e-GO! Library Español

COMO LO VIO EN TV

¡1250 LIBROS PARA LLEVAR EN SU BOLSILLO!

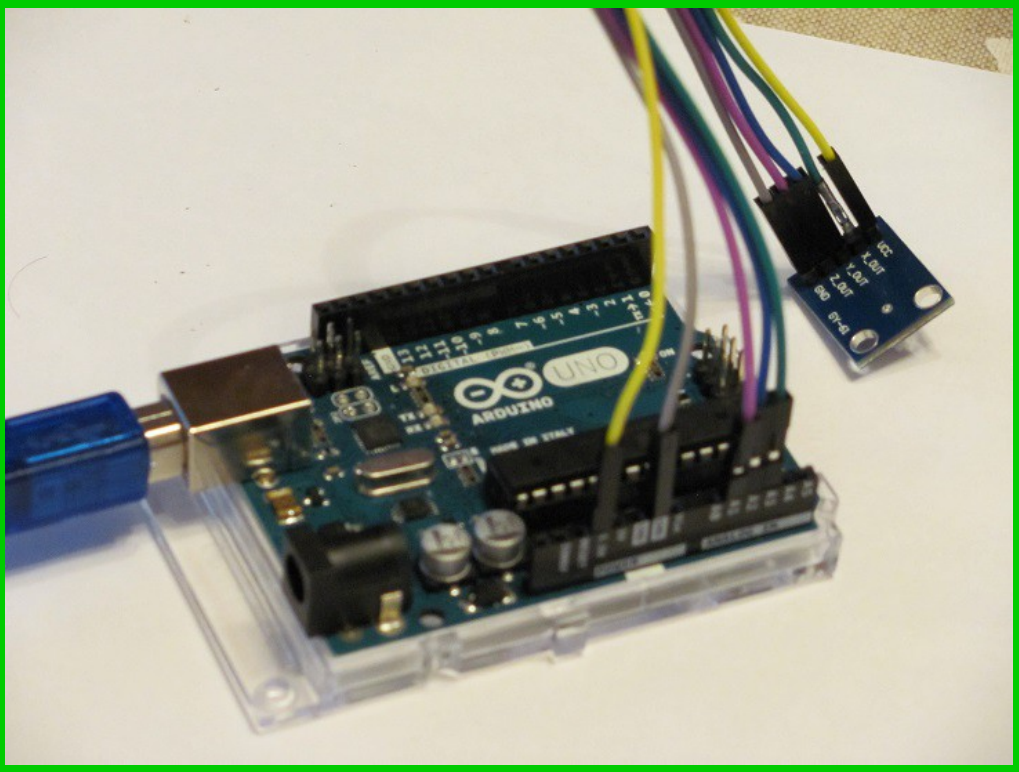
La velocidad, comodidad y movilidad son suyas. El e-GO! Library Español es una forma innovadora para tener y mantener un suministro fresco y abundante de grandes títulos. Es el mejor entretenimiento y fácil de obtener. El e-GO! Library Español es una unidad flash de memoria USB que pone a miles de los mejores libros de la actualidad su bolsillo!

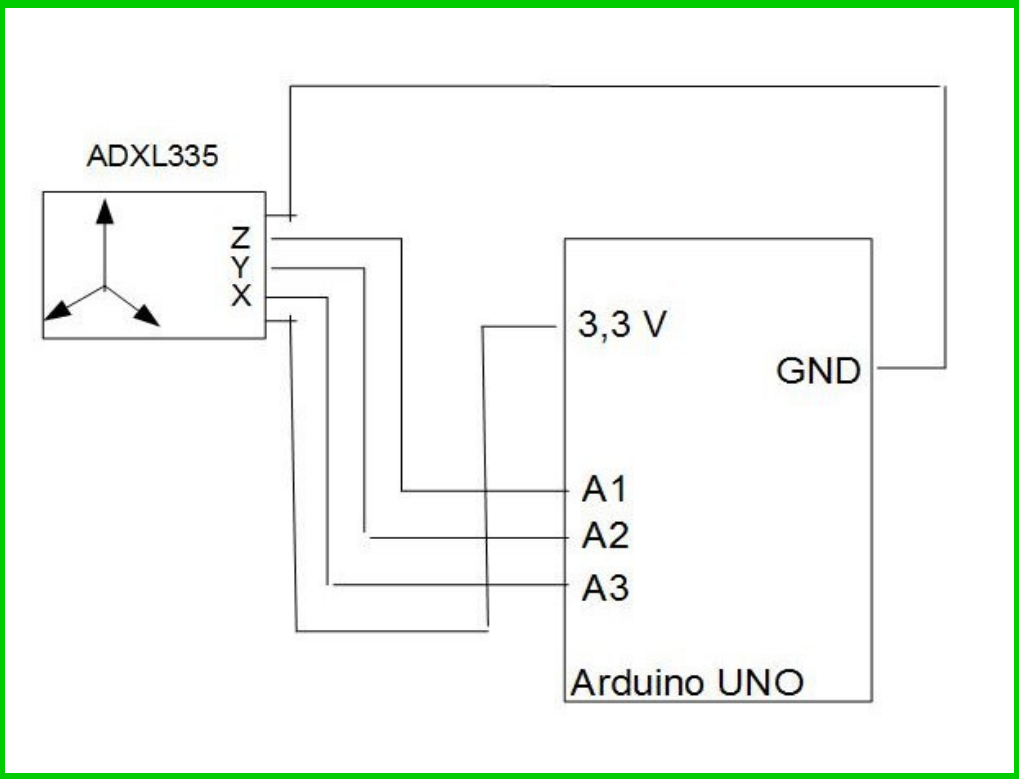
Cargue su Kindle, iPad, Nook, o cualquier dispositivo con una variedad de ficción y no ficción. En su tiempo libre, elija entre sus temas, títulos y autores independientes favoritos y categorías como: romance, ciencia ficción, misterios, finanzas, biografías, negocios y muchos más.

- ✓ **1,000 LIBROS** indipendientes más populares
- ✓ **BONO**- 250 títulos clásicos
- ✓ **CONTENIDO ÚNICO** / Autores independientes
- ✓ **LLAVE USB PRECARGADA** de 4GB
- ✓ **SIRVE CON TODOS** los lectores y dispositivos
- ✓ **IDEAL** para viajar
- ✓ **AHORRA** innumberables horas de Descargas
- ✓ **EL REGALO** Perfecto

VER MÁS

CONCEPTOS BÁSICOS DE ARDUINO





AGUSTÍN GRAU CARLES

CONCEPTOS BÁSICOS DE ARDUINO

AGUSTÍN GRAU CARLES

agcarles25@yahoo.es
http://www.academiabic.netau.net

Prólogo

Aún puedo recordar como todavía en los años noventa del pasado siglo el convertidor analógico digital y el multicanal eran dos dispositivos electrónicos bastante voluminosos en equipos de medida de radiaciones ionizantes. Hoy en día, una placa que cabe en la palma de la mano, como Arduino, puede realizar ambas funciones con una sencillez fuera de lo común, poniendo al alcance del público en general posibilidades que antes estaban únicamente limitadas a personal especializado. Esta posibilidad de disfrutar de la electrónica sin demasiadas complicaciones es el mayor logro del Proyecto Arduino. Si a esto añadimos la disponibilidad de componentes electrónicos a un precio sin igual, como la que se puede conseguir con los kits de iniciación Arduino, el disfrute en diseñar proyectos Arduino propios está garantizado.

Una de las principales dificultades de los iniciados a la electrónica es saber interpretar los esquemas de circuitos, conocer como se pueden montar sobre la placa de pruebas o saber como modificarlos en función de las necesidades que aparezcan dentro de un experimento. Creo que este objetivo se ha conseguido sobradamente en este manual.

Otra dificultad para los iniciados proviene de la programación en C. Todo aquel que pretenda comprender el funcionamiento de Arduino, deberá familiarizarse antes con lo que son los conceptos básicos de programación en C (operadores, decisiones, bucles, funciones, arrays, entradas y salidas). En este caso, puesto que los programas o sketchs son breves e intuitivos, el profano también puede aprender los rudimentos de C conforme van surgiendo las distintas necesidades.

En resumen, siempre podemos acceder a Arduino sin demasiados conocimientos iniciales. Con su placa Arduino aprenderá a diseñar rápidamente sus propios proyectos con una facilidad asombrosa.

Agustin Grau Carles

Índice General

Capítulo 1

Antes de empezar

Adquirida la placa Arduino UNO R3, o cualquiera de de sus variantes clónicas (Kuman, Sunfounder, XCSOURCE), debemos tener presentes ciertos aspectos sobre su funcionamiento. La placa Arduino debe hallarse conectada a un PC mediante un puerto de serie USB. Por otro lado, ésta debe también encontrarse unida a otra placa (inicialmente de pruebas) sobre la que se han dispuesto las diferentes componentes electrónicas. En este capítulo indicaremos como debe instalarse el software en un PC con Windows 10, los fundamentos del funcionamiento de una placa electrónica de pruebas y ciertas generalidades sobre la placa Arduino R3.

1.1. INSTALACIÓN DEL SOFTWARE NECESARIO EN WINDOWS 10

El software IDE (Integrated Development Environment) de Arduino, para cualquiera de los sistemas operativos más comunes (Windows, MacOS X o Linux), puede descargarse del sitio web: https://www.arduino.cc/en/Main/Software (Fig.1.1). En caso de que se tome como opción Windows

Fig.1.1. Pagina de descarga del software Arduino

Instaler, se abrirá una página de donación económica al proyecto Arduino. Si únicamente desea descargarse la aplicación, elegir la opción 'Just Download'. Una vez realizada la descarga, abrir la

carpeta *Descargas* del *Explorador de Archivos* y ejecutar la aplicación para Windows *arduino-1.6.12-windows*. El programa de instalación le irá guiando a lo largo del proceso. Finalizada la instalación, si dispone de una placa Arduino, podrá comprobar el correcto funcionamiento de ésta, conectándola a un puerto USB de su PC. Para ello, después de realizar un doble click sobre el icono Arduino, que se encuentra sobre su escritorio, aparecerá la ventana de la Fig.1.2.

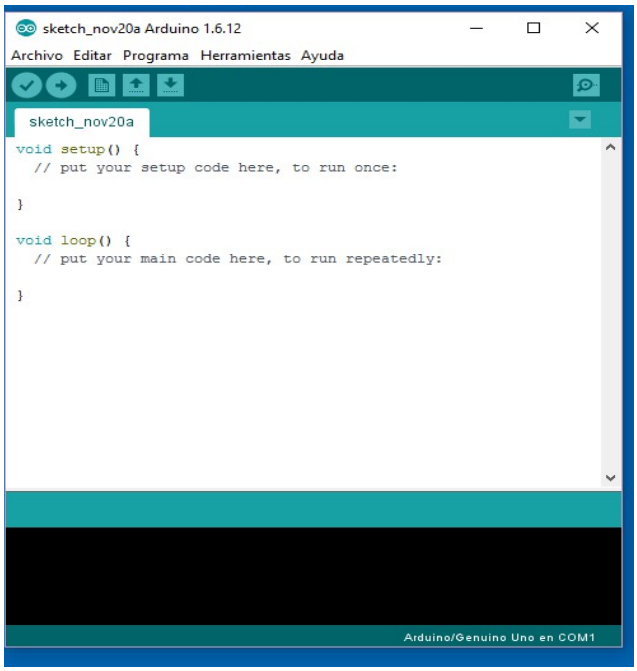


Fig.1.2. Ventana principal del programa Arduino

Seleccionando la opción del menú *Herramientas*, puede comprobarse que la instalación ha sido correcta, puesto que aparecerán, en caso de utilizarse la placa Arduino Original, las anotaciones *Placa: Arduino/Genuino Uno* y *Puerto: COM1 (Arduino/Genuino Uno)*.

No obstante, es bastante común que al compilar y subir (botones circulares de color verde debajo del menú) cualquier programa como el de la Fig. 1.2, aparezca el mensaje de error: *avrdude: ser_open(): can't open device "\\.\COM1"*. Esto tiene que ver con el hecho de que el número de puerto de serie del USB asignado no es, como debería ser, COM1. Para subsanar este error, y poder operar con Arduino, hay que abrir la carpeta *Administrador de dispositivos* de Windows 10, tal y como aparece en la Fig.1.3. A continuación, debe hacerse doble click sobre *Arduino Uno (COM3)*, seleccionar la pestaña de *Configuración de puerto*, apretar el botón *Opciones avanzadas*, cambiar el número de puerto COM3 por COM1 y desconectar y conectar de nuevo la placa al puerto USB (o apretar el botón reset de la placa Arduino).

1.2. LA PLACA DE PRUEBAS

En las placas electrónicas comerciales, las componentes suelen encontrarse soldadas, para mayor estabilidad de las mismas. Sin embargo, en una fase inicial se montan sobre una placa de pruebas como la de la Fig.1.4. Se trata, como puede apreciarse, de un tablero en el que los orificios unidos por la línea azul se encuentran conectados eléctricamente de manera interna. En la Fig.1.4 se han

dibujado líneas azules únicamente en las 5 primeras filas, aunque las restantes filas de la placa se encuentran también unidas de la misma forma.

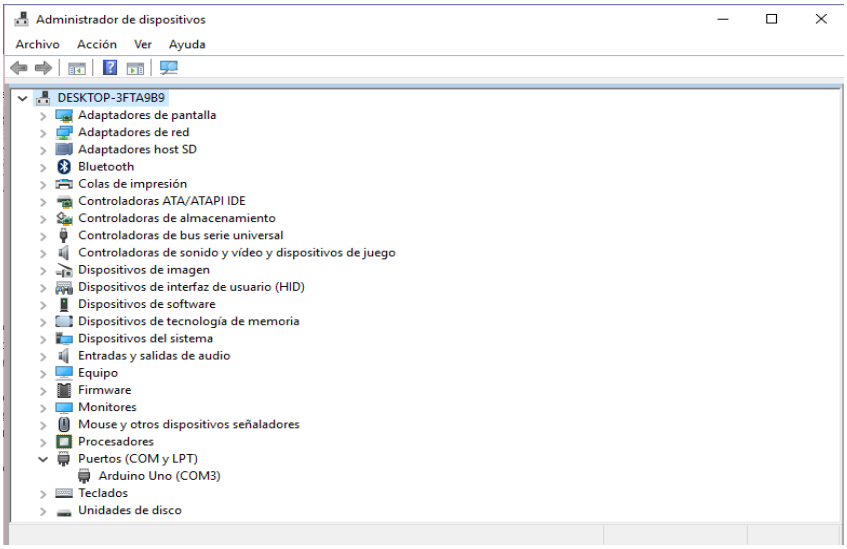


Fig.1.3. Ventana del Administrador de dispositivos de Windows 10

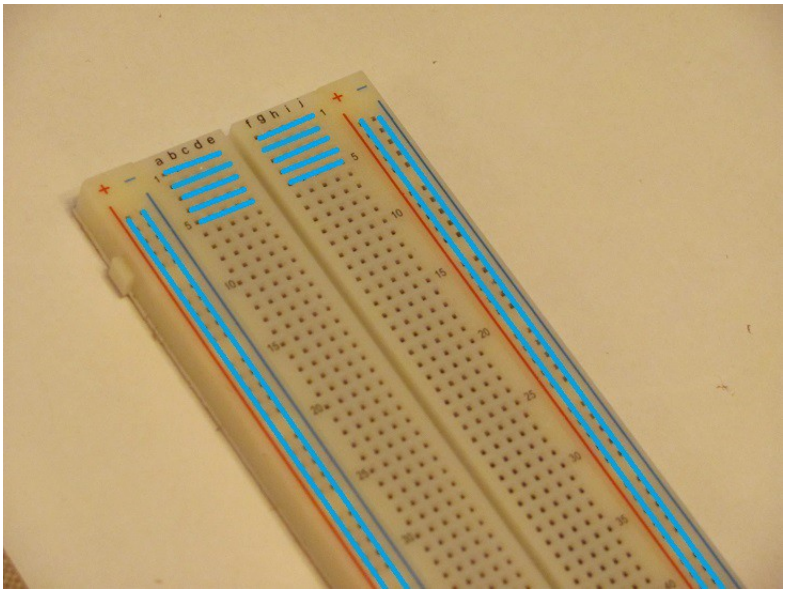


Fig.1.4. Placa de pruebas. Los orificios unidos por la linea azul se encuentran conectados internamente.

1.3. COMPONENTES ELECTRÓNICOS

Aunque los componentes electrónicos de cada proyecto pueden adquirirse por separado, es aconsejable hacerse con kits de iniciación o starter kits. En particular, se aconseja la compra del *Sunfounder Super Kit V2.0 for Arduino* y el *Kuman Super Starter Kit*. El kit de Sunfounder trae consigo un manual que describe los proyectos. Los programas o sketchs pueden descargarse del sitio web: http://www.sunfounder.com, pulsando la opción LEARN/Get Tutorials del menú. El *Kuman Super Starter Kit,* por su lado, incluye un CD con los sketchs de los proyectos.

Con los kits de iniciación pueden aprenderse perfectamente las bases sobre el manejo de Arduino. Se ha creído necesario, no obstante, complementar los proyectos allí incluidos con proyectos adicionales que afiancen los conocimientos ya adquiridos.

1.4. GENERALIDADES SOBRE LA PLACA ARDUINO

La Fig.1.5 muestra una placa original Arduino Uno R3. En ella pueden apreciarse, a derecha e izquierda, dos bandas de pines. En la de la derecha se encuentran 14 pines digitales (numerados de 0 a 13), los cuales sólo pueden suministrar voltajes con valores HIGH y LOW (es decir 1 y 0), de 5 y 0 V. Junto al pin número 13 existe un pin de conexión a tierra (GND). En la banda izquierda se encuentran 6 pines analógicos, numerados de A0 a A5, junto a dos pines de potencia, de 5 y 3,5 V, y dos de conexión a tierra (GND).

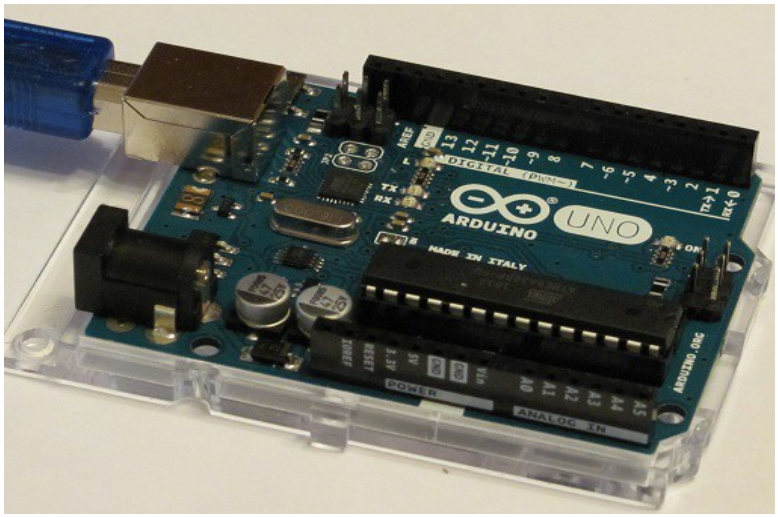


Fig.1.5. Placa Arduino Uno R3

Capítulo 2

Trabajando con LEDs

El diodo emisor de luz o LED (Light Emitting Diode) es un dispositivo electrónico de tipo semiconductor capaz de emitir luz. En los proyectos de este capítulo emplearemos dos tipos de LED: el usual con dos patas de conexión (Fig.2.1a) y el RGB con cuatro patas de conexión (Fig.2.1b), que permite obtener luz de distinto color según el voltaje que se aplique a tres de sus patas frente a la cuarta de conexión a tierra.

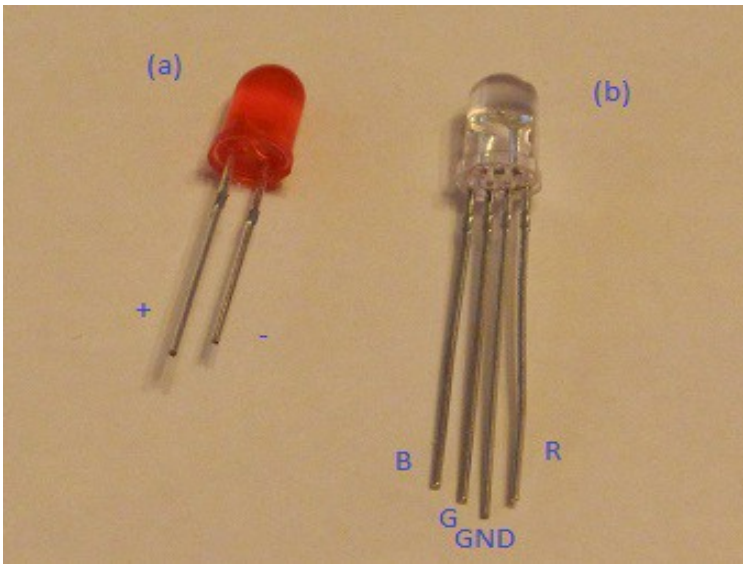


Fig.2.1. LEDs (a) usual, (b) RGB

2.1. PROYECTO 1: LED INTERMITENTE

Si se desea iluminar un LED utilizando la placa Arduino como fuente de alimentación, hay que tener en cuenta que los 5 V que proporciona Arduino son demasiados para una caída de tensión de unos 1,7 V que tienen los LEDs usuales. Es necesario, por tanto añadir una resistencia en serie al circuito para que el LED soporte los 5 V de suministrados. La pregunta que surge es: ¿Cómo calculo la resistencia necesaria?

