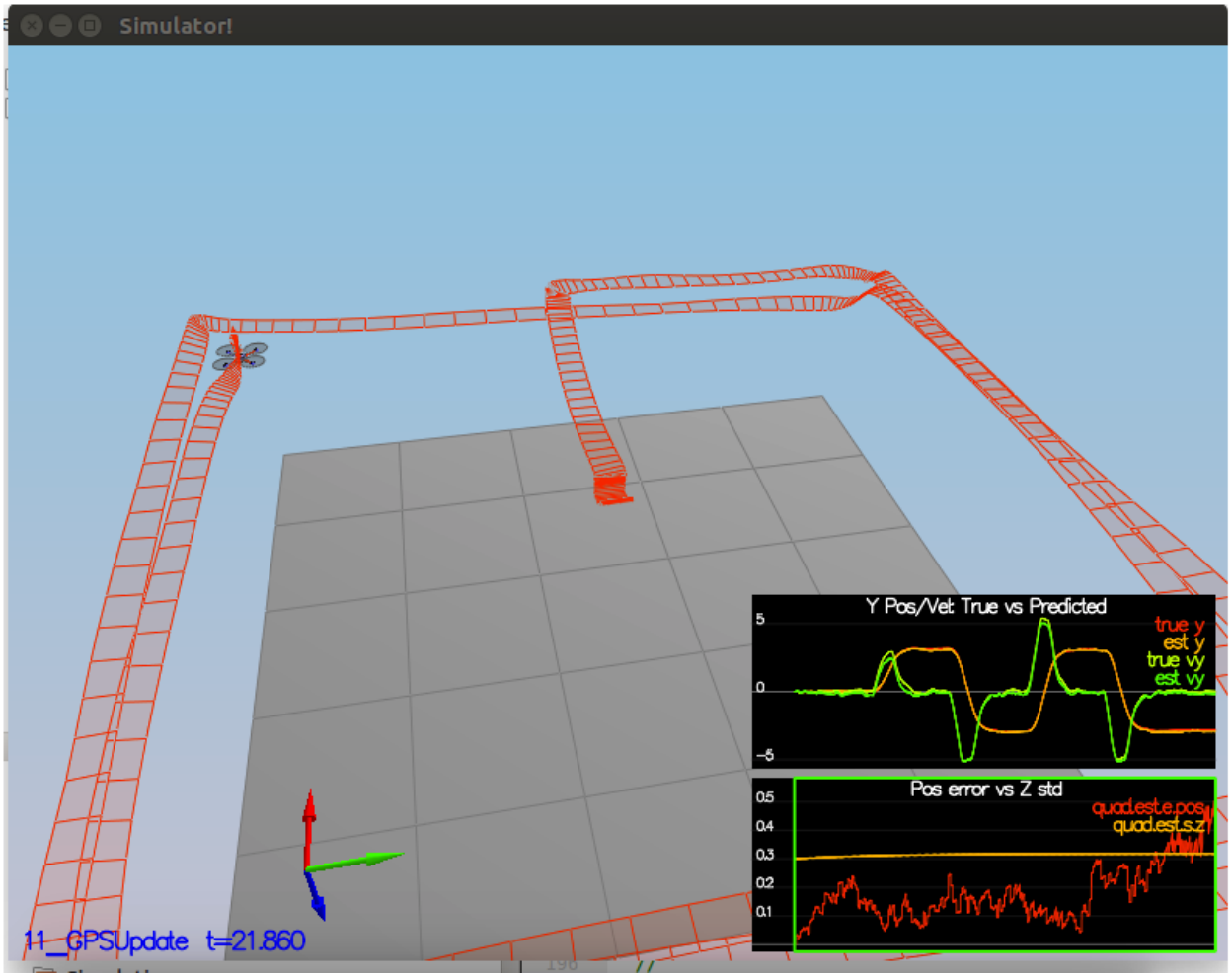
The UpdateFromGPS function is implemented in lines
QuadEstimatorEKF.cpp:306..307

After that the following parameters are tuned as requested.

