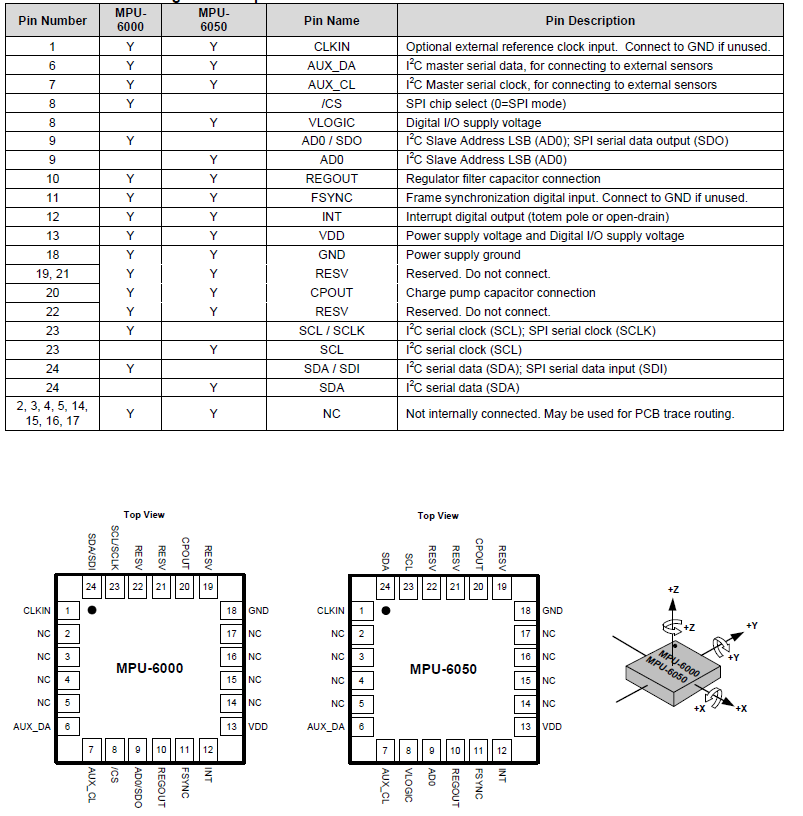
**MPU6050 for drone (cleanflight)**

[proadkim@naver.com](mailto:proadkim@naver.com) (1027.10.29)

Cleanflight에는 MPU6050 가속도 자이로 센서가 있습니다. MPU6050모듈은 가속도 자이로 센서를 이용하여 드론의 기울어진 정도와 회전한 정도를 알려줍니다. MPU6050 모듈은 가속도 3축, 자이로 3축, 온도(1% 오차)에 대한 총 7가지 센서 값을 제공합니다.



7.1 Pin out and signal description



**Accel의 Y를 드론의 앞 방향이라고 하면 Gyro의 Y회전방향이 roll, X회전방향이 pitch, Z의 회전방향이 Yaw임.**

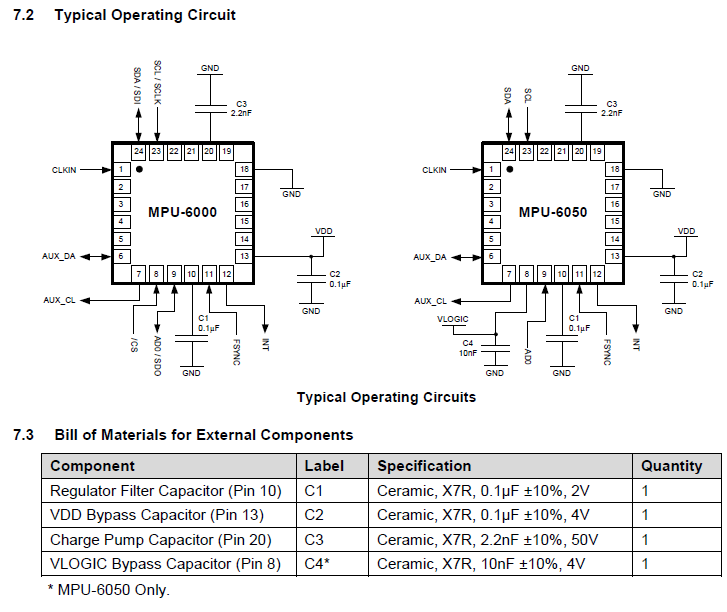
The MPU-60X0 has an auxiliary I2C bus for communicating to an off-chip 3-Axis digital output magnetometer or other sensors.

**Flexible VLOGIC reference voltage supports multiple I2C interface voltages (MPU-6050 only) (For power supply flexibility) => 1.8V+/-5% or VDD**

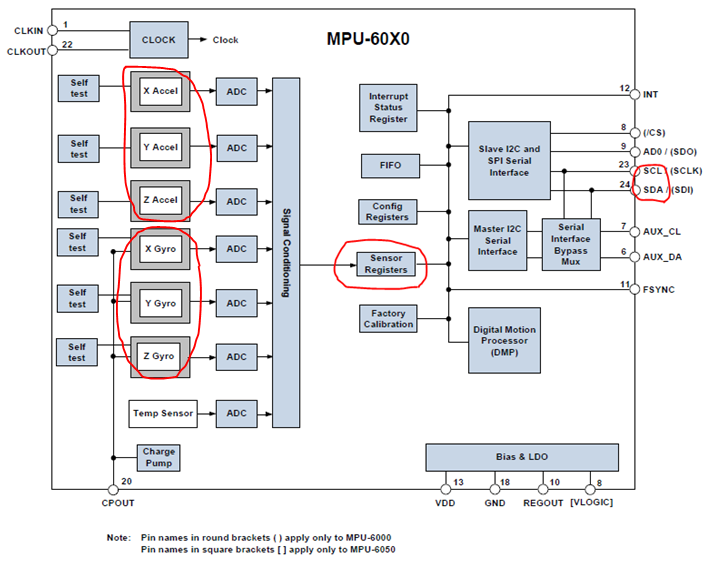


The bias and LDO section generates the internal supply and the reference voltages and currents required by the MPU-60X0. Its two inputs are an unregulated VDD of 2.375 to 3.46V and a VLOGIC logic reference supply voltage of 1.71V to VDD (MPU-6050 only). **The LDO output is bypassed by a capacitor at REGOUT**

An **on-board charge pump generates the high voltage required for the MEMS oscillators**. Its output is bypassed by a capacitor at CPOUT.