En la Fig.2.2 se muestra el circuito correspondiente, del que deseamos calcular la resistencia R. Por tratarse de dos dispositivos en serie, sabemos que la caída de potencial total (5 V) será igual a la suma de las caídas de potencial de la resistencia (V2) y del LED (V1). Por otro lado, sabemos por la ley de Ohm que la caída de potencial de la resistencia es el producto de la intensidad de

corriente (0,015 A para el LED) y la resistencia (V2=I×R). Por tanto, 5 = 1,7 + R× 0,015. De donde, despejando R, se obtiene R = 220 Ω.

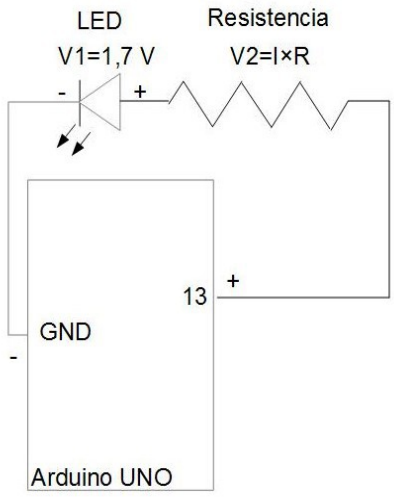


Fig.2.2. Circuito de LED y resistencia conectados a una placa Arduino

Siguiendo el esquema de la Fig.2.2 es muy fácil realizar el montaje en la placa de pruebas. Para ello conectamos el pin digital número 13 de la placa Arduino (que debe encontrarse desconectada del PC) con la resistencia de 220 Ω. El otro extremo de la resistencia lo unimos a la parte positiva del LED, que es la pata más larga, como puede apreciarse en la Fig.2.1a. Por su parte, la pata negativa del LED se conecta mediante un cable a tierra (GND) en la placa Arduino. El resultado final debe ser similar al de la Fig.2.3.

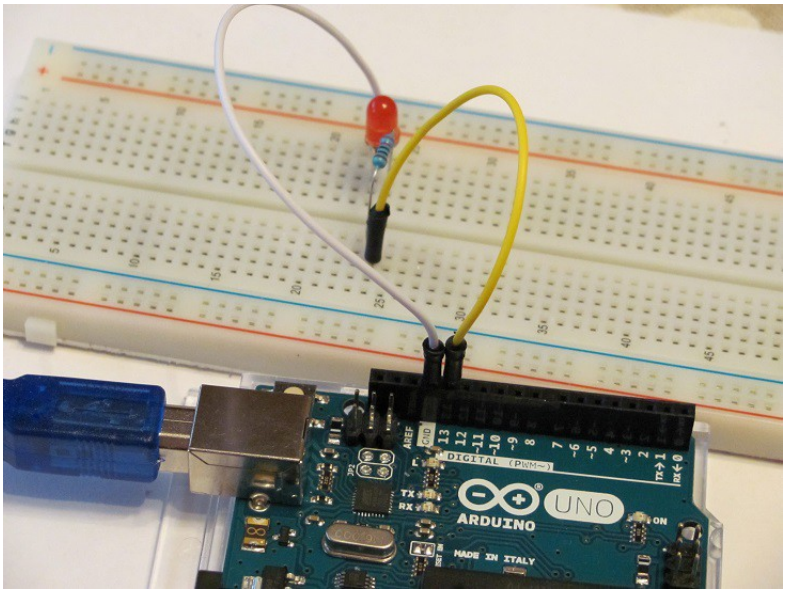


Fig.2.3. Montaje en la placa de pruebas del Proyecto 1

El super kit Sunfounder proporciona 8 resistencias de 220 Ω junto a otras resistencias de distinto valor. Para saber identificarla se puede utilizar el código de 5 bandas de color, pero lo más fiable es utilizar un multímetro para medirla, ya que muchas veces es difícil distinguir el rojo del marrón o del naranja, aunque se disponga de una lupa para observarlos. En la Fig.2.4 se muestra el valor de la resistencia proporcionado por un multímetro.

Fig.2.4. Valor de la resistencia medido con un multímetro

Realizado el montaje de la Fig.2.3, podemos conectar la placa Arduino al PC. Al hacerlo por primera vez, el LED parpadeará y luego se apagará (si no ocurriera esto, volver a desconectar y comprobar que la polaridad del LED es la correcta). Para conseguir la intermitencia del LED debemos abrir la aplicación de Arduino en nuestro PC y copiar en la ventana el programa o sketch Prog.2.1. Como puede apreciarse el lenguaje de programación es C. En él pueden distinguirse los comentarios, que bien comienzan con /* y acaban con */, o bien están encabezados en cada sentencia por //.

Todos los programas contienen dos grupos de sentencias entre llaves {}, encabezadas por void setup () y por void loop(). El primero permite definir los pines utilizados (en este caso el 13) y si son de entrada o salida (en este caso OUTPUT). El segundo repetirá las instrucciones entre llaves indefinidamente hasta que se interrumpa la conexión de la placa Arduino con el PC. En el Prog.2.1 hay un conjunto de 4 sentencias que ordenan: encender el LED, esperar 1000 ms encendido, apagar el LED y esperar 1000 ms apagado.

Prog.2.1. Programa en C que debe subirse a la placa Arduino para conseguir un LED intermitente

```
/*
Proyecto 1

LED intermitente
*/

void setup()
{
  // La placa Arduino tiene 14 pines digitales input/output.
  // Estos pines se pueden configurar mediante la función
  // pinMode().

  pinMode(13, OUTPUT);

  // El código IDE ofrece multitud de funciones de configuración.
  // Se puede obtener información sobre ellas en el sitio web:
  // http://arduino.cc/en/Reference
}


// Después que setup() finaliza, se ejecuta el comando loop() una y otra vez
// hasta que se apaga la placa o se resetea la placa.


void loop()
{
  // El pin digital 13 de la placa de Arduino envia señales
  // que pueden ser de 5 V ("HIGH") o de 0 V ("LOW").

  digitalWrite(13, HIGH);   // Enciende el LED

  // delay() es una funcion que retrasa una acción por un periodo de tiempo.
  // Toma un valor numérico en ms.

  delay(1000);              // Espera 1 s encendido

  digitalWrite(13, LOW);    // Apaga el LED

  delay(1000);              // Espera 1 s apagado

}
```

Una vez copiado Prog.2.1 en la ventana la aplicación de Arduino debemos utilizar los dos botones verdes circulares debajo del menú, para compilar el programa y subirlo a la placa Arduino. Veremos al cabo de un instante como el LED comienza su intermitencia, que proseguirá hasta que se desconecte la placa Arduino del PC.

2.2. PROYECTO 2: LED QUE AUMENTA Y DISMINUYE SU INTENSIDAD

En este proyecto describiremos dos formas de incrementar y disminuir la intensidad luminosa de un LED. Un primer método digital, empleando el denominado PWM (Pulse Width Modulation) o modulación de impulsos en anchura; y un segundo método analógico, valiéndonos de un potenciómetro.

2.2.1. Proyecto 2a: Aumento y disminución de la intensidad del LED mediante PWM

Para realizar la modulación de impulsos en anchura se emplea una onda pulsada cuadrada en la que varía la anchura del impulso, hasta conseguir modificar el valor promedio de la onda (es decir, lo que se mediría con un multímetro). Por ejemplo, si tenemos una onda pulsada cuadrada con un ciclo de trabajo de 0,75 (75%), y un valor máximo (HIGH) de 5 V, el valor promedio será 3,75 V (Fig.2.5). De esta forma, aplicando ciclos de trabajo cada vez más pequeños se puede reducir la intensidad luminosa del LED.

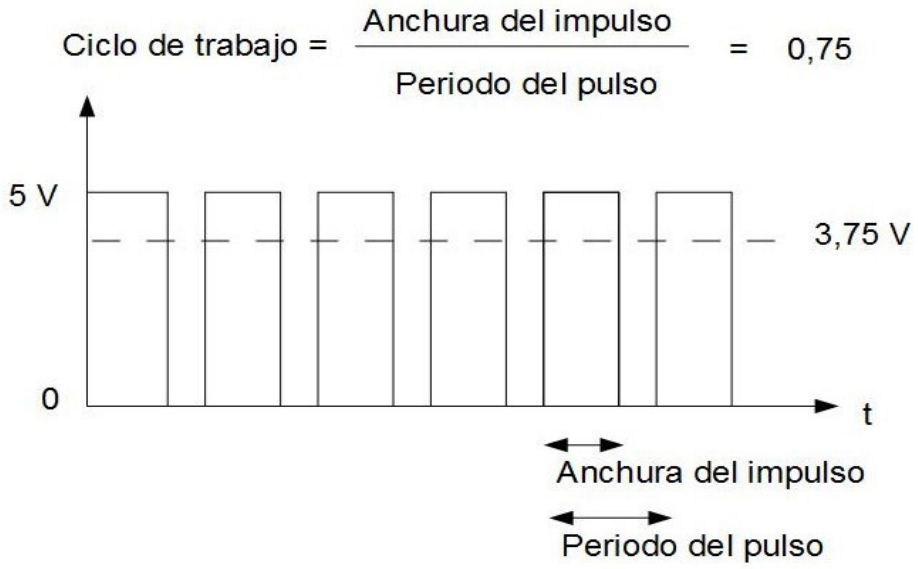


Fig.2.5. Ejemplo de ciclo de trabajo y valor promedio en una onda pulsada cuadrada

El circuito empleado en este proyecto es idéntico al de la Fig.2.2. Únicamente debemos tener en cuenta que el pin digital 13 de la placa Arduino no es capaz de realizar PWM, ya que, como indica la placa, no contiene el símbolo ~. Así pues, en lugar del pin 13, se ha elegido el pin 9, que sí contiene el símbolo ~. Por tanto, si queremos que la luz del LED pierda y gane intensidad paulatinamente subiremos a la placa Arduino el Prog.2.2.

Podemos apreciar como en esta ocasión se emplea la función anlogWrite(), la cual permite realizar PWM en una escala de 0 a 255.

Prog.2.2. Programa que produce la perdida y ganancia de intensidad del LED

```
/*
Proyecto 2a

LED que pierde y gana intensidad mediante PWM
*/


void setup ()
{
  pinMode(9, OUTPUT);              // declara el pin 9 como salida
}

void loop()
{
  for (int a=0; a<=255;a++)          // realiza un bucle de 0 to 250
  {
    analogWrite(9, a);             // hace que brille el LED según la variable a
    delay(20);                     // retrasa 20 ms
  }
    for (int a=255; a>=0;a--)        // realiza un bucle de 250 a 0
  {
    analogWrite(9, a);
    delay(20);
  }
}
```

2.2.2. Proyecto 2b: Aumento y disminución de la intensidad del LED mediante un potenciómetro

En este caso conectaremos al circuito un potenciómetro (Fig.2.6) que permitirá aumentar y disminuir manualmente la intensidad luminosa del LED. El circuito del proyecto es como se muestra la Fig.2.7. La parte del circuito correspondiente al LED y la resistencia es idéntica a la del LED intermitente. Únicamente hay que añadir el potenciómetro, que se encuentra unido al pin de 5 V de la placa Aduino por su clavija izquierda, al pin analógico A0 con su clavija central y a tierra (GND) con su clavija derecha (Fig.2.8).

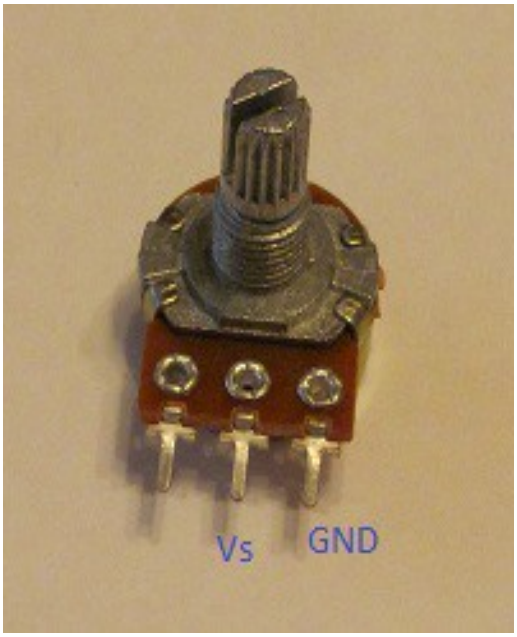


Fig.2.6. Ejemplo de potenciómetro utilizado en el circuito

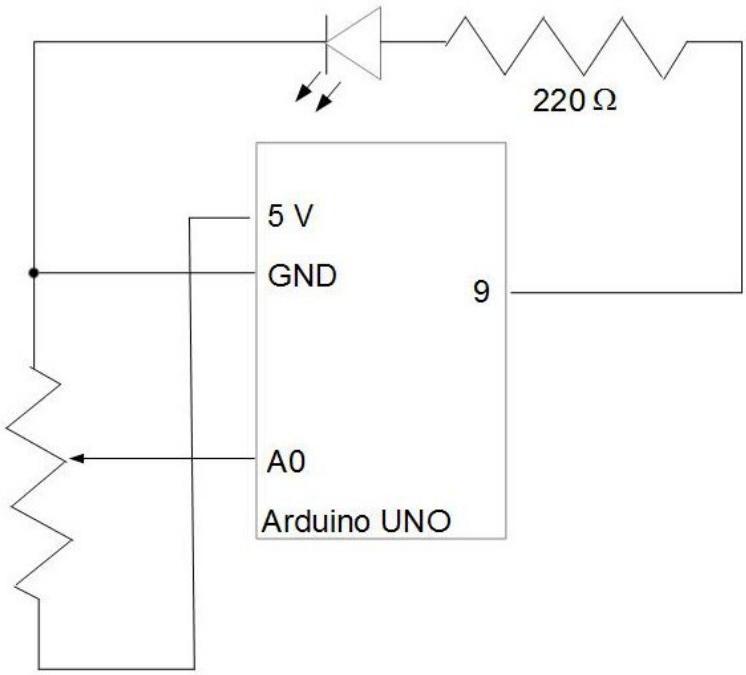


Fig.2.7. Circuito que permite reducir y aumentar la intensidad del LED con ayuda de un potenciómetro

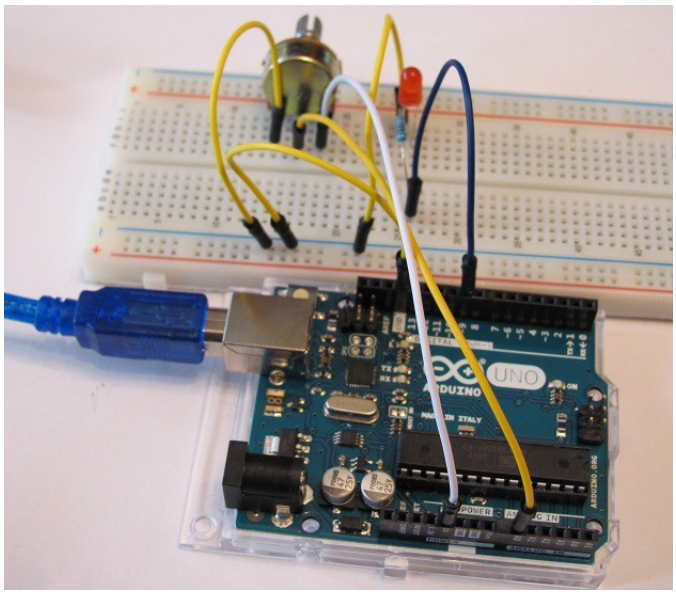


Fig.2.8. Montaje del circuito que aumenta y disminuye la intensidad del LED con ayuda de un potenciómetro.

En el Prog.2.3 se adapta la escala analógica del potenciómetro de 0 a 1023 a la de la función analogWrite() que es de 0 a 255. Para ello se utiliza la función map().

Prog.2.3. Programa de cambio de intensidad de un LED con ayuda de un potenciómetro

```
/*
Proyecto 2b

LED que va paulatinamente perdiendo
y ganando intensidad con ayuda de un potenciómetro
*/

const int PinAnalogico = A0;                  // se utiliza el pin A0 para la entrada analógica
const int Pin = 9;
int ValorEntrada = 0;                         // variable que guarda el valor de entrada del potenciómetro
int ValorSalida = 0;                          // variable que guarda el valor de salida en escala 0 a 255


void setup()
{
  pinMode(Pin, OUTPUT);
}

void loop()
{
 // Pierde y gana intensidad segun el potenciómetro
  ValorEntrada = analogRead(PinAnalogico);          // lee el valor del potenciómetro
  ValorSalida = map(ValorEntrada,0,1023,0,255);     // convierte el valor de entrada en un número de 0 a 255
  analogWrite(Pin, ValorSalida);                    // enciende el LED con intensidad del valor de salida
}
```

2.3. PROYECTO 3: HILERA DE LEDs QUE SE ENCIENDEN Y APAGAN SUCESIVAMENTE COMO EN LA PISTA DE ATERRIZAJE DE UN AEROPUERTO

Para conseguir el efecto de una pista de aterrizaje en una hilera de LEDs proponemos el circuito de la Fig.2.9. Como puede apreciarse, el efecto de encendido y apagado sucesivo en los cinco LEDs se controla con los pines del 9 al 13 de la placa Arduino. El montaje realizado se muestra en la Fig.2.10.

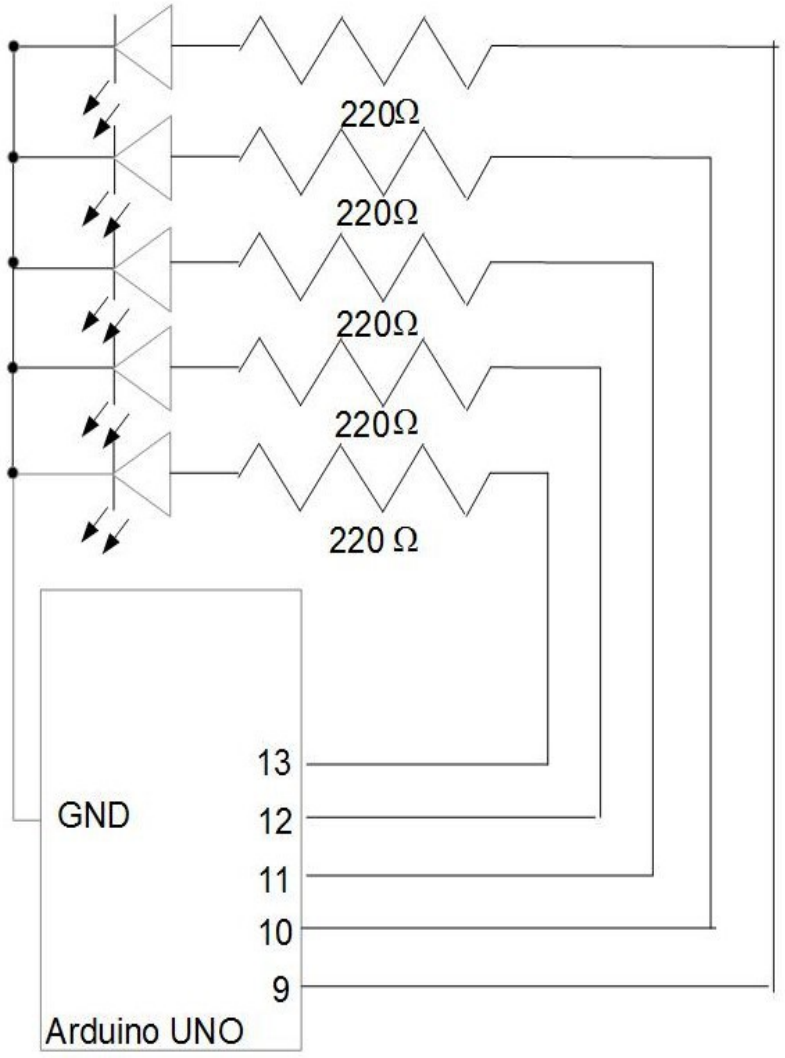


Fig.2.9. Circuito propuesto para una hilera de LEDs que simulan una pista de aterrizaje

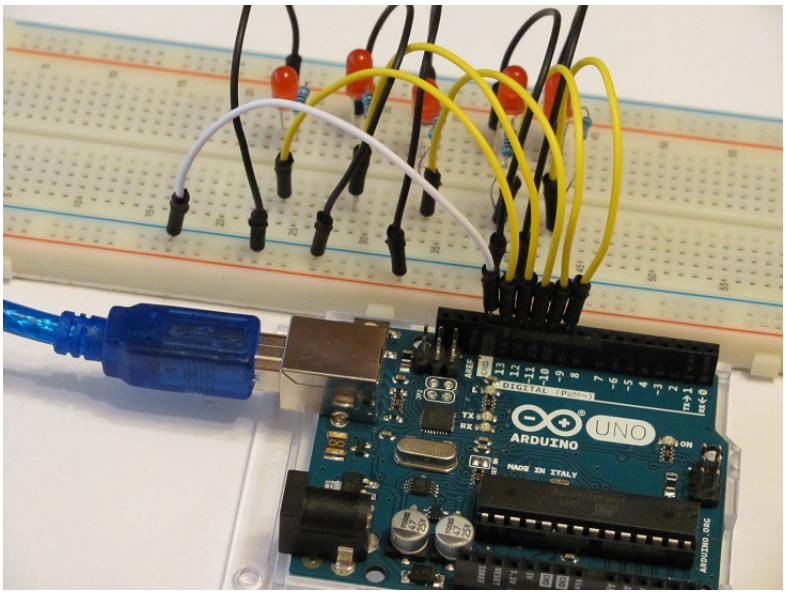


Fig.2.10. Montaje del circuito de la Fig.2.9

En cuanto al sketch se emplean bucles para acceder a cada uno de los pines de la placa Arduino. Para ello se iteran los pines del 9 al 13 desde el más bajo al más alto, tanto en void setup(), como en void loop().

