Write up - **Building a Controller** - Rubric	
Criteria	Meet Specification

Determine the standard deviation of the measurement noise of both GPS X data and Accelerometer X data.	*The calculated standard deviation should correctly capture ~68% of the sensor measurements. Your writeup should describe the method used for determining the standard deviation given the simulated sensor measurements.*
	
	Used the config/log/Graph1.txt (GPS X data) and config/log/Graph2.txt (Accelerometer X data)
	1) Saved as Graph1.csv and used Excel sheet to find standard deviation
	2) Saved as Graph2.csv and used Excel sheet to find standard deviation
	
	Tuned the 2nd and 3rd decimal places for the following
	`MeasuredStdDev_GPSPosXY = 0.705754972`
	`MeasuredStdDev_AccelXY = 0.500061436`
	
	`To get GPS Stddev 68% and Std Dev 69%`
	`Simulation #199 (../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 69% of the time`

Implement a better rate gyro attitude integration scheme in the `UpdateFromIMU()` function.	*The improved integration scheme should result in an attitude estimator of < 0.1 rad for each of the Euler angles for a duration of at least 3 seconds during the simulation. The integration scheme should use quaternions to improve performance over the current simple integration scheme.* Confident with the following Excerpt `Quaternion<float> gyro_attitude = Quaternion<float>::FromEuler123_RPY(rollEst, pitchEst, ekfState(6));`

	```
	    V3D eulerRPY =
	gyro_attitude.IntegrateBodyRate(gyro,
	dtIMU).ToEulerRPY();


	    float predictedRoll = eulerRPY[0];
	    float predictedPitch = eulerRPY[1];
	    ekfState(6) =  eulerRPY[2];    // yaw

	    // normalize yaw to -pi .. pi
	    if (ekfState(6) > F_PI) ekfState(6) -=
	2.f*F_PI;
	    if (ekfState(6) < -F_PI) ekfState(6) +=
	2.f*F_PI;
	```
	
	The Quarternion Convenience method IntegrateBodyRate helpful to improve performance over
	
	Simulation #206 (../config/07_AttitudeEstimation.txt) PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for at least 3.000000 seconds

Implement all of the elements of the prediction step for the estimator.	*The prediction step should include the state update element (PredictState() function), 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.* 1) Predict State was easy to implement with the Hints provided and **Quartenion.Rotate_BtoI** Is Handy. 2) Rgb prime matrix and gPrime settings requires patience work to place in order. The 7.2 of the Estimation for Quadrotors helped a lot. I made few attempts to get it right. My mentor was very helpful in pointing out the Matrix multiplication mistakes. 3) Predict_State_Image has screen shots of the graph for predict state.

	4) Predict_Covariance.png has screen shots of the graph for predict covariance.
Implement the magnetometer update.	*The update should properly include the magnetometer data into the state. Note that the solution should make sure to correctly measure the angle error between the current state and the magnetometer value (error should be the short way around, not the long way).* 1) Made couple of attempts to get it right with the help of the 7.3.2 of Estimation for Quadrotors. Slack and Mentor inputs helped me go in right direction. 2) Tuned the QYawStd = .09 in the QuadEstimatorEKF.txt to get it right. The output was following. **`Simulation #276 (../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 65% of the time`**

Implement the GPS update.	*The estimator should correctly incorporate the GPS information to update the current state estimate.*
	
	1)7.3.1 GPS section helped but I missed the diagonal set to 1. Again my mentor pointed out.
	
	2) The following conf used . Increased some .5 to get it pass.
	`GPSPosXYStd = 1.5`
	`GPSPosZStd = 3.5`
	`GPSVelXYStd = .2`
	`GPSVelZStd = .4`
	
	`Simulation #284 (../config/11_GPSUpdate.txt)`

	`PASS: ABS(Quad.Est.E.Pos) was less than 1.000000 for at least 20.000000 seconds`
Meet the performance criteria of each step.	*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.* All criteria are passed with the following . Attached the screens shots png file and *csv file in the supporting_files folder .

De-tune your controller to successfully fly the final desired box trajectory with your estimator and realistic sensors.	*The controller developed in the previous project should be de-tuned to successfully meet the performance criteria of the final scenario (<1m error for entire box flight).*
	
	• I don't see any bad behavior from my original control code project.
	
	-QuadController.cpp
	
	-config/QuadControlParams.txt
	
	But I re-tuned 30% less with original and have stable performance for all criteria.
	
	The following are my pos and vel gains the commented are original control project settings.
	
	# Position control gains
	kpPosXY = 21 #30 ,21
	kpPosZ = 14 #20 ,14
	KiPosZ = 24.5 #35 , 24.5
	
	# Velocity control gains
	kpVelXY = 8.41 #12.0 , 8.41
	
	kpVelZ = 5.6 #8.0 , 5.6