MÓDULO 7: ARDUINO

Realizaremos tres prácticas con el pulsómetro de nuestro kit

- Práctica 9.1 Medimos nuestras pulsaciones
- Práctica 9.2: Calculamos nuestras pulsaciones por minuto
- Práctica 9.3: Visualizamos cada pulsación en el serial plotter

Práctica 9.1 Medimos nuestras pulsaciones

El pulsómetro

Uno de los sensores que vienen con nuestro kit es este:

image-1668191315903.jpg

Este sensor se va a encargar de detectar nuestras pulsaciones. Aunque no vamos a entrar a explicar su funcionamiento.

> Si quieres saber más, puedes hacer click en este enlace.

Para detectar nuestras pulsaciones, tendremos que colocar nuestro dedo pulgar sobre la imagen del corazón blanco, sin presionar muy fuerte. Conectarlo a nuestro Arduino no puede ser más sencillo. Si miras en el lado contrario en el que está dibujado el corazón, te encontrarás esto:

image-1668191953508.jpg

Verás que, probablemente, el color de los cables de tu pulsómetro es distinto al mío, pero eso no importa. Te tienes que fijar en las letras que hay escritas al lado de cada cable.

La letra **S** es la que e encarga de transportar la señal desde nuestro sensor hasta Arduino y para esta práctica irá conectada a nuestro UNO en el pin **A0**.

El símbolo **+** es el que se encarga de alimentar a nuestro Arduino e irá en el pin **5V**.

Por último, el signo **-** es el que va a masa (ground) y en nuestro UNO irá conectado a uno de los dos pines **GND**.

Las conexiones nos quedarán como las de esta imagen:

image-1668192647513.jpg

Una vez hemos conectado nuestro sensor a Arduino, pasaremos a ver qué tenemos que hacer en la IDE de Arduino para hacerlo funcionar.

La librería Pulse Sensor Playground

Como ya sabes, una librería (también te la puedes encontrar denominada como biblioteca) de Arduino es una colección de código y ejemplos sobre un tema o dispositivo específico. Por ejemplo, para controlar el sensor que vamos a utilizar en esta práctica, un **pulsómetro**, vamos a emplear la biblioteca/librería **PulseSensor Playground.** En ella, vamos a encontrar una **colección de código y proyectos** creados exclusivamente para facilitar el uso de este sensor junto con Arduino.

Lo primero que haremos será instalar esta librería. Para ello, iremos a **Programa > Incluir Librería > Administrar Bibliotecas...** :

image-1665231393395.14.58.png

Una vez ahí, haremos click y nos aparecerá una ventana emergente en la que podremos buscar la librería, que en este caso se llama **PulseSensor Playground**, cuando aparezca, pondremos el ratón sobre ella y nos aparecerá el botón **Instalar**. La instalaremos:

image-1665231801613.15.50.png

Cerraremos la ventana emergente y veremos que si vamos a **Archivo > Ejemplos** nos aparecerán los ejemplos de la librería **PulseSensor Playground**:

image-1665232108397.27.14 copia.png

Una vez la hayamos instalado, abriremos el ejemplo **GettingStartedProject**. Y pasaré a explicarte lo que hace cada línea de código.

¡Pulsación detectada ?!

