Project Write up: Estimation

Talha Khalid

10/8/18

1. Determining Standard Deviation of Measurement Noise

The standard deviation of the measurement noise was determined by running the simulation for approximately 10 seconds then importing the recorded data into an excel spreadsheet and using the built-in standard deviation function to compute the standard deviation.

2. Integrating Improved Rate Gyro

The Rate Gyro was improved by first establishing the rotation matrix (as shown in lessons) and used to perform the Euler integration with the current gyro readings. The predicted pitch, roll and yaw were determined by multiplying the respective Euler integrated components by the time and adding it to the estimated roll, pitch, yaw.

```
float theta = rollEst;
float phi = pitchEst;

Mat3x3F rot_mat = Mat3x3F();
rot_mat(0,0)=1;
rot_mat(0,1)=sin(phi)*tan(theta);
rot_mat(0,2)=cos(phi)*tan(theta);
rot_mat(1,0)=0;
rot_mat(1,1)=cos(phi);
rot_mat(1,2)=-sin(theta);
rot_mat(2,0)=0;
rot_mat(2,1)=sin(phi)*cos(theta);
rot_mat(2,2)=cos(theta)*cos(theta);

V3F euler_int = rot_mat * gyro;

float predictedPitch = pitchEst + dtIMU * euler_int.y;
float predictedRoll = rollEst + dtIMU * euler_int.x;
ekfState(6) = ekfState(6) + dtIMU * euler_int.z;
```

3. Implementing Prediction Step

The predicted state vector was determined by settting the components accordingly. The first 3 position components are set based off the current position + the change in velocity over the given time. The second set of the 3 velocity components were determined based off the velocity as well as integrating the inertial acceleration and adding it to the current velocity. The inertial acceleration was determined by converting the provided attitude quaternion from the body frame to the inertial frame via the use of the Rotate_BtoI() function.

```
Quaternion<float> attitude = Quaternion<float>::FromEuler123_RPY(rollEst, pitchEst, curState(6));
  ////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
predictedState(0) = curState(0)+curState(3)*dt;
predictedState(1) = curState(1)+curState(4)*dt;
predictedState(2) = curState(2)+curState(5)*dt;

V3F intertial_accel = attitude.Rotate_BtoI(accel);

predictedState(3) = curState(3) + intertial_accel.x*dt;
predictedState(4) = curState(4) + intertial_accel.y*dt;
predictedState(5) = curState(5) + intertial_accel.z*dt;

  /////////////////////////////// END STUDENT CODE ////////////////////////////

  return predictedState;
```

3a. Implementing Rbg Prime

The RBG rotation matrix was determined by simply ensuring the equation presented in the provided research paper was correctly implanted in code.

```
////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
RbgPrime(0,0) = -(cos(roll)*sin(yaw));
RbgPrime(0,1) = -(sin(pitch)*sin(roll)*sin(yaw))-cos(pitch)*cos(yaw);
RbgPrime(0,2) = -(cos(pitch)*sin(roll)*sin(yaw))+sin(pitch)*sin(yaw);
RbgPrime(1,0) = cos(roll)*cos(yaw);
RbgPrime(1,1) = sin(pitch)*sin(roll)*cos(yaw)-cos(pitch)*sin(yaw);
RbgPrime(1,2) = cos(pitch)*sin(roll)*cos(yaw)+sin(pitch)*sin(yaw);
RbgPrime(2,0) = 0;
RbgPrime(2,1) = 0;
RbgPrime(2,2) = 0;
```

3b. Implementing g prime

g prime was implemented based off the equation 51 presented in the research paper. Once g prime was set covariance of the extended Kalman filter was set based off step 4 of the predict function.

```
    gPrime(0,3) = dt;
    gPrime(1,4) = dt;
    gPrime(2,5) = dt;
    gPrime(3,6) = (RbgPrime(0)*accel).sum()*dt;
    gPrime(4,6) = (RbgPrime(1)*accel).sum()*dt;
    gPrime(5,6) = (RbgPrime(2)*accel).sum()*dt;

    ekfCov = gPrime*ekfCov*gPrime.transpose()+Q;

```

4. Implementing GPS

The measurement prediction from the current state (zFromX) was set based off the current ekf state. The h prime matrix was determined off of equation 55 from the paper.

