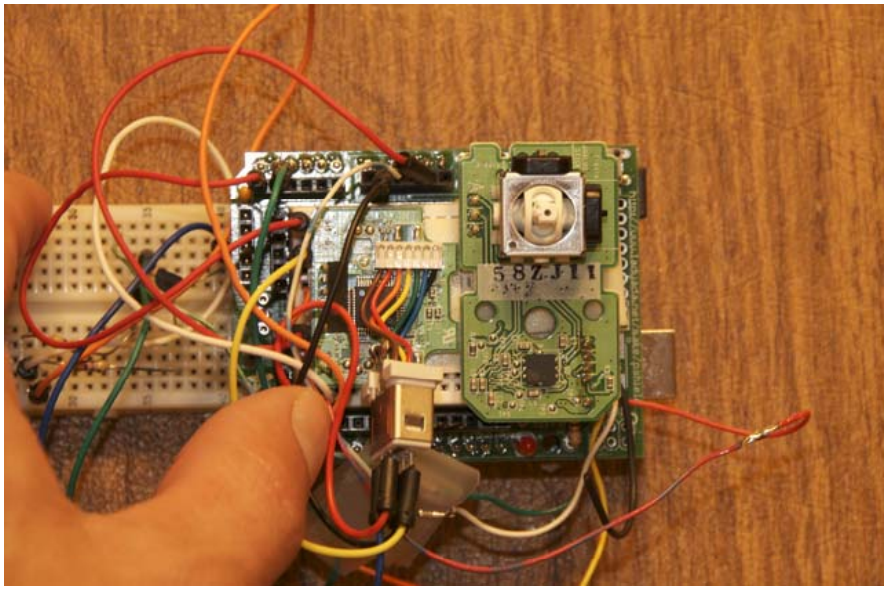
칼만 필터 살펴보기

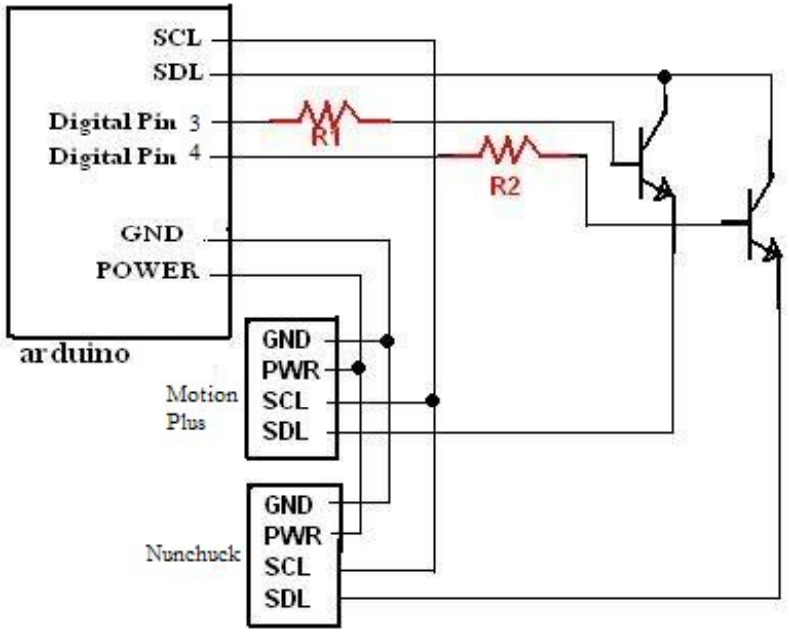
이번에는 칼만 필터(Kalman filter)에 대해서 살펴보고자합니다.
인터넷에서 칼만 필터 관련 자료를 찾아보면 많은 자료와 응용이 소개되어 있습니다.

위키백과에 칼만필터에 대해서 우리말로 잘 정리설명되어 있습니다.
http://ko.wikipedia.org/wiki/%EC%B9%BC%EB%A7%8C_%ED%95%84%ED%84%B0#.EC.B9.BC.EB.A7.8C_.ED.95.84.ED.84.B0.EC.9D.98_.EC.A0.81.EC.9A.A9_.EB.B6.84.EC.95.BC
영어로 보시려면 여기(http://en.wikipedia.org/wiki/Kalman_filter)를 참조하시면됩니다.
그리고 이곳(http://www.cs.unc.edu/~welch/kalman/)에도 칼만 필터 관련 자료가 잘 정리되어 있네요.

