Bộ lọc Kalman – giải pháp chống nhiễu tuyệt vời cho mọi dự án sử dụng cảm biến

Mô tả dự án:

Rõ ràng khi ta sử dụng cảm biến, giá trị trả về từ chúng luôn thay đổi quanh vị trí cân bằng dù là rất nhỏ, và bạn biết nguyên nhân của hiện tượng này là do nhiễu, bạn luôn muốn loại bỏ nhiễu nhưng việc đó dường như ngoài tầm với của bạn.(-.-)… Đừng lo, chúng ta đã có giải pháp, bấm đọc bài viết này thôi nào!

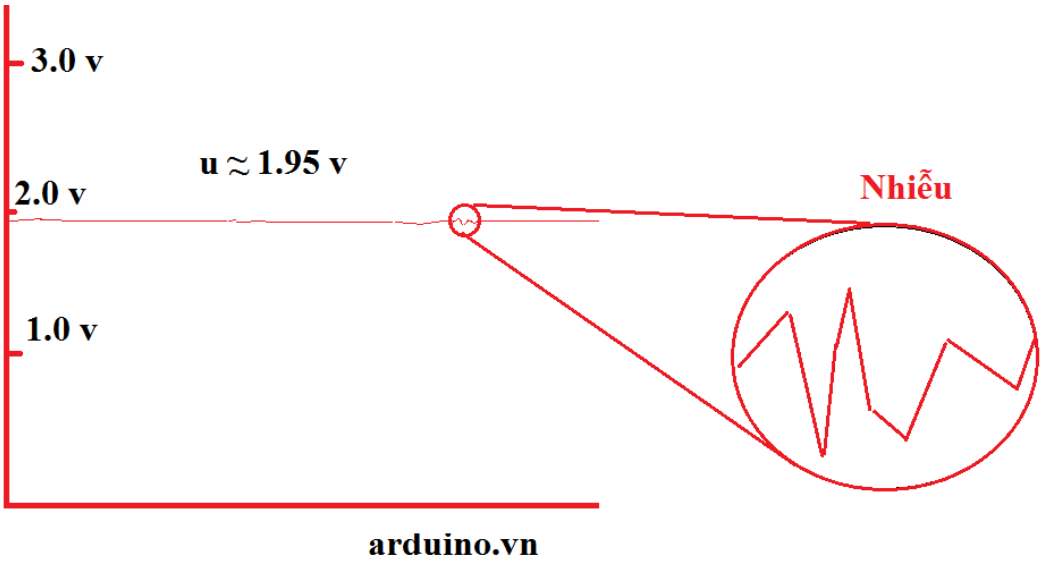
1

Nhiễu …

Trong đo lường, do các yếu tố cả chủ quan lẫn khách quan mà kết quả đo đạc luôn chỉ được coi là tương đối so với giá trị thực cần đo. Độ chênh lệch giữa giá trị đo được với giá trị thực được gọi là **`Sai số`**.

Sai số gây ra bởi nhiễu, chúng được phân làm 2 loại chính: sai số hệ thống hoặc sai số ngẫu nhiên.

Gỉa sử ta tiến hành đo giá trị điện áp của một cục pin năng lượng mặt trời, kết quả thu được trên đồ thị như sau :



Gỉa sử giá trị điện áp thực của pin mặt trời là u^0=1.9545 vol.

Do ảnh hưởng bởi nhiều yếu tố trong quá trình đo đạc, sai số xuất hiện là điều rất khó tránh khỏi, kết quả là ta chỉ có thể đo giá trị điện áp bằng cách lấy tương đối kết quả trung bình các phép đo .

Tức $u^0 = u \pm \Delta e = 1.95 \pm 1\%$, (với Δe là sai số)