Lo primero que voy a hacer es poner aquí el código, aunque imagino que ya lo tienes abierto en tu IDE de Arduino:

```
/*  PulseSensor Starter Project and Signal Tester
 *  The Best Way to Get Started  With, or See the Raw Signal of, your PulseSensor.com™ &
Arduino.
```

```
 *
 *  Here is a link to the tutorial
 *  https://pulsesensor.com/pages/code-and-guide
 *
 *  WATCH ME (Tutorial Video):
 *  https://www.youtube.com/watch?v=RbB8NSRa5X4
 *
 *
------------------------------------------------------------
1) This shows a live human Heartbeat Pulse.
2) Live visualization in Arduino's Cool "Serial Plotter".
3) Blink an LED on each Heartbeat.
4) This is the direct Pulse Sensor's Signal.
5) A great first-step in troubleshooting your circuit and connections.
6) "Human-readable" code that is newbie friendly."

*/



//  Variables
int PulseSensorPurplePin = 0;        // Pulse Sensor PURPLE WIRE connected to ANALOG PIN 0
int LED13 = 13;   //  The on-board Arduion LED


int Signal;                // holds the incoming raw data. Signal value can range from 0-1024
int Threshold = 550;            // Determine which Signal to "count as a beat", and which to
ingore.


// The SetUp Function:
void setup() {
  pinMode(LED13,OUTPUT);         // pin that will blink to your heartbeat!
   Serial.begin(9600);         // Set's up Serial Communication at certain speed.

}
```

```
// The Main Loop Function
void loop() {

  Signal = analogRead(PulseSensorPurplePin);  // Read the PulseSensor's value.
                                              // Assign this value to the "Signal" variable.

   Serial.println(Signal);                    // Send the Signal value to Serial Plotter.


   if(Signal > Threshold){                          // If the signal is above "550", then
"turn-on" Arduino's on-Board LED.
     digitalWrite(LED13,HIGH);
   } else {
     digitalWrite(LED13,LOW);                //  Else, the sigal must be below "550", so
"turn-off" this LED.
   }


delay(10);


}
```

Ahora, vamos a ir desmenuzando qué es lo que hace el código:

Todo el primer bloque, recogido entre los símbolos **/* */** es un comentario. No volveremos a explicar su función porque ya lo vimos en este apartado.

Pasaremos directamente a los párrafos que contienen las variables:

```
//  Variables
int PulseSensorPurplePin = 0;        // Pulse Sensor PURPLE WIRE connected to ANALOG PIN 0
int LED13 = 13;   //  The on-board Arduion LED


int Signal;                // holds the incoming raw data. Signal value can range from 0-1024
```

```
int Threshold = 550;            // Determine which Signal to "count as a beat", and which to
ingore.
```

Para hacer funcionar nuestro proyecto, necesitaremos crear 4 variables de números enteros (int). La primera de ellas **PulseSensorPurplePin** tiene el valor **0**, ya que va conectado el pin analógico **A0**. Aparece el color **purple**, porque en el sensor para el que en origen se creó este ejemplo, el cable que se conecta a A0 era morado.
La variable **LED13** tiene el valor 13, ya que es el pin al que va conectado el pin que va integrado en la placa de nuestro UNO.
Signal almacenará el valor que nuestro sensor produce cuando está intentando detectar una pulsación. Como se trata de un sensor analógico, la señal se corresponderá a valores discretos, dentro del rango **0 hasta 1024**.
Por último, **Threshold** será el umbral a partir del cual podremos afirmar que se ha producido un latido, una pulsación, en este caso será **550**. Ya habíamos dicho que el umbral de nuestra señal iba de **0 a 1024**, por lo que utilizar 550 es una buena referencia.

A continuación, vamos a pasar a ver la función **setup():**

```
// The SetUp Function:
void setup() {
  pinMode(LED13,OUTPUT);         // pin that will blink to your heartbeat!
   Serial.begin(9600);         // Set's up Serial Communication at certain speed.


}
```

Es bastante breve y en ella lo único que necesitamos incluir es el modo del pin que vamos a utilizar gracias a la función pinmode(). En ella indicaremos el pin y si va a ser de **entrada** (**INPUT**) o de **salida** (**OUTPUT**). Como en este caso es un LED, le diremos que es de salida. También le diremos a nuestro UNO a que velocidad tiene que enviar la información a través del puerto serial, que en este caso será **9600 baudios.**

> Recuerda que todo lo incluido en la función setup() solamente se ejecutará una vez.

La última parte de nuestro sketch se corresponde con la función **loop()**.

```
// The Main Loop Function
void loop() {


  Signal = analogRead(PulseSensorPurplePin);  // Read the PulseSensor's value.
```

```
                                          // Assign this value to the "Signal" variable.
   Serial.println(Signal);                // Send the Signal value to Serial Plotter.


   if(Signal > Threshold){                      // If the signal is above "550", then
"turn-on" Arduino's on-Board LED.
     digitalWrite(LED13,HIGH);
   } else {
     digitalWrite(LED13,LOW);               //  Else, the sigal must be below "550", so
"turn-off" this LED.
   }

delay(10);


}
```

