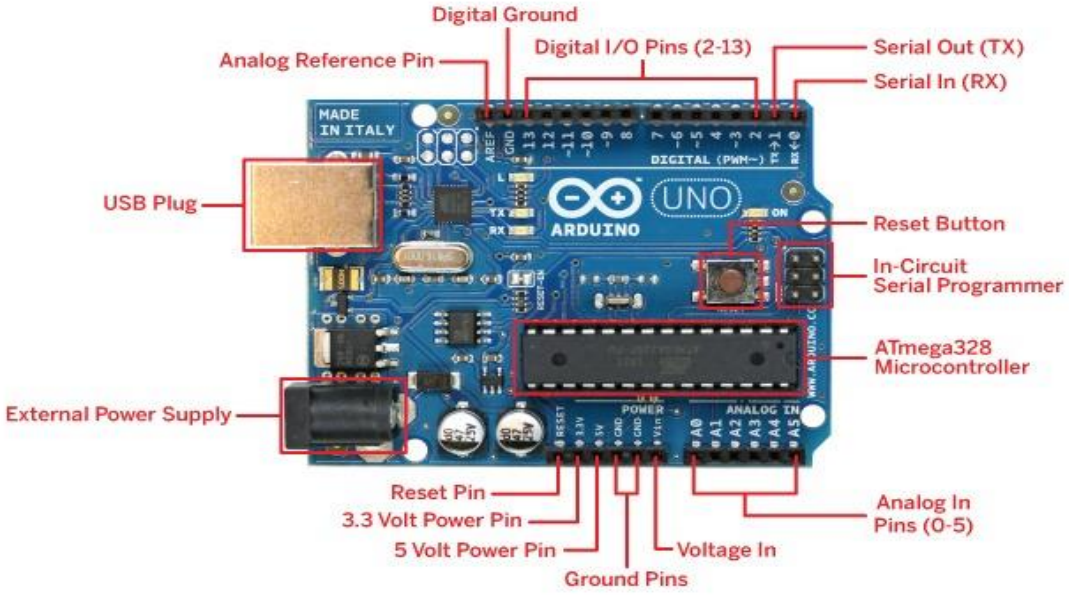
Arduino UNO R3

Características

Arduino es una plataforma de hardware libre basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares.

Arduino UNO R3, que es la placa usada para este proyecto, es una placa electrónica basada en el ATmega328 de Atmel[1] (ficha técnica). Cuenta con 14 pines digitales de entrada/salida, de los cuales 6 pueden ser para salidas PWM (salidas analógicas), 6 entradas analógicas, un resonador cerámico de 16 MHz, un conector USB, un conector de alimentación, un conector ICSP y un botón de reinicio.



Estas eran las características generales, las características específicas son estas:

Entradas y salidas

-14 pines digitales: Cada uno de estos pines se pueden utilizar como entrada o salida digital configurándolos con las funciones pinMode(), digitalRead() y digitalWrite(). Cada pin puede proporcionar o recibir un máximo de 40 mA y tienen una resistencia de pull-up (desconectada por defecto) de 20 a 50 kOhm.

-PWM 3,5,6,9,10 y 11: Modulación por anchura de pulsos de 8 bits (256 valores) con la función analogWrite().

-Serial 0(RX) y 1(TX): Se utilizan para recibir o trasmitir datos en serie TTL. Cada uno de ellos está conectado a los pines correspondientes del chip USB-to-Serial ATmega16U2.

-SPI (Serial Peripheral Interface) pines 10 (SS), 11(MOSI), 12(MISO) y 13(SCK): Estos pines se utilizan para la comunicación serie con periféricos conectados.

-LED 13: Es un LED en la placa ARDUINO UNO y conectado al pin 13. Cuando el valor de salida de este pin es de nivel alto (HIGH) el LED se enciende y cuando es de nivel bajo (LOW) el LED se apaga.

-6 entradas analógicas desde A0 a A5: Digitalizan con una resolución de 10 bits. 6 entradas con un rango de 0 V a 5 V.

-AREF: Se utiliza para cambiar el rango de las entradas analógicas con la función analogReference().

-Reset: Si pones el valor de este pin a nivel bajo (LOW) reinicializará el microcontrolador. Suele utilizarse cuando quieres añadir un botón de reinicio externo porque no puedes acceder al de la placa Arduino.

Alimentación

El Arduino UNO puede alimentarse a través de la conexión USB o con una fuente de alimentación externa. La fuente de alimentación se selecciona automáticamente. La alimentación externa se conecta mediante un Jack de 2.1 mm y puede tratarse tanto de baterías como de un convertidor AC/DC conectado a una toma eléctrica.

Aunque la placa puede funcionar con un suministro externo de 6 a 20 voltios, si se alimenta la placa con menos de 7V, el pin de 5V puede suministrar menos de este voltaje y si se alimenta con más de 12V el regulador de voltaje se puede sobrecalentar. El rango recomendable es de 7 a 12 Voltios.

Pines de alimentación:

-Vin: Puedes alimentar la placa a través de este pin.

-5V: Este pin tiene como salida 5 Voltios regulados por el regulador de la placa. No confundirlo como una entrada de alimentación, porque dañaría la placa.

-3.3V: Suministro de 3.3V generado por el regulador de tensión de la placa.

-GND: Pines de masa.

-IOREF: Este pin de la placa proporciona la referencia de tensión con la que tiene que trabajar el microcontrolador. Un protector configurado lee el voltaje de IOREF y selecciona la fuente de alimentación adecuada o habilita conversores de tensión en las salidas para trabajar con los 5V o 3.3V.

Memoria

El ATmega328 tiene 32KB disponibles para almacenar el código del programa, de estos 32KB, 0.5KB están reservados para el bootloader o gestor de arranque. Además cuenta con 2 KB de SRAM y 1 KB de EEPROM (que puede ser leído y escrito con la librería EEPROM).

Comunicación

El Arduino UNO tiene una serie de facilidades para comunicarse con un ordenador, otro Arduino, otro micro controlador o, como en este caso, con un dispositivo móvil. El ATmega328 ofrece UART TTL (5V), una comunicación en serie, que está disponible en los pines digitales 0(RX) y 1(TX). Además, gracias al firmware Atmega16U2 que utiliza los controladores USB COM estándar, la placa no necesita ningún controlador externo y aparecerá en el ordenador como un puerto COM virtual. La diferencia con respecto a las placas Arduino anteriores es que esta no utiliza el controlador de USB a Serie FTDI. En lugar de ello cuenta con la ATmega16U2 programado para convertir de USB a Serie. Los LEDs RX y TX parpadearan mientras haya una trasmisión de datos entre el chip ATmega16U2 USB-to-Serial y el puerto USB del ordenador, pero no cuando haya una trasmisión serie por los pines 0 y 1.

Con la biblioteca SoftwareSerial puedes hacer que cualquiera de los pines de la placa Arduino funcione como bus de datos serie, solo tienes que asignar un pin RX que actuara como receptor de datos y otro pin TX como trasmisor de datos y luego ya usar las funciones write y read propias de esta biblioteca.

El IDE (Integrated Development Environment) de Arduino tiene una herramienta que se llama **Serial Monitor** y es una consola o pantalla intermediaria entre el ordenador y Arduino. Se puede usar, por ejemplo, para que aparezcan los datos que se transfieren entre los dos dispositivos o cualquier otro dato que para verificar que existe o simplemente que ha llegado lo imprimes en pantalla. En este Monitor Serial se puede teclear cualquier cosa y leerla en la placa Arduino añadiéndole al código la función SerialRead(), o desde la placa escribirle algo al ordenador con la función SerialPrint(). Cabe recordar que toda esta información y comunicación se transfiere a través del puerto serie.

Características físicas

-Longitud: 68,6mm

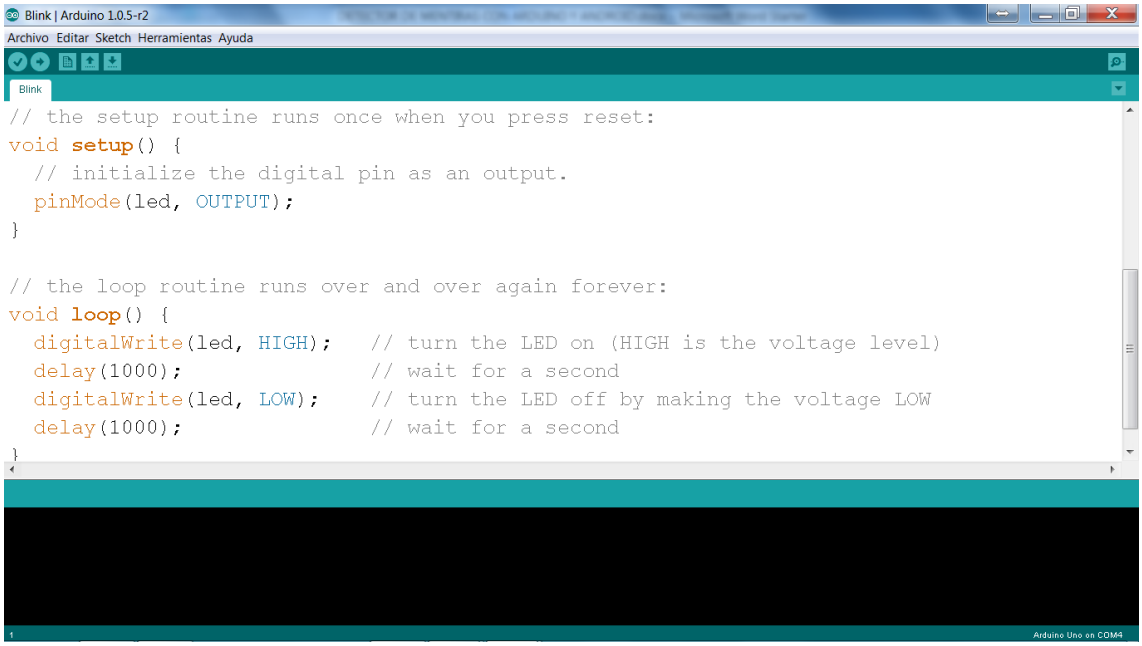
-Ancho: 53,4mm

-Peso: 25g

IDE de Arduino

El Entorno de Desarrollo Integrado o IDE por sus siglas en inglés, es un software que implementa el lenguaje de programación Processing/Wiring, sin embargo es posible utilizar otros lenguajes de programación y aplicaciones populares de Arduino, debido a que Arduino usa la trasmisión serie de datos soportada por la mayoría de los lenguajes. Para los que no soportan el formato serie de forma nativa es posible utilizar software intermediario que traduzca los mensajes enviados por ambas partes para permitir una comunicación fluida. Arduino está basado en C y soporta todas las funciones del estándar C y algunas del C++.

En este proyecto nos centramos en el IDE proporcionado por la página web oficial.



IDE Arduino con el Sketch de ejemplo Blink.

Para que el IDE reconozca la placa Arduino y poder cargar el código de los programas se debe conectar el puerto USB de la placa al puerto USB del ordenador mediante un cable USB tipo A-B, una vez conectado asegurarnos de que nos lo ha reconocido pinchando en **Herramientas>Puerto Serie** y comprobando que existe un dispositivo conectado en **COM*.** También tenemos que comprobar que la Tarjeta (tipo de Arduino) seleccionada es la correcta. La primera vez en Windows habrá que esperar a que se instale el Software del controlador.

Programa en Arduino o Sketch

El programa que se crea y el que se cargara se llama **Sketch**. El sketch utiliza una estructura muy sencilla:

1º.Introducir las bibliotecas que se van a utilizar.

2º.Inicializacion de las variables del programa.

3º. **void setup()**: Función en el que se asignan como entrada (INPUT) o salida (OUTPUT) cada uno de los pines de la placa que se van a utilizar, con la función pinMode(pin,tipo). Se ejecuta sólo una vez, al inicio.

