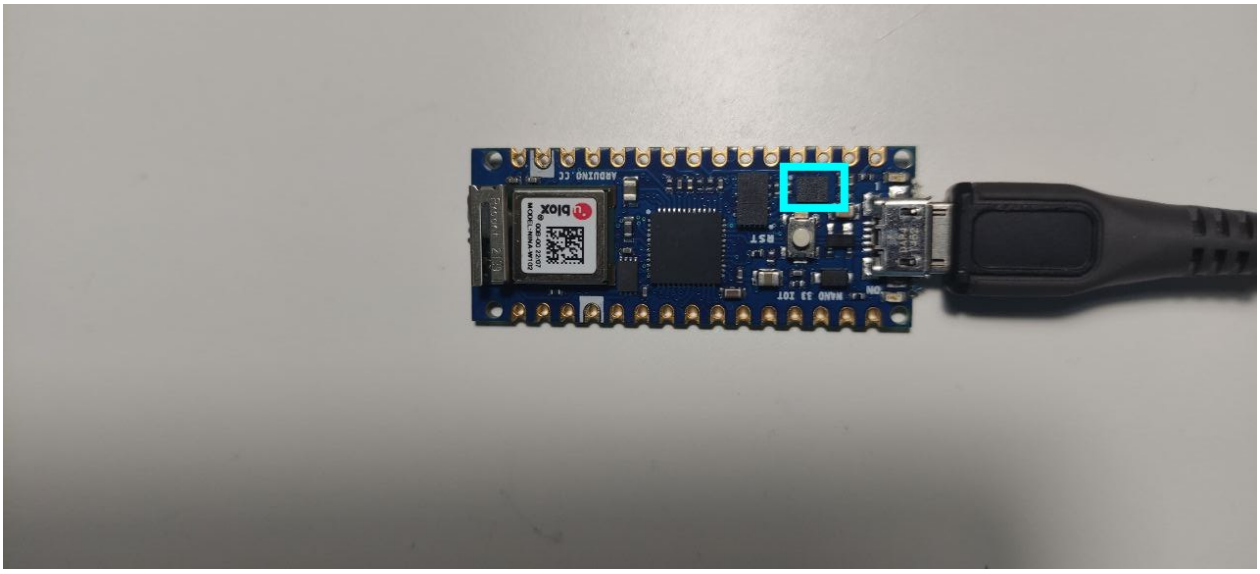
Práctica 2.2: Detectamos el movimiento y la posición de nuestro Arduino

Nuestro Nano 33 IoT fue el Arduino elegido porque, aparte de tener un tamaño bastante reducido y permitirnos realizar proyectos que incluyesen el internet de las cosas (IoT), lleva integrado un módulo que nos permite medir la posición relativa de nuestro microcontrolador. Este tipo de módulos reciben el nombre de IMU o *Inertial Measurement Unit.* En el caso de nuestro microcontrolador, el IMU que contiene es el LSM6DS3. Lo encontramos aquí:

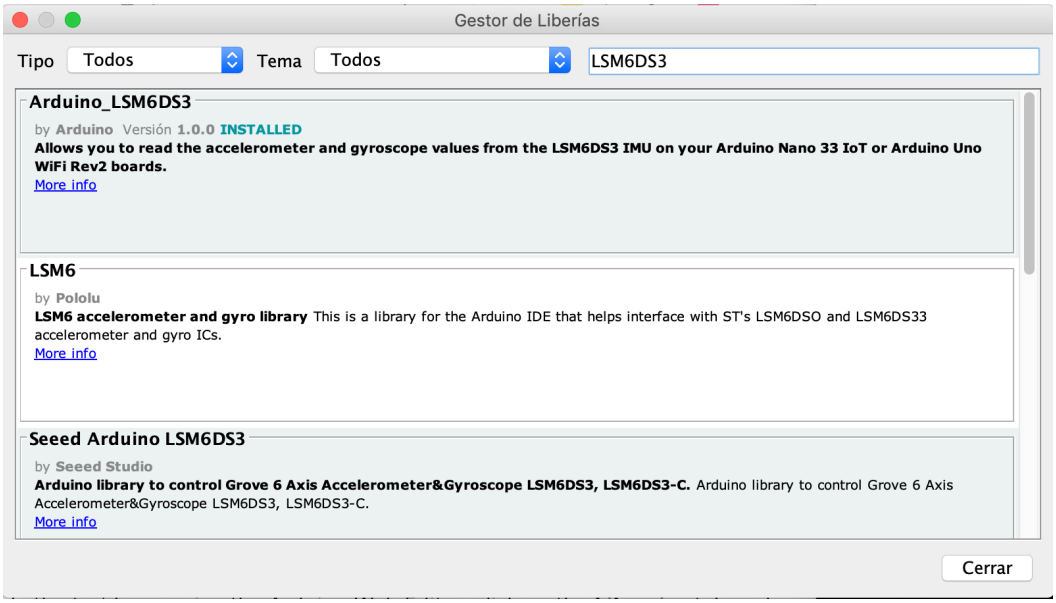


Este sensor hace uso de **acelerómetros** y **giroscopios** para calcular esa posición relativa. Un acelerómetro se encarga de medir los cambios en la velocidad de un objeto mientras que un giroscopio mide los cambios en la rotación. Ambos permiten detectar el movimiento rotacional en 3 ejes y que en nuestro ejemplo lo traduciremos a los cambios producidos en dos direcciones: de izquierda a derecha y de arriba a abajo.

La librería del LSM6DS3

Como ya comentábamos en la página anterior, las librerías nos van a facilitar las cosas a la hora de trabajar con ciertos sensores, y en esta práctica vamos a hacer uso de una de ellas. Para instalarla vamos a seguir los siguientes pasos:

1. En la parte superior de la IDE de Arduino iremos a **Programa > Incluir Librería > Administrar Bibliotecas**
2. Se nos abrirá una **nueva ventana** en la que nos aparecerán todas las librerías disponibles ordenadas alfabéticamente
3. En la parte superior derecha de esta ventana, escribiremos **LSM6DS3**



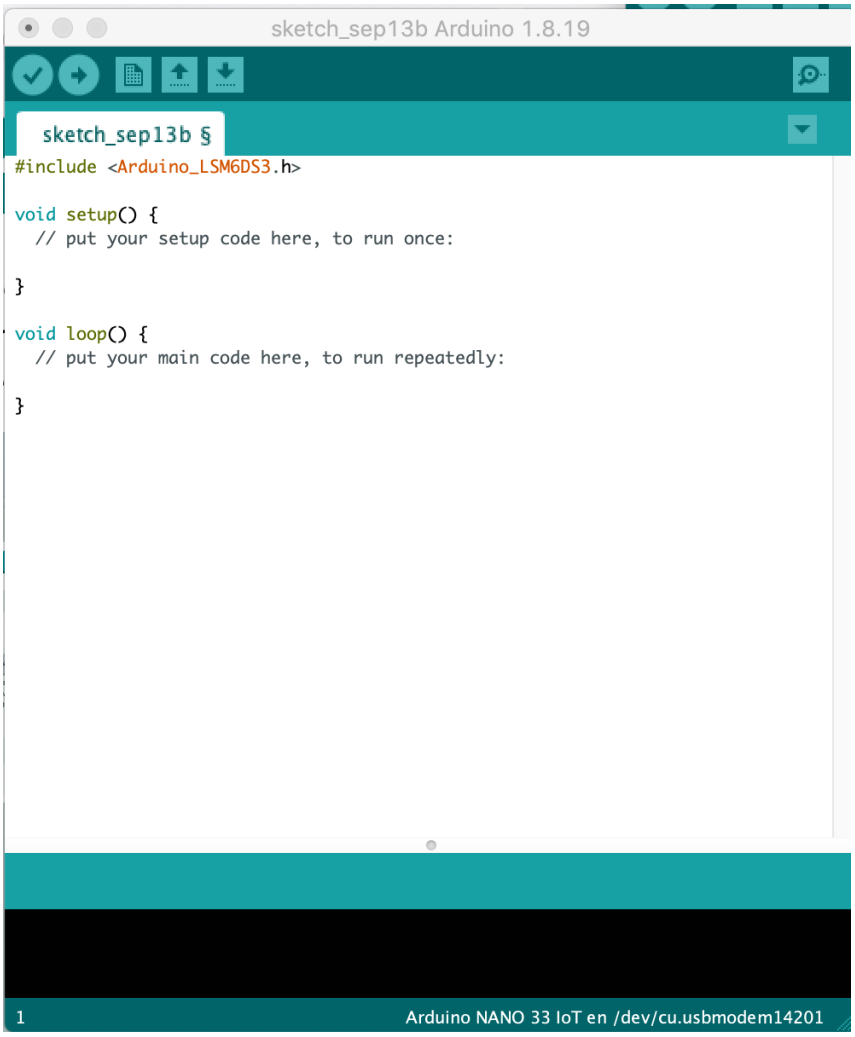
4. La primera que nos aparecerá será Arduino_LSM6DS3 y nos dará la opción de instalarla. En la imagen superior me aparece como ya instalada, pero si es la primera vez que realizáis este proceso os la permitirá instalar.

Una vez seguidos estos pasos, la librería ya está incluida en la IDE de Arduino, lista para ser incluida en los proyectos que la necesitemos.

Y... ¿cómo la incluimos?