2

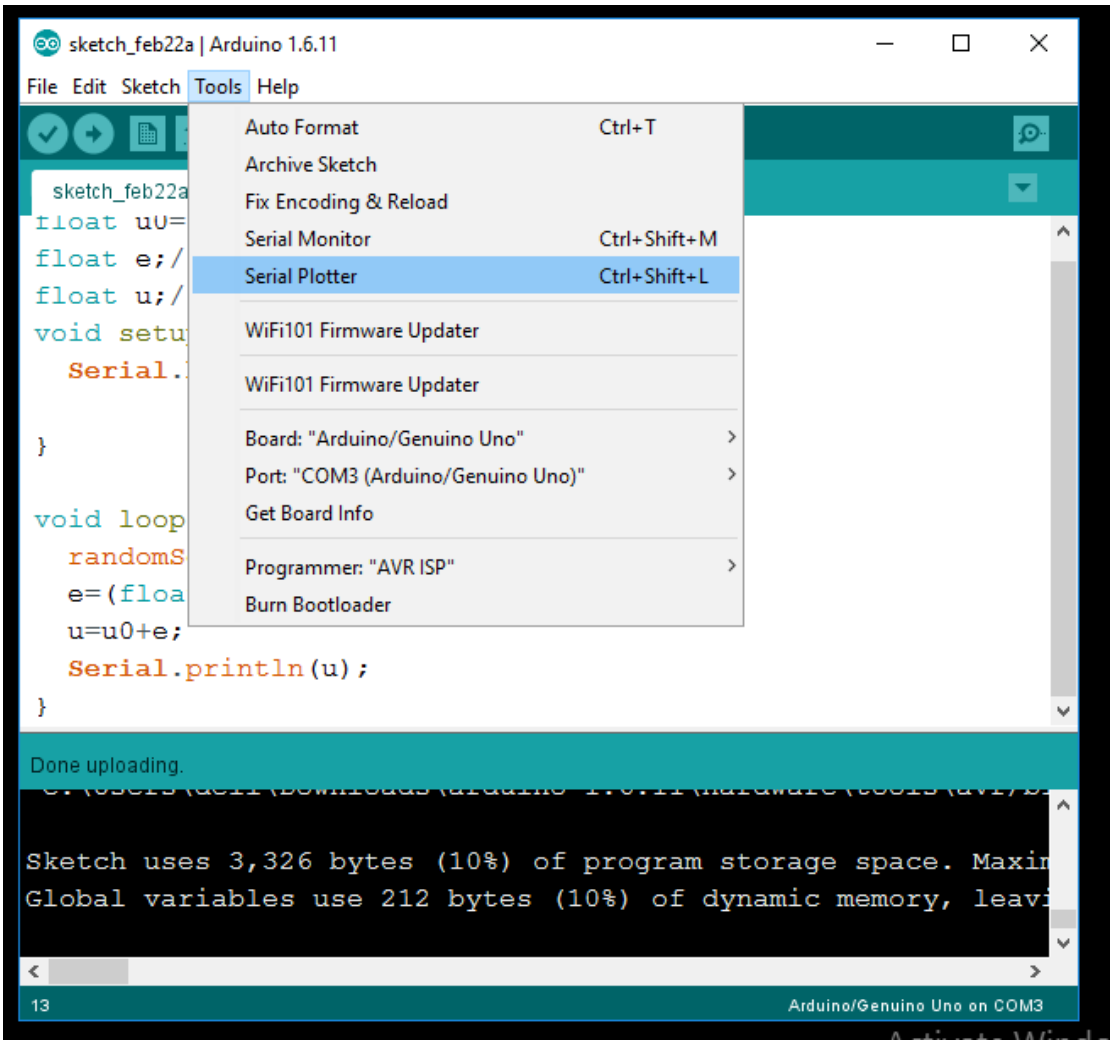
Loại bỏ nhiễu bằng thuật toán lọc Kalman

Phương pháp này được đề suất năm 1960 bởi nhà khoa học có tên Kalman.