En ella, lo primero que vamos a hacer es almacenar en la variable Signal los valores que lea nuestro sensor. Para leer esos valores, usaremos la función **analogRead()**. Indicaremos que la señal analógica que ha de ser leída es la que recibamos con la variable **PulseSensor PurplePin**. A continuación, imprimiremos por nuestro puerto serial el valor obtenido, aunque para visualizar las pulsaciones no nos va a servir de mucho, porque los números aparecen a gran velocidad. Para visualizar las posiciones usaremos el LED. ¿Cómo? Pues usaremos un condicional: "Oye, Arduino, si **(if)** la señal que lees es mayor que el umbral (**Signal > Threshold**), enciende el LED **(digitalWrite(LED13,HIGH))**, si no **(else)**, apágalo (**digitalWrite(LED13,LOW)**)".

Para finalizar, usamos un **delay** de 10 milisegundos, para darle a nuestro Arduino un pequeño respiro entre lectura y lectura.

Ahora, **conectaremos nuestro UNO al ordenador** y subiremos el código.

En este GIF puedes ver cómo la luz naranja situada más a la derecha, comienza a parpadear cuando coloco mi dedo encima del pulsómetro:

image-1666605711519.gif

> **Cuando no está el dedo colocado**, la luz también parpadeará aleatoriamente. Eso no sucede porque el sensor esté roto o no funcione bien, sino porque **el sensor realizará lecturas erróneas**. Intentará detectar pulsaciones, aunque nuestro dedo no esté colocado

sobre él. Es importante colocar **el dedo sin apretar mucho y sin moverlo** y esperaremos uno 3 o 4 segundos hasta que el sensor sea capaz de leer correctamente nuestras pulsaciones.

¿Qué tenemos que presentar?

En este caso será suficiente con que me envíes un **.gif o vide**o en el que se vea el pulsómetro sin tu dedo y luego con tu dedo sobre él, iluminándose el LED naranja, similar al video que aparece justo encima de este apartado.

FUENTES

Librería de Pulse Sensor: https://pulsesensor.com/pages/installing-our-playground-for-pulsesensor-arduino
Sobre Pulse Sensor: https://pulsesensor.com/pages/code-and-guide

Práctica 9.2: Calculamos nuestras pulsaciones por minuto

Ahora, vamos a pasar a hacer algo más interesante. Está bien poder detectar nuestras pulsaciones, pero ¿por qué no contar el número de pulsaciones que tenemos por minuto o **BPM**?

Lo haremos con el siguiente código:

```
/*  Getting_BPM_to_Monitor prints the BPM to the Serial Monitor, using the least lines of code
and PulseSensor Library.
 *  Tutorial Webpage: https://pulsesensor.com/pages/getting-advanced
 *
--------Use This Sketch To-----------------------------------------
1) Displays user's live and changing BPM, Beats Per Minute, in Arduino's native Serial
Monitor.
2) Print: "♥  A HeartBeat Happened !" when a beat is detected, live.
2) Learn about using a PulseSensor Library "Object".
4) Blinks LED on PIN 13 with user's Heartbeat.
-----------------------------------------------------------------------*/

#define USE_ARDUINO_INTERRUPTS true    // Set-up low-level interrupts for most acurate BPM
math.
#include <PulseSensorPlayground.h>     // Includes the PulseSensorPlayground Library.

//  Variables
const int PulseWire = 0;       // PulseSensor PURPLE WIRE connected to ANALOG PIN 0
const int LED13 = 13;          // The on-board Arduino LED, close to PIN 13.
int Threshold = 550;           // Determine which Signal to "count as a beat" and which to
ignore.
                               // Use the "Gettting Started Project" to fine-tune Threshold
Value beyond default setting.
```

```
                                // Otherwise leave the default "550" value.

PulseSensorPlayground pulseSensor;  // Creates an instance of the PulseSensorPlayground object
called "pulseSensor"


void setup() {

  Serial.begin(9600);          // For Serial Monitor

  // Configure the PulseSensor object, by assigning our variables to it.
  pulseSensor.analogInput(PulseWire);
  pulseSensor.blinkOnPulse(LED13);       //auto-magically blink Arduino's LED with heartbeat.
  pulseSensor.setThreshold(Threshold);

  // Double-check the "pulseSensor" object was created and "began" seeing a signal.
   if (pulseSensor.begin()) {
    Serial.println("We created a pulseSensor Object !");  //This prints one time at Arduino
power-up,  or on Arduino reset.
  }
}



void loop() {

 int myBPM = pulseSensor.getBeatsPerMinute();  // Calls function on our pulseSensor object
that returns BPM as an "int".
                                               // "myBPM" hold this BPM value now.

if (pulseSensor.sawStartOfBeat()) {            // Constantly test to see if "a beat happened".

 Serial.println("♥  A HeartBeat Happened ! "); // If test is "true", print a message "a
heartbeat happened".
 Serial.print("BPM: ");                        // Print phrase "BPM: "
 Serial.println(myBPM);                        // Print the value inside of myBPM.
```

```
}

  delay(20);                    // considered best practice in a simple sketch.

}
```