7.5 block diagram



**5.1 Gyroscope Features**

The triple-axis MEMS gyroscope in the MPU-60X0 includes a wide range of features:

Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with **a user-programmable full-scale range of ±250, ±500, ±1000, and ±2000°/sec for precision tracking of both fast and slow motions**

**External sync signal connected to the FSYNC pin supports image, video and GPS synchronization**

Integrated 16-bit ADCs enable simultaneous sampling of gyros

Enhanced bias and sensitivity temperature stability reduces the need for user calibration

Improved low-frequency noise performance

**Digitally-programmable low-pass filter**

Gyroscope operating current: 3.6mA

Standby current: 5μA

Factory calibrated sensitivity scale factor

User self-test

\* The ADC sample rate is programmable from 8,000 samples per second, down to 3.9 samples per second, and user-selectable low-pass filters enable a wide range of cut-off frequencies.

**5.2 Accelerometer Features**

The triple-axis MEMS accelerometer in MPU-60X0 includes a wide range of features:

Digital-output triple-axis accelerometer with a **programmable full scale range of ±2*g*, ±4*g*, ±8*g* and ±16*g*****for precision tracking of both fast and slow motions**

Integrated 16-bit ADCs enable simultaneous sampling of accelerometers while requiring no external multiplexer

Accelerometer normal operating current: 500μA

Low power accelerometer mode current: 10μA at 1.25Hz, 20μA at 5Hz, 60μA at 20Hz, 110μA at 40Hz

Orientation detection and signaling

Tap detection

User-programmable interrupts

High-G interrupt

User self-test

**5.3 Additional Features**

The MPU-60X0 includes the following additional features:

**9-Axis MotionFusion by the on-chip Digital Motion Processor (DMP)**

Auxiliary master I2C bus for reading data from external sensors (e.g., magnetometer)

3.9mA operating current when all 6 motion sensing axes and the DMP are enabled

VDD supply voltage range of 2.375V-3.46V

**Flexible VLOGIC reference voltage supports multiple I2C interface voltages (MPU-6050 only)**

Smallest and thinnest QFN package for portable devices: 4x4x0.9mm

Minimal cross-axis sensitivity between the accelerometer and gyroscope axes

1024 byte FIFO buffer reduces power consumption by allowing host processor to read the data in bursts and then go into a low-power mode as the MPU collects more data

Digital-output temperature sensor

User-programmable digital filters for gyroscope, accelerometer, and temp sensor

10,000 *g* shock tolerant

400kHz Fast Mode I2C for communicating with all registers

**5.4 MotionProcessing**

Internal Digital Motion Processing™ (DMP™) engine supports **3D MotionProcessing and gesture recognition algorithms**

The MPU-60X0 collects gyroscope and accelerometer data while synchronizing data sampling at a user defined rate. The total dataset obtained by the MPU-60X0 includes 3-Axis gyroscope data, 3-Axis accelerometer data, and temperature data. The MPU’s calculated output to the system processor can also include heading data from a digital 3-axis third party magnetometer.

The FIFO buffers the complete data set, reducing timing requirements on the system processor by allowing the processor burst read the FIFO data. After burst reading the FIFO data, the system processor can save power by entering a low-power sleep mode while the MPU collects more data.

**Programmable interrupt supports features such as gesture recognition, panning, zooming, scrolling, tap detection, and shake detection**

**Digitally-programmable low-pass filters**

**Low-power pedometer functionality allows the host processor to sleep while the DMP maintains the step count.**

\* The DMP acquires data from accelerometers, gyroscopes, and additional 3rd party sensors such as magnetometers, and processes the data. **The resulting data can be read from the DMP’s registers, or can be buffered in a FIFO**. The DMP has access to one of the MPU’s external pins, which can be used for generating interrupts.

**5.5 Clocking**

On-chip timing generator ±1% frequency variation over full temperature range

Optional external clock inputs of 32.768kHz or 19.2MHz

An on-chip 1024 Byte FIFO buffer helps **lower system power consumption** by allowing the system processor to read the sensor data in bursts and then enter a low-power mode as the MPU collects more data. By providing an integrated MotionFusion output, the DMP in the MPU-60X0 offloads the intensive MotionProcessing computation requirements from the system processor, minimizing the need for frequent polling of the motion sensor output.

The sensor data registers contain the latest gyro, accelerometer, auxiliary sensor, and temperature measurement data. They are read-only registers, and are accessed via the serial interface. Data from these registers **may be read anytime**. **However, the interrupt function may be used to determine when new data is available.**

Interrupt functionality is configured via the Interrupt Configuration register. Items that are configurable include the INT pin configuration, the interrupt latching and clearing method, and triggers for the interrupt. Items that can trigger an interrupt are (1) Clock generator locked to new reference oscillator (used when switching clock sources); (2) new data is available to be read (from the FIFO and Data registers); (3) accelerometer event interrupts; and (4) the MPU-60X0 did not receive an acknowledge from an auxiliary sensor on the secondary I2C bus. The interrupt status can be read from the Interrupt Status register.

Internal Register map



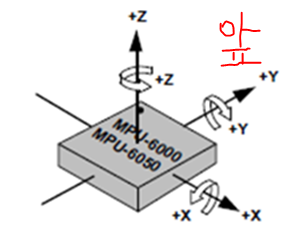
// 자이로/가속도 센서에 관한 blog

<http://blog.naver.com/helicon9/221088015521>

<http://blog.naver.com/helicon9/221028548772>

출처 : 아두이노 드론 만들고 직접 코딩하기

X, Y, Z 축에 대한 가속도와 자이로 값이 각각의 ADC block을 거쳐 센서 레지스터에 저장되고, FC controller와는 I2C로 통신하여 전달됩니다.