4º. **void loop()**: Función principal del programa donde ira el código que se ejecutará y hará funcionar al Arduino. Esta función se ejecuta continuamente, en bucle.

Pulse Sensor Sparkfun 11574

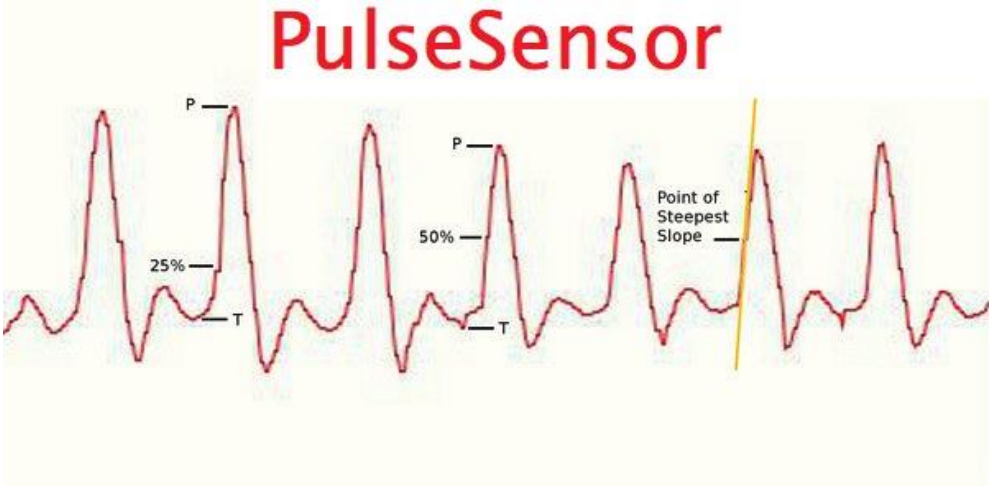
Características

El Pulse Sensor es un sensor de ritmo cardiaco para Arduino. Este sensor es usado habitualmente, como en nuestro caso, para introducir de algún modo la frecuencia cardiaca en un proyecto. Como se puede observar en la parte frontal del sensor, aparte de que hay un dibujo en forma de corazón, hay un agujero por donde sale la luz emitida desde un LED que se encuentra en la parte posterior. Justo debajo del agujero se encuentra un sensor de luz ambiente, como el que se usa en los smartphones para atenuar o subir automáticamente el brillo de la pantalla dependiendo de la luminosidad del espacio donde te encuentras. Esta parte frontal es la que está en contacto con la piel y preferiblemente en alguna parte translúcida del cuerpo ya sea la punta del dedo o el lóbulo de la oreja e irá sujetada con una cinta de velcro o un clip que ya vienen incluidos cuando adquieres el sensor.

Cuando el corazón bombea sangre a través del cuerpo, con cada latido se genera una onda de pulso (algo así como una onda de choque) que viaja por todas las arterias y venas del tejido capilar, que es donde se une con el Pulse Sensor. Esta onda viaja más rápido que la sangre. El Pulse Sensor responde a los cambios relativos en la intensidad de luz. Estos cambios de luz se generan porque en el paso de esta onda hay más cantidad de sangre que bloqueará la luz emitida por el LED y que es la percibida por el fotosensor. Si la cantidad de luz que incide sobre el sensor se mantiene constante, el valor de la señal permanecerá cerca de 512 que es la mitad del rango de medición. Si recibe más luz la señal sube y si recibe menos la señal baja.

El pulse Sensor usado para este proyecto es esencialmente un fotopletismógrafo, que es un dispositivo médico conocido utilizado para el monitoreo de la frecuencia cardiaca no invasiva. La señal del pulso del corazón que sale de un fotopletismógrafo, no es la señal eléctrica del corazón como se tendría en un electrocardiograma, sino

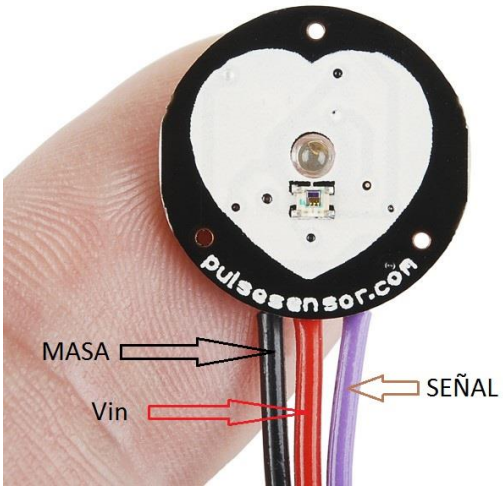
que es una fluctuación en el voltaje analógico, y tiene una forma de onda predecible como se muestra en la imagen. La representación de la onda de pulso se llama fotopletismograma o PPG.



El cometido de este dispositivo es encontrar los momentos sucesivos de cada latido y calcular el intervalo de tiempo que ha habido entre ellos llamado IBI (Inter-Beat Intervale), haciendo uso de la forma predecible y del patrón de onda del PPG.

Conexiones

El Pulse Sensor dispone de 3 cables para conectarse.



-Vin (rojo): Cable de alimentación del sensor, se puede alimentar desde 3V hasta 5V por lo tanto podrá ir tanto al pin de 3.3V de la placa Arduino como al de 5V.

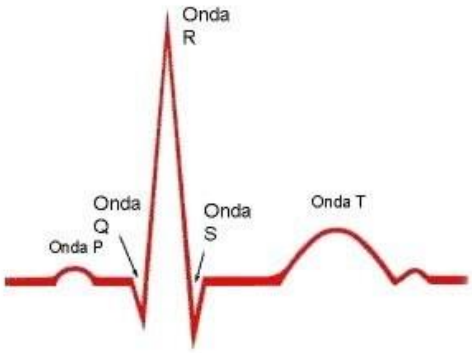
-GND (negro): Cable de masa, irá conectado al pin GND de la placa Arduino.

-Señal (morado): Cable por donde se transfieren los datos recogidos por el fotosensor. Es una señal analógica, por lo tanto, irá a una de las 6 entradas analógicas de la placa Arduino.

Consideraciones sobre el ritmo cardiaco

Antes de explicar el código del programa y para entender el funcionamiento de este haremos una pequeña explicación sobre que es el Ritmo Cardiaco y su representación en un electrocardiograma (ECG).

Un electrocardiograma es la representación gráfica de la actividad eléctrica del corazón. En un entorno médico normal se usaría un electrocardiógrafo que mide directamente la corriente del corazón con unos electrodos. El trazado típico de un ECG registrando un latido cardíaco normal consiste en una onda P, un complejo QRS y una onda T.

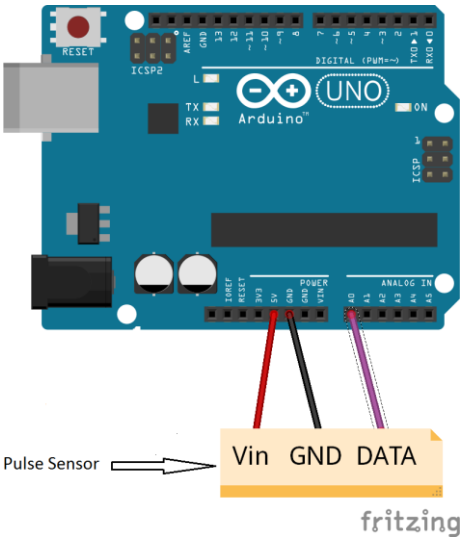


Esta variación de la actividad eléctrica representada por el ECG, es generada por polarizaciones y despolarizaciones que ocurren entre las células de los músculos en las distintas partes el corazón. Para calcular el número de latidos por minuto a través de la señal del ECG basta con saber el intervalo de tiempo transcurrido entre latido y latido y multiplicarlo por 60. Para hacer un cálculo más exacto, el intervalo se medirá entre las dos ondas R que son más pronunciadas.

Como se puede observar en las imágenes anteriores hay un parecido más que razonable entre las forma de onda del fotopletismograma, generada a partir de la variación de luz en un tejido capilar, y la del electrocardiograma, generada a partir de la actividad eléctrica del corazón. Aunque estén muy relacionadas, ya que las dos nos mostraran en el tiempo cuando se produce un latido en el corazón, cada una proviene de una fuente distinta y no hay que confundirlas.

Ritmo cardiaco con el Pulse Sensor

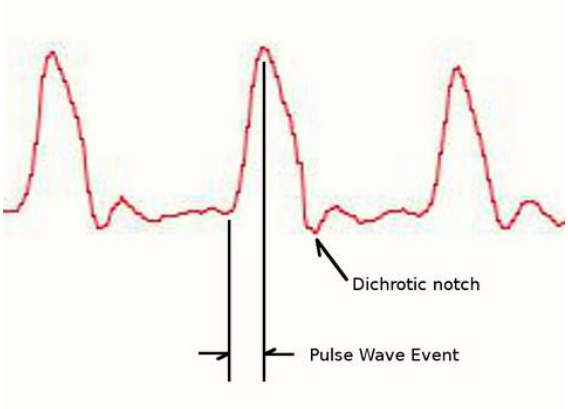
Para hacer funcionar el sensor y probar como calcula y muestra el ritmo cardiaco instantáneo se tiene que conectar el sensor a la placa Arduino de la manera que se detalla en la imagen:



El pin Vin a los 5V de la placa Arduino, el GND a masa y el pin de datos a la entrada analógica A0.

También habrá que cargar en el Arduino el sketch correspondiente que se puede descargar desde el siguiente enlace: "Ritmo Cardiaco". El código del programa es proporcionado por la página oficial de Sparkfun.

Cuando la onda de pulso pasa por debajo del sensor se produce una subida rápida en la forma de onda del PPG. Cuando ha pasado, el valor de la señal desciende rápidamente y antes de volver a su valor normal generará un valle. Esta señal también la tendremos en cuenta a la hora de detectar si la onda que nos envía el Pulse Sensor corresponde a un pulso o es solo ruido.



Para calcular las pulsaciones por minuto o BPM (Beats Per Minute) tendremos que establecer primero que punto en la onda del pulso tomaremos como referencia para poder después calcular los intervalos de tiempo entre esos puntos. Muchos médicos dicen que el pulso se genera cuando el valor de la onda llega al 25% del valor de pico, otros que cuando la pendiente es más pronunciada, nosotros tomaremos como referencia el punto en el que el valor de la señal está al 50% del valor de la amplitud de la onda.

