CCADET

CENTRO DE CIENCIAS APLICADAS Y DESARROLLO TECNOLÓGICO
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Informe Técnico

TUTORIAL SOBRE ARDUINO

Miguel Angel Bañuelos Saucedo

Abril 2017

Desarrollo tecnológico

Trabajo realizado con el apoyo del Programa UNAM-DGAPA-PAPIME-PE106816

Financiamiento externo

Título: TUTORIAL SOBRE ARDUINO

Autores: **Miguel Angel Bañuelos Saucedo**

Afiliación: Centro de Ciencias Aplicadas y Desarrollo Tecnológico

Resumen

En este reporte se presenta una introducción al uso de la tarjeta Arduino UNO. Se da una breve explicación de cómo utilizar el ambiente de desarrollo del Arduino para editar, compilar programas y cargarlos en la tarjeta Arduino. Se incluye una descripción de los circuitos electrónicos que se conectarán a la tarjeta Arduino y del uso de una tarjeta prototipo. Adicionalmente, se comentan algunos detalles del lenguaje de programación del Arduino y se dan algunos ejemplos de aplicación que incluyen la comunicación por puerto serial con una computadora personal, la adquisición de datos, el manejo de diversos sensores y sus bibliotecas de funciones. Por último, se muestra un ejemplo de construcción de un cronómetro fotoeléctrico.

Índice

Nomenclatura

Símbolo	*Significado*
CD	*Corriente Directa*
COM	*Se utiliza para enumerar los puertos de comunicación serial*
IDE	*Integrated Development Environment*
LED	*Light Emitting Diode*
PWM	*Pulse Width Modulation*
RX	*Se utiliza para identificar la línea de recepción de comunicación serial*
TX	*Se utiliza para identificar la línea de transmisión de comunicación serial*
USB	*Universal Serial Bus*

Introducción

El sistema Arduino se ha convertido en una herramienta popular para desarrollar diversos proyectos electrónicos, y para introducir al usuario al uso de microcontroladores. Existe una gran cantidad de información sobre esta plataforma en internet o en libros técnicos; sin embargo, esta misma abundancia dificulta en ocasiones encontrar los datos que se requieren para comenzar a utilizar rápidamente una tarjeta Arduino. Decidimos elaborar este breve tutorial, con la intención de invitar al lector a utilizar esta herramienta electrónica, mediante una introducción sencilla a su operación. Los contenidos fueron seleccionados después de haber impartido un curso introductorio al sistema Arduino en junio de 2015. Se espera que el lector esté familiarizado con los sistemas de numeración binaria y tenga nociones de programación. De igual manera, es conveniente tener experiencia en la interpretación de diagramas de circuitos eléctricos.

1 LA TARJETA ARDUINO

El sistema Arduino consiste en una tarjeta electrónica y un ambiente de programación. Arduino está formado por una familia de tarjetas con diferentes capacidades y costos. Arduino UNO es una tarjeta de gama baja, adecuada para proyectos sencillos y para un primer contacto con este tipo de sistemas. Esta tarjeta cuenta con un microcontrolador de la compañía Atmel (ver Fig. 1), el cual es el dispositivo encargado de almacenar y ejecutar el programa desarrollado por el usuario. Además, la tarjeta cuenta con los componentes necesarios para la operación del microcontrolador, entre ellos un circuito para alimentación de voltaje, así como un puerto USB (que se utiliza para la programación desde una computadora personal) y terminales de conexión para conectarlo a sensores, motores, LEDs, etc.

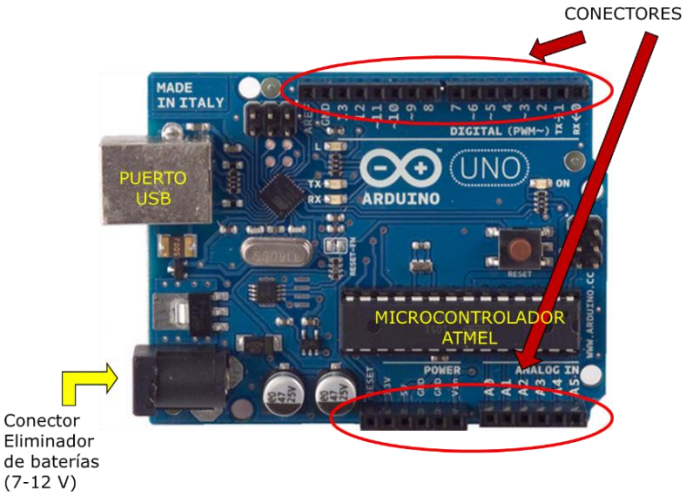


Fig. 1 Fotografía de la tarjeta Arduino UNO donde se señalan los componentes principales.

La tarjeta Arduino cuenta con una entrada para conectar un eliminador de baterías que entregue un voltaje entre 7 y 12 Volts.

El puerto USB sirve para conectar la tarjeta Arduino a una computadora personal y desde ahí programarla utilizando el lenguaje propio del Arduino. Este programa (Arduino Software IDE) es gratuito y se puede descargar de la dirección electrónica www.arduino.cc . Existen versiones para Windows, Linux y Mac OS X. En el caso de Windows se recomienda descargar la versión Windows Installer. El proceso de instalación se describe en el Apéndice A. El puerto USB también le proporciona voltaje a la tarjeta Arduino, en cuyo caso no es necesario utilizar el eliminador de baterías. Otra opción es utilizar la conexión etiquetada V_{in}, por donde también se puede suministrar un voltaje de 7 a 12 V de corriente directa.

Los programas para el Arduino reciben el nombre de Sketch. Cuando se crea un nuevo programa, el Arduino IDE incluye los bloques básicos del programa: *setup* y *loop* (ver Fig. 2 Ventana del Arduino IDE que muestra los bloques de programación básicos enmarcados en rojo.). El bloque *setup* se ejecuta primero y una sola vez. Este bloque sirve para inicializar el programa y la tarjeta Arduino. Por ejemplo, se

pueden inicializar variables, definir el modo de operación de los pines (conexiones) y llamar bibliotecas de funciones. Se utiliza un par de llaves para delimitar las líneas de programa que corresponden al bloque (ver Fig. 3). El bloque *loop* se ejecuta continuamente, es decir, una vez que se ejecuta la última línea de programa de ese bloque, se vuelve a ejecutar la primera línea y se continua con todas las siguientes, repitiéndose este proceso de manera indefinida. Los pares de paréntesis redondos después de las palabras *setup* y *loop* implican que no se les pasa ningún dato a las funciones *setup* y *loop.* La palabra *void* que antecede a las palabras *setup* y *loop* significa que dichas funciones no regresan ningún dato. El texto a la derecha de la doble diagonal '//', pero en la misma línea, es un comentario y no se interpreta como instrucciones del programa. Una breve descripción del concepto de funciones en programación se puede encontrar en el apéndice A.

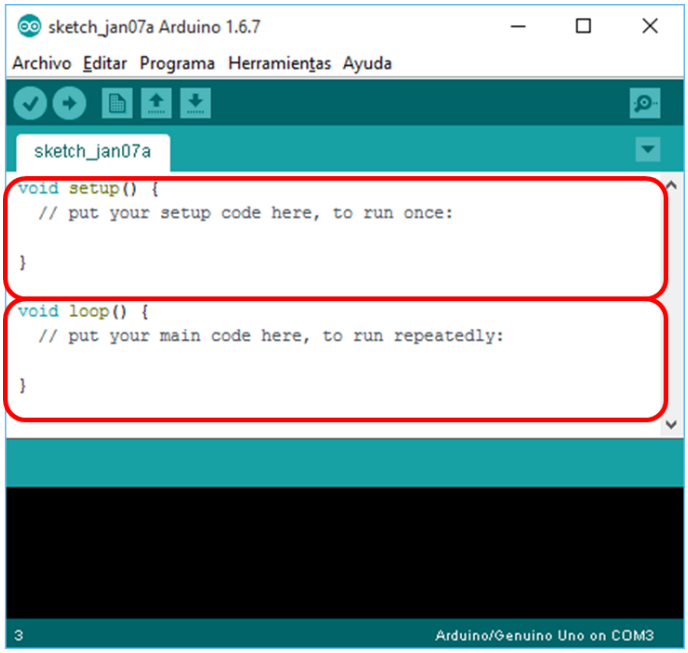


Fig. 2 Ventana del Arduino IDE que muestra los bloques de programación básicos enmarcados en rojo.

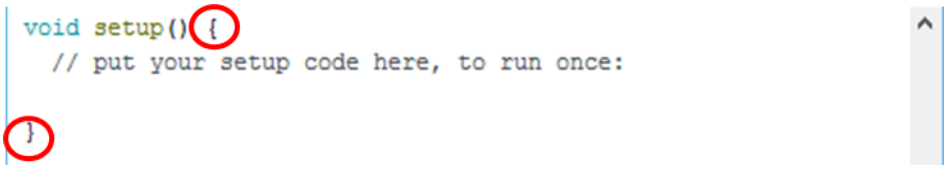


Fig. 3 Bloque de código setup *(inicialización). Los círculos rojos señalan a las llaves que delimitan el espacio del programa que contiene las instrucciones de inicialización.*

2 ENTRADAS Y SALIDAS DIGITALES

La tarjeta Arduino cuenta con un bloque de conexiones para señales digitales. Estas conexiones están numeradas del 0 al 13 y se muestran en la Fig. 4. Cada una de estas 14 líneas puede configurarse para funcionar como entrada o salida. Una señal digital es aquella que solo puede tener dos valores: encendido o apagado, justo como funciona un foco en una lámpara común. Una entrada digital puede utilizarse para detectar si fue presionado un botón, y una salida digital puede utilizarse para activar una sirena de alarma. El estado de encendido o apagado se puede asignar indistintamente a los números '0' y '1' de un sistema binario. En el caso de la tarjeta Arduino UNO, estos estados están representados por niveles de voltaje. Por ejemplo, en tarjetas cuyo microcontrolador opera a 5 V, los voltajes cercanos a 5 V pueden representar encendido, y los valores cercanos a 0 V apagado.

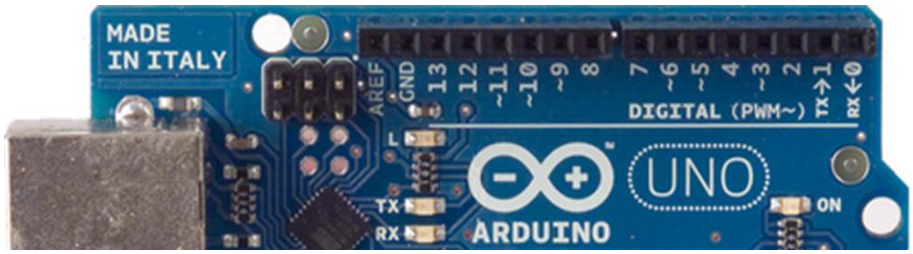


Fig. 4 Detalle de la tarjeta Arduino UNO mostrando los conectores de entrada y salida digital.

En la Fig. 4 también se observa que algunas terminales de conexión llevan una tilde ' ~ ' junto al número de conexión. Esta tilde identifica a las conexiones que pueden generar una señal PWM (Pulse Width Modulation), la cual es una señal cuadrada que se puede usar para, por ejemplo, controlar la velocidad de un motor. Una explicación breve de estas señales se proporciona en el apéndice B. Las conexiones 0 y 1 tienen una etiqueta RX y TX respectivamente y señalan a las líneas que se utilizan para transmitir (TX) y recibir (RX) información por el puerto USB. En general y para evitar conflicto con el puerto USB, omitiremos el empleo de estas dos líneas de conexión digital. La línea 13 tiene conectado un LED montado en la tarjeta Arduino UNO, por lo que también evitaremos su conexión.

Una forma sencilla de construir una entrada digital es utilizando un interruptor de presión (*push-button)* y una resistencia, como se muestra en la Fig. 5A. Cuando el interruptor es presionado, el circuito se cierra y la terminal de salida queda conectada a 5 V (ver Fig. 5B). Por otro lado, cuando el interruptor es liberado, se abre la conexión a 5 V. La terminal de salida está entonces conectada a tierra (0 V) a través de la resistencia R; esto produce un voltaje de salida cercano a los 0 V. El empleo de esta resistencia es indispensable para evitar un corto circuito cuando se cierra el interruptor. Un valor típico de la resistencia que se emplea en estos casos es R= 10 kΩ. El utilizar un valor menor puede resultar en una demanda excesiva de corriente a la fuente de alimentación.

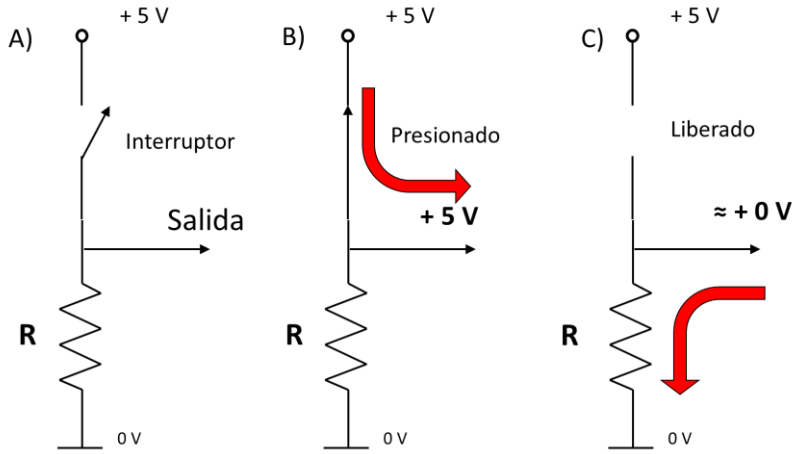


Fig. 5 Generación de una señal digital con un interruptor de presión. A) Diagrama esquemático. B) La salida es 0 V cuando el interruptor es presionado. C) La salida es casi 5 V cuando el interruptor está liberado.

Uno de los modelos de interruptor de presión más común se muestra en la Fig. 6, junto con sus diagramas esquemáticos, físicos y de conexión. Debe tomarse en cuenta que las cuatro patillas se encuentran conectadas por pares, es decir, es necesario identificar las patillas que entrarán en contacto eléctrico cuando se presione el botón. Para el interruptor de la figura, las patas más cercanas son las que se conectan al presionar el botón, mientras que las patas más alejadas están conectadas entre sí (ver parte inferior derecha de Fig. 6).

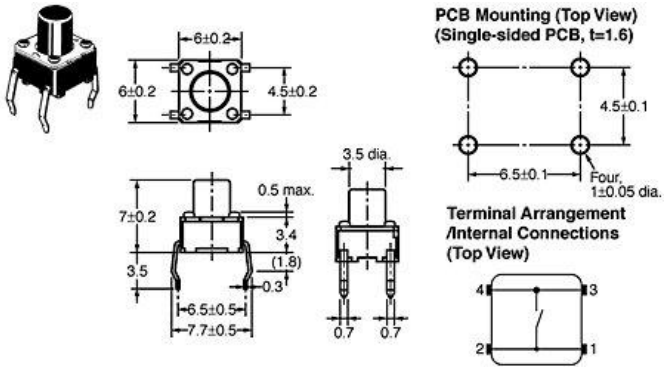


Fig. 6 Imagen de un interruptor de presión y su correspondiente diagrama físico y eléctrico (Sparkfun, 2017).

El ejemplo más sencillo de la utilización de una señal de salida digital consiste en encender un foquito, o un LED. El LED es un dispositivo de que emite luz, pero presenta diferencias de funcionamiento respecto a un foquito. Los LED pueden emitir luz de diferentes colores, y en versiones microscópicas forman parte de algunas pantallas digitales en la actualidad. Un foco puede conectarse de manera indistinta a las terminales positiva y negativa de una fuente de voltaje. Pero esto no sucede con un LED, el cual requiere una conexión particular a las terminales positiva y negativa, y además necesita una resistencia para limitar el flujo de corriente eléctrica (ver Fig. 7). Aplicar más de 5 V a un LED con la conexión invertida puede destruirlo.

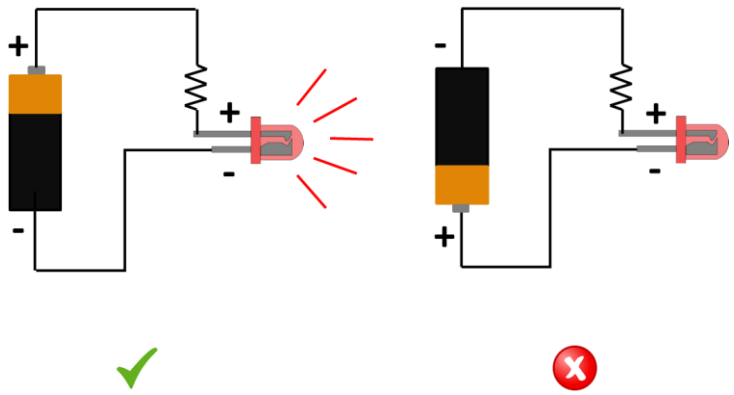


Fig. 7 Ilustración de la conexión correcta de un LED. Con figuras tomadas de (Was a bee, 2017).

Un LED tiene dos terminales llamadas ánodo (positiva) y cátodo (negativa). La identificación de las terminales es posible por dos métodos: en el primero, si se observa el LED desde la parte superior, se puede ubicar una muesca plana que señala al cátodo; en el segundo, y suponiendo que no se han trozado las patillas del dispositivo, entonces la más larga indicará al ánodo (+) y la más corta al cátodo (-) (ver Fig. 8).

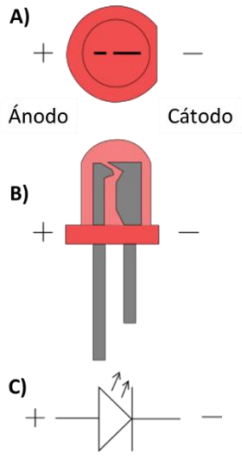


Fig. 8 Detalle de las terminales de un LED. A) Vista superior. B) Vista lateral. C) Símbolo eléctrico (Was a bee, 2017).

Un ejemplo sencillo de uso de señales digitales de entrada y salida con el Arduino consiste en encender el LED cuando el Arduino detecta que se ha presionado el botón (ver Fig. 9). Para armar el circuito correspondiente se requiere una resistencia de 10 kΩ (bandas: café, negro, naranja, oro), y una de 220 Ω (bandas: rojo, rojo, café, oro).

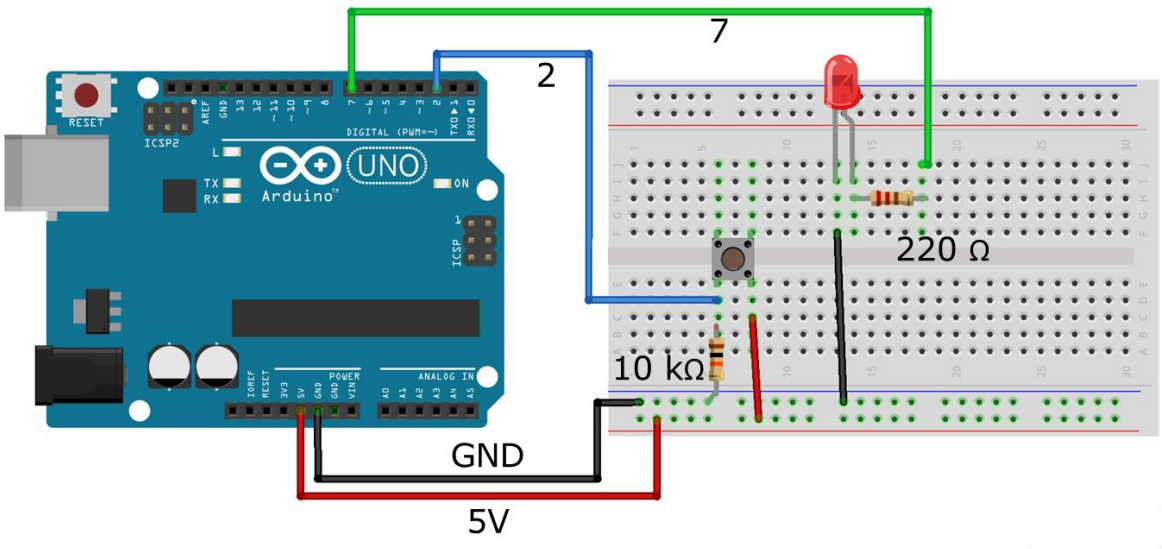


Fig. 9 Diagrama de conexiones de un circuito con interruptor y LED.

La placa blanca (el color puede variar) donde se insertan los componentes se denomina *protoboard* y tiene un arreglo de conexiones internas según se muestra en la Fig. 10. Las líneas verticales están conectadas por grupos de 5 orificios, y las líneas horizontales por grupos de 25 (o más dependiendo del tamaño de la placa). En la Fig. 11 se muestra una fotografía del circuito armado utilizando la protoboard, y un detalle de la conexión del LED en la Fig. 12. El programa para este ejercicio se encuentra en el catálogo de ejemplos del Arduino (en el Arduino IDE), para cargarlo basta con seleccionar el menú *Archivo-> Ejemplos -> Digital -> Button* (ver Fig. 13). El listado del programa se muestra en la Tabla 1.

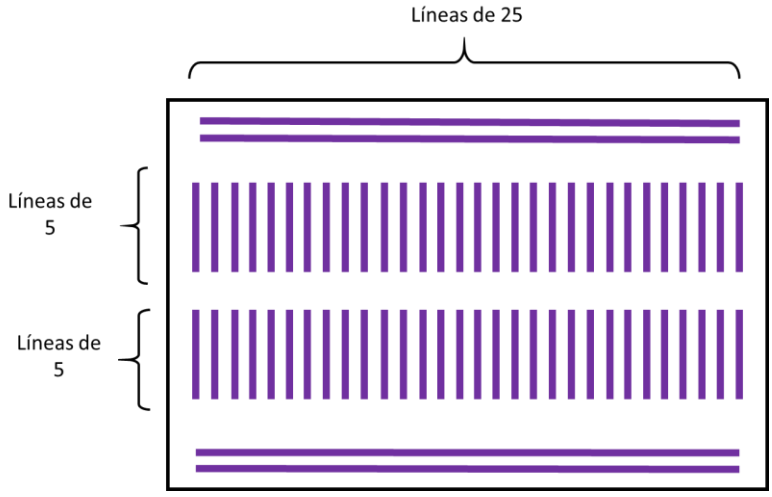


Fig. 10 Diagrama de conexiones internas de una protoboard.

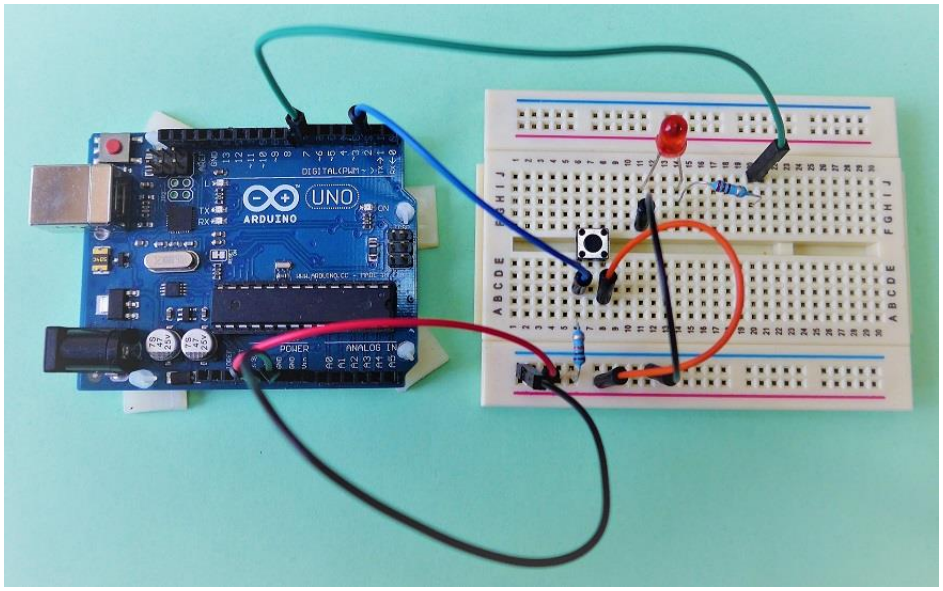


Fig. 11 Fotografía del circuito conectado a la tarjeta Arduino UNO.

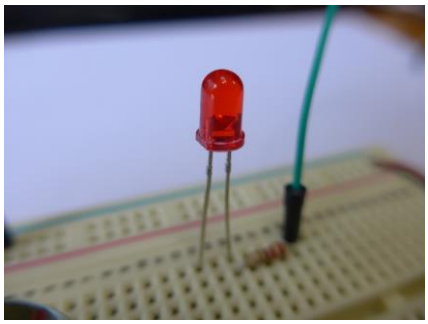


Fig. 12 Fotografía que muestra la forma de conectar el diodo emisor de luz (LED). Observe la orilla plana que señala el cátodo (conexión negativa).

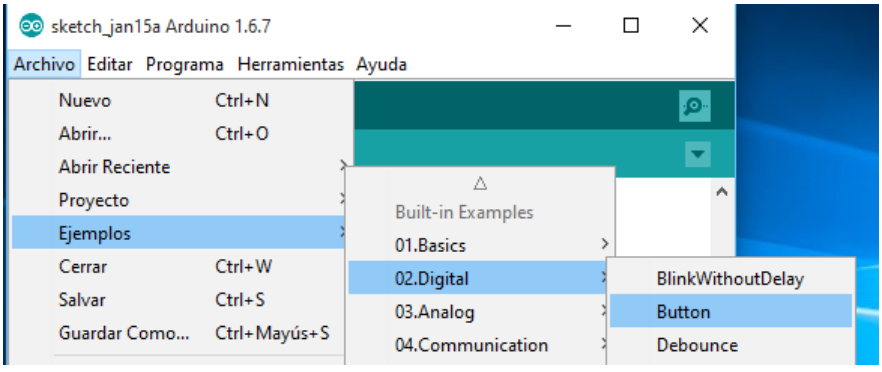


Fig. 13 Cargando un programa ejemplo.

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Tabla 1 Programa de ejemplo para encender un LED presionando un botón.

La primera parte del programa es una sección de comentarios (misma que ha sido omitida en el listado de la Tabla 1). Esta sección está delimitada por los pares de caracteres /* y */, los cuales marcan el inicio y fin de la sección de comentarios (ver Fig. 14).

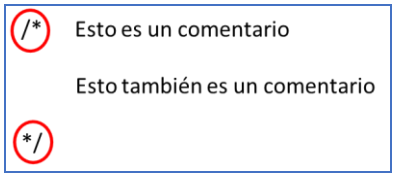


Fig. 14 Ejemplo de un bloque de comentarios de texto.

Después de la sección de comentarios, aparece una sección donde se definen algunas constantes del programa (ver Tabla 2). Se define una constante de valor entero *buttonPin=2.* Se denomina constante, porque su valor no cambiará en el resto del programa. Esto nos ayudará a definir la terminal digital

número 2, como la encargada de recibir la señal del interruptor de presión. En lo sucesivo, sólo se tendrá que hacer referencia a la palabra *buttonPin,* y el programa sabrá que está conectado en el pin 2 de las conexiones digitales. De manera similar se define la constante entera *ledPin=13.* Para hacer compatible el programa con el circuito mostrado en la Fig. 9, deberemos cambiar la instrucción para que quede *ledPin=7,* ya que será en el canal digital 7 donde estará conectado el LED*.*

A continuación, aparece una sección de declaración de variables globales, es decir, serán variables válidas en todos los demás bloques de programa (ver Tabla 3). Con la instrucción *buttonState=0*, se declara una variable entera y se le asigna un valor inicial de cero (esta asignación es opcional). Observe que ya no se utiliza la palabra *const*, la cual fue usada para definir las constantes del bloque precedente. Esta variable será utilizada para almacenar el valor del botón.

```
// Las constantes no cambian
// Se definen los pines que se utilizarán:
const int buttonPin = 2;     // El pin 2 corresponde al botón
const int ledPin =  13;      // El pin corresponde al LED
```

Tabla 2 Bloque de declaración de constantes. e inicialización de variables.

```
// Las variables sí cambian:
int buttonState = 0;         // Variable para leer el estado del botón
```

Tabla 3 Bloque de declaración e inicialización de variables.

A continuación, se puede grabar el programa del Arduino en nuestra computadora utilizando el menú Archivo -> Guardar como, y seleccionando un nombre adecuado a nuestro proyecto (p. ej. LED_boton). Después podemos compilar el código de nuestro programa con el botón izquierdo (con forma de palomita) de la barra de botones (ver Fig. 15). Este proceso revisa que no haya errores de sintaxis. Para evitar problemas de comunicación con la tarjeta, es conveniente verificar que está correctamente seleccionada en el menú “Herramientas -> Placa”, donde debe aparecer el modelo de nuestra tarjeta (p. ej. Arduino UNO). En la Fig. 16 se muestra cómo verificar que se ha seleccionado el puerto correcto de comunicación USB, mediante el menú “Herramientas -> Puerto:” (p. ej. COM4 (Arduino/Genuino UNO). El número de puerto puede variar, pero debe seleccionarse aquel que tenga a su lado el mensaje Arduino/Genuino UNO. Después de revisar esta configuración ya podemos subir el programa a la tarjeta Arduino (con el botón -flecha a la derecha-). Finalmente se puede probar que el LED enciende cada vez que se presiona el botón.

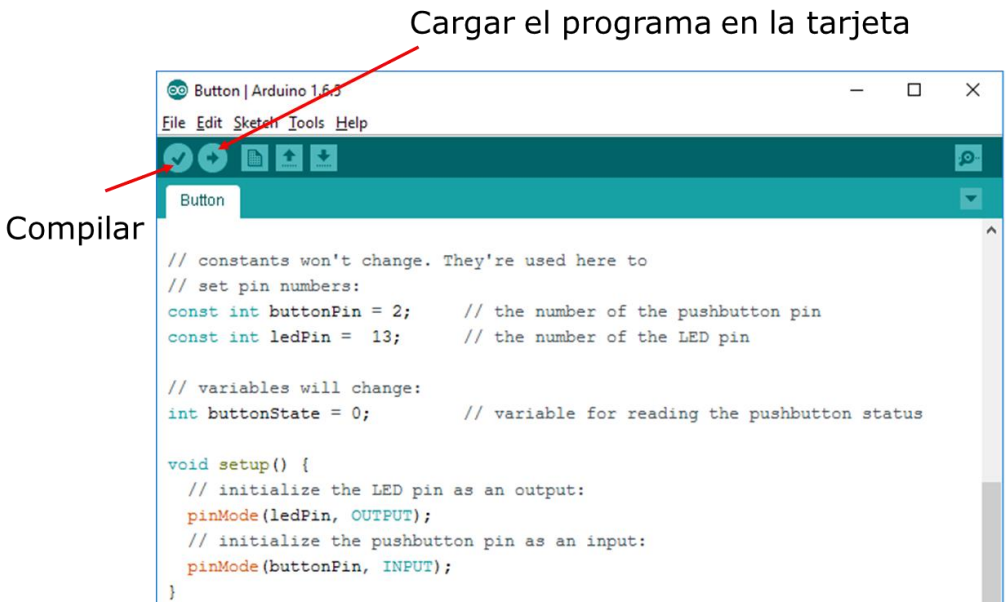


Fig. 15. Barra de botones del ambiente Arduino.

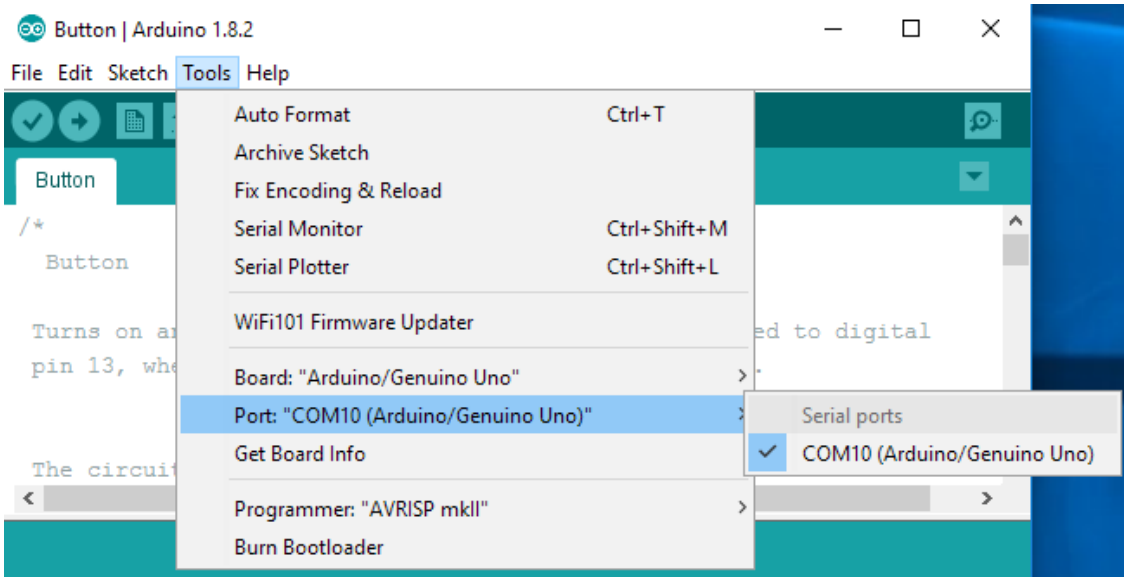


Fig. 16 Menú de herramientas mostrando las opciones Placa *y* Puerto *configuradas para utilizar una tarjeta Arduino UNO.*

La rutina principal del programa se muestra en la Tabla 4. La instrucción

```
buttonState = digitalRead(buttonPin);
```

lee el estado (alto o bajo) del pin 2 de la tarjeta Arduino (recordar que se definió previamente la constante buttonPin=2) y el resultado se asigna a la variable buttonState. La lectura de un pin digital solo puede arrojar dos resultados cero o uno, que se interpretan como LOW o HIGH por el programa, respectivamente.

La instrucción

```
if (buttonState == HIGH)
```

sirve para preguntar si el botón fue presionado (y por lo tanto se lee un voltaje ALTO en el pin). En caso afirmativo se ejecuta la instrucción que enciende el LED:

```
digitalWrite(ledPIN, HIGH);
```

y en caso contrario la instrucción para apagar el LED:

```
digitalWrite(ledPin, LOW);
```

Se pueden ejecutar varias instrucciones cuando la condición (buttonState == HIGH) se cumple, y estas instrucciones están delimitadas por un par de llaves. Lo mismo ocurre para el caso de que la condición no se cumpla.

El programa puede modificarse para hacer que el LED parpadee cada vez que se presiona el botón. Para ello, basta con cambiar algunas instrucciones dentro de la sentencia IF, para que quede como se muestra en la Tabla 5. El resto del programa se conserva sin cambios.

```
void loop() {
  // Lee el estado del botón
  buttonState = digitalRead(buttonPin);

  // Revisa si el botón fue presionado
  if (buttonState == HIGH)  {         // Pregunta si buttonState es HIGH:
    // Entonces enciende el LED:
    digitalWrite(ledPin, HIGH);
  }
else
{                 // En caso contrario
    // Apaga el LED
    digitalWrite(ledPin, LOW);
  }
}
```


Tabla 4 Rutina principal del programa de encendido de un LED.

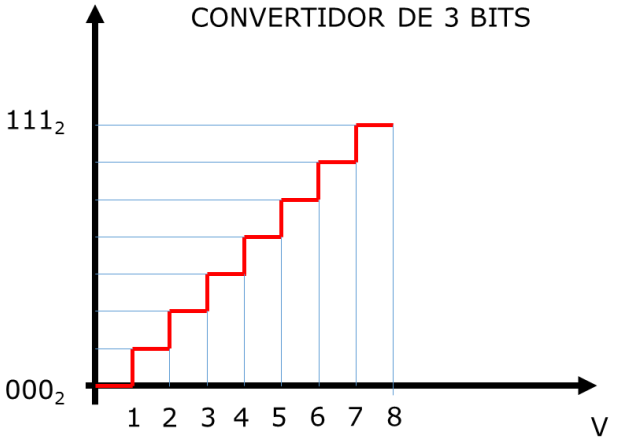
```
if (buttonState == HIGH) {
   // Se enciende el LED y parpadea:
   digitalWrite(ledPin, HIGH);   // Se enciende el LED)
   delay(300);              // Espera 300 milisegundos
   digitalWrite(ledPin, LOW);    // Apaga el LED
   delay(300);              // Espera 300 milisegundos
  }
else {
```

Tabla 5 Código que se modifica para hacer que el LED parpadee.

3 ENTRADAS Y SALIDAS ANALÓGICAS

La tarjeta Arduino UNO cuenta con 6 pines de entrada analógica, etiquetados de A0 a A5. Usualmente se utilizan para leer voltajes entre 0 y 5 V CD. La lectura se hace por medio de un convertidor analógico digital interno de 10 bits, es decir, entrega códigos que van de 0 000 000 000 a 1 111 111 111, esto es, entre 0 y 1023 en base decimal.

Una señal analógica puede tomar cualquier valor entre sus límites. Por ejemplo, si la señal varía entre 0 y 5 V, entonces un valor posible es 2.5, y otro sería 2.500001. Los valores de una señal digital, en cambio, están limitados por el número de bits que se utilizan. En el caso de un convertidor analógico-digital de 3 bits, solo hay 8 posibles combinaciones, tal y como se muestra en la columna derecha de la tabla de la Fig. 17. En esa misma figura, se ilustra el caso en que una señal analógica de 0 a 8 V se convierte a su valor digital. Cualquier valor de voltaje entre 1 V y 2 V generará el mismo código binario (001_2), y ocurre algo similar para cualquier valor entre los voltajes 2 V y 3 V. Esto produce una respuesta escalonada. Al proceso de tomar una señal analógica (con un número infinito de valores) y convertirla en una señal con solo un número limitado de valores se le denomina *cuantización.*



VOLTAJE	CÓDIGO BINARIO
0-1	000
1-2	001
2-3	010
3-4	011
4-5	100
5-6	101
6-7	110
7-8	111

Fig. 17 Ejemplo del funcionamiento de un convertidor analógico-digital de 3 bits.

En el caso del convertidor analógico-digital de la tarjeta Arduino UNO, el proceso de cuantización a 10 bits se vuelve tan fino, que la gráfica no se vería escalonada, sino continua (ver Fig. 18).

Para ejemplificar el uso de una entrada analógica utilizaremos un sensor de temperatura. El circuito integrado LM35 (Texas Instruments, 2016) es un dispositivo que se puede alimentar con 5 V y entrega en su terminal de salida un voltaje que es proporcional a la temperatura en grados centígrados a razón de 10 mV por cada 1°C (ver Fig. 19). Es decir, si la temperatura es de 25 °C, el sensor entregará un voltaje de 250 mV.

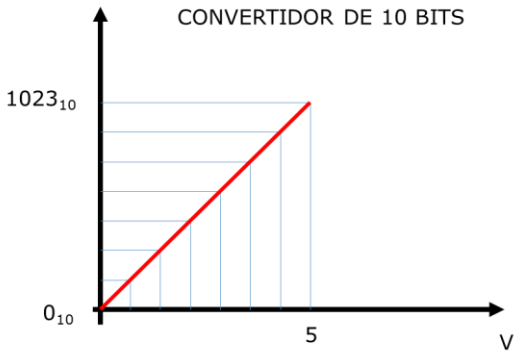


Fig. 18 Gráfica de cuantización de un convertidor analógico-digital de 10 bits.

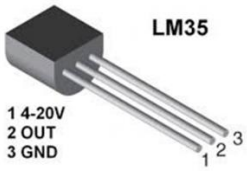


Fig. 19 Sensor de temperatura LM35 (Instructables, 2015).

Para su utilización podemos seguir el diagrama de conexiones mostrado en la Fig. 20. La terminal de salida del sensor de temperatura (terminal central) se conecta a una de las entradas analógicas de la tarjeta Arduino (A0). En la Fig. 21, se muestran detalles de la conexión del sensor LM35.

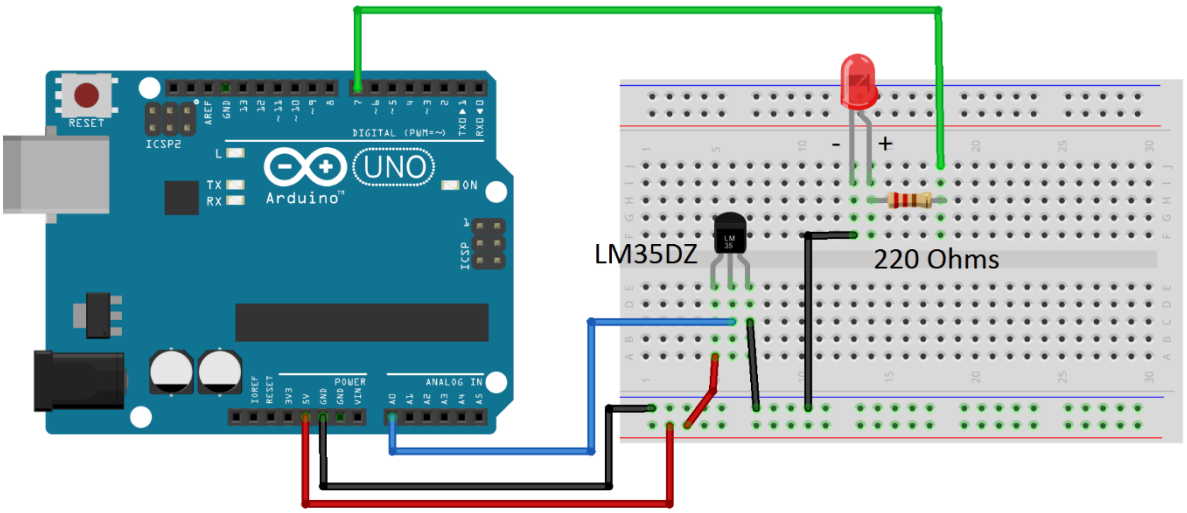


Fig. 20 Esquema de conexiones del sensor de temperatura y un LED.

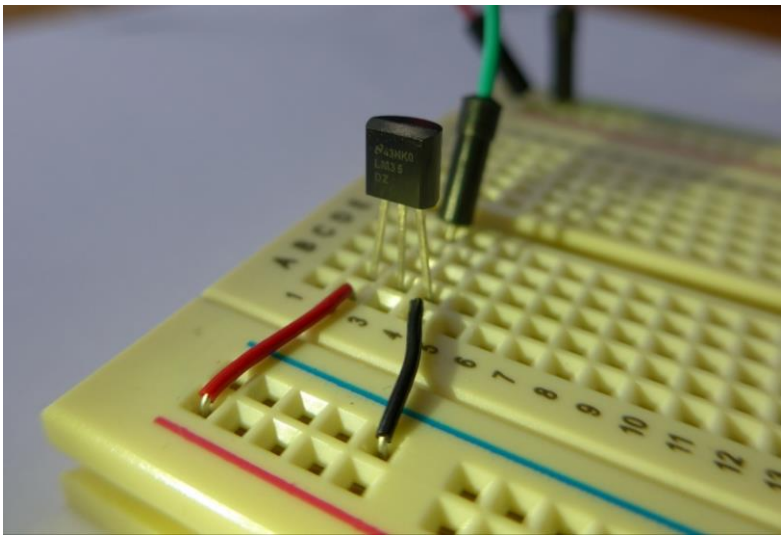


Fig. 21 Detalle de la conexión del sensor LM35.

En la Tabla 6, se muestra el código de un programa que lee el sensor de temperatura y enciende un LED cuando la temperatura es mayor a 25 °C. Para comprobarlo, basta con presionar con los dedos el sensor para que su temperatura aumente. La instrucción que se encarga de leer al sensor es

```
valorSensor = analogRead(pinSensor);
```

El valor que toma la variable *valorSensor* estará entre 0 y 1023, aunque en la práctica corresponda a un intervalo menor debido a que no se alcanzan temperaturas lo suficientemente altas como para que el sensor entregue una señal de 5 V. Obsérvese que la primera sentencia del bloque *loop* (programa principal) es

```
float milivolts;
```

esta instrucción declara (define) una variable de punto flotante (y que solo puede ser usada dentro del bloque *loop*).

```
/*  Ejemplo con el sensor de temperatura LM35
      Si la temperatura supera un valor umbral se encenderá un LED  */
// En esta sección se definen las constantes
        const int pinSensor = A0;     // Entrada analógica. Se conecta al sensor de temperatura
        const int pinLed    = 7;      // Pin de salida digital para el LED
        const int umbral    = 25;     // Umbral de la temperatura en grados centígrados
        int valorSensor = 0;          // Variable para guardar el valor leído del sensor
        float temperatura = 0;        // Variable para guardar la temperatura
// Sección de inicialización
        void setup() {
        pinMode(pinLed, OUTPUT);   // Se define el pin 7 como salida
        }
// Programa principal
        void loop() {
            float milivolts;   // Se define una variable de punto flotante
           valorSensor = analogRead(pinSensor);   // leemos el valor del sensor
           float milivolts = 5000*(valorSensor / 1023.0);  // Se convierte a milivolts
           temperatura = milivolts/10;    // Se convierte a grados centígrados
           if (temperatura > umbral){      // Se compara la temperatura con el umbral
              digitalWrite(pinLed, HIGH);  // Si se supera el umbral se enciende el LED
           } else {
              digitalWrite(pinLed, LOW);     // En caso contrario se apaga el LED
           }
         delay(1000);   // Se introduce un retardo de 1 segundo antes de repetir el programa
        }
```

Tabla 6 Código del programa que controla la intensidad de un LED con un potenciómetro.

El Arduino puede generar señales de pulsos modulados (PWM, *pulse width modulation)* que funcionan como *salidas analógicas* en ciertas condiciones. Estas señales de PWM se pueden utilizar para controlar la intensidad de un LED o la velocidad de un motor de corriente directa. En el Apéndice B se encuentra una breve explicación de este tipo de señales.

Como ejemplo de salida analógica se propone controlar la intensidad de un LED por medio de un potenciómetro. El potenciómetro es un dispositivo resistivo con tres terminales. El símbolo eléctrico se muestra en la Fig. 22. Al rotar el vástago se modifica la resistencia entre las terminales A-W y B-W. El diagrama de conexiones se muestra en la Fig. 23. En la Fig. 24 se muestra un ejemplo de conexión del potenciómetro. Sus terminales pueden incrustarse firmemente en la tableta protoboard. El LED se ha conectado al pin 6 que tiene la capacidad de producir una señal PWM (lo cual se indica con una tilde ~ junto al número). El programa se muestra en la Tabla 7.

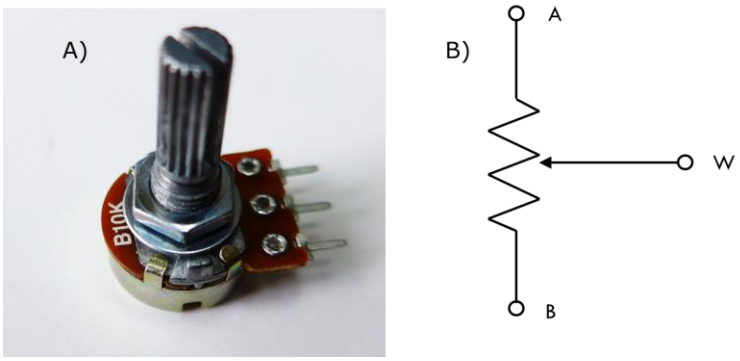


Fig. 22 Potenciómetro. A) Fotografía. B) Diagrama eléctrico.

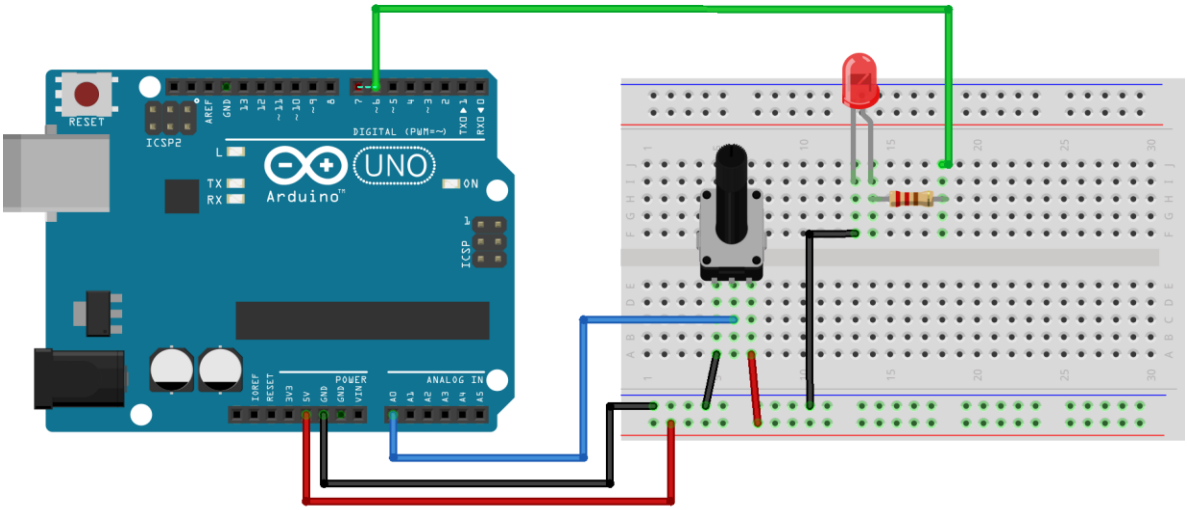


Fig. 23 Esquema de conexiones para el LED y el potenciómetro.

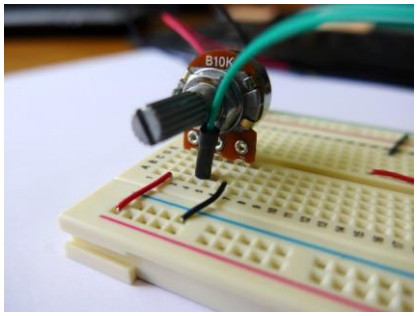


Fig. 24 Fotografía mostrando una forma de conectar el potenciómetro.

La instrucción

```
analogWrite(pinLed, brilloLed);
```

es la encargada de generar la señal de PWM. Esta instrucción acepta valores entre 0 y 255. Como la lectura del potenciómetro produce un dato entre 0 y 1023, se hace un re-escalamiento del valor mediante la instrucción

```
brilloLed = map(valorPotenciometro, 0, 1023, 0, 255);
```

```
/*   Programa que el valor de un potenciómetro
            y lo usa para ajustar el brillo a un led usando una señal PWM  */
// Declaración de constantes
            const int pinSensor = 0;   // Sensor analógico conectado al potenciómetro
            const int pinLed    = 6;   // Salida PWM conectada al LED
// Declaración de variables
            int brilloLed         = 0;  // Almacena el valor de brillo que se dará al LED
            int valorPotenciometro = 0;  // Almacena el valor leído del potenciómetro

// Inicialización
            void setup() {
               pinMode(pinLed, OUTPUT);    // Define al pin 6 como salida PWM
            }
// Programa principal
            void loop() {
               valorPotenciometro = analogRead(pinSensor);  // Lee el valor del potenciómetro
                                                  // Se obtiene un valor entre 0 y 1023

               brilloLed = map(valorPotenciometro, 0, 1023, 0, 255);  // Reescala el valor del
                                                  // potenciómetro al intevalo 0 a 255

               analogWrite(pinLed, brilloLed);  // Se ajusta la salida del pin PWM
               delay(100);     // Retardo de 100 milisegundos
            }
```

Tabla 7.Código del programa que controla la intensidad de un LED con un potenciómetro.

Es conveniente mencionar que la instrucción *analogWrite* no tiene nada que ver con los pines de entrada analógica ni con la instrucción *analogRead.*

4 COMUNICACIÓN SERIAL

El puerto USB que se utiliza para programar el Arduino también se puede usar para establecer comunicación con una computadora personal (PC). De esta manera, con solo algunas instrucciones y un programa instalado en la PC es posible enviar y recibir secuencias de caracteres.

Como ejemplo elaboraremos un programa que envíe a la PC la temperatura registrada por un sensor de temperatura. Para ello, necesitamos el mismo circuito de la Fig. 20 y el programa de la Tabla 8. Se utiliza la instrucción *Serial.print* para enviar letreros y datos a la PC. Por otro lado, la instrucción

```
Serial.println
```

añade una instrucción de cambio de línea (retorno de carro).

```
/*  Programa que utiliza un sensor de temperatura LM35, lee
  *  la temperatura y envía el dato por el puerto serial a la PC */

// Declaración de constantes
    const int pinSensor = A0;     // pin del sensor de temperatura LM35

// Declaración de variables
    int valorSensor = 0;          // Guarda el valor leído del sensor
    float temperatura = 0;        // Guarda el valor de la temperatura

// Inicialización
    void setup() {
       pinMode(pinLed, OUTPUT);  // Define el pin de salida
       Serial.begin(9600);       // Establece la velocidad de comunicación serial
    }

// Programa principal
    void loop() {
       valorSensor = analogRead(pinSensor);   // Lee el valor del sensor de temperatura
       float milivolts = 5000*(valorSensor / 1023.0);   // Convierte el valor a miliVolt
       temperatura = milivolts/10;   // Calcula el valor en grados centígrados
       Serial.print("La temperatura es de: ");     // Envía un letrero a la PC
       Serial.print(temperatura);                  // Envía el dato de la temperatura a la PC
       Serial.println(" grados centigrados");      // Envía un letrero a la PC
       delay(1000);  // Retardo de 1000 milisegundos
    }
```

Tabla 8 Código del programa para leer un sensor LM35 y enviar el dato por puerto serie a la PC.

Para poder visualizar los datos que recibe la PC se activa la función de monitor serial, dentro de la barra de funciones (ver Fig. 25). Los resultados entonces serán desplegados como se muestra en la Fig. 26. Obsérvese que en la parte inferior derecha aparece un letrero con la leyenda BAUDS 9600, ahí se configura la velocidad de comunicación y debe de coincidir con lo establecido dentro del programa.

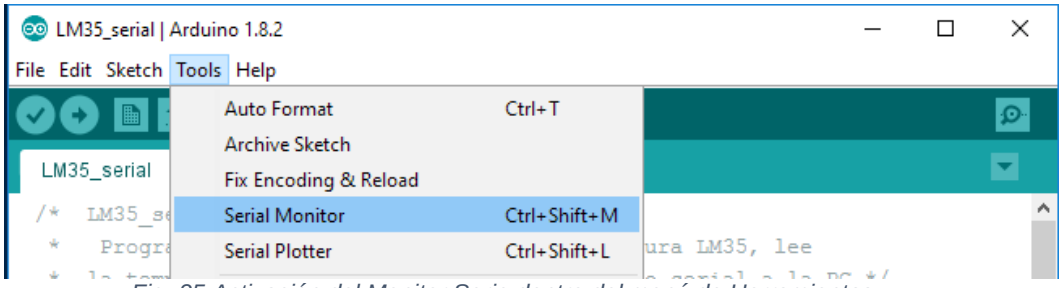


Fig. 25 Activación del Monitor Serie dentro del menú de Herramientas.

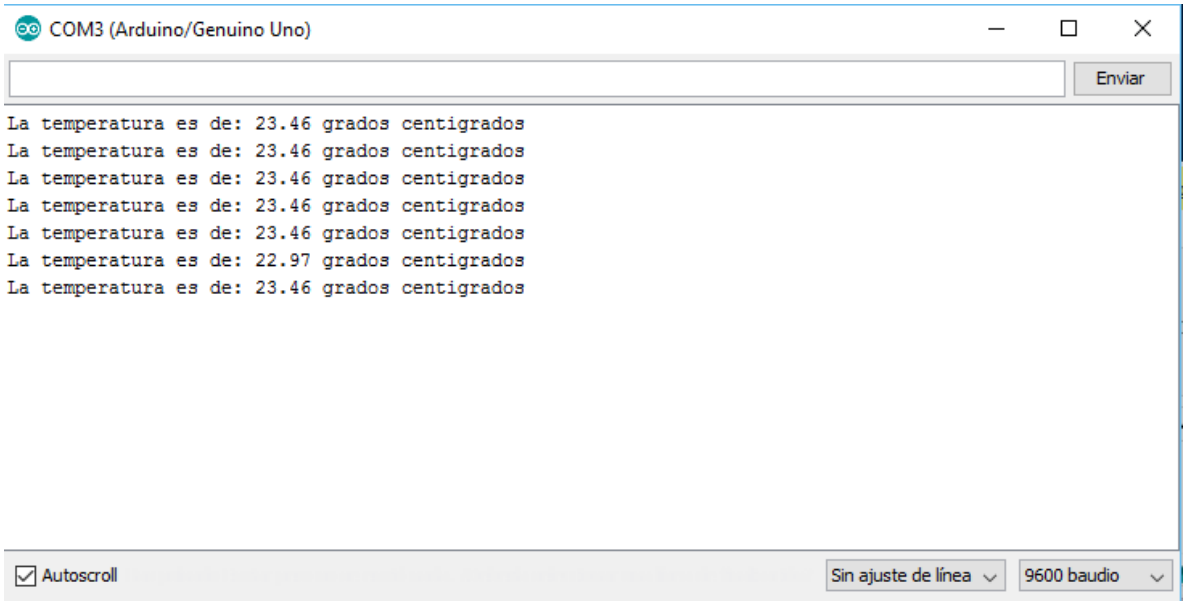


Fig. 26 Ventana del Monitor Serial desplegando los datos de la temperatura.

Ahora se propone realizar otro ejemplo, utilizando el mismo circuito de la Fig. 20 y elaborando un programa para controlar el encendido del LED mediante las teclas S y N, en el teclado de la computadora. El objetivo de este programa es que el LED se encienda si el usuario presiona la tecla 'S' o 's' en la computadora, y se apague cuando el usuario presione la tecla 'N' o 'n'. El código se muestra en la Tabla 9.

```
/*  Programa que utiliza la comunicación serial
  *  para encender un LED */
// Declaración de constantes
    const int pinLed    = 7;      // pin del LED
// Declaración de variables
    int valorLed = 0;          // Guarda el estado del LED, encendido o apagado
    int datoRecibido = 0;        // Es el dato que se recibe por el puerto serial
// Inicialización
    void setup() {
       pinMode(pinLed, OUTPUT);  // Define el pin de salida
       Serial.begin(9600);       // Velocidad de la comunicación serial
       Serial.println(" Encender el LED (Y/N)");   // Envía un letrero a la PC
    }
// Programa principal
    void loop() {
       if (Serial.available() ) {
          datoRecibido = Serial.read();
          if (datoRecibido == 'S' or datoRecibido == 's') {
            digitalWrite(pinLed, HIGH);    // Si se recibe S o s enciende el LED
            Serial.println("LED encendido");
          }
          else if (datoRecibido == 'N' or datoRecibido == 'n') {
            digitalWrite(pinLed, LOW);     // Si se recibe N o n se apaga el LED
            Serial.println("LED apagado");  }
          else {
            Serial.println("Tecla incorrecta");
            }
          }
       delay(1000);   // Espera un segundo
    }
```

Tabla 9 Código de un programa para controlar el encendido de un LED desde la PC.

Una opción más versátil que el Monitor Serie del Arduino es el programa CoolTerm, el cual es de uso gratuito y cuenta con funciones de registro de datos (es decir, se pueden guardar todos los datos capturados en un archivo), y se puede descargar de la siguiente liga

http://freeware.the-meiers.org/ (Roger Meier, 2017)

Podemos seleccionar descargar la versión para Windows y el dar clic en la liga correspondiente nos permite descargar el archivo CoolTerm_Win.zip. En el Apéndice D, se encuentra una breve explicación de cómo descomprimir el archivo.

Para utilizar el nuevo programa de comunicación serial podemos modificar el programa que lee la temperatura de un sensor LM35 y envía el dato por el puerto serial (Tabla 8), con unos pequeños cambios en la rutina principal, de tal manera que solo envíe los datos, sin ningún letrero. Las modificaciones al programa se muestran en la Tabla 10.

```
// Programa principal
    void loop() {
       valorSensor = analogRead(pinSensor);   // Lee el valor del sensor de temperatura
       float milivolts = 5000*(valorSensor / 1023.0);   // Convierte el valor a miliVolt
       temperatura = milivolts/10;   // Calcula el valor en grados centígrados
    //    Serial.print("La temperatura es de: ");     // Envía un letrero a la PC
       Serial.println(temperatura);                  // Envía el dato de la temperatura
    //    Serial.println(" grados centigrados");      // Envía un letrero a la PC
       delay(1000);  // Retardo de 1000 milisegundos
    }
```

Tabla 10 Modificación al código del programa que lee un sensor LM35 y envía el dato a una PC.

Las modificaciones consisten en eliminar los textos que se envían (añadiendo caracteres de comentario al principio), y modificando la instrucción que envía el valor de la temperatura, para que incluya el código de fin de línea (Serial.println). El Monitor Serial del Arduino identifica automáticamente en qué puerto está conectada la tarjeta Arduino, sin embargo, al utilizar el programa CoolTerm, debemos seleccionar manualmente el número de puerto. Para ello, debemos anotar qué puerto utiliza el Arduino IDE, esto se muestra en cualquiera de dos lugares dentro del programa. El primero es bajo la persiana de selección Tools->Port, y el segundo es en la parte inferior derecha de la ventana del Arduino IDE. En la Fig. 27 se muestran estos dos métodos. Para el caso ilustrado, la tarjeta Arduino está conectada en el puerto COM10. Los puertos de comunicación serial se designan por las tres letras 'COM' seguidas de un número.

Una vez anotado el puerto de comunicación de la tarjeta Arduino, podemos ejecutar el programa CoolTerm.exe. Ahí, debemos seleccionar el mismo puerto y para ello entramos al menú *Options->Serial Port*, tal como se muestra en la Fig. 28. Con la tarjeta Arduino conectada a la PC y cargada con el programa ya modificado, se puede dar clic en el botón *Connect* para que el programa CoolTerm comience a recibir los datos. Si ahora deseamos iniciar un registro de ellos, debemos ir a la opción *Connection-> Capture textfile -> Start* (ver Fig. 29). Se utiliza la opción Stop para finalizar el registro. También es posible añadir la hora a la que se adquirió el dato usando el menú *Connection-> Options -> Receive -> Add timestamps to received data*. Adicionalmente se puede seleccionar que sean marcas de tiempo relativo en la opción *Type*.

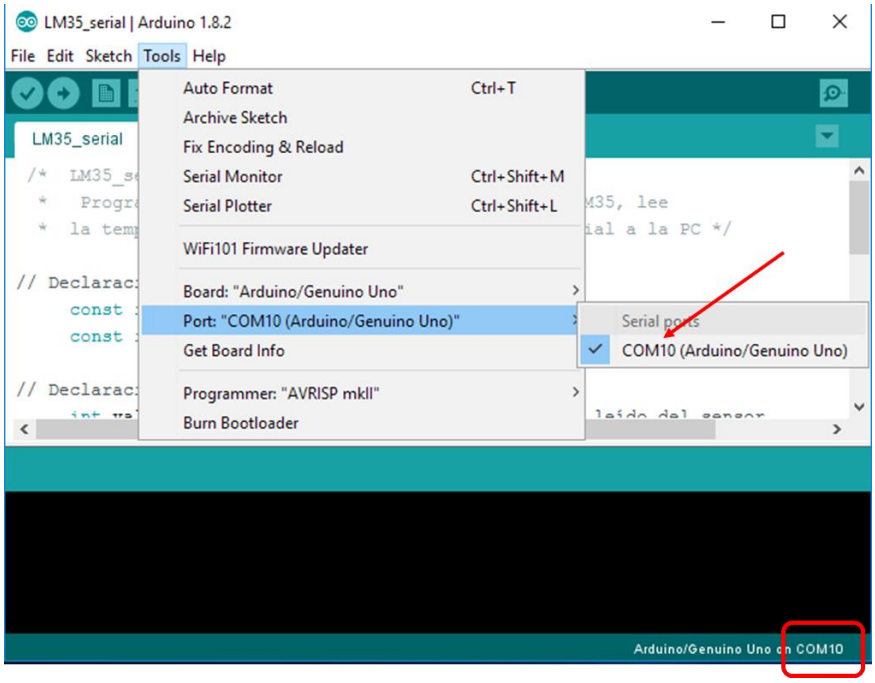


Fig. 27 Determinación del puerto de comunicación que utiliza la tarjeta Arduino UNO.

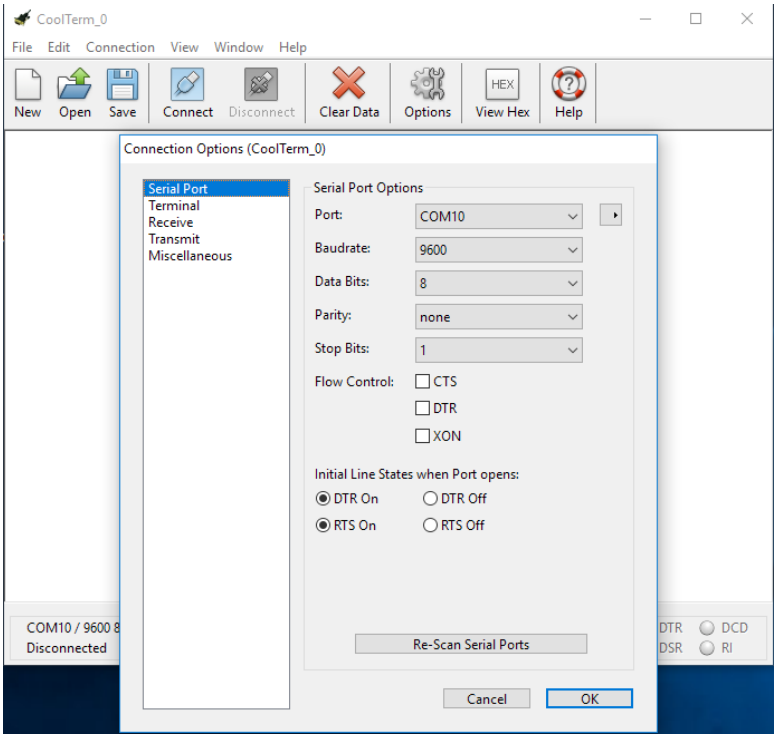


Fig. 28 Ventana de configuración del puerto serial en el programa CoolTerm.

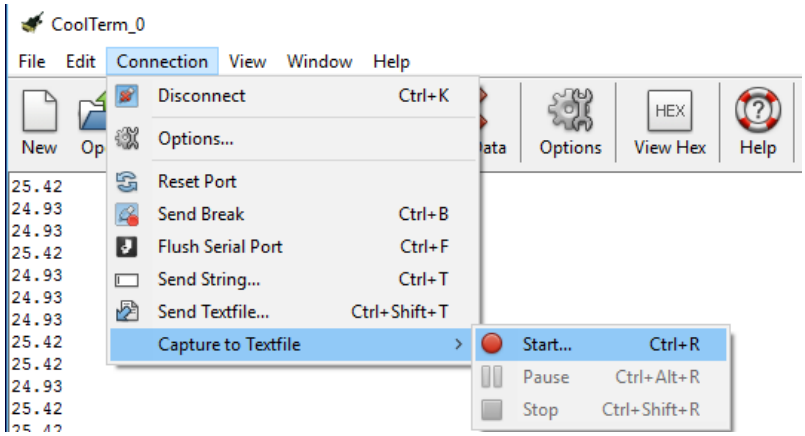


Fig. 29 Activación del registro de datos en el programa CoolTerm.

Los datos son guardados en formato texto (.txt) pudiendo abrirse y graficarse utilizando una hoja de cálculo como Excel (ver Fig. 30)

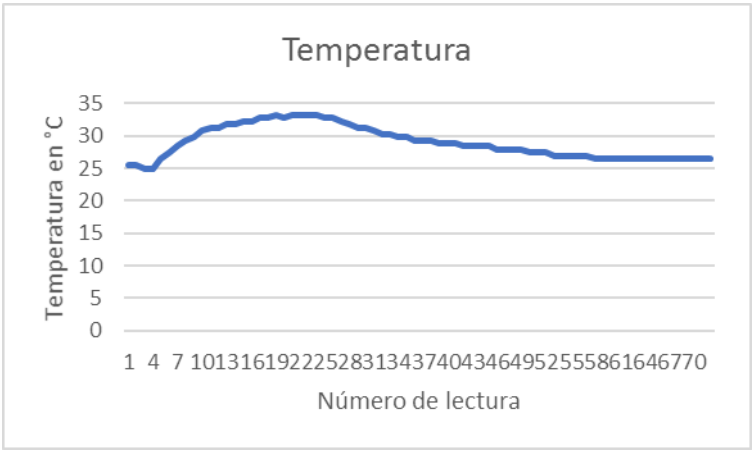


Fig. 30 Ejemplo de graficación de datos de temperatura adquiridos usando el programa CoolTerm.

5 MANEJO DE UN DISPLAY LCD

El manejo de un display de cristal líquido (LCD) se simplifica utilizando las bibliotecas de funciones del sistema Arduino. Como ejemplo, se propone construir un termómetro que muestre la temperatura en la pantalla del display. El sensor de temperatura que se utiliza es un circuito integrado LM35.

Existen diferentes tipos de pantallas de cristal líquido (LCD, por sus siglas en inglés), como existen diversos fabricantes, es común que simplemente se especifiquen por el número de letras (caracteres) y renglones que es posible desplegar. En nuestro ejemplo utilizaremos una de 16x2, es decir, dos renglones de 16 letras (caracteres) cada uno. El diagrama de conexiones con una tarjeta Arduino UNO se muestra en la Fig. 31. Se utiliza una resistencia de 220 Ω (rojo, rojo, café) para encender la luz de retroiluminación. También se utilizan dos resistencias (8.2 kΩ y 1 kΩ), para regular el contraste de la pantalla. En la Tabla 11, se muestra el código de un programa para leer la temperatura y desplegarla en la pantalla.

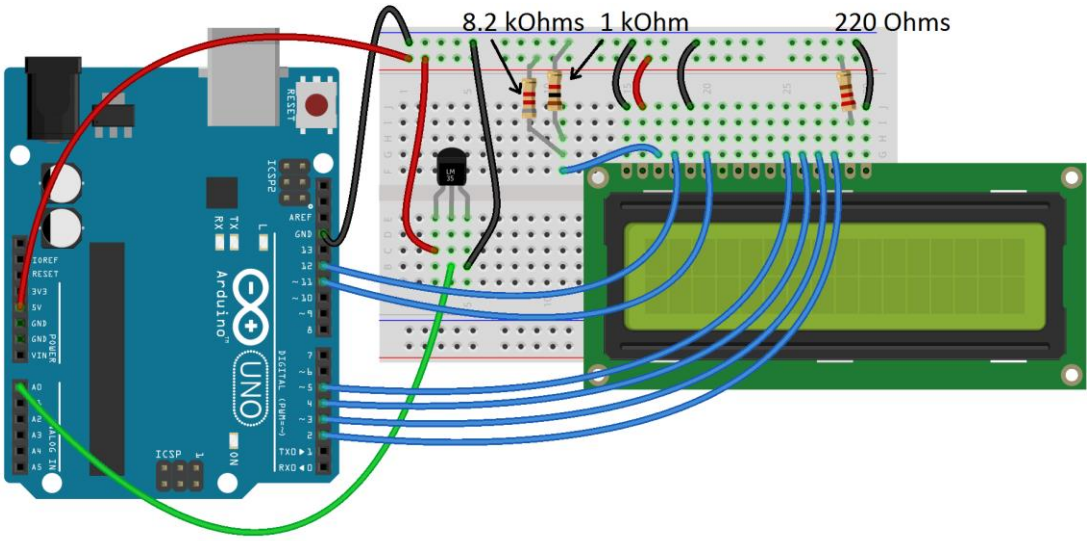


Fig. 31 Conexiones para el ejemplo del manejo de un LCD y un sensor de temperatura.

Para poder conectar el display en la protoboard es necesario soldarle una tira de pines como se muestra en la Fig. 32.

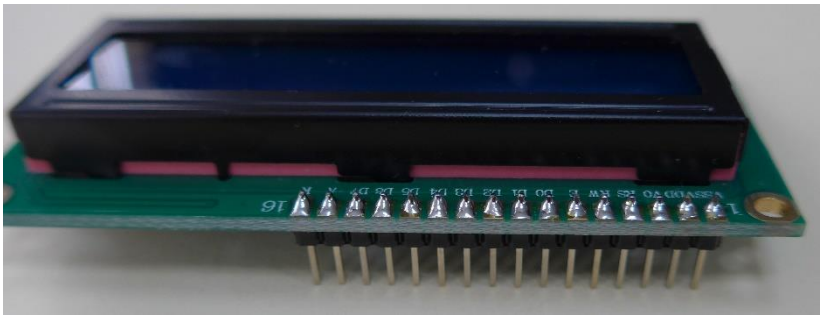


Fig. 32 Tira de pines soldada al display LCD.

A continuación, se comentan algunos detalles del programa. Para manejar el display se requiere utilizar una biblioteca y eso se logra añadiendo al programa la siguiente instrucción

```
#include <LiquidCrystal.h>
```

```
// Se requiere la biblioteca de uso del display
            #include <LiquidCrystal.h>
// Se definen los pines que se comunican con el display
            LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
            const int pinSensor = 0;   // A0 es el pin del sensor analógico
  int valorSensor = 0;   // Guarda el valor del sensor
  float temperatura = 0;
// Inicialización
            void setup() {
              lcd.begin(16, 2);   // LCD es de 2 renglones y 16 caracteres
    pinMode(pinSensor, INPUT);
              lcd.print("La temp es:");    // Manda un letrero al display
}
            void loop() {          // Programa principal
   // lcd.setCursor(col, row) Formato de control de la posición de impresión
    lcd.setCursor(0, 1);   // Selecciona caracter 0, fila 1 (segunda)
    valorSensor = analogRead(pinSensor);
    temperatura = 500*(valorSensor/1023.0);
        lcd.print("      ");
        lcd.setCursor(0, 1);
            lcd.print(temperatura);
        delay(300);
           }
```

Tabla 11 Código del programa que usa un sensor de temperatura LM35 y muestra el resultado en un display.

El display cuenta con dos pines de conexión que sirven para suministrar alimentación al propio display, y dos pines especiales para alimentar la retroiluminación. Además, se usa un pin para regular el contraste de la pantalla, cuatro para mandarle datos, y dos más para enviarle señales de control. La secuencia de valores de estas señales es controlada por la biblioteca *LiquidCrystal.h* y para poder enviar mensajes al display solo se requiere seleccionar la ubicación del texto, y enviarlo. La ubicación se hace con la instrucción

```
lcd.setCursor(columna, fila);
```

y el envío de texto con la instrucción

```
lcd.print(cadena);
```

La biblioteca incluye otros comandos que pueden consultarse en (Arduino, 2017c)
https://www.arduino.cc/en/Reference/LiquidCrystalPrint

Un ejemplo del funcionamiento del programa se muestra en la Fig. 33. En este caso, se utilizó un display modelo 1602A (Shenzen Eone Electronics Co, 2006), el cual tiene retroiluminación azul.

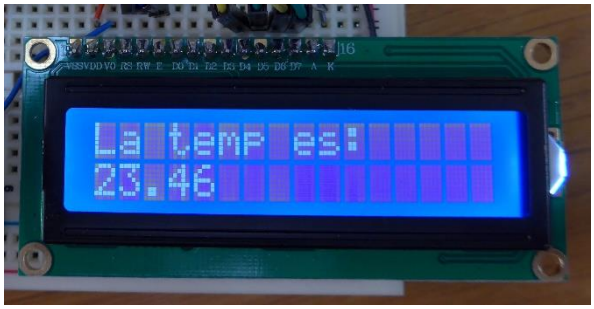


Fig. 33 El display LCD mostrando la temperatura ambiente.

6 BIBLIOTECAS PARA EL MANEJO DE SENSORES

Una de las ventajas de la popularidad del sistema Arduino es que muchos fabricantes de sensores han desarrollado bibliotecas compatibles con esta plataforma. Como ejemplo utilizaremos el sensor digital de temperatura DS18B20 (Maxim Integrated, 2008), el cual es un circuito integrado digital que proporciona un dato de 9 bits a 12 bits que representa la temperatura en grados centígrados. Tiene una exactitud de ±0.5 °C y un tiempo de conversión a 12 bits de 750 ms. Además, es posible conseguir una versión a prueba de agua, la cual se puede sumergir en líquidos cuya temperatura sea inferior a 100 °C. Este dispositivo se comunica mediante un protocolo llamado 1-wire, por ello, es necesario instalar dos bibliotecas: una para la comunicación 1-wire, y otra para manejar el sensor DS18B20.

La biblioteca 1-wire se puede descargar de: https://github.com/PaulStoffregen/OneWire (Paul Stoffregen, 2017) dando clic en el botón que dice Download ZIP. Con esto se descarga el archivo *OneWire-master.zip*.

De la misma manera, se puede descargar la biblioteca del sensor DS18B20 de:

https://github.com/milesburton/Arduino-Temperature-Control-Library (Miles Burton, 2016) dando clic en el botón que dice Download ZIP. Con esto se descarga el archivo *Arduino-Temperature-Control-Library-master.zip.*

Las bibliotecas se instalan usando el menú *Programa->Incluir Librería->Añadir Librería .ZIP*, según se muestra en la Fig. 34.

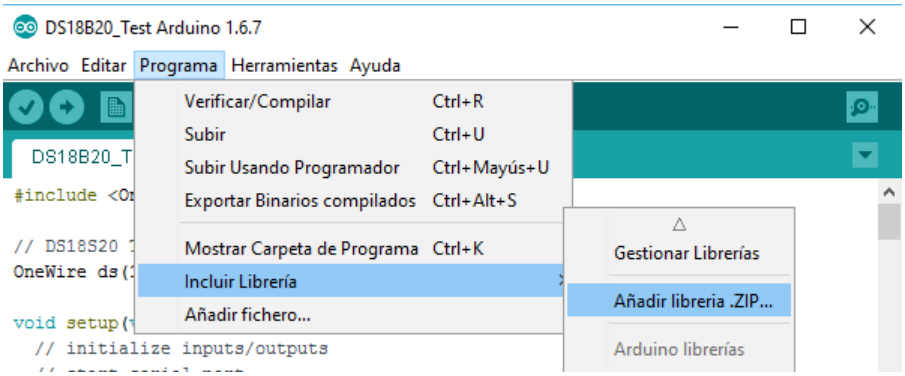


Fig. 34 Menú para instalar una nueva biblioteca en formato ZIP.

Información adicional sobre cómo instalar una biblioteca se puede encontrar en

http://arduino.cc/en/Guide/Libraries (Arduino, 2017b)

El listado del programa se muestra en la Tabla 12. En las primeras líneas se encuentran las sentencias

#include <OneWire.h>

#include <DallasTemperature.h>

```
/* Programa que lee un sensor de temperatura DS18B20 y
  *  despliega el resultado en un display LCD   */
// Se requiere la biblioteca de uso del display
            #include <LiquidCrystal.h>
// Se requiere la biblioteca de comunicación 1-wire, y del sensor
   #include <OneWire.h>
   #include <DallasTemperature.h>
// Se seleciona el pin 8 para lectura del sensor
#define ONE_WIRE_BUS 8

// Se active un controlador de protocolo OneWire.
OneWire oneWire(ONE_WIRE_BUS);

// Se asigna el controlador de protocol OneWire al sensor DS18B20 y se le asigna el nombre de
// sensors
DallasTemperature sensors(&oneWire);

// Se definen los pines que se comunican con el display
            LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
   float temperatura = 0;
// Inicialización
            void setup() {
  // Se inicializa el controlador que se denominó sensors
     sensors.begin();
               lcd.begin(16, 2);   // LCD es de 2 renglones y 16 caracteres
               // Manda un letrero al display
     pinMode(pinSensor, INPUT);
               lcd.print("La temp es:");
}
// Programa principal
            void loop() {
              // Selecciona el renglón 0, caracter 1
              // (Nota: el renglón 1 es el segundo
              lcd.setCursor(0, 1);
     sensors.requestTemperatures(); // Se envía el comando para leer las temperaturas
          lcd.print("      ");
          lcd.setCursor(0, 1);
               lcd.print(sensors.getTempCByIndex(0));  // Se despliega la temperatura del sensor 0
          delay(300);
             }
```

Tabla 12 Programa que lee un sensor de temperatura digital DS18B20 y envía el resultado a un display LCD.

las cuales se requieren para utilizar las funciones de las bibliotecas OneWire y del sensor de temperatura digital DS18B20, respectivamente. Otros detalles del programa se mencionan en los comentarios incrustados en el código. El diagrama de conexiones del circuito que incluye el display LCD y el sensor de temperatura DS18B20 se muestra en la Fig. 35. Observe que el sensor DS18B20 tiene tres cables que corresponden al voltaje de alimentación Vcc (rojo), a la tierra (azul) y a la línea de datos (amarillo) (ver Fig. 36). La resistencia que se conecta entre Vcc (+5 V) y la terminal amarilla del sensor es de 4.7 kΩ. Es posible que los colores pudieran variar dependiendo del proveedor del sensor.

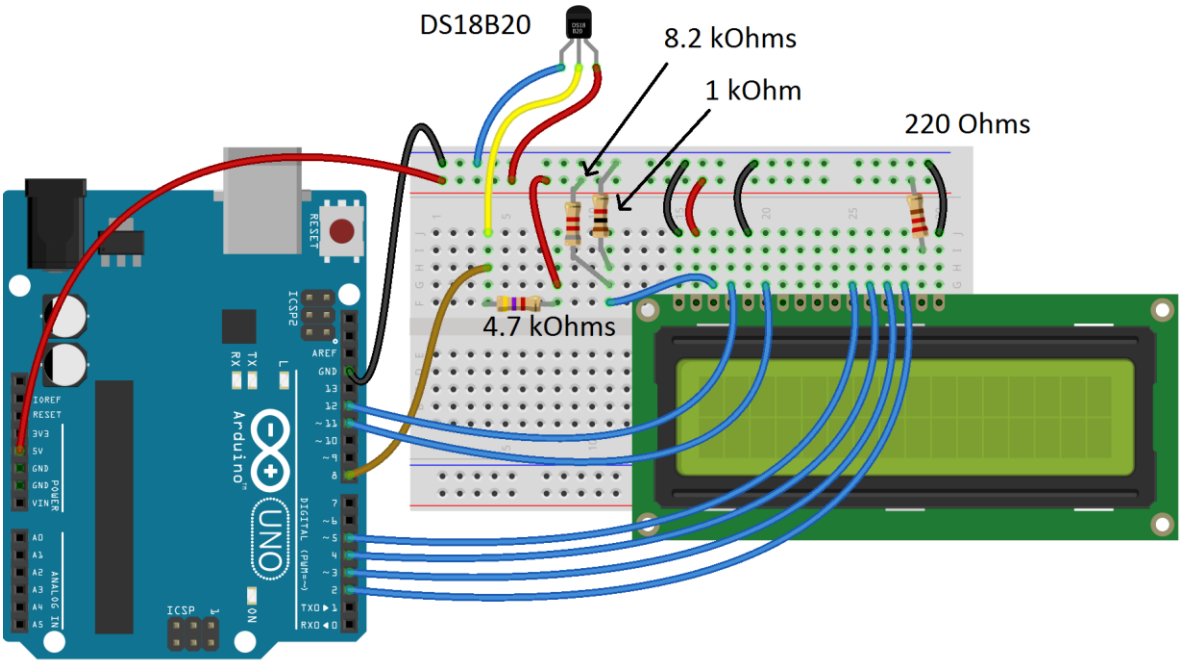


Fig. 35 Diagrama de conexiones del LCD y un sensor de temperatura DS18B20.

Fig. 36 Fotografía del sensor de temperatura DS18B20.

7 IMPLEMENTACIÓN DE UN CRONÓMETRO

Ahora utilizaremos la experiencia adquirida en el manejo de entradas y salidas digitales y la comunicación serial con la PC para implementar un cronómetro. Para ello utilizaremos una clase de sensores ópticos que se denominan optointerruptores. Empleando dos de ellos, es posible iniciar el cronómetro al detectar que un objeto atraviesa frente al primer sensor y detener el cronómetro cuando el objeto atraviesa frente al segundo sensor.

Los opto-interruptores que utilizaremos son del tipo reflexivo (ver Fig. 37A). Consisten de un LED infrarrojo, y un fototransistor. El fototransistor detecta cuando la luz infrarroja que emite el LED es reflejada por un objeto (ver Fig. 37B). En la Fig. 37C se muestra el símbolo electrónico del opto-interruptor, el cual se compone del símbolo de un LED y del símbolo de un fototransistor agrupados.

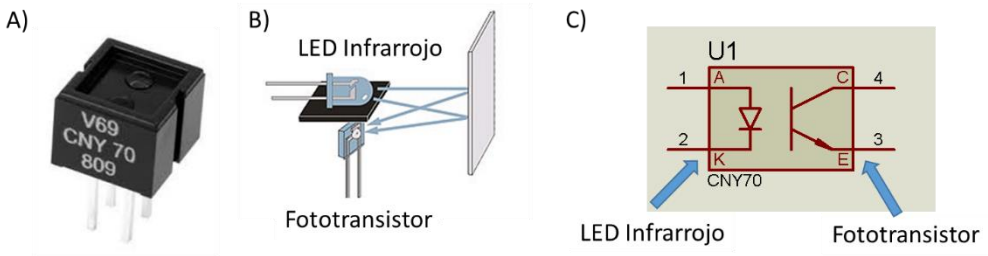


Fig. 37 A) Imagen de un opto-interruptor reflexivo (Vishay, 2012). B) Funcionamiento de un opto-interruptor (MIT Media Laboratory, 2017). C) Símbolo electrónico.

El opto-interruptor requiere dos resistencias para su funcionamiento, según se muestra en la Fig. 38. El LED infrarrojo se encuentra siempre encendido. Cuando no hay algún objeto que refleje la luz infrarroja, el fototransistor se encontrará “apagado” y eso equivale a que se comporte como un circuito abierto (ver Fig. 38A). En esta condición el voltaje de salida es aproximadamente 5 V. Si se coloca frente al opto-interruptor un objeto, éste reflejará la luz infrarroja y el transistor cambiará a “encendido”, lo cual equivale a que se comporte como un circuito cerrado (ver Fig. 38B). En este caso, la salida será aproximadamente 0 V.

En la Fig. 39, se muestra el diagrama de conexiones de los dos opto-interruptores, y de dos LEDs que servirán como comprobación (testigos) de que se han activado los sensores ópticos. Este circuito se deberá conectar a las terminales D2 a D5 de la tarjeta Arduino, según se indica. El código del programa cronómetro se muestra en la Tabla 14.

```
int const pinOpto1 = 2;    // Pin 2 recibe la señal del opto-interruptor 1
int const pinOpto2 = 4;   // Pin 4 recibe la señal del opto-interruptor 2
int const pinLed1 = 3;     // Pin 3 es la salida para el LED 1
int const pinLed2 = 5;     // Pin 5 es la salida para el LED 2
int valorOpto1, valorOpto2;    // Almacenan el estado del opto-interruptor (ON/OFF)
unsigned long tiempo, tiempo1, tiempo2;   //  tiempo1 es el tiempo inicial, tiempo2 es el tiempo final

void setup() {
  pinMode(pinOpto1, INPUT);
  pinMode(pinOpto2, INPUT);
  pinMode(pinLed1, OUTPUT);
  pinMode(pinLed2, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  valorOpto1 = 0;
  while(!valorOpto1){   // Espera a que se active opto1
    valorOpto1 = digitalRead(pinOpto1);   // Lee el primer opto
  }
  while(valorOpto1){
    valorOpto1 = digitalRead(pinOpto1); // Espera a que se desactive opto1
  }
    tiempo1 = millis();    // Guarda el inicial tiempo en milisegundos
    digitalWrite(pinLed1, HIGH);    // Enciende el LED de arranque
    valorOpto2 = 0;
    while(!valorOpto2) {
    valorOpto2 = digitalRead(pinOpto2);  // Espera a que se active opto2
  }
  while(valorOpto2){
    valorOpto2 = digitalRead(pinOpto2);  // Espera a que se desactive opto2
  }
    tiempo2 = millis();     // Guarda el tiempo final en milisegundos
    digitalWrite(pinLed2, HIGH);   // Enciende el LED de paro

  if(tiempo2 > tiempo1)  {
    tiempo = tiempo2 - tiempo1;
    Serial.print("Tiempo: ");
    Serial.println(tiempo);
    delay(1000);   // Espera un segundo
    digitalWrite(pinLed1, LOW);    // Apaga el led de arranque
    digitalWrite(pinLed2, LOW);    // Apaga el led de paro
  }
}
```

Tabla 13 Código del cronómetro optoelectrónico.

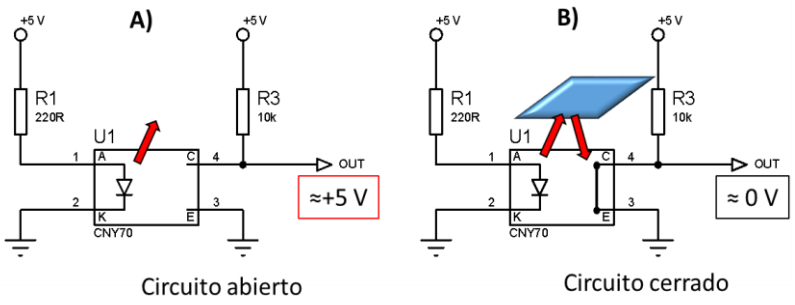


Fig. 38 A) Operación cuando no hay un objeto reflejante. B) Operación cuando sí hay un objeto reflejante.

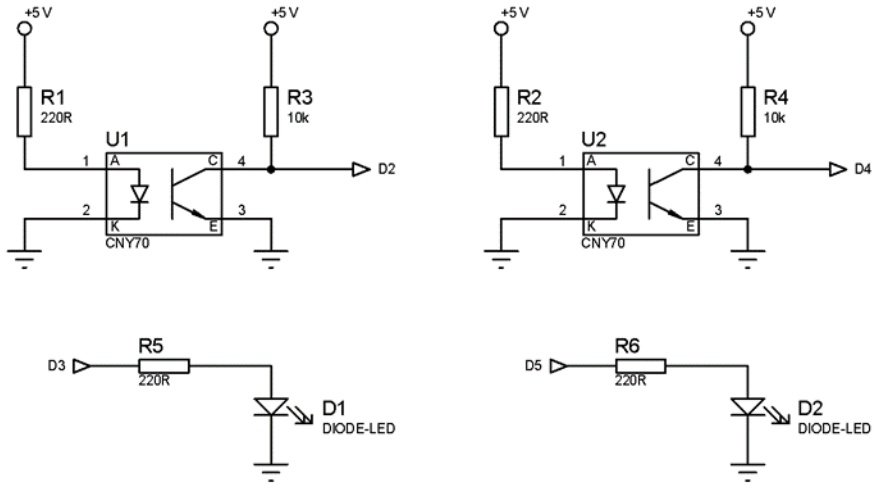


Fig. 39 Diagrama electrónico de los sensores y LEDs para realizar un cronómetro. Se indican las conexiones D2 a D5 hacia la tarjeta Arduino.

El programa espera a que se active el opto-interruptor 1 y cuando se desactiva, lo cual indica que el objeto ha dejado de estar enfrente, se lee el tiempo inicial mediante la instrucción

```
tiempo1=mills();
```

La función *mills()* proporciona un número en milisegundos que representa el tiempo que lleva la tarjeta Arduino ejecutando el programa actual. La misma función se utiliza para detectar el paso de un objeto sobre el opto-interruptor 2 y se almacena en la variable *tiempo2*. La diferencia entre *tiempo1* y *tiempo2*, es el tiempo cronometrado y se envía a la PC por el puerto serial. El resultado se puede examinar utilizando el Monitor Serie del ambiente Arduino (ver Fig. 40).

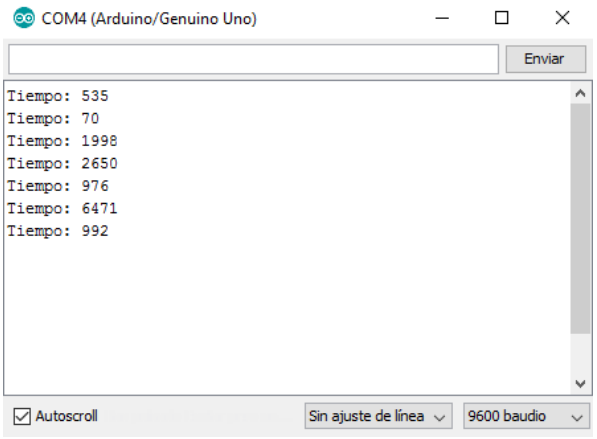


Fig. 40 Ventana del Monitor Serie recibiendo datos del cronómetro.

APÉNDICE A. INSTALACIÓN DEL AMBIENTE DE PROGRAMACIÓN DEL ARDUINO

En primer lugar, debemos entrar a la página web www.arduino.cc, y seleccionar la opción *Software* (ver Fig. 41). Esto nos llevará a la página de descargas que se muestra en la Fig. 42.

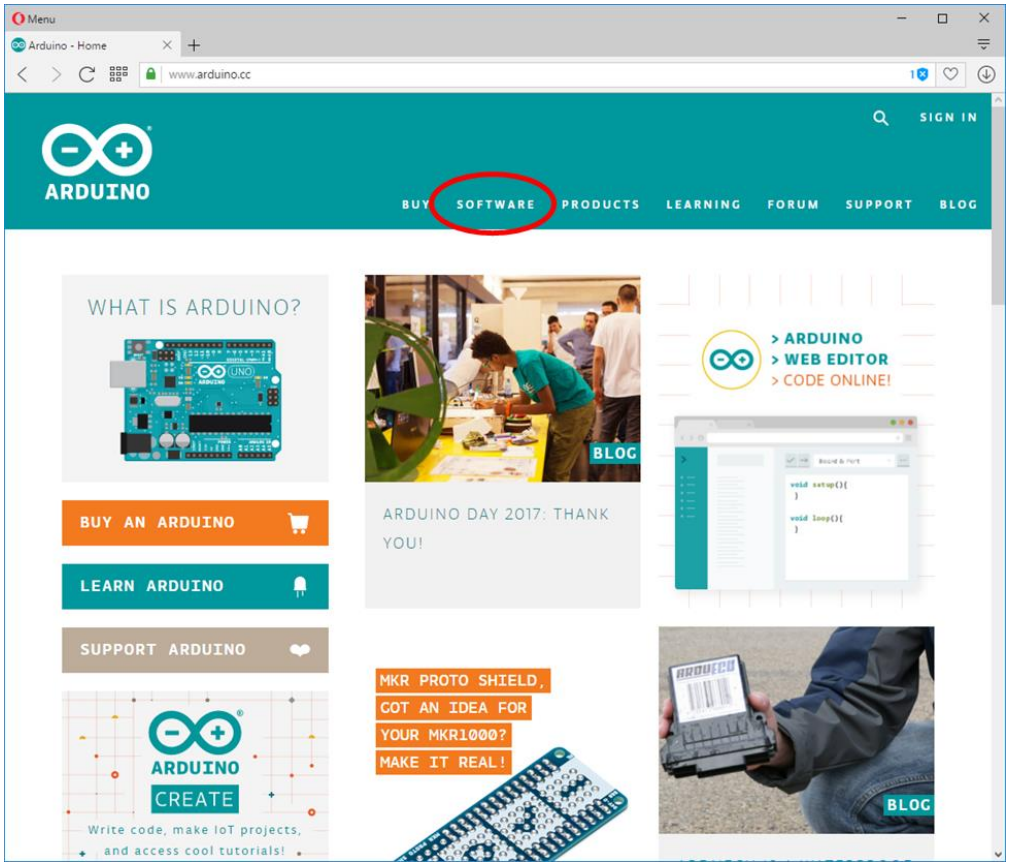


Fig. 41 Página principal del sitio oficial de Arduino (Arduino, 2017e).

Si nuestra computadora tiene instalado el sistema operativo Windows, podemos entonces dar clic en la opción *Windows Installer*. Eso nos llevará a una página donde se nos pregunta si deseamos contribuir económicamente con los desarrolladores, si no se desea contribuir en ese momento, basta con seleccionar la opción *Just Download* (ver Fig. 43).

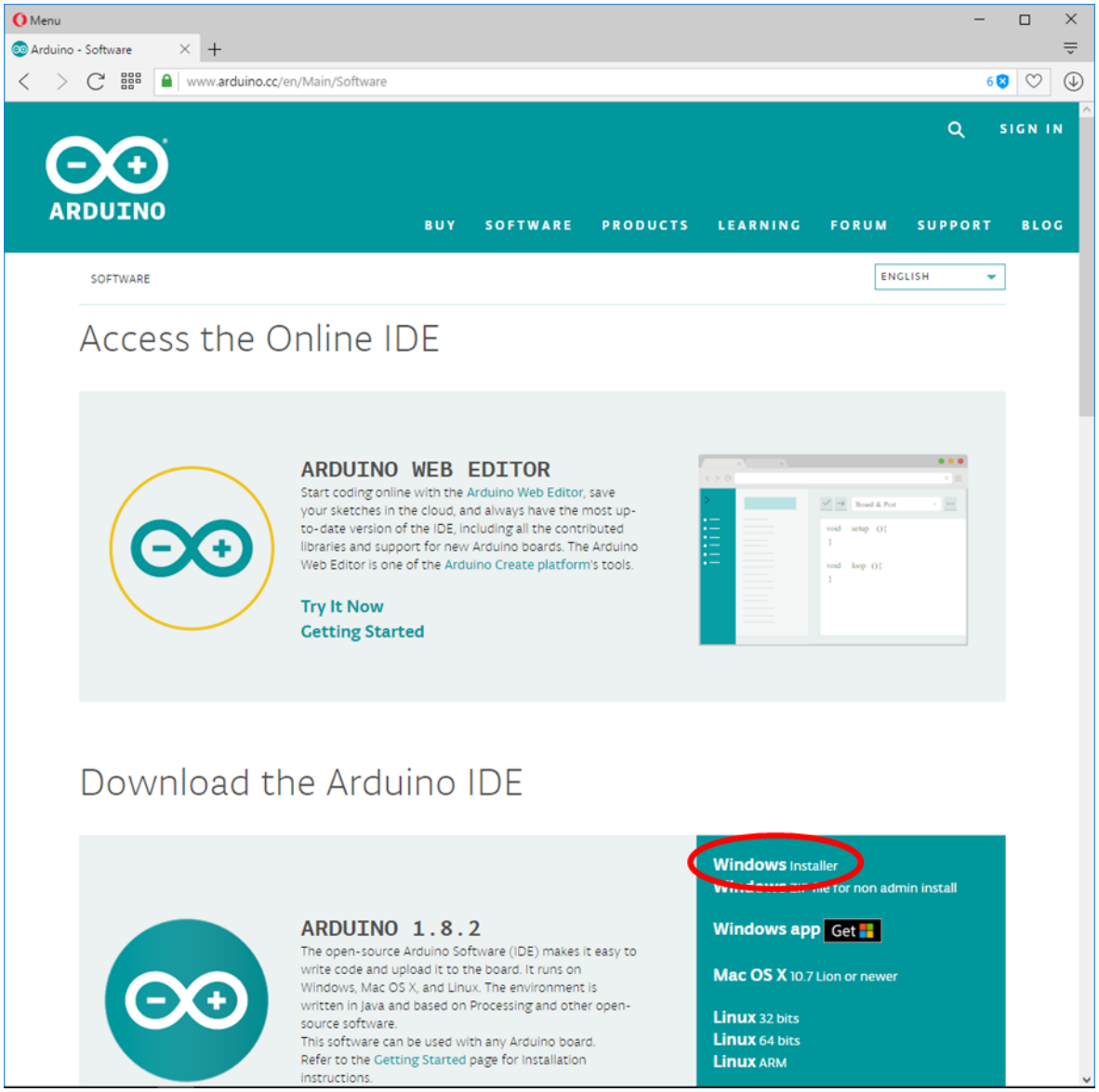


Fig. 42 Página de descargas del ambiente de programación del Arduino (Arduino, 2017f).

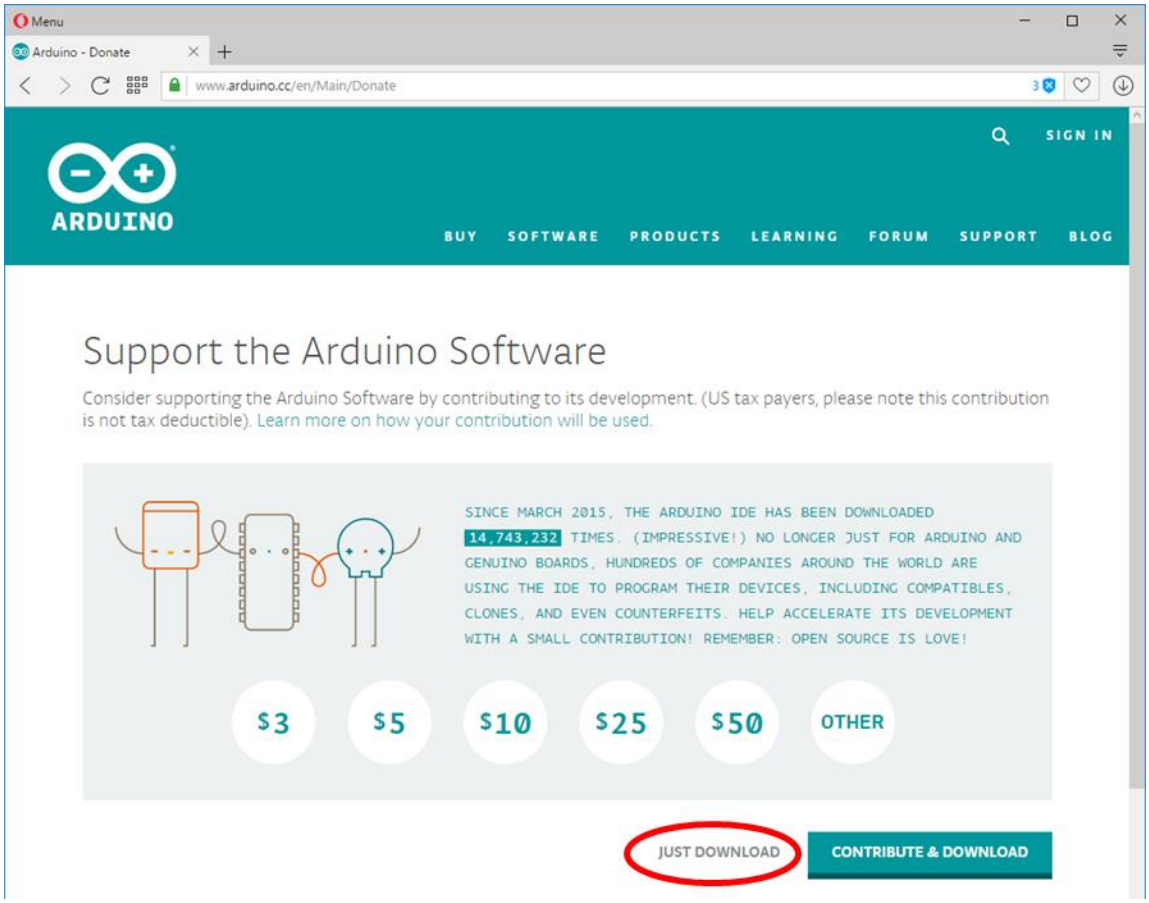


Fig. 43 Ventana de solicitud de contribución económica (Arduino, 2017g).

APÉNDICE B. PROGRAMACIÓN DE FUNCIONES

Para explicar el concepto de una función en programación, supongamos que queremos calcular la raíz cuadrada de un número, y para ello se crea una función llamada *squareroot().* Dentro de los paréntesis debemos indicar el número del cual queremos obtener la raíz cuadrada. El resultado del cálculo es otro número, por lo que la función puede definirse como

int squareroot(int)

De esta manera se declara una función a la cual se le pasa como parámetro un número entero (*int*) y entrega como resultado otro número entero.

Un ejemplo de utilización sería:

```
int a;

a=squareroot(16);
```

donde la variable entera **a** recibe el dato que entrega la función *squareroot().*

La definición completa de la función podría ser:

```
int squareroot(int x) {    // El valor que recibe la función se almacena en la variable x
int result;          // Define una variable temporal
result=x^0.5;
return result;   // regresa el valor almacenado en result
}
```

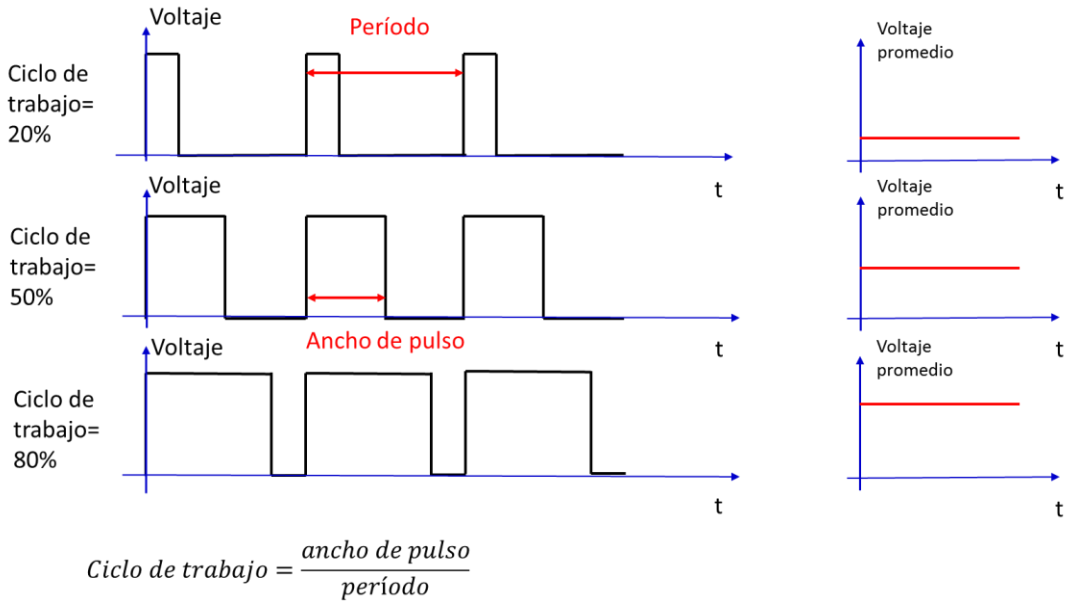
Las funciones se deben declarar afuera de los bloques *setup* y *loop.*

Se puede encontrar más información en:

https://www.arduino.cc/en/Reference/FunctionDeclaration (Arduino, 2017a)

APÉNDICE C. PWM (PULSE WIDTH MODULATION)

La modulación por ancho de pulso o PWM por sus siglas en inglés, consiste en la generación de una señal cuadrada de voltaje con una frecuencia constante, pero cuya relación entre el tiempo que la señal está en alto con respecto a bajo cambia. En la Fig. 44 se muestran tres ejemplos de una señal PWM. El período, es decir el tiempo que tarda la señal en repetir su patrón, es constante. Sin embargo, el ancho de pulso, es decir el tiempo que la señal permanece en un valor alto, cambia. La relación entre el tiempo que el pulso está en alto, y la duración del período se denomina ciclo de trabajo. Si esta señal PWM es aplicada a un sistema lento (es decir, que no puede cambiar de encendido a apagado a la misma velocidad de la señal PWM), entonces el sistema interpretará la señal como un voltaje promedio, tal como se muestra en las gráficas del lado derecho de la Fig. 44. De esta manera se está produciendo un efecto equivalente al de una salida analógica continua.



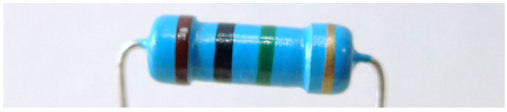
$$Ciclo\ de\ trabajo = \frac{ancho\ de\ pulso}{período}$$

Fig. 44 Ejemplos de una señal de modulación por ancho de pulso.

Información adicional se puede consultar en

https://www.arduino.cc/en/Tutorial/PWM (Arduino, 2017d)

APÉNDICE D. TABLA DE COLORES DE RESISTENCIAS



Color	1ra. banda	2da. banda	3ra. banda	Tolerancia
Negro	0	0	x 10^0	
Café	1	1	x 10^1	1 %
Rojo	2	2	x 10^2	2 %
Naranja	3	3	x 10^3	
Amarillo	4	4	x 10^4	
Verde	5	5	x 10^5	
Azul	6	6	x 10^6	
Violeta	7	7	x 10^7	
Gris	8	8	x 10^8	
Blanco	9	9	x 10^9	
				Dorado 5%
				Plata 10%

Tabla 15 Tabla de colores de las resistencias.

APÉNDICE E. CÓMO DESCOMPRIMIR UN ARCHIVO .ZIP

Para descomprimir un archivo .zip se pueden utilizar las funciones propias del sistema operativo o bien tener instalado un programa que maneje archivos comprimidos. Para esta última opción se sugiere instalar el programa WinRAR, descargándolo de la siguiente liga:

http://www.rarlab.com/download.htm (RARLAB, 2017)

Se debe escoger la versión de acuerdo al idioma y al tipo de sistema operativo (32 bits o 64 bits). En caso de duda se puede instalar la versión de 32 bits. Como ejemplo, hemos descargado la versión 5.4 en español de 64 bits:

winrar-x64-540es.exe

Se instala con las opciones que marca el programa por omisión. El programa activa unas opciones que aparecerán en el menú contextual del sistema operativo (es decir, las que aparecen al hacer clic con el botón derecho del ratón).

Ahora es posible seleccionar el archivo que queremos descomprimir, por ejemplo *CoolTerm_Win.zip*, y usando el botón derecho del ratón aparece un menú donde se selecciona la opción extraer en CoolTerm_Win\ como se muestra en la Fig. 45. Esto crea una carpeta donde quedará descomprimido el contenido del archivo .zip.

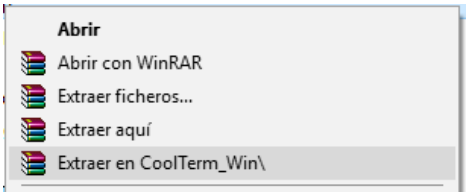


Fig. 45 Menú contextual (botón derecho del ratón) para extraer el contenido de un archivo .ZIP en una carpeta.

Si se desea, se puede mover la carpeta a otra ubicación, o crear un acceso directo del programa CoolTerm.exe (mediante el menú contextual, botón derecho del ratón) y moverlo al escritorio. En otro caso, basta con dar doble clic en *CoolTerm.exe,* para abrir el programa.

APÉNDICE F. CÁLCULO DE LA RESISTENCIA ASOCIADA A UN LED

Para calcular el valor adecuado de la resistencia asociada al LED, es necesario conocer dos valores: el voltaje típico de operación del LED y la corriente típica de operación. Valores comunes son 2.2 V para el voltaje de operación y 10 mA para la corriente. De esta manera, si se considera un voltaje de alimentación de 5 V, la resistencia estimada es de 280 Ω, según se muestra en la Fig. 46. En la práctica es común utilizar resistencias de 220 Ω, o 330 Ω; aunque 270 Ω es un valor más cercano al calculado y también corresponde a un valor comercial. El valor de una resistencia se identifica mediante unas bandas de colores (ver Apéndice C).

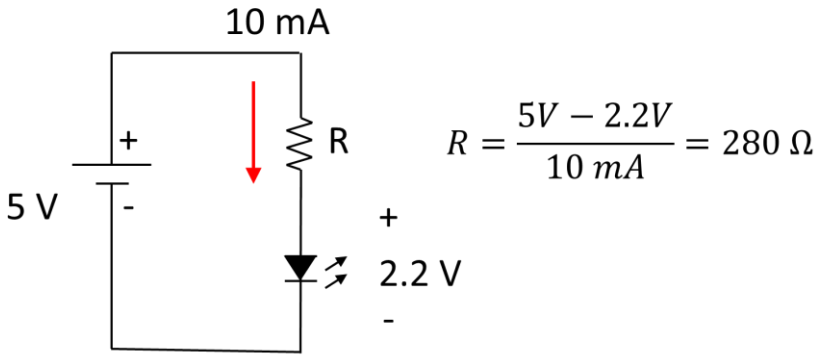


Fig. 46 Cálculo de la resistencia asociada a un LED.

APÉNDICE G. LISTA DE MATERIAL

Para la realización de los experimentos sugeridos en este tutorial se requieren los siguientes componentes:

Cantidad	Descripción
1	Tarjeta Arduino UNO con cable USB
1	Pantalla LCD 16x2 caracteres retroiluminada azul
1	Protoboard mediana
1	Potenciómetro mini 10 k
2	Jumper 6" macho/macho paq 10
2	Led 5 mm rojo
2	CNY70 Interruptor optoreflexivo
1	DS18B20 Sensor de temperatura One-wire sumergible
1	Circuito Integrado LM35DZ Sensor de temperatura
1	Push button
1	Tira 36 pines sencilla
4	Resistencias de carbón 220 Ohms 5%
2	Resistencias de carbón 10 kOhms 5%
1	Resistencia de carbón 1 kOhm 5%
1	Resistencia de carbón 8.2 kOhms 5%
1	Resistencia de carbón 4.7 kOhms 5%

Referencias bibliográficas

Arduino (2017a). **Functions** [Internet], Arduino AG. Disponible desde: <https://www.arduino.cc/en/Reference/FunctionDeclaration>[Acceso el 7 de abril de 2017]

Arduino (2017b). **Installing Additional Arduino Libraries** [Internet], Arduino AG. Disponible desde: <https://www.arduino.cc/en/Guide/Libraries>[Acceso el 7 de abril de 2017]

Arduino (2017c). **LiquidCrystal Library** [Internet], Arduino AG. Disponible desde: <https://www.arduino.cc/en/Reference/LiquidCrystal>[Acceso el 7 de abril de 2017]

Arduino (2017d). **PWM** [Internet], Arduino AG. Disponible desde: <https://www.arduino.cc/en/Tutorial/PWM>[Acceso el 7 de abril de 2017]

Arduino (2017e). **Arduino** [Internet], Arduino AG. Disponible desde: <https://www.arduino.cc>[Acceso el 7 de abril 2017]

Arduino (2017f). **Software** [Internet], Arduino AG. Disponible desde: <https://www.arduino.cc/en/Main/Software> [Acceso el 7 de abril de 2017]

Arduino (2017g). **Support the Arduino Software** [Internet], Arduino AG. Disponible desde: <https://www.arduino.cc/en/Main/Donate>[Acceso el 7 de abril de 2017]

Instructables (2015). **LM35 Temperature Sensor** [Internet], Autodesk, Inc. Disponible desde: <http://www.instructables.com/id/LM35-Temperature-Sensor/>[Acceso el 7 de abril de 2017]

Maxim Integrated (2008) . **DS18B20 Programmable Resolution 1-Wire Digital Thermometer datasheet** [Internet]. Maxim Integrated. Disponible desde <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>[Acceso el 24 de mayo de 2017]

Miles Burton (2016). **Arduino-Temperature-Control-Library** [Internet], GitHub, Inc. Disponible desde: <https://github.com/milesburton/Arduino-Temperature-Control-Library>[Acceso el 7 de abril de 2017]

MIT Media Laboratory (2017). **Light reflective IR LED image** [Internet], MIT Media Laboratory. Disponible desde: <http://learning.media.mit.edu/projects/gogo/documents/images/light_reflective_IR_LED.jpg> [Acceso el 24 de mayo de 2017]

Paul Stoffregen (2017). **One Wire Library** [Internet], GitHub, Inc. Disponible desde: <https://github.com/PaulStoffregen/OneWire>[Acceso el 7 de abril de 2017]

RARLAB (2017). **WinRAR and RAR archiver downloads** [Internet], Berlin, Germany, win.rar GmbH. Disponible desde: <http://www.rarlab.com/download.htm>[Acceso el 7 de abril de 2017]

Roger Meier (2017). **Roger Meier's Freeware** [Internet], Roger Meier. Disponible desde: <http://freeware.the-meiers.org>[Acceso el 7 de abril de 2017]

Shenzen Eone Electronics Co, Ltd. (2006). **Specification for LCD Module 1602A-1 V1.2 datasheet** [Internet]**.** Openhacks.com. Disponible desde: <https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf>[Acceso el 24 de mayo de 2017]

Sparkfun (2017). **Push button datasheet** [Internet], Niwot, Colorado, USA. Sparkfun Electronics. Disponible desde: https://www.sparkfun.com/datasheets/Components/General/00097.jpg>[Acceso el 21 de febrero de 2017]

Texas Instruments, I. (2016). **LM35 Precision Centigrade Temperature Sensors datasheet** [Internet], Texas Instruments Inc. Disponible desde: < www.ti.com/lit/ds/symlink/lm35.pdf>[Acceso el 24 de mayo de 2017]

Vishay Semiconductors (2012). **Reflective Optical Sensor with Transistor Output datasheet.** [Internet] Vishay Semiconductors. Disponible desde: <https://www.vishay.com/docs/83751/cny70.pdf>[Acceso el 24 de mayo de 2017]

Was a bee. (2017). **LED**. [Internet], wikipedia.org. Disponible desde: <https://en.wikipedia.org/wiki/LED_circuit#/media/File:%2B-_of_LED_2.svg>[Acceso el 21 de febrero de 2017]