**Accel의 Y를 드론의 앞 방향이라고 하면 Gyro의 Y회전방향이 roll, X회전방향이 pitch, Z의 회전방향이 Yaw임.**

직선 3축과 곡선 3축 총 6개의 축이 있는데 직선 3축은 기울기(가속도) 센서가 사용하며, 곡선 3축은 자이로 센서가 사용합니다. 직선 3축은 각 축에 대해 중력방향을 기준으로 센서의 기울어진 정도를 측정할 때 사용하고, 곡선 3축은 각 축 진행방향에 대한 회전 정도를 측정할 때 사용합니다.

직선 축의 +X, +Y, +Z는 가속도 센서의 +값의 기준이 됩니다. 예를 들어, 센서가 정적인 상태에서 +X가 중력 방향을 보게 되면 가속도 센서 X\_Accel은 양수 값을 갖습니다. MPU6050 센서를 수평면에 두었을 때, 직선 축 +Z는 중력과 정반대 방향을 보게 되며, 가속도 센서 Z\_Accel은 양수값을 가지게 됩니다.

곡선축의 +X, +Y, +Z는 센서가 회전할 경우에 자이로 센서의 +값의 기준이 됩니다. 예를 들어, 센서가 +X 방향으로 돌면 자이로(각속도 또는 회전속도) 센서 X\_Gyro는 양수 값을 갖습니다.

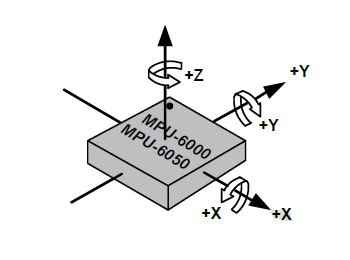
드론이 앞으로 기울어진 상태와 오른쪽으로 기울어진 상태에 대해서 생각해 보겠습니다. 드론이 앞으로 기울어지면 직선 축 +Y는 중력방향을 향하게 되므로 기울기 센서 Y\_Accel은 음수값을 갖게 됩니다. 드론이 오른쪽으로 기울어지면 직선 축 +X는 중력 방향을 향하게 되므로 기울기 센서 X\_Accel은 음수값을 갖습니다.

드론이 앞으로 기울면서 회전이 발생할 경우 곡선 축 +X 반대 방향으로 돌게 됩니다. 따라서 자이로 센서 X\_Gyro는 음수값을 가지고 되고 드론이 오른쪽으로 기울면서 회전이 발생할 경우 곡선 축 +Y 방향으로 돌게 됩니다. 따라서 자이로 센서 Y\_Gyro는 양수 값을 갖습니다.

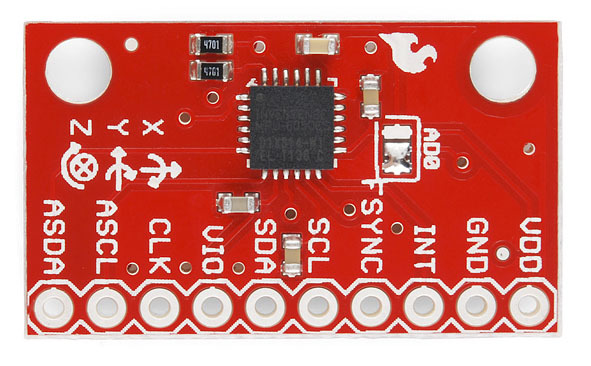
도론이 앞으로 회전하면서 기울어질 경우, 곡선 축 +X방향의 반대로 회전이 발생하면서 X\_Gyro는 음수 값을 갖게 되며, 직선 축 +Y는 중력 방향을 가리키면서 Y\_Accel은 음수 값을 갖습니다. 드론이 오른쪽으로 회전하면서 기울어질 경우, 곡선 축 +Y 방향으로 회전이 발생하면서 Y\_Gyro는 양수 값을 갖게 되며, 직선 축 +X는 중력 방향을 가리키면서 X\_Accel은 음수 값을 갖습니다.

출처 : <http://alnova2.tistory.com/1084>

MPU-6050 은 3축의 자이로와 3축의 가속도를 센싱 할 수 있다. 다음의 그림을 보면

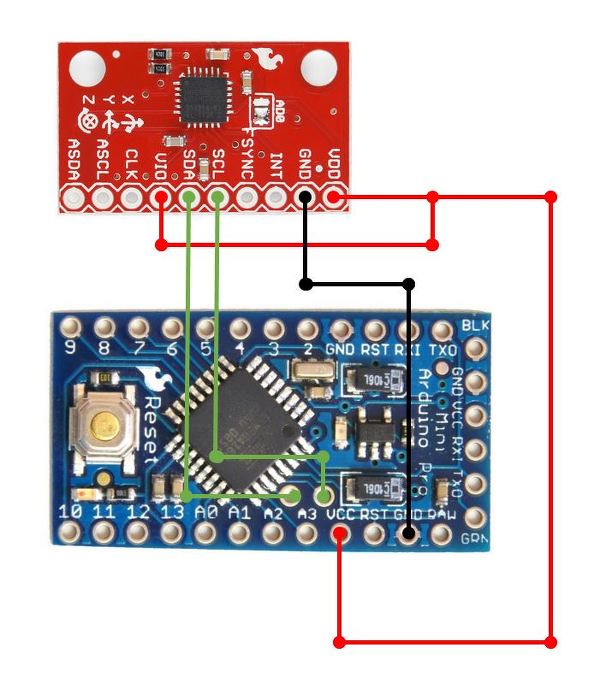


X, Y, Z 방향으로 기울기와 가속도를 알 수 있다. 가속도 센싱은 중력 방향과의 반대 방향일 떄 + 가 되고 중력 방향과 같은 방향일떄 -가 된다. 자이로는 회전하는 방향에 따라서 +/-가 된다.  본 포스팅에서 사용할 MPU6050 모듈은 Sparkfun에서 나온 모듈이다.



그림에서 보면 X,Y,Z 에 대한 방향 및 회전 방향이 그려져 있는데, 화살표 방향이 중력 방향과 반대 방향이면 +, 중력 방향이면 -가 되며, 회전 회살표 방향으로 회전할 경우 자이로 값이 +, 그렇지 않은 경우 -가 된다.