Aclarado todo esto, pasaremos a explicar el funcionamiento del programa en sí. Lo primero que se hace es configurar una interrupción cada 2ms, que nos generará una frecuencia de muestreo de 500 Hz. Cada 2 ms se llamará a la función *ISR()* (Rutina de Servicio de Interrupción) que leerá el valor de la entrada analógica A0 y buscará si ha habido un pulso. El valor de la entrada analógica se guarda en una variable llamada "S". Dentro de esta rutina hay un contador llamado *sampleCounter* que nos ayudará a controlar el tiempo dentro de la interrupción. A continuación se hará un seguimiento de los valores más altos y más bajos de la onda para obtener una medida exacta de la amplitud de esta. Las variables P y T son el valor de pico de la señal y el valor del valle, respectivamente. En un principio la variable umbral, que será la mitad del valor de la amplitud de la onda, se inicializará a 512 que es el valor medio del rango analógico e irá cambiando a lo largo del programa. También haremos que transcurra un intervalo mínimo de tiempo entre lectura y lectura, que se corresponderá con 3/5 de IBI, y así, evitar el ruido y las lecturas falsas. Código:

```
void interruptSetup(){
  TCCR2A = 0x02;
  TCCR2B = 0x06;
  OCR2A = 0x7C;
  TIMSK2 = 0x02;
  sei();
}

ISR(TIMER2_COMPA_vect){
  Signal = analogRead(pulsePin);
  sampleCounter += 2;
  int N = sampleCounter - lastBeatTime;

//busqueda de la pendiente y del valle

   if(Signal < thresh && N > (IBI/5)*3){
      if (Signal < T){
        T = Signal;
      }
    }
    if(Signal > thresh && Signal > P){
        P = Signal;
      }
```

Ahora que ya tenemos las variables asignadas e inicializadas vamos a comprobar y a ver si tenemos un pulso. Antes de iniciar la búsqueda de un latido pondremos la

condición de que no lo busque mientras N sea menor de 250. N es igual al tiempo del contador interno menos el tiempo del último latido. Hay que esperarse por lo menos 250ms a buscar otro latido porque sería muy raro encontrar dos latidos del corazón con un intervalo menor de 250ms, esto significaría que el corazón está latiendo a más de 240 pulsaciones por minuto, sería raro y peligroso. Cuando han pasado estos 250ms y 3/5 partes de tiempo del último IBI y la señal supera el valor del umbral, significará que ha habido un latido. Entonces se encenderá el LED del pin 13, se calculará el tiempo transcurrido desde el último latido para calcular el IBI y se cambiará el valor de la variable lastBeatTime. Código:

```
if (N > 250){

if ( (Signal > thresh) && (Pulse == false) && (N > ((IBI/5)*3) ){
    Pulse = true;
    digitalWrite(pulsePin,HIGH);
    IBI = sampleCounter - lastBeatTime;
    lastBeatTime = sampleCounter;

//ha habido un latido!
```

El siguiente código se utiliza para asegurarnos de que se empieza con un valor de BPM realista.

```
if(secondBeat){
      secondBeat = false;
      for(int i=0; i<=9; i++){
        rate[i] = IBI;
      }
    }

                if(firstBeat){
                   firstBeat = false;
                   secondBeat = true;
                   sei():
                   return;
                }
```

Ahora vamos a proceder a calcular los BPM. Se crea una variable llamada RunningTotal que va almacenando todos los valores de IBI encontrados. Cuando vayamos obteniendo valores de los intervalos entre latidos lo que haremos será realizar una media de los 10 últimos, para hacer un promediado, y este valor se lo dividiremos a 60000 para obtener los BPM o pulsos que se tienen en un minuto. Ejemplos:

-Si el promedio de los últimos 10 latidos es de 1000ms:

$$IBI = 1000ms \ ; \ BPM = \frac{60000}{1000} = 60 \ pulsaciones \ por \ minuto$$

-Si el promedio de los últimos 10 latidos es 750ms:

$$IBI = 750ms \ ; \ BPM = \frac{60000}{750} = 80 \ pulsaciones \ por \ minuto$$

Ahora que ya lo tenemos todo calculado, tendremos que actualizar las variables que guardan los valores tanto de los IBI como de los BPM que son las variables Q y B, respectivamente. Código:

```
word  runningTotal = 0;
    for(int i=0; i<=8; i++){
      rate[i] = rate[i+1];
      runningTotal += rate[i];
    }
    rate[9] = IBI;
    runningTotal += rate[9];
    runningTotal /= 10;
    BPM = 60000/runningTotal;
    QS = true;
    }
```

Ahora hay que preparar las variables para la búsqueda del siguiente latido. Cuando anteriormente el valor de la señal había sobrepasado el valor del umbral habíamos declarado que se había generado un latido, ahora cuando el valor de la señal baje del valor del umbral indicaremos que ese latido ha finalizado y apagaremos el LED del pin 13. Calcularemos el valor del nuevo umbral y se lo asignaremos a las variables P y T. Código:

```
if (Signal < thresh && Pulse == true){
    digitalWrite(13,LOW);
    Pulse = false;
    amp = P - T;
    thresh = amp/2 + T;
    P = thresh;
    T = thresh;
  }
```

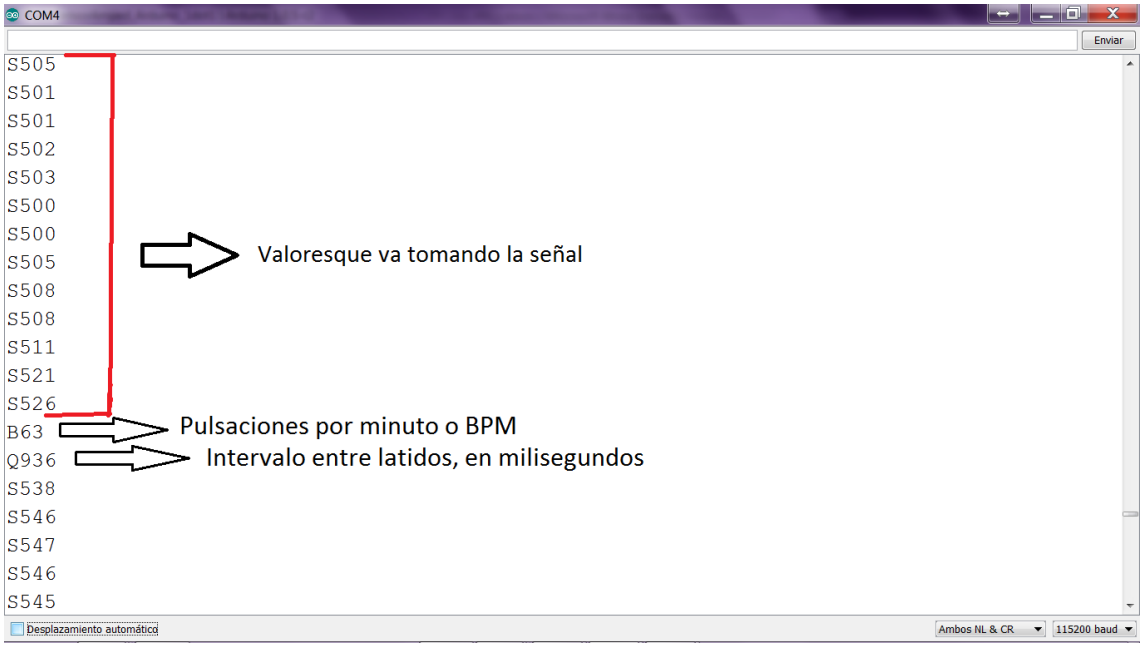
El siguiente código es por si durante un intervalo de tiempo de 2,5 segundos no ha habido ningún latido. En ese caso se reinicializarán las variables a los valores de arranque. Código:

```
if (N > 2500){
    thresh = 512;
    P = 512;
    T = 512;
    firstBeat = true;
    secondBeat = false;

    lastBeatTime = sampleCounter;

  }
```

Aquí termina el servicio de interrupción.

Cuando ya se han calculado el IBI y los BPM, la interrupción salta al programa principal para mostrar estos valores por pantalla a través del Monitor Serie. El valor de la señal se va mostrando todo el rato con una letra indicadora “S” seguida del valor. Las variables IBI y BPM se mostrarán por pantalla y como ya hemos dicho anteriormente, irán precedidas de una letra indicativa, “B” para el caso de los BPM y “Q” para el caso del intervalo entre pulsos.

En la siguiente imagen se muestra como aparece en el Monitor Serie.



Visualización del fotopletismograma (PPG) en Processing

Ya hemos visto cómo con Arduino y el sensor Pulse Sensor de Sparkfun podemos calcular el ritmo cardíaco instantáneo además de diferentes valores. Pero la página web oficial del sensor nos proporciona también un sketch para correrlo en Processing que nos permitirá visualizar en el momento la variación de luz transformada en onda captada por el sensor, formando un fotopletismograma. Primero haremos una breve introducción de lo que es Processing.

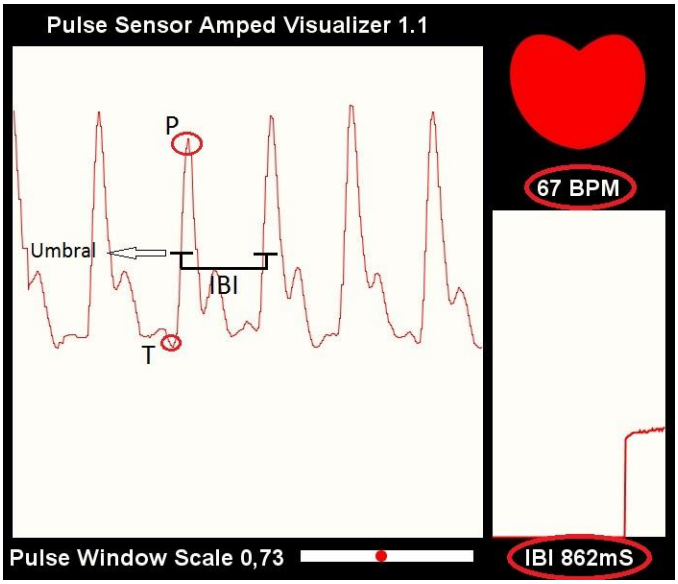
“Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto (open source) basado en Java. Suele utilizarse para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. Al estar basado en Java puede heredar todas sus funcionalidades y ser una herramienta poderosa a la hora de encarar proyectos complejos.” (fuente: entrada de Processing en Wikipedia)”.

El programa de Processing se puede descargar desde su página web oficial y desde aquí también podéis descargar el sketch que habrá que ejecutar con dicho programa. Solo hay que seguir los siguientes pasos:

1º. Conectar el Arduino y el Pulse Sensor como hemos descrito anteriormente. Colocarse el sensor en la punta del dedo y ejecutar el sketch "PulseSensorAmped_Arduino_1dot1" en la placa Arduino.

2º.Una vez se está corriendo el programa en Arduino, y habiéndonos descargado previamente el programa de Processing y su sketch, deberemos ejecutar Processing y correr el sketch llamado "PulseSensorAmpd_Processing_1dot1".

Nos tiene que aparecer una ventana como esta:



Se puede observar como aparece la señal del PPG y los diferentes puntos, como el pico P, el valle T y el umbral, que nos ayudarán a reconocer cada pulso y realizar los cálculos con ellos para determinar el intervalo entre latidos y la cantidad de pulsos por minuto, que dichos valores también aparecen representados en la pantalla de Processing. El dibujo del corazón en rojo latirá al ritmo de nuestras pulsaciones.

El programa en Processing funciona porque en el sketch de Arduino está esta función:

```
void sendDataToProcessing(char symbol, int data ){
  Serial.print(symbol);        // este simbolo indicará a Processing que tipo de dato esta recibiendo
  Serial.println(data);        // Valor a enviar seguido de un salto de línea y retorno de carro
```

Que lo que hace es enviar por el puerto serie e imprimir en pantalla los valores que van teniendo cada variable, y poder tomarlos desde el sketch de Processing para ir dibujando la onda. Por eso es imprescindible correr el programa en Arduino antes de

abrir el sketch en Processing, porque si no se envían estos datos el programa de Processing no puede hacer nada.

Error inherente del Pulse Sensor

Para saber la fiabilidad del sensor a la hora de calcular la variación de luz habida en la punta del dedo, como no podemos saber cada cuanto tiempo pasa de haber más oposición al paso de la luz a haber menos y de ahí ya sacar el error de captura que nos mete el sensor, lo que podemos hacer es crear un sistema donde estas variaciones de luz las tengamos controladas y sepamos cada cuanto se producen, medirlas con el sensor y dependiendo de los valores obtenidos sabremos lo preciso que es el sensor a la hora de calcular dichas variaciones.