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

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

```

5. Implementing Magnetometer

The magnetometer was implemented by first computing the difference between measured and estimated yaw. The result was then normalized between +/- 2 times PI.

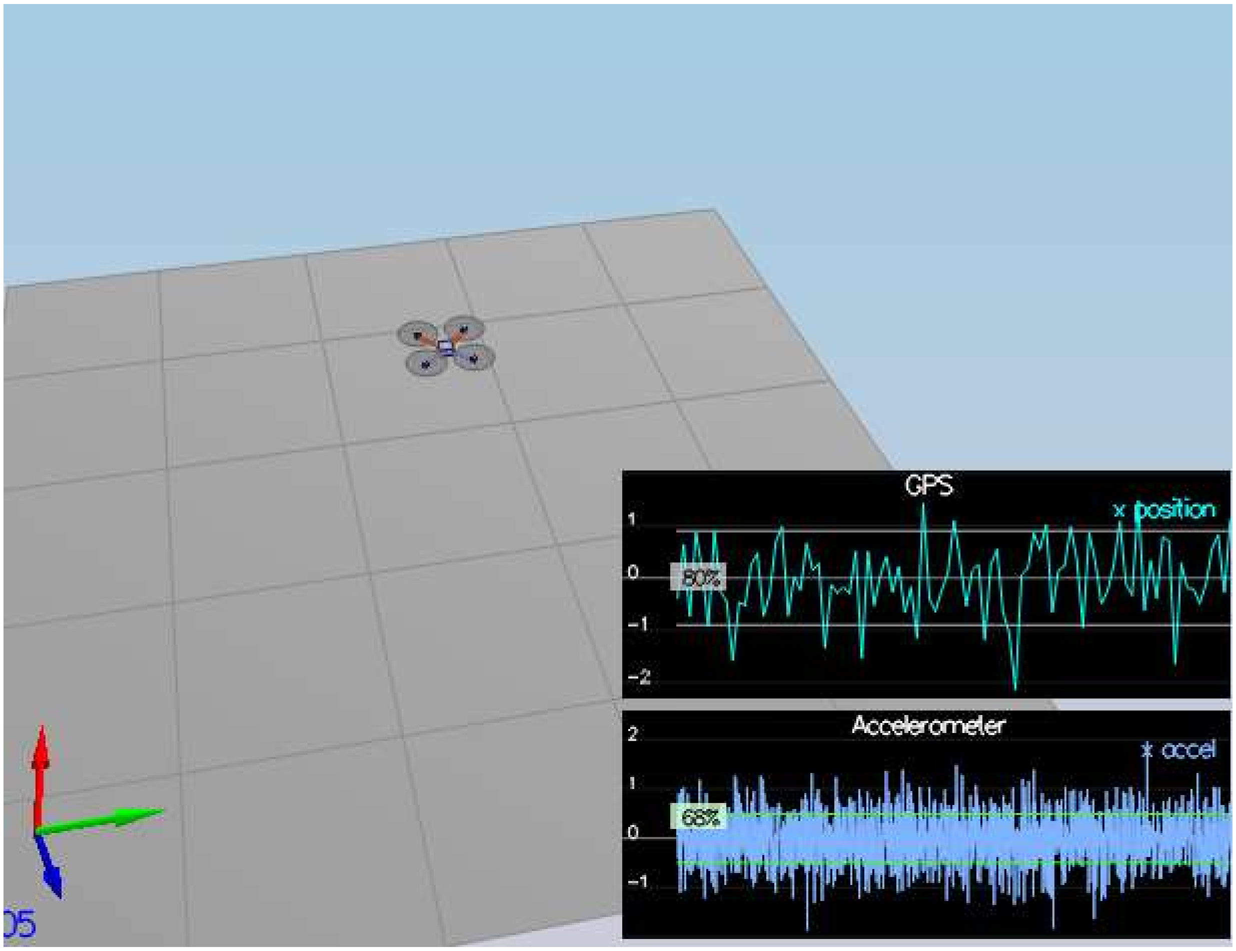
```
    zFromX(0) = ekfState(6);
    float diff = magYaw-zFromX(0);
    if ( diff > F_PI ) {
      zFromX(0) += 2.f*F_PI;
    } else if ( diff < -F_PI ) {
      zFromX(0) -= 2.f*F_PI;
    };

    hPrime(0, 6) = 1;
```

Scenario Results

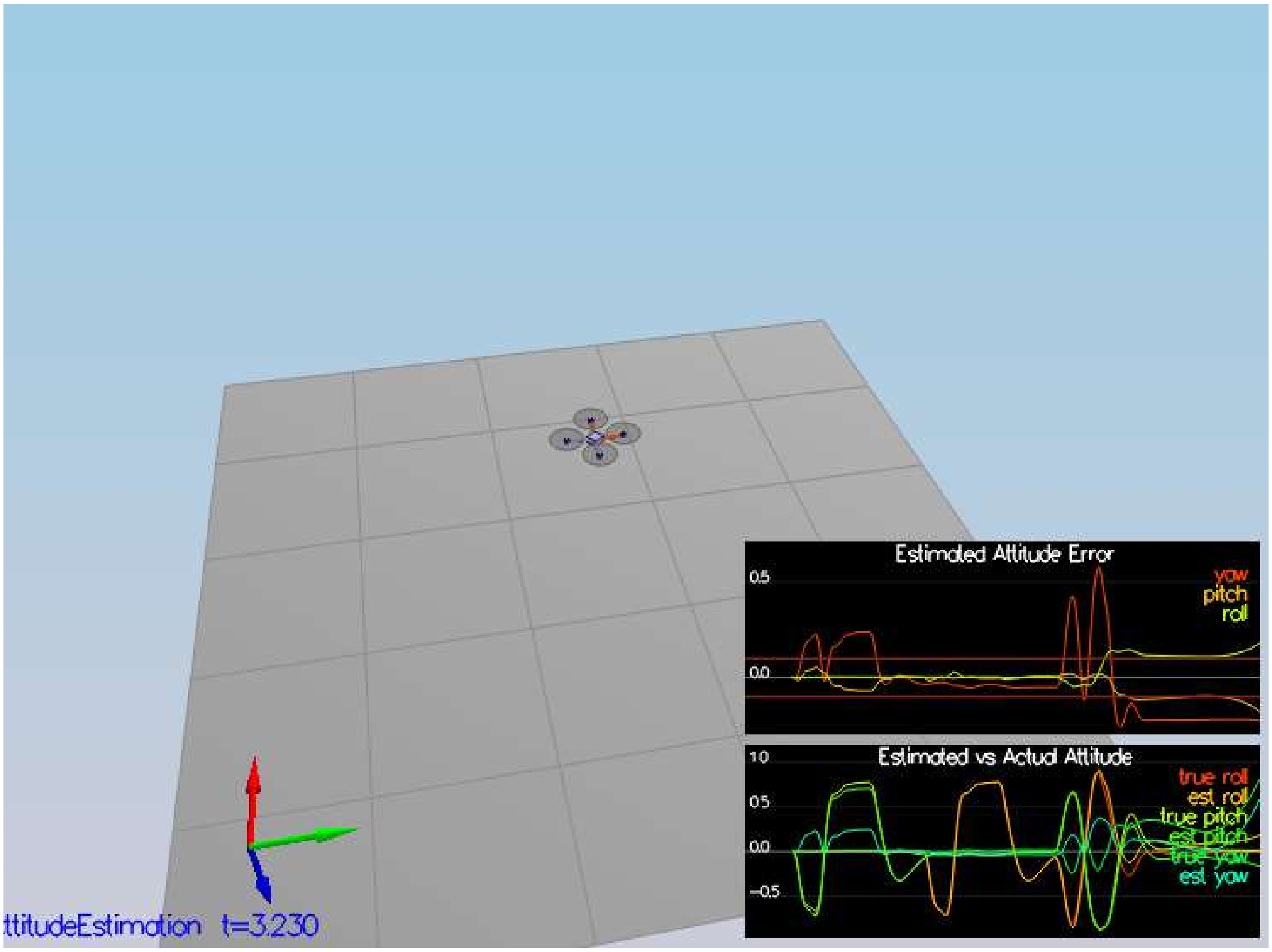
1. **Scenario 6**

   Scenario 6 was successful



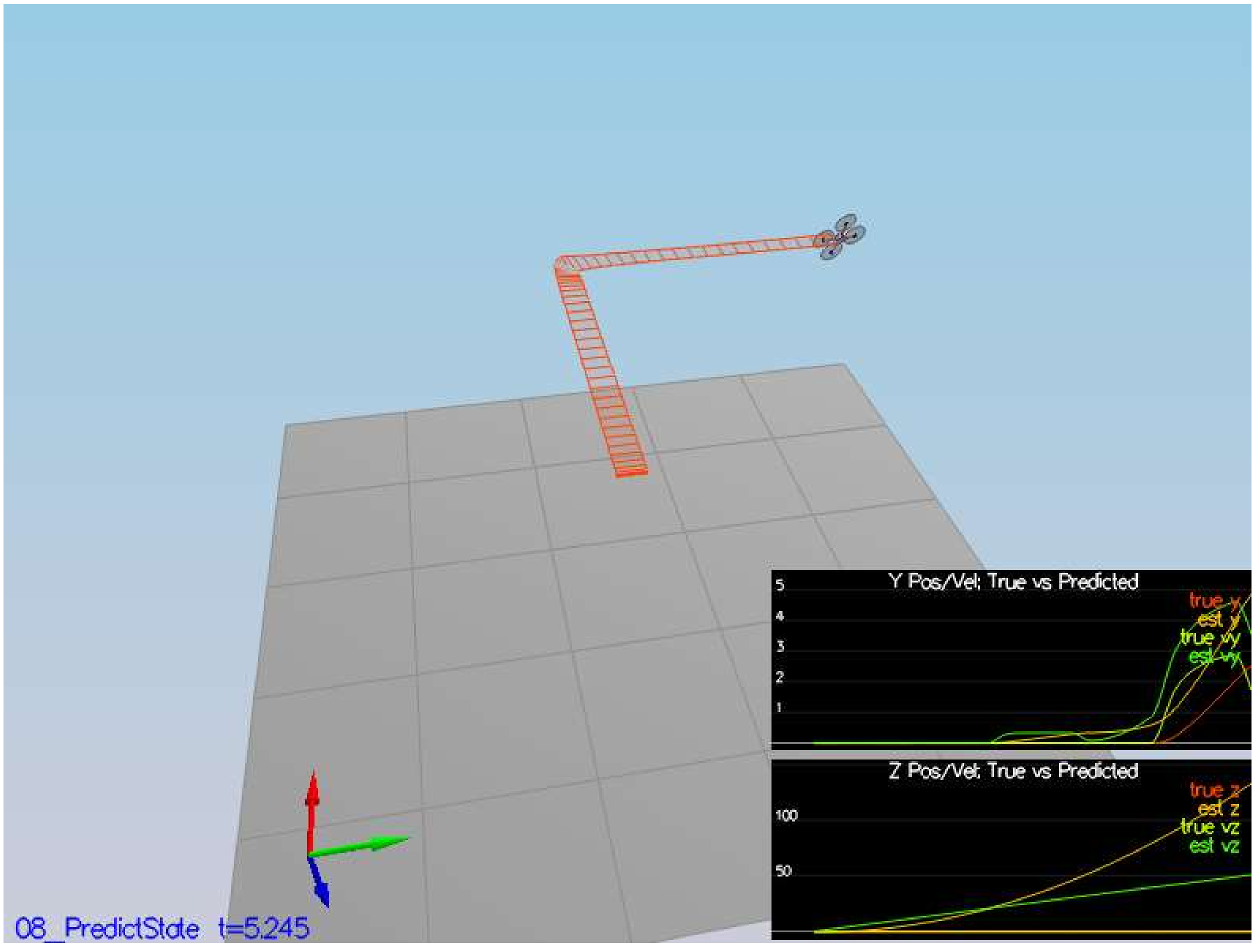
2. **Scenario 7**

Scenario 7 is also working



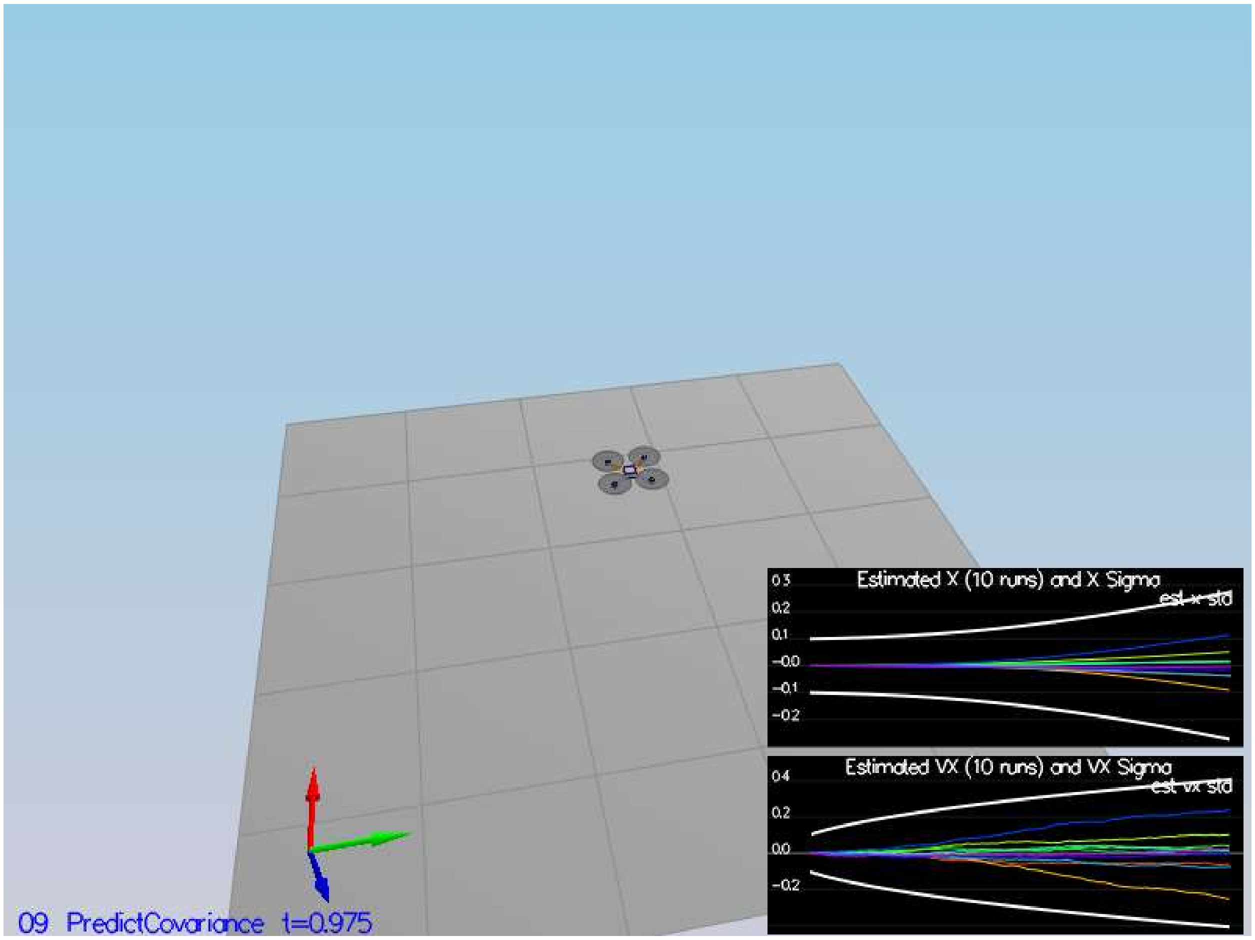
3. **Scenario 8**

Scenario 8 is also fulfilled.

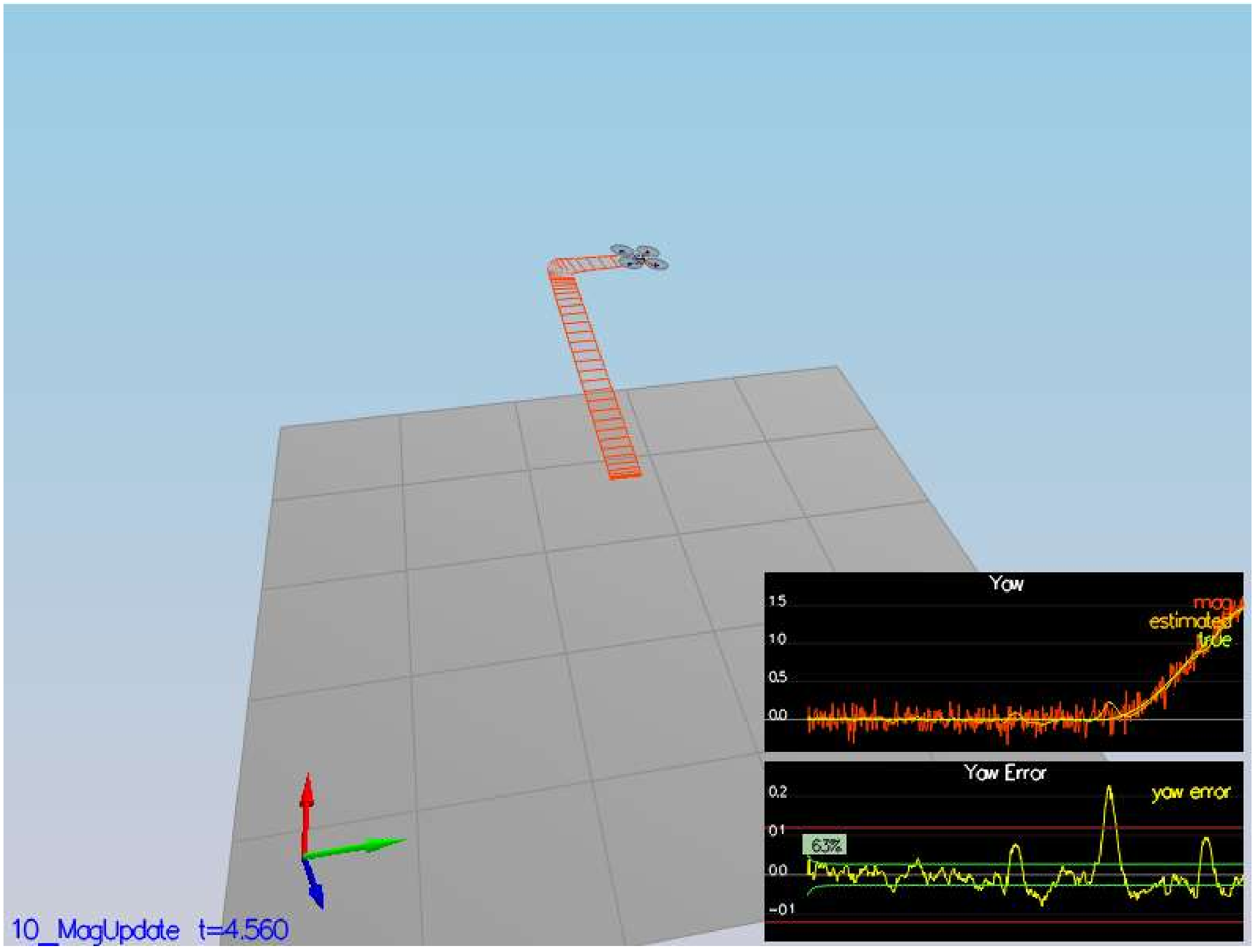


4. Scenario 9

Scenario 9 compares to the image provided in the read me file.



5. Scenario 10
   Scenario 10 is also working



6. Scenario 11

Scenario 11 has problems running after changing the controls of the quadrotor.