Como puedes ver, no tiene muchas líneas, ¿por qué? Pues porque la mayor parte de procesos los realizan funciones que nos vienen preprogramadas en la librería PulseSensor Playground. Así, nosotras no tenemos que preocuparnos demasiado de esa parte y nos resultará más sencillo integrar nuestro pulsómetro con otros sensores o actuadores, haciendo que no sean necesarias tantas líneas de código.

Desmenuzando el código

Como ya he hecho en anteriores páginas, no voy a volver a explicarte la parte de los **comentarios** , aunque te invito a leerlos, porque de esta manera entenderás mejor el propósito del código y las partes de este.

En esta práctica, vamos a ver un concepto nuevo en el que no vamos a ahondar demasiado: las **interrupciones**.

```
#define USE_ARDUINO_INTERRUPTS true    // Set-up low-level interrupts for most acurate BPM
math.
#include <PulseSensorPlayground.h>     // Includes the PulseSensorPlayground Library.
```

Las interrupciones

Las interrupciones se encargan de indicarle a nuestro Arduino que cierto proceso ha de ser atendido con una prioridad mayor a cualquier otro. De esta manera se interrumpe cualquier actividad del Arduino para atender la información recibida a través de alguno de los pines.

En nuestro ejemplo concreto, para asegurar una correcta lectura de nuestras pulsaciones, usamos la orden **#define USE_ARDUINO_INTERRUPTS true.** Todo el manejo de las interrupciones permanece oculto, pero es necesario para la correcta lectura de nuestras pulsaciones.

> Si quieres saber más sobre interrupciones, puedes echarle un vistazo a este enlace.

Las librerías

La siguiente línea que encontramos importa la **librería PulseSensorPlayground.h.** A Arduino necesitaremos indicárselo usando la palabra reservada **#include** seguida de la librería que queramos incluir, entre **< >**. Las librerías, son archivos con la terminación **.h**. Estos documentos pueden ser abiertos en un editor de texto y podemos ver su contenido y modificarlos, aunque esto último no te lo aconsejo... ya que si no estamos seguros de qué estamos modificando y borramos o editamos algo que no deberíamos, esta librería dejaría de funcionar y tendríamos que reinstalarla.

Las variables y constantes

Si avanzamos, encontramos las siguientes líneas, las cuales aparecen explicadas en los comentarios que las acompañan a la derecha:

```
const int PulseWire = 0;       // PulseSensor PURPLE WIRE connected to ANALOG PIN 0
const int LED13 = 13;          // The on-board Arduino LED, close to PIN 13.
int Threshold = 550;           // Determine which Signal to "count as a beat" and which to
ignore.
                               // Use the "Gettting Started Project" to fine-tune Threshold
Value beyond default setting.
                               // Otherwise leave the default "550" value.
```

En Arduino, habíamos visto que existen distintos tipos de datos para almacenar información. Por ejemplo, si estamos trabajando con números, podíamos usar, entre otros, números enteros o decimales, int o float respectivamente. A parte de elegir el tipo de datos, podemos indicarle a nuestro Arduino si el valor que vamos a almacenar va a variar o siempre va a ser el mismo. Si el valor puede ser modificado a lo largo del código, escribiremos las variables como hemos hecho hasta ahora, pero si no queremos que ese valor cambia, si queremos que sean constantes, tendremos que escribir delante la palabra **const**.