Prog.2.4. Sketch del Proyecto 3

```
/*
Proyecto 3

Conjunto de LEDs en línea que se encienden y apagan
sucesivamente como en una pista de aterrizaje
*/

const int PinBajo = 9;          // el pin de numeración más baja
const int PinAlto = 13;         // el pin de numeración más alta


void setup()
{
 //poner todos los pines de 9 a 13 como salida

 for(int Pin = PinBajo;Pin <= PinAlto;Pin++)
 {
   pinMode(Pin,OUTPUT);
 }
}
```

```
void loop()
{
 // iterar los pines desde el más bajo al más alto
 for(int Pin = PinBajo;Pin <= PinAlto;Pin++)
 {
  digitalWrite(Pin,HIGH);       // encender el LED
  delay(100);                   // esperar 100 ms encendido
 }
 // apagar todos los LED
 for(int Pin = PinBajo;Pin <= PinAlto;Pin++)
 {
  digitalWrite(Pin,LOW);        // apagar el LED
 }
 delay(100);
  // iterar los pines desde el más bajo al más alto
 for(int Pin = PinBajo;Pin <= PinAlto;Pin++)
 {
  digitalWrite(Pin,HIGH);       // encender el LED
  delay(100);                   // esperar 100 ms encendido
 }
}
```

2.4. PROYECTO 4: JUEGO DE COLORES CON EL LED RGB

Como se puede apreciar en la Fig.2.1b, el LED RGB posee cuatro patas. La más larga es la de conexión a tierra (GND). La aislada, a un lado de la más larga, es la relativa al color rojo R, mientras que las dos restantes, al otro lado de la más larga, son la azul B, en un extremo, y la verde G.

2.4.1. Proyecto 4a: Colores combinados con un LED RGB

En este proyecto no sólo generaremos en un solo LED los tres colores fundamentales: rojo, verde y azul, sino que también seremos capaces de crear colores combinados como el naranja, el amarillo, el púrpura o el blanco. El circuito necesario se muestra en la Fig.2.11, mientras que el montaje asociado puede verse en la Fig.2.12.

Se han elegido los pines 9, 10 y 11 para los colores rojo, verde y azul, ya que contienen el simbolo ~, que permite utilizar la función analogWrite() en el Prog.2.5. Por otro lado, se ha definido la función color() de tres variables enteras, que permite encender el LED RGB en el color determinado por las tres coordenadas RGB. Por ejemplo, las tres coordenadas (255,255,255) corresponden al blanco, mientras que (255,233,0) son las del amarillo.

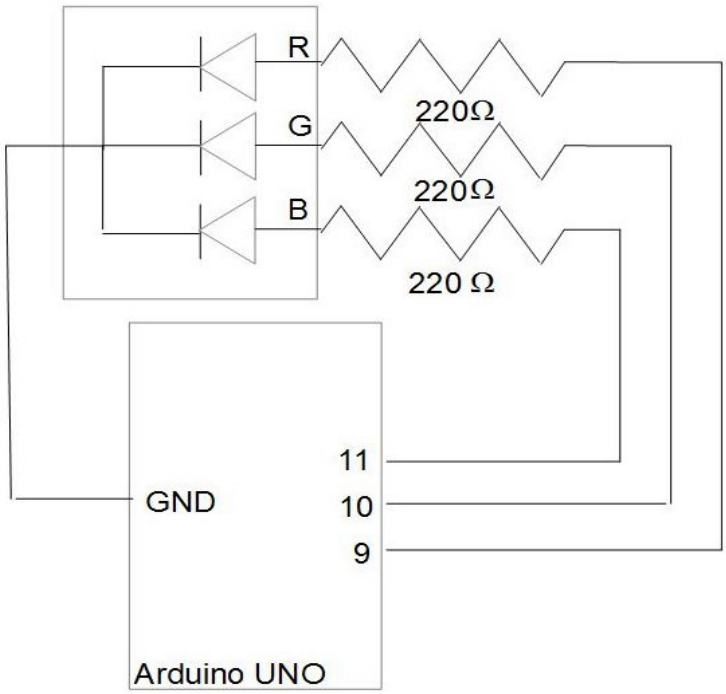


Fig.2.11. Circuito de un LED RGB

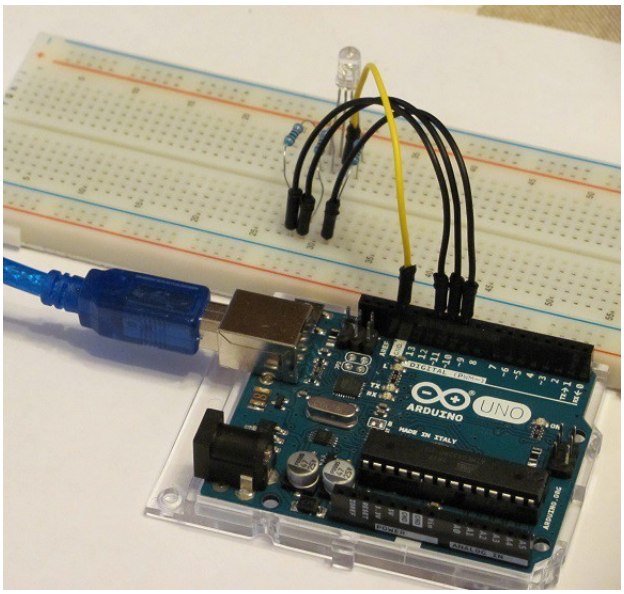


Fig.2.12. Montaje asociado al circuito de la Fig.2.11

Prog.2.5. Programa que permite jugar con los colores en el LED RGB

```
/*
Proyecto 4a

LED RGB que se enciende en distintos colores
*/

const int PinRojo = 9;
const int PinVerde = 10;
const int PinAzul = 11;

void setup()
{
  pinMode(PinRojo, OUTPUT);
  pinMode(PinVerde, OUTPUT);
  pinMode(PinAzul, OUTPUT);
}

void loop()
{
  // Enciende 1 s los colores básicos: rojo, verde, azul
  color(255, 0, 0);
  delay(1000);
  color(0,255, 0);
  delay(1000);
  color(0, 0, 255);
  delay(1000);

  // Enciende 1 s colores combinados
  color(230,95,0); // naranja
  delay(1000); //
  color(255,233,0); // amarillo
  delay(1000);
  color(125,33,129); // púrpura
  delay(1000);
  color(255,255,255); // blanco
  delay(1000);
}
// Define la función de tres variables enteras color()
void color (int Rojo, int Verde, int Azul)
{
  analogWrite(PinRojo, Rojo);
  analogWrite(PinVerde, Verde);
  analogWrite(PinAzul, Azul);
}
```

2.4.2. Proyecto 4b: Cambio de intensidad en un color combinado

Vamos a disminuir y aumentar la intensidad de luz de un color combinado como el amarillo, de la misma forma que hicimos en el Proyecto 2a, por modulación de impulsos en anchura (PWM). Puesto que el color descrito por las coordenadas RGB se asignan al LED a través de las patas roja, verde y azul, respecto a tierra, no parece en principio posible disminuir la intensidad luminosa del LED RGB, manteniendo un color determinado. Para ello hay que tener en cuenta que si dividimos las tres coordenadas por un valor constante, el color del LED RGB se mantiene. Por ejemplo, en el amarillo, si utilizamos la constante *c* para definir las coordenadas (255/*c*,233/*c*,0), el color seguirá siendo amarillo, aunque su intensidad luminosa disminuirá en un factor *c*. Así pues, si al montaje de la Fig.2.12 añadimos el sketch del Prog.2.5, obtendremos un LED que aumenta y disminuye su intensidad manteniendo el color amarillo.

Prog.2.6. Sketch del LED RGB en color amarillo que pierde y gana intensidad luminosa

```
/*
Proyecto 4b

El LED RGB en color amarillo va paulatinamente perdiendo
y ganando intensidad
*/

// Define los pines digitales
const int PinRojo = 9;
const int PinVerde = 10;
const int PinAzul = 11;
// Define el color amarillo
const int ColorR = 255;
const int ColorG = 233;
const int ColorB = 0;
// Define el mismo amarillo con diferente intensidad
int Ca,Cb,Cc;

void setup()
{
  pinMode(PinRojo, OUTPUT);
  pinMode(PinVerde, OUTPUT);
  pinMode(PinAzul, OUTPUT);
}
```

```
void loop()
{
 // Pierde y gana intensidad el color amarillo
  for (int a=100; a>=1;a--)
  {
    Ca = int(ColorR*a/100.);
    Cb = int(ColorG*a/100.);
    Cc = int(ColorB*a/100.);
    color(Ca,Cb,Cc);
    delay(20);
  }
  for (int a=1; a<=100;a++)
  {
    Ca = int(ColorR*a/100.);
    Cb = int(ColorG*a/100.);
    Cc = int(ColorB*a/100.);
    color(Ca,Cb,Cc);
    delay(20);
  }

}

// Define la función de tres variables enteras color()
void color (int Rojo, int Verde, int Azul)
{
  analogWrite(PinRojo, Rojo);
  analogWrite(PinVerde, Verde);
  analogWrite(PinAzul, Azul);
}
```

2.4.3. Proyecto 4c: Cambio de intensidad en un color combinado con ayuda de un potenciómetro

En este proyecto aumentaremos y disminuiremos la intensidad luminosa de un color combinado en un LED RGB mediante la utilización de un potenciómetro. El esquema del circuito y su montaje se muestran en las Figs. 2.12 y 2.13. El sketch necesario se puede ver en Prog.2.7.

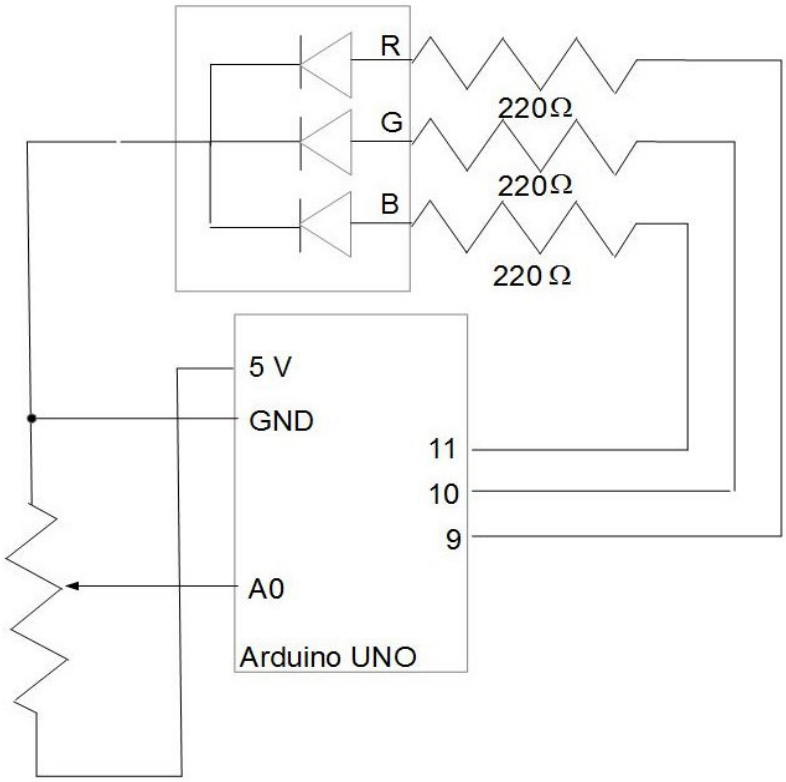


Fig.2.12. Circuito de un LED RGB en el que se cambia la intensidad de un color combinado con un potenciómetro.

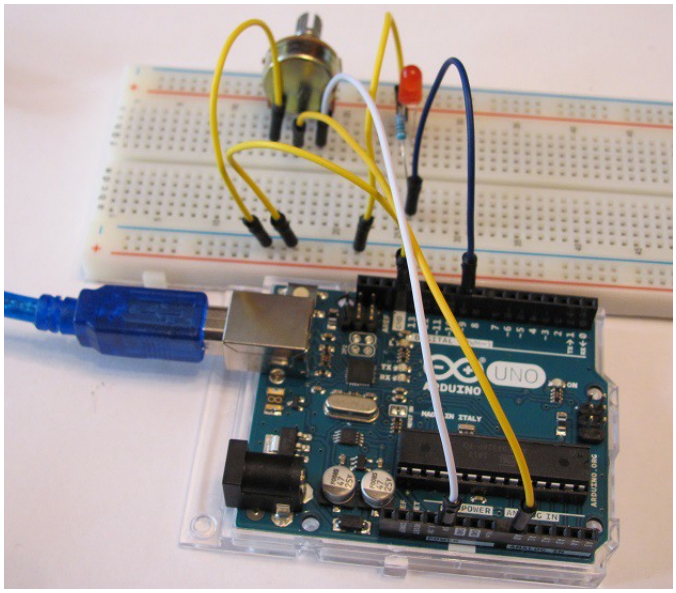


Fig.2.13. Montaje asociado al circuito de la Fig.2.12

Prog.2.7. Sketch del LED RGB en color amarillo que cambia su intensidad luminosa con un potenciómetro

```
/*
Proyecto 4c

El LED RGB en color amarillo va paulatinamente perdiendo
y ganando intensidad con ayuda de un potenciómetro
*/

const int PinAnalogico = A0;                // se utiliza el pin A0 para la entrada analógica
// Define los pines digitales
const int PinRojo = 9;
const int PinVerde = 10;
const int PinAzul = 11;
// Define el color amarillo
const int ColorR = 255;
const int ColorG = 233;
const int ColorB = 0;
int ValorEntrada = 0;                       // variable que guarda el valor de entrada del potenciómetro
int ValorSalida = 0;                        // variable que guarda el valor de salida en escala 0 a 255
// Define las coordenadas RGB con intensidad modificada
int Ca,Cb,Cc;


void setup()
{
   pinMode(PinRojo, OUTPUT);
   pinMode(PinVerde, OUTPUT);
   pinMode(PinAzul, OUTPUT);
}

void loop()
{
 // Pierde y gana intensidad el color combinado según el potenciómetro
  ValorEntrada = analogRead(PinAnalogico);          // lee el valor del potenciómetro
  ValorSalida = map(ValorEntrada,0,1023,0,255);     // convierte el valor de entrada en un número de 0 a 255
  Ca = int (ValorSalida * ColorR / 255.);
  Cb = int (ValorSalida * ColorG / 255.);
  Cc = int (ValorSalida * ColorB / 255.);
  color(Ca,Cb,Cc);                          // enciende el LED
}

// Define la función de tres variables enteras color()
void color (int Rojo, int Verde, int Azul)
{
  analogWrite(PinRojo, Rojo);
  analogWrite(PinVerde, Verde);
  analogWrite(PinAzul, Azul);
}
```

2.5. PROYECTO 5: LED QUE SE ACTIVA CON UN BOTÓN

La Fig.2.14 muestra el anverso y el reverso de un botón. Las flechas azules indican aquellas clavijas que se encuentran conectadas internamente, es decir, que su resistencia medida con un multímetro es nula.

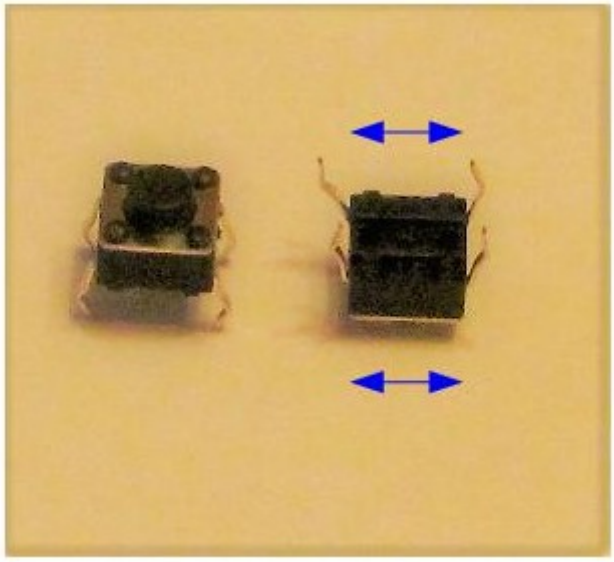


Fig.2.14. Anverso y reverso de un botón. Las clavijas sin discontinuidad en el plástico son las que están conectadas internamente

2.5.1. Proyecto 5a: Luz intermitente de un LED activada por un botón

Vamos a conseguir que, tras accionar un botón, la luz LED se ponga intermitente tres veces, para luego apagarse. El circuito necesario para conseguirlo es el mostrado en la Fig.2.15. Como puede apreciarse se ha conectado en serie con el botón una resistencia de 10 kΩ, cuya misión es evitar interferencias mientras se acciona el botón. El pin 9 que se conecta al circuito entre el botón y la resistencia de 10 kΩ, informa al sketch sobre el estado del botón, activo (HIGH) o inactivo (LOW). La intermitencia del LED se produce cuando queda inactivo el botón al soltarlo después de pulsarlo. El montaje sobre la placa de pruebas se muestra en la Fig.2.16, mientras que el sketch del Proyecto 5a puede copiarse del Prog.2.8.

