INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

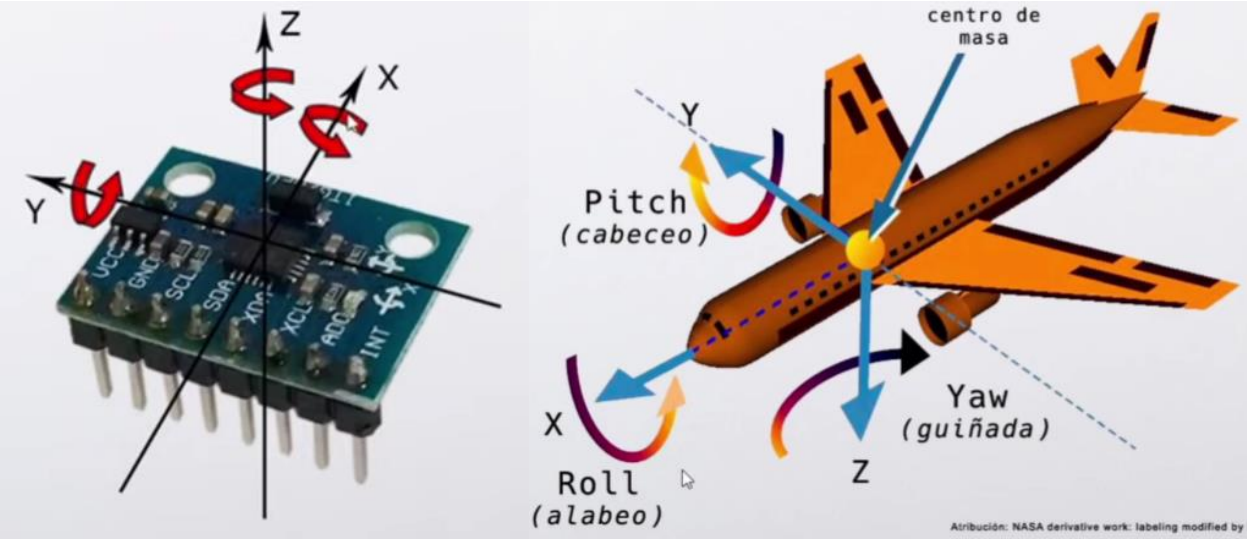
Comunicación I2C, Acelerómetro y Giroscopio MPU6050 con Arduino

Contenido

Acelerómetro y Giroscopio MPU6050

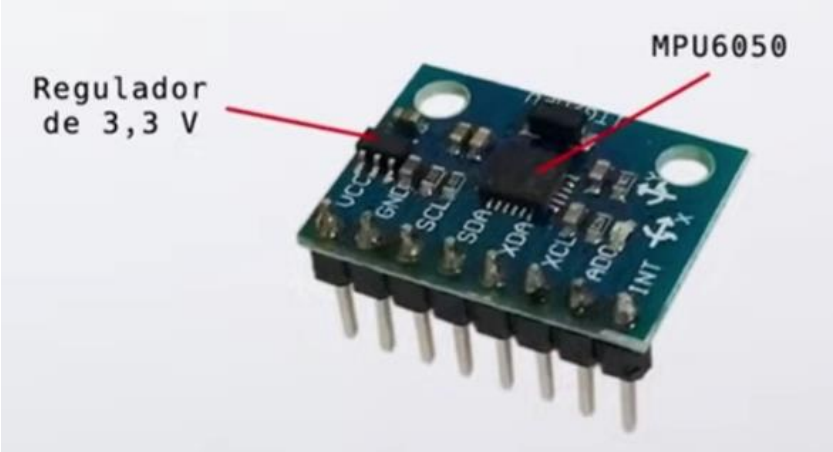
El MPU6050 es un módulo de tipo IMU (Inertial Measure Unit), el cual **sirve para medir la orientación de un dispositivo, tanto en inclinación, como en su dirección en el plano 2D**, para ello incluye un **acelerómetro que detecta la aceleración lineal** y un **giróscopo que censa la aceleración angular (la cual al ser integrada indica el ángulo de inclinación)**, **esto se realiza a través de 6 grados de libertad** y el sensor se debe colocar en el centro de masa del dispositivo que se quiere controlar, para así lograr una medición correcta. El término **"grado de libertad"** se refiere al sentido en el cual se puede medir una magnitud en un espacio tridimensional, en este caso son 6 porque se tienen **3 ejes coordenados** (**X**, **Y**, **Z**), donde se miden las direcciones positivas y negativas de dos magnitudes diferentes; ya que **en cada eje no solamente se pueden cuantificar las aceleraciones que van en dirección recta** (**indicadas por el acelerómetro**), sino **también las velocidades de rotación en ambos sentidos** (**por medio del giroscopio**), osea **las rotaciones en sentido de las manecillas del reloj** y **en sentido contrario**, los datos recibidos del acelerómetro y giróscopo son combinados para medir de forma precisa la orientación. Este sensor es muy utilizado en aeronaves, celulares, drones, gimbals, etc.

- **Mediciones de los 6 DOF (Degrees of Freedom)**:
  - **Acelerómetro de 3 ejes**: Considera las direcciones lineales y el sentido de sus ejes **XY** está pintado encima del módulo para saber cuál es cada uno.
    1. Aceleraciones horizontales en el plano 2D **± X**.
    2. Aceleraciones verticales en el plano 2D **± Y**.
    3. Aceleraciones de altura en el plano 3D **± Z**.
  - **Giróscopo de 3 ejes**: Considera las direcciones rotacionales en cada eje coordenado.
    4. Aceleración rotacional con dirección de **giro CW (en sentido de las manecillas del reloj)** y **CCW (en sentido contrario a las manecillas del reloj)** sobre el eje coordenado **X**. **Al integrar la aceleración rotacional, se obtiene el ángulo.**
    5. Aceleración rotacional con dirección de **giro CW** y **CCW** en el eje coordenado **Y**.
    6. Aceleración rotacional con dirección de **giro CW** y **CCW** en el eje coordenado **Z**.

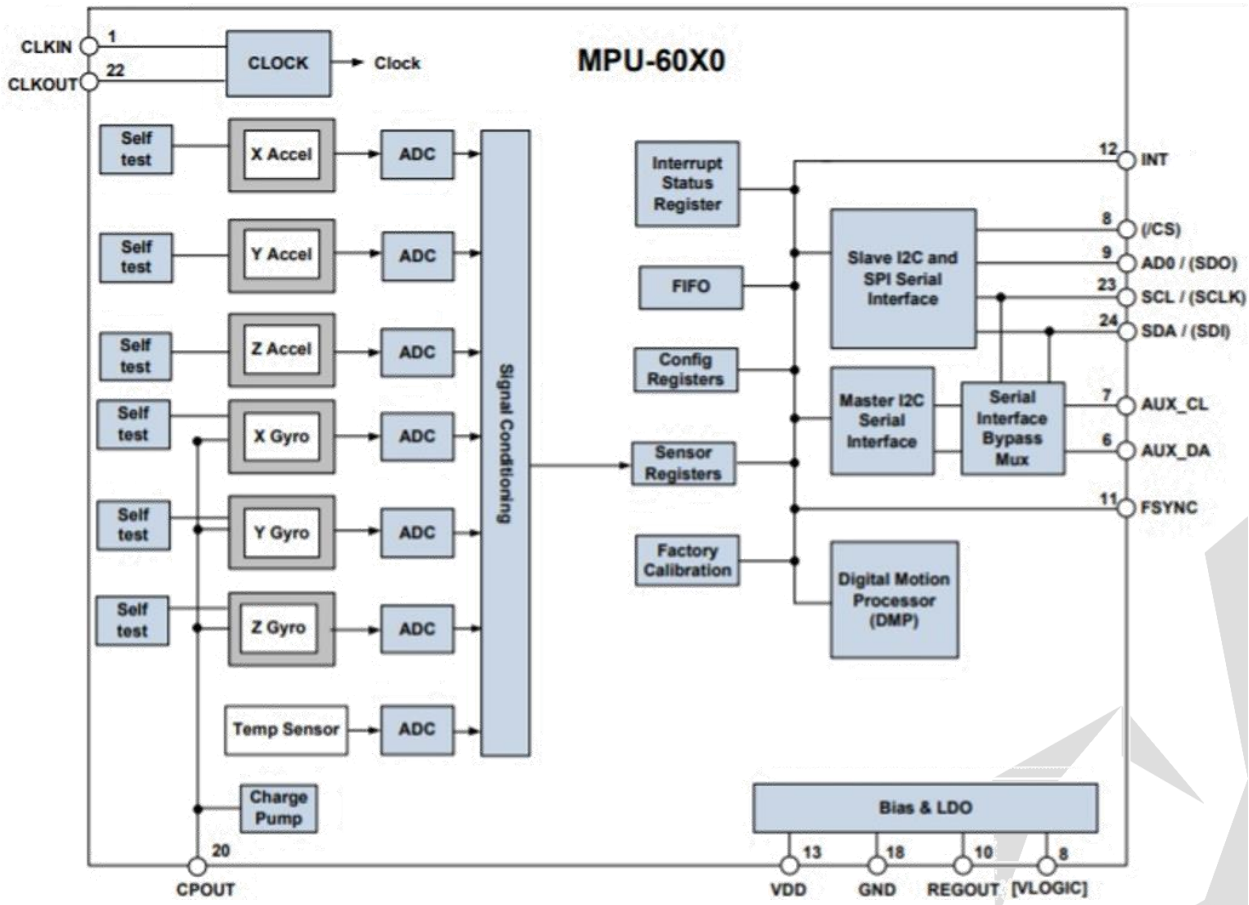


El circuito integrado (CI) **es alimentado con una tensión de 3.3V**, pero como tiene un regulador de tensión incluido, **se le puede proporcionar una tensión de 5V en su pin y el regulador bajará esa tensión**. También

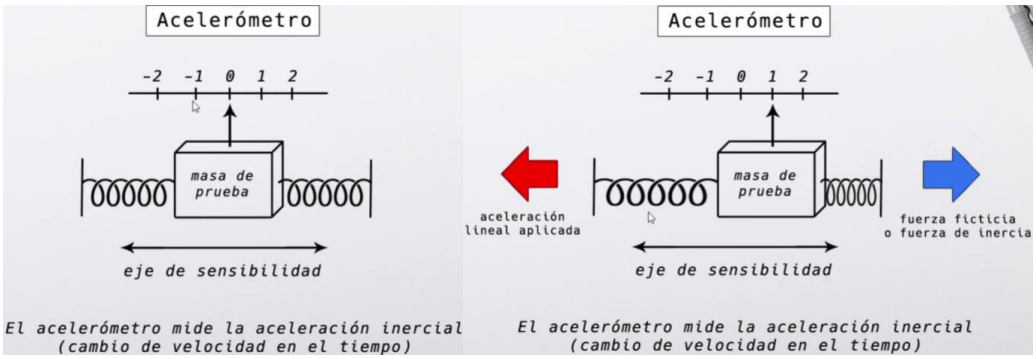
cuenta con un **DMP (Digital Movement Processor)**, que es un controlador interno encargado de aplicar un procesamiento de los datos de **aceleración lineal** y **rotacional** para mandarlos a través de un **bus serial con protocolo I2C a un microcontrolador que los interprete y utilice**.



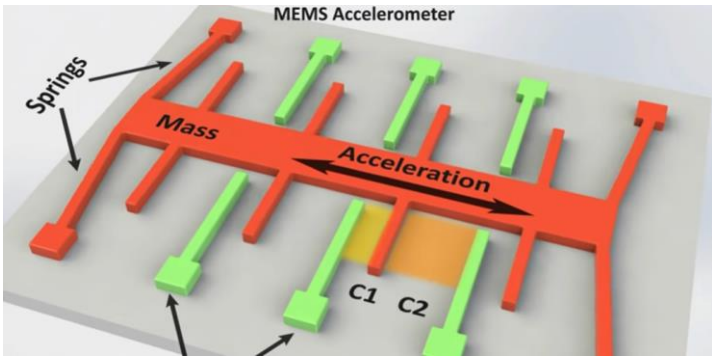
En el diagrama esquemático del circuito se pueden observar los 6 sensores de tipo **MEMS** (**Micro Electro Mechanic Systems**) que miden cada uno de los grados de libertad del **acelerómetro y giróscopo MPU6050** para obtener la orientación del dispositivo: **X Accel**, **Y Accel**, **Z Accel**, **X Gyro**, **Y Gyro** y **Z Gyro**. Estos representan sistemas mecánicos donde ocurren movimientos microscópicos que permiten medir fuerzas del entorno y así censar la **aceleración lineal** y **rotativa**.



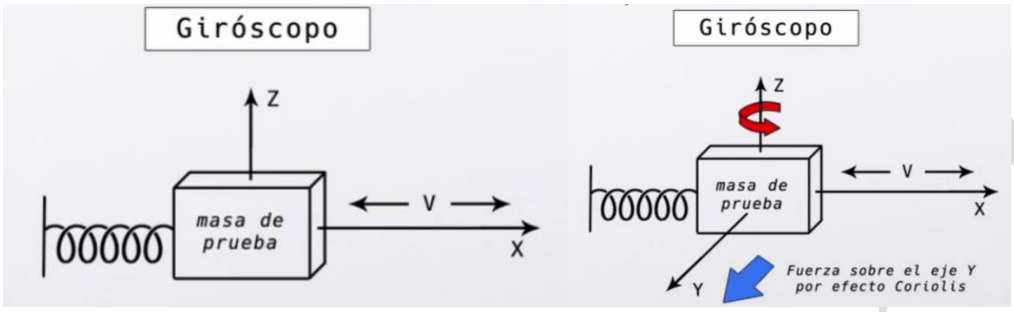
La forma en la que los MEMS del acelerómetro miden la aceleración lineal es a través de masas de prueba y resortes microscópicos, ya que, al someter el sensor a una aceleración, la masa se moverá hacia el otro lado debido a su inercia. Siguiendo la segunda ley de Newton donde: $\boldsymbol{F = m * a}$, se podrá calcular su magnitud, porque el resorte en el sentido contrario del movimiento se comprimirá y el otro se expandirá. Se usa un acelerómetro distinto para calcular individualmente las aceleraciones lineales de cada eje **X**, **Y**, **Z**; y su resultado se mide con **unidades g** (de gravedad), que individualmente valen 9.81 $\left[\frac{m}{s^2}\right]$.

