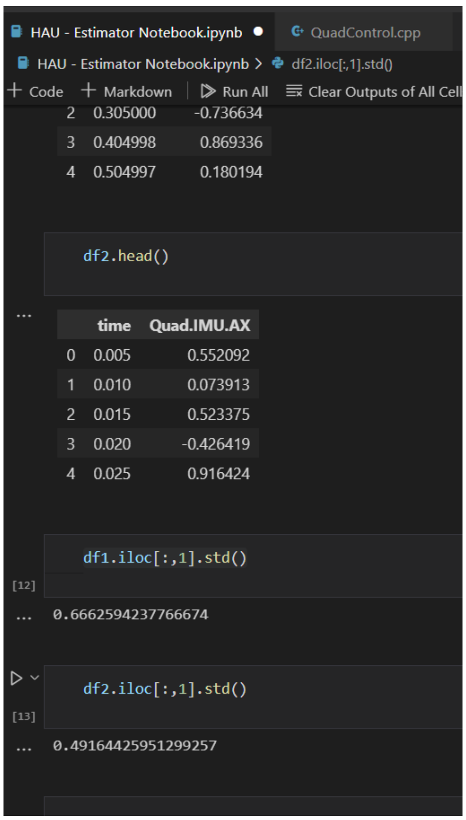
Udacity - Flying Cars and Autonomous Flight Engineer Nanodegree
Estimation Project Write Up
Yu Hin Hau
12/17/2021

In this project, I implemented an Extended Kalman Filter to estimate the drone's state variable. The EKF work by estimating the state via an internal physics model. It is then updated by measurements from the GPS and magnetometer whenever the readings come in one at a time. The drone's attitude is determined by an non-linear complementary filter.

1) Determine Standard Deviation of the Measurement Noise Data

I calculated the standard deviation by starting a jupyter notebook, loading the data into Pandas and performed an analysis on the GPS and accelerometer X data.



2) Implement a better attitude estimation scheme

To improve attitude estimation, I implemented a non-linear complemetary filter via quanterion.

```cpp

Quaternion<float> q = Quaternion<float>().FromEuler123_RPY(this->rollEst, this->pitchEst, this->ekfState(6));
Quaternion<float> q_bar = q.IntegrateBodyRate(gyro, this->dtIMU);

float roll_gyro = q_bar.Roll();
float pitch_gyro = q_bar.Pitch();
float yaw_gyro = q_bar.Yaw();

//cout << "Gyro: " << (float) roll_gyro << "\tAccel: " << (float) accelRoll << endl;

float predictedPitch = (this->attitudeTau / (this->attitudeTau + this->dtIMU)) * pitch_gyro +(this->dtIMU / (this->attitudeTau + this->dtIMU)) * accelRol
float predictedRoll = (this->attitudeTau / (this->attitudeTau + this->dtIMU)) * roll_gyro +(this->dtIMU / (this->attitudeTau + this->dtIMU)) * accelPitch

ekfState(6) = yaw_gyro;


// SMALL ANGLE GYRO INTEGRATION:
// (replace the code below)
// make sure you comment it out when you add your own code -- otherwise e.g. you might integrate yaw twice


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

3) Implement StatePredict, RbgPrime matrix and Covariance State Update

I implemented the equation as noted in the technical paper included in the course.

```cpp
V3F accel_xyz = attitude.Rotate_BtoI(accel);
V3F gyro_xyz = attitude.Rotate_BtoI(gyro);


predictedState(3) = curState(3) + accel_xyz.x * dt; // x_dot
predictedState(4) = curState(4) + accel_xyz.y * dt; // y_dot
predictedState(5) = ( curState(5) - 9.81 * dt )  + accel_xyz.z * dt;  // z_dot

predictedState(0) = curState(0) + predictedState(3) * dt; // x
predictedState(1) = curState(1) + predictedState(4) * dt; // y
predictedState(2) = curState(2) + predictedState(5) * dt; // z

predictedState(6) - curState(6);



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

return predictedState;
}
```

```
///////////////////////////// BEGIN STUDENT CODE ///////////////////////////
float rc = cos(roll);
float rs = sin(roll);
float pc = cos(pitch);
float ps = sin(pitch);
float yc = cos(yaw);
float ys = sin(yaw);

//Rbg(0, 0) = rc * yc;
//Rbg(1, 0) = rc * ys;
//Rbg(2, 0) = -rs;

//Rbg(0, 1) = ps * rs * yc - pc * ys;
//Rbg(1, 1) = ps * rs * ys + pc * yc;
//Rbg(2, 1) = rc * ps;

//Rbg(0, 2) = pc * rs * yc + ps * ys;
//Rbg(1, 2) = pc * rs * ys - ps * yc;
//Rbg(2, 2) = rc * pc;