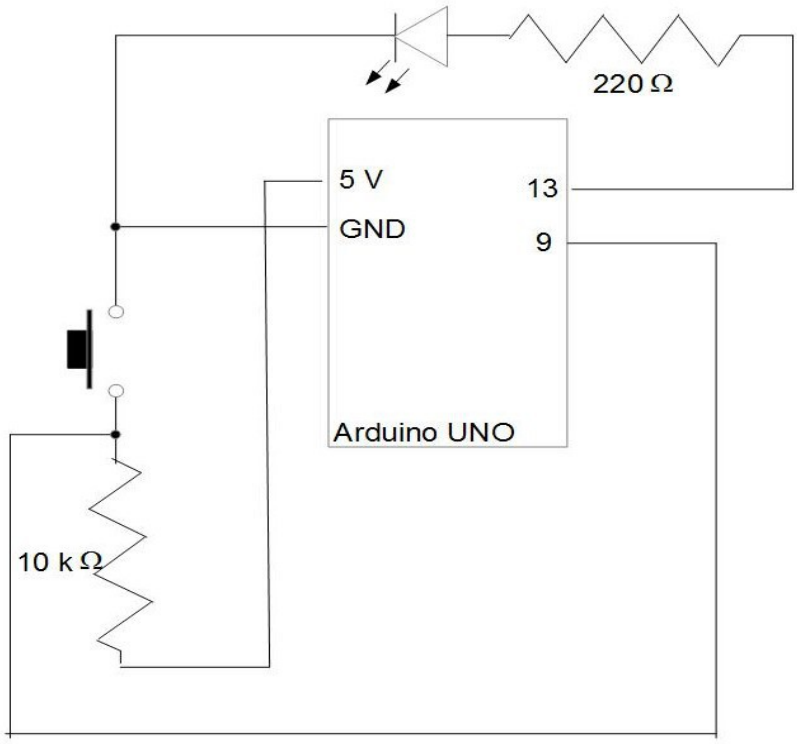


Fig.2.15. Circuito del Proyecto 5a

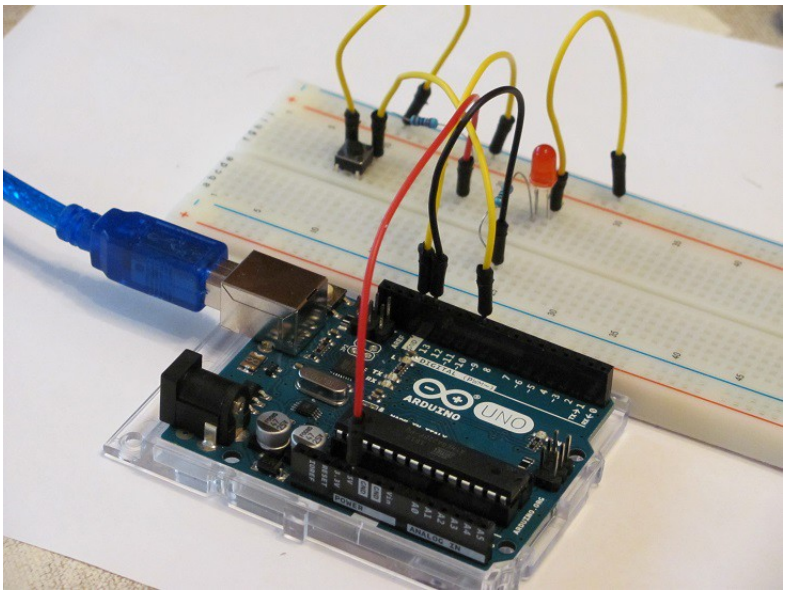


Fig.2.16. Montaje en la placa de pruebas del Proyecto 5a

Prog.2.8. Sketch del Proyecto 5a

```
/*
Proyecto 5a

Se controla el parpadeo de un LED apretando un botón
*/


// Define constantes como pines 9 y 13
const int PinBoton = 9;
const int PinLed =  13;

void setup()
{
 // Define el pin del botón como entrada
 pinMode(PinBoton, INPUT);

 // Define el pin del LED como salida
 pinMode(PinLed, OUTPUT);
}


void loop()
{
 // Define la variable EstadoBoton como la lectura digital del PinBoton
 int EstadoBoton;
 EstadoBoton = digitalRead(PinBoton);
 if (EstadoBoton == LOW)
 // Si el botón se pulsa, el LED procede a parpadear 3 veces
 {
   for (int a=1; a<=3;a++)
   {
     digitalWrite(PinLed, HIGH);
     delay(1000);
     digitalWrite(PinLed, LOW);
     delay(1000);
   }
 }
 else
 // Si el botón no se pulsa, el LED permanece apagado
 {
     digitalWrite(PinLed, LOW);
 }

}
```

2.5.2. Proyecto 5b: La luz del LED pasa del rojo al blanco al pulsar un botón

En este proyecto vamos a conseguir que un LED RGB se muestre de distinto color si se acciona un botón. La respuesta tiene lugar como en el Proyecto 5b, cuando el botón pasa a inactivo (LOW),

después de pulsarlo. El circuito y el montaje se muestran en la Fig.2.17 y 2.18. El sketch está en Prog.2.9.

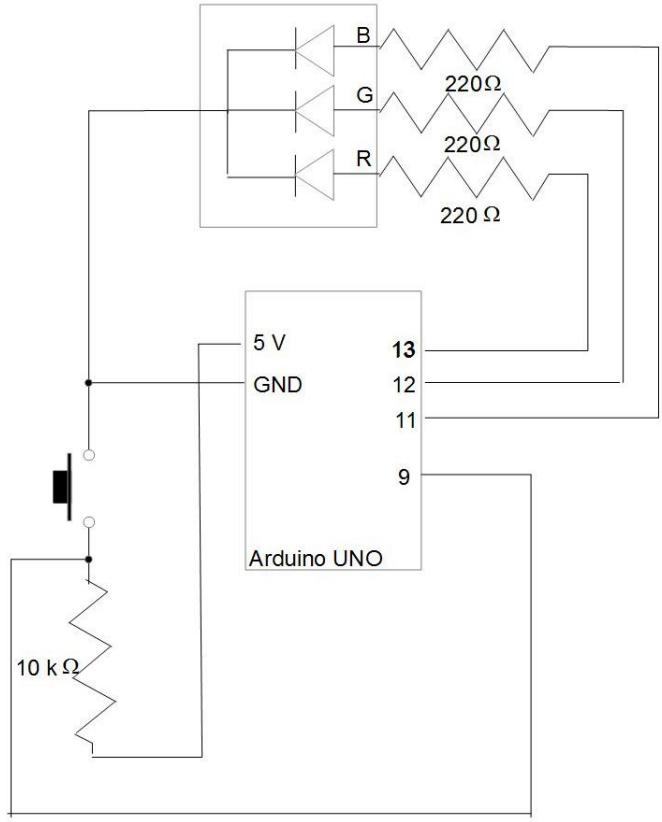


Fig.2.17. Circuito del Proyecto 5b

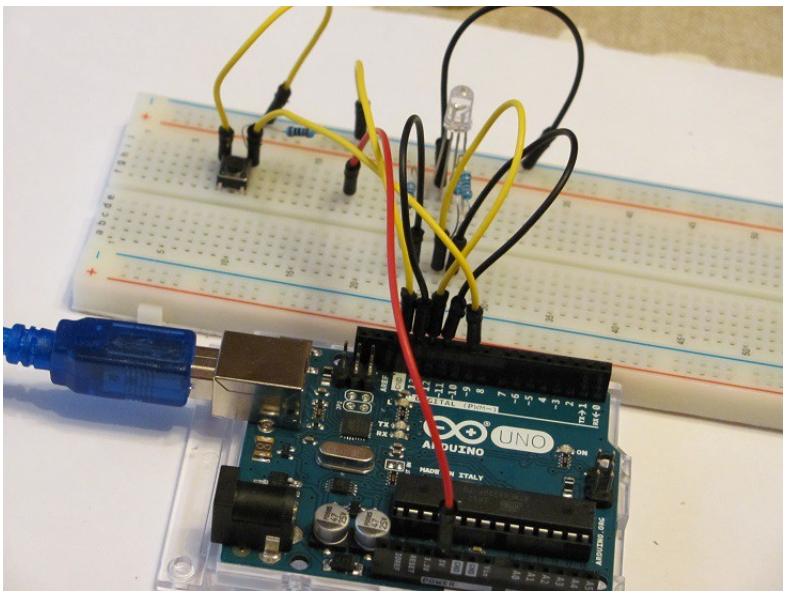


Fig.2.18. Montaje del Proyecto 5b

Prog.2.9. Sketch del Proyecto 5b

```
/*
Proyecto 5b

Se controla el color de un LED apretando un botón
*/


// Define constantes como pines 9 y 11,12 y 13
const int PinBoton = 9;
const int PinRojo = 13;
const int PinVerde = 12;
const int PinAzul = 11;

void setup()
{
 // Define el pin del botón como entrada
 pinMode(PinBoton, INPUT);

 // Define los pines del LED como salida
 pinMode(PinRojo, OUTPUT);
 pinMode(PinVerde, OUTPUT);
 pinMode(PinAzul, OUTPUT);
}


void loop()
{
 // Define la variable EstadoBoton como la lectura digital del PinBoton
 int EstadoBoton;
 EstadoBoton = digitalRead(PinBoton);
 if (EstadoBoton == LOW)
 // Si el botón se pulsa, el LED es blanco
 {
  color(255,255,255);
 }
 else
 // Si el botón no se pulsa, el LED es rojo
 {
  color(255,0,0);
 }

}
// Define la función de tres variables enteras color()
void color (int Rojo, int Verde, int Azul)
{
 analogWrite(PinRojo, Rojo);
 analogWrite(PinVerde, Verde);
 analogWrite(PinAzul, Azul);
}
```

Capítulo 3

Sensores

Se define sensor como cualquier componente electrónico capaz de transformar una magnitud física o química en una magnitud eléctrica. Un ejemplo de sensor es el termistor, el cual es capaz de transformar la temperatura en una magnitud eléctrica fácilmente medible, como es la resistencia eléctrica. En este capítulo nos familiarizaremos con el empleo de sensores térmicos (LM35), de intensidad lumínica (CDS), de infrarrojos (AX-1838HS), de contacto (TTP223-BA6), de vibración (SW-520D), de llama (un tipo particular de sensor de infrarrojos), de aceleración (ADXL335) y de proximidad (HC-SR04).

3.1. PROYECTO 6: SENSOR DE TEMPERATURA LM35

La Fig.3.1 muestra un sensor de temperatura LM35. Rigurosamente hablando no se trata de un termistor o resistencia eléctrica, sino de un circuito integrado o chip. Un LM35 se caracteriza por poseer tres patas y una cabeza de plástico (muy parecido a un transistor). Poniendo de frente la cara plana de la cabeza de plástico, su polo negativo (GND) estaría en la pata derecha, su polo positivo en la pata izquierda, y el voltaje de la señal de salida Vs en su pata central. Es importante no cambiar la polaridad cuando se inserta en la placa de pruebas, puesto que en caso de cambiarla, al conectar el montaje al PC, el LM35 se calentará rápidamente, hasta el punto de quedar inutilizado si no se interrumpe inmediatamente la corriente. Un LM35 no precisa ser calibrado, puesto que la temperatura en grados centígrados puede calcularse a partir del voltaje Vs aplicando la expresión: Temp.(ºC) = 100 × Vs (Voltios). El rango de medida de un LM35 es de -55ºC a 150ºC.

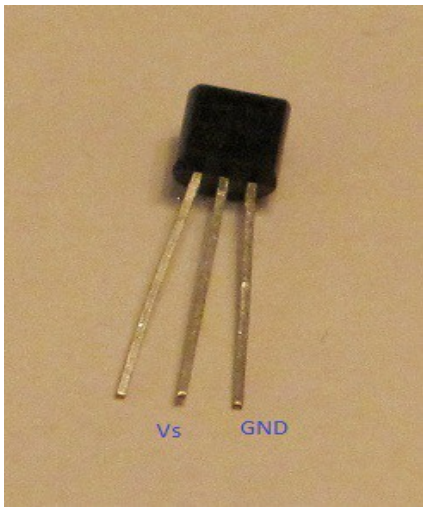


Fig.3.1. Sensor de temperatura LM35. Es importante no cambiar su polaridad

No conviene confundir el sensor de temperatura LM35 con el TMP36. Ambos sensores son idénticos a primera vista. Únicamente se diferencian en el código (LM35 o TMP36) serigrafiado en la cara plana de la cabeza del sensor. Esta indicación es a veces difícil de ver, incluso con ayuda de una lupa. En caso de tratarse de un TMP36, la expresión para calcular la temperatura en grados centígrados varía ligeramente, adoptando la forma siguiente: Temp(ºC) =100 × (Vs – 0,5).

Las Figs.3.2 y 3.3 muestran el circuito y el montaje sobre la placa de pruebas, respectivamente.

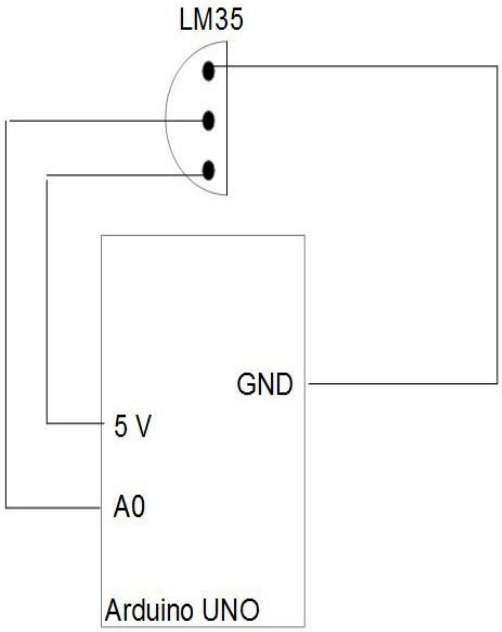


Fig.3.2. Circuito del Proyecto 6

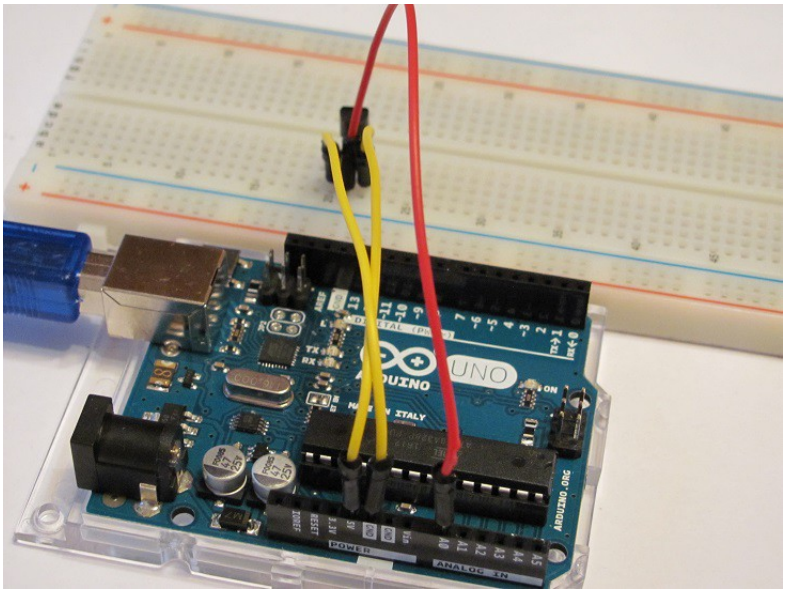


Fig.3.3. Montaje del Proyecto 6

A este proyecto se adjunta un sketch (Prog.3.1), capaz de enviar al monitor de serie una tabla, que incluye el tiempo transcurrido en segundos desde el comienzo de la medida, el voltaje de salida Vs y la temperatura ambiente en grados centígrados (Fig.3.4). El monitor de serie se activa haciendo click en el botón verde a la derecha, bajo la barra de menú, en la ventana de la aplicación Arduino. El sketch realiza una lectura a través del pin A0 cada 10 s y la plasma en la tabla del monitor de serie.

La instrucción Serial.begin(9600) abre la comunicación entre la placa Arduino y el monitor de serie, con una velocidad de transmisión de 9600 baudios por segundo. Las instrucciones Serial.print y Serial.println son las encargadas de imprimir en el monitor de serie la tabla de la Fig.3.4 con el formato adecuado.

La lectura del voltaje Vs por el pin A0 es cada 10 s. Sin embargo, la lectura está en una escala de 0 a 1023, por lo que debe cambiarse a voltios. Esta es la misión de la función LeerVoltaje(), que multiplica la lectura analógica de A0 por el factor 5 / 1024 = 0,0048828.

Podemos comprobar que la lectura del voltaje Vs es correcta conectando un multímetro a las patas central y derecha (Fig.3.1) del LM35, como muestra la Fig.3.5.

Prog.3.1. Sketch del Proyecto 6

```
/*
Proyecto 6

Medimos la temperatura ambiente con un sensor de temperatura
*/

// Usamos el pin analógico A0 para leer la temperatura del sensor
const int PinTemperatura = A0;
int Tiempo0 = 10;
int Tiempo = 0;

void setup()
{
  // Usamos el puerto de serie de Arduino para mandar mensajes
  // de texto al PC. La velocidad de transmisión se mide en
  // baudios por segundo. La tasa de baudios más común es 9600.

  Serial.begin(9600);
  Serial.print("Tiempo      ");
  Serial.print("Voltaje     ");
  Serial.println("Temp. (C)");
  Serial.print("------      ");
  Serial.print("------      ");
  Serial.println("--------");
}
```

```
void loop()
{
 float Voltaje, GradosC;

 // Definimos el tiempo
 Tiempo = Tiempo + Tiempo0;

 // El valor proporcionado por el pin analogico A0 es un número
 // entre 0 y 1023. Definimos la función LeerVoltaje() a aquella
 // que permite transformar dicho valor en una medida de voltaje
 // entre 0 y 5 V.

 Voltaje = LeerVoltaje(PinTemperatura);

 // Convertimos el voltaje en grados Celsius.
 // Para TEMP36
 // GradosC = (Voltaje - 0.5) * 100.0;
 // Para LM35
  GradosC = Voltaje * 100.0;

 // Se mandan los datos de Aruino a la ventana de serie del monitor.

 Serial.print("  ");
 Serial.print(Tiempo);
 Serial.print("         ");
 Serial.print(Voltaje);
 Serial.print("         ");
 Serial.println(GradosC);

 delay(Tiempo0*1000);
}

float LeerVoltaje(int Pin)
{
 // Definimos la función LeerVoltaje()
 return (analogRead(Pin) * 5. / 1024.);
}
```

Fig.3.4. Monitor de serie mostrando la tabla de resultados

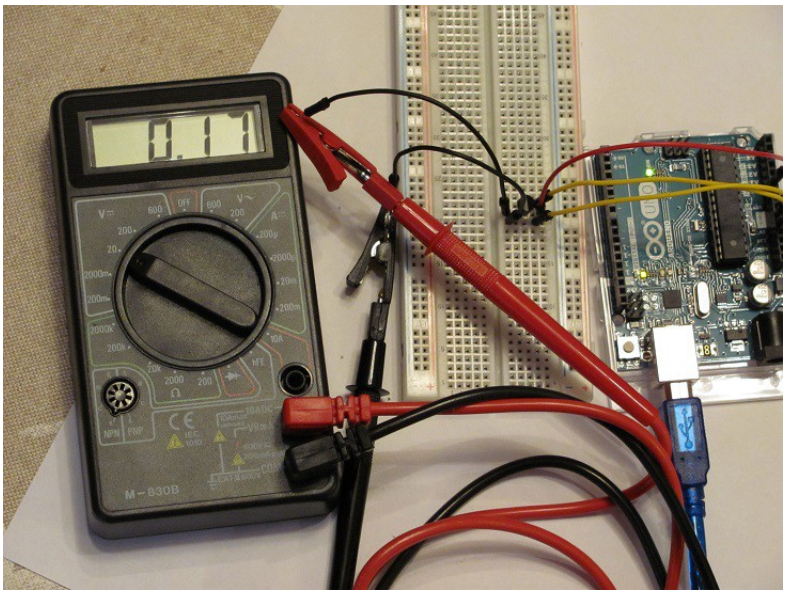


Fig.3.5. Lectura del voltaje Vs con un multímetro

3.2. PROYECTO 7: ACTIVACIÓN DE UN LED CUANDO LA LUZ AMBIENTE ES POCA

La Fig.3.6 muestra una fotorresistencia (LDR, Light-Dependent Resistor) del tipo CDS. Este sensor funciona variando el valor de la resistencia, cuando se modifica la intensidad de luz que incide sobre ella. En las fotorresistencias aumenta la resistencia de forma apreciable conforme oscurece la luz ambiental. El Proyecto 7 lo dividiremos en dos partes: una primera, de calibración; y la segunda, de construcción de un circuito con un LED que se enciende cuando la luz ambiente es poca.

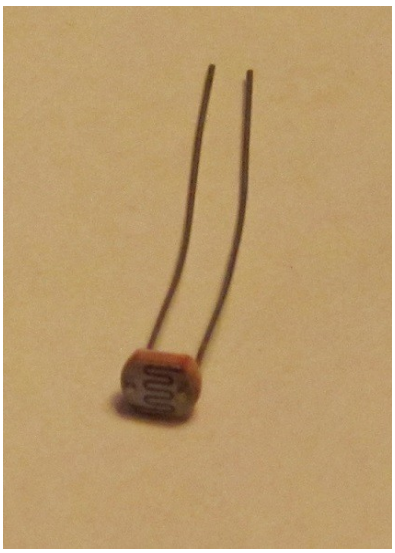


Fig.3.6. Fotorresistencia CDS

3.2.1. Proyecto 7a: Calibración de una fotorresistencia

Vamos a calibrar la fotorresistencia, sometiéndola a distintos valores de intensidad luminosa, con el fin de delimitar el valor del voltaje que corresponde a una luz ambiental suficientemente oscura. Para ello proponemos el circuito de la Fig.3.7, donde se ha incluido un botón. De esta forma, cada vez que se pulse el botón, se generará una medida del voltaje, que transmitiremos al monitor de serie a través del pin A0. Para evitar que se transmitan muchas medidas durante el tiempo en que el botón está pulsado, estableceremos un retardo de 1 s después de que se active el botón (HIGH) y se imprima la primera medida. Todo ello lo implantaremos en el sketch de Prog.3.2.

