The Estimation Project

Writeup

(Scenario – 06) Determine Measurement Noise

The task was to calculate standard deviations for the measurement noise distributions using the log data retrieved after running the scenario_6 of the simulator.

I copied the data from log files one by one for accelerometer and GPS sensors into an excel sheet. For each sensor data log I performed the following steps to calculate the std. deviations

1- Calculate mean of the sensor data
2- Subtract mean from each sensor reading and square the result
3- Calculate mean of the squared difference. This gives us the variance of the distribution
4- Take square root of the distributions to calculate standard deviation

After calculating the std. deviations the results were plugged into the 06_SensorNoise.txt file provided with the project code.

(Scenario – 07) Rate Gyro Integration Scheme

The task was to implement a better rate gyro integration scheme. The provided integration was a basic complementary filter. A better integration was implemented that used the currently estimated attitude (Roll Pitch Yaw) to integrate the body rates received from the gyro sensor.

I used quaternions to implement this functionality. First I created a quaternion from the current estimated attitude. Next I integrated the body rates provided by the gyro using this quaternion's IntegrateBodyRate() function. This method takes a vector and time delta and creates another quaternion based on this input and multiplies it with the existing quaternion. Next I used Pitch() and Roll() methods of the integrated quaternion to get the angles in inertial frame based on the gyro rates in body frame.

I also retrieved the Yaw() from the same quaternion. The roll/pitch angles from gyro were then fused with roll/pitch angles from the accelerometer to get better estimates of these angles.

This code is implemented in updateFromIMU() method in QuadEstimatorEKF.cpp file.

(Scenario – 08) Predict Step (predicted estimates of state)

The task was to estimate the position and velocity of the state vector based on accelerometer. For the EKF estimation, accelerometer measurements were treated as control inputs since it provides a better estimate of the true state of the vehicle.

I implemented the method predictState() for this task. I used the formula for g() that is

G() = Ax + Bu * dt

A is a 7 x 1 matrix that contains the predicted estimate of the state based on the input control (which is accelerometer readings). B matrix is a 4 x 7 matrix that contain elements from the rotation matrix R. These elements transform the control input which is provided in body frame to inertial frame. So control vector u is transformed using matrix b and then scaled with time delta dt to integrate the control accelerations to calculate the predicted velocities.

(Scenario – 09) Predict Covariance

The next step was to calculate the predicted covariance in order to better estimate the error uncertainity. For this I needed to calculate the jacobian of G[ ] called G-Prime[ ]. To calculate gPrime I first calculated RbgPrime in function

GetRbgPrime() and then used it in Predict() function to calculate G-Prime[ ]. Next I used the EKF equation for calculating predicted covariance in the predict method.

This equation uses gPrime, current covariance and process noise matrix Q to update the covariance.

(Scenario – 10) Update from magnetometer

Given the heading measurement from the magnetometer, I needed to now calculate the final estimate of the vehicle's heading. In EKF this is modeled like a gaussian distribution which basically determines the maximum likelihood of the measurements given the predicted state. For this I calculated the measurement model of the magnetometer and its jacobian, H and H-Prime. H is the current estimated yaw. This was implemented in the method updateFromMag(). This method calls the update( ) method to estimate the new state based on h and hPrime, measurement and measurement covariance.

(Scenario – 11) Update from GPS

Similarly for GPS measurements I calculated H and H-Prime. The H-Prime in this case was an identity matrix with an additional column of zero elements. I implemented this in the function updateFromGPS( ). This method calls the update( ) method to estimate the new state based on h and hPrime, measurement and measurement covariance.

Flight Evaluation - Estimator Parameter Tuning

The resulting estimator needed to pass all the scenarios given in the simulator by tracking the true state with estimated plus capture the process noise and measurement noise of the sensor using the covariance. I tuned the process noise parameters to achieve these. All scenarios from 6 to 11 yield the required results.

Custom Controller Performance

The last requirement was to switch the provided ideal controller with my custom controller from previous project and relax it to pass all the scenarios from 6 to 11. I copied the control code and params files from previous control project to the estimation project. When I ran the estimator crashed. Next according to the provided instruction I reduced the position and

velocity gains and the resulting controller worked flawlessly with my new estimator to pass all the scenarios.