아두이노 관련 소스를 보시려면 http://arduino.cc/ 에서 kalman filter를 검색하시면 몇 개의 소스 들(예 http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1253202298)을 볼 수 있습니다. 좀 살펴봐서 참조하시면 됩니다.

저는 이곳 (http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1248889032/45)에서 **duckhead** 의 소스 좀 살펴보니 마음에 드네요.
Duckhead씨는 눈척과 모션플러스를 아래 사진과 같이 꾸며서 실험했다 하는데, 2개의 연결을 아래 오른쪽 그림과 같이 연결해서 했다네요.
이 방법은 2개의 트랜지스터를 사용하여 눈척과 모션플러스를 선택 액세스하는데, 꾸며서 동작시켜보니 나름 괜찮습니다.





칼만 필터 살펴보기

RealSYS

소스의 머리말:

```
// Modified Kalman code using Roll, Pitch, and Yaw from a Wii MotionPlus and X, Y, and Z accelerometers from a Nunchuck.
// Also uses "new" style Init to provide unencrypted data from the Nunchuck to avoid the XOR on each byte.
// Requires the WM+  and Nunchuck to be connected to Arduino pins D3/D4 using the schematic from Knuckles904 and johnnyonthespot
// See http://randomhacksofboredom.blogspot.com/2009/07/motion-plus-and-nunchuck-together-on.html
// Kalman Code By Tom Pycke. http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data
// Original zipped source with very instructive comments: http://tom.pycke.be/file_download/4
// Some of this code came via contributions or influences by Adrian Carter, Knuckles904, evilBunny, Ed Simmons, & Jordi Munoz
// There are two coordinate systems at play here:
// The body-fixed coordinate system, which is fixed to the platform. That is, the x-axis is always pointing to the nose of the
// aircraft (nosecone on a rocket), the y-axis points to the right, and the z-axis points down on an airplane, (parallel to the
// earth away from you on a rocket).
// The aircraft will move with respect to the navigation frame, which is a fixed coordinate system.
// In that frame, the x-axis points north, the y-axis points east and the z-axis points down. This is called the "NED system", or
// the "Local Tangent Plane" or the "Local Geodetic Frame".
// Created by Duckhead    v0.6    Yaw is done via integration of gyro data and smoothed via a Runge-Kutta 4, but there are still issues with it
```

하드웨어 연결은 이곳 (http://randomhacksofboredom.blogspot.com/2009/07/motion-plus-and-nunchuck-together-on.html)에서 참조하시고,
칼만 소스 코드는 여기(http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data)에서 참조 했다하는 데, 여러가지 자료들이 좀 있고, 칼만필터 관련 자료로 http://academic.csuohio.edu/simond/courses/eec644/kalman.pdf 와
이론적인 자료 http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf 를 링크해줍니다. 또한 dsPIC30을 사용한 소스 KalmanTestBoardV1.zip 도 링크해주는데, 아두이노 소스와 비슷합니다.

네이버에서 칼만필터를 검색해보면
http://kin.naver.com/qna/detail.nhn?d1id=11&dirId=1118&docId=57854287&qb=7Lm866eM7ZWE7YSw&enc=utf8&section=kin&rank=1&search_sort=0&spq=0&pid=gLRNMwoi5TVsss4/aiGsss--271841&sid=TNT2FAvw1EwAAC15Dr4
나름대로 설명이 잘되어 있습니다.

저는 아직 이론적인 지식이 부족하여
소스 설명및 기초 이론은 고수들님께서 해주시기를 기대합니다.

칼만 필터 쉬운 설명

RealSYS

칼만필터는 1960년 R.E.Kalman (Kalman, Rudolph, Emil) 의 논문 "A New Approach to Linear Filtering and Prediction Problems"
에 그 시초를 두고 있다. 여기서는 처음 칼만필터의 내용을 접하는 초보자를 위해 한글로 몇가지 중요한 내용을 정리해본다.
칼만필터는 흔히 " an optimal recursive data processing algorithm" 이라고 불린다.
이에 대한 직접적인 설명을 하기 전에 간단한 예로써 그 의미를 전달해보자.