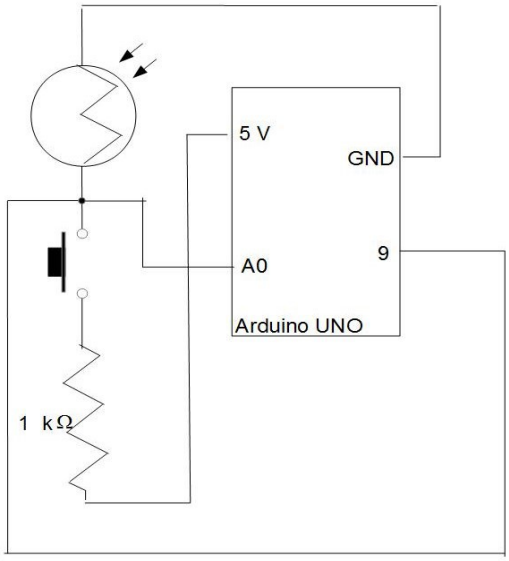


Fig.3.7. Circuito para la calibración de la fotorresistencia

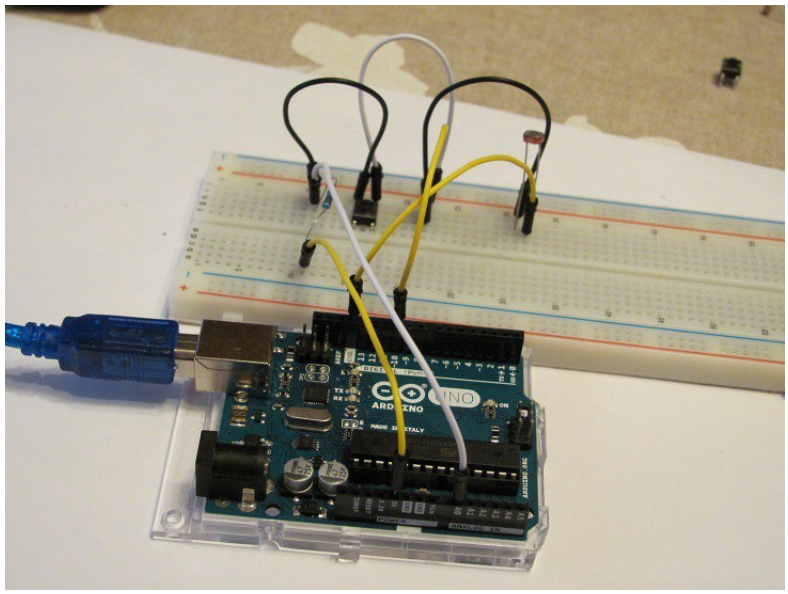


Fig.3.8. Montaje del circuito de la Fig.3.7

Prog.3.2. Sketch para la calibración de la fotorresitencia

```
/*
Proyecto 7a

Calibramos la fotorresistencia con ayuda de un botón
*/

// Usamos el pin analógico A0 para leer el voltaje
const int PinVoltaje =A0;
const int PinBoton = 9;
int Medida = 1;

void setup()
{
  pinMode(PinBoton, INPUT);
  Serial.begin(9600);
  Serial.print("Medida      ");
  Serial.println("Voltaje     ");
  Serial.print("------      ");
  Serial.println("------      ");
}

void loop()
{
  float Voltaje;
  int EstadoBoton;
  EstadoBoton = digitalRead(PinBoton);
  Voltaje = LeerVoltaje(PinVoltaje);
  if (EstadoBoton == HIGH)
  // Se mandan los datos de Aruino al monitor de serie
  {
    Serial.print("  ");
    Serial.print(Medida);
    Serial.print("         ");
    Serial.println(Voltaje);
    Medida++;
    delay(1000);
  }
}

float LeerVoltaje(int Pin)
{
  // Definimos la función LeerVoltaje()
  return (analogRead(Pin) * 5. / 1024.);
}
```

Subiendo el sketch a la placa Arduino, haremos tres tipos de medidas: una primera con las luces de la lámpara del salón encendidas, una segunda con la luz ambiental que entra por la ventana y una tercera cubriendo con la mano la fotorresistencia. El resultado del voltaje detectado por el pin A0 al apretar el botón en cada uno de los supuestos es el que ilustra la Fig.3.9.

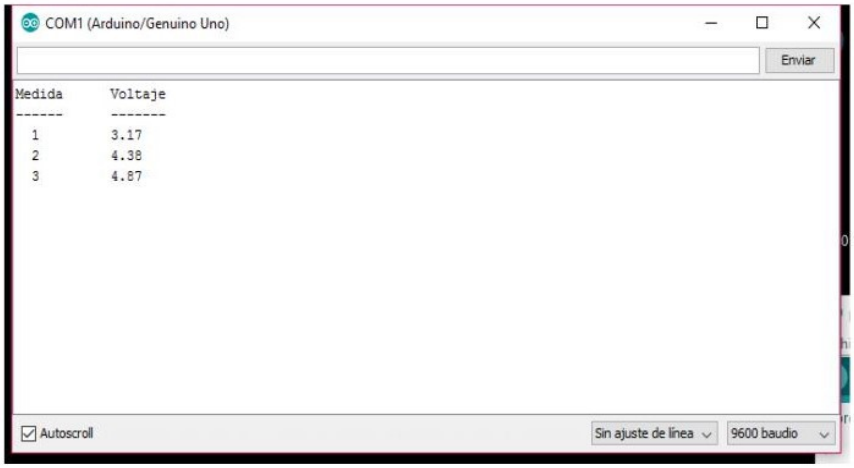


Fig.3.9. Voltajes medidos por el pin A0 en los tres supuestos establecidos en el texto

Como puede apreciarse, el voltaje se aproxima a 5 V, conforme la intensidad de luz recibida por la fotorresistencia se aproxima a cero. Desconectando el circuito del PC, vamos a analizar con un multímetro la resistencia de la fotorresistencia en los tres supuestos. Para ello, procedemos como indica la Fig.3.10, conectando un multímetro a ambas patas de la fotorresistencia en la placa de pruebas. Aplicando los tres supuestos obtenemos valores de: 1,6 kΩ con la luz artificial del salón, 77 kΩ con la luz ambiental de la ventana y 550 kΩ cuando la fotorresistencia se cubre con la mano. Vemos pues que la resistencia de 1 kΩ, en serie con la fotorresistencia, cumple la misión, no sólo de evitar interferencias en el botón, sino también la de fraccionar el voltaje detectado por el pin A0. Los resultados de la Fig.3.9 nos indican que si queremos activar un LED en la oscuridad, tendremos que elegir un voltaje para el pin A0 mayor que 4,5 V.

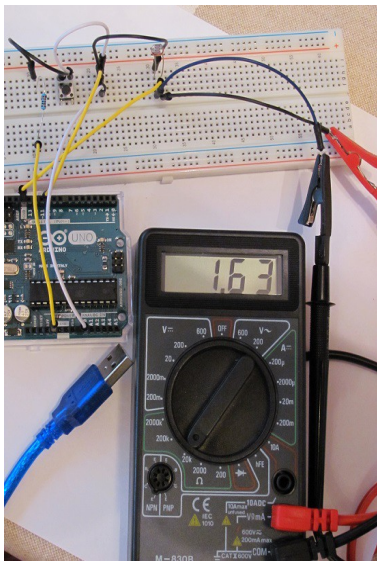


Fig.3.10. Medida de la resistencia de la fotorresistencia en el primer supuesto

3.2.2. Proyecto 7b: Activación de un LED cuando la luz ambiente es poca

En el circuito necesario para encender un LED cuando la luz ambiente es escasa y apagarlo cuando hay luz suficiente, eliminamos el botón y añadimos el LED y la resistencia en serie de 220 Ω, como muestra la Fig.3.11. Asignando al pin 13 la responsabilidad del encendido y apagado el LED, su estado dependerá de que el voltaje leído por pin A0 sea mayor o menor que un umbral de 4,5 V.

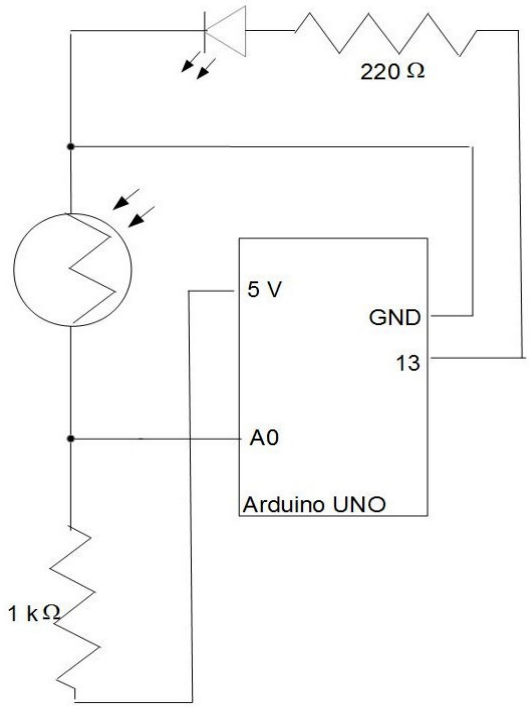


Fig.3.11. Circuito que enciende o apaga un LED según la luz ambiental

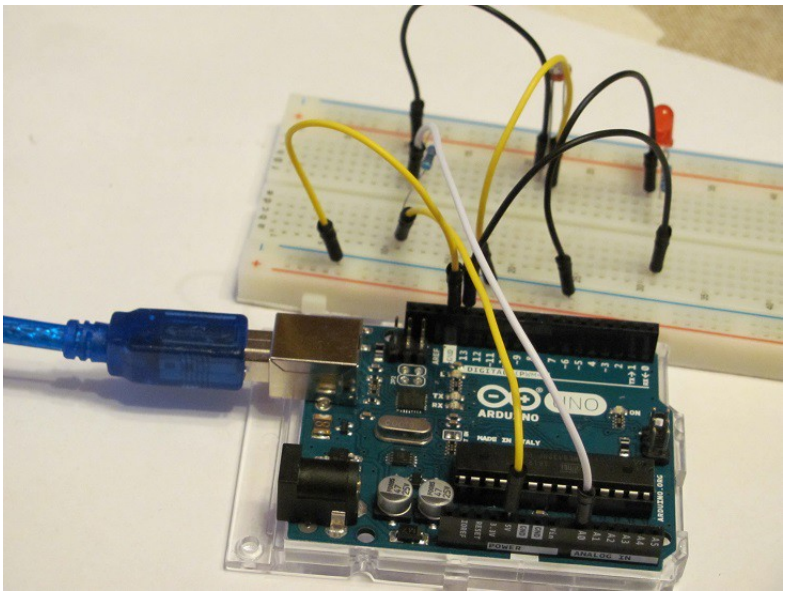


Fig. 3.12. Montaje del circuito de la Fig.3.11

Prog.3.3. Sketch del Proyecto 7b

```
/*
Proyecto 7b

Enciende un LED cuando oscurece y lo apaga si hay luz
*/

// Usamos el pin analógico A0 para leer el voltaje
const int PinVoltaje = A0;
const int PinLed = 13;
const float VoltajeOscuro = 4.5;

void setup()
{
  pinMode(PinLed, OUTPUT);
}

void loop()
{
  float Voltaje;

  Voltaje = LeerVoltaje(PinVoltaje);
  if (Voltaje >= VoltajeOscuro)
  // En caso que el voltaje supere el umbral enciende el LED
  {
    digitalWrite(PinLed, HIGH);
  }
  else
  // Si el voltaje no supera el umbral apaga el LED
  {
    digitalWrite(PinLed, LOW);
  }
}

float LeerVoltaje(int Pin)
{
  // Definimos la función LeerVoltaje()
  return (analogRead(Pin) * 5. / 1024.);
}
```

3.3. PROYECTO 8: DETECCIÓN DE OBJETOS PRÓXIMOS POR ULTRASONIDOS CON UN HC-SR04. SENSOR DE MOVIMIENTO

El sensor de proximidad HC-SR04 (Fig.3.13) es una unidad de alcance ultrasónico (Ultrasonic Ranging Module) capaz de determinar a que distancia se encuentran objetos cercanos (entre 2 cm y 4 m). Para ello, el sensor emite un paquete de ultrasonidos, que rebotan en un objeto próximo. Posteriormente los ultrasonidos rebotados son analizados por el sensor, resultando de su análisis la distancia entre el sensor y el objeto. El sensor HC-SR04 posee cuatro clavijas. La primera y la cuarta son respectivamente las conexiones a la fuente de 5 V (polo positivo) y a tierra (polo negativo). Las clavijas de en medio, que poseen etiquetas de Trigger y Echo, tienen que ver con los paquetes de ultrasonidos de salida (emitido) y de entrada (rebotado).

La forma de proceder es como sigue. Primero, se crea un impulso eléctrico cuadrado de 10 μs de anchura. A continuación, el sensor HC-SR04 transforma el impulso eléctrico en un paquete de ultrasonidos de 40 kHz, detectando si existe una señal de retorno. En caso de detectar una señal de retorno se analiza su anchura, la cual tiene que ver con la distancia recorrida por el paquete de ultrasonidos. La distancia al objeto es A × v / 2, en la que A es la anchura del impulso en μs y v es la velocidad del sonido (340 m/s =0,034 cm/μs).

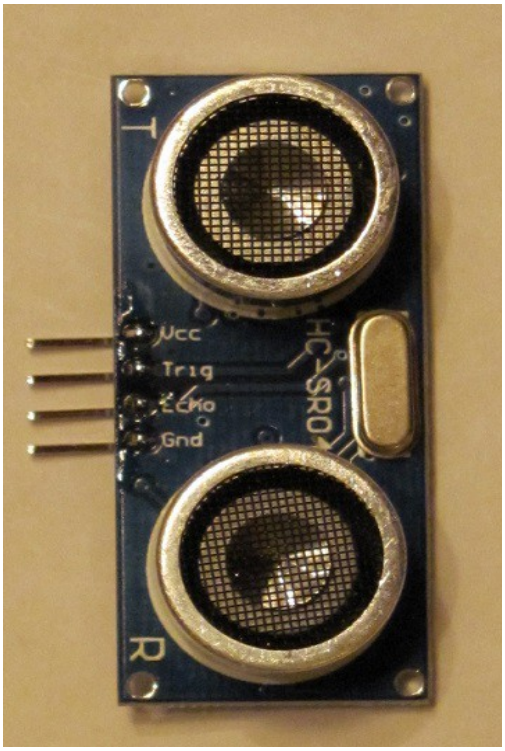


Fig.3.13. Sensor de proximidad HC-SR04

3.3.1. Proyecto 8a: Medida de distancias a objetos próximos

Con ayuda del sensor de proximidad HC-SR04 vamos a medir, a intervalos de 5 s, la distancia en centímetros a una mano que se acerca y aleja del sensor. El circuito y el montaje se muestran en las Figs.3.14 y 3.15.

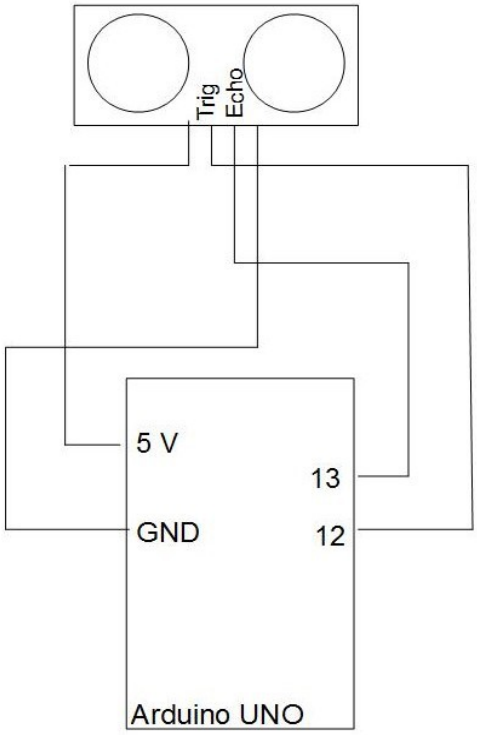


Fig.3.14. Circuito para medir la distancia a una mano que se acerca y aleja

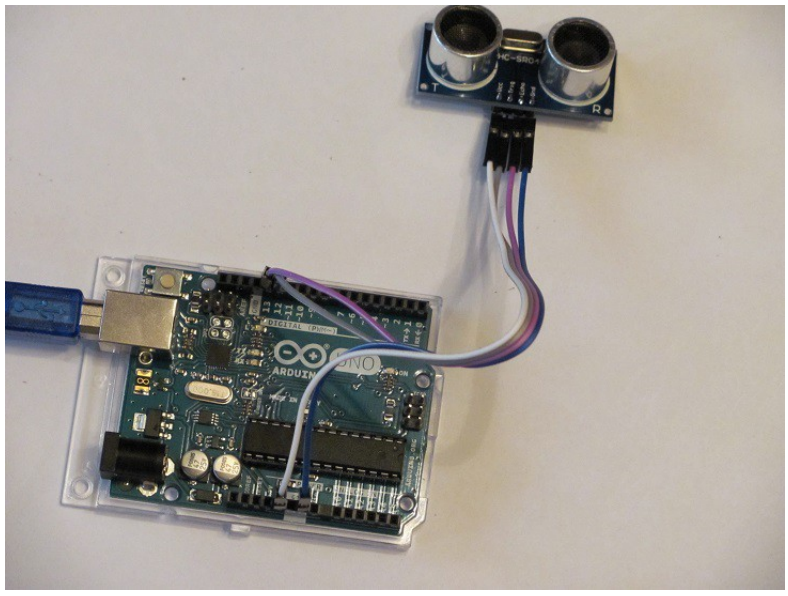


Fig.3.15. Montaje del circuito de la Fig. 3.14

El sketch que permite obtener una tabla en el monitor de serie con la anchura de impulso en ms y la distancia en cm de la mano se muestra en Prog.3.4.

Prog.3.4. Sketch del Proyecto 8a

```
/*
Proyecto 8a

Medimos distancia a un objeto próximo
*/

// Definimos los pines de los impulsos de entrada y salida
const int PinPulsoSalida = 12;
const int PinPulsoEntrada = 13;
int Medida = 1;

void setup()
{
  pinMode(PinPulsoEntrada, INPUT);
  pinMode(PinPulsoSalida, OUTPUT);
  Serial.begin(9600);
  Serial.print("Medida      ");
  Serial.print("Anchura     ");
  Serial.println("Distancia (cm)");
  Serial.print("------      ");
  Serial.print("-------     ");
  Serial.println("------------");
}

void loop()
{
  float Anchura;
  float Distancia;
  // Generamos el impulso de salida
  digitalWrite(PinPulsoSalida, HIGH);
  delayMicroseconds(10);
  digitalWrite(PinPulsoSalida, LOW);
  // Determina la anchura del impulso de entrada
  Anchura = pulseIn(PinPulsoEntrada, HIGH);
  // Determina la distancia al objeto
  Distancia = Anchura * 0.034 / 2.;
  if (Distancia < 800)
  // Evita interferencias con la señal de entrada
  {
    Serial.print("  ");
    Serial.print(Medida);
    Serial.print("         ");
    Serial.print(Anchura);
    Serial.print("         ");
    Serial.println(Distancia);
    Medida++;
    delay(5000);
  }
}
```

3.3.2. Proyecto 8b: Sensor de movimiento

En este caso se conseguirá que una luz LED se encienda cuando el sensor de proximidad HC-SR04 detecte un cambio brusco en las distancias medidas, es decir, cuando algo se mueve. Para ello se propone el circuito de la Fig.3.16, el correspondiente montaje de la Fig.3.17 y el sketch de Prog.3.5.