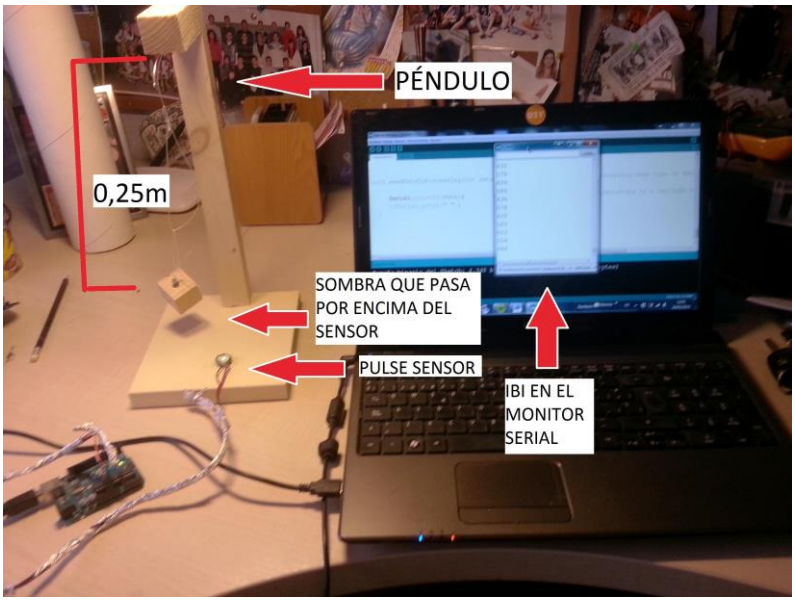
El sistema creado para obtener variaciones de luz controladas y saber su periodo será un péndulo con una masa atada en su extremo y colocada justo arriba del sensor para que pueda bloquear la cantidad de luz que le llega, así, con cada oscilación, creará un instante de luz y un instante de sombra. Se ha elegido un péndulo porque el periodo de oscilación de este, que va directamente ligado al periodo de variación de la luz, es independiente tanto de la masa que se ata como del ángulo desde el que se hace oscilar, solo depende de la longitud de la cuerda y se rige por la siguiente ecuación:

$$T \approx 2\pi\sqrt{\frac{\ell}{g}}$$

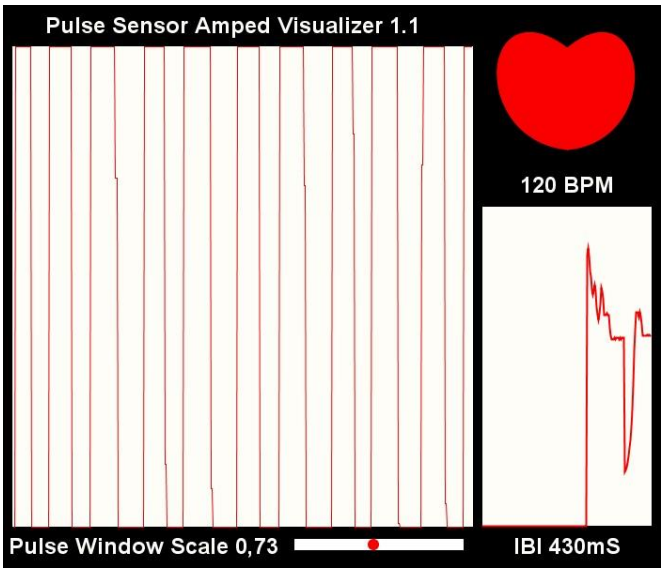
Que define que el periodo del péndulo es igual a 2π por la raíz cuadrada de la longitud de la cuerda partido por la aceleración de la gravedad. En esta fórmula no interviene la masa porque la aceleración de la caída de un objeto es independiente de su masa solo influye la aceleración de la gravedad[2]. Como se puede observar son todo constantes excepto la longitud y siendo la gravedad g=9,8m/s^2. Por eso si queremos tener un periodo de T=1 segundo, por ejemplo, despejamos la longitud y nos sale que la cuerda debería medir 0.25 metros.

$$l = \frac{1 \times 9.8}{4\pi^2} \approx 0.25\ metros$$

El sistema que hay que montar para medir la precisión del sensor a la hora de detectar variaciones de luz es el siguiente:



Para hacer una primera prueba, colocaremos el sensor como en la imagen y correremos los programas como hemos hecho anteriormente cuando lo teníamos enganchado en el dedo, para poder apreciar visualmente en Processing la gráfica de la variación de luz y la cantidad de pulsaciones por minuto que nos da. Hay que tener en cuenta que si hemos calculado para que el periodo del péndulo sea de 1 segundo, este periodo es el tiempo que tarda en hacer una oscilación completa por lo que pasará dos veces, una al ir y la otra al volver, por encima del sensor provocando dos instantes de luz/sombra cada 500 milisegundos. Por lo que si lo extrapolamos al ritmo cardiaco si hay una pulsación/variación de luz cada 500ms tendremos 120 pulsaciones por minuto. Processing nos muestra la siguiente gráfica:



Podemos comprobar como efectivamente el sensor detecta variaciones de luz muy bruscas que crean una onda que es un pulso rectangular con un periodo aproximado de 500ms y en consecuencia calcula 120 BPM que era lo que, de antemano, pensábamos obtener. Lo que nos confirma que es un sensor bastante fiable porque es lo suficientemente preciso como para detectar variaciones de luz programadas.

Pero la comprobación del error inherente que tiene el sensor no acaba aquí, queremos saber el valor exacto de los intervalos entre latidos que va calculando, por lo que vamos a realizar lo siguiente. Como solo queremos que el programa nos muestre los intervalos tenemos que hacer una pequeña modificación en el código del programa. Tenemos que comentar (poniendo "//" delante) las sentencias en las que indicamos que muestre por pantalla el valor de la señal y el valor de los BPM, y como solo queremos el valor del intervalo sin la letra "Q" delante también hay que hacer una pequeña modificación en la función "SendDataToProcessing()" que nos muestra los valores por pantalla. El código quedaría así:

……

……

```
void loop(){
  //sendDataToProcessing('S', Signal);    //Comentado para que no salga
  if (QS == true){
        fadeRate = 255;
        //sendDataToProcessing('B',BPM); //Comentado para que no salga
        sendDataToProcessing(IBI);//Solo envia el valor de IBI sin la letra
        QS = false;
     }
```

……

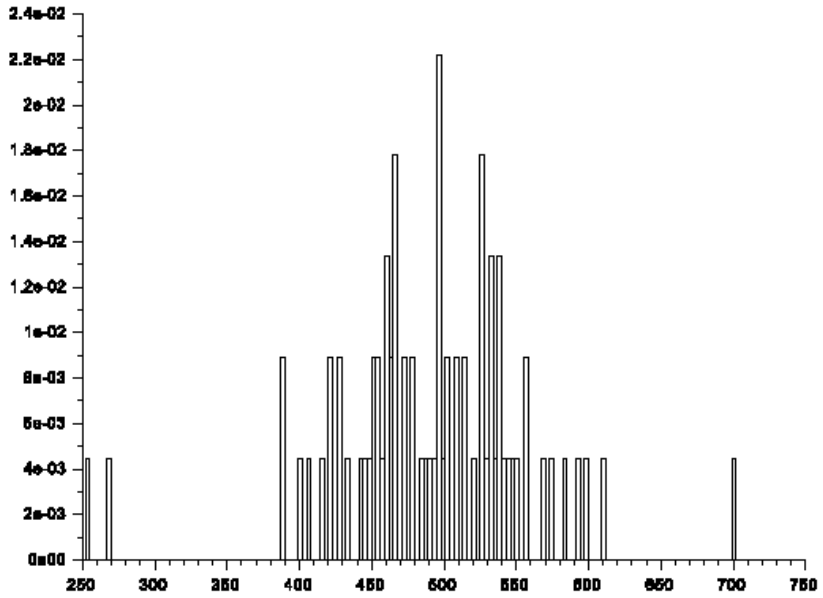
……..

```
void sendDataToProcessing(int data){  //Ponemos que solo envie el valor
//'data' sin la letra indicativa
     Serial.print(data);               // Imprime en pantalla el valor
     Serial.print(" ");   //Añade un espacio de separación para que salgan
//correlativos y legibles
}
```

Una vez modificado el código y montado todo, tendremos que hacer oscilar el péndulo por encima del sensor y que nos muestre en pantalla (Monitor Serial) el tiempo de los intervalos entre las variaciones de luz y sombra. Cuando lo tengamos oscilando entre 15 y 20 segundos cogeremos y copiaremos todos los valores que nos ha dado. Con

estos valores lo que haremos será generar un histograma que nos muestre su dispersión, esta grafica la crearemos con cualquier herramienta de software matemático, el elegido en este caso es SciLab. Para generar un histograma con los valores que hemos obtenido, generamos un vector que contenga todos estos valores y le aplicamos la función 'histplot()'.

Entonces nos saldrá esta gráfica:



Histograma de los valores obtenidos del péndulo.

Como se puede observar en la gráfica los valores crean una campana de Gauss concentrando el mayor número de valores en los 500ms y entre los 450ms y 550 ms. No obstante, también hay registros de intervalos con valores de 400ms y 600ms. Esto se podría deber, además de al error inherente del sensor que estamos calculando, al error que nos da el péndulo, la fricción del aire va ralentizando su periodo, hay rozamiento en el enganche entre la cuerda y la masa, etc.

Al final la conclusión que sacamos es que el sensor tiene un **error de ±100ms** a la hora de calcular las variaciones de luz. Aun así garantizamos la plena fiabilidad de este a la hora de calcular el ritmo cardíaco instantáneo basándonos en los datos obtenidos con Processing y comprobando como efectivamente nos calculaba 120 pulsaciones por minuto que era justo lo que esperábamos y debíamos obtener.

Módulo Bluetooth HC-05

Características

El modulo Bluetooth HC-05 (data sheet) es uno de los dispositivos más populares usados para añadirle la funcionalidad de la comunicación Bluetooth a proyectos con Arduino o micro-controladores. Se trata de dispositivos relativamente económicos y que se adquieren en un formato que se permite insertarlo en una protoboard directamente y cablearlo a cualquier microcontrolador sin necesidad de soldaduras.

El módulo HC-05 tiene dos modos de funcionar **Maestro** y **Esclavo**. Esto significa que no solo trabaja en la función esclavo que sería esperando ordenes que le puedes enviar desde un PC o desde un Smartphone, como es este el caso, sino que también puede actuar como maestro y ser él quien se conecta a otro dispositivo Bluetooth, pudiendo así crear una conexión bidireccional punto a punto entre dos módulos permitiendo transferir y recibir información.

Conexiones

El HC-05 dispone de 6 pines de conexión. El modulo se encajara en una placa protoboard y desde ahí se cableara hasta los pines correspondientes de la placa Arduino.

Las conexiones son las siguientes:

-KEY: Este pin se puede conectar a cualquier otro pin da la placa Arduino, siempre y cuando este tenga salida digital. El pin KEY se debe activar a nivel alto cuando queremos entrar en el modo de configuración del módulo Bluetooth, una vez que hayamos entrado en este modo podremos cambiar sus parámetros con los comandos AT que luego explicaremos.

-RXD: Pin por donde se recibirán los datos que nos lleguen desde la placa Arduino. Este pin tiene que ir preciso cableado hasta el pin 1 TX de la placa Arduino que es por donde se trasmiten todos los datos que van por el puerto Serie o, en el caso de haber creado un puerto Serie virtual con otros dos pines de la placa, tendría que ir al que hayamos escogido como pin de TX.

-TXD: Pin por donde se trasmitirán datos desde el módulo Bluetooth hacia la placa Arduino. Este pin tiene que ir preciso conectado al pin 0 RX de Arduino o, en el caso de haber creado un puerto Serie virtual con otros dos pines de la placa Arduino, al pin que hayamos escogido como pin de RX.

-5.0: Pin de alimentación a 5 voltios del módulo Bluetooth. Irá conectado al pin 5V de la placa Arduino.