El objeto pulseSensor

Para hacer uso de las funciones preprogramadas en la librería que queremos utilizar, tenemos que crear "una instancia del objeto PulseSensorPlayground". Eso es lo que nos pone en el comentario que acompaña a la siguiente línea de código. Pero, ¿qué significa eso?

```
PulseSensorPlayground pulseSensor;  // Creates an instance of the PulseSensorPlayground object
called "pulseSensor"
```

Bueno, si te suena raro, voy a explicártelo con coches. Por un lado existe la idea abstracta de coche y, por otro, los coches reales que podemos conducir con unas características que los diferencian según la marca, el año, etc. Bueno, hasta aquí todo bien. Pues nuestra librería es como la idea de coche, también tiene una serie de características y para usarlas, tendremos que crear un objeto que funcione como un pulsómetro.

En el caso de los coches, una instancia de la idea de coche podría ser un ALFA ROMEO Giulietta 1.6 JTD 120CV. En el caso de nuestra librería, podemos crear una instancia de un pulsómetro con la línea de código: **PulseSensorPlayground pulseSensor;**

Ahí, lo que estamos diciendo es que a partir de la **idea de coche/librería** PulseSensorPlayground, vamos a obtener un **coche/pulsómetro** que se va a llamar pulseSensor. Una vez hayamos creado este objeto, podremos utilizar las funciones que en la librería han sido programadas.

void setup(): lo que se ejecuta una sola vez

Una vez tenemos todo preparado, será necesario comenzar la comunicación a través del puerto serie y configurar nuestro objeto pulseSensor. Para ello, usaremos funciones vinculadas a él. Estas funciones tendremos que escribirlas a continuación de nuestro objeto y separadas de este por un **'.'** Una de ellas es **pulseSensor.analogInput(PulseWire).** Ahí, lo que estamos diciendo es que para nuestro pulsómetro, el pin de entrada va a ser el **pin A0**. Si recuerdas, la constante **pulseWire** ya la hemos definido antes.

Las otras dos funciones que la siguen, lo que dicen es que el LED que parpadeará será el 13 (**pulseSensor.blinkOnPulse(LED13))** y que el umbral para detectar la pulsación sea el que hemos definido antes con la constante Threshold (**pulseSensor.setThreshold(Threshold))** .

```
void setup() {

  Serial.begin(9600);          // For Serial Monitor

  // Configure the PulseSensor object, by assigning our variables to it.
  pulseSensor.analogInput(PulseWire);
  pulseSensor.blinkOnPulse(LED13);       //auto-magically blink Arduino's LED with heartbeat.
  pulseSensor.setThreshold(Threshold);
```

```
  // Double-check the "pulseSensor" object was created and "began" seeing a signal.
   if (pulseSensor.begin()) {
    Serial.println("We created a pulseSensor Object !");  //This prints one time at Arduino
power-up,  or on Arduino reset.
  }
}
```

A continuación, encontramos un condicional en el que, si hemos comenzado la comunicación por el puerto serie, imprimiremos: **We created a pulseSensor object!**.

void loop(): lo que se ejecuta todo el rato

A continuación, vamos a guardar en una variable las pulsaciones por minuto que va a calcular la función **.getBeatsPerMinute()**. Esto lo va a calcular la función directamente, así que no tendremos que preocuparnos de cómo se hace.

En el siguiente condicional, lo que vamos a hacer es que cada vez que se detecte una pulsación, con la función **.sawStartOfBeat()**, se va a imprimir por el puerto serie un aviso de que ha ocurrido una pulsación e imprimirá las pulsaciones por minuto actuales.

Para finalizar, se añade un pequeño **delay** para que la lectura de pulsaciones se realice correctamente.

```
void loop() {

 int myBPM = pulseSensor.getBeatsPerMinute();  // Calls function on our pulseSensor object
that returns BPM as an "int".
                                               // "myBPM" hold this BPM value now.

if (pulseSensor.sawStartOfBeat()) {            // Constantly test to see if "a beat happened".

 Serial.println("♥  A HeartBeat Happened ! "); // If test is "true", print a message "a
heartbeat happened".
 Serial.print("BPM: ");                        // Print phrase "BPM: "
 Serial.println(myBPM);                        // Print the value inside of myBPM.
}
```

```
  delay(20);                    // considered best practice in a simple sketch.

}
```

Ahora que ya entendemos el código, conectamos nuestro Arduino al ordenador y subimos el código.