어느 고등학교 선생님이 자신의 학급 학생들의 수학점수 평균을 알고 싶다고 가정하자.
물론 모든 학생의 점수를 더한 후 이를 학생 수로 나누는 방법이 가장 쉽고 간편할 것이다.
그러나, 한가지 재미를 더하기 위해 상황을 설정하자면 아직 선생님은 학생들의 점수를 전혀 알고 있지 못한 상황이고, 학생들은 한번에 한 명씩만 선생님께 점수를 말해준다고 가정하자.
이때 선생님이 평균을 짐작해보기 위한 가장 좋은 방법은 학생 한명 한명이 점수를 말할 때마다 이들의 평균으로 전체의 평균을 짐작해보는 것이다.
즉 다음과 같은 관계식을 생각해볼 수 있다.

X'(1) = X1
X'(2) = (X1+X2)/2 = (X'(1)*1+X2)/2
X'(3) = (X1+X2+X3)/3 = (X'(2)*2+X3)/3
X'(4) = (X1+X2+X3+X4)/4 = (X'(3)*3+X4)/4
:
X'(n) = (X1+X2+.....+Xn)/n = (X'(n-1)*(n-1)+Xn)/n

위 식의 일반 식은 다음과 같이 변형할 수도 있다.

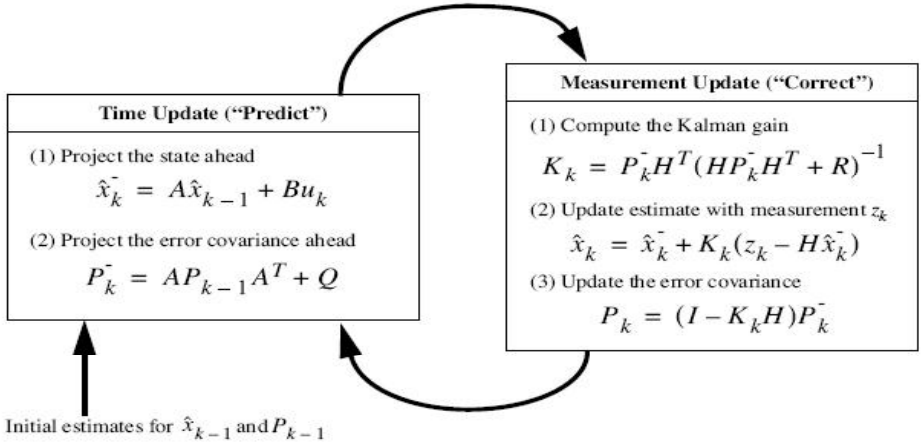
X'(n) = X'(n-1)*((n-1)/n) + (1/n)*Xn ; ← 손으로 좀 정리해보면 나옵니다.

그럼 위 식이 갖고 있는 특징은 무엇일까?
위와 같은 식을 사용할 때의 간편한 점은 지나간 개개의 값들을 모두 기억해 둘 필요가 없다는 것이다.
즉 n번째 평균을 구할 때 선생님은 X'(n-1) 과 지금 n번째의 값 Xn 두 개의 값만 있으면 아무 문제없이 n번째의 평균을 구할 수 있게 된다.

