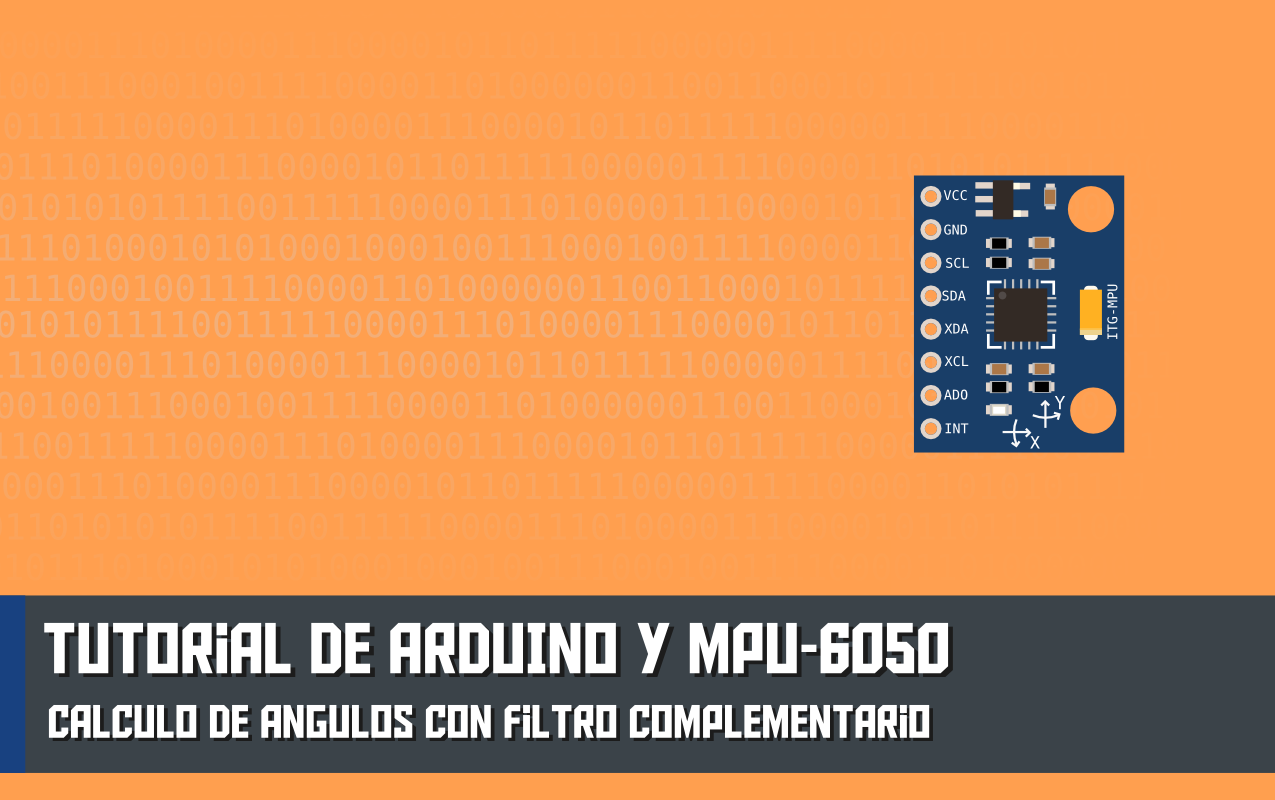
**Tutorial de Arduino y MPU-6050**

Posted on [octubre 15, 2014](http://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/) by [Tr4nsduc7or](http://robologs.net/author/7472616e73647563746f72/)

[](http://robologs.net/wp-content/uploads/2014/10/portada_IMU.png)

[Iniciando módulos de interacción social]

Saludos, humano. Al habla Transductor. ¿Alguna vez has intentado configurar una IMU con Arduino y obtener lecturas? Si tu respuesta es afirmativa, habrás notado que no es nada fácil. Muchas IMUs utilizan protocolos como I2C o SPI, dan valores extraños, requieren trigonometría… oh, ¿he mencionado ya que las IMUs dan siempre lecturas erróneas?

Si has acabado en esta página es que necesitas un empujón. Sabes que una IMU es un dispositivo electrónico. Y que no se come. Pero ahí se acabó. Probablemente no tienes ni idea de cómo conectarla a Arduino y ya no digamos obtener una lectura decente.