En este .gif podemos verlo en acción:

image-1666685682260.gif

¿Qué tenemos que presentar?

En este caso, al igual que con la práctica anterior, será suficiente con que me envíes un **.gif o video** en el que se vea el pulsómetro sin tu dedo, luego con tu dedo sobre él, y que posteriormente la cámara enfoque al monitor y con el puerto serial abierto, sea posible ver las pulsaciones. Similar al video que aparece justo encima de este apartado.

Práctica 9.3: Visualizamos cada pulsación en el serial plotter

Como ya hemos visto, al abrir el **puerto serial**, comenzamos a transmitir información entre nuestro Arduino y nuestro ordenador. Lo que vamos a ver ahora son las dos maneras en que podemos visualizar esa transmisión de información. Hasta ahora, hemos utilizado el monitor serial, pero en esta práctica vamos a emplear el plotter.

Habíamos visto que para abrir el puerto serie, había que hacer click sobre el **icono de lupa** en la esquina superior derecha. Para visualizarlo como plotter no será necesario hacer click en ningún otro sitio, pero cuando nuestro ratón este sobre el icono de la lupa, tendremos que pulsar la **tecla shift (flecha hacia arriba).** De esta manera, en lugar de aparecer las palabras Monitor Serie, cuando nuestro ratón esté sobre la lupa pondrá **Serial Plotter**.

image-1667292915456.47.29.png

¿Para qué necesitamos el Serial Plotter?

Es otra manera de visualizar la información. A través del monitor serie, podemos visualizarla en forma de números y caracteres; mientras que el plotter nos permite visualizar esa información gráficamente, como en el caso que nos ocupa de las pulsaciones. En el anterior ejemplo veíamos el **número** de pulsaciones por minuto y ahora lo que vamos a hacer es **visualizar** cada pulsación como una línea, al igual que en un **electrocardiograma**.

El código

Este es el código que pasaremos a comentar:

```
/*
   Code to detect pulses from the PulseSensor,
   using an interrupt service routine.

   Here is a link to the tutorial\
   https://pulsesensor.com/pages/getting-advanced

   Copyright World Famous Electronics LLC - see LICENSE
   Contributors:
     Joel Murphy, https://pulsesensor.com
     Yury Gitman, https://pulsesensor.com
     Bradford Needham, @bneedhamia, https://bluepapertech.com

   Licensed under the MIT License, a copy of which
   should have been included with this software.

   This software is not intended for medical use.
*/

/*
   Every Sketch that uses the PulseSensor Playground must
   define USE_ARDUINO_INTERRUPTS before including PulseSensorPlayground.h.
   Here, #define USE_ARDUINO_INTERRUPTS true tells the library to use
   interrupts to automatically read and process PulseSensor data.

   See ProcessEverySample.ino for an example of not using interrupts.
*/
#define USE_ARDUINO_INTERRUPTS true
#include <PulseSensorPlayground.h>

/*
   The format of our output.

   Set this to PROCESSING_VISUALIZER if you're going to run
    the Processing Visualizer Sketch.
    See https://github.com/WorldFamousElectronics/PulseSensor_Amped_Processing_Visualizer
```

```
   Set this to SERIAL_PLOTTER if you're going to run
    the Arduino IDE's Serial Plotter.
*/
const int OUTPUT_TYPE = SERIAL_PLOTTER;

/*
   Pinout:
     PULSE_INPUT = Analog Input. Connected to the pulse sensor
      purple (signal) wire.
     PULSE_BLINK = digital Output. Connected to an LED (and 220 ohm resistor)
      that will flash on each detected pulse.
     PULSE_FADE = digital Output. PWM pin onnected to an LED (and resistor)
      that will smoothly fade with each pulse.
      NOTE: PULSE_FADE must be a pin that supports PWM. Do not use
      pin 9 or 10, because those pins' PWM interferes with the sample timer.
*/
const int PULSE_INPUT = A0;
const int PULSE_BLINK = 13;    // Pin 13 is the on-board LED
const int PULSE_FADE = 5;
const int THRESHOLD = 550;   // Adjust this number to avoid noise when idle

/*
   All the PulseSensor Playground functions.
*/
PulseSensorPlayground pulseSensor;

void setup() {
  /*
     Use 115200 baud because that's what the Processing Sketch expects to read,
     and because that speed provides about 11 bytes per millisecond.

     If we used a slower baud rate, we'd likely write bytes faster than
     they can be transmitted, which would mess up the timing
     of readSensor() calls, which would make the pulse measurement
     not work properly.
```

```
  */
  Serial.begin(115200);

  // Configure the PulseSensor manager.

  pulseSensor.analogInput(PULSE_INPUT);
  pulseSensor.blinkOnPulse(PULSE_BLINK);
  pulseSensor.fadeOnPulse(PULSE_FADE);

  pulseSensor.setSerial(Serial);
  pulseSensor.setOutputType(OUTPUT_TYPE);
  pulseSensor.setThreshold(THRESHOLD);

  // Now that everything is ready, start reading the PulseSensor signal.
  if (!pulseSensor.begin()) {
    /*
       PulseSensor initialization failed,
       likely because our particular Arduino platform interrupts
       aren't supported yet.

       If your Sketch hangs here, try PulseSensor_BPM_Alternative.ino,
       which doesn't use interrupts.
    */
    for(;;) {
      // Flash the led to show things didn't work.
      digitalWrite(PULSE_BLINK, LOW);
      delay(50);
      digitalWrite(PULSE_BLINK, HIGH);
      delay(50);
    }
  }
}

void loop() {
  /*
     Wait a bit.
```

```
      We don't output every sample, because our baud rate
      won't support that much I/O.
   */
   delay(20);

   // write the latest sample to Serial.
  pulseSensor.outputSample();

   /*
      If a beat has happened since we last checked,
      write the per-beat information to Serial.
    */
   if (pulseSensor.sawStartOfBeat()) {
    pulseSensor.outputBeat();
   }
}
```

