Building an Estimator

README

I. Adding sensor noise

The goal here is to measure the standard deviation of the GPS measurement and the Accelerometer measurement which are Gaussian. Samples of data are collected from each of these distributions.

For the GPS, it is recorded in Graph1.txt. To measure the standard deviation, I followed the steps below:

- I took the first 10 samples;
- I computed the mean using the formula $\hat{x} = \frac{1}{10}\sum_{i=1}^{10} x_i$;
- Having the mean, I computed the variance using

$\hat{\sigma}^2 = \frac{1}{10}\sum_{i=1}^{10}(x_i - \hat{x})$;

- The standard deviation $std = \sqrt{\hat{\sigma}^2} = 0.6521$

I followed the same process for the Accelerometer ($std = \sqrt{\hat{\sigma}^2} = 0.576$).

Having the two standard deviations, I tuned them slightly to meet the ~68% requirement.

II. Implemented Estimator

➢ Attitude Estimation

It is responsible for improving the complementary filter using the measurements from the accelerometer end the gyro.

```
////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
// 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 r = rollEst, p = pitchEst;
float v[9] = { 1, sin(r)*tan(p), cos(r)*tan(p), 0, cos(r), -sin(r), 0, sin(r)/cos(p), cos(r)/cos(p)};
Mat3x3F R(v); // Matrix that turns intantaneous turn rate in body frame to inertial frame

//Transform the angular velocity from body frame to inertial frame
V3F i_rate = R* gyro;

//Create a predicted attitude in the global frame
float predictedPitch = pitchEst + dtIMU * i_rate.y ;
float predictedRoll = rollEst + dtIMU * i_rate.x;
ekfState(6) = ekfState(6) + dtIMU * i_rate.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;

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

➢ **The Prediction step**

It is responsible for predicting the next state of the vehicle given the current acceleration in world frame and the angular velocity in the z-axis.

It uses 3 functions:

- The predictState function which predicts the state forward excluding the yaw angle.

```
////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
V3F accel_gf = attitude.Rotate_BtoI(accel * dt);

predictedState(0) = predictedState(0) + predictedState(3)*dt;
predictedState(1) = predictedState(1) + predictedState(4)*dt;
predictedState(2) = predictedState(2) + predictedState(5)*dt;
predictedState(3) = predictedState(3) + accel_gf.x;
predictedState(4) = predictedState(4) + accel_gf.y;
predictedState(5) = predictedState(5) - 9.81f * dt + accel_gf.z;

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

- The GetRbgPrime which computes the Jacobian at the current state.

```
//////////////////////////////// BEGIN STUDENT CODE ///////////////////////////

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);
RbgPrime(2, 0) = 0;
RbgPrime(2, 1) = 0;
RbgPrime(2, 2) = 0;
//////////////////////////////// END STUDENT CODE /////////////////////////////
```

- The predict function which predicts the current covariance forward.

```
////////////////////////////// BEGIN STUDENT CODE ///////////////////////////

//Generating gPrime
VectorXf ac_prime(3), ac(3);
ac(0) = accel.x;
ac(1) = accel.y;
ac(2) = accel.z;
ac_prime = RbgPrime * ac;
gPrime(0, 3) = dt;
gPrime(1, 4) = dt;
gPrime(2, 5) = dt;
gPrime(3, 6) = ac_prime(0) *dt;
gPrime(4, 6) = ac_prime(1) *dt;
gPrime(5, 6) = ac_prime(2) *dt;

//Calculating the predicted covariance
ekfCov = gPrime * ekfCov;
gPrime.transposeInPlace();
ekfCov = ekfCov * gPrime + Q;

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

- **The Magnetometer update**

It updates the value of the yaw angle given measurement from the magnetometer.

```
//////////////////////////////// BEGIN STUDENT CODE ///////////////////////////
zFromX(0) = ekfState(6);
hPrime(0, 6) = 1;

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

```
zt = z - zFromX;
if (z.size() == 1)//If the update function is call whule uptading yaw, normalize the difference btw the measured and the esti
{
    if (zt(0) > F_PI) zt(0) -= 2.f*F_PI;
    if (zt(0) < -F_PI) zt(0) += 2.f*F_PI;
}
```

The GPS update

It updates the vehicle position and velocity given measurement from the GPS.

```
////////////////////////////// BEGIN STUDENT 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);
for (int i = 0; i < 6; i++)
{
    hPrime(i, i) = 1;
}

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