이와 같은 기법을 "반복적 자료 처리 (recursive data processing)" 이라고 하며, 최초 Kalman Filter 이론의 개발 이유이기도 하다.
처리할 자료의 양이 그리 많지 않을 때는 모든 자료를 기억해두었다가 한번에 계산하는 것이 간편할 수도 있겠으나, 그 자료의 양이 방대하다면,
저장 장소 및 처리 시간을 고려하지 않을 수 없게 된다.
칼만필터가 개발된 60년대는 컴퓨터의 발달이 초보적인 상태였기에 자료의 효율적 저장 및 처리가 절실할 시절임을 감안한다면 어쩌면 당연한 고안이라고 생각할 수 있겠다.
그럼 이제 "최적 (optimal)" 이라는 단어에 초점을 둬 보자. 위에서 제시한 예는 개개의 데이터 (각 학생의 수학점수)에 동일한 가중치, 즉 새로 받아들이는 모든 값에는 1/n의 가중치가 두어졌다. 그렇다면 가중치가 다른 경우에는 어떠할까?
같은 점수를 갖고 있는 학생 3명이 함께 와서 "우리점수는 ㅇㅇ점 입니다."라고 한다면 이점수에 대한 가중치는 3/n을 주어야 할 것이다.
이와 같이 각 데이터가 갖고 있는 평균에 대한 중요도를 "경중률"이라고 한다.
경중률에 대한 개념은 측량에 있어서 여러 방법으로 길이를 재고 이들의 평균을 구해봄으로써 더욱 쉽게 이해할 수 있다.

칼만 필터 쉬운 설명

이번에는 위의 수학선생님이 학생 두 명에게 각기 다른 방법으로 교실의 길이를 재보라고 했다고 하자.
한 학생은 자신의 보폭을 이용하고, 다른 학생은 줄자를 이용하여 교실의 길이를 측정했을 때, 첫 번째 학생은 10m, 두 번째 학생은 12m 라는 값을 얻었다. 만약 둘 모두 같은 방법으로 길이를 측정하였다면, 선생님은 주저 없이 두 값을 평균하여 11m 를 교실의 길이라고 말할 수 있겠다.
그러나, 누가 보더라도 보폭으로 잰 길이보다는 줄자로 잰 길이가 정확하다고 느낄 것이다. 그렇다면 이들 두 값을 어떻게 평균해야 할까? 이때 도입되는 개념이 경중율이다.
측량학에 있어서 관측 값의 경중률은 각 값이 갖고 있는 표준편차를 기준으로 정한다. 즉 각 관측치에 대한 경중률은 표준편차 제곱에 반비례한다.
표준편차가 크면 클수록 그 값의 경중률은 떨어지며 평균에 대한 기여도가 적어지게 된다.
줄자로 잰 거리가 10m이지만, 수많은 경험 또는 수많은 반복 측정을 통해 얻어진 표준편차가 +-0.1m라고 가정한다면,
그리고 보측은 12m +-1.0m라고 가정한다면, 이들의 평균은 다음과 같이 구해진다.
평균 = (10*(1.0)^2 + 12*(0.1)^2) / (0.1^2+1.0^2) = 10.02m
위의 식을 일반화 하면,
X = (X1*(sd2)^2 + X2*(sd1)^2) / ((sd1)^2+(sd2)^2) = (X1*(sd1)^2 - X1*(sd1)^2 + X1*(sd2)^2 + X2*(sd1)^2) / ((sd1)^2+(sd2)^2)
= X1 + (X2-X1)* [(sd1)^2 / ((sd1)^2+(sd2)^2)]
여기서, X : 상태변수 (state variables) sd : 표준편차 (standard deviation)
위의 마지막 식이 칼만 필터에 있어서 시스템 출력 값 (위에서는 X1) 과 새로운 입력값 (X2) 을 이용하여 새로운 최적값 (optimized state variables) 을 계산하는 "관측갱신 알고리즘 (measurement update algorithm)" 의 스칼라 형태이다. (일반적으로 칼만필터의 기본식은 행렬 또는 벡터형태로 나타나 있다.)
더불어 새로운 최적값, X, 의 표준편차는 다음과 같이 계산된다.
(sd)^2 = [(sd1)^2+(sd2)^2] / [(sd1)^2*(sd2)^2]
이제 위의 두 식, 최적값 X와 표준편차 sd를 구하는, 을 이용하면 앞에서 말한 두 관측 값 이외에 다른 관측값을 추가적으로 얻어서 다시 새로운 최적값을 구할때도 같은 방법을 계속 사용할 수 있다. 이때는 앞에서 얻어진 X 는 X1 이 되고, sd 는 sd1 이 되며, 새로운 관측값과 표준편차(경중률)은 각각 X2 와 sd2 가 된다.
이와 같은 반복적인 (recursive) 연산 (data processing) 을 통해 최적 (optimal) 값을 추적하는 것이 칼만 필터의 기본개념이라 할 수 있겠다.