La primera parte, el comentario en el que se explica el propósito del código, no la comentaremos. Y en este caso, tampoco las dos líneas siguientes en las que incluimos el uso de interrupciones y de la librería PulseSensorPlayground.h, porque acabamos de explicarlo en la práctica anterior.

La siguiente línea de código que encontramos es una en la que definimos una **constante** llamada **OUTPUT_TYPE.** Ahí estamos obligando a nuestro Arduino a establecer su salida de información en el **SERIAL_PLOTTER**. Al ser una constante, ese valor no cambiará a lo largo del sketch. Es decir, que la información tendremos que visualizarla necesariamente a través del Serial Plotter:

```
const int OUTPUT_TYPE = SERIAL_PLOTTER;
```

Más constantes

Aparte del tipo de salida, vamos a definir cuatro constantes más:

```
const int PULSE_INPUT = A0;
const int PULSE_BLINK = 13;    // Pin 13 is the on-board LED
const int PULSE_FADE = 5;
```

```
const int THRESHOLD = 550;   // Adjust this number to avoid noise when idle
```

Como pone en el comentario que aparece en el sketch sobre estas líneas, lo primero que estamos definiendo es desde que pin estamos recibiendo la información de nuestro pulsómetro (A0).

La siguiente línea nos dice que con cada pulsación encenderemos el LED 13, que es el LED que nuestro Arduino lleva integrado.

En el caso de que quisiéramos que la intensidad de un LED variase de la misma manera que varía la intensidad en cada pulsación, deberíamos conectar un LED y una resistencia al pin digital 5. Pero no lo haremos en esta práctica.

En la última línea estamos diciéndole a nuestro Arduino que todo lo que no llegue a 550 no debe considerarlo como una pulsación, para evitar la mayor cantidad de ruido posible.

El objeto pulseSensor

Como ya hemos visto en la práctica anterior, para hacer uso de las funciones de la librería es necesario crear una instancia de nuestra librería, un pulsómetro:

```
PulseSensorPlayground pulseSensor;
```

void setup(): lo que se ejecuta una sola vez

A continuación, pasaremos a ver lo que solamente ha de ejecutarse un vez y que por tanto aparece dentro de la función setup().