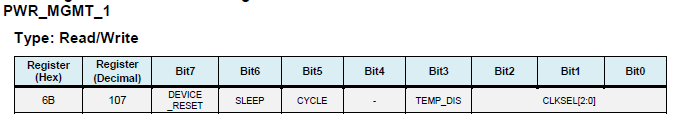
 Arduino Pro Mini와 Sparkfun MPU6050 모듈은 다음과 같이 연결한다.



 arduino pro mini는 A4 가 SDA, A5가 SCL이 된다. Arduino 예제는

<http://playground.arduino.cc/Main/MPU-6050>

에 나와 있는 첫번째 코드를 이용한다.



This register allows the user to configure the power mode and clock source. It also provides a bit for resetting the entire device, and a bit for disabling the temperature sensor.

**Parameters:**

*DEVICE\_RESET*

When set to 1, this bit resets all internal registers to their default values. The bit automatically clears to 0 once the reset is done. The default values for each register can be found in Section 3.

*SLEEP*

When set to 1, this bit puts the MPU-60X0 into sleep mode.

*CYCLE*

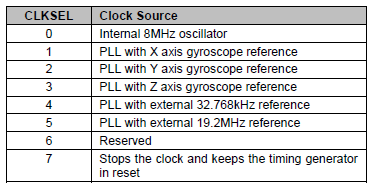
When this bit is set to 1 and *SLEEP* is disabled, the MPU-60X0 will cycle between sleep mode and waking up to take a single sample of data from active sensors at a rate determined by *LP\_WAKE\_CTRL* (register 108).

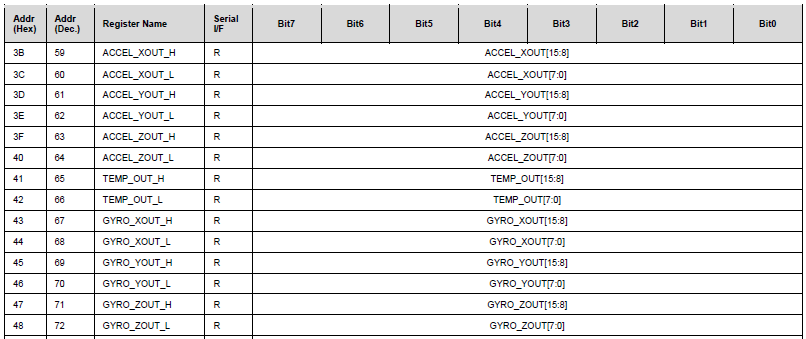
*TEMP\_DIS*

When set to 1, this bit disables the temperature sensor.

*CLKSEL*

3-bit unsigned value. Specifies the clock source of the device. Upon power up, the MPU-60X0 clock source defaults to the internal oscillator. However, it is highly recommended that the device be configured to use one of the gyroscopes (or an external clock source) as the clock reference for improved stability. The clock source can be selected according to the following table.



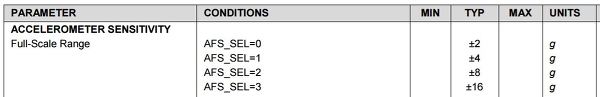


|  |
| --- |
| // MPU-6050 Short Example Sketch  // By Arduino User JohnChi  // August 17, 2014  // Public Domain  #include<Wire.h>    **const int MPU\_addr=0x68;  // I2C address of the MPU-6050**  int16\_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;  void setup(){    Wire.begin();  Wire.beginTransmission(MPU\_addr);    **Wire.write(0x6B);  // PWR\_MGMT\_1 register**    Wire.write(0);     // set to zero (wakes up the MPU-6050)    Wire.endTransmission(true);    Serial.begin(115200);  }  void loop(){    Wire.beginTransmission(MPU\_addr);  **Wire.write(0x3B);  // starting with register 0x3B (ACCEL\_XOUT\_H)**    Wire.endTransmission(false);    Wire.requestFrom(MPU\_addr,14,true);  // request a total of 14 registers    AcX=Wire.read()<<8|Wire.read();  // 0x3B (ACCEL\_XOUT\_H) & 0x3C (ACCEL\_XOUT\_L)    AcY=Wire.read()<<8|Wire.read();  // 0x3D (ACCEL\_YOUT\_H) & 0x3E (ACCEL\_YOUT\_L)    AcZ=Wire.read()<<8|Wire.read();  // 0x3F (ACCEL\_ZOUT\_H) & 0x40 (ACCEL\_ZOUT\_L)    Tmp=Wire.read()<<8|Wire.read();  // 0x41 (TEMP\_OUT\_H) & 0x42 (TEMP\_OUT\_L)    GyX=Wire.read()<<8|Wire.read();  // 0x43 (GYRO\_XOUT\_H) & 0x44 (GYRO\_XOUT\_L)    GyY=Wire.read()<<8|Wire.read();  // 0x45 (GYRO\_YOUT\_H) & 0x46 (GYRO\_YOUT\_L)    GyZ=Wire.read()<<8|Wire.read();  // 0x47 (GYRO\_ZOUT\_H) & 0x48 (GYRO\_ZOUT\_L)    Serial.print("AcX = "); Serial.print(AcX);    Serial.print(" | AcY = "); Serial.print(AcY);    Serial.print(" | AcZ = "); Serial.print(AcZ);    Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53);  //equation for temperature in degrees C from datasheet    Serial.print(" | GyX = "); Serial.print(GyX);    Serial.print(" | GyY = "); Serial.print(GyY);    Serial.print(" | GyZ = "); Serial.println(GyZ);    delay(333);  } |

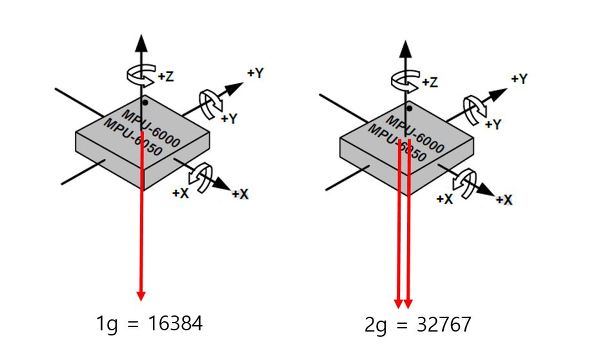
 실행 시키고 X,Y,Z 화살표 방향을 중력 방향을 중심으로 회전 시켜 보자. 자이로도 회전 방향으로 +, -로 변화하는 것을 확인 가능하다.

 MPU6050 센서를 통해서 들어오는 값은 16비트 크기를 가진다. 즉 각 값은 int16\_t 의 값을 가진다. 16비트로 최대 -32768 에서 32767 의 정수 값을 표현 가능하다. 값의 의미는 데이터 Sheet를 참조해서 설명한다. (<https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>)

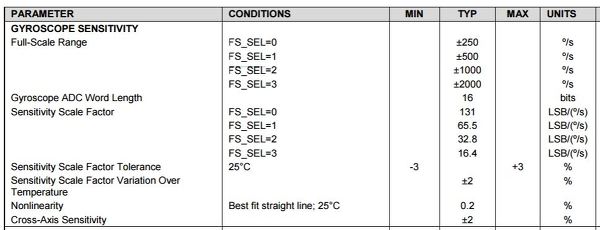
[가속도 값의 의미]



MPU6050 datasheet를 보면 AFS\_SEL 레지스터 값의 설정에 따라 측정되는 값이 달라짐을 알 수 있다. AFS\_SEL=0일 경우 +-2g 까지 표현이 되는데 -2g는 -32768에 +2g는 +32767 에 매핑이 된다. 1g는 16384 가 되며 이는 중력 가속도와 동일한 값이다. 예를 들어 수평에서 움직이지 않고 가만히 있으면 +Z 값은 16384가 되는 것이다. +z 방향의 반대로 중력가속도만큼 가속도를 받으면 32767 이 된다.



[자이로 값의 의미]



자이로 센서의 값도 -32768에서 32767 값을 가질 수 있다. FS\_SEL 레지스터가 0일 경우 -250 에서 +250까지 측정되며 이 값은 -32768에서 32767까지 매핑된다. **1초 동안 일정한 속도로 1도 회전하였다고 한다면 250도/s 는 32767 이므로 1도/s는 (32767/250) = 131이 된다.**

<http://blog.naver.com/dkwltmdgus/220429505827>

드론에서의 가속도 센서는 드론의 수평을 잡아주는 역할을 합니다., 중력의 힘이 각각의 X,Y,Z에 얼마나 힘이 작용하느냐를 통해 수평을 맞추는 것입니다. 가속도 센서의 장점으로는 변화하는 값의 오차가 누적되지 않습니다. 하지만 노이즈와 이동, 진동에 취약하다는 치명적인 단점이 있습니다. 드론은 계속이동하므로 가속도 센서 하나만으로는 드론의 기울기를 정확하게 측정할 수 없습니다. 이 단점을 보완하기 위해 자이로 센서를 사용합니다. 자이로 센서는 물체가 X, Y, Z축 방향으로 얼마나 기울었는지와 기운쪽의 방향을 알수 있습니다. 드론제작에 적용시켜보자면 자이로 센서는 위 2가지 정보를 가지고 적분하여 모터의 회전량을 알수 있습니다. 그래서 장점으로는 pitch, roll회전을 원하는 만큼 할 수 있습니다. 단점으로는 적분하는 과정에서 센서의 노의즈로 인하여 오차가 생기고 계속해서 누적됩니다. 위 현상을 드리프트라고 합니다. 위 드리프트 현상이 지속되면 얼마나 기울었는지 정확한 값을 측정 못하게 됩니다. 또한 자이로 센서는 초기 값을 잡아주지 않으면 정확하게 수평을 못 맞춥니다. 그래서 자이로센서는 초기값과 오차값이 일정하게 유지되는 가속도 센서와 같이 사용합니다. 하지만 가속도, 자이로 센서로는 pitch, roll, yaw값들의 회전량밖에 못구합니다. 한마디로 어디가 머리인지를 모릅니다. 그래서 머리가 어딘지를 알기 위해서 자기장 센서, 즉, 지자계 센서를 사용합니다. 장점으로는 어디가 앞인지를 알수 있지만 자석에 아주 취약합니다.

//------------

<http://www.avrfreaks.net/forum/tut-hard-gyros-and-accelerometers-basics?name=PNphpBB2&file=viewtopic&t=89294>

a) So what are "Degrees of Freedom" (DoF)?

This refers to the number of independent dimensions that an Inertial Measurement Unit (IMU) is capable of measuring. There are three linear dimensions (e.g. acceleration) and three angular dimensions (e.g. angular rate - gyros). Hence, 6DoF.

For a "fully observable" IMU, you need to be able to measure all 6 degrees of freedom. Otherwise, you'll need to "guess" the other motion parameters.

Some units of "9DoF" or "8-axis". Usually, this refers to an IMU that also contains a 2-axis or 3-axis magnetometer. It's marketing talk, not true "degrees of freedom" as such, but the compass \*is\* useful (so long as you know there's no metal nearby!)

//------------

<http://www.geekmomprojects.com/gyroscopes-and-accelerometers-on-a-chip/>

It is important to note that the accelerometer results provide **accurate orientation angles** as long as gravity is the only force acting on the sensor.  However, when moving and rotating the sensor, we are applying forces to it, which causes the measurements to **fluctuate**.  The net result is that accelerometer data tends to be very noisy, with brief but significant perturbations.  If these can be averaged out, the accelerometer provides accurate results over timescales longer than the perturbations.

Computing orientation from the gyroscope sensor is different, since the gyroscope measures ***angular velocity (the rate of change in orientation angle),* not angular orientation itself**.  To compute the orientation**, we must first initialize the sensor position with a known value (possibly from the accelerometer)**, then measure the angular velocity (ω) around the X, Y and Z axes at measured intervals (Δt).   **Then ω × Δt = change in angle**.  The new orientation angle will be the original angle plus this change.  **The problem with this approach is that we are *integrating***– adding up many small computed intervals – to find orientation.  Repeatedly adding up increments of  ω × Δt will result in small systematic errors becoming magnified over time.  This is the cause of **gyroscopic drift**, and over long timescales the gyroscope data will become increasingly inaccurate.

As explained above, both the accelerometer and gyroscope data are prone to systematic errors.  The accelerometer provides accurate data over the long term, but is noisy in the short term.  The gyroscope provides accurate data about changing orientation in the short term, but the necessary integration causes the results to drift over longer time scales.

The solution to these problems is to fuse the accelerometer and gyroscope data together in such a way that the errors cancel out.  The standard method of combining these two inputs is with a [*Kalman Filter*](http://en.wikipedia.org/wiki/Kalman_filter)*,* which is quite a complex methodology.  Fortunately, there is a simpler approximation for combining these two data types, called a *Complementary Filter*.  Far better explanations than I could ever write are given [here](http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=89294) (AVRFreaks) and [here](http://web.mit.edu/~jinstone/Public/filter.pdf) (Shane Colton), but the approximate formula to combine the accelerometer and gyroscope data is:

**Filtered Angle = α × (Gyroscope Angle) + (1 − α) × (Accelerometer Angle)**where

**α = τ/(τ + Δt)**and**(Gyroscope Angle) = (Last Measured Filtered Angle) + ω×Δt**

**Δt** = sampling rate, **τ**= time constant greater than timescale of typical accelerometer noise

//----------

출처 : <http://alnova2.tistory.com/1085>

이번 포스팅에서는 센서에서 측정 되는 값에서 노이즈를 제거하기 위하여 필터를 적용하는 방법에 대해서 이야기 한다.

 MPU6050 센서 값을 Processing 이라는 툴을 가지고 Visual 하게 보여주는 프로젝트가 있다. 매우 유명한 사이트로 한번 쯤 가서 내용과 영상을 봐도 좋다.

<http://www.geekmomprojects.com/gyroscopes-and-accelerometers-on-a-chip/>

 MPU6050 센서는 가속도와 자이로 값을 측정하는데, 가속도는 중력 가속도에 대한 기울어진 각도를 측정한다.

 가속도 센서는 움직이거나 회전시킬 때 민감하게 반응하기 때문에 노이즈가 심하가 나타난다. 특히 드론에서 모터 회전은 고속의 미세 움직임을 발생시키는데, 가속도 센서에서는 이런 노이즈에 취약하다. 가속도 센서는 정적 상태에서 각속도를 비교적 정확하게 측정 가능하지만 움직임에 따른 노이즈에 취약하다고 볼 수 있다.

 자이로 센서는 센서의 움직임에 대한 각속도를 측정하게 되는데, +X, +Y, +Z 방향에 대한 회전 각속도를 측정하게 되는 것이다. 각속도는 시간에 누적되는데, 반복적인 누적에 따라 오차가 발생하게 된다. 즉 시간이 지날 수록 오차가 많이 발생하게 되는 것이다. 결과적으로 자이로 센서는 짧은 시간에 대하여 정확한 데이터를 제공하지만 오랜 시간에 대한 데이터는 오류가 발생한다. (반복적 시간 누적에 따른 적분 값인데, 축적할 수록 값에 오류가 발생하는 현상으로 드리프트라 부른다.)

**즉 가속도 센서는 가만히 있으면 정확하지만 움직이면 노이즈에 취약하고, 평균값이 정확하며 자이로 센서는 순간 순간의 움직임은 정확하지만 장시간에 대하여 오류가 발생하고 평균값은 오류가 있는 두 센서의 보정이 필요하다.**

 이 두가지를 합쳐서 값을 보정하는 것을 상보 필터라고 한다. 위 Gekkmonproject 사이트에 보면 이 필터에 대한 계산식이 나온다.

|  |
| --- |
| **Filtered Angle = α × (Gyroscope Angle) + (1 − α) × (Accelerometer Angle)**  **α = τ/(τ + Δt)**  **(Gyroscope Angle) = (Last Measured Filtered Angle) + ω×Δt**  **Δt** = sampling rate, **τ**= time constant greater than timescale of typical accelerometer noise |

 위 사이트에서는  **α** = 0.96 으로 Sampling Rate는 0.04 초로 Time Constant 는 1 초로 하였다고 한다. 가속도 센서의 각도는 다음의 공식으로 구한다. (위 참조 사이트에서 가져 옴)

 다음은 위 사이트의 아두이노 코드 중 주요한 사항들을 분석해 보도록 하겠다.

1. calibrate\_sensors

|  |
| --- |
| void calibrate\_sensors() {    int                   num\_readings = 10;    float                 x\_accel = 0; **// pitch**    float                 y\_accel = 0; **// roll**    float                 z\_accel = 0; **// yaw**    float                 x\_gyro = 0; **// pitch**    float                 y\_gyro = 0; **// roll**    float                 z\_gyro = 0; **// yaw**    accel\_t\_gyro\_union    accel\_t\_gyro;      //Serial.println("Starting Calibration");    // Discard the first set of values read from the IMU    read\_gyro\_accel\_vals((uint8\_t \*) &accel\_t\_gyro);      // Read and average the raw values from the IMU    for (int i = 0; i < num\_readings; i++) {      read\_gyro\_accel\_vals((uint8\_t \*) &accel\_t\_gyro);      x\_accel += accel\_t\_gyro.value.x\_accel;      y\_accel += accel\_t\_gyro.value.y\_accel;      z\_accel += accel\_t\_gyro.value.z\_accel;      x\_gyro += accel\_t\_gyro.value.x\_gyro;      y\_gyro += accel\_t\_gyro.value.y\_gyro;      z\_gyro += accel\_t\_gyro.value.z\_gyro;      delay(100);    }    x\_accel /= num\_readings; // num\_readings번 읽어서 평균    y\_accel /= num\_readings; // num\_readings번 읽어서 평균    z\_accel /= num\_readings; // num\_readings번 읽어서 평균    x\_gyro /= num\_readings; // num\_readings번 읽어서 평균    y\_gyro /= num\_readings; // num\_readings번 읽어서 평균    z\_gyro /= num\_readings; // num\_readings번 읽어서 평균      // Store the raw calibration values globally    base\_x\_accel = x\_accel;    base\_y\_accel = y\_accel;    base\_z\_accel = z\_accel;    base\_x\_gyro = x\_gyro;    base\_y\_gyro = y\_gyro;    base\_z\_gyro = z\_gyro;      //Serial.println("Finishing Calibration");  } |

 가속도 및 자이로 센서의 초기 값을 조정하는 함수이다. 바닥에 그대로 두고, 가속도와 자이로의 평균 값을 구해서 초기 상태의 값을 calibration 한다

