Estimation Project

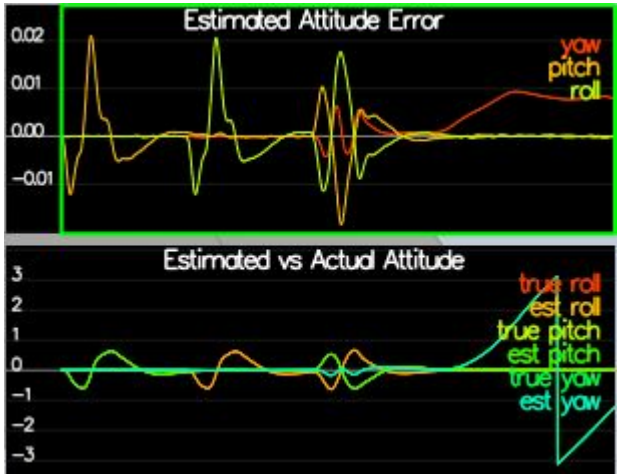
James Tyree 5/29/18

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

I determined the standard deviation by importing the sensor data into this Google Sheet and then applying the appropriate formulas.

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

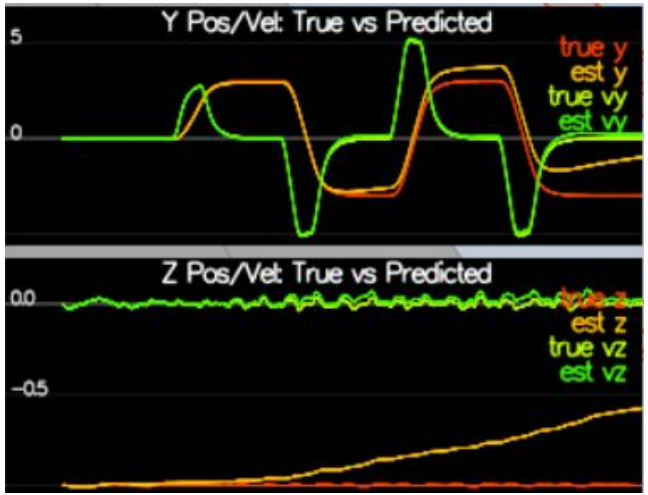
I created a quaternion from the estimated roll, pitch, and yaw rates, and then used the IntegrateBodyRate() method to predict new values. The new values were fed into a complementary filter to combine them with accelerometer data to create the final attitude estimate.




```
Quaternion<float> attitude = Quaternion<float>::FromEuler123_RPY(rollEst, pitchEst, ekfState(6));
V3D bodyRates = V3D(gyro.x, gyro.y, gyro.z);
Quaternion<float> predictedAttitude = attitude.IntegrateBodyRate(bodyRates, dtIMU);

float predictedPitch = predictedAttitude.Pitch();
float predictedRoll = predictedAttitude.Roll();
ekfState(6) = predictedAttitude.Yaw();
```

Implement all of the elements of the prediction step for the estimator: 08_PredictState

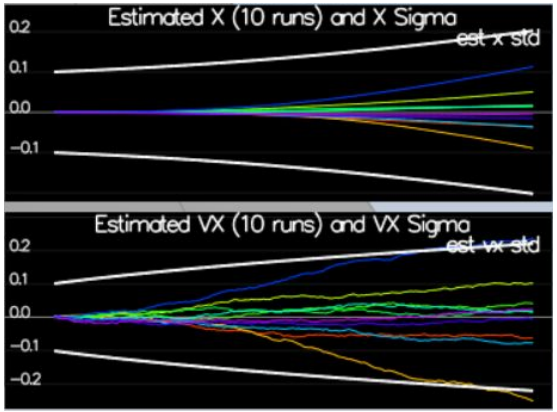
PredictState() rotates the acceleration vector from the body frame to the inertial frame and then updates the state vector using dt to advance the positions and velocities.




```
V3F iAccel = attitude.Rotate_BtoI(accel);
VectorXf trueState(QUAD_EKF_NUM_STATES);
predictedState(0) = curState(0) + curState(3) * dt;
predictedState(1) = curState(1) + curState(4) * dt;
predictedState(2) = curState(2) + curState(5) * dt;
predictedState(3) = curState(3) + iAccel.x * dt;
predictedState(4) = curState(4) + iAccel.y * dt;
predictedState(5) = curState(5) + (iAccel.z - 9.81f) * dt;
predictedState(6) = curState(6);
```

Implement all of the elements of the prediction step for the estimator: 09_PredictCov

Predict() calculates RbgPrime using equations directly from equation 52 of the Estimation for Quadrotors document. The remaining math comes from the PREDICT function in Algorithm 2 of the same document.

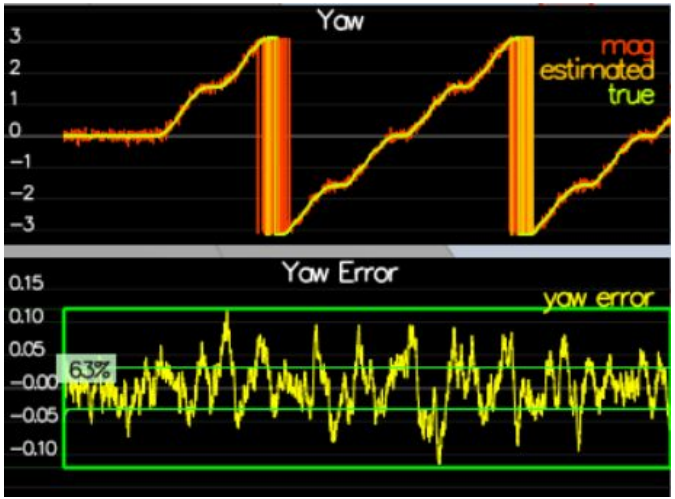



```
gPrime(0, 3) = dt;
gPrime(1, 4) = dt;
gPrime(2, 5) = dt;

gPrime(3, 6) = (RbgPrime(0, 0) * accel.x + RbgPrime(0, 1) * accel.y + RbgPrime(0, 2) * (accel.z - 9.81f)) * dt;
gPrime(4, 6) = (RbgPrime(1, 0) * accel.x + RbgPrime(1, 1) * accel.y + RbgPrime(1, 2) * (accel.z - 9.81f)) * dt;
gPrime(5, 6) = (RbgPrime(2, 0) * accel.x + RbgPrime(2, 1) * accel.y + RbgPrime(2, 2) * (accel.z - 9.81f)) * dt;

MatrixXf gPrimeTranspose(QUAD_EKF_NUM_STATES, QUAD_EKF_NUM_STATES);
MatrixXf sigG(QUAD_EKF_NUM_STATES, QUAD_EKF_NUM_STATES);

gPrimeTranspose = gPrime;
gPrimeTranspose.transposeInPlace();

sigG = gPrime * (ekfCov * gPrimeTranspose) + Q;
ekfCov = sigG;
```

```
// (Vertical Row Down, Horizontal Col Right)

RbgPrime(0, 0) = -cos(pitch) * sin(yaw);
RbgPrime(0, 1) = -sin(roll) * sin(pitch) * sin(yaw) - cos(roll) * cos(yaw);
RbgPrime(0, 2) = -cos(roll) * sin(pitch) * sin(yaw) + sin(roll) * cos(yaw);

RbgPrime(1, 0) = cos(pitch) * cos(yaw);
RbgPrime(1, 1) = sin(roll) * sin(pitch) * cos(yaw) - cos(roll) * sin(yaw);
RbgPrime(1, 2) = cos(roll) * sin(pitch) * cos(yaw) + sin(roll) * sin(yaw);

// Row 2 is all zero already
```

Implement the magnetometer update

At first, I was trying to write my own Update() function, then I saw the call right after the student code!!!
hPrime came directly from equation 58. All that was left was to normalize the difference between measured and estimated yaw.

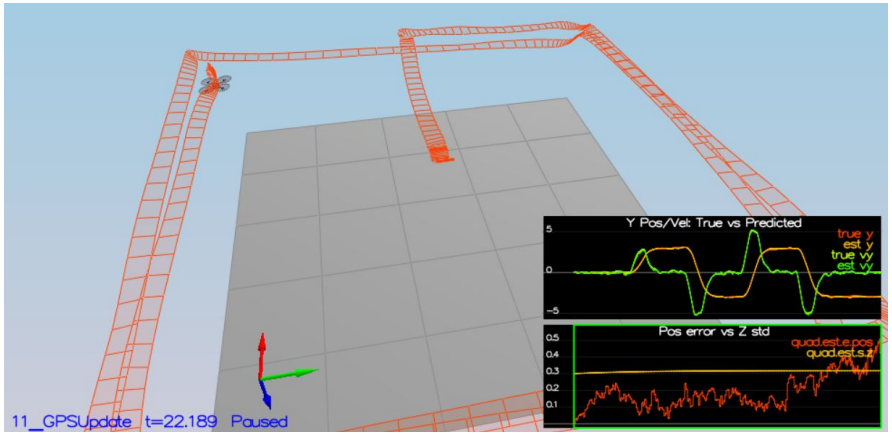



```
// hPrime from Estimation for Quadrotors paper - all others are already zero
hPrime(0, 6) = 1;

// zFromX: measurement prediction based on current state
zFromX(0) = ekfState(6);

// Normalize difference between measured and estimated yaw
float deltaZ;
deltaZ = z(0) - zFromX(0);
if (deltaZ > F_PI) z(0) -= 2.f*F_PI;
if (deltaZ < -F_PI) z(0) += 2.f*F_PI;
```

Implement the GPS update

Once again, Update() was already written and could be re-used. hPrime came directly from equation 55. hPrime, the measurements, the estimates, and the covariance matrix had to be passed to Update().


```
hPrime(0, 0) = 1;
hPrime(1, 1) = 1;
hPrime(2, 2) = 1;
hPrime(3, 3) = 1;
hPrime(4, 4) = 1;
hPrime(5, 5) = 1;

zFromX(0) = ekfState(0);
zFromX(1) = ekfState(1);
zFromX(2) = ekfState(2);
zFromX(3) = ekfState(3);
zFromX(4) = ekfState(4);
zFromX(5) = ekfState(5);
```



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

Here is the final trajectory running my controls code with my detuned constants. XY, and Z position control gains had to be substantially reduced from the values in the previous project that were tuned for perfect sensors. Velocity, angle, and angle rate gains did not need to be adjusted.