Obviando los comentarios, la primera línea de código que nos encontramos es el comienzo de la comunicación a través del puerto serie, en este caso a una mayor velocidad que en prácticas anteriores, a 115200 baudios. Esto nos permitirá enviar y recibir información de una manera más rápida y hará que funcione bien nuestro pulsómetro, como se nos indica en el comentario:

```
void setup() {
  /*
     Use 115200 baud because that's what the Processing Sketch expects to read,
     and because that speed provides about 11 bytes per millisecond.

     If we used a slower baud rate, we'd likely write bytes faster than
     they can be transmitted, which would mess up the timing
```

```
    of readSensor() calls, which would make the pulse measurement
    not work properly.
*/
Serial.begin(115200);
```

> Después, cuando abras tu Serial Monitor, en la esquina inferior derecha de tu Serial monitor seguramente pondrá 9600 baudios, así que tendrás que abrir la pestaña y cambiarlo a 115200 baudios.

Lo siguiente que vamos a hacer es preparar nuestro objeto pulsómetro. Al igual que en la práctica anterior, tendremos que decirle cuál es el pin de entrada (**pulseSensor.analogInput(PULSE_INPUT)**), el pin que parpadeará (**pulseSensor.blinkOnPulse(PULSE_BLINK)**) y el pin que variará su intensidad (**pulseSensor.fadeOnPulse(PULSE_FADE)**), aunque nosotras no vamos a conectar nada en este tercero.

```
// Configure the PulseSensor manager.

pulseSensor.analogInput(PULSE_INPUT);
pulseSensor.blinkOnPulse(PULSE_BLINK);
pulseSensor.fadeOnPulse(PULSE_FADE);

pulseSensor.setSerial(Serial);
pulseSensor.setOutputType(OUTPUT_TYPE);
pulseSensor.setThreshold(THRESHOLD);
```

Los tres siguientes se encargan de decirle a nuestro Arduino que vamos a usar el **puerto serie**, que nuestra salida va a ser el **SERIAL_PLOTTER** (esa constante la hemos definido al principio de nuestro código) y la última línea configura como **umbral**, el que ya hemos definido en la constante **THRESHOLD**.

Lo siguiente que haremos es comenzar la comunicación con el puerto serie. Para ello, utilizaremos un condicional que detectará si la comunicación serie ha comenzado o no:

```
if (!pulseSensor.begin()) {
  for(;;) {
```

```
      // Flash the led to show things didn't work.
      digitalWrite(PULSE_BLINK, LOW);
      delay(50);
      digitalWrite(PULSE_BLINK, HIGH);
      delay(50);
    }
  }
```

Dentro de este condicional solamente entraremos si la comunicación no funciona (cosa que con nuestro Arduino UNO no ocurrirá), y ahí encontraremos un bucle sin fin creado gracias a la expresión **for (; ;)** Lo que hace el código encerrado dentro de ese bucle es encender y apagar constantemente el LED que hemos definido anteriormente como el que marcará nuestras pulsaciones:

```
    for(;;) {
      digitalWrite(PULSE_BLINK, LOW);
      delay(50);
      digitalWrite(PULSE_BLINK, HIGH);
      delay(50);
    }
```

void loop(): lo que se ejecuta todo el rato

Y por último, encontramos la función loop() con el código que ha de ejecutarse continuamente. A continuación te escribo el código sin comentarios, para que te resulte más fácil verlo:

```
void loop() {

  delay(20);

 pulseSensor.outputSample();

  if (pulseSensor.sawStartOfBeat()) {
   pulseSensor.outputBeat();
  }
}
```

Lo primero que encontramos es un **delay** para mantener las lecturas en orden y sin problemas. Lo siguiente, es utilizar la función **.outputSample()** que es la que escribe los valores recibidos por el puerto serie.

El condicional que lo sigue, dice que si hemos detectado una pulsación, tendremos que dibujar esa información en nuestro **plotter**. Con el Arduino conectado al ordenador, subimos el código y podremos visualizar en nuestro Serial Plotter algo similar a esto:

image-1666685814556.gif

¿Qué tenemos que presentar?

En este caso, al igual que con la práctica anterior, será suficiente con que me envíes un **.gif o video** en el que se vea el pulsómetro sin tu dedo, luego con tu dedo sobre él, y que posteriormente la cámara enfoque al monitor y con el puerto serial abierto, sea posible ver tus pulsaciones en el plotter. Similar al video que aparece justo encima de este apartado.