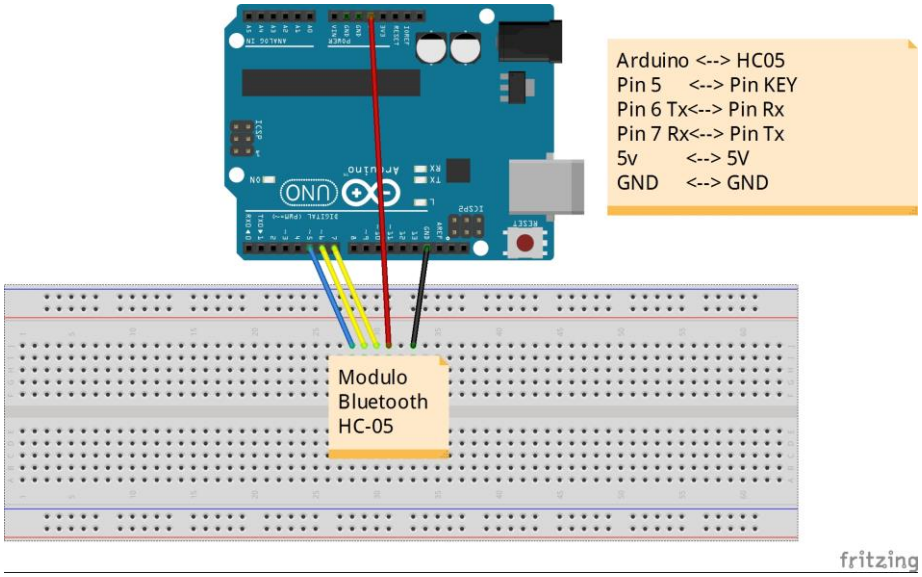
3.3: Pin de alimentación a 3.3 voltios. Irá conectado al pin 3.3V de la placa Arduino. El HC-05 se puede alimentar con cualquiera de estas dos tensiones.

-GND: Pin de masa del módulo HC-05.

Comandos AT

Los “comandos AT” fueron en sus comienzos, un pequeño grupo de instrucciones para comunicar el ordenador con un módem telefónico. Con los años a este tipo de instrucciones para comunicarse con diversos dispositivos se les popularizó como comandos AT y los módulos Bluetooth se pueden comunicar con un ordenador, con un microcontrolador o con cualquier otro dispositivo que posea una conexión serie (Rx/Tx) a través de estas instrucciones.

La conexión requerida para llevar a cabo la configuración del módulo es la que a continuación se detalla.



El pin KEY lo conectamos al pin 5 de la placa Arduino al que, posteriormente, le asignaremos como salida HIGH (5V) que es lo que hará que el HC-05 se ponga en modo de configuración preparado para recibir los comandos AT.

Una vez que tengamos conectados los dos dispositivos, tendremos que conectar el Arduino al PC mediante el cable USB, y desde el entorno de desarrollo cargar el sketch que previamente hemos escrito, que es este:

```
/*como la arduino uno solo dispone de un puerto serie, añadiremos
esta libreria para configurar los pines que
queramos como un puerto serie extra*/
#include <SoftwareSerial.h>
int Rx = 7;
int Tx = 6;
/*Definimos un nuevo puerto serie para la Arduino:
pin 6(Rx)<-->pin Tx del HC-05
pin 7(Tx)<-->pin Rx del HC-05*/
SoftwareSerial serial2(Rx,Tx);

void setup(){
  /*Abrimos puertos Serial(Arduino<->PC)
 y serial2(Arduino<->hc-05) y le asignamos
 los baudios correspondientes*/
 Serial.begin(9600);
 serial2.begin(38400);

  /*Asignamos el pin 5(KEY) como salida
  y le damos HIGH(5V) para poder activar la configuracion*/
  int KEY = 5;
  pinMode(KEY,OUTPUT);
  digitalWrite(KEY,HIGH);
  Serial.println("Introduzca los comandos AT");  }

void loop(){
  /*Le indicamos a la arduino que lo que reciba
```

```
 del PC se lo envie al hc-05 y viceversa*/
  if (Serial.available())
   serial2.write(Serial.read());


  if(serial2.available())
   Serial.write(serial2.read());
}
```

Como se puede observar en el código, hemos creado un puerto serie virtual en los pines 6 y 7 de la placa Arduino. Se ha hecho así para diferenciar el puerto serie que conecta PC<>Arduino y el puerto serie que conecta Arduino<>HC-05. Así, al hacer el Arduino de intermediario podrá trasmitirle al módulo Bluetooth por el puerto Serial2 lo que reciba del PC desde el puerto Serial y viceversa, transmitirle al PC a través del puerto Serial lo que reciba del módulo Bluetooth desde el puerto Serial2. Cabe recalcar que hay que abrir los distintos puertos serie a distintos baudios para la correcta comunicación, el Serial a 9600 Baudios y el Serial2 a 38400 Baudios.

Importante, para cargar el sketch se tiene que hacer con el Modulo Bluetooth apagado, es decir, desconectando el pin de alimentación de 5V y una vez cargado el sketch volverlo a conectar y observaremos como el LED parpadea alternativamente cada 2 segundos, esto indica que ha entrado en modo configuración, si en algún momento de las pruebas observamos que parpadea muy rápido, cada 0,5 segundos aproximadamente, tendremos que repetir el paso de desconexión y conexión del pin de alimentación. También tener en cuenta que el HC-05 detecta el comando cuando hay una serie de caracteres y se termina con un “\n” salto de línea y “\r” retorno de carro, indicando que ha finalizado el comando. En algún otro entorno de programación lo tendríamos que incluir al final del comando pero en el IDE de Arduino es tan fácil como seleccionar en el Monitor Serial la opción de “Ambos NL & CR” que se encuentra abajo a la derecha. Y para finalizar, por supuesto que se tiene que hacer la configuración con los comandos AT no estando el HC-05 conectado a ningún otro dispositivo.

Estos son los comandos más frecuentes y los que nosotros usaremos para configurar el módulo Bluetooth HC-05:

-**AT:** comando de test, si devuelve OK está listo.

-**AT+NAME:** Devuelve el nombre del módulo que por defecto será HC05 Si deseamos modificarlo hay que añadirle como parámetro el nuevo nombre.

AT+NAME=ARDUTOOTH

-**AT+PSWD:** Devuelve la contraseña que tendremos que usar para vincularnos con él, por defecto es “1111”. Si la deseamos modificar tendremos que añadir la nueva contraseña como parámetro.

AT+PSWD=1234

-AT+ADDR: Devuelve la dirección MAC del módulo Bluetooth. Es la que usaremos posteriormente para conectarnos desde el teléfono móvil.

-AT+ADDR

- ADDR=98:D3:31:B1:F1:55

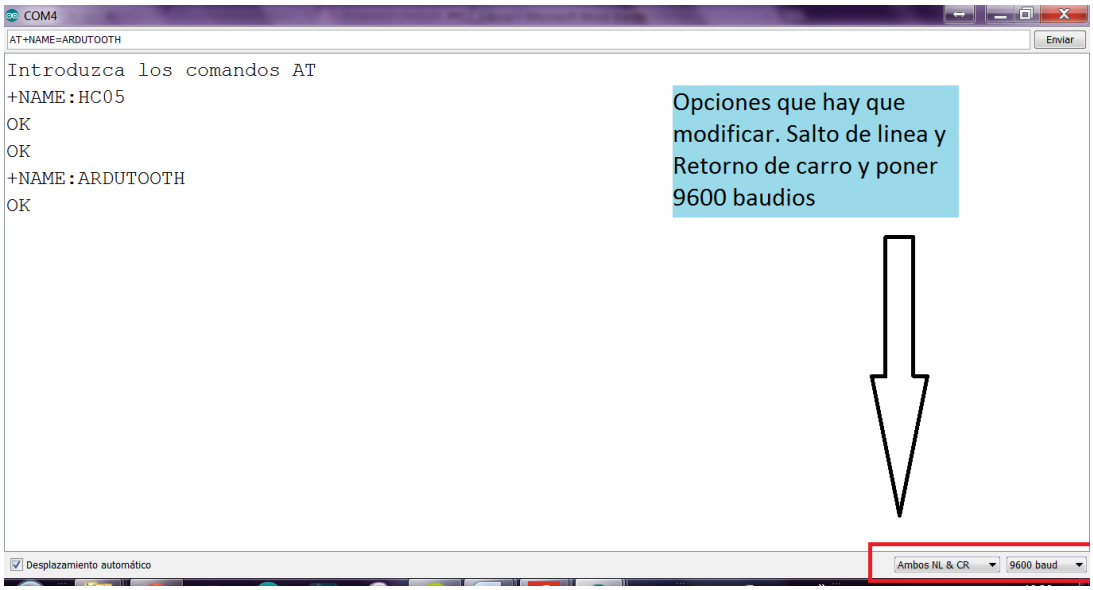
-**AT+ROLE:** Nos devuelve el rol del módulo que nos indicara la forma de trabajar que tiene. Si nos devuelve “0” indicara que está en modo esclavo (SLAVE), en este modo el HC-05 está a la espera de que otro dispositivo se conecte a él. Si nos devuelve “1” estará en modo maestro (MASTER), en este modo es el propio HC-05 el que se conecta a otro dispositivo. Y si nos devuelve “2” indicara que está en modo Slave-loop. El que usaremos para este proyecto es el modo esclavo.

AT+ROLE=0

-**AT+UART:** Obtendremos una respuesta del tipo “+UART:9600,0,0” estos números nos indican: el primero los baudios de trasmisión a los que trabaja el modulo, el segundo el bit de stop y el tercero el bit de paridad. Para modificarlo deberíamos escribir el comando AT+UART=”nuevos parámetros”, pero lo dejaremos como está.

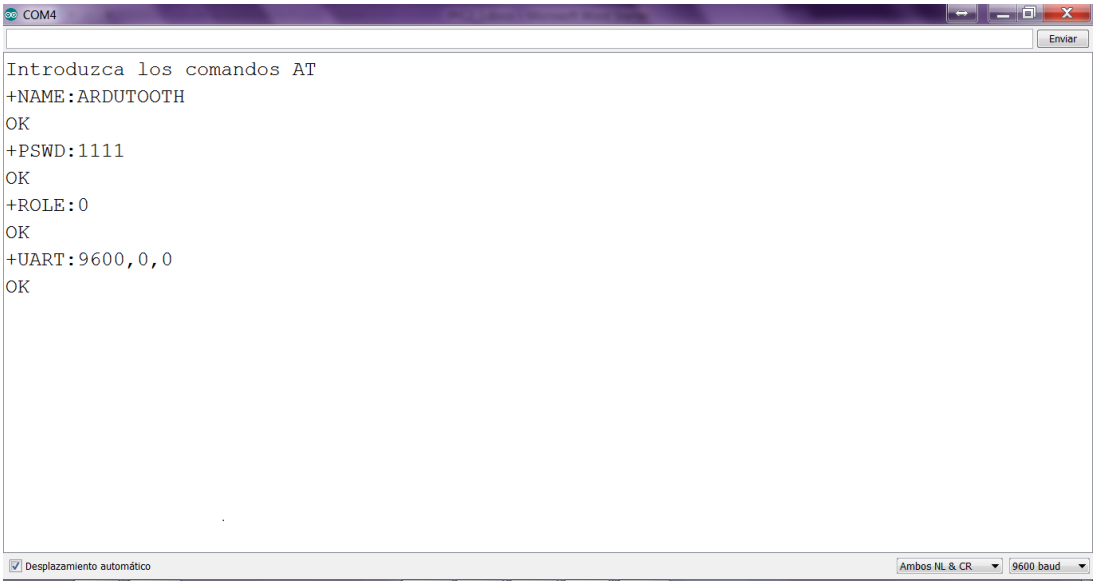
-**AT+VERSION:** Nos devuelve la versión que corre el HC-05.

En esta imagen se muestra como se ha modificado el nombre del módulo Bluetooth, primero se llamaba “HC05” y lo hemos modificado a “ARDUTOOTH”.