Aunque para esta práctica, vamos a utilizar un sketch en el que ya aparece incluida, vamos a abrir un sketch en blanco para ver el proceso.

1. Abrimos un nuevo sketch de Arduino en blanco en **Archivo > Nuevo**.
2. Vamos a **Programa > Incluir Librería > Arduino LSMD6DS3.**
3. Veremos que en la primera línea de nuestro sketch aparece **#include <Arduino_LSM6DS3.h>** . Esa sentencia le indica a nuestro programa que tiene que hacer uso, **#include**, de la librería cuyo nombre se encuentra entre los símbolos **< >**.



Este archivo en blanco no lo guardaremos, solamente lo hemos usado para ver cómo se incluye una librería. A continuación veremos el código que emplearemos en esta práctica y qué hace cada una de sus líneas.

Cómo calcular la posición relativa

El código que vamos a utilizar en esta práctica se corresponde a uno de los ejemplos que acompañan a la librería de nuestro IMU, con ligeras modificaciones. Primero lo explicamos y luego ya podrás comprobarlo tú.

```
/*
  Arduino LSM6DS3 - Accelerometer Application
  This example reads the acceleration values as relative direction and degrees,
  from the LSM6DS3 sensor and prints them to the Serial Monitor or Serial Plotter.


  The circuit:
- Arduino Nano 33 IoT
```

```
  Created by Riccardo Rizzo

  Modified by Jose García

  27 Nov 2020

  This example code is in the public domain.
*/
```

Esta primera parte es un comentario. Como ya hemos visto al principio de todos los programas, se utiliza para indicar la finalidad del algoritmo, el autor y aquellas especificaciones que el autor crea convenientes. Como vemos en la última línea, el código se encuentra en el dominio público, lo que significa que podemos utilizarlo y modificarlo libremente en nuestros proyectos.

La siguiente línea que nos encontramos es la que ya hemos visto y que se refiere a la inclusión de la librería en nuestro sketch:

```
#include <Arduino_LSM6DS3.h>
```

Posteriormente, antes de comenzar a escribir nuestros bucles void setup() y void loop() encontramos las variables que utilizaremos en nuestro programa. La primera línea nos dice que vamos a tener 3 variables del tipo **float** (decimal) que van a ser **x, y, z**. A esas tres variables no les hemos asignado un valor inicial. En cambio, a las dos siguiente, que son del tipo **int** (entero), sí que les hemos asignado el valor 0. Es decir, nuestro IMU se va a considerar que comienza su movimiento en 0 grados respecto al eje X e Y:

```
float x, y, z;
int degreesX = 0;
int degreesY = 0;
```

Lo próximo que nos encontramos es la ya conocida llamada a **void setup()** En ella inicializaremos aquellos procesos que solamente han de ejecutarse una vez.

El primero será el puerto serie, sobre el que ya hablamos en este apartado anterior.
Iniciaremos la comunicación en serie con **Serial.begin(9600)**.

Seguido a ello, tenemos un **bucle while,** lo que nos dice este bucle es: "mientras que el Serial no reciba información, entra en bucle y no hagas nada, solo espera a que se establezca conexión a través del puerto serie". El signo de exclamación **!** significa **NOT**, es la negación para nuestro Arduino. Por lo tanto, **while (!serial)** es lo mismo que **while(serial NOT true)** es lo mismo que **while(serial==false).**

Con nuestro NANO, no deberíamos tener problema, ni entrar en ese bucle while y deberíamos rápidamente ver en nuestro monitor serie la palabra "**Started**".

```
void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Started");
```

A continuación encontramos un condicional en el que se indica qué hacer en caso que nuestro IMU no esté enviando información. Al igual que con el puerto serie, comprobamos si nuestro IMU ha iniciado su actividad. Si no ha sido así, imprimiremos por pantalla que ha fallado.

Si todo ha salido bien y nuestro Arduino se comunica perfectamente con nuestro ordenador a través del puerto serie, nuestro programa imprimirá por el puerto serie la velocidad de muestreo de nuestro IMU:

```
if (!IMU.begin()) {
Serial.println("Failed to initialize IMU!");
  }
  Serial.print("Accelerometer sample rate = ");
  Serial.print(IMU.accelerationSampleRate());
  Serial.println("Hz");
```

Ahora que ya hemos establecido la comunicación correctamente, ha llegado el momento de averiguar la posición relativa. Esto lo vamos a lograr haciendo uso de unas cuantas sentencias condicionales **if** dentro del bucle **void loop()**.