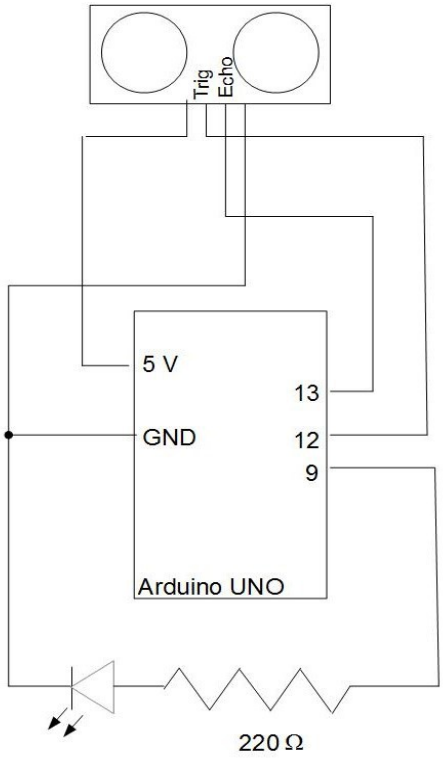


Fig.3.16. Circuito del Proyecto 8b

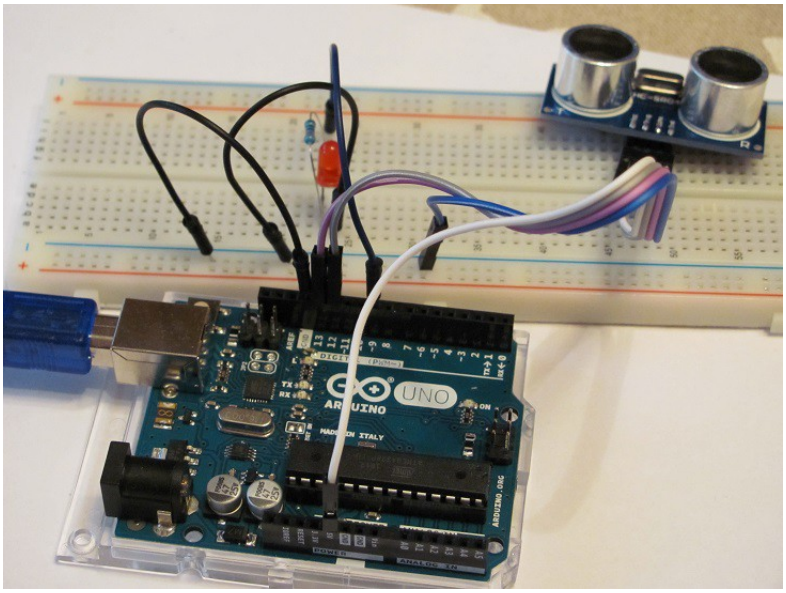


Fig.3.17. Montaje del circuito de la Fig.3.16

Prog.3.5. Sketch del Proyecto 8b

```
/*
Proyecto 8b

Detectamos si existe un objeto en movimiento
*/

// Definimos los pines de los impulsos de entrada y salida
const int PinPulsoSalida = 12;
const int PinPulsoEntrada = 13;
const int PinLed = 9;

void setup()
{
  pinMode(PinPulsoEntrada, INPUT);
  pinMode(PinPulsoSalida, OUTPUT);
  pinMode(PinLed, OUTPUT);
}

void loop()
{
  float Distancia1, Distancia2;
    Distancia1 = ValorDistancia();
    Distancia2 = ValorDistancia();
  if (Distancia1 > 2. * Distancia2)
  // Detecta si hay un objeto que se mueve
  {
    digitalWrite(PinLed, HIGH);
    delay(10000);
  }
  else
  {
    digitalWrite(PinLed, LOW);
  }
}

float ValorDistancia()
{
  float Anchura;
  // Generamos el impulso de salida
  digitalWrite(PinPulsoSalida, HIGH);
  delayMicroseconds(10);
  digitalWrite(PinPulsoSalida, LOW);
  // Determina la anchura del impulso de entrada
  Anchura = pulseIn(PinPulsoEntrada, HIGH);
  // Hace otra medida una décima de segundo más tarde
  delay(10);
  return(Anchura * 0.034 / 2.);
}
```

3.4. PROYECTO 9: SENSOR DE CONTACTO TTP223-BA6

La Fig.3.18 muestra el sensor de contacto TTP223-BA6. Como puede apreciarse el sensor cuenta con tres clavijas: la primera de conexión a tierra (GND), la segunda de conexión a la fuente de 5 V de la placa Arduino y la última de señal (SIG), la cual indica, con un cambio en el voltaje, si se ha tocado la zona de círculos concéntricos. Nuestra primera misión será medir dicho cambio de voltaje.

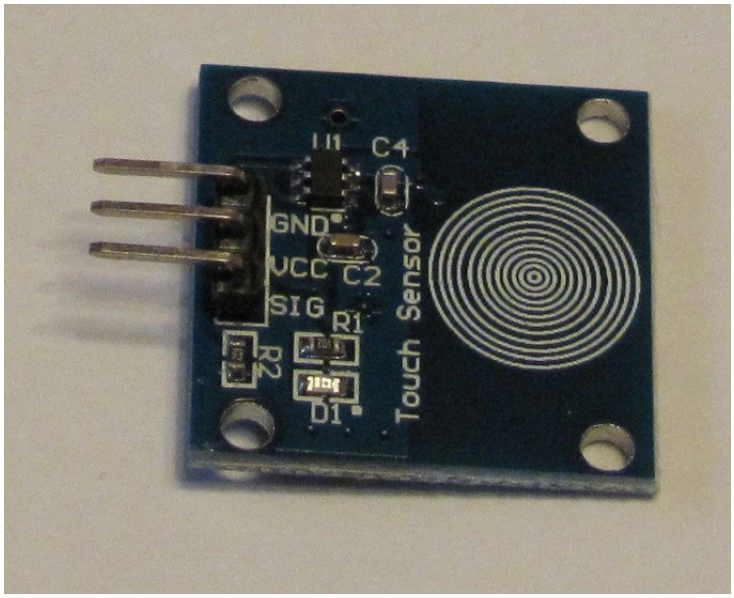


Fig.3.18. Sensor de contacto TTP223-BA6

3.4.1. Proyecto 9a: Análisis de la señal del sensor de contacto

En este caso el circuito (Fig.3.19), el montaje (Fig.3.20) y el sketch (Prog.3.6) serán muy similares al del sensor de temperatura del Proyecto 6. Con el pin A0 realizaremos medidas del voltaje a intervalos de 1 s. Como vemos en la tabla de la Fig.3.21, cuando tocamos el sensor de círculos concéntricos, después de 4 segundos, el voltaje de SIG cambia de 0 a 5 V.

3.4.2. Proyecto 9b: Enciende una luz LED al contacto con el sensor

Para que se encienda un LED al contacto con el sensor proponemos el circuito de la Fig.3.22. El montaje correspondiente a este circuito se muestra en la Fig.3.23. En último lugar, el sketch se transcribe en Prog.3.7. Es importante señalar que la condición de contacto no es `if(Voltaje ==5.)`. En su lugar se ha empleado `if(Voltaje > 1.)`. La razón de ello es que Voltaje no es estrictamente igual a 5 V, pudiendo ser 4,999 V.

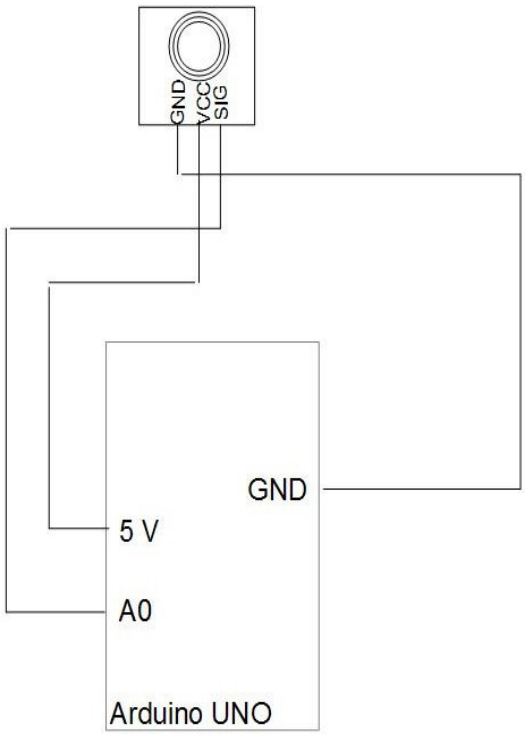


Fig.3.19. Circuito del Proyecto 9a

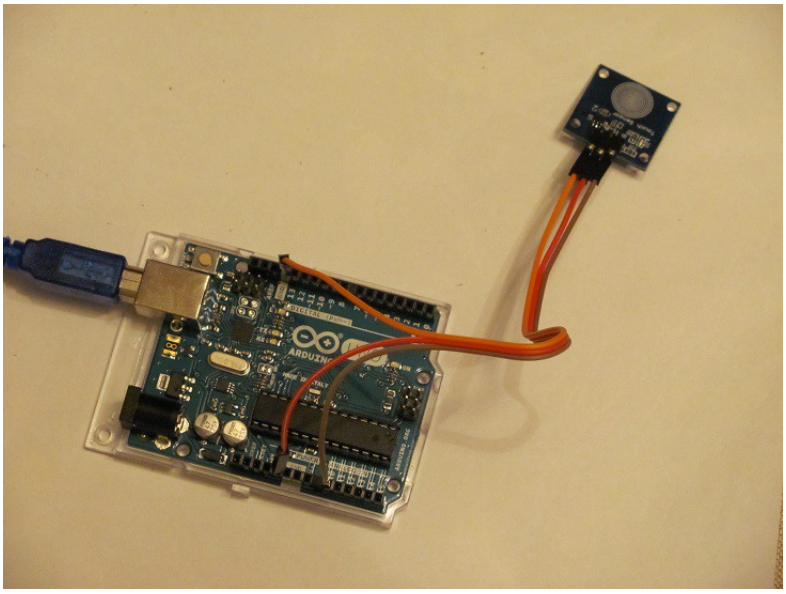


Fig.3.20. Montaje del Proyecto 9a

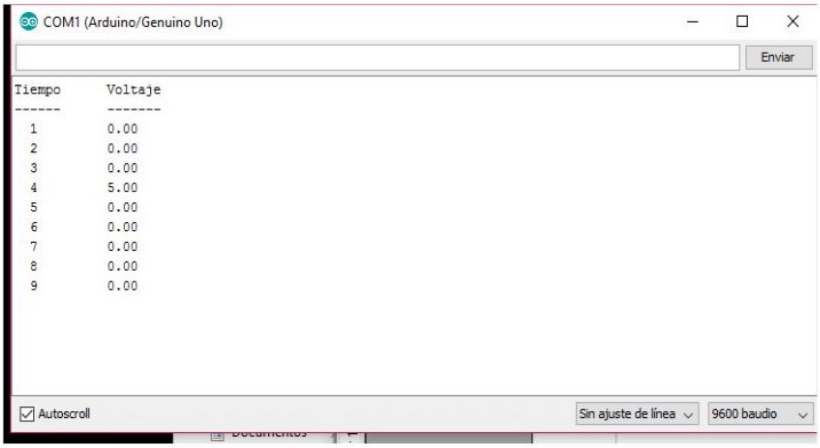


Fig.3.21. Tabla obtenida en el monitor de serie con el Proyecto 9a

Prog.3.6. Sketch del Proyecto 9a

```
/*
Proyecto 9a

Medimos el voltaje correspondiente al contacto con el sensor
*/

// Usamos el pin analógico A0 para leer el voltaje
const int PinVoltaje = A0;
int Tiempo0 = 1;
int Tiempo = 0;

void setup()
{
  // Creamos la cabecera de la tabla
  Serial.begin(9600);
  Serial.print("Tiempo      ");
  Serial.println("Voltaje     ");
  Serial.print("------      ");
  Serial.println("-------     ");
}

void loop()
{
  float Voltaje;

  // Definimos el tiempo
  Tiempo = Tiempo + Tiempo0;
  Voltaje = LeerVoltaje(PinVoltaje);
  // Se mandan los datos de Aruino a la ventana de serie del monitor.
  Serial.print("  ");
  Serial.print(Tiempo);
  Serial.print("         ");
  Serial.println(Voltaje);

  delay(Tiempo0*1000);
}

float LeerVoltaje(int Pin)
{
  // Definimos la función LeerVoltaje()
  return (analogRead(Pin) * 5. / 1024.);
}
```

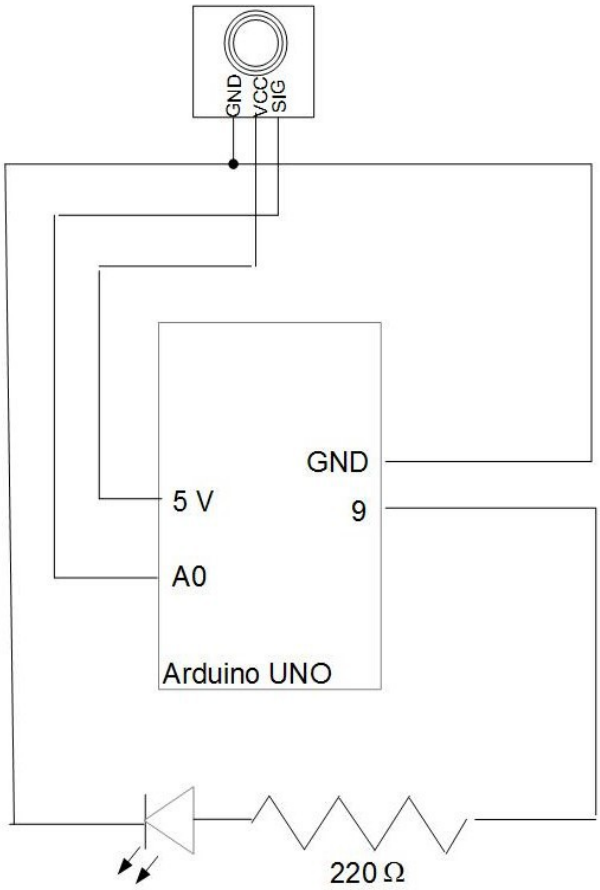


Fig.3.22. Circuito del Proyecto 9b

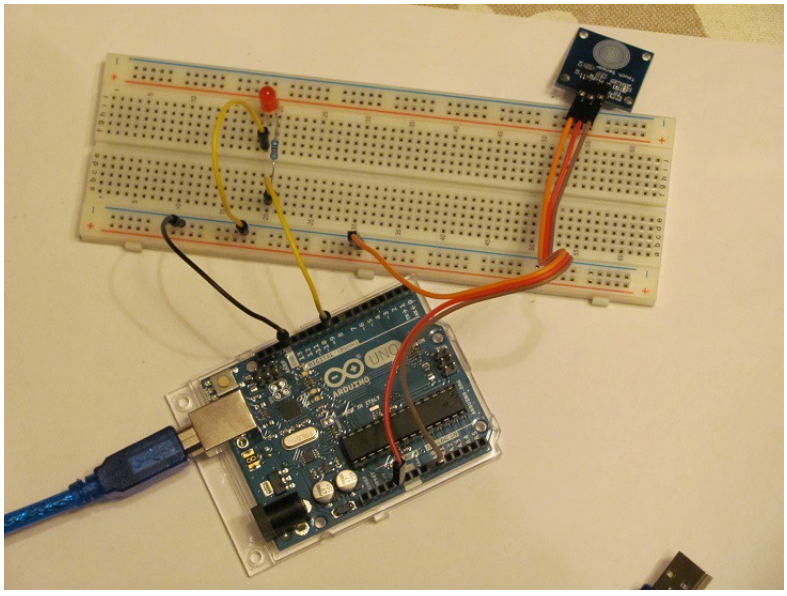


Fig.3.23. Montaje del Proyecto 9b

Prog.3.7. Sketch del Proyecto 9b

```
/*
Proyecto 9b

Enciende un LED al contacto con el sensor
*/

// Usamos el pin analógico A0 para leer el voltaje
const int PinVoltaje =A0;
const int PinLed =9;

void setup()
{
  pinMode(PinLed, OUTPUT);
}

void loop()
{
  float Voltaje;
  // Se mide el voltaje
  Voltaje = LeerVoltaje(PinVoltaje);
  if (Voltaje > 1.)
  // Si el voltaje es 5 V se ha producido el contacto
  {
    digitalWrite(PinLed, HIGH);
    delay(1000);
  }
  else
  {
    digitalWrite(PinLed, LOW);
  }
}

float LeerVoltaje(int Pin)
{
  // Definimos la función LeerVoltaje()
  return (analogRead(Pin) * 5. / 1024.);
}
```

3.5. PROYECTO 10: SENSOR DE DETECCIÓN DE LLAMA

El sensor de detección de llama es, como ya hemos indicado, un caso especial de sensor de infrarrojos. En este caso lo que detecta es la radiación infrarroja emitida por la llama, siendo especialmente sensible a la luz entre 760 y 1100 nm. Como muestra la Fig.3.24 se trata de un sensor muy parecido a un LED, aunque de color negro. Cuando la radiación infrarroja llega al sensor, modifica su resistencia interna, de manera muy parecida a como sucedía con la fotorresistencia en el

Proyecto 7. Hay que cuidar en este caso la polaridad del sensor, puesto que tiene un polo positivo (pata más larga) y uno negativo (pata más corta). Cuando la llama está apagada la resistencia interna del sensor, medida por un multímetro, es de 78 kΩ (Fig.3.25). Si acercamos al sensor la llama de un encendedor a gas, la resistencia baja a 33 kΩ.

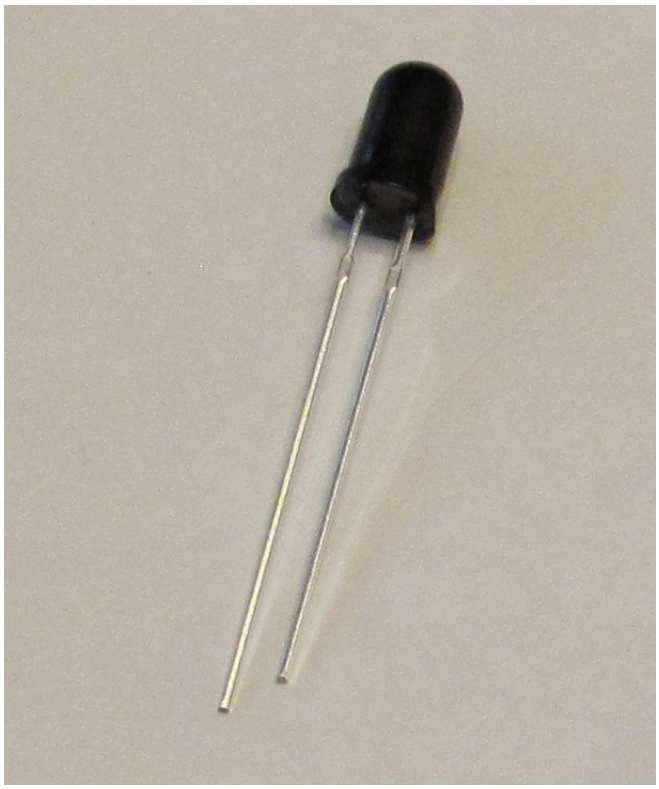


Fig.3.24. Sensor de llama

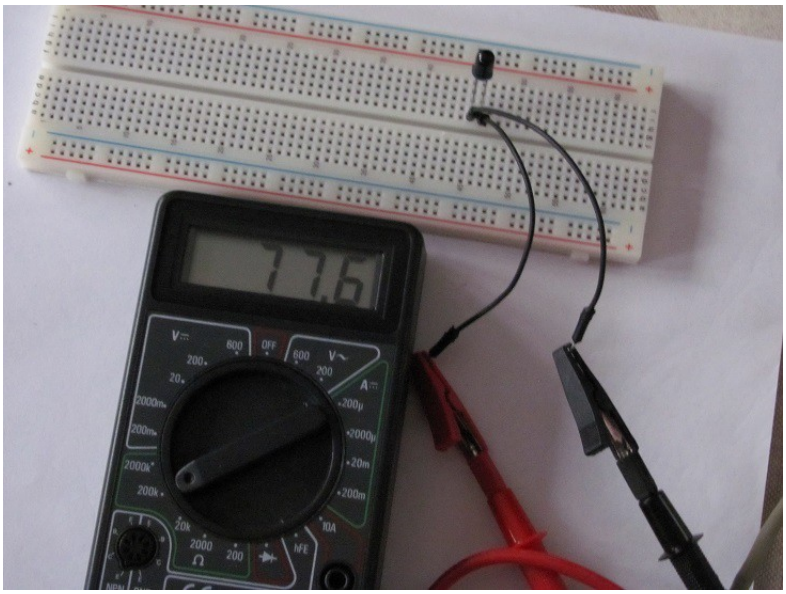


Fig.3.25. Medida con un multímetro de la resistencia interna de un sensor de llama

Es importante destacar que otras fuentes de calor, como las bombillas incandescentes, emiten en el infrarrojo cercano. Por tanto, conviene utilizar la luz natural como única fuente de luz, mientras se realizan experimentos con un sensor de llama.

3.5.1. Proyecto 10a: Calibración de un sensor de llama

Puesto que la placa Arduino no puede medir la resistencia directamente, tendremos que recurrir al truco, como ya hicimos en el Proyecto 7 de añadir una resistencia en serie de 1 MΩ, con el fin de fraccionar el voltaje. Las Figs.3.26 y 3.27 muestran el circuito y el montaje, respectivamente.