Modificación del nombre

En la siguiente imagen se mostraría como se quedan los valores para que funcione correctamente el modulo Bluetooth HC-05 en este proyecto:

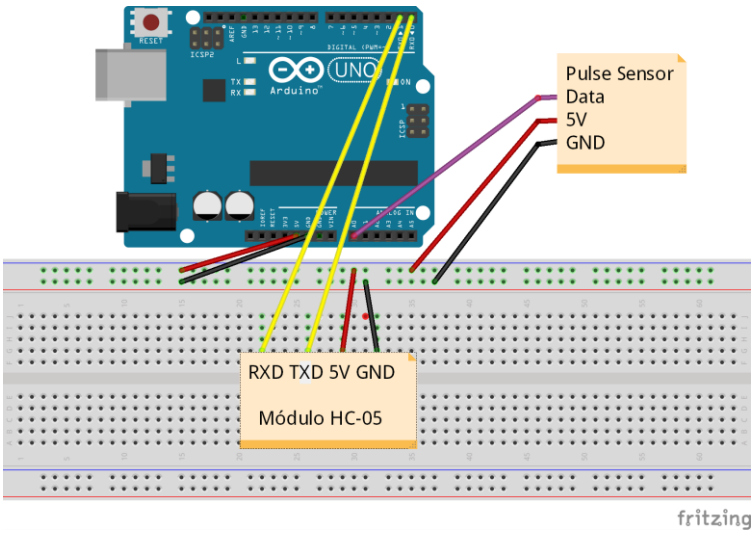


Captura de pantalla del Monitor Serial después de introducir los comandos de configuración.

Conjunto Arduino, Pulse Sensor y módulo Bluetooth

Descritos los tres dispositivos anteriores, pasaremos a explicar el funcionamiento cuando trabajan en conjunto. Este apartado es esencial porque constituye una de las dos grandes partes que conforman el proyecto, la otra seria el apartado de la aplicación en Android.

Conexionado y funcionamiento



En la imagen se muestran las conexiones establecidas entre cada uno de los pines de los distintos dispositivos. Vamos a detallarlas:

-Alimentación: La placa Arduino, como bien hemos explicado en su apartado, se puede alimentar de varias maneras, nosotros hemos elegido una batería externa de 9V. Tanto el Pulse Sensor como el módulo HC-05, para alimentarse se conectarán al pin de 5V y al GND de la placa Arduino. Para concentrar estas conexiones hemos llevado el pin 5V de Arduino mediante un cable macho-macho a una placa protoboard, y desde aquí ya conectar la alimentación de los dispositivos. Lo mismo hemos hecho con el pin GND, lo hemos llevado a una placa protoboard y ahí ya hemos conectado los pines de masa tanto del Pulse Sensor como del módulo Bluetooth.

-Datos: El cable por donde viajan los datos captados por el Pulse Sensor debe ir conectado a la entrada analógica A0 de la placa Arduino. El apartado de la conexión de los pines de datos entre Arduino y el HC-05 es importante. El pin 0RX de la placa Arduino debe ir conectado al pin TXD del Módulo Bluetooth y el pin 1TX de Arduino al pin RXD del módulo Bluetooth. Así, cualquier comando escrito desde el Monitor Serial del IDE de Arduino mientras se está corriendo el sketch y conectado al ordenador o

cualquier texto que escribamos con la función print(), saldrá por el pin 1TX de la placa Arduino, entrará por el pin RXD del módulo HC-05 y se enviará directamente vía Bluetooth al otro dispositivo que esté conectado. Y viceversa, cualquier comando que recibamos vía Bluetooth desde el dispositivo al cual estamos conectados, saldrá por el pin TXD del módulo HC-05 y entrará por el pin 0RX de la placa Arduino, y se podrá leer dicho comando con la función Serial.read().

Una vez este todo conectado correctamente, solo tendremos que hacer una pequeña modificación en el sketch de Arduino para que se envíen al móvil solo los datos que queremos. Al igual que cuando hemos calculado el error inherente del Pulse sensor y solo queríamos el valor de los intervalos entre latidos, pues en este caso solo queremos enviarle a la aplicación Android el valor instantáneo de los BPM, que será lo único que mostrará en pantalla. Para ello comentaremos las sentencias sendDataToProcessing() de los valores Signal e IBI, que en realidad solo indican que se muestren los valores por pantalla, y solo dejaremos los BPM. Recordar que se debe enviar solo el valor sin la letra indicativa. Así los valores serán enviados por el puerto serie y a través del pin 1TX de la placa Arduino llegaran al pin RXD del módulo HC-05 y se enviaran directamente vía Bluetooth al móvil.

Y por último una cosa importante, después de la modificación anterior, asegurarnos de que a la hora de enviar los datos de los BPM por el puerto serie se haga con la función println() que añade un salto de línea al final del texto enviado. Esto será crucial a la hora de decodificar el texto desde Android.

Android

Vamos a pasar a explicar qué es el sistema operativo Android y cómo se desarrollan y funcionan sus aplicaciones. Se va a explicar esta parte porque en el proyecto se va a crear una aplicación para el móvil que reciba vía Bluetooth a través del módulo HC-05, los BPM calculados por Arduino gracias al Pulse Sensor.

Android es un sistema operativo basado en Linux. Fue diseñado principalmente para teléfonos móviles, pero en la actualidad este SO corre en todo tipo de dispositivos, como relojes inteligentes, tablets, televisores o incluso automóviles.

Las aplicaciones en Android están escritas en código de programación Java, pero el sistema operativo no interpreta directamente este código porque no tiene una máquina virtual Java integrada en el. Lo que se hace es compilar el bytecode en un ejecutable Dalvik para que corra en la Máquina Virtual Dalvik, que fue creada específicamente para Android y esta optimizada para dispositivos móviles con batería, y memoria y procesador limitados.

Eclipse: el IDE de Android

El entorno de desarrollo usado para desarrollar las aplicaciones es Eclipse. Para poder desarrollar aplicaciones para el sistema operativo Android nos hará falta el plug-in de Android Developers Tools (ADT). Que lo podemos instalar clicando en: Help>Install new Software y añadiendo la dirección:

https://dl-ssl.google.com/android/eclipse/

desde la cual se instalara el nuevo software.

Una vez instalado este plug-in en Ecplise nos aparecerán dos apartados nuevos el Software Development Kit (SDK Manager) y el Android Virtual Devices (AVD).

-SDK Manager: Se usa para gestionar las descargas de las APIs de Android, que son todas las librerías de las diferentes versiones que van saliendo del Sistema Operativo. Con cada nueva versión se incluyen más novedades y funcionalidades que los desarrolladores pueden usar para sus aplicaciones.

-AVD: Se usa para crear dispositivos virtuales con ciertas características y poder correr las aplicaciones en ellos. Es decir, puedes probar como funcionaria tu aplicación en un terminal con cierto tamaño de pantalla, cierta capacidad de memoria RAM, etc.

Cuando ya tengamos Eclipse con el plug-in ADT instalado y las APIs descargadas a través del SDK manager ya estamos listos para crear un nuevo proyecto para Android.

Partes del proyecto y "Hola Mundo"

Para crear una aplicación seleccionamos: File>Project>Android Application Project. A la hora de crear el proyecto hay que tener en cuenta la API o versión que escogemos para crearlo, cuanto más alta sea la versión de Android más funcionalidades podremos incorporar en nuestra aplicación pero tendrá menos compatibilidad con los dispositivos que hay en el mercado ya que todos los usuarios no disponen del ultimo terminal que ha salido con la última versión de Android. Como esta aplicación va a ser sencilla escogeremos la API 8 que coincide con la versión de Android 2.2 Froyo, y la que nos proporcionara de sobra las funciones para llevar a cabo el proyecto y además será compatible con todos los teléfonos del momento.

Una vez creado el proyecto vemos como se estructura en diferentes partes, vamos a explicar las más importantes:

-Layouts: Dentro de esta carpeta se encuentran las Interfaces de Usuario o User Interface(UI). Son archivos .xml que serán todas las vistas y páginas que conformaran la aplicación, todos los botones, listas, cuadros de texto, etc. que verá el usuario y con los que podrá interactuar. A cada una de estas pantallas se le llamará Activity.

-MainActivity.java: Es el archivo donde se encuentra el verdadero código de funcionamiento de la aplicación, el código Java.

-AndroidManifest: Es el archivo donde se describen todas las características generales: nombre y versión de la aplicación, versión de compilación, permisos que usa,etc.

-R. java: Es un archivo donde se almacenan todas las variables de todo tipo que se van generando: int, double, boolean, string, char,etc

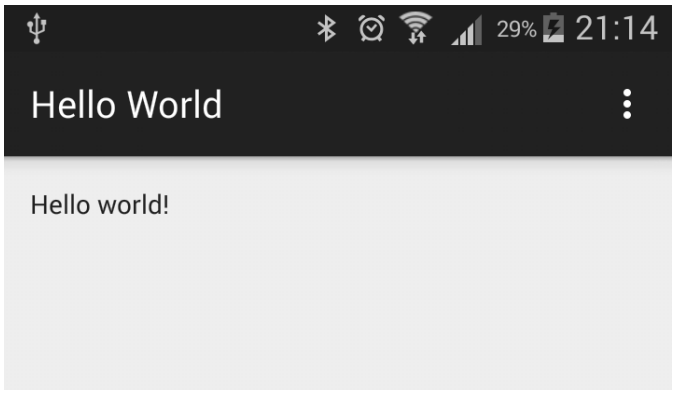
-Drawable: Aquí se guardaran todas las imágenes usadas en la aplicación.

-Raw: Aquí se guardan todos los sonidos usados en la aplicación.

-Values: Aquí se almacenan variables estáticas, es útil para variables que se usan muchas veces a lo largo del desarrollo de la aplicación, así si hay que modificar esa variable, en lugar de ir a todos los sitios donde la has usado, la cambias solo una vez desde Values.

Para probar si funciona todo y podemos empezar a desarrollar la aplicación tenemos que hacer una primera prueba creando por ejemplo un 'Hola Mundo' y abriéndola en un dispositivo. Aunque haya dispositivos en el AVD, yo he hecho todas las pruebas de las aplicaciones con mi teléfono móvil, para ello lo he conectado con un cable USB al ordenador y he activado la depuración USB en el móvil y habilitado la opción de instalación de aplicaciones desconocidas, desde opciones de desarrollador. Una vez

creada la aplicación de prueba y corrida con los comandos Run> Android Application vemos que todo funciona perfectamente

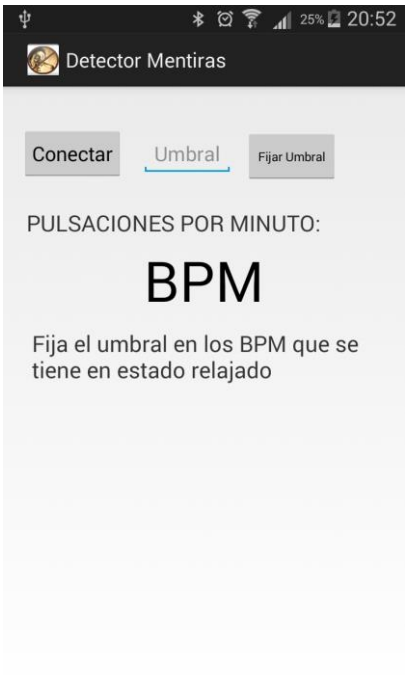


Detector de mentiras en Android

Vamos a pasar a la explicación de la aplicación. Primero vamos a recordar cual era la idea principal de esta aplicación y después explicaremos detalladamente como hemos llevado a cabo su funcionamiento.

Lo que va a poder realizar esta aplicación va a ser emparejarse vía Bluetooth con el módulo HC-05 que está conectado a la placa Arduino y poder recibir las pulsaciones calculadas por el Pulse Sensor que engancharemos a nuestro dedo. Una vez reciba las pulsaciones introduciremos nosotros manualmente las pulsaciones que tenemos en estado relajado. Esto definirá un umbral para que cuando se sobrepase en 10 pulsaciones por minuto más que en estado relajado, avisarnos de que la persona se ha puesto nerviosa y por lo tanto ha dicho una mentira.

Interface de usuario



Esta es la vista que se tiene nada más abres la aplicación. Partes que formas la Interface de Usuario:

-Botón 'Conectar': es el que pulsaremos para conectar el móvil con el modulo Bluetooth. Una vez nos hayamos conectado el botón se deshabilitará. Código XML:

```
<Button
        android:id="@+id/btConnect"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Conectar"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="10dp"
        android:padding="10dp"/>
```

-Edit Text 'Umbral': Es un recuadro en el que podemos insertar texto. Nos servirá para introducir las pulsaciones que se tiene en estado relajado. Para hacerlo más intuitivo hemos hecho uso de un "hint" que es una palabra indicativa que aparece en el cuadro de texto en un color sombreado, también, como solo vamos a introducir números hemos hecho que cuando se pulse sobre el cuadro de texto aparezca un teclado numérico de 3x4. Código XML:

```
<EditText
         android:id="@+id/ETUmbral"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Umbral"
        android:numeric="integer"
        android:inputType="number"
        android:layout_alignBaseline="@id/btConnect"
        android:layout_toRightOf="@id/btConnect"
        android:layout_marginLeft="15dp"/>
```

-Botón 'Fijar Umbral': Botón usado para guardar el umbral de las pulsaciones por minuto en estado relajado. Código XML:

```
<Button
       android:id="@+id/BTUmnral"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fijar Umbral"
        android:textSize="11dp"
        android:layout_alignBaseline="@id/btConnect"
        android:layout_toRightOf="@id/ETUmbral"
        android:layout_marginLeft="10dp"/>
```

-Text View 'Pulsaciones por minuto': es un cuadro en el que solo aparece ese texto. Código XML:

```
<TextView
        android:id="@+id/TVbpm"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
```

```
        android:text="PULSACIONES POR MINUTO:"
        android:textSize="20dp"
        android:layout_below="@id/btConnect"
        android:padding="5dp"
        android:layout_marginTop="10dp"/>
```

-Text View 'Numero de BPM': Aquí será donde irán apareciendo las pulsaciones por minuto que vayamos recibiendo de Arduino. Primero aparecerá "BPM" que es un texto indicativo y nada más nos conectemos a Arduino irán apareciendo los números indicando los BPM que tenemos. Código XML:

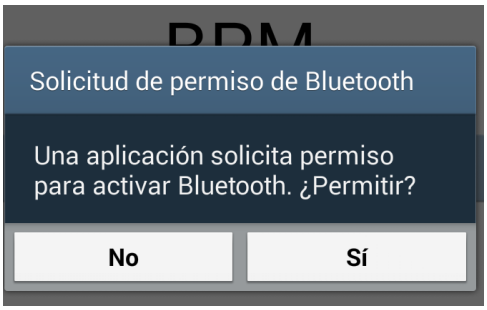
```
<TextView
        android:id="@+id/TVbpmNumber"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="BPM"
        android:textSize="50dp"
        android:textColor="@android:color/black"
        android:layout_below="@id/TVbpm"
        android:gravity="center"/>
```

-ImageView 'Imagen Veredicto': Es un recuadro en el que se pueden colocar imágenes. Primero hay un texto explicativo para que el usuario sepa qué tiene que introducir en el recuadro de umbral, pero luego incluiremos unas imágenes de "Verdad" o "Mentira". Código XML:

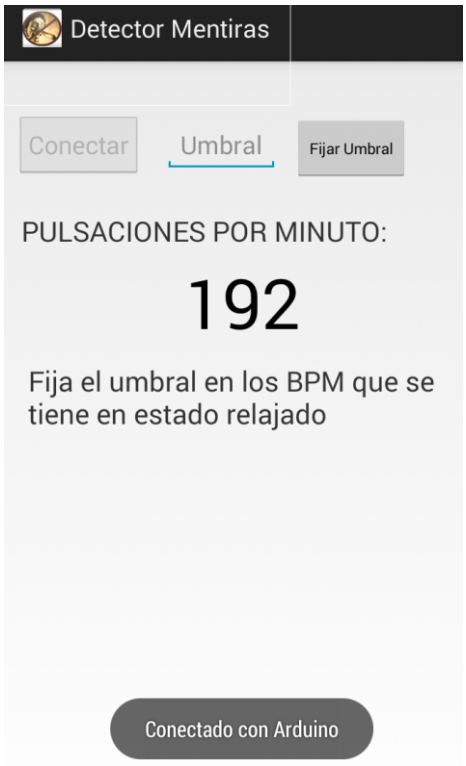
```
<ImageView
        android:id="@+id/IVVeredicto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@id/TVbpmNumber"
        android:padding="5dp"
        />

    <TextView
        android:id="@+id/TVdeIV"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fija el umbral en los BPM que se tiene en estado
relajado"
        android:textSize="20sp"
        android:layout_alignLeft="@id/IVVeredicto"
        android:layout_alignRight="@id/IVVeredicto"
        android:layout_alignTop="@id/IVVeredicto"
        android:layout_alignBottom="@id/IVVeredicto"
        android:padding="10dp"
        />
```

Otras vistas que aparecen en la aplicación son Dialogs y Toasts, pero estas se generan desde el apartado de Java.

-Dialog de conexión Bluetooth: Si se abre la aplicación y la opción de Buetooth esta deshabilitada, saltara un cuadro advirtiendo de que la aplicación pide permiso para conectarse al Bluetooth y si deseamos activarlo. En el caso de decirle que no y luego pulsamos el botón Conectar, nos volverá a salir este Dialog.



-Toasts: Los Toasts son cuadros de texto, normalmente informativos, que aparecen en la pantalla durante un corto periodo de tiempo. Hemos puesto Toasts por ejemplo cuando intentas conectarte y no se encuentra el modulo Bluetooth o cuando se ha establecido la conexión correctamente.



Código Java de la aplicación

Antes de empezar hay que dotar a la aplicación con los permisos específicos para su correcto funcionamiento. Estos son los permisos para poder acceder al Bluetooth del dispositivo y todo lo que ello conlleva: conexión, envío de información, etc. Para ello nos dirigiremos al AndroidManifest.xml y en el apartado de Permissions

añadiremos los tres permisos de Bluetooth: BLUETOOTH, BLUETOOTH_ADMIN, BLUETOOTH_PRIVILEGED.

Para empezar en el **OnCreate()** hemos hecho que si el dispositivo móvil no tiene Bluetooth directamente se cierre la aplicación porque no la va a poder utilizar, y si tiene Bluetooth pues que salte el Dialog que habíamos mencionando anteriormente pidiendo permisos para habilitar el Bluetooth del móvil. Por lo general también se crean las variables y se asignan cada una de ellas a una variable de la Interfaz de Usuario para que estén interrelacionadas, pero al ser esto igual en todos los proyectos Android y no específico del nuestro no vamos a entrar en más detalles.

La función **Conectar()** se ejecutará cuando se pulse el botón conectar. El comportamiento de esta función es el siguiente:

Primero se asegura de que el Bluetooth del dispositivo está habilitado, en el caso de no estarlo vuelve a abrir el Dialog pidiendo permisos para habilitarlo. Cuando ya está disponible el Bluetooth hace un escaneo de todos los dispositivos que tiene alrededor y les guarda en un array llamado Devices, una vez encontrados todos le pedimos que se conecte al que tenga la dirección MAC especifica de nuestro módulo HC-05, es decir, la que previamente hemos sacado con el comando AT+ADDR. Una vez encontrado el módulo Bluetooth correcto se creará un Socket entre ellos dos, que no es más que una pasarela por donde viajará la información. Cuando se ha generado el Socket se crea un InputStream, que es una puerta de entrada por donde la aplicación recibirá los datos enviados desde Arduino. Después de crear el InputStream se llama a la función StartDatalistener que lo que hará será abrir un hilo secundario o Thread llamado DataListener (escuchador de información), este Thread lo que hace es estar continuamente comprobando si entran nuevos datos por el InputStream. Una vez creado el Socket, abierto el InputStream y llamado a la función StartDataListener la conexión se ha establecido correctamente, mostraremos un Toast informando de ello y deshabilitaremos el botón Conectar ya que una vez conectados no nos va a servir. Todos estos pasos se realizan con el conjunto try/catch para que si fallara alguno de ellos se guardara el mensaje de error en el LogCat y nos informara con un Toast en la aplicación indicando en que paso se ha producido el error. Esto último no lo vamos a repetir porque va a ser igual para todo el proceso de desarrollo de la aplicación.

Este es el código de la función:

```
	public void conectar(){
		try {
			if(mBluetoothAdapter.isEnabled()){
				mBluetoothAdapter.startDiscovery();
	Set<BluetoothDevice> devices = mBluetoothAdapter.getBondedDevices();

				for(BluetoothDevice device : devices)

					address = "98:D3:31:B1:F1:55";//Direccion MAC
					del modulo Bluetooth HC-05
mBluetoothDevice = mBluetoothAdapter.getRemoteDevice(address);

					BluetoothSocket tmp = null;
```

```
                                try {
Method m = mBluetoothDevice.getClass().getMethod("createRfcommSocket", new
Class[] {int.class});

tmp = (BluetoothSocket) m.invoke(mBluetoothDevice, Integer.valueOf(1));

                                } catch (Exception e) {
                                msg("La conexion del socket tmp no ha
                                funcionado");

                                }
                                mBluetoothSocket = tmp;

                                try {
                                      mBluetoothSocket.connect();

                                } catch (Exception e) {
                                      e.printStackTrace();
                                      msg("No se ha podido crear el socket
                                      de Bluetooth");
                                }
                                //CREAR EL THREAD Y EL INPUTSTREAM
                                try {
                          myInStream = mBluetoothSocket.getInputStream();

                                } catch (Exception e) {
                                msg("Error:" + e.getMessage());

                                }
                                      startdataListener();

                                      msg("Conectado con Arduino");
                                      btConnect.setEnabled(false);
                                      conectado = true;
                                }
            } catch (Exception e) {
                  msg("No se ha podido conectar con arduino");
                  e.printStackTrace();
            }
      }
```

La función **StartDataListener** es la que ejecutará el thread **DataListener** que es el encargado de leer la información recibida desde Arduino a través del Socket y el InputStream, decodificarla y entenderla, ya que se envía a través de paquetes de bytes y, una vez tengamos los BPM registrados por el pulse Sensor, saber si sobrepasan el umbral definido y así advertirnos de si se está diciendo una verdad o un mentira. El funcionamiento es el siguiente:

Mientras no haya una interrupción en el thread se comprueba si el InputStream está disponible, si lo está empieza a guardar en un buffer la información que se va recibiendo. Este buffer es un contenedor o almacenador temporal. Cuando haya información disponible en el buffer empezará a leerla y a decodificarla. Importante, la información enviada desde Arduino hacia Android es el valor en número de los BPM

pero en lugar de enviar el valor en número se enviará su valor equivalente de la tabla ASCII codificado en formato decimal. Así que si se envía que el valor de los BPM es 60, en lugar de enviar el 6 y el 0 se enviarán sus equivalentes ASCII en decimal que son el 54 y el 48 y además en paquetes de bytes. Como se están recibiendo continuamente el valor de los BPM instantáneos, para poder separarlos y hacer cada vez una lectura correcta de solo un valor de BPM lo que haremos será separarlos entre sí con un salto de línea. Así estaremos leyendo los datos que recibimos desde Arduino hasta que detectemos un salto de línea, que corresponderá con el valor 10 en decimal de la tabla ASCII. Cuando hayamos detectado este 10 o salto de línea sabremos que la información que tenemos corresponde con un valor de los BPM. La única modificación que tendremos que hacer desde Arduino es que cuando le indiquemos que envíe por puerto serie el valor de los BPM hay que hacerlo con el comando **println()**, para que imprima por pantalla el valor y añada ese salto de línea. Cuando tengamos ya el valor de los BPM decodificado, solo tendremos que cambiar el valor del cuadro TextView llamado TVNumberBPM por el valor recibido.

Dentro de este thread también se encuentra el detector de mentiras que lo que hace es coger el valor recibido de los BPM y compararlo con el valor del umbral, que recordemos que eran las pulsaciones por minuto que teníamos en estado relajado. Si el valor de los BPM supera en 10 el valor del umbral fijado será consecuencia de que la persona a la que estamos midiendo las pulsaciones se ha puesto nerviosa y le han subido bruscamente. Gracias a esto detectaremos que la persona ha dicho una mentira y haremos que en la aplicación aparezca una imagen con la palabra"MENTIRA" y haya una señal acústica que nos advierta de ello. Cuando las pulsaciones estén por debajo del valor umbral+10 aparecerá una imagen con la palabra "CORRECTO" acompañada de una señal acústica, indicando que la persona se ha vuelto a relajar y por lo tanto dice la verdad.

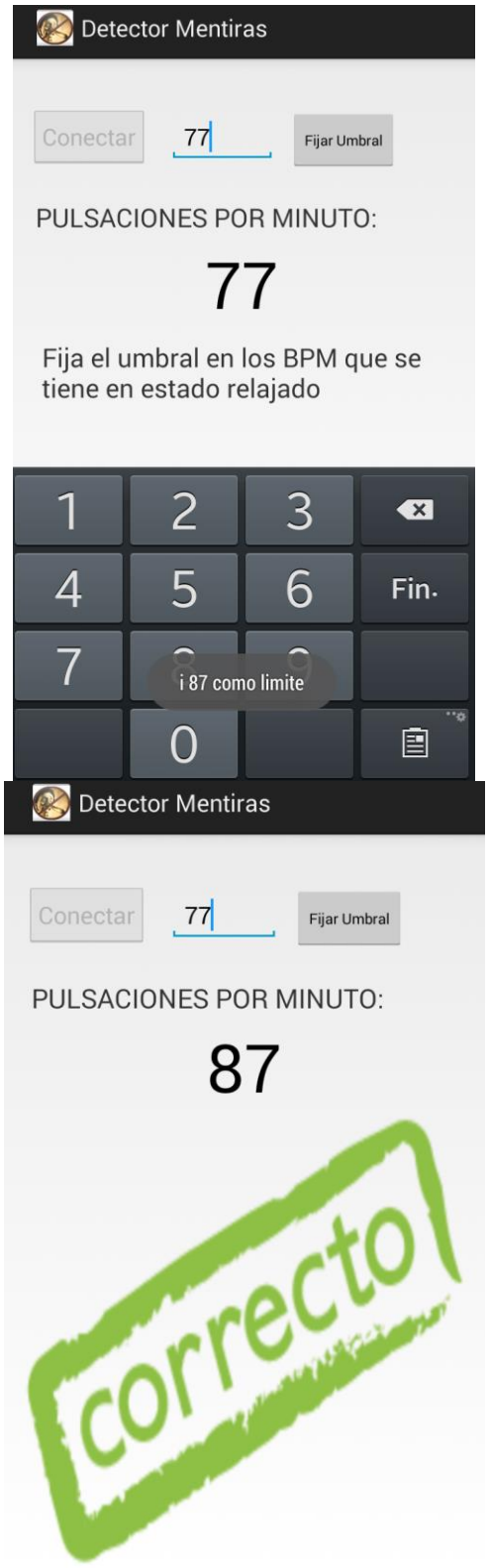
El código es este:

```
void startdataListener(){
        final Handler handler = new Handler();
        final byte delimiter = 10;//Codigo ASCII de nueva linea

        stopWorker = false;
        readBufferPosition = 0;
        readBuffer = new byte[1024];
        workerThread = new Thread(new Runnable() {

        @Override
        public void run() {
        while(!Thread.currentThread().isInterrupted() && !stopWorker){
                try {
                int bytesAvailable = myInStream.available();
                if(bytesAvailable > 0){
                        byte[] packetBytes = new byte[bytesAvailable];
                        myInStream.read(packetBytes);
                                for(int i=0; i<bytesAvailable; i++){
                                byte b = packetBytes[i];
```

```
                                        if(b == delimiter);
                                    byte[] encodedBytes = new byte[readBufferPosition];

        System.arraycopy(readBuffer, 0, encodedBytes, 0, encodedBytes.length);
                final String data0 = new String(encodedBytes); //"US-ASCII");

final String data = data0.substring(0, data0.length()-1); //"US-ASCII");


                                                  readBufferPosition = 0;

                                                    handler.post(new Runnable() {
                                                    public void run() {

                                                    TVbpmNumber.setText(data);


                                          String msg;

                        if(intUmbral!=-1){
                        TVdeIV.setText("");

                              try {
                               intData = Integer.parseInt(data);

                              if(intData<=umbralLimite){
                          IVVeredicto.setBackgroundResource(R.drawable.correcto);

                              if(mentira == true){
                              soundVerdad.start();
                               mentira=false;
                               }
                               }else if(intData>umbralLimite){
                        IVVeredicto.setBackgroundResource(R.drawable.mentira);

                               if(mentira == false){
                               soundMentira.start();
                               mentira=true;
                              }
                         }

                        } catch (Exception e) {

                        msg  = e.getMessage();
                        }
                      }
                    }
                  });
               }else{
               readBuffer[readBufferPosition++] = b;
                    }                                                  }
                 }
              } catch (Exception e) {
                stopWorker = true;
               }
            }//while
          }
        });//threa
        workerThread.start();
```

```
}
```

Y estas son las imágenes de las diferentes pantallas que va teniendo la aplicación:


Detector Mentiras
Conectar
77
Fijar Umbral
PULSACIONES POR MINUTO:
77
Fija el umbral en los BPM que se tiene en estado relajado
1
2
3
4
5
6
Fin.
7
0
i 87 como limite
Detector Mentiras
Conectar
77
Fijar Umbral
PULSACIONES POR MINUTO:
87
correcto

Para terminar con la explicación de la aplicación en Android comentar que para que haya una buena comunicación Bluetooth entre ambos dispositivos, los dos tienen que trabajar en la misma tasa binaria para que haya una correcta coordinación entre la trasmisión y la recepción de la información. Antes de realizar esta aplicación se creó una en la que había comunicación bidireccional entre Android y Arduino, cada dispositivo podía tanto enviar información escrita como recibirla y decodificarla, y uno de los problemas mayores que tuvimos, que no por ello más complejo porque al final era una cosa súper sencilla, fue averiguar la tasa binaria en la que Arduino tenía que leer la información. Los bits sí que se recibían pero al Android enviarlo más rápido de lo que Arduino leía, no se decodificaban los caracteres correctamente y aparecían símbolos del tipo "◊○&%Δς". Al final trabajando a una tasa binaria de 9600 bits/s se consiguió la correcta comunicación, así que es una cosa importante a tener en cuenta.

Desde aquí se puede descargar el proyecto "Detector de mentiras" con el código entero de la aplicación para Android. Solo se tendrá que descargar y abrir desde Eclipse.

Conclusiones

Vamos a realizar un análisis crítico del trabajo realizado con sus correspondientes conclusiones.

El trabajo en general abarca bastantes materias:

- Tenemos la electrónica de Arduino, hemos tenido que crear circuitos con los dispositivos que teníamos y conectar bien todos sus pines, ya fueran de datos, de alimentación o de configuración, para que todo funcionase correctamente.
- Por otro lado tenemos la física de la radiación y absorción de luz que la hemos aplicado en el Pulse Sensor para la medición de las variaciones de cantidad de sangre en la punta del dedo.
- También tenemos toda la teoría sobre el ritmo cardiaco instantáneo, las ondas que forman los electrocardiogramas con todas sus partes explicadas y cómo a partir de estas se calculan los BPM.
- Hemos tocado también la comunicación Bluetooth con todo lo que ello conlleva: vinculación y comunicación entre dispositivos a través de Sockets, direcciones MAC, asignaciones de baudios para la correcta comunicación, codificación en la emisión y decodificación en la recepción de la información enviada, etc.

Y por supuesto que tenemos la programación. Hemos creado dos aplicaciones para dos dispositivos distintos, con dos entornos de desarrollo distintos y dos lenguajes de programación distintos. Lo que quiero decir con esto es que para que el proyecto funcione como es debido, se han de entender a la perfección todas las características de cada una de las partes que lo conforman y saber en cada momento que es lo que se está haciendo y porque.

El inconveniente de este proyecto es que está dividido en varias partes y por eso tienen que estar todos los parámetros de todas las configuraciones posibles que se pueden modificar en cada uno de los dispositivos, correctamente definidos. Con fallar solo uno de esos parámetros, el problema se arrastrará a los demás y el resultado final no funcionará correctamente.

Sí que es verdad, y en este punto voy a ser crítico conmigo mismo, que la aplicación en Android podría estar mejor desarrollada. Uno de los aspectos a mejorar seria a la hora de conectarse por Bluetooth con el módulo HC-05. En la aplicación creada para este proyecto tenemos el inconveniente de que para que el teléfono móvil sepa a cual módulo Bluetooth conectarse, tenemos que introducir manualmente en el código Java su dirección MAC física. Esto se podría solucionar cuando una vez escaneados los dispositivos Bluetooth que el móvil tiene a su alrededor, aparezca una lista en la pantalla del teléfono con todos ellos y seleccionar al cual queremos conectarnos. Así se podría estandarizar y utilizar para cualquier teléfono móvil y cualquier módulo HC-05.

Lo que también se debería perfeccionar es la forma de detectar la mentira. En mi opinión no basta con que se sobrepase en 10 las pulsaciones por minuto que se tenía en estado relajado, más que nada porque una persona puede contestar con una mentira a una pregunta y mantener las mismas pulsaciones. Lo que sí que está bien es

que esas pulsaciones en estado relajado las podamos introducir nosotros manualmente desde la interface de la aplicación ya que en cada persona son diferentes. Al principio pensamos en añadir, a parte del Pulse Sensor, un sensor de humedad, para detectar el exceso de sudoración de la persona en el caso de decir una mentira. Pero esta opción la descartamos ya que en el supuesto caso de que la persona sudara un poco más, esta mínima variación no se podría detectar con los sensores de humedad que hay hoy en día disponibles para Arduino.

En conclusión, el proyecto en sí no va a ser el método más eficiente para llevar a cabo la tarea para la cual ha sido creado, que es detectar mentiras. Pero sí que nos ha servido para conocer a fondo todos los detalles sobre la creación de proyectos en Arduino, el desarrollo de una aplicación para Android y como combinar estas dos plataformas para que interactúen entre sí.
Yo creo que lo principal ha sido eso, la creación de dos aplicaciones de dos plataformas distintas que se puedan comunicar y entender entre sí a la perfección. Una vez creada esta base lo que hagas después, ya sea enviar los BPM calculados por un sensor de ritmo cardiaco, enviar si un LED esta encendido o apagado o activar el relé para que se encienda un ventilador; eso ya es lo de menos.

Bibliografía

He tenido que acceder a muchos sitios web para buscar información para la realización de cada uno de los pasos que iba realizando y sobre todo he tenido que buscar y leer mucho para solucionar cada uno de los problemas que me iban surgiendo a lo largo del desarrollo del proyecto.

Las páginas web principales a las que he recurrido a la hora de buscar información han sido:

-La página web oficial de Arduino sacando la información de sus diferentes secciones y de los foros internos construidos por desarrolladores.

-Página oficial de desarrolladores de Android.

-Wikipedia.

- Youtube. Esta plataforma de videos online me ha servido para ver diferentes tutoriales, tanto para la creación de la parte de Arduino como de Android.

-El libro "El gran libro de Android" de Jesús Tomás Gironés.

-El libro "Desarrollo de aplicaciones para Android" de Joan Ribas Lequerica, que por cierto, fue el crucial para la correcta decodificación de la información recibida en Android.

Internet. Parece obvio, pero no solo he recurrido a los sitios descritos anteriormente, he recurrido a un sinfín de foros, páginas web y blogs para obtener información, y citarlos todos sería imposible.