RbgPrime(0, 0) = -rc * ys;
RbgPrime(1, 0) = rc * yc;
RbgPrime(2, 0) = 0;

RbgPrime(0, 1) = - ps * rs * ys - pc * yc;
RbgPrime(1, 1) = ps * rs * yc - pc * ys;
RbgPrime(2, 1) = 0;

RbgPrime(0, 2) = -pc * rs * ys + ps * yc;
RbgPrime(1, 2) = pc * rs * yc + ps * ys;
RbgPrime(2, 2) = 0;

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

```
// Skip the Yaw, as it is not needed for the covariance calculation
//VectorXf u(4);
VectorXf u(3);
u(0) = accel.x;
u(1) = accel.y;
u(2) = accel.z;
//u(3) = gyro.z;


// Build gPrime Matrix (g is done at state update)
gPrime(0, 3) = dt;
gPrime(1, 4) = dt;
gPrime(2, 5) = dt;

VectorXf rbgPrime_R1(3);
rbgPrime_R1(0) = RbgPrime(0, 0);
rbgPrime_R1(1) = RbgPrime(0, 1);
rbgPrime_R1(2) = RbgPrime(0, 2);
gPrime(3, 6) = rbgPrime_R1.dot(u) * dt;

VectorXf rbgPrime_R2(3);
rbgPrime_R2(0) = RbgPrime(1, 0);
rbgPrime_R2(1) = RbgPrime(1, 1);
rbgPrime_R2(2) = RbgPrime(1, 2);
gPrime(4, 6) = rbgPrime_R2.dot(u) * dt;

VectorXf rbgPrime_R3(3);
rbgPrime_R3(0) = RbgPrime(2, 0);
rbgPrime_R3(1) = RbgPrime(2, 1);
rbgPrime_R3(2) = RbgPrime(2, 2);
gPrime(5, 6) = rbgPrime_R3.dot(u) * dt;

// Update covariance matrix

this->ekfCov = gPrime * this->ekfCov * gPrime.transpose() + this->Q;

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

3) Implement Magnetometer

I implemented the magnetometer and use it to update yaw via the EKF.

```cpp

// MAGNETOMETER UPDATE
// Hints:
//  - Your current estimated yaw can be found in the state vector: ekfState(6)
//  - Make sure to normalize the difference between your measured and estimated yaw
//    (you don't want to update your yaw the long way around the circle)
//  - The magnetomer measurement covariance is available in member variable R_Mag
////////////////////////////// BEGIN STUDENT CODE ///////////////////////////


hPrime(0, 6) = 1;
zFromX(0) = this->ekfState(6);

if (z(0) - zFromX(0) > F_PI)
    z(0) -= 2. * F_PI;
else if (z(0) - zFromX(0) < - F_PI)
    z(0) += 2. * F_PI;


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

Update(z, hPrime, R_Mag, zFromX);
```

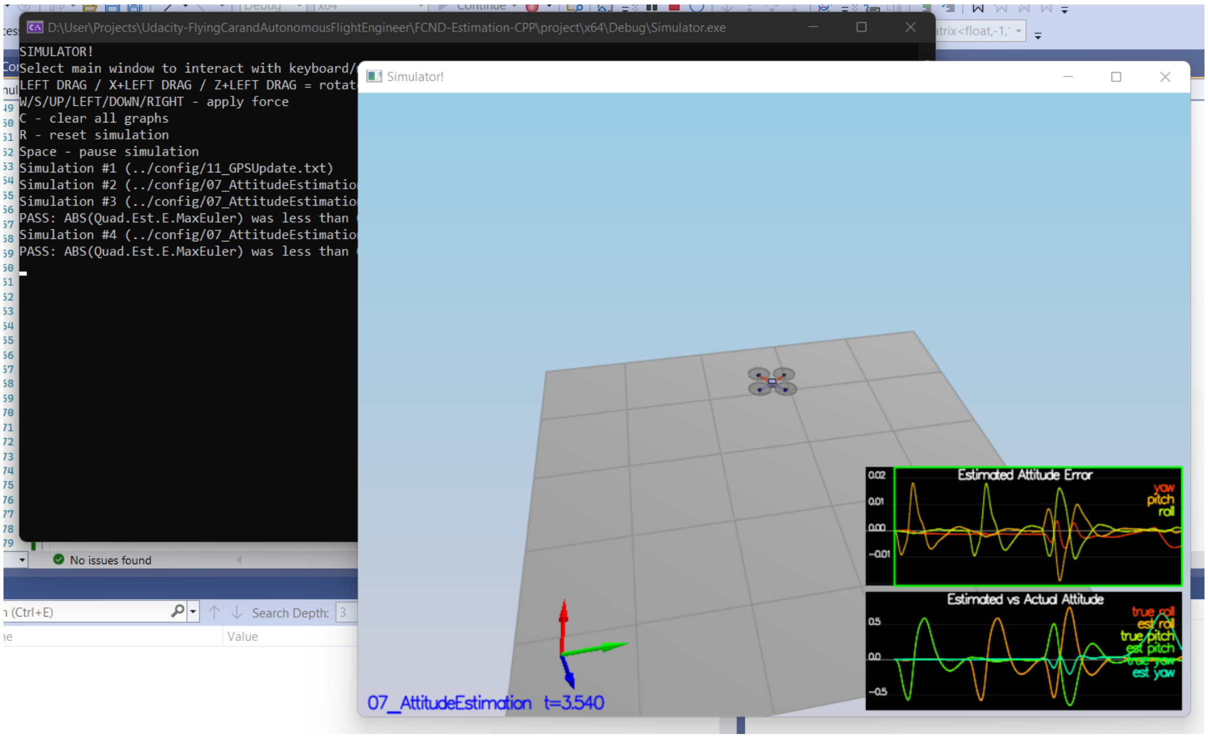
4) Implement GPS

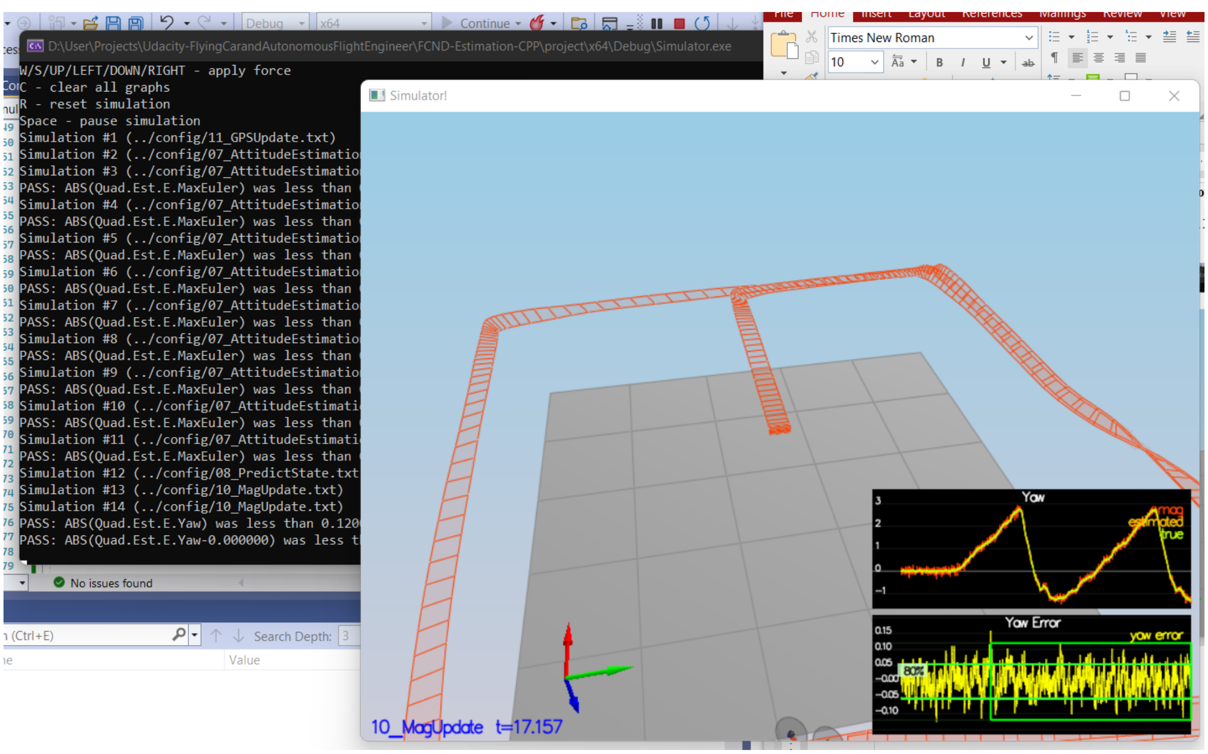
I used a for loop to update all the states from the GPS to enhance the EKF's internal model.

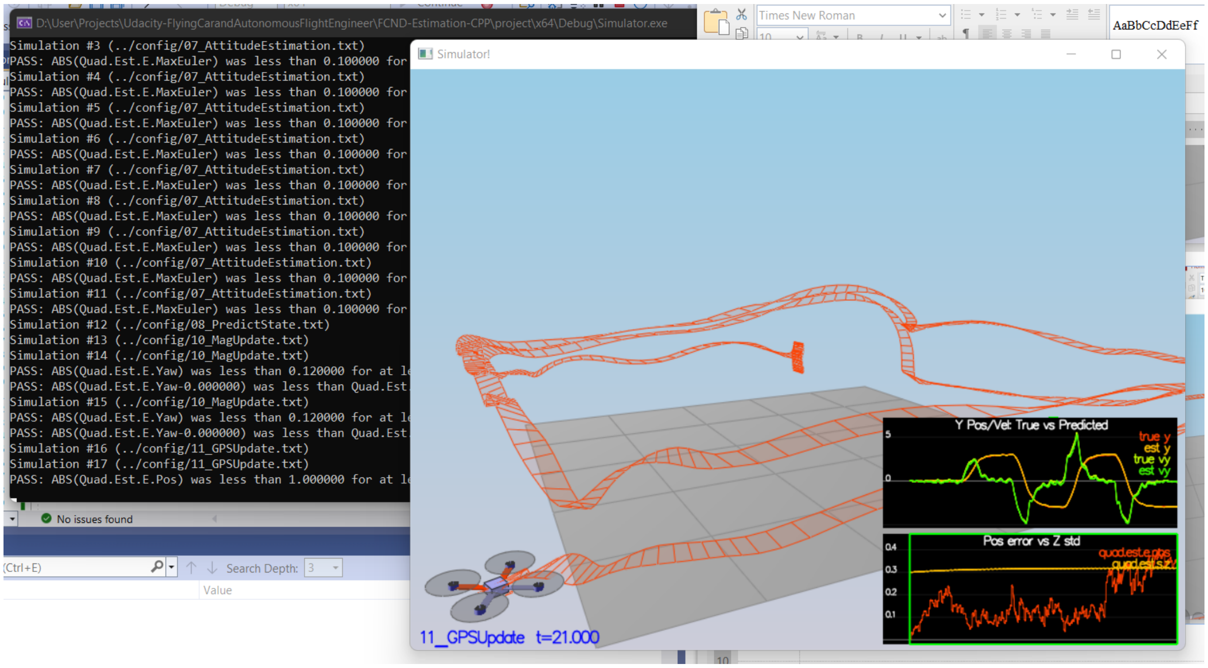
```cpp
}