칼만 필터 쉬운 설명

RealSYS

먼저 시스템 방정식과 관측방정식을 살펴보자.
칼만필터를 도입하기 위해서는 기본적으로 위와 같은 두 선형방정식이 필요하다.

비선형 방정식에 대한 "확장형 칼만필터 (Extended Kalman Filter)" 는 비선형 방정식을 테일러 급수전개 등을 이용하여 선형화 한 후 적용한 형태일 뿐이다. (다만 비선형의 선형화에 따른 변환계수 -그림에서 A 또는 H와 같은- 의 형태가 달라질 뿐이다.
이에 대해서는 "확장형 칼만필터"에서 다시 자세히 설명하겠다.)

시스템 방정식에서 x는 우리의 관심, 즉 최적화를 하고자 하는 상태변수를 의미하고 계수 A 는 한 단계에서의 상태변수와 다음 단계의 상태변수를 연결하는 변환계수를 표현한다.
B 와 u 는 한 덩어리로 인식할 수 있으며 이들은 시스템에 무관한 추가 입력값이다.
마지막으로 w 는 k 단계에서 상태변수 x 의 참값과의 차이값 또는 "시스템 오차(system error or system noise)" 이다. (우리가 알고 있는 x 는 참값에 근접한 계산값일 뿐이다.) w 는 개별적으로 값을 구하거나 지정할 수 없으며, 단지 오랜 관측 및 시스템의 제작 시부터 알고 있는 참값에 대한 표준편차로써 -칼만필터 안에서는 분산의 형태로써- "Q" 라는 변수로 적용된다.

관측방정식에서 z 는 관측값이고 이는 상태변수 x 와 변환계수 H 에 의해 표현되며 v 는 관측값 z 와 관측참값과의 오차 (measurement error or measurement noise) 이다. v 는 w 와 마찬가지로 개개의 값을 알 수는 없고 관측 참값에 대한 분산인 "R" 라는 변수로써 칼만필터 안에서 사용된다.

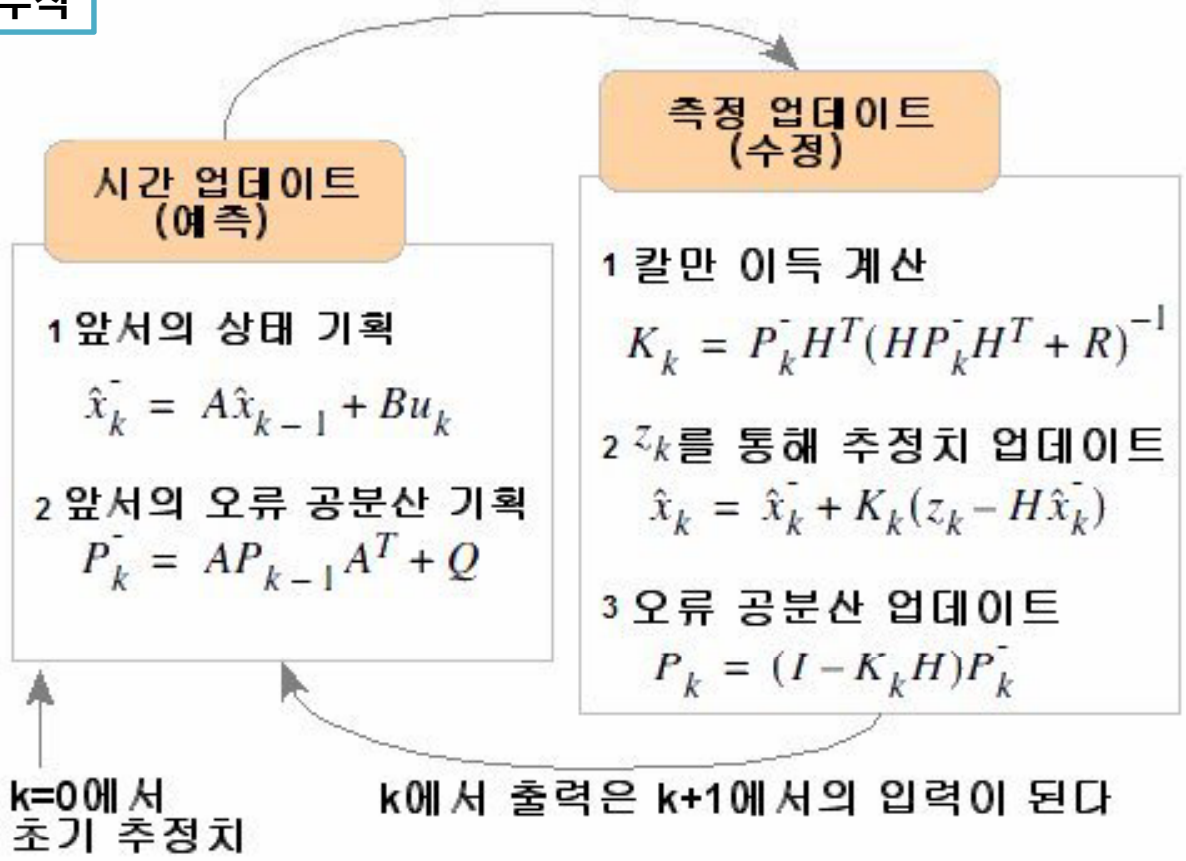
여기에서 칼만필터의 기본가정을 소개하자면, 이는 [오차 w 및 관측방정식에서의 v 는 각각의 참값에 대해 정규분포하며 그 평균은 0 이고 분산은 각각 Q 와 R 이다.] 이다.