Quad.UseIdealEstimator to 0

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

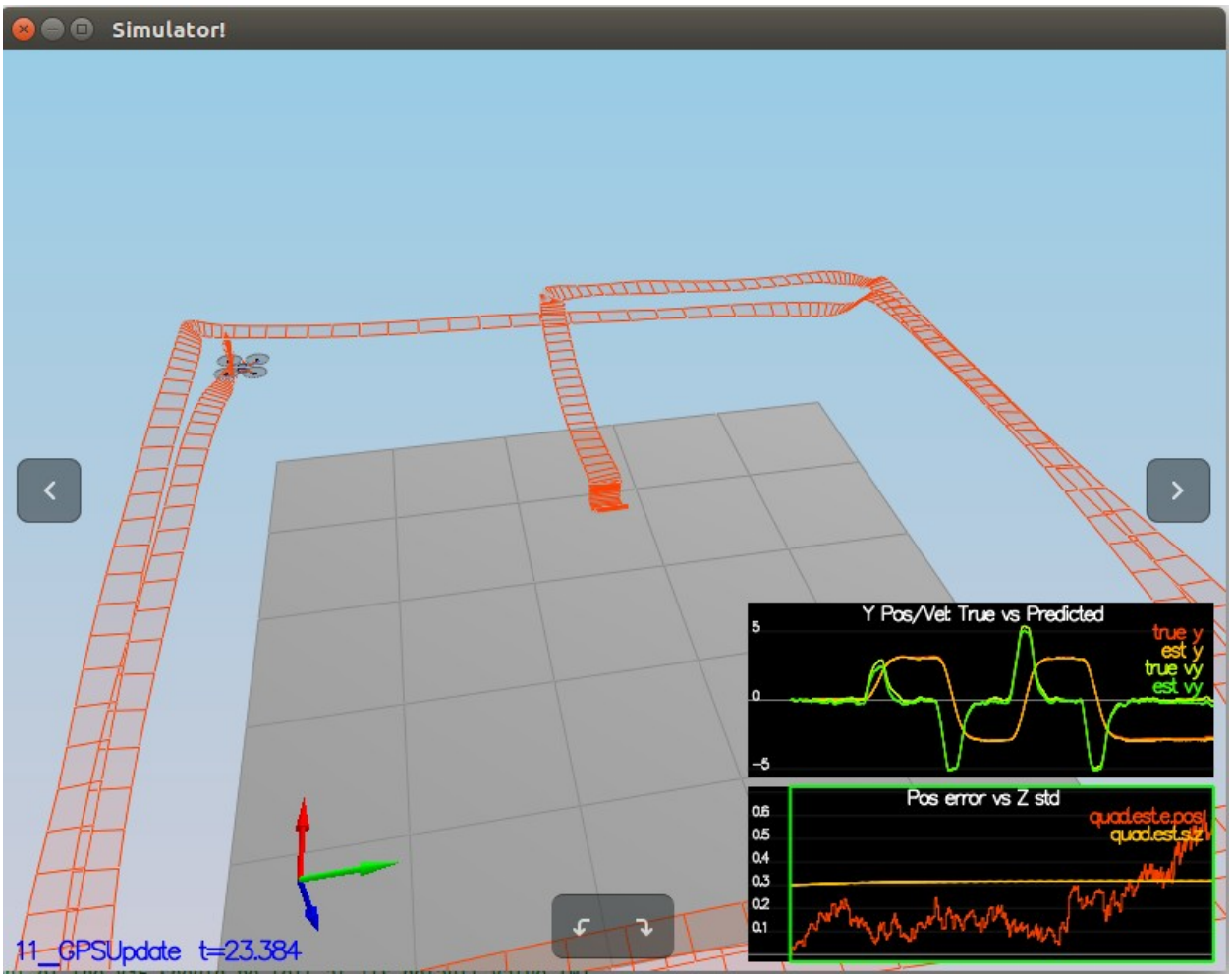
Now when we run the scenario we can see the simulation cycle is completed with a position error of less than 1m. Please see the screen shot as below.



1.6. Adding your controller

Now I have replaced QuadControl.cpp and QuadControlParams.txt from the controls project and ran the 11_GPS_Update again and the behavior is same as the standard one that came with estimation project.

The following is the screen shot.



This shows that the estimation project integrated with controls project is work fine.