Kalman Filter

Roddy G. Posada

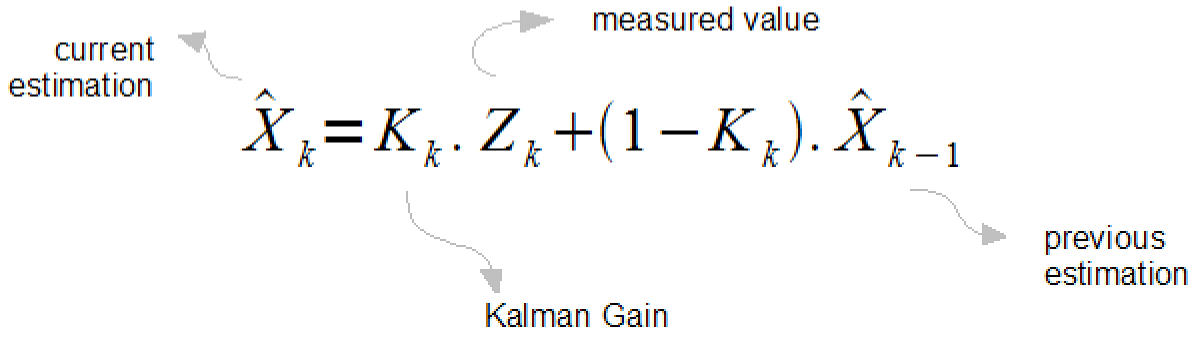
Contents

Introduction

Rudolf Kalman was born in Budapest, Hungary, and obtained his bachelor's degree in 1953 and master's degree in 1954 from MIT in electrical engineering. His doctorate in 1957 was from Columbia University. Kalman is an electrical engineer by training, and is famous for his co-invention of the Kalman filter, a mathematical technique widely used in control systems and avionics to extract a signal from a series of incomplete and noisy measurements. Kalman filters were used during the Apollo program.

The Kalman Filter developed in the early sixties by R.E. Kalman is a recursive state estimator for partially observed non-stationary stochastic processes. It gives an optimal estimate in the least squares sense of the actual value of a state vector from noisy observations. [2]

The Math behind the filter



$$\hat{X}_k = K_k . Z_k + (1 - K_k) . \hat{X}_{k-1}$$

A great document that explains the math behind the Kalman filter can be found on http://www.tristanfletcher.co.uk/LDS.pdf , but the reality is that it is much more simple to say that the Filter is just a tool that once is program adequately, it receives current data, predicts or projects a possible outcome in a recursive manner, meaning that once new data comes in, it is compared with the last predicted data and immediately is adjusted. Simply explained:

“***Kalman filter finds the most optimum averaging factor for each consequent state. Also somehow remembers a little bit about the past states***.” [3]

Code Implementation

The Filter needs input information in order to work, the input information provided comes from the IMU unit, which is a 9 degrees of freedom (DOF), meaning that is possess 3-axis Gyro (X,Y,Z) , 3-axis Accelerometer (X,Y,Z), and a compass for orientation. Nano/Mini/Quad copter uses just the first two embedded devices measurements to feed the Kalman code.

```
/* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.

 This software may be distributed and modified under the terms of the GNU
 General Public License version 2 (GPL2) as published by the Free Software
 Foundation and appearing in the file GPL2.TXT included in the packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works based
 on this software must also be made publicly available under the terms of
 the GPL2 ("Copyleft").

 Contact information
 -------------------

 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
 */

#ifndef _Kalman_h
#define _Kalman_h

class Kalman {
public:
    Kalman() {
        /* We will set the varibles like so, these can also be tuned by the user */
        Q_angle = 0.001;
        Q_bias = 0.003;
        R_measure = 0.03;

        bias = 0; // Reset bias
        P[0][0] = 0; // Since we assume that the bias is 0 and we know the starting angle
(use setAngle), the error covariance matrix is set like so - see:
http://en.wikipedia.org/wiki/Kalman_filter#Example_application.2C_technical
        P[0][1] = 0;
        P[1][0] = 0;
        P[1][1] = 0;
    };
    // The angle should be in degrees and the rate should be in degrees per second and
the delta time in seconds
    double getAngle(double newAngle, double newRate, double dt) {
        // KasBot V2  -  Kalman filter module - http://www.x-firm.com/?page_id=145
        // Modified by Kristian Lauszus
        // See my blog post for more information:
http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-
implement-it

        // Discrete Kalman filter time update equations - Time Update ("Predict")
        // Update xhat - Project the state ahead
        /* Step 1 */
        rate = newRate - bias;
        angle += dt * rate;

        // Update estimation error covariance - Project the error covariance ahead
        /* Step 2 */
        P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angle);
        P[0][1] -= dt * P[1][1];
        P[1][0] -= dt * P[1][1];
        P[1][1] += Q_bias * dt;

        // Discrete Kalman filter measurement update equations - Measurement Update
("Correct")
        // Calculate Kalman gain - Compute the Kalman gain
        /* Step 4 */
        S = P[0][0] + R_measure;
        /* Step 5 */
        K[0] = P[0][0] / S;
        K[1] = P[1][0] / S;

        // Calculate angle and bias - Update estimate with measurement zk (newAngle)
        /* Step 3 */
        y = newAngle - angle;
        /* Step 6 */
        angle += K[0] * y;
        bias += K[1] * y;

        // Calculate estimation error covariance - Update the error covariance
        /* Step 7 */
        P[0][0] -= K[0] * P[0][0];
        P[0][1] -= K[0] * P[0][1];
        P[1][0] -= K[1] * P[0][0];
        P[1][1] -= K[1] * P[0][1];

        return angle;
    };
    void setAngle(double newAngle) { angle = newAngle; }; // Used to set angle, this
should be set as the starting angle
    double getRate() { return rate; }; // Return the unbiased rate

    /* These are used to tune the Kalman filter */
    void setQangle(double newQ_angle) { Q_angle = newQ_angle; };
    void setQbias(double newQ_bias) { Q_bias = newQ_bias; };
    void setRmeasure(double newR_measure) { R_measure = newR_measure; };

private:
    /* Kalman filter variables */
    double Q_angle; // Process noise variance for the accelerometer
    double Q_bias; // Process noise variance for the gyro bias
    double R_measure; // Measurement noise variance - this is actually the variance of
the measurement noise

    double angle; // The angle calculated by the Kalman filter - part of the 2x1 state
matrix
    double bias; // The gyro bias calculated by the Kalman filter - part of the 2x1 state
matrix
    double rate; // Unbiased rate calculated from the rate and the calculated bias - you
have to call getAngle to update the rate

    double P[2][2]; // Error covariance matrix - This is a 2x2 matrix
    double K[2]; // Kalman gain - This is a 2x1 matrix
    double y; // Angle difference - 1x1 matrix
    double S; // Estimate error - 1x1 matrix
};

#endif
```

How it works

In the first section of the code below the data received from the sensors is organized and prepared to be used as an input for the Kalman filter

```
// made by Kristian Lauszus - see http://arduino.cc/forum/index.php/topic,58048.0.html for
information
#define gX A0
#define gY A1
#define gZ A2

#define aX A3
#define aY A4
#define aZ A5

double zeroValue[6] = { 0 }; // gyroX, gyroY, gyroZ, accX, accY, accZ

/* All the angles start at 180 degrees */
double gyroXangle = 180;
double gyroYangle = 180;
double gyroZangle = 180;

// Complimentary filter
double compAngleX = 180;
double compAngleY = 180;

// Used for timing
unsigned long timer;

void setup() {
  analogReference(EXTERNAL); // 3.3V
  Serial.begin(115200);
  delay(100);//wait for the sensor to get ready

  // Calibrate all sensors in horizontal position
  for (uint8_t i = 0; i < 100; i++) { // Take the average of 100 readings
    zeroValue[0] += analogRead(gX);
    zeroValue[1] += analogRead(gY);
    zeroValue[2] += analogRead(gZ);
    zeroValue[3] += analogRead(aX);
    zeroValue[4] += analogRead(aY);
    zeroValue[5] += analogRead(aZ);
    delay(10);
  }
  zeroValue[0] /= 100;
  zeroValue[1] /= 100;
  zeroValue[2] /= 100;
  zeroValue[3] /= 100;
  zeroValue[4] /= 100;
  zeroValue[5] /= 100;
  zeroValue[5] -= 102.3; // Z value is -1g when facing upwards - Sensitivity =
0.33/3.3*1023=102.3

  timer = micros(); // start timing
}

void loop() {
  double gyroXrate = -((analogRead(gX)-zeroValue[0])/1.0323); // (gyroXadc-
gryoZeroX)/Sensitivity - in quids - Sensitivity = 0.00333/3.3*1023=1.0323
```

```
  gyroXangle += gyroXrate*((double)(micros()-timer)/1000000); // Without any filter

  double gyroYrate = -((analogRead(gY)-zeroValue[1])/1.0323);
  gyroYangle += gyroYrate*((double)(micros()-timer)/1000000);

  /*
  double gyroZrate = -((analogRead(gZ)-zeroValue[2])/1.0323);
  gyroZangle += gyroZrate*((double)(micros()-timer)/1000000);
  Serial.println(gyroZangle); // This is the yaw
  */

  double accXval = (double)analogRead(aX)-zeroValue[3];
  double accYval = (double)analogRead(aY)-zeroValue[4];
  double accZval = (double)analogRead(aZ)-zeroValue[5];

  // Convert to 360 degrees resolution
  // atan2 outputs the value of -π to π (radians) - see http://en.wikipedia.org/wiki/Atan2
  // We are then convert it to 0 to 2π and then from radians to degrees
  double accXangle = (atan2(accXval, accZval)+PI)*RAD_TO_DEG;
  double accYangle = (atan2(accYval, accZval)+PI)*RAD_TO_DEG;

  /* You might have to tune the filters to get the best values */
  compAngleX = (0.98*(compAngleX+(gyroXrate*(double)(micros()-
timer)/1000000)))+(0.02*(accXangle));
  compAngleY = (0.98*(compAngleY+(gyroYrate*(double)(micros()-
timer)/1000000)))+(0.02*(accYangle));
  double xAngle = kalmanX(accXangle, gyroXrate, (double)(micros()-timer));
  double yAngle = kalmanY(accYangle, gyroYrate, (double)(micros()-timer));

  timer = micros(); // reset timing

  /* print data to processing */
  Serial.print(gyroXangle);Serial.print("\t");
  Serial.print(gyroYangle);Serial.print("\t");

  Serial.print(accXangle);Serial.print("\t");
  Serial.print(accYangle);Serial.print("\t");

  Serial.print(compAngleX);Serial.print("\t");
  Serial.print(compAngleY); Serial.print("\t");

  Serial.print(xAngle);Serial.print("\t");
  Serial.print(yAngle);Serial.print("\t");

  Serial.print("\n");

  delay(10);
}
```

References

1. Fletcher , T. ”Kalman filter explained” , http://www.tristanfletcher.co.uk/LDS.pdf
2. S. J. Julier and J. K. Uhlmann. Unscented filtering and nonlinear estimation. Proceedings of the IEEE, 92(3):401–422, March 2004.
3. Kalman filter for dummies, http://bilgin.esme.org/BitsBytes/KalmanFilterforDummies.aspx