Project: Estimation for Quadrotor

Irving Vasquez

May 29, 2018

1 Introduction

In this project I have implemented a state estimation for a quadrotor with a Extended Kalman Filter. Next I will how I have address each query.

2 Implementation

2.1 Standard deviation

I just simply imported the data into openoffice Calc and I have used the Standard deviation formula. The computed values are the following:

```
MeasuredStdDev_GPSPosXY = 0.68
MeasuredStdDev_AccelXY = .48
```

2.2 Attitude Estimation

In this assignment, I have computed the next attitude using the quaternion integration provided in the quaternion class. Next, I have updated the predicted state. Here is the code:

```
//create a quaternion and assing the current attitude belief
Quaternion<float> q_t =
Quaternion<float>::FromEuler123_RPY(rollEst, pitchEst, ekfState(6));

float predictedPitch, predictedRoll;

// Integrate body rates
q_t.IntegrateBodyRate(gyro, dtIMU);

// go back to Euler
predictedPitch = q_t.Pitch();
predictedRoll =  q_t.Roll();
ekfState(6) = q_t.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;
```

2.3 Prediction Step

In this section, I have implemented the transition function in order to compute the next state. I used equation 49 from [1]. To implemented it, I have used two extra variables A and B. A provides the linear part, while B provides the nonlinear advance of the state. Here is the code:

```
VectorXf A(7);
A[0] = curState[0] + curState[3] * dt;
A[1] = curState[1] + curState[4] * dt;
A[2] = curState[2] + curState[5] * dt ;
A[3] = curState[3];
A[4] = curState[4];
A[5] = curState[5] - CONST_GRAVITY * dt;
A[6] = curState[6];

V3F s_dot_dot = attitude.Rotate_BtoI(accel);

VectorXf B(7);
B[0] = 0.0;
B[1] = 0.0;
B[2] = 0.0;
B[3] = s_dot_dot.x * dt;
B[4] = s_dot_dot.y * dt;
B[5] = s_dot_dot.z * dt;
B[6] = 0.0; // avoids double integration

predictedState = A + B;
```

2.4 Magnetometer Update

In order to complete the update function, I just copied the corresponding values and checked the correct range. See the code here:

```
hPrime(0,6) = 1;
zFromX[0] = ekfState(6);

float reading = magYaw;
float current = ekfState(6);

if (reading < 0){
if (current > 0)
```

```
{
if (reading<-M_PI/2 && current> M_PI/2)
reading = 2*M_PI + reading;
}
}
else
{
if (current < 0)
if (current < -M_PI/2 && reading> M_PI/2)
{
reading =  reading -2*M_PI;
}
}

z(0) = reading;
```

2.5 Closed Loop + GPS Update

I have modified the requested variables. Then, I implemented the `UpdateFromGPS()` function. I just have copied the correct values to the adequate variables. See the code:

```
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;
```

2.6 Adding Your Controller

I have copied from my previous project the dictated files. It is worth to say that my previous control parameters were enough to achieve the desired accuracy. See a running at: `https://youtu.be/ilmUDKkePnY`

References

[1] S. Tellex, A. Brown, S. Lupashin, *Estimation for Quadrotors*, Udacity, 2018.