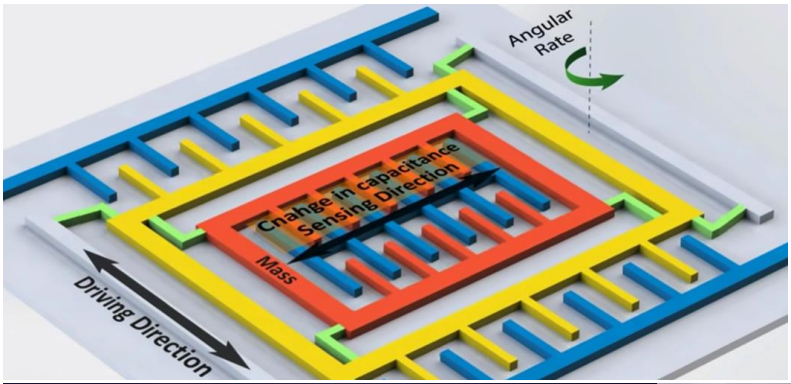


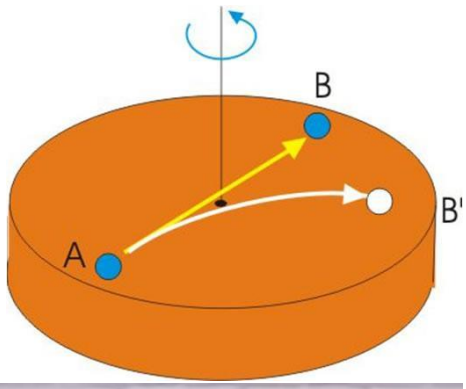
La medición del acelerómetro se realiza mediante un nivel de capacitancia, el cual varía porque la masa mueve placas paralelas microscópicas, logrando que estas se acerquen y alejen una de la otra como se explicó previamente. **Mientras más juntas estén las placas, mayor será su capacitancia y viceversa.**

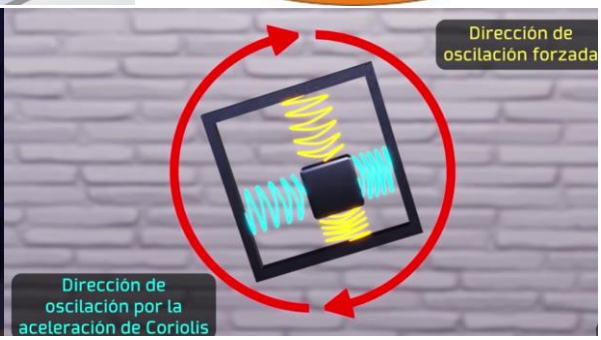


La forma en la que los MEMS del giróscopo miden la aceleración angular en cada eje **es también a través de una masa de prueba y un resorte microscópico, pero a este modelo siempre se estará aplicando una fuerza que haga oscilar el sistema**. Al someter el sensor a una aceleración angular sobre alguno de los ejes, **la masa se moverá hacia el eje donde no se está aplicando la oscilación del sistema por una fuerza ocasionada debido a un efecto llamado Aceleración de Coriolis**, cuya magnitud se mide en unidades de grados por segundo $\left[\frac{rad}{s}\right]$.

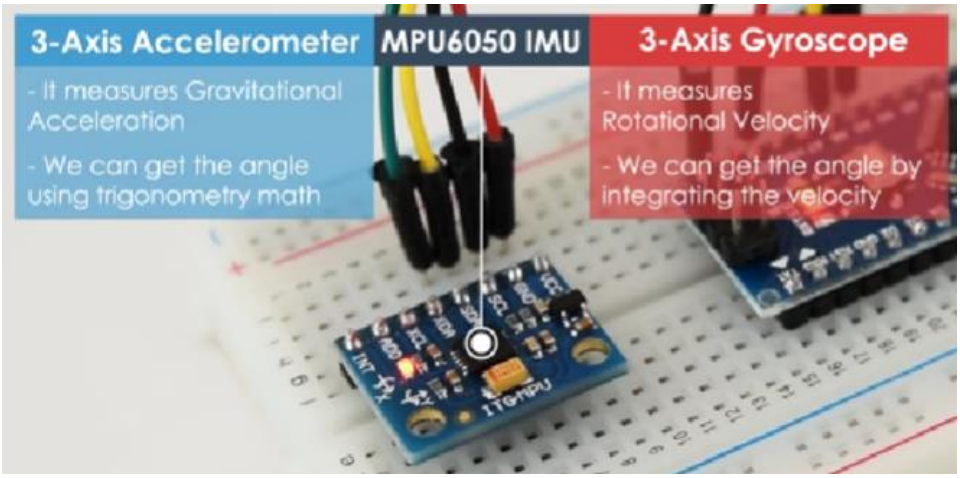








Los datos obtenidos del **acelerómetro** y **giroscopio** **son combinados** para obtener el **ángulo de inclinación y orientación plana** **del módulo MPU6050**, en el caso del giroscopio, la aceleración angular se integra para obtener el ángulo de inclinación y en el caso del acelerómetro, se realizan operaciones de trigonometría para obtener la orientación plana XY.

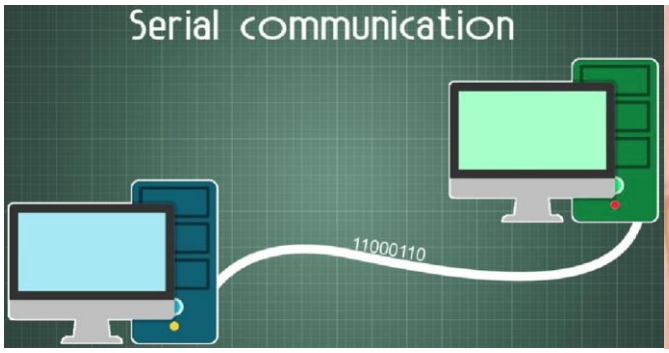


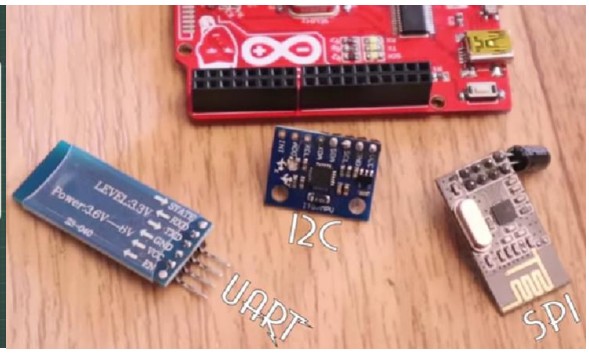
Protocolos de Comunicación Seriales

La transferencia de datos digitales ejecutada por el acelerómetro y giróscopo MPU6050 es realizada a través del **protocolo de comunicación I2C**, a continuación, se explicará en que consiste:

- **Comunicación en paralelo:** Este tipo de comunicación se realiza **a través de un bus de forma paralela**, donde **un bus representa el conjunto de varios cables**, del cual **se extraen o mandan bits de información de forma individual**. Todos los datos se enviarán cuando se reciba un pulso de la **señal de reloj** del sistema, por lo que esta opción es más rápida, pero de muchas conexiones.

- **Comunicación serial (en serie):** Este tipo de comunicación manda o recibe **datos a través de uno o pocos cables**, moviendo la información un bit a la vez, usualmente pasando el dato de un **Flip Flop a otro** (**que es la unidad de almacenamiento más pequeña, almacenando solo 1 bit**) a través de su conexión en serie. Cada dato se enviará cuando se reciba un pulso de la **señal de reloj** del sistema, por lo que esta opción es más lenta, debido a que tiene que esperar varios ciclos de reloj para enviar la información completa, pero tiene muy pocas conexiones.
    - **Existen varios tipos de comunicación serial, pero los más usados son 3:** UART, I2C y SPI.





Protocolo de Comunicación Serial Asíncrona UART

No todos los protocolos seriales necesitan el pulso de una señal de reloj para funcionar, uno de esos ejemplos es el protocolo UART (Universal Asyncronous Reciever Transmitter), el cual solamente usa dos cables:

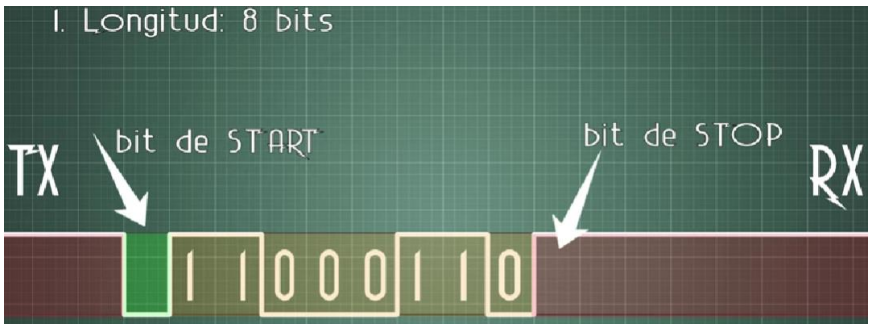
- **TX/RX**: Cable **de transmisión** (**TX**, que manda) **o recepción** (**RX**, que colecta) **de datos**.
- **GND**: Cable conectado a tierra.

La forma en la que los cables **TX** y **RX** reconocen los bits 0 y 1 enviados y recibidos de la señal digital es a través de 3 características, que deben estar configuradas con el mismo valor tanto en el dispositivo transmisor como el receptor que están utilizando el protocolo UART de comunicación serial:

- **Longitud de dato:** Indica **cuantos bits conforman a cada dato que se quiere mandar o recibir**, usualmente se agrupan en **bytes**, que **se conforma de 8 bits**, **pero se podría utilizar otra longitud de datos**.
- **Bits de START y STOP**: Es un **bit que se encuentra antes y después del dato mandado o recibido**, el cual **puede adoptar valores de 0 o 1 para indicar su inicio y fin**, para ello se debe tener ya asignada la longitud de datos y usualmente **el valor del bit de START es 0** y **el de STOP es 1**.

- **Velocidad de transmisión:** Esta velocidad **expresa la duración de cada bit en la información mandada/recibida**, la unidad que medida que tiene es de **baudios por segundo** y las velocidades más utilizadas son las siguientes.
    - **9,600** $\left[\frac{Baudios}{segundo}\right]$**:** Es la velocidad de transmisión más comúnmente utilizada, ya que es compatible con la mayoría de los dispositivos y programas. **Representa un tiempo de duración por bit de:** $duracionBit = \frac{1}{Velocidad\ de\ Transmisión} = \frac{1}{9600} = 104.1666\ [\mu s]$
    - **115,200 o 57,600** $\left[\frac{Bd}{s}\right]$: Si se necesita realizar una transferencia de datos más rápida y el hardware/software lo admiten, se puede optar por estas velocidades más altas, pero corremos con el riesgo de que por la gran velocidad de transmisión se pierdan datos.

Esto significa que, lo que hace el protocolo UART para reconocer cada bit de la información transportada es: Contar el tiempo transcurrido desde que percibe el **bit de START, medir cuanto es que dura cada estado lógico 1 o 0 para reconocer de forma individual los bits de los datos mandados/recibidos y** finalmente cuando perciba el **bit de STOP, reconocer que ese es solo 1 de los datos transportados, separando así la información en paquetes**. Ya en la realidad, se deja pasar un tiempo de 52 µs para asegurarnos que se captó correctamente cada bit. **Al repetir este proceso para cada dato se podrá mandar y recibir información en forma binaria**.



La placa de desarrollo de Arduino usa el protocolo UART para mandar y recibir datos por consola, osea del monitor serial del IDE de Arduino.

Además, dentro de los módulos que usan este tipo de comunicación se cuenta con los pines DTR y CTS para indicar cuando es que el dispositivo está mandando información y cuando está recibiendo, para así evitar colisión de datos, mientras que **los pines TX y RX se utilizan para mandar y recibir datos**.

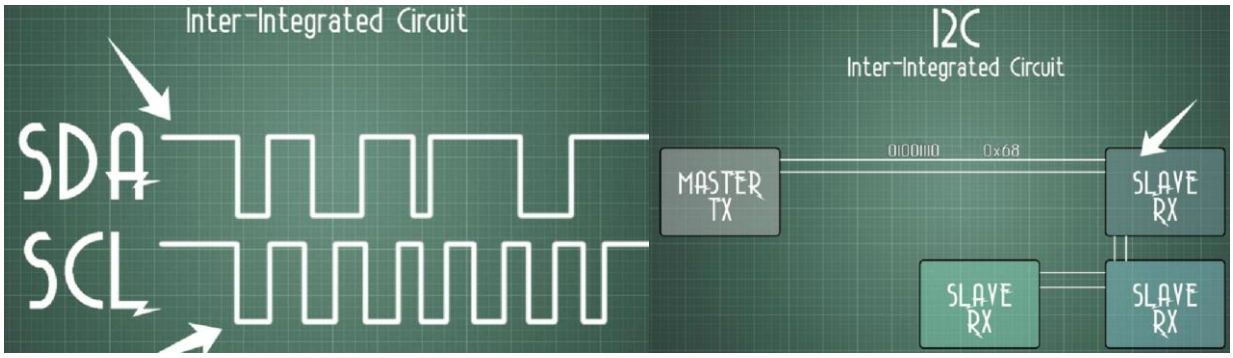
Protocolo de Comunicación Serial Síncrona I2C

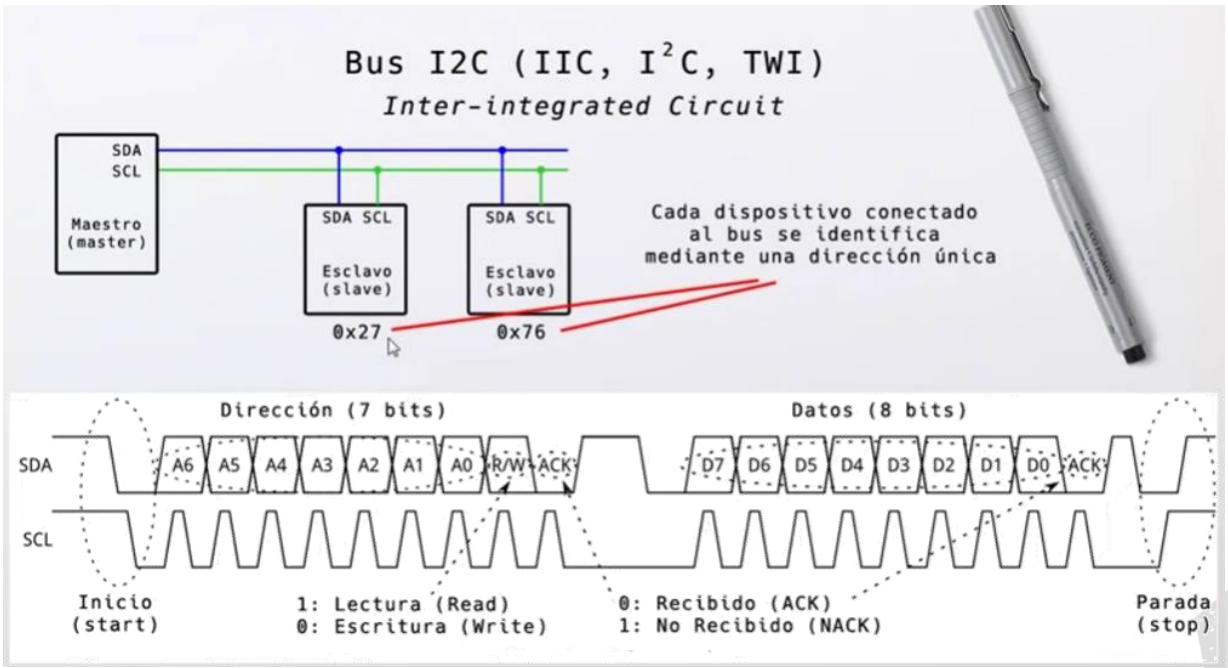
El tipo de comunicación I2C (Inter Integrated Circuit) solo puede enviar o recibir datos a la vez, no hacer las dos al mismo tiempo y para ello **utiliza el paso de la señal de reloj**. **Durante su uso se deben de utilizar 3 cables para la transmisión y recepción de datos:**

- **SDA (Serial Data)**: Cable **de transmisión** o **recepción** de datos, cuya información tiene una **longitud** de 15 o 16 bits.
- **SCL (Serial Clock)**: Pin que recibe una **señal de reloj que marca el paso de la transmisión de datos**, cuya frecuencia es la misma de los bits que conforman la información transportada. **Cada que se envíe un dato, se activará la señal de reloj, sino esta se mantiene en valor de 0 lógico**.
- **GND**: Cable conectado a tierra.

Los dispositivos que utilizan el protocolo de comunicación I2C **pueden** mandar una misma información **a** varios dispositivos **a la vez**, para ello se les pueden asignar dos tipos de roles:

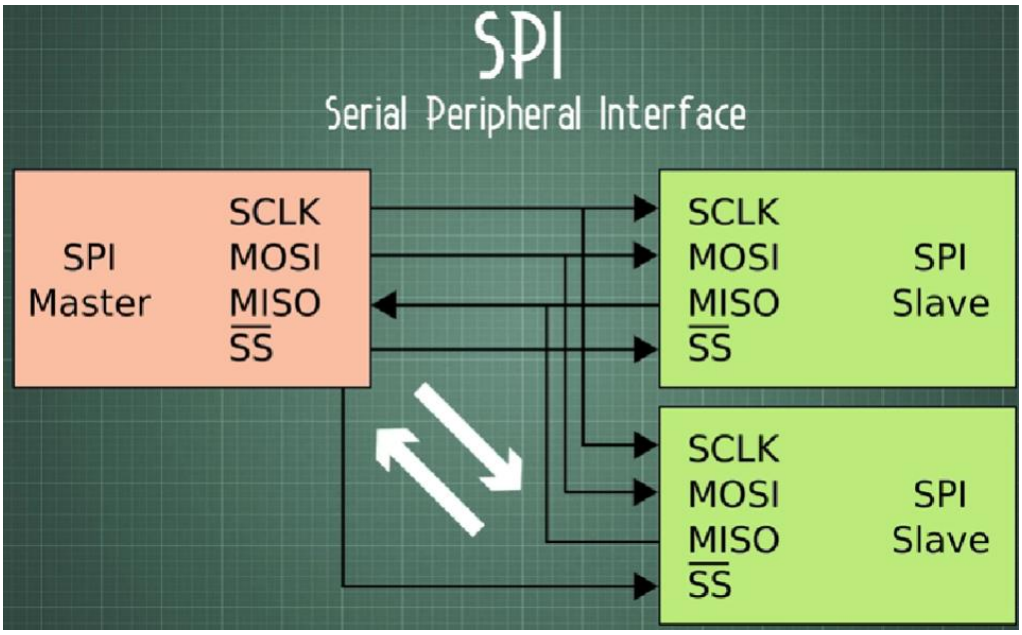
- MASTER: Este rol se asigna a los dispositivos que **mandan datos**.
    - **El** transmisor **primero envía** la dirección del receptor, que es de 7 bits **y luego los datos,** que son de 8 bits. Solamente en el dispositivo SLAVE que tenga asignada dicha dirección, **se podrán almacenar los datos mandados**. También en la dirección del receptor se manda un bit RW **que indica si este recibe** (lectura) **o transmite** (escritura) **los datos**.
- SLAVE: Este rol se asigna a los dispositivos que **reciben y almacenan los datos enviados**.
    - Cada receptor tiene asignada una **dirección diferente**.



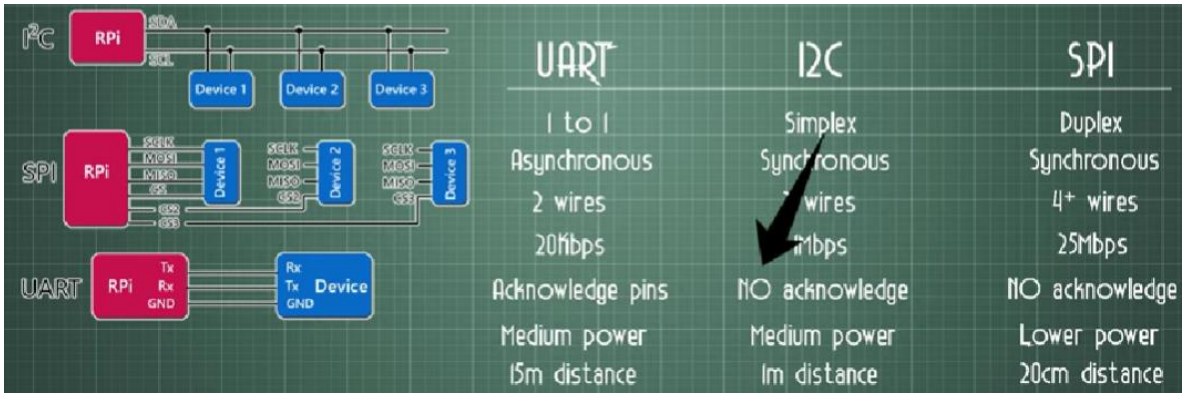


Protocolo de Comunicación Serial Síncrona SPI

La comunicación SPI (Serial Peripheral Interface) es muy parecida a la I2C, pero esta puede mandar y recibir datos a la vez, para ello **utiliza el paso de la señal de reloj** y un pin llamado **Slave o Chip Select (SS)** para indicar a donde se mandar y/o reciben datos, los cuales se mandan por los pines MOSI **(**Master Outpu Slave Input**)** y MISO **(**Master Input Slave Output**).**



Comparación de las características de las 3 comunicaciones seriales: UART, I2C y SPI



Registros del Módulo MPU6050

Los registros que deben ser escritos o leídos del módulo MPU6050 son los siguientes, cuyas instrucciones se encuentran indicadas en el datasheet:

- **Registro 6B hexadecimal = 107 decimal:** A este registro se debe mandar el valor hexadecimal 0X00 en sus 8 bits para apagar el modo de bajo consumo e indicar que el reloj que se está utilizando en la comunicación I2C entre el microcontrolador y el MPU6050 es el interno del sensor, cuya frecuencia es de 8MHz.

4.28 Register 107 – Power Management 1
PWR_MGMT_1

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
6B	107	DEVICE _RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		

CLKSEL	Clock Source
0	Internal 8MHz oscillator
1	PLL with X axis gyroscope reference
2	PLL with Y axis gyroscope reference
3	PLL with Z axis gyroscope reference
4	PLL with external 32.768kHz reference
5	PLL with external 19.2MHz reference
6	Reserved
7	Stops the clock and keeps the timing generator in reset

- **Registro 1C hexadecimal = 28 decimal:** A este registro se debe mandar alguna de las siguientes opciones hexadecimales para asignar un rango de medición al acelerómetro. **Se elige un rango mayor cuando el dispositivo al que se quiere incorporar el MPU6050 estará sometido a aceleraciones muy grandes o expuesto a fuerzas extremas**, como lo puede ser en vehículos, aeronaves, monitoreo de impactos, robótica, sistemas de movimiento rápido, etc.:
  - **0X00:** Para indicar que el rango de medición del acelerómetro sea de **±2g**.
  - **0X08:** Para indicar que el rango de medición del acelerómetro sea de **±4g**.
  - **0X10:** Para indicar que el rango de medición del acelerómetro sea de **±8g**.
  - **0X18:** Para indicar que el rango de medición del acelerómetro sea de **±16g**.

4.5 Register 28 – Accelerometer Configuration ACCEL_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

AFS_SEL	Full Scale Range
0	± 2*g*
1	± 4*g*
2	± 8*g*
3	± 16*g*

- **Dependiendo de qué rango de medición se haya elegido para el acelerómetro**, al realizar la lectura de los siguientes registros, se deberá dividir su valor entre cierta cantidad:
  - **3B y 3C: La aceleración lineal en X** se indica a través de **un dato de 16 bits**, pero **como cada registro solo puede contener 8 bits**, el dato se conforma de la **combinación del contenido de los registros 3B y 3C**.
  - **3D y 3E: La aceleración lineal en Y** se indica a través de **un dato de 16 bits**, pero **como cada registro solo puede contener 8 bits**, el dato se conforma de la **combinación del contenido de los registros 3D y 3E**.

- **3F y 40: La aceleración lineal en Z** se indica a través de **un dato de 16 bits**, pero **como cada registro solo puede contener 8 bits**, el dato se conforma de la **combinación del contenido de los registros 3F y 40**.

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

- **Registro 1B hexadecimal = 27 decimal:** A este registro se debe mandar alguna de las siguientes opciones hexadecimales para asignar un **rango de medición al giróscopo**. **Se elige un rango mayor cuando el dispositivo al que se quiere incorporar el MPU6050 estará sometido a aceleraciones muy grandes o expuesto a fuerzas extremas**, como lo puede ser en vehículos, aeronaves, monitoreo de impactos, robótica, sistemas de movimiento rápido, etc.:
  - **0X00:** Para indicar que el rango de medición del acelerómetro sea de **±250°/s**.
  - **0X08:** Para indicar que el rango de medición del acelerómetro sea de **±500°/s**.
  - **0X10:** Para indicar que el rango de medición del acelerómetro sea de **±1000°/s**.
  - **0X18:** Para indicar que el rango de medición del acelerómetro sea de **±2000°/s**.

4.4 Register 27 – Gyroscope Configuration
GYRO_CONFIG

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

- **Dependiendo de qué rango de medición se haya elegido para el giroscopio**, al realizar la lectura de los siguientes registros, se deberá dividir su valor entre cierta cantidad:
  - **43 y 44: La aceleración angular en X** se indica a través de **un dato de 16 bits**, pero **como cada registro solo puede contener 8 bits**, el dato se conforma de la **combinación del contenido de los registros 3B y 3C**.
  - **45 y 46: La aceleración angular en Y** se indica a través de **un dato de 16 bits**, pero **como cada registro solo puede contener 8 bits**, el dato se conforma de la **combinación del contenido de los registros 3D y 3E**.

- **47 y 48: La aceleración angular en Z** se indica a través de **un dato de 16 bits**, pero **como cada registro solo puede contener 8 bits**, el dato se conforma de la **combinación del contenido de los registros 3F y 40**.

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

Cabe mencionar que los resultados de orientación Pitch, Roll y Yaw no se obtienen por sí solos, se deben calcular de forma manual dentro del código a través de las fórmulas de ángulo de Euler:

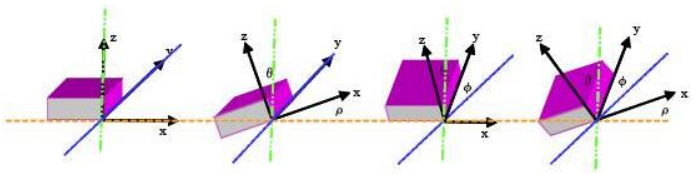
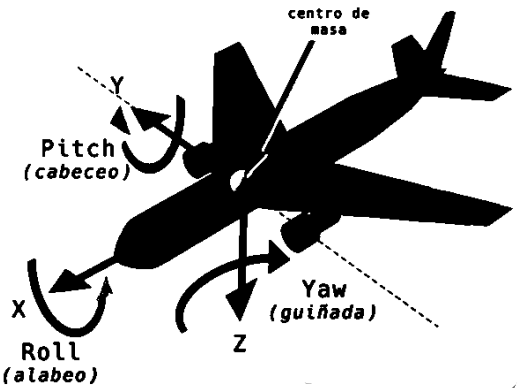


Figure 8. Three Axis for Measuring Tilt

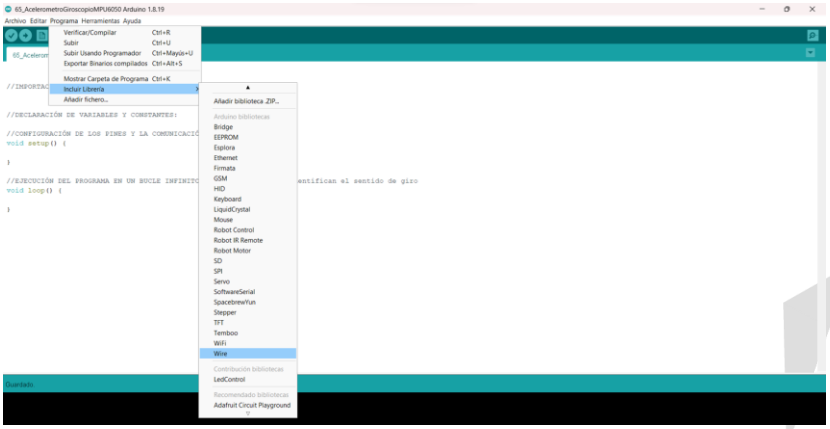
$$\rho = \arctan\left(\frac{A_X}{\sqrt{A_Y^2 + A_Z^2}}\right)$$

$$\phi = \arctan\left(\frac{A_Y}{\sqrt{A_X^2 + A_Z^2}}\right)$$

$$\theta = \arctan\left(\frac{\sqrt{A_X^2 + A_Y^2}}{A_Z}\right)$$

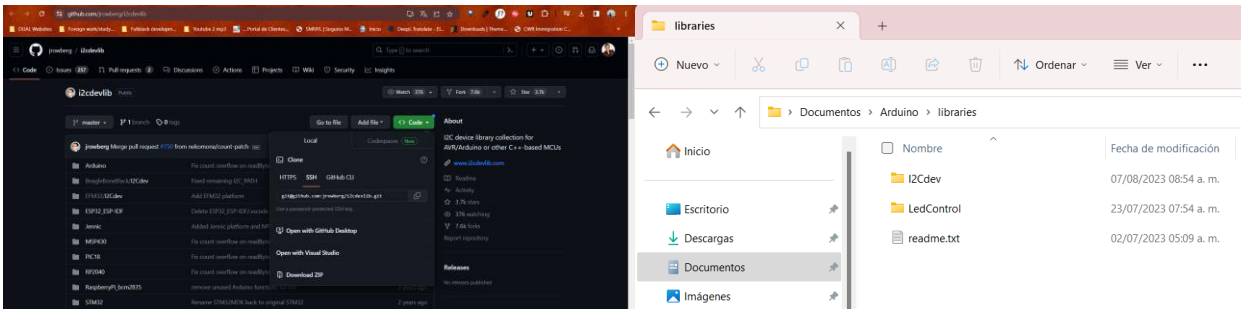


Para realizar la lectura del sensor con Arduino a fuerza se debe alguna librería, **Wire es de las más usadas en Arduino** y **su función principal es establecer la conexión I2C**, pero **posee un error que aumenta gradualmente en la lectura de giro alrededor del eje Z (yaw)**, por lo que **una alternativa para establecer la conexión I2C es la librería i2cdevlib**.



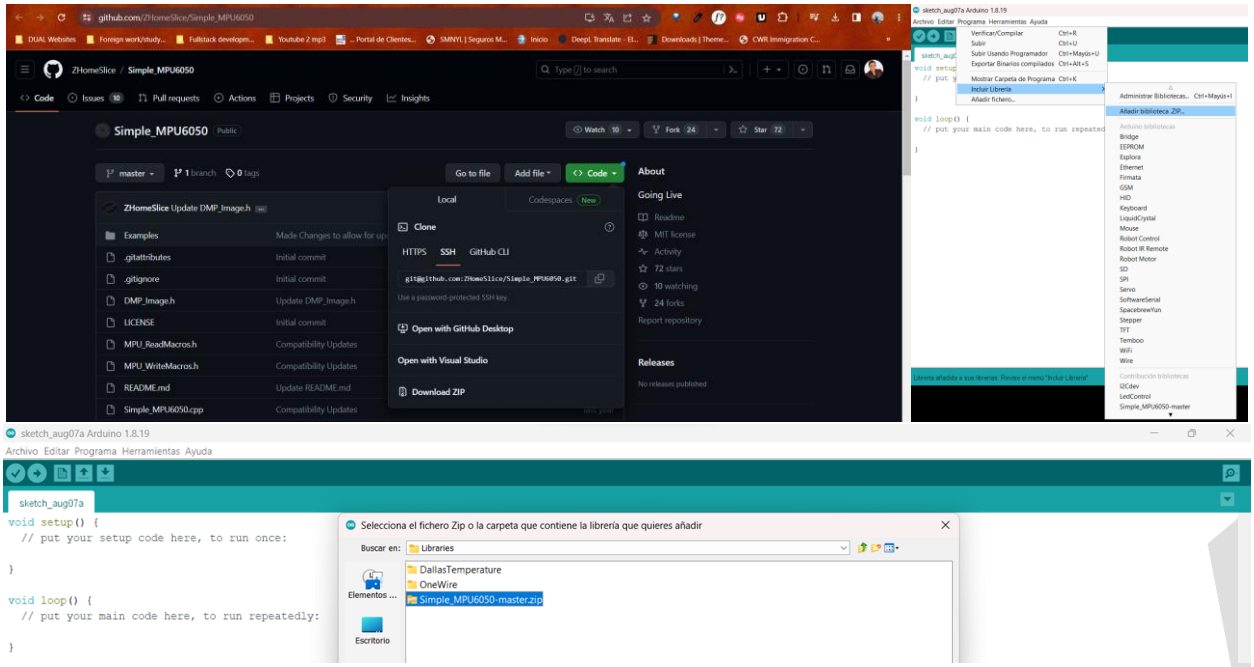
La librería **i2cdevlib** debe ser extraída del siguiente enlace de GitHub y **esta se utiliza simplemente para establecer la conexión I2C**. Para ello se debe extraer el contenido de su carpeta ZIP, luego encontrar la carpeta **I2Cdev** en el directorio **i2cdevlib-master\i2cdevlib-master\Arduino** y finalmente copiar y pegarla dentro de la carpeta **libraries del Arduino IDE**:

https://github.com/jrowberg/i2cdevlib



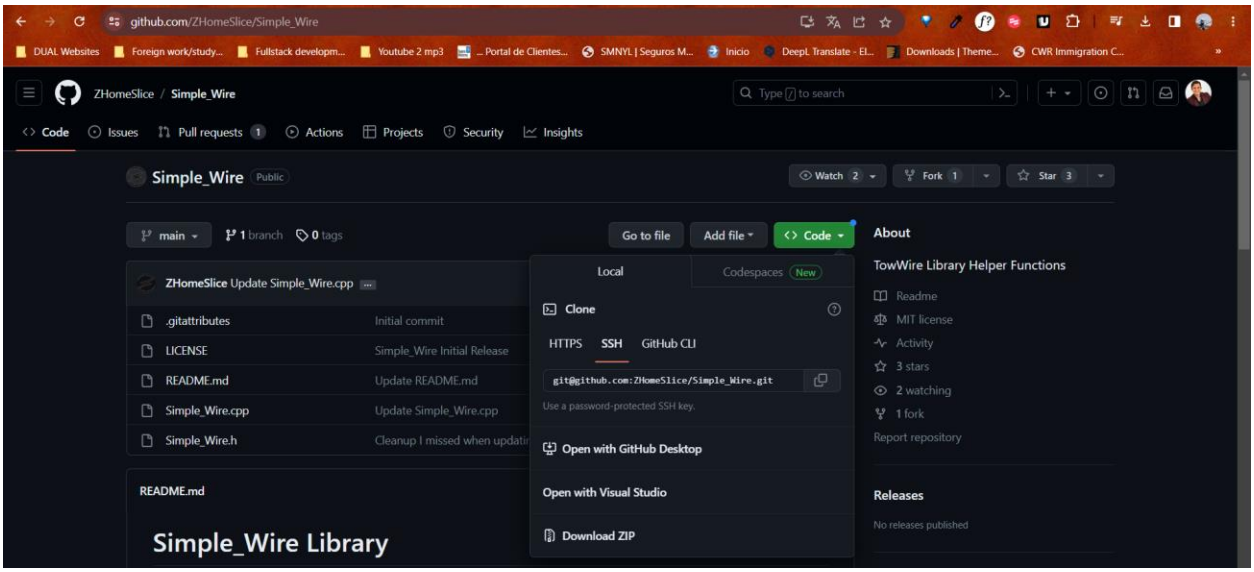
Mientras que, **para acceder a los registros del sensor MPU6050** se utiliza una librería adicional llamada **Simple_MPU6050**, esta se descargará del siguiente link de GitHub, pero **a diferencia de la librería i2cdevlib que establece la conexión I2C**, su contenido no deberá ser extraído, sino que **se importará directamente dentro del Arduino IDE a través de la opción** Programa → Incluir librería → Añadir biblioteca .ZIP… → Simple_MPU6050-master.zip:

https://github.com/ZHomeSlice/Simple_MPU6050



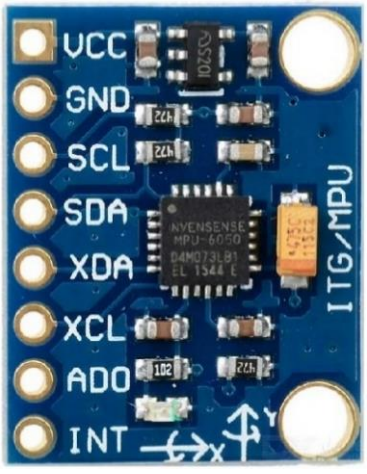
Para poder utilizar la biblioteca **Simple_MPU6050**, **también se requiere instalar la librería Simple_Wire**, el procedimiento de instalación es exactamente el mismo que el utilizado con la carpeta comprimida .ZIP de la librería anterior, y esta se obtiene del siguiente enlace:

https://github.com/ZHomeSlice/Simple_Wire

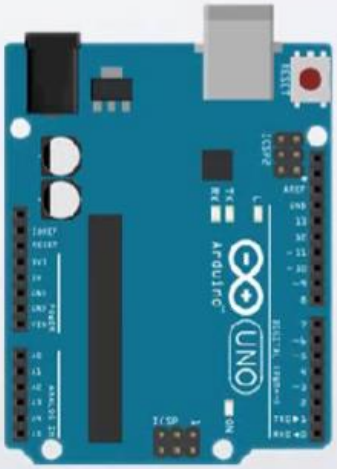


Acelerómetro y Giróscopo MPU6050:

- **VCC** y **GND**: El módulo puede recibir 5 o 3.3V de alimentación entre sus pines **VCC** y **GND**, debido a que tiene un regulador interno que reduce la tensión ingresada a ser siempre de 3.3V.
- **SCL** y **SDA:** Estos pines son los que establecen la **comunicación I2C** del sensor.
    - **SDA:** Pin que manda o recibe datos de 16 bits por medio de la comunicación serial I2C.
        - En la tarjeta **Arduino UNO** **el pin A4 corresponde al SDA de comunicación I2C**.
    - **SCL:** Pin que recibe una **señal de reloj para marcar el paso de la transmisión de datos**, cuya frecuencia es la misma de los bits que conforman la información transportada. **Cada que se envíe un dato, se activará la señal de reloj, sino esta se mantiene con valor de 0 lógico.**
        - En la tarjeta **Arduino UNO** **el pin A5 corresponde al SCL de comunicación I2C**.
- **AD0:** A través de este pin se pueden conectar varios módulos MPU entre sí, debido a la comunicación I2C donde se puede tener varios SLAVES, si este está conectado a GND, el address del MPU6050 será 0X68, pero si está conectado a 5V, el address será 0X69 hexadecimal.
- **INT:** Este pin genera interrupciones externas para que cuando se cense un cambio en la orientación del sensor, este dato tome prioridad en el código del microcontrolador.



Cuando se vaya a efectuar un control con un módulo MPU6050 en Arduino, es recomendable conectar su **pin INT a uno de los puertos de interrupciones de la placa de desarrollo**, para que así no importando que es lo que esté haciendo el código en ese momento, ponga en pausa su instrucción actual y haga caso al dato de orientación mandado por el sensor, esto es más fácil implementarlo con la librería **i2cdevlib**, no con la **Wire**.

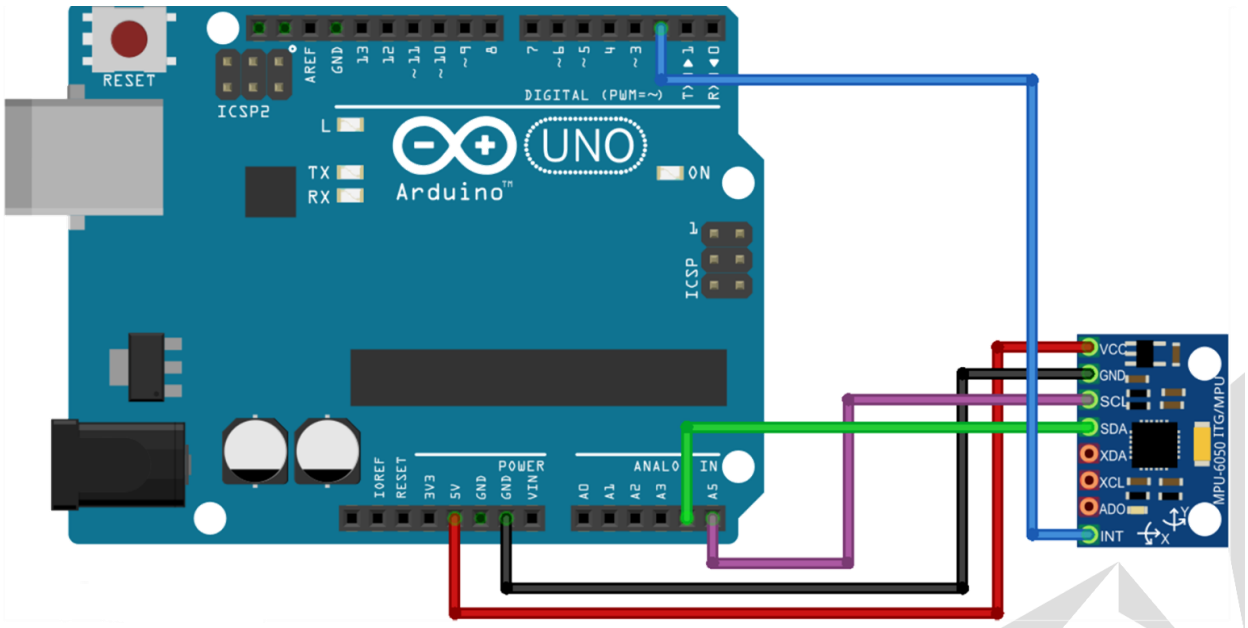


MODELO	PINES INTERRUPCIONES
Uno, Nano, Mini y otras basadas en 328	2, 3
Mega, Mega2560 y MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo y otras basadas en 32u4	0, 1, 2, 3, 7
Zero	todos los pines digitales excepto el 4
Due	todos los pines digitales

Diagrama de conexión con el Arduino

MPU6050	Arduino	Mega
VCC	5V	5V
GND	GND	GND
SCL	A5	21
SDA	A4	20
INT	2	2

AD0	Dirección I2C
GND (sin conexión)	0x68
VCC	0x69



Código Arduino – Acelerómetro y Giróscopo MPU6050 con Librería Wire:

Al usar la **librería Wire** no se conecta el Pin 2 de interrupciones externas y los cálculos se realizan de forma manual al acceder directamente a los registros del **acelerómetro** y **giroscopio**, pero **el problema de hacer esto es que se arrastra un error que aumenta paulatinamente en la lectura del Yaw**, aunque a veces se puede extender a las otras lecturas de orientación Roll y Pitch:

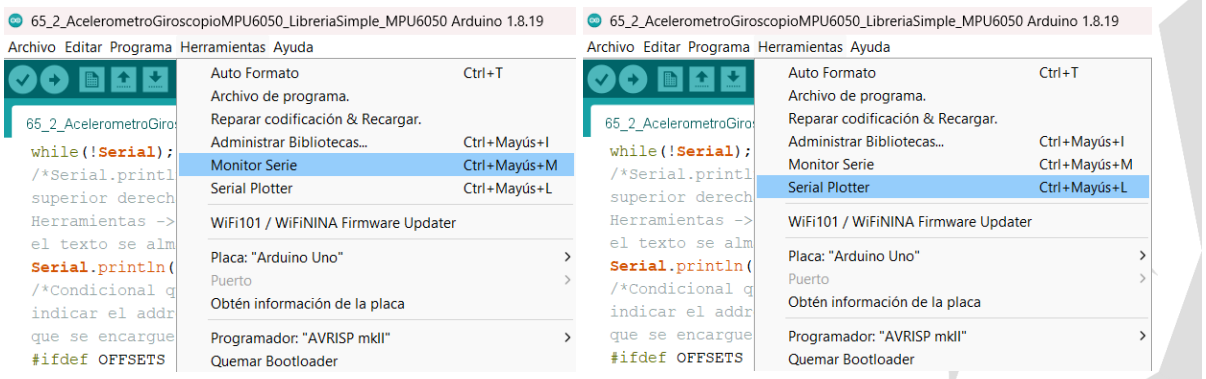
```
/*65.1.-El MPU6050 es un módulo de tipo IMU (Inertial Measure Unit), el cual sirve para medir la orientación de un dispositivo,
tanto en inclinación, como en su dirección en el plano 2D, para ello incluye un acelerómetro que detecta la aceleración lineal
y un giróscopo que censa la aceleración angular, esto se realiza a través de 6 grados de libertad y el sensor se debe colocar
en el centro de masa del dispositivo que se quiere controlar, para así lograr una medición correcta. Los datos recibidos del
acelerómetro y giróscopo serán combinados para medir de forma precisa la orientación. Aunque, en el programa que utiliza la
librería Wire existe el problema de que el YAW tiene un error que aumenta al pasar el tiempo, además de que la calibración se
debe realizar de forma manual, pero solamente se utilizará de forma didáctica y luego se mostrará un programa que utilice la
librería i2cdevlib que realiza una calibración automática, además de que utiliza el DMP (Digital Motion Processor) incluido
en el sensor para realizar los cálculos también de manera automática. Este sensor es muy utilizado en aeronaves, celulares,
drones, gimbals, etc.*/
//IMPORTACIÓN DE LA LIBRERÍA WIRE:
#include <Wire.h> //Librería que habilita la comunicación I2C en los pines A4 (SDA) y A5 (SCL) del Arduino.

//DECLARACIÓN DE VARIABLES:
//Si el PIN A0 está conectado a GND o a nada, el address del MPU6050 será 0X68, pero si está conectado a 5V, será 0X69.
const int MPU = 0X68;                                          //Dirección I2C del SLAVE MPU6050 = 0X68.
//Datos crudos recabados del sensor MPU6050:
float AccX, AccY, AccZ;                                        //3 grados de libertad acelerómetro.
float GyroX, GyroY, GyroZ;                                     //3 grados de libertad giroscopio.
//Datos previos a la orientación resultante después de ejecutar las primeras operaciones matemáticas sobre los datos crudos:
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;   //Orientación: Dirección XY y ángulo de inclinación 3D.
float accErrorX, accErrorY, gyroErrorX, gyroErrorY, gyroErrorZ;   //Errores de orientación XY y ángulo de inclinación.
int pruebasError = 0;                                          //Número de pruebas realizadas para promediar el error.
//Resultados de orientación: Roll (giro eje x), Pitch (giro eje y) y Yaw (giro eje z).
float roll, pitch, yaw;
float elapsedTime, currentTime, previousTime;                  //Tiempos recabados por medio del método millis().

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL:
void setup() {
  /*Serial.begin(baudRate): Este método inicializa la comunicación serial entre la placa Arduino y la computadora, además de
  que configura su velocidad de transmisión dada en unidad de baudios (bit trasmitido por segundo) que recibe como su único
  parámetro:
      - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es compatible con la mayoría de los
        dispositivos y programas.
      - Si se necesita una transferencia de datos más rápida y el hardware/software lo admiten, se puede optar por velocidades
        más altas como 115200 o 57600 baudios, pero en comunicación I2C habilitada por la librería Wire, se pueden usar
        velocidades de 19,200 hasta 115,200 baudios.
  Es importante asegurarse de que la velocidad de transmisión especificada coincida con la velocidad de comunicación del otro
  dispositivo al que se conecta el Arduino. Si la velocidad de transmisión no coincide, los datos pueden no transmitirse o
  recibirse correctamente.*/
  Serial.begin(19200);          //Velocidad de transmisión serial I2C: 19,200 baudios.
  /*Wire.begin(): Método que inicializa la comunicación I2C, después de este se deberá indicar con qué dirección de SLAVE se
  está estableciendo la conexión y la velocidad del reloj.*/
  Wire.begin();                 //Método que inicia la comunicación I2C.
  /*Wire.beginTransmission(dirección): Método que permite a un dispositivo mandar o recibir datos de una dirección SLAVE en
  específico. MASTER es un dispositivo que puede mandar datos y SLAVE es uno que los puede recibirlos mediante el protocolo
  I2C, pero para que un SLAVE pueda recibir datos se debe indicar su dirección, ya que un MASTER puede mandar datos a varios
  SLAVE a la vez.*/
  //CONFIGURACIÓN DE LA SEÑAL DE RELOJ Y MODO DE BAJO CONSUMO DEL MPU6050:
  Wire.beginTransmission(MPU);  //Método que indica a qué dirección de SLAVE se enviarán datos con el protocolo I2C.
  /*Wire.write(): Método que se debe usar dos veces, la primera vez permite indicar a qué registro del SLAVE se quiere acceder
  y la segunda vez permite mandar un byte de información a dicho registro del SLAVE, cuya dirección fue previamente
  inicializada con el método Wire.beginTransmission().
  El registro 6B = 107 del MPU6050 permite encender un modo de bajo consumo e indicar cuál será la fuente de la señal de reloj.
  Mandando el valor de 0X00 se indica que el modo de bajo consumo se apague y que la señal de reloj sea la interna del módulo
  de 8MHz.*/
  Wire.write(0x6B);             //Método que accede al registro 6B del SLAVE con dirección 0X68.
  Wire.write(0x00);             //Método que escribe el valor 00 en el registro 6B del SLAVE con dirección 0X68.
  Wire.endTransmission(true);   //Método que finaliza la transmisión de datos mandados a cualquier dirección de SLAVE.
  //CONFIGURACIÓN DE LA SENSIBILIDAD DE MEDICIÓN DE LA ACELERACIÓN EN EL MPU6050:
  Wire.beginTransmission(MPU);  //Método que indica a qué dirección de SLAVE se enviarán datos con el protocolo I2C.
  /*El registro 1C = 28 del MPU6050 permite indicar el rango de medición del acelerómetro, yendo desde ±2g hasta ±16g, se elije
  un rango mayor cuando el dispositivo al que se quiere incorporar el MPU6050 estará sometido a aceleraciones muy grandes o
  expuesto a fuerzas extremas, como lo puede ser en vehículos y aeronaves, monitoreo de impactos, robótica y sistemas de
  movimiento rápido, etc. Mandando el valor de 0X00 se indica que el rango de medición del acelerómetro sea el mínimo de ±2g.*/
  Wire.write(0x1C);  //Método que accede al registro 1C del SLAVE con dirección 0X68.
  Wire.write(0x00);  //Método que escribe el valor 00 en el registro 1C del SLAVE con dirección 0X68, obteniendo un rango de ±2g.
  Wire.endTransmission(true);   //Método que finaliza la transmisión de datos mandados a cualquier dirección de SLAVE.
  //CONFIGURACIÓN DE LA SENSIBILIDAD DE MEDICIÓN DE LA ACELERACIÓN EN EL MPU6050:
  Wire.beginTransmission(MPU);  //Método que indica a qué dirección de SLAVE se enviarán datos con el protocolo I2C.
  /*El registro 1B = 27 del MPU6050 permite indicar el rango de medición del giroscopio, yendo desde ±250°/s hasta ±2000°/s, se
  elije un rango mayor cuando el dispositivo al que se quiere incorporar el MPU6050 estará sometido a aceleraciones muy grandes
  o expuesto a fuerzas extremas, como lo puede ser en vehículos y aeronaves, monitoreo de impactos, robótica, sistemas de
  movimiento rápido, etc. Mandando el valor de 0X00 se indica que el rango de medición del acelerómetro sea el mínimo de
  ±250°/s.*/
  Wire.write(0x1B);         //Método que accede al registro 1B del SLAVE con dirección 0X68.
  Wire.write(0x00);         //Escribe el valor 00 en el registro 1B del SLAVE con dirección 0X68, obteniendo un rango de ±250°/s.
  Wire.endTransmission(true);   //Método que finaliza la transmisión de datos mandados a cualquier dirección de SLAVE.
  //FUNCIÓN PROPIA DE ESTE CÓDIGO PARA CALCULAR EL ERROR DE LOS DATOS RECABADOS DEL ACELERÓMETRO GIROSCOPIO MPU6050:
  calcularErrorSensor();
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos.*/
  delay(20);
```

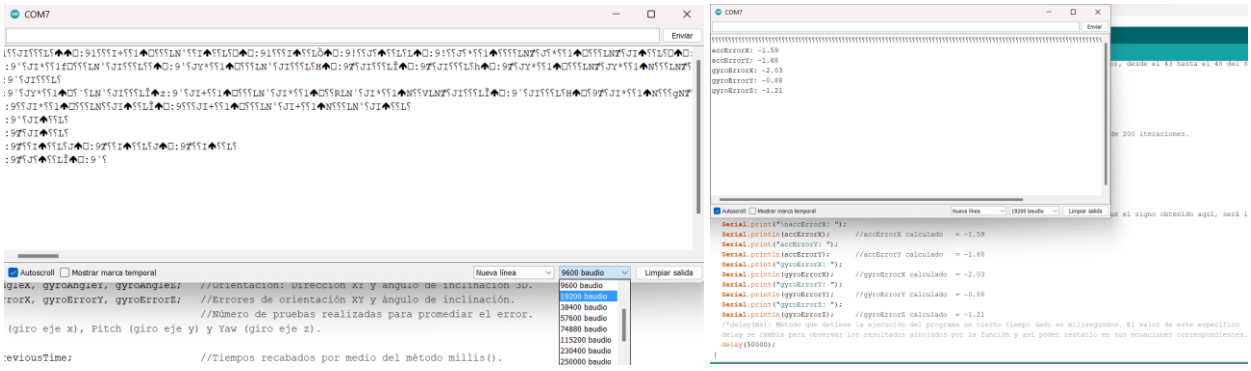
```
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO: Operaciones matemáticas que calculan el sentido de giro y orientación del sensor.
void loop() {
  //LECTURA DE LOS DATOS DEL ACELERÓMETRO:
  /*En los 6 registros 3B, 3C, 3D, 3E, 3F y 40, cuyo correspondiente decimal son los 59, 60, ..., 64, viene contenida la
  información de los acelerómetros correspondientes a los 3 ejes X, Y, Z. Estos vienen agrupados en dos registros diferentes
  porque la información completa es de 16 bits, pero cada registro como máximo puede transportar 8.*/
  Wire.beginTransmission(MPU);  //Método que indica a qué dirección de SLAVE se enviarán datos con el protocolo I2C.
  Wire.write(0x3B);             //Método que accede al registro 3B del SLAVE con dirección 0X68.
  Wire.endTransmission(false);  //Método que deja abierta la transmisión de datos mandados a cualquier dirección de SLAVE.
  /*Wire.requestFrom(address, quantity, stop): Método que permite solicitar datos a un dispositivo SLAVE conectado al bus I2C,
  para ello previamente se tuvieron que haber ejecutado los métodos: Wire.beginTransmission(address), Wire.write(registro) y
  Wire.endTransmission(false), ya que con ellos se indica a partir de qué registro de un SLAVE en específico se extraerá el
  número de datos con longitud de 8 bits (byte) indicado en este método.*/
  //Método que extrae datos de 6 registros (2 por acelerómetro), desde el 3B hasta el 40 del SLAVE con dirección 0X68.
  Wire.requestFrom(MPU, 6, true);
  /*Wire.read(): Permite leer un dato de un SLAVE, para ello previamente se tuvo que haber ejecutado el método
  Wire.requestFrom(address, quantity, stop). Como la aceleración X, Y, Z se comparte a través de 6 registros, donde cada 2
  corresponden a una medición de 16 bits, se realiza la operación shift << y luego se aplica una compuerta OR | para primero
  recorrer 8 bits a la derecha el contenido del primer registro dentro de un número binario y luego unir eso con el contenido
  del segundo registro, obteniendo al final la información completa de 16 bits en una misma variable. Dependiendo del rango de
  medición del acelerómetro, yendo desde ±2g hasta ±16g, se deberá dividir el dato obtenido de cada una de las 3 aceleraciones
  entre cierto número, indicado en la página 29 del datasheet, bajo el nombre de LSB sensitibity. Para ±2g, LSB = 16,384.*/
  AccX = (Wire.read() << 8 | Wire.read())/16384.0;  //Aceleración lineal medida en el eje X.
  AccY = (Wire.read() << 8 | Wire.read())/16384.0;  //Aceleración lineal medida en el eje Y.
  AccZ = (Wire.read() << 8 | Wire.read())/16384.0;  //Aceleración lineal medida en el eje Z.

  //CALCULO DE LOS ÁNGULOS DE ROTACIÓN ALREDEDOR DEL EJE "X" (ROLLL), "Y" (PITCH): Se deben convertir a grados, porque
  //inicialmente vienen en radianes.
  //CORRECCIÓN DE LAS SALIDAS DEL ACELERÓMETRO CON EL ERROR CALCULADO POR MEDIO DE LA FUNCIÓN PROPIA calcularErrorSensor():
  //accErrorX obtenido de la función calcularErrorSensor() = -1.42
  //Inclinación alrededor del eje X = Roll  = arctan(Ay/√(Ax^2+Az^2))
  accAngleX = (atan((AccY)/sqrt(pow(AccX, 2) + pow(AccZ, 2))) * (180/PI)) - (-1.42);
  //accErrorY obtenido de la función calcularErrorSensor() = -3.24
  //Inclinación alrededor del eje Y = Pitch = arctan(Ax/√(Ay^2+Az^2))
  accAngleY = (atan(-1*(AccX)/sqrt(pow(AccY, 2) + pow(AccZ, 2))) * (180/PI)) - (-3.24);
  //LECTURA DE LOS DATOS DEL GIROSCOPIO:
  previousTime = currentTime;   //Variable que se actualiza para medir intervalos de tiempo.
  /*millis(): Método que devuelve el tiempo transcurrido en milisegundos, empezando a contar desde que se enciende la placa
  Arduino y no deteniéndose hasta que esta se apague o llegue a su límite, que es el mismo dado por el tipo de dato unsigned
  long: De 0 a 4,294,967,295 milisegundos = 1,193.0464 horas = 49.7102 días = 49 días y 17 horas.
  Cada que se utilice el método millis, se estará actualizando el tiempo guardado en una variable cualquiera, en este caso se
  guarda en dos variables para crear un temporizador que mida un tiempo de duración, realizando una simple resta entre dos
  medidas de tiempo.*/
  currentTime = millis();       //Guarda el tiempo transcurrido desde que se prendió el Arduino.
  elapsedTime = (currentTime - previousTime)/1000;  //Intervalo de tiempo medido en segundos.
  /*En los 6 registros 43, 44, 45, 46, 47 y 48, cuyo correspondiente decimal son los 67, 68, ..., 72, viene contenida la
  información de los giróscopios correspondientes a los 3 ejes X, Y, Z. Estos vienen agrupados en dos registros diferentes
  porque la información completa es de 16 bits, pero cada registro como máximo puede transportar 8.*/
  Wire.beginTransmission(MPU);  //Método que indica a qué dirección de SLAVE se enviarán datos con el protocolo I2C.
  Wire.write(0x43);             //Método que accede al registro 43 del SLAVE con dirección 0X68.
  Wire.endTransmission(false);  //Método que deja abierta la transmisión de datos mandados a cualquier dirección de SLAVE.
  /*Wire.requestFrom(address, quantity, stop): Método que permite solicitar datos a un dispositivo SLAVE conectado al bus I2C,
  para ello previamente se tuvieron que haber ejecutado los métodos: Wire.beginTransmission(address), Wire.write(registro) y
  Wire.endTransmission(false), ya que con ellos se indica a partir de qué registro de un SLAVE en específico se extraerá el
  número de datos con longitud de 8 bits (byte) indicado en este método.*/
  //Método que extrae datos de 6 registros (2 por giróscopo), desde el 43 hasta el 48 del SLAVE con dirección 0X68.
  Wire.requestFrom(MPU, 6, true);
  /*Wire.read(): Permite leer un dato de un SLAVE, para ello previamente se tuvo que haber ejecutado el método
  Wire.requestFrom(address, quantity, stop). Como la aceleración angular X, Y, Z se comparte a través de 6 registros, donde
  cada 2 corresponden a una medición de 16 bits, se realiza la operación shift << y luego se aplica una compuerta OR | para
  primero recorrer 8 bits a la derecha el contenido del primer registro dentro de un número binario y luego unir eso con el
  contenido del segundo registro, obteniendo al final la información completa de 16 bits en una misma variable. Dependiendo del
  rango de medición del acelerómetro, yendo desde ±2g hasta ±16g, se deberá dividir el dato obtenido de cada una de las 3
  aceleraciones entre cierto número, indicado en la página 31 del datasheet, bajo el nombre de LSB sensitibity. Para ±250°/s,
  LSB sensitibity = 131.*/
  GyroX = (Wire.read() << 8 | Wire.read())/131.0;  //Aceleración angular medida en el eje X.
  GyroY = (Wire.read() << 8 | Wire.read())/131.0;  //Aceleración angular medida en el eje Y.
  GyroZ = (Wire.read() << 8 | Wire.read())/131.0;  //Aceleración angular medida en el eje Z.

  //CORRECCIÓN DE LAS SALIDAS DEL GIROSCOPIO CON EL ERROR CALCULADO POR MEDIO DE LA FUNCIÓN PROPIA calcularErrorSensor():
  GyroX = GyroX - (-2.01); //gyroErrorX = -2.01
  GyroY = GyroY - (-0.79); //gyroErrorY = -0.79
  GyroZ = GyroZ - (-1.02); //gyroErrorZ = -1.02

  //CÁLCULO DEL ÁNGULO DE ROTACIÓN XY AL MULTIPLICAR EL DATO DEL GIROSCOPIO °/S POR EL TIEMPO DE DURACIÓN DEL TEMPORIZADOR,
  //ESTO ES EL EQUIVALENTE A INTEGRAR EL VALOR DE LA VELOCIDAD ANGULAR PARA OBTENER EL ÁNGULO Y ADEMÁS SE OBTIENE EL VALOR DE
  //YAW:
  gyroAngleX = gyroAngleX + GyroX * elapsedTime;    //gyroAngleX = °/s * s = grados.
  gyroAngleY = gyroAngleY + GyroY * elapsedTime;    //gyroAngleY = °/s * s = grados.
  yaw = yaw + GyroZ * elapsedTime;                  //gyroAngleZ = yaw = °/s * s = grados.

  //EL ÁNGULO EN XY MEDIDO CON EL ACELERÓMETRO SE COMBINA CON EL ÁNGULO XY MEDIDO CON EL GIRÓSCOPO PARA OBTENER EL RESULTADO DE
  //ROLL Y PITCH, ESTE SE CONSIDERA COMO UN FILTRO DE LA SEÑAL:
  roll  = 0.96*gyroAngleX + 0.04*accAngleX;
  pitch = 0.96*gyroAngleY + 0.04*accAngleY;
  /*Serial.println(): Método que escribe un mensaje en la consola serial de Arduino, a la cual se puede acceder en la esquina
  superior derecha donde se encuentra una lupa, pero si en vez de usar el Monitor Serie, nos introducimos en la opción de
  Herramientas -> Serial Plotter, podremos ver tres gráficas que indican el cambio de cada magnitud.*/
  Serial.print("Roll:" + String(roll));
  Serial.print(String(", "));
  Serial.print("Pitch:" + String(pitch));
  Serial.print(String(", "));
  Serial.print("Yaw:" + String(yaw));
```

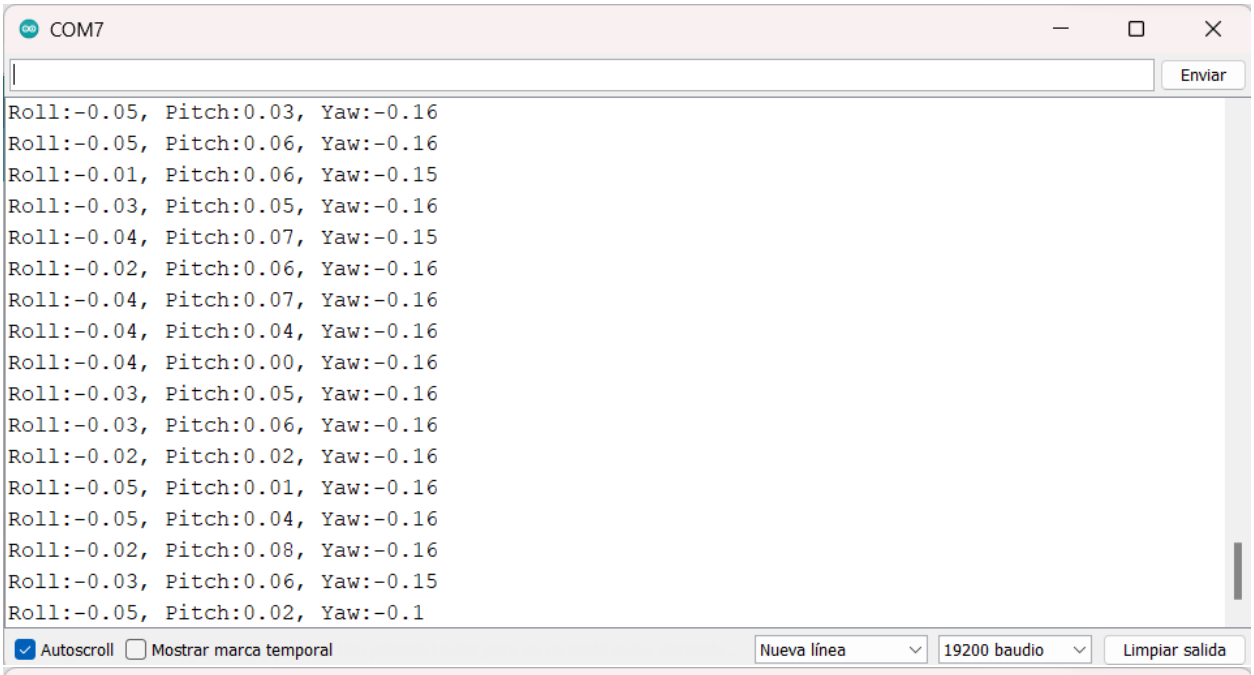
```
  Serial.print(String("\n"));
}

//FUNCIÓN PROPIA calcularErrorSensor(), PARA OBTENER UNA BUENA LECTURA DEL ERROR, EL SENSOR SE DEBE ENCONTRAR INICIALMENTE EN
//UNA ESTRUCTURA PLANA:
void calcularErrorSensor(){
  /*Para calcular el error se realizan 200 lecturas durante el setup del programa, se suma el resultado de todas ellas y luego
  se divide entre 200 para obtener así un promedio del error, este se imprimirá en pantalla y se deberá restar en el cálculo de
  la función loop() para así hacer la corrección de dicho error.*/
  //CÁLCULO DEL ERROR EN EL ACELERÓMETRO:
  while(pruebasError < 200){
    Wire.beginTransmission(MPU);  //Método que indica a qué dirección de SLAVE se enviarán datos con el protocolo I2C.
    Wire.write(0x3B);             //Método que accede al registro 3B del SLAVE con dirección 0X68.
    Wire.endTransmission(false);  //Método que deja abierta la transmisión de datos mandados a cualquier dirección de SLAVE.
    //Método que extrae datos de 6 registros (2 por acelerómetro), desde el 3B hasta el 40 del SLAVE con dirección 0X68.
    Wire.requestFrom(MPU, 6, true);
    AccX = (Wire.read() << 8 | Wire.read())/16384.0;  //Aceleración lineal medida en el eje X.
    AccY = (Wire.read() << 8 | Wire.read())/16384.0;  //Aceleración lineal medida en el eje Y.
    AccZ = (Wire.read() << 8 | Wire.read())/16384.0;  //Aceleración lineal medida en el eje Z.
    accErrorX = accErrorX + (atan((AccY)/sqrt(pow(AccX, 2) + pow(AccZ, 2))) * (180/PI));   //Actualización del error en el eje X.
    accErrorY = accErrorY + (atan(-1*(AccX)/sqrt(pow(AccY, 2) + pow(AccZ, 2))) * (180/PI));//Actualización del error en el eje Y.
    pruebasError++;               //Aumento de la variable pruebasError, para ejecutar el loop de 200 iteraciones.
  }
  //CÁLCULO DEL ERROR PROMEDIO EN EL ACELERÓMETRO:
  accErrorX = accErrorX / 200;
  accErrorY = accErrorY / 200;
  pruebasError = 0;               //Reinicio de la variable pruebasError
  //CÁLCULO DEL ERROR EN EL GIROSCOPIO:
  while(pruebasError < 200){
    Wire.beginTransmission(MPU);  //Método que indica a qué dirección de SLAVE se enviarán datos con el protocolo I2C.
    Wire.write(0x43);             //Método que accede al registro 43 del SLAVE con dirección 0X68.
    Wire.endTransmission(false);  //Método que deja abierta la transmisión de datos mandados a cualquier dirección de SLAVE.
    //Método que extrae datos de 6 registros (2 por acelerómetro), desde el 43 hasta el 48 del SLAVE con dirección 0X68.
    Wire.requestFrom(MPU, 6, true);
    GyroX = (Wire.read() << 8 | Wire.read());     //Aceleración angular medida en el eje X.
    GyroY = (Wire.read() << 8 | Wire.read());     //Aceleración angular medida en el eje Y.
    GyroZ = (Wire.read() << 8 | Wire.read());     //Aceleración angular medida en el eje Z.
    gyroErrorX = gyroErrorX + (GyroX / 131.0);    //Actualización del error en el eje X.
    gyroErrorY = gyroErrorY + (GyroY / 131.0);    //Actualización del error en el eje Y.
    gyroErrorZ = gyroErrorZ + (GyroZ / 131.0);    //Actualización del error en el eje Z.
    pruebasError++;               //Aumento de la variable pruebasError, para ejecutar el loop de 200 iteraciones.
  }
  //CÁLCULO DEL ERROR PROMEDIO EN EL GIRÓSCOPO:
  gyroErrorX = gyroErrorX / 200;
  gyroErrorY = gyroErrorY / 200;
  gyroErrorZ = gyroErrorZ / 200;
  pruebasError = 0;               //Reinicio de la variable pruebasError

  //IMPRESIÓN EN CONSOLA DEL PROMEDIO DE LOS ERRORES: Estps siempre se deberán restar, por lo que el signo obtenido aquí, será
  //invertido en el código.
  Serial.print("\naccErrorX: ");
  Serial.println(accErrorX);      //accErrorX calculado   = -1.42
  Serial.print("accErrorY: ");
  Serial.println(accErrorY);      //accErrorY calculado   = -3.24
  Serial.print("gyroErrorX: ");
  Serial.println(gyroErrorX);     //gyroErrorX calculado  = -2.01
  Serial.print("gyroErrorY: ");
  Serial.println(gyroErrorY);     //gyroErrorY calculado  = -0.79
  Serial.print("gyroErrorZ: ");
  Serial.println(gyroErrorZ);     //gyroErrorZ calculado  = -1.02
  /*delay(ms): Método que detiene la ejecución del programa un cierto tiempo dado en milisegundos. El valor de este específico
  delay se cambia para observar los resultados arrojados por la función y así poder restarlo en sus ecuaciones
  correspondientes.*/
  delay(1000);
}
```