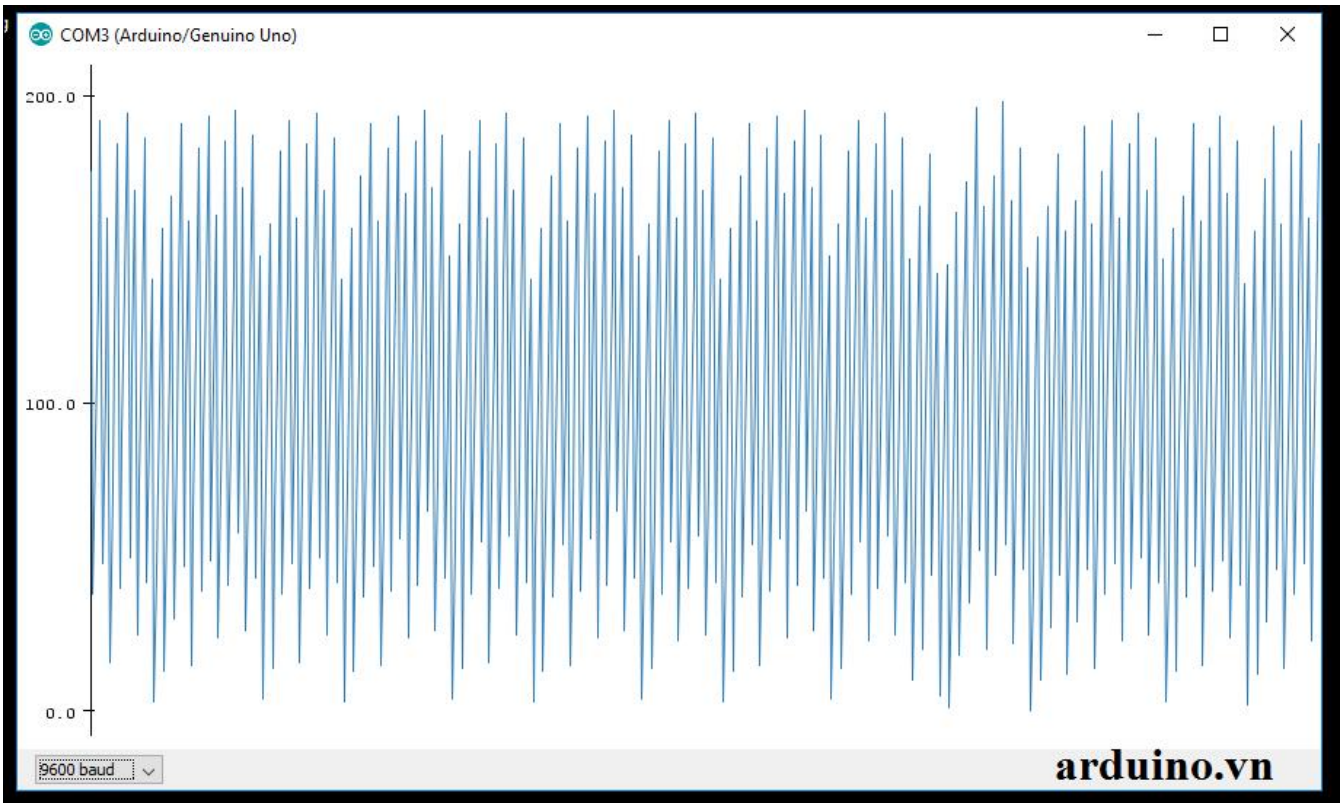
Để chứng minh hiệu quả của phương pháp này chúng ta sẽ có một phép thử mô phỏng như sau.

```
1. float u0 = 100.0; // giá trị thực (không đổi)
2. float e; // nhiễu
3. float u; // giá trị đo được (có thêm nhiễu)
4. void setup()
5. {
6.     Serial.begin(9600);
7. }
8. void loop()
9. {
10.      randomSeed(millis());
11.      e = (float)random(-100, 100);
12.      u = u0 + e;
13.      Serial.println(u);
14.  }
```

Nạp code cho arduino rồi sau đó mở cổng Serial plotter để xem dưới dạng đồ thị:



Kết quả hiển thị trên Serial plotter:



Xem lại code bên trên ta thấy có vài điều như sau:

Gọi $u_0=100.0$ là giá trị thực tế của vật thể, cũng là giá trị mà ta mong muốn thu được, vì u_0 là hằng số, (nếu như không có nhiễu). Lý tưởng thì trên đồ thị ta sẽ thu được một đường thẳng song song với trục thời gian t.

Thường thì nhiễu chỉ dao động trong khoảng $e=\pm 10\%$ giá trị thực đã được coi là rất ồn rồi (noise).

Để tăng độ khó, mình đã cố ý cho $e=\pm 100\%$ u_0 bằng hàm Random khiến cho giá trị đo bị nhiễu hoàn toàn và gần như rất khó để thu thập lẫn tính toán sau này.

Sử dụng bộ lọc Kalman

Như đã thống nhất, trong thực tế u_0 là giá trị chúng ta không biết, việc sử dụng bộ lọc sẽ phải giúp ta loại bỏ các nhiễu, khi đó giá trị đo được phải gần đường $u_0=100$ hơn .

Vì đây là mô phỏng nên giá trị u_0cần được cho trước (chỉ mình và bạn biết) để có thể kiểm chứng tính đúng đắn của kết quả trước và sau khi lọc. (bằng cách trộn u_0 với nhiễu rồi cho arduino lọc)

So với code bên trên ,phần code này chỉ cần thêm một dòng lệnh duy nhất:

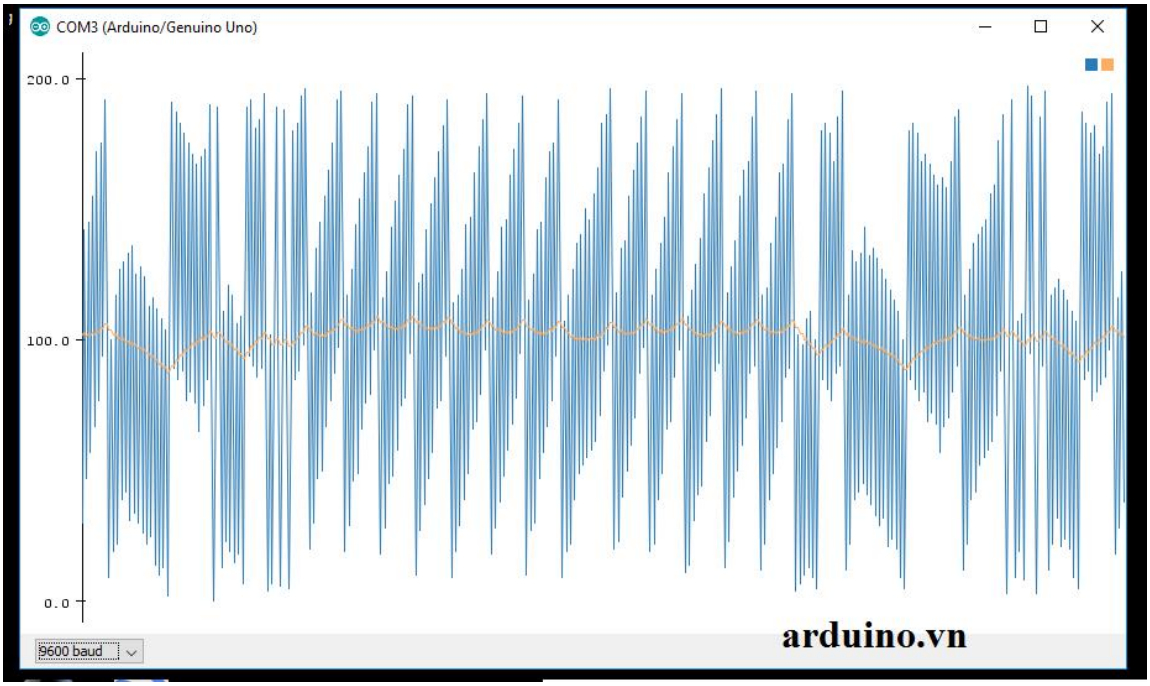
Gọi u_kalman là giá trị đo đã qua bộ lọc Kalman:

u_kalman=bo_loc.updateEstimate(u);

Code

```
#include <SimpleKalmanFilter.h>
SimpleKalmanFilter bo_loc(2, 2, 0.001);

float u0 = 100.0; // giá trị thực (không đổi)
float e; // nhiễu
float u; // giá trị đo được (có thêm nhiễu)
float u_kalman; // giá được lọc nhiễu
void setup()
{
      Serial.begin(9600);
 }
 void loop()
 {
      randomSeed(millis());
      e = (float)random(-100, 100);
      u = u0 + e;
      Serial.print(u);
      Serial.print(",");
      u_kalman = bo_loc.updateEstimate(u);
      Serial.print(u_kalman);
      Serial.println();
 }
```

Và đây là kết quả khi sử dụng thêm bộ lọc:



Đường màu xanh: `u`.