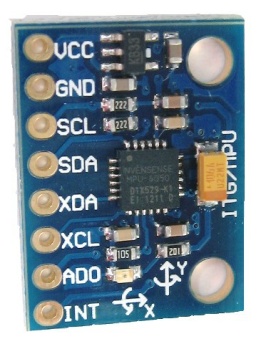
Así pues, has venido al lugar indicado. Transductor va a explicarte cómo conseguir unas lecturas precisas con una IMU MPU-6050 y Arduino. Depuraremos los valores, calcularemos el ángulo de inclinación y filtraremos el ruido con un Filtro Complementario (Complementary Filter).

Pero hay que empezar por el principio. Y el principio es…

**Da Theory**

Antes de empezar, ¿qué es una **IMU**? Se trata de un dispositivo capaz de mesurar la fuerza (acceleración) y la velocidad. Genéricamente consta de un **Acelerómetro** y un **Giroscopio**. Por lo tanto: ***una IMU no mesura ángulos***. Por lo menos no directamente, requiere algunos cálculos.

El**MPU-6050** es una IMU de **6DOF** (se lee “**6 Degrees Of Freedom**“). Esto significa que lleva un**acelerómetro** y un **giroscopio**, ambos de **3 ejes** (3+3 = 6DOF). Hay IMUs de 9DOF, en este caso también llevan un **magnetómetro**. Otras pueden tener 5DOF, en cuyo caso el giroscopio sólo mide dos ejes, etc.

[](http://robologs.net/wp-content/uploads/2014/10/mpu-6050.jpg)

*MPU-6050*

El **MPU-6050** opera con 3.3 voltios, aunque algunas versiones (como la mía) llevan un regulador que permite conectarla a 5V.

El **MPU-6050** utiliza el protocolo de comunicación I2C. En este tutorial no voy a entrar en detalles de cómo funciona este protocolo, pero puedes encontrar más información [aquí](http://playground.arduino.cc/Learning/I2C).

Ya tienes claro lo que es una IMU. Veamos cada una de sus partes.

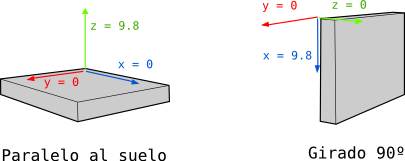
**El acelerómetro**

El **acelerómetro** mide la aceleración. Quién iba a decirlo. La aceleración puede expresarse en 3 ejes: X, Y y Z, las tres dimensiones del espacio. Por ejemplo, si mueves la IMU hacia arriba, el eje Z marcará un cierto valor. Si es hacia delante, marcará el eje X, etc.

Si haces memoria de tus clases de la ESO (o equivalentes), recordarás que la gravedad de la Tierra tiene **una aceleración de aprox. 9.8 m/s²**, perpendicular al suelo como es lógico. Así pues, la IMU también detecta la aceleración de la gravedad terrestre.

¿Esto es un problema? ¡Al contrario, lector! Gracias a la gravedad terrestre puedes usar las lecturas del acelerómetro para saber cuál es el **ángulo de inclinación** respecto al eje X o eje Y.

Supongamos que la IMU esté perfectamente alineada con el suelo. Entonces, como puedes ver en la imagen, **el eje Z marcará 9.8**, y los otros dos ejes marcarán 0. Ahora supongamos que **giramos la IMU 90 grados**. Ahora es el eje X el que está perpendicular al suelo, por lo tanto marcará la aceleración de la gravedad.

[](http://robologs.net/wp-content/uploads/2014/10/imu_ac_ejemplos.png)

Si sabemos que la gravedad es 9.8 m/s², y sabemos que mesura dan los tres ejes del acelerometro, por trigonometría es posible calcular el ángulo de inclinación de la IMU. Una buena fórmula para calcular el ángulo es:

[CodeCogsEqn(3)](http://robologs.net/wp-content/uploads/2014/10/codecogseqn3.gif)[CodeCogsEqn(2)](http://robologs.net/wp-content/uploads/2014/10/codecogseqn2.gif)

El eje Z se suele ignorar.

Hay otras fórmulas equivalentes, pero yo prefiero la tangente.

**El Giroscopio**

En un principio, los giroscopios eléctricos eran unos voluminosos artefactos que valían la mayor parte del presupuesto militar de un estado. Más tarde, durante la segunda guerra mundial se emplearon para dirigir cohetes y torpedos. Por suerte, gracias la revolución digital y la miniaturización de circuitos, hoy en día cualquier aficionado a la electrónica puede permitirse uno. Aunque no para construir misiles.

[](http://robologs.net/wp-content/uploads/2014/10/v2b.jpg)

*Los giroscopios eléctricos se utilizaron en los cohetes V-2*

El **giroscopio** mide la **velocidad angular**. Si no tienes muy frescas tus lecciones de física del instituto voy a recordarte que la velocidad angular es el número de **grados que se gira en un segundo**.