void QuadEstimatorEKF::UpdateFromGPS(V3F pos, V3F vel)
{
  VectorXf z(6), zFromX(6);
  z(0) = pos.x;
  z(1) = pos.y;
  z(2) = pos.z;
  z(3) = vel.x;
  z(4) = vel.y;
  z(5) = vel.z;

  MatrixXf hPrime(6, QUAD_EKF_NUM_STATES);
  hPrime.setZero();

  // GPS UPDATE
  // Hints:
  //  - The GPS measurement covariance is available in member variable R_GPS
  //  - this is a very simple update
  ////////////////////////////// BEGIN STUDENT CODE ///////////////////////////

  for (int i = 0; i < 6; i++)
  {
      hPrime(i, i) = 1;
      zFromX(i) = this->ekfState(i);
  }


```

5 + 6) Meet all critera in Simulator and Retune PID Controller

I retuned my PID controller, decrease translational gain in XY direction and increase Z direction responsiviness. And it finally is able to follow the path without crashing.







D:\User\Projects\Udacity-FlyingCarandAutonomousFlightEngineer\FCND-Estimation-CPP\project\x64\Debug\Simulator.exe
Simulation #3 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for
Simulation #4 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for
Simulation #5 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for
Simulation #6 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for
Simulation #7 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for
Simulation #8 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for
Simulation #9 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for
Simulation #10 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for
Simulation #11 (../config/07_AttitudeEstimation.txt)
PASS: ABS(Quad.Est.E.MaxEuler) was less than 0.100000 for
Simulation #12 (../config/08_PredictState.txt)
Simulation #13 (../config/10_MagUpdate.txt)
Simulation #14 (../config/10_MagUpdate.txt)
PASS: ABS(Quad.Est.E.Yaw) was less than 0.120000 for at le
PASS: ABS(Quad.Est.E.Yaw-0.000000) was less than Quad.Est.
Simulation #15 (../config/10_MagUpdate.txt)
PASS: ABS(Quad.Est.E.Yaw) was less than 0.120000 for at le
PASS: ABS(Quad.Est.E.Yaw-0.000000) was less than Quad.Est.
Simulation #16 (../config/11_GPSUpdate.txt)
Simulation #17 (../config/11_GPSUpdate.txt)
PASS: ABS(Quad.Est.E.Pos) was less than 1.000000 for at le
No issues found
(Ctrl+E)
Search Depth:
Value
Times New Roman
AaBbCcDdEeFf
Simulator!
Y Pos/Vel: True vs Predicted
true y
est y
true vy
est vy
Pos error vs Z std
11_GPSUpdate t=21.000