2. 자이로 센서를 각속도로 변환하기

 loop 코드에서 **자이로 센서의 값**을 각속도로 변환하는 코드가 있다.

|  |
| --- |
| // Convert gyro values to degrees/sec  float FS\_SEL = **131;** // 자이로 센서의 값도 -32768에서 32767 값을 가질 수 있다. FS\_SEL // 레지스터가 0일 경우 -250 에서 +250까지 측정되며 이 값은  // - 32768에서 32767까지 매핑된다. **1초 동안 일정한 속도로 1도  // 회전하였다고 한다면 250도/s 는 32767 이므로 1도/s는 (32767/250) =  // 131이 된다.**  **// 측정된 gyro 값에서 앞의 평균값을 빼고 131로 나누어 줌.**  float gyro\_x = (accel\_t\_gyro.value.**x\_gyro** - base\_x\_gyro) / FS\_SEL;  float gyro\_y = (accel\_t\_gyro.value.**y\_gyro** - base\_y\_gyro) / FS\_SEL;  float gyro\_z = (accel\_t\_gyro.value.**z\_gyro** - base\_z\_gyro) / FS\_SEL;  // Compute the (filtered) gyro angles  float dt = (t\_now - get\_last\_time()) / 1000.0;  **// gyro에서 측정된 값은 각속도 이므로 시간(dt)를 곱해주면 dt시간동안  // 돌아간 각도가 나옴. 그 값을 이전값과 더해줌**  float gyro\_angle\_x = gyro\_x\*dt + get\_last\_x\_angle();  float gyro\_angle\_y = gyro\_y\*dt + get\_last\_y\_angle();  float gyro\_angle\_z = gyro\_z\*dt + get\_last\_z\_angle();  // Compute the drifting gyro angles  float unfiltered\_gyro\_angle\_x = gyro\_x\*dt + get\_last\_gyro\_x\_angle();  float unfiltered\_gyro\_angle\_y = gyro\_y\*dt + get\_last\_gyro\_y\_angle();  float unfiltered\_gyro\_angle\_z = gyro\_z\*dt + get\_last\_gyro\_z\_angle(); |

 앞선 포스팅에서 MPU6050 의 각속도의 각 1도/Sec 은 131 이 된다고 하였다. 따라서 측정되는 값에서 기본 값을 빼고 131로 나누면 값이 나온다.

3. **가속도 센서**에 각 측정하기

|  |
| --- |
| // Get raw acceleration values  //float G\_CONVERT = 16384;  float accel\_x = accel\_t\_gyro.value.**x\_accel**;  float accel\_y = accel\_t\_gyro.value.**y\_accel**;  float accel\_z = accel\_t\_gyro.value.**z\_accel**;  // Get angle values from accelerometer  **// 가속도 센서로부터 나오는 값은 radian값이므로 degree로 바꾸려면 180/pi를 곱해줌**  float RADIANS\_TO\_DEGREES = 180 / 3.14159;  //  float accel\_vector\_length = sqrt(pow(accel\_x,2) + pow(accel\_y,2) + pow(accel\_z,2));  **float accel\_angle\_y = atan(-1 \* accel\_x / sqrt(pow(accel\_y, 2) + pow(accel\_z, 2)))\*RADIANS\_TO\_DEGREES;**  **float accel\_angle\_x = atan(accel\_y / sqrt(pow(accel\_x, 2) + pow(accel\_z, 2)))\*RADIANS\_TO\_DEGREES;**  **// 가속도 센서로 Z축의 틀어진 각도를 알수는 없으므로 0임.   float accel\_angle\_z = 0;** |

4. 상보필터 적용를 현재 드론에 대해 X, Y, Z 축으로 틀어진 각도를 알수 있음.

|  |
| --- |
| float alpha = 0.96;  float angle\_x = alpha\*gyro\_angle\_x + (1.0 - alpha)\*accel\_angle\_x;  float angle\_y = alpha\*gyro\_angle\_y + (1.0 - alpha)\*accel\_angle\_y;  float angle\_z = gyro\_angle\_z;  //Accelerometer doesn't give z-angle |