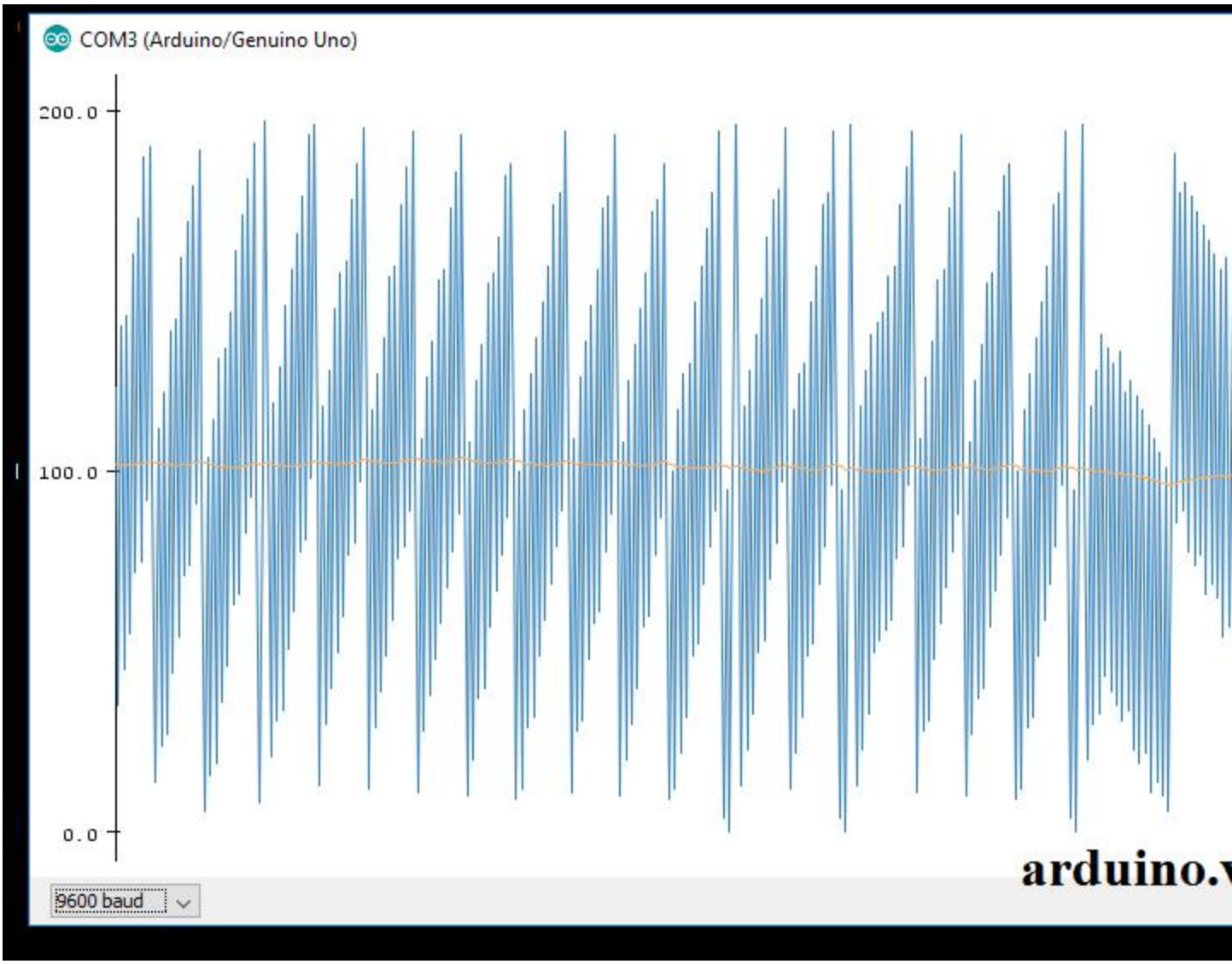
Đường màu vàng: `u_kalman`.

Dừng lại một chút để quan sát đồ thị, hẳn bạn cũng đồng ý với mình thuật toán lọc Kalman tỏ ra rất hiệu quả, có những lúc nhiễu dồn ra biên cực đại ($\pm 100\% u_0$). nhưng giá trị vẫn khá sát đường u_0.

Ghép tầng các bộ lọc ta thu được kết quả chính xác hơn, tất nhiên nó sẽ đáp ứng trễ hơn 1 tầng

```
#include <SimpleKalmanFilter.h>
SimpleKalmanFilter bo_loc(2, 2, 0.001);

float u0 = 100.0; // giá trị thực (không đổi)
float e; // nhiễu
float u; // giá trị đo được (có thêm nhiễu)
float u_kalman; // giá được lọc nhiễu
void setup()
{
      Serial.begin(9600);
 }
 void loop()
 {
      randomSeed(millis());
      e = (float)random(-100, 100);
      u = u0 + e;
      Serial.print(u);
      Serial.print(",");
      u_kalman = bo_loc.updateEstimate(u); // tầng 1
      u_kalman = bo_loc.updateEstimate(u_kalman); // tầng 2
      u_kalman = bo_loc.updateEstimate(u_kalman); // tầng 3
      u_kalman = bo_loc.updateEstimate(u_kalman); // tầng 4
      Serial.print(u_kalman);
      Serial.println();
 }
```

Kết quả cho 4 tầng lọc



Bạn nào còn nghi ngờ kết quả giống mình thì copy code rồi test lại dùm mình nhé, thật khó tin phải không ! ^^ 😈

3

Ma thuật hay trí tuệ nhân tạo ?!!

Đây là thành quả nghiên cứu có từ 56 năm trước, chủ nhân của thuật toán là ông ***Rudolf (Rudy) E. Kálmán***

Bộ lọc này được sử dụng rộng rãi trong các máy tính kỹ thuật số của các hệ thống điều khiển, Hệ thống định vị, Hệ thống điện tử, RADA, vệ tinh dẫn đường để lọc nhiễu.

Bộ lọc hoạt động ổn định đến mức, nó đã được sử dụng trong chương trình Apollo, tàu con thoi của NASA, tàu ngầm Hải quân và xe tự hành không gian không người lái và vũ khí, tên lửa hành trình.