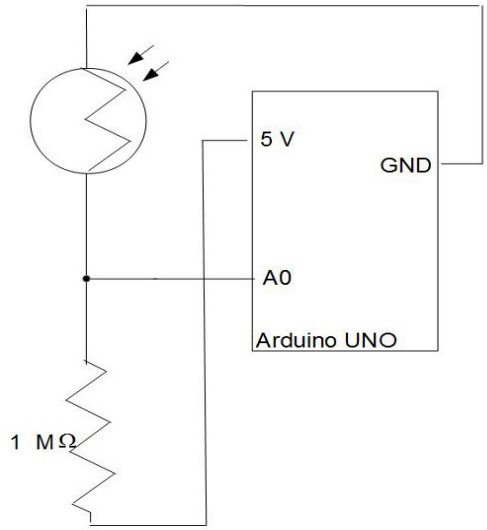


Fig.3.26. Circuito propuesto para calibrar un sensor de llama

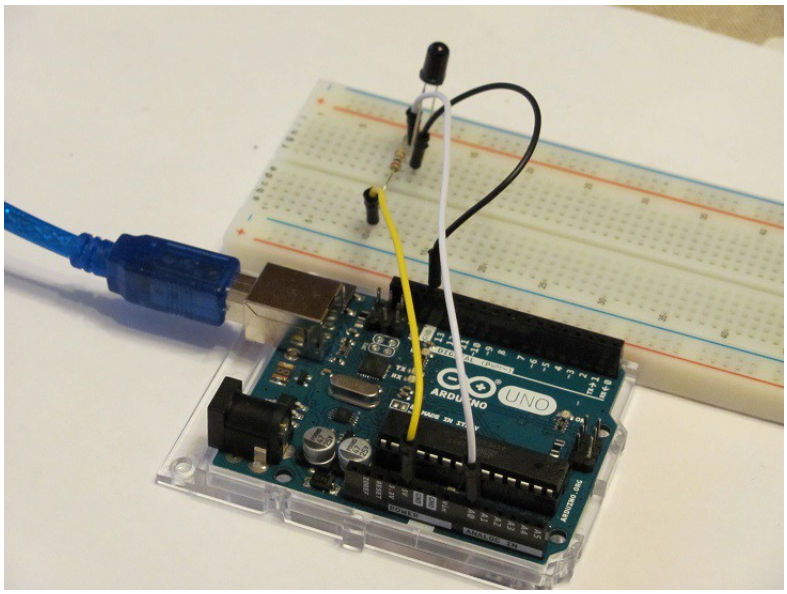


Fig.3.27. Montaje del circuito de la Fig.3.26

Por lo que se refiere al sketch, suprimimos las sentencias relativas al botón del Proyecto 7a, obteniendo Prog.3.8.

Prog.3.8. Sketch del Proyecto 10a

```
/*
Proyecto 10a

Calibramos el sensor de llama
*/

// Usamos el pin analógico A0 para leer el voltaje
const int PinVoltaje =A0;
int Medida = 1;

void setup()
{
  Serial.begin(9600);
  Serial.print("Medida      ");
  Serial.println("Voltaje     ");
  Serial.print("------      ");
  Serial.println("-------     ");
}

void loop()
{
  float Voltaje;
  Voltaje = LeerVoltaje(PinVoltaje);
  // Se mandan los datos de Aruino al monitor de serie
  Serial.print("  ");
  Serial.print(Medida);
  Serial.print("         ");
  Serial.println(Voltaje);
  Medida++;
  delay(1000);
}

float LeerVoltaje(int Pin)
{
  // Definimos la función LeerVoltaje()
  return (analogRead(Pin) * 5. / 1024.);
}
```

Si conectamos el montaje de la Fig.3.27 al PC, y subimos el sketch Prog.3.8 a la placa Arduino, obtenemos, después de abrir la ventana del monitor de serie, la tabla de la Fig. 3.28. Dicha tabla muestra los resultados del voltaje proporcionado por el pin A0, al poner en marcha un encendedor a gas entre la medidas 3 y 6. Vemos que el voltaje baja de 4,9 V, con el encendedor apagado, a 4,3 V, con encendedor encendido. De esta forma podemos afirmar que en caso de que el voltaje sea menor que 4,5 V habremos detectado una llama. Eso sí, siempre y cuando no interfieran otras fuentes de calor, como bombillas incandescentes.

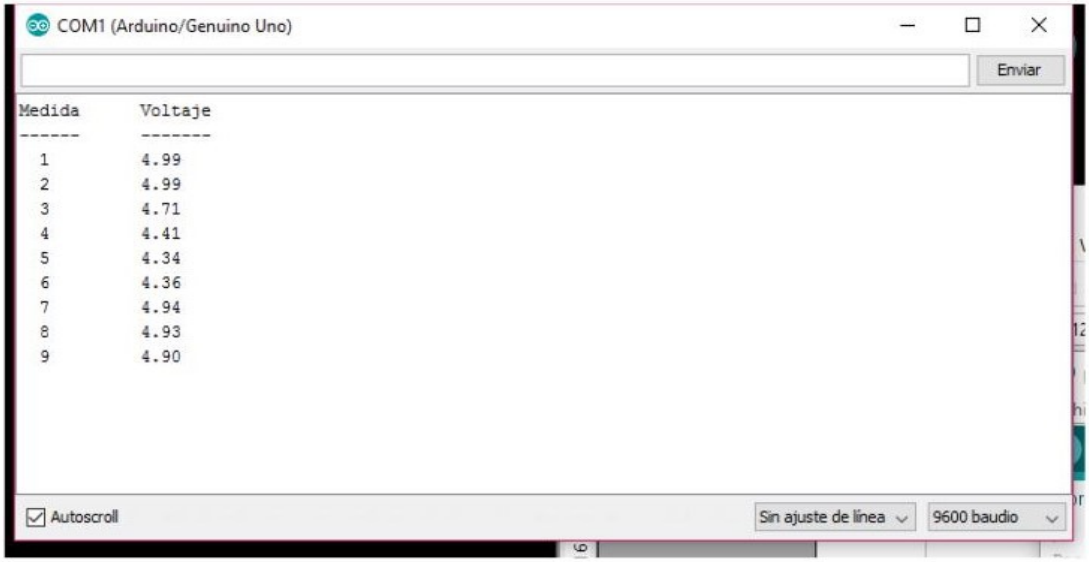


Fig.3.28. Disminución del voltaje entre las medidas 3 y 6 debido a la presencia de una llama

3.5.2. Proyecto 10b: Se enciende un LED cuando se detecta una llama

En el caso de que utilicemos un LED para detectar la presencia de una llama, utilizaremos el circuito y el montaje de las Figs.3.29 y 3.30. Por su parte, el sketch relativo a este proyecto puede verse en Prog.3.9.

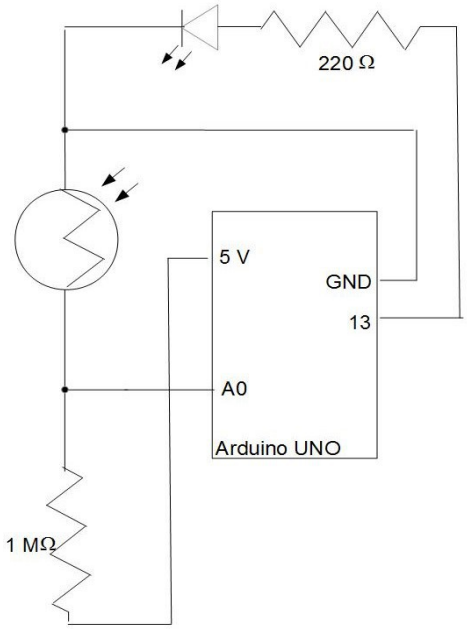


Fig.3.29. Circuito del Proyecto 10b

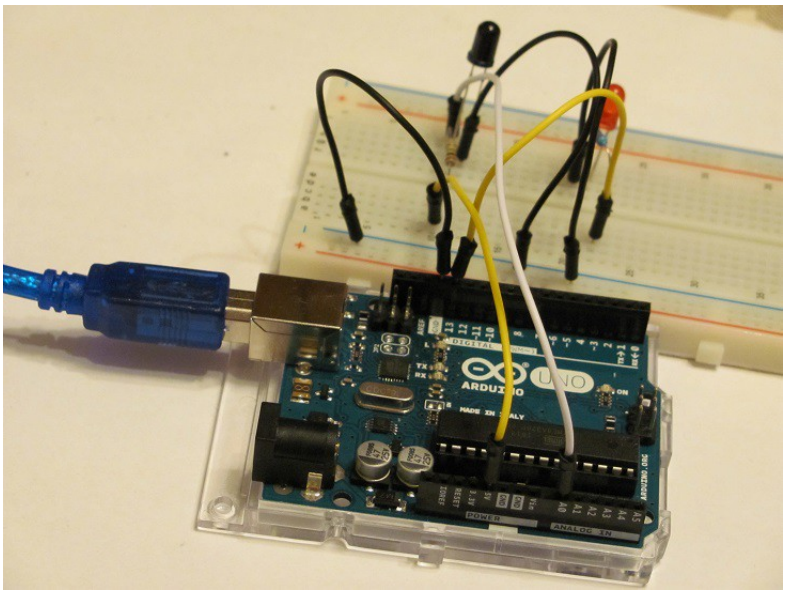


Fig.3.30. Montaje del Proyecto 10b

Prog.3.9. Sketch del Proyecto 10b

```
/*
Proyecto 10b

Enciende un LED cuando se acerca una llama
*/

// Usamos el pin analógico A0 para leer el voltaje
const int PinVoltaje = A0;
const int PinLed = 13;
const float VoltajeLlama = 4.5;

void setup()
{
  pinMode(PinLed, OUTPUT);
}

void loop()
{
  float Voltaje;

  Voltaje = LeerVoltaje(PinVoltaje);
  if (Voltaje <= VoltajeLlama)
  // En caso que el voltaje esté por debajo del umbral enciende el LED
  {
    digitalWrite(PinLed, HIGH);
  }
  else
  // Si el voltaje supera el umbral apaga el LED
  {
    digitalWrite(PinLed, LOW);
  }
}

float LeerVoltaje(int Pin)
{
  // Definimos la función LeerVoltaje()
  return (analogRead(Pin) * 5. / 1024.);
}
```

3.6. PROYECTO 11: DECODIFICADO DE SEÑALES INFRARROJAS CON UN SENSOR DE INFRARROJOS AX-1838HS

En la Fig.3.31 se muestran el emisor (mando a distancia) y el receptor (sensor) de señales infrarrojas distribuidos por el *Kuman Super Starter Kit*. Como puede apreciarse el mando a distancia posee 21 botones con funciones diferentes. Por su parte, el sensor es un AX-1838HS, capaz de detectar las señales infrarrojas emitidas por el mando proporcionado en el kit, o por cualquier otro mando a distancia comercial. El sensor AX-1838HS consta de tres patas. Mirado de frente, como muestra la Fig.3.31, la pata central se conecta a tierra (GND), la derecha a la fuente de 5 V de la placa Arduino y la izquierda proporciona la señal de salida Vs.

La señal infrarroja generada por un mando a distancia sigue un protocolo que permite su decodificación. Existe un protocolo universal para la decodificación de señales infrarrojas denominado NEC. Sin embargo, muchas firmas comerciales (como SONY, JVC, SAMSUNG, etc) emplean sus propios protocolos de decodificación.

En este proyecto utilizaremos el protocolo NEC, el cual viene caracterizado por un tren de impulsos digitales de aproximadamente 68 ms de duración. En la cabecera de dicho tren se encuentra una ráfaga de impulsos de 9 ms. A continuación, se halla un espacio libre de 4,5 ms. Después, en numeración binaria, se encuentra la dirección del dispositivo receptor (si se trata de un TV, DVD, etc) y su inverso lógico, con un total de 27 ms. Seguidamente, también en numeración binaria, aparece el comando (correspondiente a cada botón del mando a distancia) y su inverso lógico, con un total de 27 ms. Finalmente, en la cola del tren, está una ráfaga de impulsos de 562,5 µs de duración que indica el final de la transmisión.

La numeración binaria de la dirección del dispositivo receptor y del comando está caracterizada por una sucesión de 8 ceros y unos. En el protocolo NEC, el cero es una ráfaga de impulsos de 562,5 µs, seguida de un espacio de 562,5 µs. Por su parte, el uno es una ráfaga de impulsos de 562,5 µs, seguida de un espacio de 1,6875 ms. De esta forma, el espacio entre ráfagas es en tiempo el doble en el uno que en el cero.

Fig.3.31. Sensor de infrarrojos y mando a distancia empleados en el Proyecto 11

Afortunadamente no tenemos que programar la decodificación del tren de impulsos para el protocolo NEC (o cualquier otro). El trabajo de programarlo ya ha sido realizado por Ken Shirriff. El conjunto de librerías necesarias, que deben incluirse en el Sketch, se denominan IRremote.h y pueden descargarse del sitio WEB http://z3t0.github.io/Arduino-IRremote/. La forma de proceder es la siguiente. En primer lugar, se descarga la carpeta comprimida zip. Seguidamente, se descomprime. A continuación, se traslada la carpeta descomprimida a Disco local (C:) >Archivos de programa (x86) > Arduino > libraries. Finalmente, hay que extraer la carpeta RobotIRremote de la carpeta libreries para evitar confusiones.

Para simplificar las cosas en el Proyecto 11 trabajaremos únicamente con los botones 1, 2 y 3 del mando a distancia. Estos corresponden a los comandos en numeración hexadecimal: FF30FC, FF18E7 y FF7A85. La Tabla 3.1 muestra los comandos en numeración hexadecimal de cada uno de los 21 botones del mando a distancia del kit.

Tabla 3.1. Numeración hexadecimal de los distintos botones del mando a distancia

Número	Función	Numeración
1	CH-	FFA25D
2	CH	FF629D
3	CH+	FFE21D
4	PREV	FF22DD
5	NEXT	FF02FD
6	PLAY/PAUSE	FFC23D
7	VOL-	FFE01F
8	VOL+	FFA857
9	EQ	FF906F
10	0	FF6897
11	100	FF9867
12	200	FFB04F
13	1	FF30CF
14	2	FF18E7
15	3	FF7A85
16	4	FF10EF
17	5	FF38C7
18	6	FF5AA5
19	7	FF42BD
20	8	FF4AB5
21	9	FF52AD

3.6.1. Proyecto 11a: Decodificación de los botones 1, 2 y 3 del mando a distancia

Como primer paso debemos comprobar que el mando a distancia contiene pila y es un emisor de radiación infrarroja. Para ello, podemos utilizar una cámara fotográfica digital como detector de luz infrarroja y visualizar el destello que se produce cuando pulsamos un botón del mando (Fig.3.32). En el Proyecto 11a trataremos de decodificar las señales infrarrojas de los botones 1, 2 y 3 del mando a distancia. Para ello, se propone el circuito y el montaje de las Figs.3.33 y 3.34. El sketch requerido se muestra en Prog.3.10. En los resultados impresos por el monitor de serie está, en primer lugar, si el protocolo es el universal NEC. A continuación, como puede apreciarse en la Fig.3.36, cada vez que se pulse un botón aparecerá un mensaje distinto según el botón que se haya pulsado.

Fig.3.32. Destello luminoso detectado por una cámara digital al apretar un botón del mando a distancia

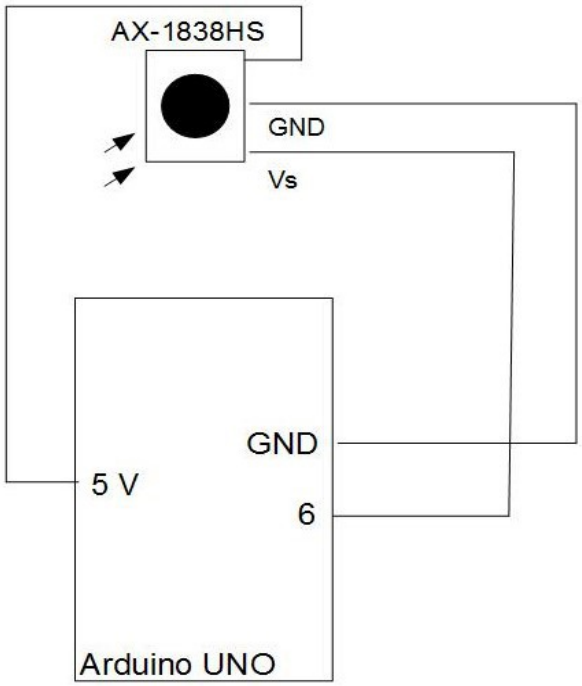


Fig.3.33. Circuito del Proyecto 11a

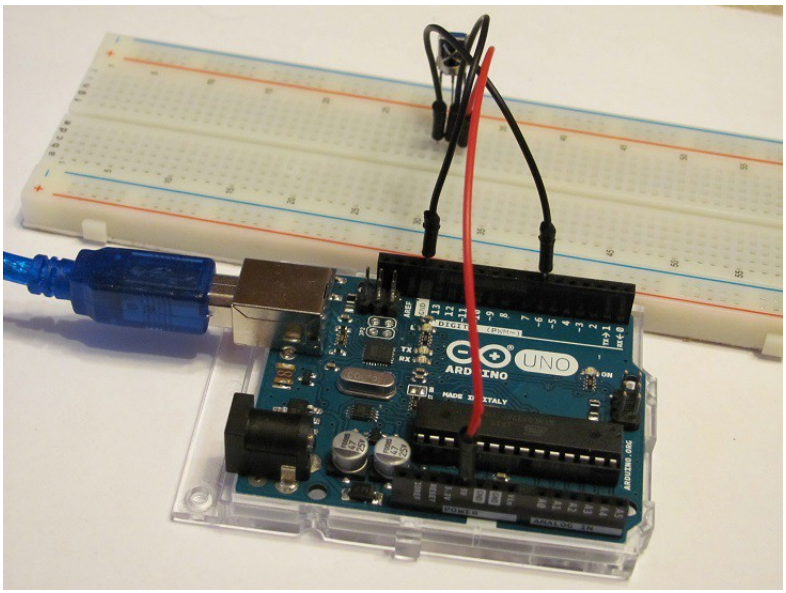


Fig.3.34. Montaje del Proyecto 11a

Prog.3.10. Sketch del Proyecto 11a

```
/*
Proyecto 11a

Decodificación de los botones 1, 2 y 3 en un mando
a distancia
*/

#include "IRremote.h"

const int PinSensor = 6;

IRrecv sensor(PinSensor);
decode_results resultados;

void setup()
{
  Serial.begin(9600);
  Serial.println("Test de decodificacion de los botones del mando a distancia");
  // Se pone en marcha el sensor
  sensor.enableIRIn();
}


void loop()
{
  if (sensor.decode(&resultados))
  //¿Se ha recibido la señal?
  {
    if (resultados.decode_type == NEC)
      {
        Serial.println("El protocolo es NEC");
        comparaIR();
      }
      else
      {
        Serial.println("El protocolo no es NEC");
        delay(500);
      }
     sensor.resume();
  }
}
```

```
void comparaIR()
// Compara el valor hexadecimal de la señal con los botones

{
 if (resultados.value == 0xFF30CF)
 {
  Serial.println("Se ha pulsado el 1");
 }
 else
 {
   if (resultados.value == 0xFF18E7)
   {
    Serial.println("Se ha pulsado el 2");
   }
   else
   {
    if (resultados.value == 0xFF7A85)
    {
     Serial.println("Se ha pulsado el 3");
    }
    else
    {
     Serial.println("Se ha pulsado un boton distinto de 1, 2 o 3");
    }
   }
 }
 delay(500);
}
```

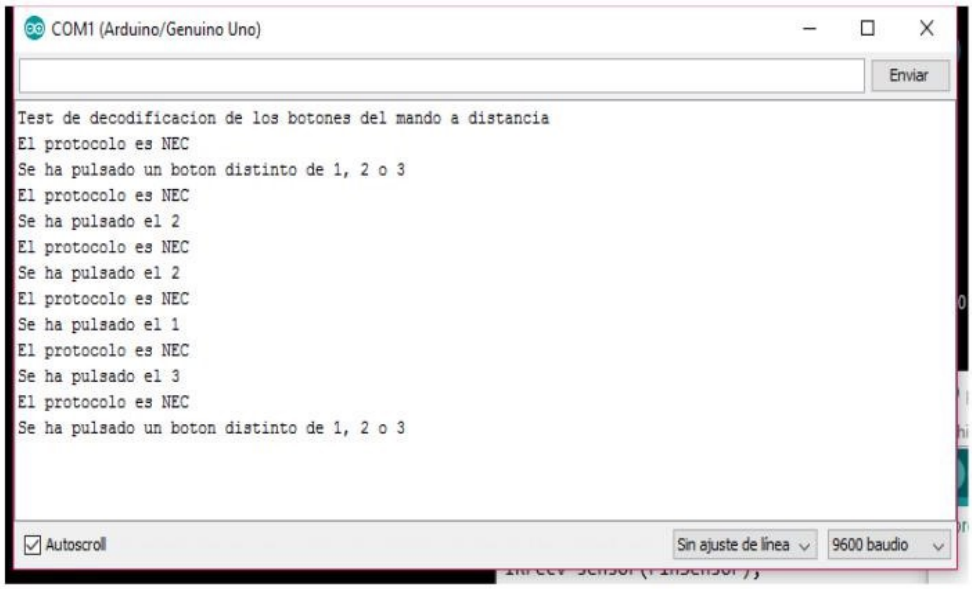


Fig.3.36. Mensajes obtenidos al pulsar distintos botones del mando a distancia

3.6.2. Proyecto 11b: Un LED RGB que cambia de color usando el mando a distancia

En este proyecto conseguiremos que un LED RGB cambie de color según se pulsen los botones 1, 2 o 3 del mando a distancia. Para ello utilizaremos el circuito de la Fig.3.37, el montaje de la Fig.3.38 y el sketch de Prog.3.11.