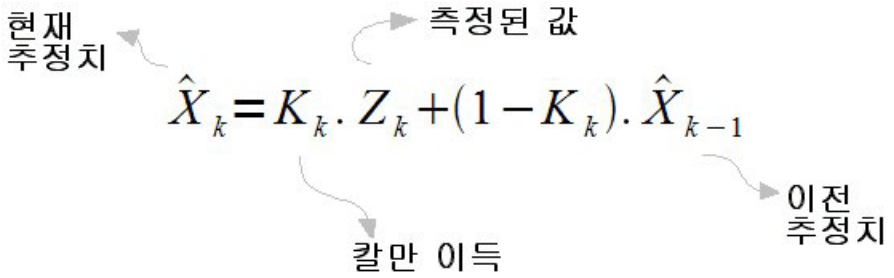
이 가정은 칼만필터의 가장 큰 장점인 반면 또한 가장 큰 단점으로 작용하기도 한다.
시스템 연산값 및 관측값이 참값에 대해 정규분포하는 일반적인 경우에는 위와 같이 간단한 칼만필터 알고리즘을 통해 최적화 할 수 있는 강력한 도구가 되지만, 그렇지 않은 경우 -오차가 참값에 대해 정규분포하지 않는 경우 또는 그렇다 하더라도 그 분산을 알 수 없는 경우- 에는 그 적용에 있어 문제와 어려움이 크다. 그러나 오차의 확률분포가 정규분포가 아닌 경우라 하더라도 이를 정규분포로 간주함에 있어 그리 큰 무리가 없는 경우에는 칼만필터는 아직까지 유용하고 강력한 도구로 사용된다.(???수정필요!!!)

위에 설명한 시스템 방정식과 관측방정식의 이해를 위해 간단한 예를 들어 본다면, 등속운동을 하고 있으며 때때로 추가적인 가속 또는 감속을 하고 있는 자동차를 생각해보자. 이는 기본적으로 등속운동을 하고 있으니 k-1 단계에서의 속도와 k 에서의 속도는 동일하므로 A는 "1" 이다.
또한 추가적인 가감속에 따른 속도변화는 그것이 이루어진 단계에서 Bu 에 그 값을 적용할 수 있다.
그리고 그 자동차를 외부에서 스피드건을 이용해 관측한다면 이는 그 속도를 직접 관측하는 것이므로 H 역시 "1" 이다.

이 문구는 인터넷 블로그 곳곳에서 인용되고 있네요.

예측, Update 수식


시간 업데이트
(예측)
1 앞서의 상태 기획
$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$
2 앞서의 오류 공분산 기획
$P_k^- = AP_{k-1}A^T + Q$
측정 업데이트
(수정)
1 칼만 이득 계산
$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
2 z_k를 통해 추정치 업데이트
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
3 오류 공분산 업데이트
$P_k = (I - K_k H)P_k^-$
k=0에서
초기 추정치
k에서 출력은 k+1에서의 입력이 된다


현재
추정치
측정된 값
$\hat{X}_k = K_k . Z_k + (1 - K_k) . \hat{X}_{k-1}$
칼만 이득
이전
추정치

```
// Modified Kalman code using Roll, Pitch, and Yaw from a Wii MotionPlus and X, Y, and Z accelerometers from a Nunchuck.
// Also uses "new" style Init to provide unencrypted data from the Nunchuck to avoid the XOR on each byte.
// Requires the WM+ and Nunchuck to be connected to Arduino pins D3/D4 using the schematic from Knuckles904 and johnnyonthespot
// See http://randomhacksofboredom.blogspot.com/2009/07/motion-plus-and-nunchuck-together-on.html
// Kalman Code By Tom Pycke. http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data
// Original zipped source with very instructive comments: http://tom.pycke.be/file_download/4
// Some of this code came via contributions or influences by Ed Simmons, Jordi Munoz, and Adrian Carter
// Created by Duckhead   v0.2   some cleanup and other optimization work remains.  also need to investigate correlation of nunchuk and WMP (which axis is which)
// simplify test by KCO 2010.11.06
#include <math.h>
#include <Wire.h>
////////////////////////////////////////
struct GyroKalman{
  /* These variables represent our state matrix x */
  float x_angle,
        x_bias;
  /* Our error covariance matrix */
  float P_00,
        P_01,
        P_10,
        P_11;
  /*
   * Q is a 2x2 matrix of the covariance. Because we
   * assume the gyro and accelerometer noise to be independent
   * of each other, the covariances on the / diagonal are 0.
   * Covariance Q, the process noise, from the assumption
   *    x = F x + B u + w
   * with w having a normal distribution with covariance Q.
   * (covariance = E[ (X - E[X])*(X - E[X])' ]
   * We assume is linear with dt
   */
```

```
float Q_angle, Q_gyro;
  /*
   * Covariance R, our observation noise (from the accelerometer)
   * Also assumed to be linair with dt
   */
  float R_angle;
};
struct GyroKalman accX;

float accelAngleX=0;              //NunChuck X angle
float accelAngleY=0;              //NunChuck Y angle
float accelAngleZ=0;              //NunChuck Z angle
byte buf[6];                      // array to store arduino output
int cnt = 0;                      //Counter
unsigned long lastread=0;
/*
 * R represents the measurement covariance noise.  In this case,
 * it is a 1x1 matrix that says that we expect 0.3 rad jitter
 * from the accelerometer.
 */
static const float                R_angle         = 0.5; //.3 default
/*
 * Q is a 2x2 matrix that represents the process covariance noise.
 * In this case, it indicates how much we trust the acceleromter
 * relative to the gyros.
 */
static const float                Q_angle         = 0.01; //0.01 (Kalman)
static const float                Q_gyro          = 0.04; //0.04 (Kalman)
//These are the limits of the values I got out of the Nunchuk accelerometers (yours may vary).
const int lowX = -215;
const int highX = 221;
const int lowY = -215;
const int highY = 221;
const int lowZ = -215;
const int highZ = 255;
```