Với thành công đó, ông cũng đã nhận được rất nhiều giải thưởng lớn khác nhau.

https://vi.wikipedia.org/wiki/Rudolf...

4

Thuật toán Kalman trong C

Đây là thuật toán đơn giản mình tìm được, hãy vào địa chỉ bên dưới để hiểu được ý nghĩa của các tham số K (Kalman Gian) ,P,Q,R….

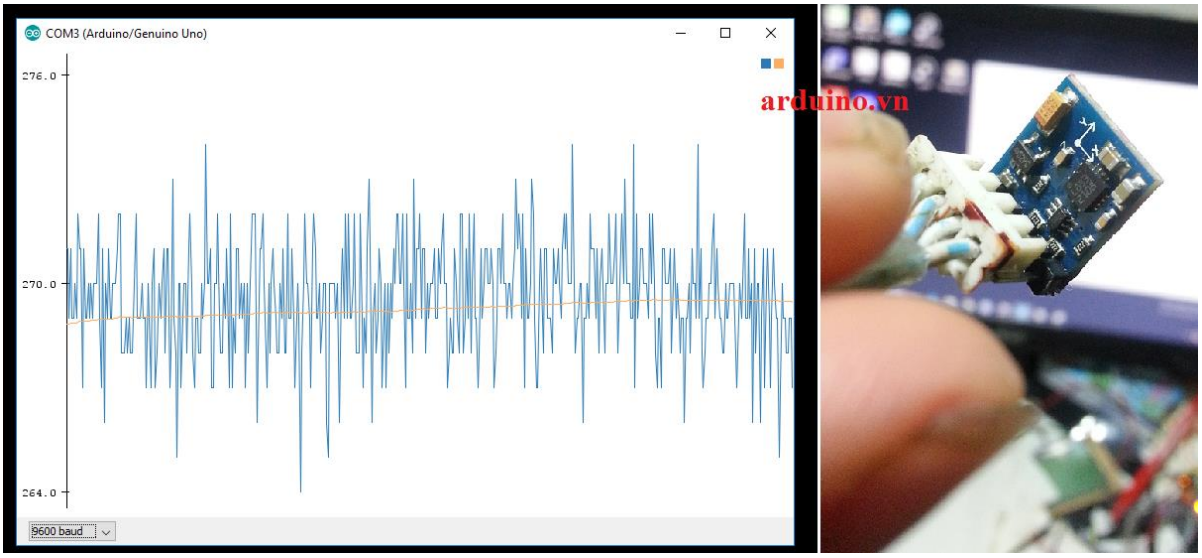
https://malcolmmielle.wordpress.com/...

```
/*
* Need to tweak value of sensor and process noise
* arguments :
* process noise covariance
* measurement noise covariance
* estimation error covariance */

class Kalman_Filter_Distance {
protected:
      double _q; //process noise covariance
      double _q_init;
      double _r; //measurement noise covariance
      double _r_init;
      double _x; //value
```

```
    double _p; //estimation error covariance
    double _p_init;
    double _k; //kalman gain

public:
    /*
* Need to tweak value of sensor and process noise
* arguments :
* process noise covariance
* measurement noise covariance
* estimation error covariance */
    Kalman_Filter_Distance(double q, double r, double p)
        : _q(q)
        , _q_init(q)
        , _r(r)
        , _r_init(0)
        , _x(0)
        , _p(p)
        , _p_init(p)
        , _k(_p / (_p + _r)){};
    virtual void init(double x) { _x = x; }
    virtual void setProcessNoiseCovariance(double i)
    {
        _q = i;
        _q_init = i;
    }
    virtual void setMeasurementNoiseCovariance(double i)
    {
        _r = i;
        _r_init = i;
    }
    virtual void setEstimatiomErrorCovariance(double i)
    {
        _p = i;
        _p_init = i;
    }
    virtual double kalmanUpdate(double measurement);
    virtual void reset()
    {
        _q = _q_init;
        _r = _r_init;
        _p = _p_init;
    };
};

double Kalman_Filter_Distance::kalmanUpdate(double measurement)
{
    //prediction update
    //omit _x = _x
    _p = _p + _q;

    //measurement update
    _k = _p / (_p + _r);
    _x = _x + _k * (measurement - _x);
```

```
    _p = (1 - _k) * _p;

    return _x;
}
```

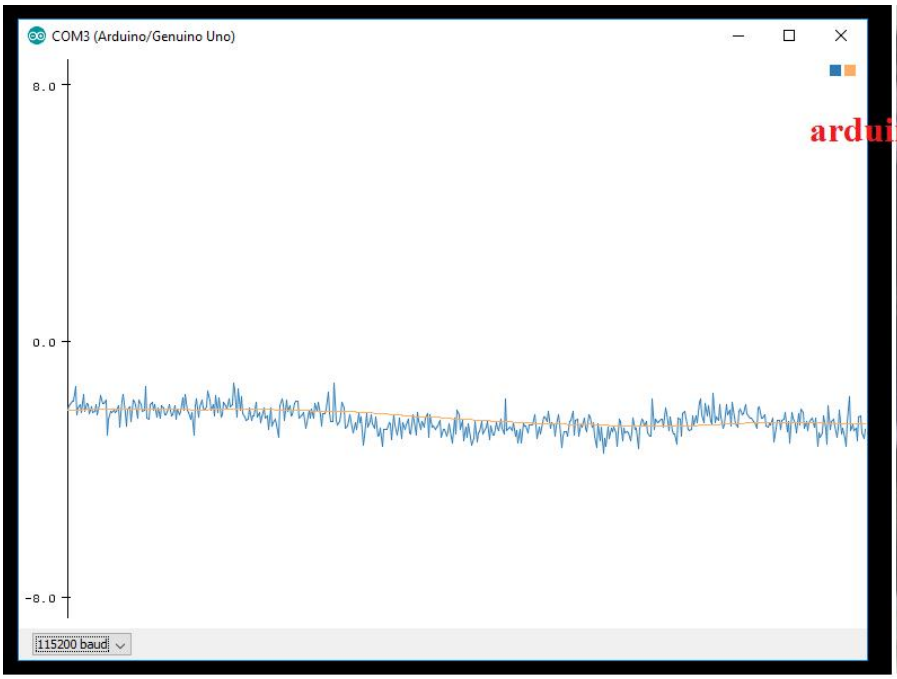
5




```
#include <Wire.h>
#include <HMC5883L.h>

#include <SimpleKalmanFilter.h>
SimpleKalmanFilter pressureKalmanFilter(1, 1, 0.001);
HMC5883L compass;

float x_kalman;
void checkSettings()
 {
 }
 void setup()
 {
    Serial.begin(9600);

    // Initialize HMC5883L
    Serial.println("Initialize HMC5883L");
    while (!compass.begin()) {
        Serial.println("Could not find a valid HMC5883L sensor, check wiring!");
        delay(500);
    }

    compass.setRange(HMC5883L_RANGE_1_3GA);
    compass.setMeasurementMode(HMC5883L_CONTINOUS);
    compass.setDataRate(HMC5883L_DATARATE_15HZ);
```

```
26.      compass.setSamples(HMC5883L_SAMPLES_1);
27.
28.      // Check settings
29.      checkSettings();
30.  }
31.
32.  void loop()
33.  {
34.      Vector raw = compass.readRaw();
35.      Vector norm = compass.readNormalize();
36.      x_kalman = pressureKalmanFilter.updateEstimate(raw.XAxis);
37.      Serial.print(raw.XAxis);
38.      Serial.print(",");
39.
40.      Serial.print(x_kalman, 3);
41.      Serial.println();
42.      delay(100);
43.  }
```

Test thực tế với cảm biến áp suất

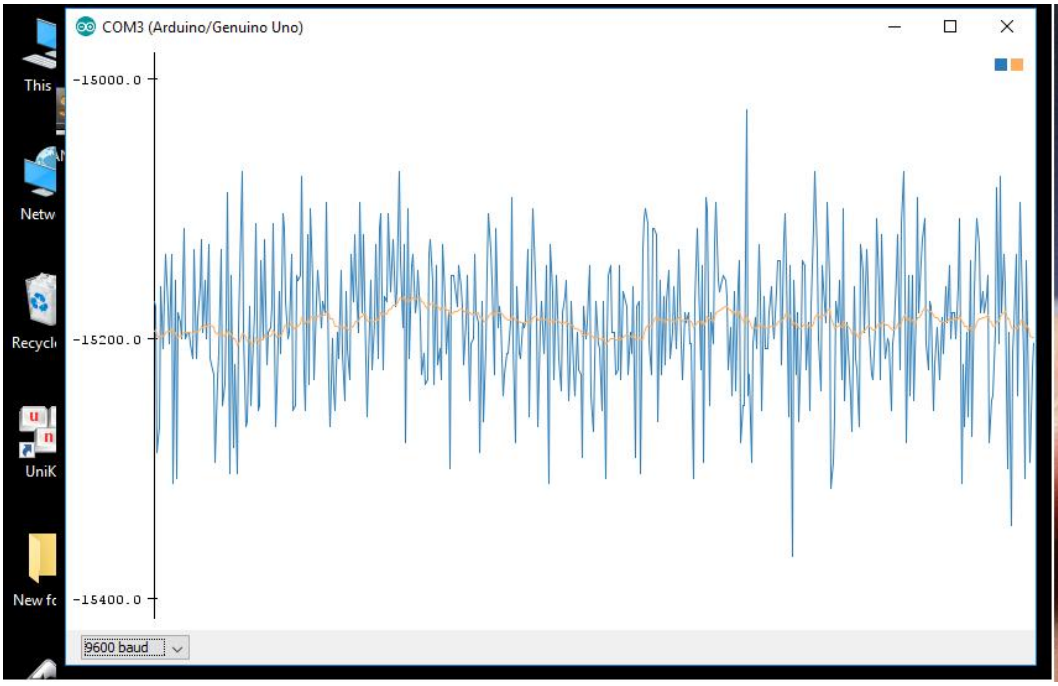



```
1. #include <SimpleKalmanFilter.h>
2. #include <SFE_BMP180.h>
3.
4. /*
5.  This sample code demonstrates how the SimpleKalmanFilter object can
   be used with a
6.  pressure sensor to smooth a set of altitude measurements.
7.  This example needs a BMP180 barometric sensor as a data source.
8.  https://www.sparkfun.com/products/11824
9.
```

```
 SimpleKalmanFilter(e_mea, e_est, q);
 e_mea: Measurement Uncertainty
 e_est: Estimation Uncertainty
 q: Process Noise
 */
SimpleKalmanFilter pressureKalmanFilter(1, 1, 0.01);

SFE_BMP180 pressure;

// Serial output refresh time
const long SERIAL_REFRESH_TIME = 100;
long refresh_time;

float baseline; // baseline pressure

double getPressure()
{
    char status;
    double T, P;
    status = pressure.startTemperature();
    if (status != 0) {
        delay(status);
        status = pressure.getTemperature(T);
        if (status != 0) {
            status = pressure.startPressure(3);
            if (status != 0) {
                delay(status);
                status = pressure.getPressure(P, T);
                if (status != 0) {
                    return (P);
                }
            }
        }
    }
}

void setup()
{

    Serial.begin(115200);

    // BMP180 Pressure sensor start
    if (!pressure.begin()) {
        Serial.println("BMP180 Pressure Sensor Error!");
        while (1)
            ; // Pause forever.
    }
    baseline = getPressure();
}

void loop()
{

    float p = getPressure();
```

```
64.      float altitude = pressure.altitude(p, baseline);
65.      float estimated_altitude =
   pressureKalmanFilter.updateEstimate(altitude);
66.
67.      if (millis() > refresh_time) {
68.          Serial.print(altitude, 6);
69.          Serial.print(",");
70.          Serial.print(estimated_altitude, 6);
71.          Serial.println();
72.          refresh_time = millis() + SERIAL_REFRESH_TIME;
73.      }
74.  }
```

Test thư viện với cảm biến gia tốc



```
1. #include <SimpleKalmanFilter.h>
2. #include <Wire.h>
3. #include "I2C.h"
4. #define RESTRICT_PITCH
5. SimpleKalmanFilter simpleKalmanFilter(1, 1, 0.001);
6.
7. uint8_t i2cData[14]; // Buffer for I2C data
8.
9. int16_t accX;
10.  float accX_kalman;
11.
12.  void setup()
13.  {
14.      Serial.begin(9600);
15.      Wire.begin();
16.  #if ARDUINO >= 157
```

```
    Wire.setClock(400000UL); // Set I2C frequency to 400kHz
#else
    TWBR = ((F_CPU / 400000UL) - 16) / 2; // Set I2C frequency to 400kHz
#endif

    i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz
    i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz Gyro filtering, 8 KHz sampling
    i2cData[2] = 0x00; // Set Gyro Full Scale Range to ±250deg/s
    i2cData[3] = 0x03; // Set Accelerometer Full Scale Range to ±16g
    while (i2cWrite(0x19, i2cData, 4, false))
        ; // Write to all four registers at once
    while (i2cWrite2(0x6B, 0x01, true))
        ; // PLL with X axis gyroscope reference and disable sleep mode
}
void loop()
{

    while (i2cRead(0x3B, i2cData, 14)) {
        ;
    }
    accX = (int16_t)((i2cData[0] << 8) | i2cData[1]);
    accX_kalman = simpleKalmanFilter.updateEstimate((float)accX);
    Serial.print(accX);
    Serial.print(",");
    Serial.print(accX_kalman, 3);
    Serial.println();
}
```