Sólo que en vez de mesurarse en grados por segundo, suele mesurarse en otra unidad que son**radianes por segundo** (**1rad/s = 180/PI grados/s**)

Si sabemos el **ángulo inicial** de la IMU, podemos sumarle el valor que marca el giroscopio para saber el**nuevo ángulo** a cada momento. Supongamos que iniciamos la IMU a 0º. Si el giroscopio realiza una medida cada segundo, y marca 3 en el eje X, tendremos el ángulo con esta sencilla fórmula:

[FormulaAct](http://robologs.net/wp-content/uploads/2015/04/codecogseqn11.gif)

Dónde **Δ**t es el **tiempo que transcurre** cada vez que se calcula esta fórmula.

Y lo mismo pasa con los ejes X, Z. Sólo que al igual que con el acelerómetro se suele ignorar el eje Z.

Ya tenemos las lecturas. ¿Simple, verdad? Pues en realidad ***no tanto***.

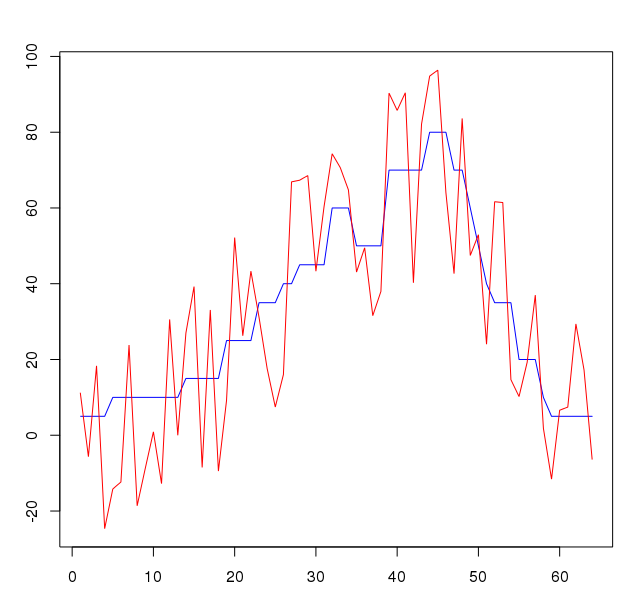
**Error en las medidas**

En un mundo ideal donde hadas y elfos cantan alegres y dulces canciones alrededor de hogueras, donde los dragones surcan los cielos, los árboles hablan con las setas y C++ no provoca errores en tiempo de ejecución, las IMUs son unos artefactos mágicos que con un poco de trigonometría puden dar un ángulo con total exactitud.

Pero estás en el mundo real, hijo. Y hay dos problemas muy importantes: el **ruido** y los **errores**.

El **ruido** son todas aquellas **interferencias** que afectan a los dispositivos electrónicos. El acelerómetro es capaz de medir cualquier ángulo, sin embargo sus**lecturas son ruidosas** y tienen un cierto margen de error.

Si alguna vez quieres dibujar un gráfico de las medidas de un acelerómetro en función del tiempo, verás algo de este estilo:

[](http://robologs.net/wp-content/uploads/2014/10/ruido_imu.png)El **ángulo real** (ideal) está marcado en **azul**, y las **medidas reales** están en **rojo**. Puedo afirmar que no cumple a rajatabla la definición de “preciso”.

Por si esto fuera poco, el acelerómetro también **detecta cualquier aceleración** que no sea la de la gravedad. Por tanto, si mueves la IMU sin girarla, al aplicar una aceleración en otro eje, la IMU lo detectará como un cambio de rotación.

Por otra parte tenemos el giroscopio. A diferencia del acelerómetro, da las medidas con **mucha precisión**. Pero al realizar los cálculos del ángulo es inevitable que se produzca un **pequeño error**, que con el tiempo **va acomulándose** hasta que cualquier similitud con la realidad es pura coincidencia. Esto en inglés se llama **drift**.

¿Significa esto que debes darte por vencido? Para nada. Hay una (varias) forma(s) de combinar los datos del acelerómetro y el giroscopio para así obtener medidas acuradas. Y aquí es dónde entra en juego:

**Tu amigo el filtro**

La idea es muy simple: hay que conseguir eliminar el **ruido**, el **drift** y conseguir que el acelerómetro no cambie de ángulo al detectar otra fuerza que no sea la gravedad.

Hay distintos algoritmos, llamados **filtros**, que hacen esta tarea. Uno de los mejores es el famoso**Filtro de Kálman**, del que puede que hayas oído a hablar. Se utiliza en los **aviones**, **cohetes** y**satélites** geoestacionarios.

El filtro de Kálman es sorprendente. Es considerado uno de los mayores hallazgos del siglo pasado, y con razón. Capaz de calcular el error de cada medida a partir de las medidas anteriores, eliminarlo y dar el valor real del ángulo. En cierto modo es un algoritmo que aprende en cada iteración. No suena nada mal, ¿verdad?

Sin embargo tiene dos problemas:

1. Tiene un coste de procesamiento algo elevado
2. Es muy complicado de entender. Y pongo énfasis en el “muy”. [Échale un vistazo](http://es.wikipedia.org/wiki/Filtro_de_Kalman).

Así pues, tenemos otros filtros a nuestra disposición. El que vamos a utilizar es conocido como **Filtro Complementario o Complementary Filter** en inglés. Es ideal para implementar con Arduino: fácil de utilizar, bajo coste de procesamiento y con una precisión muy buena.

¿En qué consiste exactamente? El **Filtro Complementario** es en realidad una unión de dos filtros diferentes: un **High-pass Filter** para el giroscopio y un **Low-pass Filter** para al acelerómetro. El primero deja pasar únicamente los valores por encima de un cierto límite, al contrario que el Low-pass filter, que sólo permite a los que están por debajo.

La fórmula resultante de combinar (**complementar**, de aquí el nombre) los dos filtros es:

[CodeCogsEqn(5)](http://robologs.net/wp-content/uploads/2014/10/codecogseqn5.gif)

Dónde **AnguloGyro** es el ángulo del Giroscopio que hemos calculado previamente, y **AnguloAccel**con el Acelerómetro. Esta fórmula es la misma para el eje X, Y.

NOTA: Si lo deseas, puedes probar de cambiar el valor de 0.98 y 0.02 por un valor personalizado. Eso sí, ambos tienen que sumar 1.

Y así se termina Da Theory. Ahora hay que utilizar las fórmulas en un código para Arduino.

**In Praxis**

Las fórmulas y los números viven tranquilas en el abstracto mundo de las matemáticas. A pesar de que este lugar está custodiado por unos seres divinos llamados **matemáticos**, cuya función es asegurarse de que sólo los licenciados puedan utilizar las fórmulas en el mundo real, vamos a implementar estos algoritmos en una placa Arduino.

Las conexiones son muy simples. Con los 5V del USB tendrás energía de sobra.

* MPU Vcc -> Arduino 3.3v (o 3v3 en algunos modelos)
* MPU Gnd -> Arduino Gnd
* MPU SCL -> Arduino A5
* MPU SDA -> Arduino A4

***IMPORTANTE!***Estas conexiones están probadas en una Arduino UNO y Arduino Nano, cuyos pines SDA y SCL son A4 y A5 respectivamente. Si utilizas otra placa puede que los pines sean otros. [En este enlace](http://arduino.cc/en/reference/wire) aparece una lista con los pines SDA y SCL de otras versiones de Arduino (Due, Leonardo…)

Así pues, tienes las fórmulas. Tienes la IMU. Tienes una placa Arduino. Tienes el circuito. ¿Qué falta? Saber cómo interaccionar con la IMU para obtener lecturas. El MPU-6050 dará unos**raw values** (“valores en bruto”) que después habrá que **refinar** (dividir por una constante) a fin de conseguir valores utilizables.

La web oficial de Arduino dispone de un ejemplo para leer datos de la MPU-6050. Partiré de ella, aunque habrá que modificarla considerablemente ya que sólo da los valores sin refinar del Giroscopio y el Acelerómetro.

Empezamos por las declaraciones.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include <Wire.h>    //Direccion I2C de la IMU  #define MPU 0x68    //Ratios de conversion  #define A\_R 16384.0  #define G\_R 131.0    //Conversion de radianes a grados 180/PI  #define RAD\_A\_DEG = 57.295779    //MPU-6050 da los valores en enteros de 16 bits  //Valores sin refinar  int16\_t AcX, AcY, AcZ, GyX, GyY, GyZ;    //Angulos  float Acc[2];  float Gy[2];  float Angle[2]; |

Voy a comentarlo. La primera línea incluye la librería Wire.h, necesaria para la interacción vía protocolo I2C.

#define MPU 0x68 es la dirección I2C de la IMU que se especifica en la documentación oficial.

Los ratios de conversión son los especificados en la documentación. Deberemos dividir los valores que nos dé el Giroscopio y el Acelerómetro entre estas constantes para obtener un valor coherente. RAD\_A\_DEG es la conversión de radianes a grados.

La IMU da los valores en enteros de 16 bits. Como Arduino los guarda en menos bits, hay que declarar las variables que almacenarán los enteros provinientes de la IMU como un tipo de enteros especiales. int16\_t AcX, AcY, AcZ, GyX, GyY son, pues, los raw\_values de la IMU.

Finalmente tenemos tres arrays (Acc[], Gy[], Angle[]) que guardan el ángulo X, Y del Acelerómetro, el Giroscopio y el resultado del Filtro respectivamente. [0] se corresponde a X. [1] a Y.

La función setup es la siguiente:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | void setup()  {  Wire.begin();  Wire.beginTransmission(MPU);  Wire.write(0x6B);  Wire.write(0);  Wire.endTransmission(true);  Serial.begin(9600);  } |

Se inicia la comunicación por I2C con el dispositivo MPU, y se “activa” enviando el comando 0. También se inicia el puerto de série para ver los resultados.

El void loop es un poco más complejo. Se leen y guardan los datos de la IMU, se calcula el ángulo y se aplica el filtro complementario.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | void loop()  {     //Leer los valores del Acelerometro de la IMU     Wire.beginTransmission(MPU);     Wire.write(0x3B); //Pedir el registro 0x3B - corresponde al AcX     Wire.endTransmission(false);     Wire.requestFrom(MPU,6,true); //A partir del 0x3B, se piden 6 registros     AcX=Wire.read()<<8|Wire.read(); //Cada valor ocupa 2 registros     AcY=Wire.read()<<8|Wire.read();     AcZ=Wire.read()<<8|Wire.read();        //Se calculan los angulos Y, X respectivamente.     Acc[1] = atan(-1\*(AcX/A\_R)/sqrt(pow((AcY/A\_R),2) + pow((AcZ/A\_R),2)))\*RAD\_TO\_DEG;     Acc[0] = atan((AcY/A\_R)/sqrt(pow((AcX/A\_R),2) + pow((AcZ/A\_R),2)))\*RAD\_TO\_DEG;       //Leer los valores del Giroscopio     Wire.beginTransmission(MPU);     Wire.write(0x43);     Wire.endTransmission(false);     Wire.requestFrom(MPU,4,true); //A diferencia del Acelerometro, solo se piden 4 registros     GyX=Wire.read()<<8|Wire.read();     GyY=Wire.read()<<8|Wire.read();       //Calculo del angulo del Giroscopio     Gy[0] = GyX/G\_R;     Gy[1] = GyY/G\_R;       //Aplicar el Filtro Complementario     Angle[0] = 0.98 \*(Angle[0]+Gy[0]\*0.010) + 0.02\*Acc[0];     Angle[1] = 0.98 \*(Angle[1]+Gy[1]\*0.010) + 0.02\*Acc[1];       //Mostrar los valores por consola     Serial.print("Angle X: "); Serial.print(Angle[0]); Serial.print("n");     Serial.print("Angle Y: "); Serial.print(Angle[1]); Serial.print("n------------n");       delay(10); //Nuestra dt sera, pues, 0.010, que es el intervalo de tiempo en cada medida  } |

Escribe estas tres funciones en un sólo código, compila, carga. Si ahora abres el Monitor de Série, deberías ver como van oscilando los números muy rápidamente. Puedes probar a girar la IMU y detener el Desplazamiento Automático para ver como cambian los ángulos.

El filtro tarda un tiempo a adaptarse a los cambios, lo cuál es normal al filtrar las frecuencias.

Y esto es todo. Espero que esta guía te sirva de referencia para futuros proyectos. Los filtros de este tipo suelen utilizarse conjuntamente con un [Controlador PID](http://en.wikipedia.org/wiki/PID_controller) (PID controller) para controlar y corregir el movimiento de motores en función de las lecturas de la IMU. Si estás interesado en realizar un proyecto como un drone o un robot balanceador, puede ser de utilidad comprender cómo funciona.

Finalmente debo decir que existen librerías para Arduino que incluyen Filtros de Kálman y Filtros Complementarios. ¿Por qué no las he mencionado? Mi consejo cuándo construyas tu proyecto es que, si puedes, evites las librerías de terceros. Aprender a implementar este tipo de algoritmos por tu cuenta mejorará tus conocimientos y posibilidades mucho más que si te limitas a las funciones prefabricadas.

Y al fin y al cabo, ¿qué es la robótica sin pelearse con las fórmulas, problemas y código?

Final de línea.