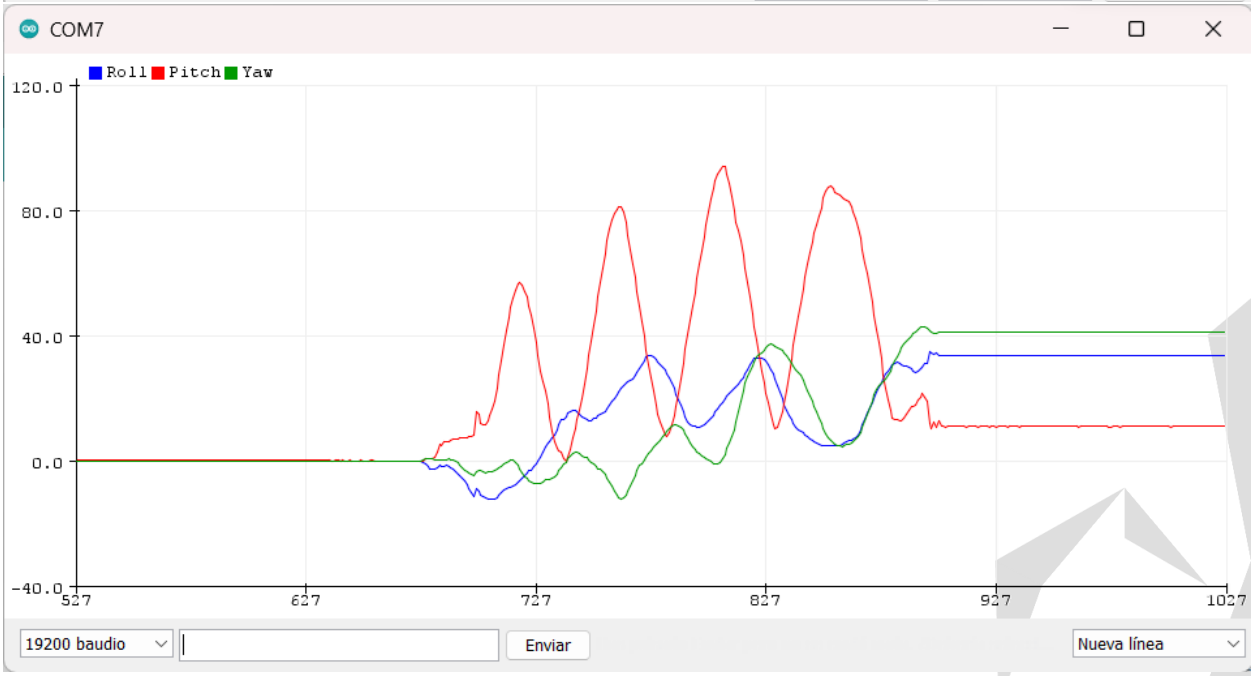
Resultados de orientación: Roll, Pitch y Yaw

Para observar los resultados de las señales Roll, Pitch y Yaw se puede utilizar el monitor serie para ver solo números o el Serial Plotter, para ver el resultado en forma de gráficas.






COM7
Enviar
Roll:-0.05, Pitch:0.03, Yaw:-0.16
Roll:-0.05, Pitch:0.06, Yaw:-0.16
Roll:-0.01, Pitch:0.06, Yaw:-0.15
Roll:-0.03, Pitch:0.05, Yaw:-0.16
Roll:-0.04, Pitch:0.07, Yaw:-0.15
Roll:-0.02, Pitch:0.06, Yaw:-0.16
Roll:-0.04, Pitch:0.07, Yaw:-0.16
Roll:-0.04, Pitch:0.04, Yaw:-0.16
Roll:-0.04, Pitch:0.00, Yaw:-0.16
Roll:-0.03, Pitch:0.05, Yaw:-0.16
Roll:-0.03, Pitch:0.06, Yaw:-0.16
Roll:-0.02, Pitch:0.02, Yaw:-0.16
Roll:-0.05, Pitch:0.01, Yaw:-0.16
Roll:-0.05, Pitch:0.04, Yaw:-0.16
Roll:-0.02, Pitch:0.08, Yaw:-0.16
Roll:-0.03, Pitch:0.06, Yaw:-0.15
Roll:-0.05, Pitch:0.02, Yaw:-0.1
Autoscroll
Mostrar marca temporal
Nueva línea
19200 baudio
Limpiar salida


COM7
Roll
Pitch
Yaw
120.0
80.0
40.0
0.0
-40.0
527
627
727
827
927
1027
19200 baudio
Enviar
Nueva línea

Código Arduino – Sensor MPU6050 con Librería i2cdevlib y Simple_MPU6050:

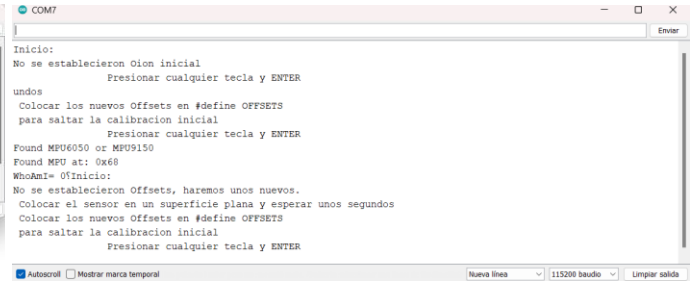
Al utilizar la **librería i2cdevlib** para establecer la comunicación I2C y la **librería Simple_MPU6050** para obtener las señales de orientación, si se conecta el Pin 2 de interrupciones externas y los resultados se obtienen de forma automática a través del **DMP (Digital Movement Processor)**, este nos evita tener que acceder directamente a los registros del **acelerómetro** y **giroscopio**, **evitando que se cree el error de lectura del Yaw**:

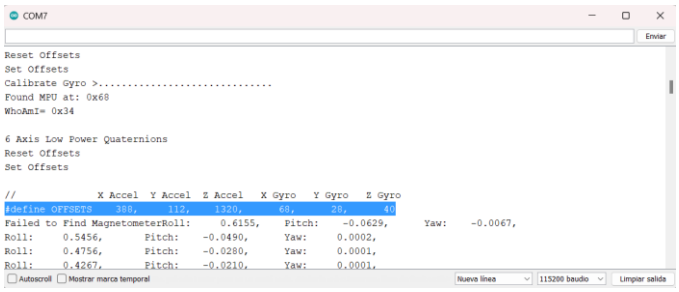
```
/*65.2.-El MPU6050 es un módulo de tipo IMU (Inertial Measure Unit), el cual sirve para medir la orientación de un dispositivo,
tanto en inclinación, como en su dirección en el plano 2D, para ello incluye un acelerómetro que detecta la aceleración lineal
y un giróscopo que censa la aceleración angular, esto se realiza a través de 6 grados de libertad y el sensor se debe colocar
en el centro de masa del dispositivo que se quiere controlar, para así lograr una medición correcta. Cuando se utilice la
librería i2cdevlib para establecer la comunicación I2C y la biblioteca Simple_MPU6050 para el manejo del sensor, los datos
recibidos del acelerómetro y giróscopo serán combinados y procesados para obtener la orientación de forma automática por medio
del DMP (Digital Motion Processor) incluido dentro del sensor, además de que elimina el error acumulativo en la lectura del Yaw.
Este sensor es muy utilizado en aeronaves, celulares, drones, gimbals, etc.*/
//IMPORTACIÓN DE LA LIBRERÍA Simple_MPU6050, QUE UTILIZA LAS LIBRERÍAS i2cdevlib Y Simple_Wire:
#include "Simple_MPU6050.h" //Librería que habilita la comunicación I2C en los pines A4 (SDA), A5 (SCL) y Pin2 (INT) del Arduino.

//DECLARACIÓN DE CONSTANTES:
//Si el PIN A0 está conectado a GND o a nada, el address del MPU6050 será 0X68, pero si está conectado a 5V, será 0X69.
#define MPU6050_ADDRESS_AD0_LOW   0X68                    //Adress del MPU6050 si el Pin A0 está conectado a GND.
#define MPU6050_ADDRESS_AD0_HIGH  0X69                    //Adress del MPU6050 si el Pin A0 está conectado a 5V.
#define MPU6050_DEFAULT_ADDRESS   MPU6050_ADDRESS_AD0_LOW  //Dirección I2C del SLAVE MPU6050 = 0X68.
/*La calibración se realiza de manera automática por el DMP incluido en el módulo MPU6050, pero si por alguna razón esos valores
calculados son erróneos, también se pueden incluir de forma manual. Cabe mencionar que los datos calculados automáticamente por
el sensor, siempre se muestran hasta el principio de la lectura de valores en consola y si estos no se incluyen en el código,
al ejecutar el programa aparecerá un mensaje en pantalla que no lo dejará correr hasta que introduzcamos cualquier letra y demos
clic en enter, al agregar la constante OFFSETS esto se deja de mostrar.*/
#define OFFSETS    316,     112,    1304,      70,      28,      36 //Valores personalizados de calibración para el sensor.

//DECLARACIÓN DE OBJETOS:
/*Objeto de la clase Simple_MPU6050 que permite establecer la comunicación I2C con el sensor MPU6050, obteniendo los
datos de aceleración lineal y rotativa para realizar cálculos con ellos y obtener la orientación del dispositivo.*/
Simple_MPU6050 mpu;

//FUNCIONES PROPIAS DECLARADAS EN UNA LÍNEA:
/*La sintaxis utilizada en C++, que es el lenguaje que se usa para programar los microcontroladores de Arduino, es la sig:
#define   nombreFunción(parámetros)   Acción a ejecutar*/
/*spamtimer(): Función propia de 1 sola línea que crea un delay para evitar problemas al mostrar los datos de orientación
en consola.*/
#define spamtimer(t) for (static uint32_t SpamTimer; (uint32_t) (millis() - SpamTimer) >= (t); SpamTimer = millis())
/*printfloatx(): Función propia de 1 sola línea que proporciona una forma personalizada de imprimir datos en consola, para
ello recibe 5 parámetros:
  Texto que muestra el nombre de la variable,
  variable,
  Número de caracteres que muestran el valor de la variable,
  número de decimales mostrados,
  texto puesto al final.
Esto es muy útil al mostrar los resultados de orientación Roll, Pitch y Yaw obtenidos del sensor MPU6050.*/
#define printfloatx(Name,Variable,Spaces,Precision,EndTxt) print(Name); {char S[(Spaces + Precision + 3)];Serial.print(F(" "));
Serial.print(dtostrf((float)Variable,Spaces,Precision ,S));}Serial.print(EndTxt);

//CONFIGURACIÓN DE LOS PINES Y LA COMUNICACIÓN SERIAL:
void setup() {
  /*uint8_t: Número binario sin signo que puede adoptar valores de 0 a 255. La presencia de la letra "t" del final significa
  "tipo" y se utiliza para hacer que la variable sea portátil y consistente en diferentes plataformas.*/
  uint8_t val;
  /*Condicional que evalúa a través de un condicional if cuál herramienta de la librería i2cdevlib se utilizará para implementar
  la comunicación del protocolo I2C, ya sea la proporcionada por la biblioteca Wire de Arduino (I2CDEV_ARDUINO_WIRE) o una
  implementación optimizada llamada Fastwire (I2CDEV_BUILTIN_FASTWIRE). En ambas opciones de inicialización se elige una señal
  de reloj de 400,000 Hz = 400 kHz. Se utiliza la sintaxis de directiva #if para el condicional porque en esta parte se está
  analizando una condición de la configuración del programa, por lo que se ejecuta en el momento de la compilación.*/
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    /*Wire.begin(): Método que inicializa la comunicación I2C, después de este se deberá indicar con qué dirección de SLAVE se
    está estableciendo la conexión y la velocidad del reloj.*/
    Wire.begin();              //Método que inicia la comunicación I2C con la librería Wire.
    /*Wire.setClock(): Método que configura la velocidad de transmisión (frecuencia del reloj) del bus I2C en Arduino.*/
    Wire.setClock(400000);     //Reloj de 400,000 Hz = 400 kHz = 2.5 µs.
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true); //Método que inicia la comunicación I2C con la librería Fastwire.
  #endif
  /*Serial.begin(baudRate): Este método inicializa la comunicación serial entre la placa Arduino y la computadora, además de
  que configura su velocidad de transmisión dada en unidad de baudios (bit trasmitido por segundo) que recibe como su único
  parámetro:
      - En general, 9600 baudios es una velocidad de transmisión comúnmente utilizada y es compatible con la mayoría de los
        dispositivos y programas.
      - Si se necesita una transferencia de datos más rápida y el hardware/software lo admiten, se puede optar por velocidades
        más altas como 115200 o 57600 baudios, pero en comunicación I2C habilitada por la librería Wire, se pueden usar
        velocidades de 19,200 hasta 115,200 baudios.
  Es importante asegurarse de que la velocidad de transmisión especificada coincida con la velocidad de comunicación del otro
  dispositivo al que se conecta el Arduino. Si la velocidad de transmisión no coincide, los datos pueden no transmitirse o
  recibirse correctamente.*/
  Serial.begin(115200);  //Baudios = 115,200
  while(!Serial);
  /*Serial.println(): Método que escribe un mensaje en la consola serial de Arduino, a la cual se puede acceder en la esquina
  superior derecha donde se encuentra una lupa, pero si en vez de usar el Monitor Serie, nos introducimos en la opción de:
  Herramientas -> Serial Plotter, podremos ver tres gráficas que indican el cambio de cada magnitud. La función F(), permite que
```

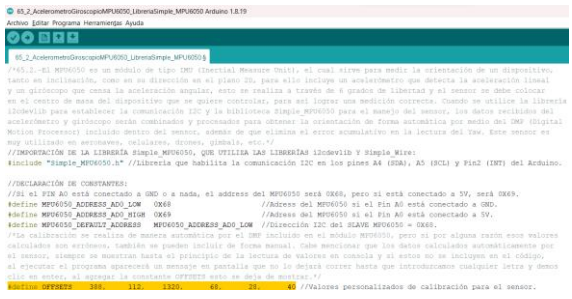
```
  el texto se almacene en la memoria de programa, lo que ayuda a ahorrar espacio en la memoria RAM.*/
  Serial.println(F("Inicio: "));
  /*Condicional que evalúa si se han declarado valores de calibración para el sensor en la directiva OFFSETS o no, luego pasa a
  indicar el address SLAVE del MPU6050, calibrar su rango de medición y cargar dichos datos al DMP (Digital Motion Processor)
  para que se encargue de realizar los cálculos de orientación.*/
  #ifdef OFFSETS
    Serial.println(F("Usando offsets predefinidos"));
    /*Simple_MPU6050.SetAddress(): Este método de la librería Simple_MPU6050 lo que hace es establecer la dirección SLAVE del
    sensor MPU6050 para poder efectuar una comunicación I2C entre el Arduino y él.
    Simple_MPU6050.CalibrateMPU(): Este método de la librería Simple_MPU6050 lo que hace es indicar el rango de medición del
    acelerómetro y giroscopio del MPU6050 para ajustar la precisión de sus lecturas al uso que se le vaya a dar.
    Simple_MPU6050.load_DMP_Image(): Este método carga una imagen (firmware) en el sensor MPU6050, utilizando el DMP (Digital
    Motion Processor), el cual es un procesador dentro del sensor que puede realizar cálculos y filtrar datos de forma autónoma,
    lo que simplifica el procesamiento en el microcontrolador principal y corrige el error acumulativo en la lectura del Yaw que
    se tenía al utilizar la librería Wire. Cargar una imagen DMP en el sensor permite aprovechar estas capacidades.
    OFFSETS es una directiva declarada en este programa que permite calibrar el sensor, */
    mpu.SetAddress(MPU6050_ADDRESS_AD0_LOW).load_DMP_Image(OFFSETS);
  #else
    Serial.println(F("No se establecieron Offsets, haremos unos nuevos.\n"  // muestra texto estático
                   " Colocar el sensor en un superficie plana y esperar unos segundos\n"
                   " Colocar los nuevos Offsets en #define OFFSETS\n"
                   " para saltar la calibracion inicial \n"
                   " \t\tPresionar cualquier tecla y ENTER"));
    while(Serial.available() && Serial.read());
    while(!Serial.available());
    while(Serial.available() && Serial.read());
    /*Simple_MPU6050.SetAddress().CalibrateMPU().load_DMP_Image(): Este método combinado permite indicar la dirección SLAVE del
    MPU, calibrar su rango de medición dependiendo de su aplicación y cargar una imagen al DMP (Digital Motion Processor) que
    corrija el error acumulativo en la lectura del Yaw que se tenía al usar la librería Wire.*/
    mpu.SetAddress(MPU6050_ADDRESS_AD0_LOW).CalibrateMPU().load_DMP_Image();
  #endif
  /*Simple_MPU6050.on_FIFO(): Este método se utiliza para habilitar la lectura continua de datos desde el FIFO del sensor
  MPU6050. FIFO es una pequeña memoria introducida dentro del sensor, cuyas siglas significan First Input First Output y cuando
  está llena de información es porque el DMP (Digital Motion Processor) ha terminado de realizar el cálculo de los datos 0de
  orientación: Roll, Pitch y Yaw, por lo que están listos para mostrarse en consola, por eso recibe como parámetro la función
  mostrar_valores().*/
  mpu.on_FIFO(mostrar_valores);
}

//EJECUCIÓN DEL PROGRAMA EN UN BUCLE INFINITO: Interrupciones que identifican el sentido de giro y orientación del sensor
MPU6050.
void loop() {
  /*Simple_MPU6050.dmp_read_fifo():  Este método se utiliza para realizar la lectura de datos que se encuentran dentro de la
  memoria FIFO en el sensor MPU6050.*/
  mpu.dmp_read_fifo();
}

//FUNCIÓN PROPIA DECLARADA EN VARIAS LINEAS:
/*mostrar_valores(): Esta función propia es llamada cada que se ejecute la interrupción conectada al Pin digital 2.*/
void mostrar_valores(int16_t*gyro, int16_t*accel, int32_t*quat, int32_t*timestamp){
  uint8_t SpamDelay = 100;          //Delay aplicado en ms cada vez que se muestran los valores de orientación en consola.
  /*Quaternion: Es un tipo de dato que representa una rotación en el espacio tridimensional, el cual almacena la orientación
  calculada por el sensor.*/
  Quaternion q;
  /*VectorFloat: Es un tipo de dato que representa un vector de tres dimensiones con valores decimales de punto flotante que se
  utilizan para almacenar la información de la gravedad calculada por el sensor.*/
  VectorFloat gravity;
  float ypr[3] = {0, 0, 0};               //Vector que almacena los resultados de la orientación: Yaw, Pitch y Roll (ypr).
  float xyz[3] = {0, 0, 0};               //Vector que considera los 3 ejes coordenados tridimensionales XYZ.
  /*spamtimer(SpamDelay): Función propia que se ejecuta cada 100 milisegundos (indicado por la variable SpamDelay), la cual
  realiza los cálculos del ángulo de Euler 3D para obtener los resultados de orientación: Yaw, Pitch y Roll en unidad de
  grados.*/
  spamtimer(SpamDelay){
    mpu.GetQuaternion(&q, quat);          //Representa la orientación del sensor en términos de una rotación tridimensional.
    mpu.GetGravity(&gravity, &q);         //Cálculo de la gravedad en términos de un vector tridimensional.
    mpu.GetYawPitchRoll(ypr, &q, &gravity);//Cálculo para la obtención de los resultados de orientación: Yaw, Pitch y Roll (ypr).
    mpu.ConvertToDegrees(ypr, xyz);    //Conversión de radianes a grados, ya que el resultado del cálculo anterior se da en rad.
    /*Serial.printfloatx(): Método que permite utilizar el formato de la función propia printfloatx() para imprimir en consola
    los datos recabados del sensor MPU6050 de forma personalizada, recibiendo como parámetros: El texto que muestra antes del
    valor de la variable, el valor de la variable, número de caracteres que muestran el valor de la variable, número de decimales
    mostrados y el texto que aparece al final.*/
    Serial.printfloatx(F("Roll: "),   xyz[2], 9, 4, F(",\t"));  //Giro alrededor del eje X = Roll.
    Serial.printfloatx(F("Pitch: "),  xyz[1], 9, 4, F(",\t"));  //Giro alrededor del eje Y = Pitch.
    Serial.printfloatx(F("Yaw: "),    xyz[0], 9, 4, F(",\t"));  //Giro alrededor del eje Z = Yaw.
    Serial.println();
    //OPERACIONES CONDICIONALES UTILIZANDO LOS VALORES DE ORIENTACIÓN OBTENIDOS POR EL SENSOR:
    if(xyz[2] >= 90){
      Serial.println("Roll = 90°;   Rotación alrededor del eje X.");
    }else if(xyz[1] >= 90){
      Serial.println("Pitch = 90°;  Rotación alrededor del eje Y.");
    }else if(xyz[0] >= 90){
      Serial.println("Yaw   = 90°;  Rotación alrededor del eje Z.");
    }
  }
}
```

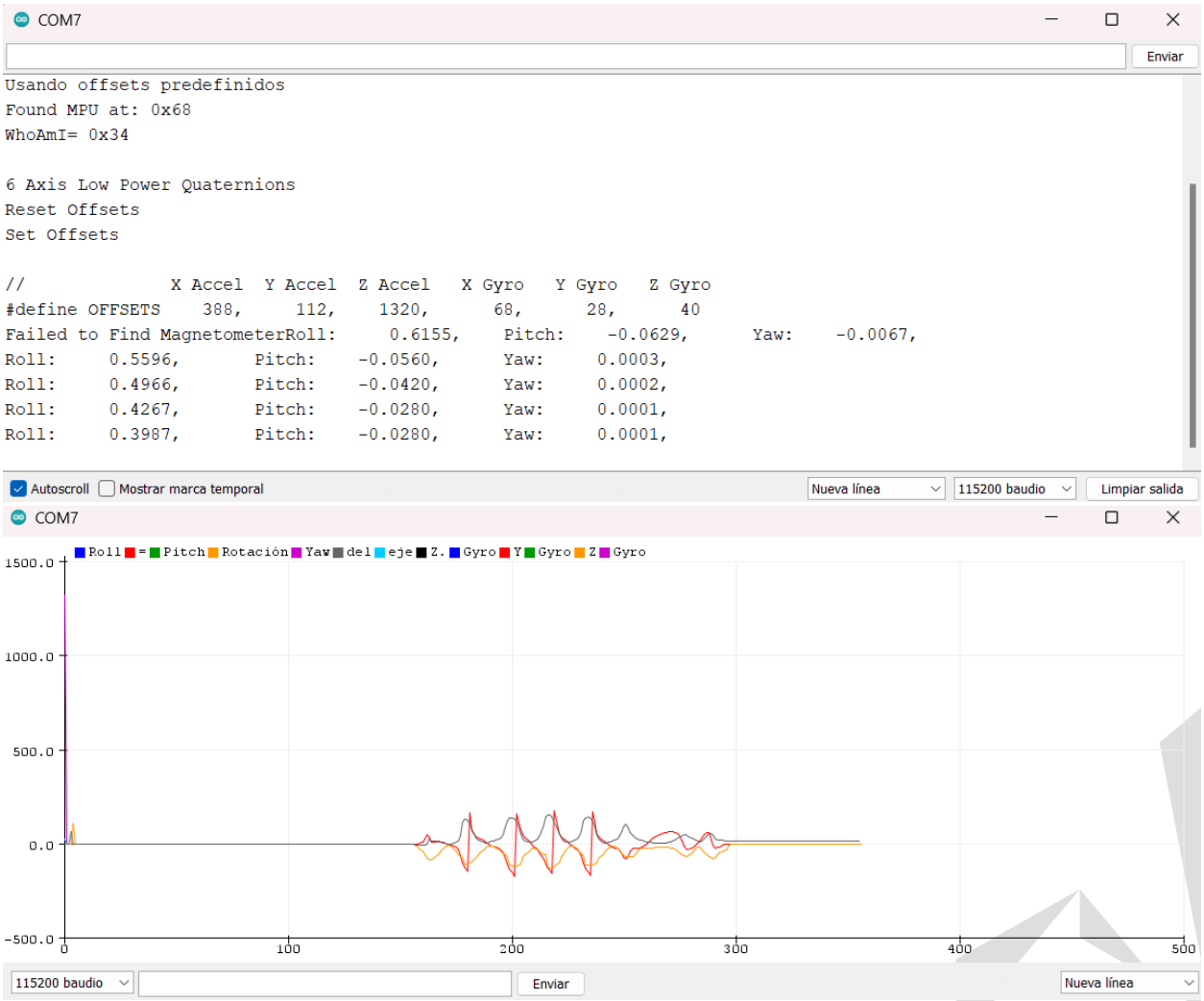
Resultados de orientación: Roll, Pitch y Yaw

Para observar los resultados de las señales Roll, Pitch y Yaw se puede utilizar el monitor serie para ver solo números o el Serial Plotter, para ver el resultado en forma de gráficas.


COM7
Enviar
Inicio:
No se establecieron Oion inicial
Presionar cualquier tecla y ENTER
undos
Colocar los nuevos Offsets en #define OFFSETS
para saltar la calibracion inicial
Presionar cualquier tecla y ENTER
Found MPU6050 or MPU9150
Found MPU at: 0x68
WhoAmI= ÓŸÍnicio:
No se establecieron Offsets, haremos unos nuevos.
Colocar el sensor en un superficie plana y esperar unos segundos
Colocar los nuevos Offsets en #define OFFSETS
para saltar la calibracion inicial
Presionar cualquier tecla y ENTER
Autoscroll
Mostrar marca temporal
Nueva línea
115200 baudio
Limpiar salida


COM7
Enviar
Reset Offsets
Set Offsets
Calibrate Gyro >.....
Found MPU at: 0x68
WhoAmI= 0x34
6 Axis Low Power Quaternions
Reset Offsets
Set Offsets
// X Accel Y Accel Z Accel X Gyro Y Gyro Z Gyro
#define OFFSETS 388, 112, 1320, 68, 28, 40
Failed to Find MagnetometerRoll: 0.6155, Pitch: -0.0629, Yaw: -0.0067,
Roll: 0.5456, Pitch: -0.0490, Yaw: 0.0002,
Roll: 0.4756, Pitch: -0.0280, Yaw: 0.0001,
Roll: 0.4267, Pitch: -0.0210, Yaw: 0.0001,
Autoscroll
Mostrar marca temporal
Nueva línea
115200 baudio
Limpiar salida




COM7
Enviar
Usando offsets predefinidos
Found MPU at: 0x68
WhoAmI= 0x34
6 Axis Low Power Quaternions
Reset Offsets
Set Offsets
// X Accel Y Accel Z Accel X Gyro Y Gyro Z Gyro
#define OFFSETS 388, 112, 1320, 68, 28, 40
Failed to Find MagnetometerRoll: 0.6155, Pitch: -0.0629, Yaw: -0.0067,
Roll: 0.5596, Pitch: -0.0560, Yaw: 0.0003,
Roll: 0.4966, Pitch: -0.0420, Yaw: 0.0002,
Roll: 0.4267, Pitch: -0.0280, Yaw: 0.0001,
Roll: 0.3987, Pitch: -0.0280, Yaw: 0.0001,
Autoscroll
Mostrar marca temporal
Nueva línea
115200 baudio
Limpiar salida
COM7
Roll = Pitch Rotación Yaw del eje Z. Gyro Y Gyro Z Gyro
1500.0
1000.0
500.0
0.0
-500.0
0
100
200
300
400
500
115200 baudio
Enviar
Nueva línea

Referencias

Bitwise Ar, “Arduino desde cero en Español - Capítulo 65 - MPU6050 acelerómetro y giróscopo Teoría y Práctica”, 2021 [Online], Available: https://www.youtube.com/watch?v=TwFZ4BJUX5c

How To Mechatronics, “How MEMS Accelerometer Gyroscope Magnetometer Work & Arduino Tutorial”, 2015 [Online], Available: https://www.youtube.com/watch?v=eqZgxR6eRjo

How To Mechatronics, “DIY Gimbal | Arduino and MPU6050 Tutorial”, 2015 [Online], Available: https://www.youtube.com/watch?v=UxABxSADZ6U

VirtualBrain, “Cómo Funciona un Giroscopio ⚡ Qué es un Giroscopio”, 2021 [Online], Available: https://www.youtube.com/watch?v=0cH2JCT8xbU&t=510s

ELECTRONOOBS en Español, “PROTOCOLOS: UART - I2C - SPI - Comunicación Serie #001”, 2020 [Online], Available: https://www.youtube.com/watch?v=G7aQB6x0LHc&t=150s

Bitwise Ar, “Arduino desde cero en Español - Capítulo 35 - LCD I2C adaptador e instalación de librería específica”, 2018 [Online], Available: https://www.youtube.com/watch?v=kuLgPLrg-cY&t=798s