아두이노 소스

```
/*
 * Nunchuk accelerometer value to degree conversion.
Output is -90 to +90 (180 degrees of motion).
 */
float angleInDegrees(int lo, int hi, int measured) {
  float x = (hi - lo)/180.0;
  return (float)measured/x;
}
void nunchuck_init () //(nunchuck){
  // Uses New style init - no longer encrypted so no need to XOR bytes
later... might save some cycles
  Wire.beginTransmission (0x52); // transmit to device 0x52
  Wire.send (0xF0);              // sends memory address
  Wire.send (0x55);              // sends sent a zero.
  Wire.endTransmission ();       // stop transmitting
  delay(100);
  Wire.beginTransmission (0x52);// transmit to device 0x52
  Wire.send (0xFB);              // sends memory address
  Wire.send (0x00);              // sends sent a zero.
  Wire.endTransmission ();       // stop transmitting
}
void initGyroKalman(struct GyroKalman *kalman, const float Q_angle,
const float Q_gyro, const float R_angle) {
  kalman->Q_angle = Q_angle;
  kalman->Q_gyro  = Q_gyro;
  kalman->R_angle = R_angle;

  kalman->P_00 = 0;
  kalman->P_01 = 0;
  kalman->P_10 = 0;
  kalman->P_11 = 0;
}
```

```
 /* The kalman predict method.  See
http://en.wikipedia.org/wiki/Kalman_filter#Predict
 * kalman     the kalman data structure
 * dotAngle  Derivitive Of The (D O T) Angle.  This is the change in
the angle from the gyro.  This is the value from the
 *            Wii MotionPlus, scaled to fast/slow.
 * dt         the change in time, in seconds; in other words the
amount of time it took to sweep dotAngle
 * Note: Tom Pycke's ars.c code was the direct inspiration for this.
However, his implementation of this method was inconsistent
 *        with the matrix algebra that it came from.  So I went with
the matrix algebra and tweaked his implementation here. */
void predict(struct GyroKalman *kalman, float dotAngle, float dt) {
  kalman->x_angle += dt * (dotAngle - kalman->x_bias);
  kalman->P_00 += -1 * dt * (kalman->P_10 + kalman->P_01) +
dt*dt * kalman->P_11 + kalman->Q_angle;
  kalman->P_01 += -1 * dt * kalman->P_11;
  kalman->P_10 += -1 * dt * kalman->P_11;
  kalman->P_11 += kalman->Q_gyro;
}
/* The kalman update method.  See
http://en.wikipedia.org/wiki/Kalman_filter#Update
 * kalman     the kalman data structure
 * angle_m    the angle observed from the Wii Nunchuk
accelerometer, in radians */
float update(struct GyroKalman *kalman, float angle_m) {
  const float y = angle_m - kalman->x_angle;
  const float S = kalman->P_00 + kalman->R_angle;
  const float K_0 = kalman->P_00 / S;
  const float K_1 = kalman->P_10 / S;
  kalman->x_angle += K_0 * y;
  kalman->x_bias  += K_1 * y;
  kalman->P_00 -= K_0 * kalman->P_00;
  kalman->P_01 -= K_0 * kalman->P_01;
  kalman->P_10 -= K_1 * kalman->P_00;
  kalman->P_11 -= K_1 * kalman->P_01;
  return kalman->x_angle;
}
```

프로세싱 소스

```
// 2010.11.06 KCO
import processing.serial.*;
Serial port;  // Create object from Serial class
float  MAX_VAL = 65535.0;
int val0,val1;      // Data received from the serial port
int[] values0;
int[] values1;
int getY(int val) {
  return (int)(val/MAX_VAL * (height-1));
}
void setup() {
  int tmp;
  size(800, 600);
  port = new Serial(this, "COM155", 115200);
  values0 = new int[width];
  values1 = new int[width];
  smooth();
}
int cr;
int cnt=0,cnt_old=0;
void draw(){
int tmp;
  while(port.available() >= 6){
    tmp = port.read();
    if(tmp=='K'){
      val0 = (port.read() << 8) | (port.read());
      val1 = (port.read() << 8) | (port.read());
      cr = port.read();
      cnt++;
    }
  }
 if(cnt != cnt_old){
    cnt_old = cnt;
    for (int i=0; i<width-1; i++){
      values0[i] = values0[i+1];
      values1[i] = values1[i+1];
    }
```

```
values0[width-1] = val0;
    values1[width-1] = val1;

    background(0);
    for (int x=1; x<width; x++) {
      stroke(255,0,0);  // Red
      line(width-x,    height-1-getY(values0[x-1]), width-1-x, height-1-getY(values0[x]));
      stroke(0,255,0);  // Green
      line(width-x,    height-1-getY(values1[x-1]), width-1-x, height-1-getY(values1[x]));
    }
  }
}
```

실행 파형: 적색=입력 X값, 녹색=칼만 필터 처리 값