La primera de ellas va a depender de si existe aceleración, es decir si ha habido un cambio de posición a algún punto. Eso lo conseguimos con la función **accelerationAvailable(),** que acompaña a nuestra palabra reservada IMU. Si dicha aceleración existe, leeremos esa aceleración en los tres ejes, x, y, z:

```
void loop() {
  if (IMU.accelerationAvailable()) {
    IMU.readAcceleration(x, y, z);
  }
```

A continuación, lo que nos queda por hacer es realizar los cálculos y transformaciones para obtener los grados que se está moviendo nuestro IMU. Existen cuatro bloques if que se encargan de ello, dos para el eje X y dos para el eje Y. Nos limitaremos a comentar los dos referentes al bloque X, porque los del bloque Y son similares.

En el primer bloque **if** nos encontramos con la condición de que exista una variación en X mayor que 0.1, que sería lo mínimo perceptible para nuestro sensor. Si esto ocurre, multiplicaremos ese valor de x por 100 y realizaremos un mapeo de estos valores.

¿Cómo? Empleando la función **map()**, la cual se encarga de mapear un valor de un rango a otro. La estructura de esta función sería:

map(valor, valorInferiorInicial, valorSuperiorInicial, valorInferiorFinal, valorSuperiorFinal)

Por lo que estaríamos mapeando el valor de x desde el valor inicial 0 al 97, que son los valores que nos da el IMU, hasta los valores en grados 0 y 90; es decir, que la inclinación que vamos a conseguir abarcará desde los 0 a los 90 grados.

Lo mismo haremos si la x varía de forma negativa -0.1, eso significará que lo inclinamos hacia el lado contrario. En este caso el mapeo irá de los valores 0 y -100 hasta el 0 y 90 grados.

Visto cómo funciona en el eje x, no se considera necesario explicar lo que sucede con eje Y, ya que es exactamente igual:

```
if (x > 0.1) {
    x = 100 * x;
    degreesX = map(x, 0, 97, 0, 90);
    Serial.print("Tilting up ");
    Serial.print(degreesX);
    Serial.println("  degrees");
  }

  if (x < -0.1) {
    x = 100 * x;
    degreesX = map(x, 0, -100, 0, 90);
    Serial.print("Tilting down ");
    Serial.print(degreesX);
    Serial.println("  degrees");
  }

  if (y > 0.1) {
    y = 100 * y;
    degreesY = map(y, 0, 97, 0, 90);
    Serial.print("Tilting left ");
    Serial.print(degreesY);
    Serial.println("  degrees");
  }
```

```
  if (y < -0.1) {
    y = 100 * y;
    degreesY = map(y, 0, -100, 0, 90);
    Serial.print("Tilting right ");
    Serial.print(degreesY);
    Serial.println("  degrees");
  }
```

La última sentencia que nos encontramos es:

```
delay(1000);
```

En ella le decimos a nuestro programa que se detenga durante un segundo, con la función delay() de la que ya habíamos hablado.

El código completo lo puedes copiar de aquí y pegarlo en un nuevo sketch de Arduino para poder probarlo:

```
/*
  Arduino LSM6DS3 - Accelerometer Application
  This example reads the acceleration values as relative direction and degrees,
  from the LSM6DS3 sensor and prints them to the Serial Monitor or Serial Plotter.

  The circuit:
  - Arduino Nano 33 IoT

  Created by Riccardo Rizzo
  Modified by Jose García

  27 Nov 2020
  This example code is in the public domain.
*/



#include <Arduino_LSM6DS3.h>



float x, y, z;
int degreesX = 0;
int degreesY = 0;
```

```
void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Started");


  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
  }


  Serial.print("Accelerometer sample rate = ");
  Serial.print(IMU.accelerationSampleRate());
  Serial.println("Hz");

}


void loop() {

  if (IMU.accelerationAvailable()) {
    IMU.readAcceleration(x, y, z);
  }


  if (x > 0.1) {
    x = 100 * x;
    degreesX = map(x, 0, 97, 0, 90);
    Serial.print("Tilting up ");
    Serial.print(degreesX);
    Serial.println("  degrees");
  }

  if (x < -0.1) {
    x = 100 * x;
    degreesX = map(x, 0, -100, 0, 90);
    Serial.print("Tilting down ");
    Serial.print(degreesX);
```

```
    Serial.println("  degrees");
  }

  if (y > 0.1) {
    y = 100 * y;
    degreesY = map(y, 0, 97, 0, 90);
    Serial.print("Tilting left ");
    Serial.print(degreesY);
    Serial.println("  degrees");
  }

  if (y < -0.1) {
    y = 100 * y;
    degreesY = map(y, 0, -100, 0, 90);
    Serial.print("Tilting right ");
    Serial.print(degreesY);
    Serial.println("  degrees");
  }

  delay(1000);

}
```

Para obtener los valores correctos es necesario sujetar el Arduino en esta posición y posteriormente inclinarlo de manera lateral y frontal, veremos cómo los valores cambian en la ventana de nuestro puerto serie: