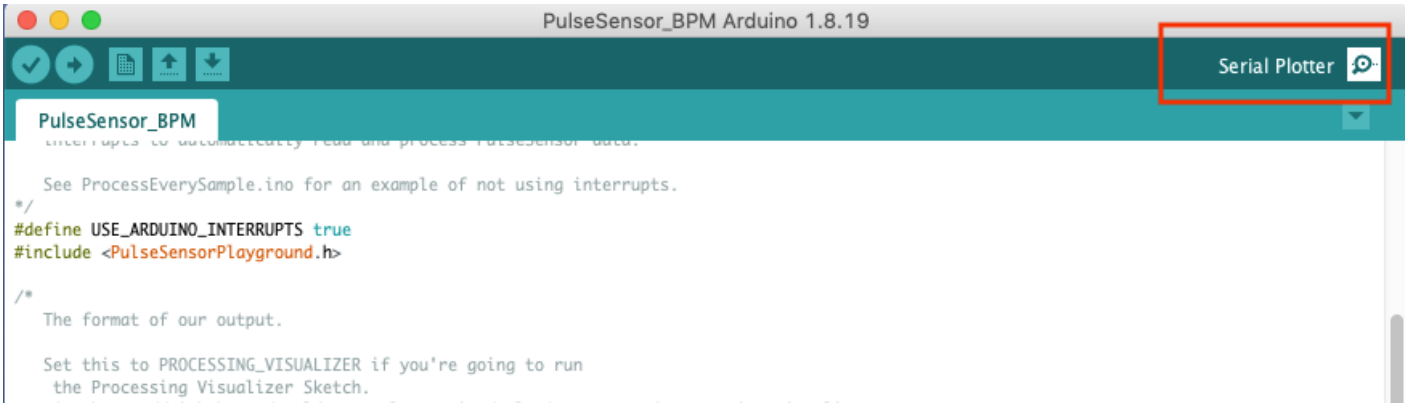
Práctica 9.3: Visualizamos cada pulsación en el serial plotter

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**.



¿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.
*/
```

Copyright 2026

```
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.
```

Copyright 2026

```
      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:

Copyright 2026

```
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

Copyright 2026

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);
  }
```

Copyright 2026

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);

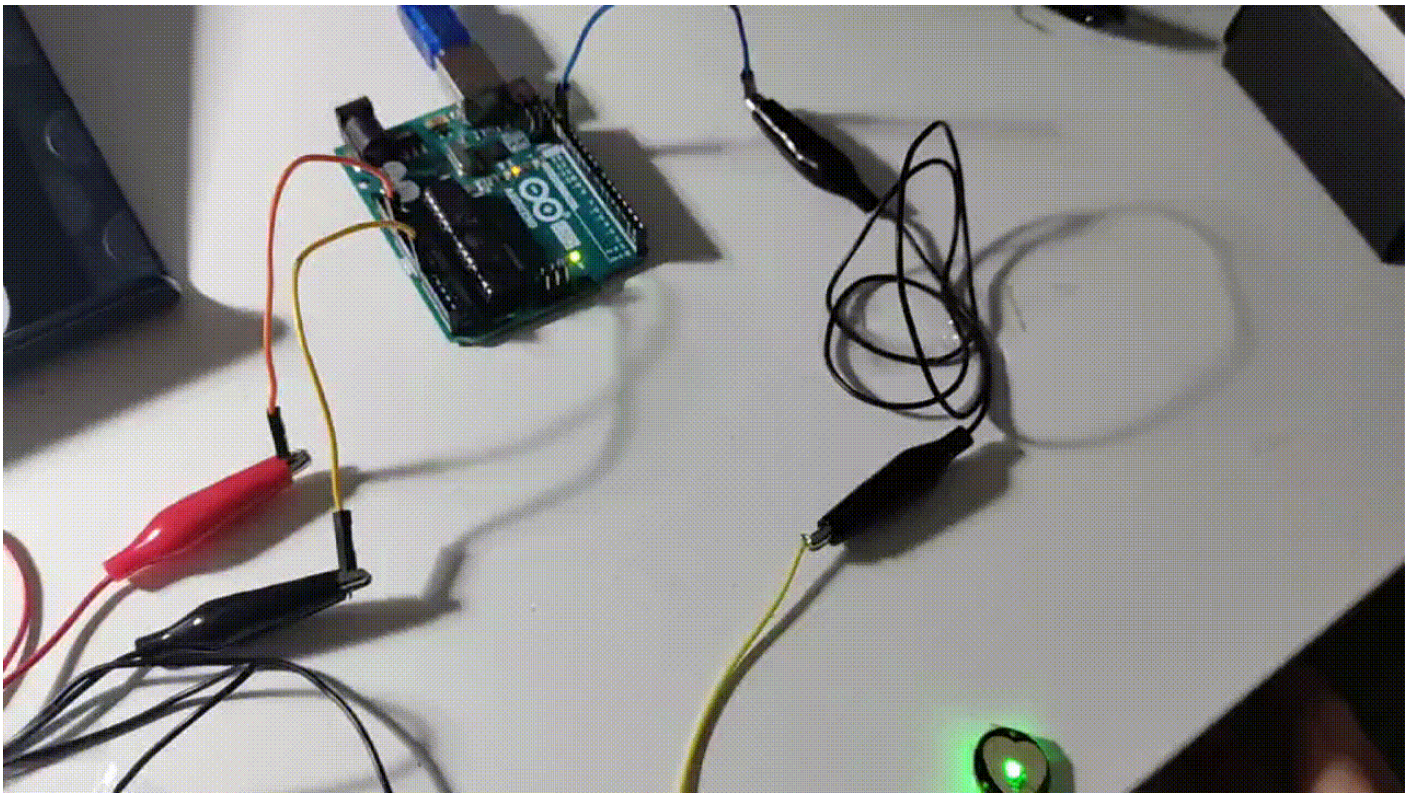
 pulseSensor.outputSample();

  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:

Copyright 2026



Copyright 2026