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | | --- | | 상보필터(mpu6050)  출처 : <http://blog.naver.com/roboholic84/220401407348> |   상보필터는 가속도 + 자이로의 형태를 가집니다. 가속도는 말그대로 가속도를 측정합니다.   가속도란 ? 시간에 따라 속도가 변하는 량 입니다.  다만 여기서 가속도 센서는 주로 어느정도 기울어 졌는지 '각도'측정에 사용됩니다. 이때 각도는 '중력 가속도'를 이용합니다. 못느낄지 몰라도 우리는 항상 중력을 받고 있습니다. 연직방향으로 작용하는 중력을 어느 각이 어느정도 받고있는지 이용하여 기울어진 정도를 측정합니다. 이때 **mpu6050 기준으로 z축과 다른 축의 값을 atan2 연산하여 연직방향에 비해 얼마나 기울어 졌는지를 측정할 수 있습니다**.  http://postfiles15.naver.net/20150625_222/roboholic84_1435224387857jchz2_PNG/mpu6050.png?type=w1  가속도값을 실제로 측정해보면 위 그림에서의 축과 반대방향으로 값이 증가합니다. 예를들어 센서를 위와같이 바닥에 두었을 때 z값이 16383, x, y가 0이 나옵니다. 문제는 이 가속도 값만 가지고 각도를 측정했을 경우 문제가 발생합니다. 어느 방향으로든 움직일 경우 그 방향으로 가속도가 발생하기 때문입니다. 이 가속도는 각도를 측정하고 있는 센서의 값을 이상하게 만듭니다. 누적되는 값은 아니지만 순간순간의 값이 중요할 경우 치명적입니다.  그러면 자이로 센서는 어떨까요? 자이로 센서는 '각속도'를 측정하는 센서로 특정 축을 기준으로 회전하는 속도를 나타냅니다. 자이로 값을 적분하면 우리는 각도를 얻을 수 있습니다. 제일 초기값을 0이라고 했을 경우 각 방향으로 얼마나 움직였는지를 알면 초기값과 비교하여 현재 각도를 알 수 있지요.다만 항상 정확한 각도를 측정할 순 없습니다. 센서에도 오차가 있기에 매 순간순간 조금씩 틀어지는 값이 발생하고, 소수점 아래 몇자리 이하는 버리면서 계산을 하기에 적분을 해서 값을 축적하면 축적할수록 실제 각도와 자이로센서가 측정한 각도가 틀어지는 것입니다. 이것을 드리프트라고 부릅니다. 초기에는 보다 정확한 값을 내어놓지만 시간이 지나면 지날수록 부정확해집니다.  가만히 있을땐 정상이지만 움직이면 값이 바뀌어 버리는 센서 가만히 있을땐 아무런 값이 없지만 움직이면 값이 변하는 센서  매 순간순간 데이터는 확신할 수 없지만 평균적인 값은 믿을 수 있는 센서 매 순간순간 값은 보다 정확하지만 평균적인 값은 믿을 수 없는 센서  이 두가지를 합쳐서 이상적인 값을 얻는것을 상보필터라 합니다.  상보필터는 기본적으로 아래와 같은 공식을 따르고 있습니다.  http://www.pieter-jan.com/images/equations/CompFilter_Eq.gif  angle은 출력할 각도, gyrData는 자이로값, dt는 적분할 시간, accData는 가속도를 이용한 각도 데이터입니다. angle이 2개가 있는데 오른쪽에 있는 angle은 이전 각도값을 의미합니다. 이 각도값은 매번 업데이트됩니다.  여기서 단위를 보면 이해가 쉽습니다. 위 공식에서 angle은 각도값(deg)이며 gyrData는 각속도(deg / sec), dt는 시간, accData는 각도(deg) 입니다.  서로 단위가 다르면 더할 수 없기에 단위를 통일해 주어야 합니다. 그렇기에 자이로값에 적분할 시간을 곱해주어 각도로 단위를 통일해준 후 더하는 것입니다. 그러면 실제 코드를 보도록 하겠습니다.   |  | | --- | | #include <Wire.h>  **#define mpu\_add 0x68 //mpu6050 address**  long ac\_x, ac\_y, ac\_z, gy\_x, gy\_y, gy\_z ; //acc, gyro data  double angle = 0, deg ; // angle, deg data  double dgy\_x ; //double type acc data  void setup() {    // put your setup code here, to run once:    Serial.begin(9600) ;  //set serial baud    Wire.begin() ;  //set I2C  **Wire.beginTransmission(mpu\_add) ;**  **Wire.write(0x6B) ;**  **Wire.write(0) ;**    Wire.endTransmission(true) ;  }  void loop() {    // put your main code here, to run repeatedly:    Wire.beginTransmission(mpu\_add) ; //get acc data  **Wire.write(0x3B) ; // accelerator**    Wire.endTransmission(false) ;    Wire.requestFrom(mpu\_add, 6, true) ;    ac\_x = Wire.read() << 8 | Wire.read() ;    ac\_y = Wire.read() << 8 | Wire.read() ;    ac\_z = Wire.read() << 8 | Wire.read() ;    Wire.beginTransmission(mpu\_add) ; //get gyro data  **Wire.write(0x43) ; // gyro**    Wire.endTransmission(false) ;    Wire.requestFrom(mpu\_add, 6, true) ;    gy\_x = Wire.read() << 8 | Wire.read() ;    gy\_y = Wire.read() << 8 | Wire.read() ;    gy\_z = Wire.read() << 8 | Wire.read() ;    deg = atan2(ac\_x, ac\_z) \* 180 / PI ;  //rad to deg  **dgy\_x = gy\_y / 131. ;  //16-bit data to 250 deg/sec**    angle = (0.95 \* (angle + (dgy\_x \* 0.001))) + (0.05 \* deg) ; //complementary filter    Serial.println(angle) ;  } |   쭉 내려가다가 보면 위에선 설명되지 않은 알 수 없는 두 연산이 있습니다.  deg = atan2(ac\_x, ac\_z) \* 180 / PI ;  //rad to deg  dgy\_x = gy\_y / 131. ;  //16-bit data to 250 deg/sec  위 두줄입니다. atan2는 값을 라디안값으로 출력하기에 이를 deg값으로 변환해주어야 합니다. 그렇기에 \* 180 / PI를 이용하여 변환해 주었습니다.  **자이로센서 값의 출력 범위는 ±250°/sec 인데 출력 값은 ±16383입니다. 이 말이 무슨말이냐? 만약 mpu-6050이 측정한 각속도가 250°/sec라면 16383이라는 값을 출력합니다.125°/sec일 경우는 약 8191을 출력할 것이구요. 이 출력된 값을 각속도로 변환해주는 과정이 /131.입니다.**  변환이 다 끝났으므로 위에서 봤던 상보필터 공식에 각각 값을 대입합니다.  그렇게 해서  angle = (0.95 \* (angle + (dgy\_x \* 0.001))) + (0.05 \* deg) ; //complementary filter  라는 공식이 나왔습니다. 위에 나왔던 공식과 바뀐게 있습니다. 바로 0.98이 0.95로, 0.02가 0.05로 바뀌었습니다. 이 값은 사용자가 임의대로 결정하는 파라미터값입니다. 가속도쪽에 곱해지는 값이 상대적으로 작고 자이로 값 + 이전값 쪽에 더해지는 값이 상대적으로 큽니다. 이는 가속도 값을 상대적으로 약화시킵니다. 실시간으로 변하는 값에 약한 가속도 값의 중요성을 줄이고 이전값 + 자이로 값을 더 신뢰함으로써 어느정도 순간적으로 변하는 값을 막아준 것입니다. 가속도 값의 영향력이 작다고 해도 지속적으로 영향을 끼치기에 이 값으로 인해 변하게 되는 것입니다.밸런싱 로봇의 경우 매 순간순간 값을 측정하고 움직여야 하므로 가속도보다는 자이로에 중심을 두고 계산하게 됩니다.하지만 여유롭게 계산해도 될 경우 예를들어 단순히 손으로 잡고 왔다갔다 하는 경우는 가속도 값의 비중을 더 주는것이 정확할 수 있습니다. 이는 상황에 맞게 설정해주어야 하며 이 상수값을 찾는것이 센서 튜닝의 첫번째 과정입니다.  이상으로 상보필터의 설명을 마치도록 하겠습니다. |