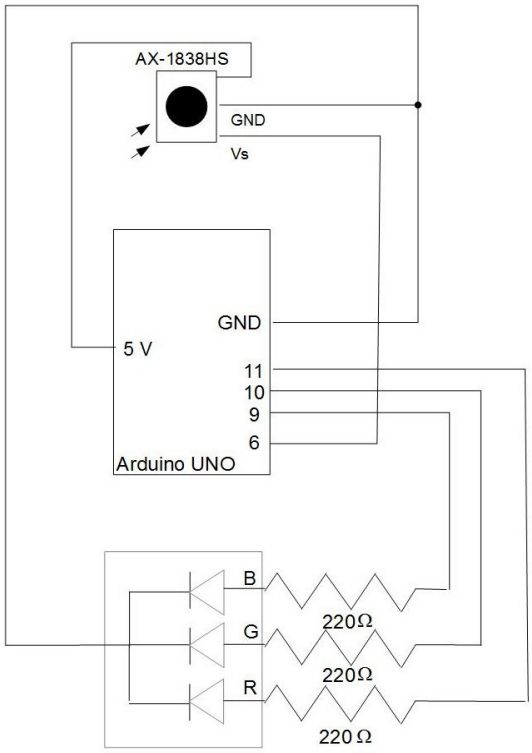


Fig.3.37. Circuito del Proyecto 11b

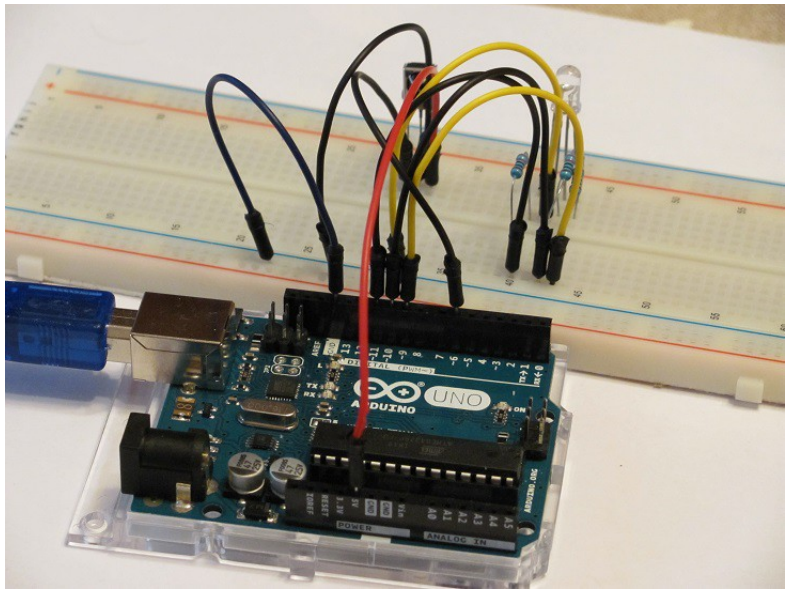


Fig.3.38. Montaje del circuito de la Fig.3.37

Prog.3.11. Sketch del Proyecto 11b

```
/*
Proyecto 11b

Aparecen los colores rojo, verde y azul al pulsar
los botones 1, 2 y 3 del mando a distancia. Aparece
el color blanco con cualquier otro botón
*/

#include "IRremote.h"

const int PinSensor = 6;
const int PinRojo = 11;
const int PinVerde = 10;
const int PinAzul = 9;

IRrecv sensor(PinSensor);
decode_results resultados;

void setup()
{
  pinMode(PinRojo, OUTPUT);
  pinMode(PinVerde, OUTPUT);
  pinMode(PinAzul, OUTPUT);
  // Se pone en marcha el sensor
  sensor.enableIRIn();
}


void loop()
{
  if (sensor.decode(&resultados))
  //¿Se ha recibido la señal?
  {
    comparaIR();
    sensor.resume();
  }
}
```

```
void comparaIR()
// Compara el valor hexadecimal de la señal con los botones

{
 if (resultados.value == 0xFF30CF)
 {
  Serial.println("Se ha pulsado el 1");
 }
 else
 {
   if (resultados.value == 0xFF18E7)
   {
    Serial.println("Se ha pulsado el 2");
   }
   else
   {
    if (resultados.value == 0xFF7A85)
    {
     Serial.println("Se ha pulsado el 3");
    }
    else
    {
     Serial.println("Se ha pulsado un boton distinto de 1, 2 o 3");
    }
   }
 }
 delay(500);
}
```

3.7. PROYECTO 12: SENSOR DE VIBRACIÓN SW-520D

El sensor de vibración SW-520D (Fig.3.39) es capaz de percibir cambios bruscos de la inclinación mediante un cambio de la resistencia eléctrica. El diseño del sensor lo constituyen dos esferas conductoras dentro de una carcasa cilíndrica de metal. Si el sensor permanece en posición vertical, o se inclina lentamente, existe una conexión eléctrica entre las esferas y una resistencia eléctrica de aproximadamente 1 Ω. Cuando se hace vibrar al sensor, o se inclina hasta los 90º, la conexión entre las dos esferas metálicas se pierde, aumentando la resistencia eléctrica considerablemente. Todo esto se puede comprobar conectando las dos patas del sensor a un multímetro (Fig.3.40).

3.7.1. Calibración del sensor de vibración

Al igual que hemos hecho en proyectos anteriores, procedemos a calibrar el sensor. Para ello proponemos el circuito de la Fig.3.41, el montaje de la Fig.3.42 y el sketch de Prog.3.12 (que es idéntico al del Proyecto 10a). Obsérvese que, en este caso, se ha conectado una resistencia en serie de 220 Ω, lo cual permite fraccionar el voltaje del pin A0 adecuadamente. De esta forma, obtenemos la tabla de la Fig.3.43 en el monitor de serie. Dicha tabla nos indica que existe vibración (entre las medidas 4 y 7) cuando se producen voltajes mayores que 0,1 V. La posición de reposo corresponde a un voltaje de 0,05-0,08 V.

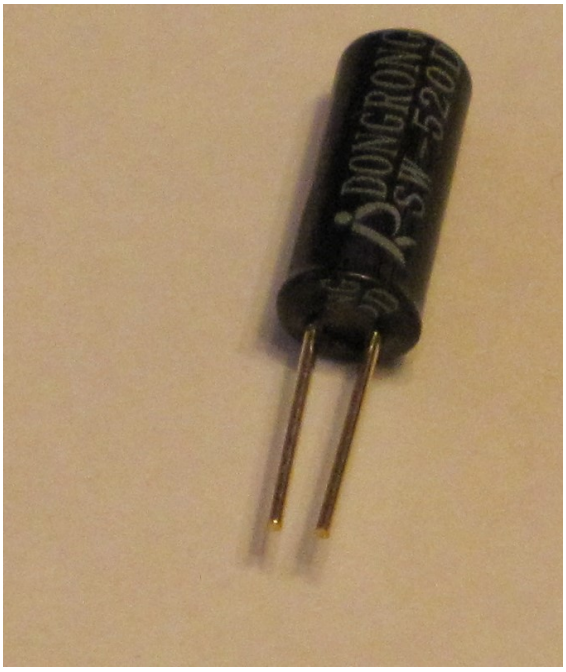


Fig.3.39. Sensor de vibración SW-520D

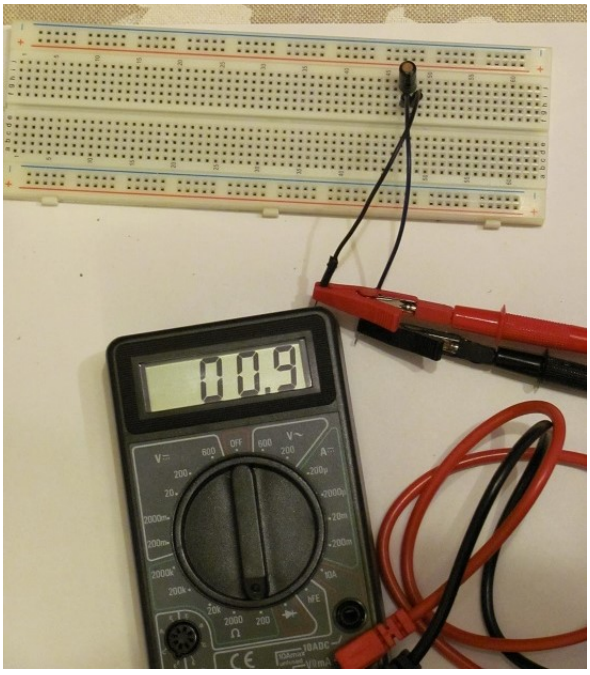


Fig.3.40. Resistencia eléctrica de un sensor de vibración en posición estática

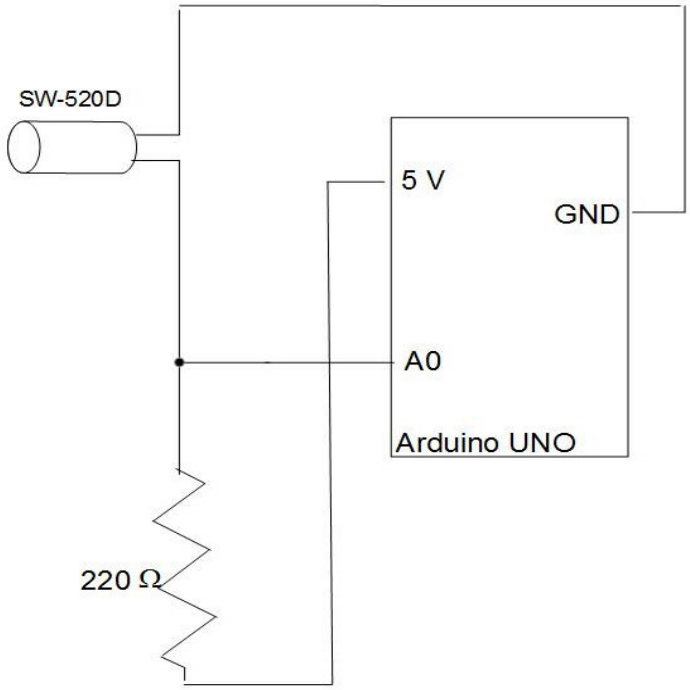


Fig.3.41. Circuito para la calibración del sensor de vibración

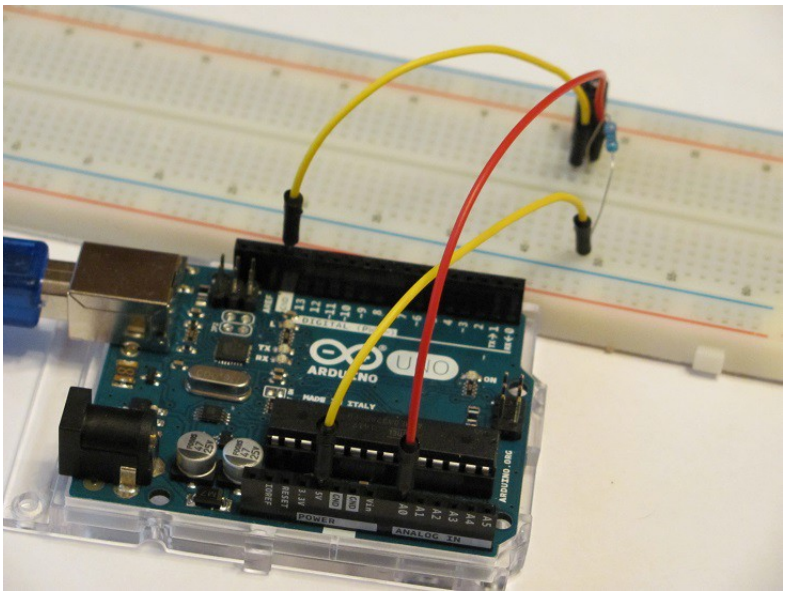


Fig.3.42. Montaje del Proyecto 12a

Prog.3.12. Sketch del Proyecto 12a

```
/*
Proyecto 12a

Calibramos el sensor de vibración
*/

// Usamos el pin analógico A0 para leer el voltaje
const int PinVoltaje =A0;
int Medida = 1;

void setup()
{
 Serial.begin(9600);
 Serial.print("Medida      ");
 Serial.println("Voltaje     ");
 Serial.print("------      ");
 Serial.println("------     ");
}

void loop()
{
 float Voltaje;
 Voltaje = LeerVoltaje(PinVoltaje);
 // Se mandan los datos de Aruino al monitor de serie
 Serial.print("  ");
 Serial.print(Medida);
 Serial.print("         ");
 Serial.println(Voltaje);
 Medida++;
 delay(1000);
}

float LeerVoltaje(int Pin)
{
 // Definimos la función LeerVoltaje()
 return (analogRead(Pin) * 5. / 1024.);
}
```

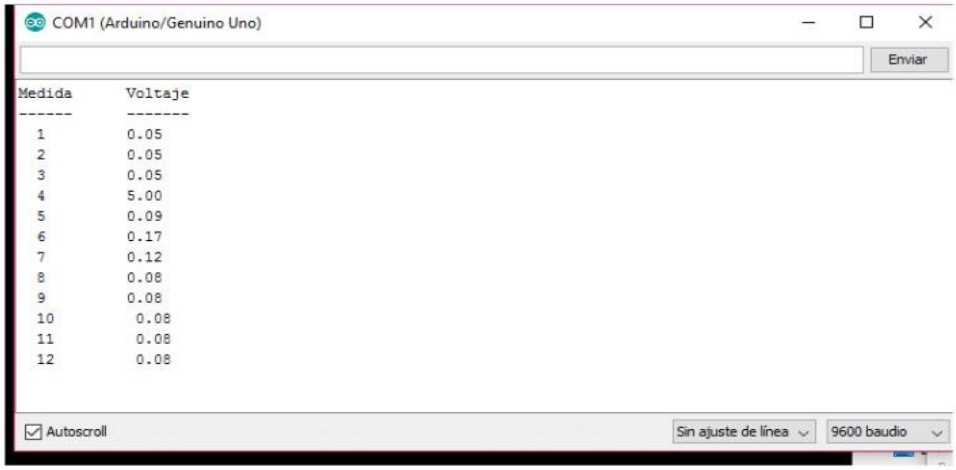


Fig.3.43. Datos del voltaje en las medidas realizadas para la calibración del sensor de vibración

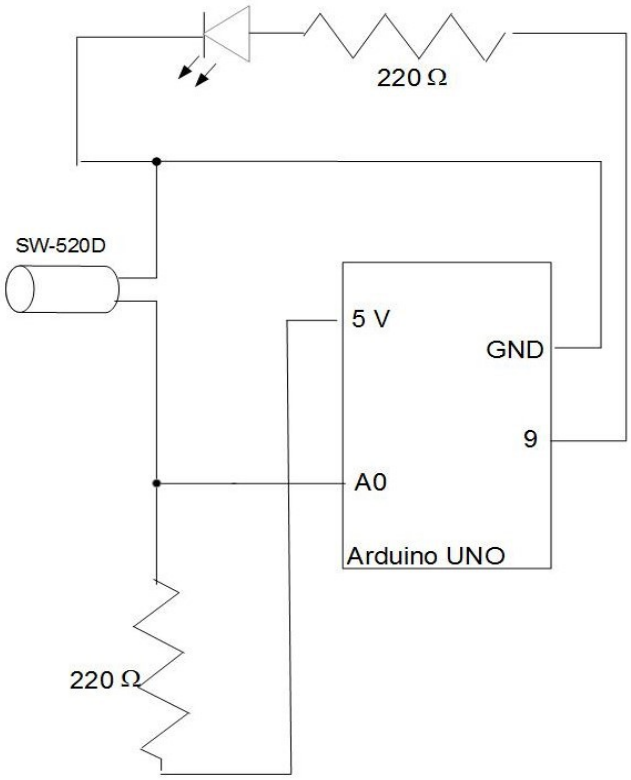


Fig.3.44. Circuito del Proyecto 12b

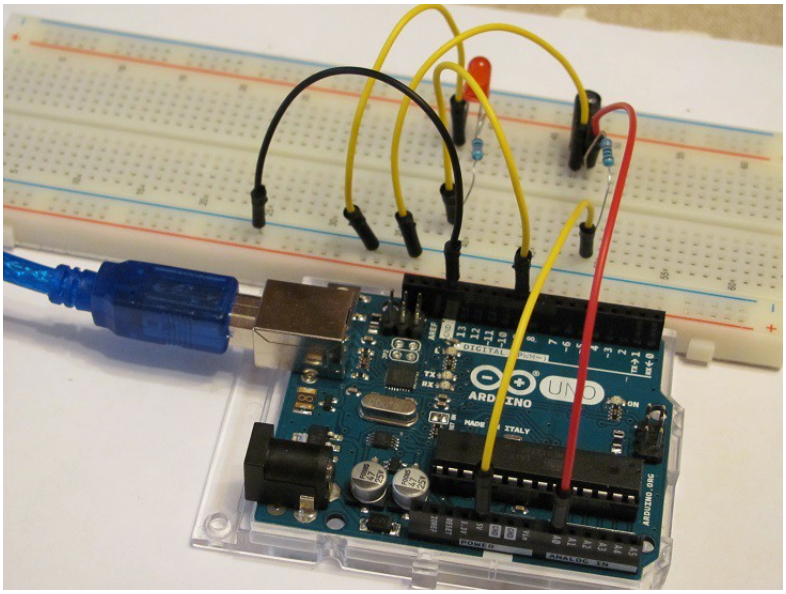


Fig.3.45. Montaje del Proyecto 12b

Prog.3.13. Sketch del Proyecto 12b

```
/*
Proyecto 12b

Enciende un LED cuando vibra el sensor
*/

// Usamos el pin analógico A0 para leer el voltaje
const int PinVoltaje = A0;
const int PinLed = 9;
const float VoltajeVibra = 0.10;

void setup()
{
  pinMode(PinLed, OUTPUT);
}

void loop()
{
  float Voltaje;

  Voltaje = LeerVoltaje(PinVoltaje);
  if (Voltaje >= VoltajeVibra)
  // En caso que el voltaje esté por encima del umbral enciende el LED
  {
    digitalWrite(PinLed, HIGH);
  }
  else
  // Si el voltaje está por debajo del umbral apaga el LED
  {
    digitalWrite(PinLed, LOW);
  }
}

float LeerVoltaje(int Pin)
{
  // Definimos la función LeerVoltaje()
  return (analogRead(Pin) * 5. / 1024.);
}
```

3.8. PROYECTO 13: EL ACELERÓMETRO ADXL335

Con el sensor de aceleración o acelerómetro ADXL335 (Fig.3.46) es posible medir aceleraciones en cualquiera de las tres direcciones de los ejes coordenados (X,Y, Z), siendo X la dirección paralela al lado más corto de la placa del sensor, Y la dirección paralela al lado más largo de la placa y Z la dirección perpendicular a ésta. Se trata de un acelerómetro de tipo capacitivo, que se diferencia claramente de los del tipo piezoeléctrico. Esta tecnología está basada en los cambios de la capacidad de un condensador doble debido al desplazamiento de la armadura central móvil por efecto de la aceleración. Es importante destacar que el acelerómetro ADXL335 debe conectarse a la fuente de alimentación de Arduino de 3,3 V (NUNCA a la de 5 V).

El sensor de aceleración ADXL335 posee cinco clavijas. Mirando de frente, según la Fig.3.6, la primera, empezando por arriba, corresponde a la conexión a la fuente de alimentación de 3,3 V. La segunda, la tercera y la cuarta tienen que ver con las señales de salida del acelerómetro según las direcciones de los ejes X, Y y Z. La quinta es la conexión a tierra (GND).

Nuestra primera misión será comprobar que el acelerómetro proporciona voltajes en las salidas X_OUT, Y_OUT y Z_OUT no nulos aunque el acelerómetro se encuentre en reposo. Dichos voltajes, que denominaremos VoltajeX0, VoltajeY0 y VoltajeZ0, deben restarse a los voltajes de salida, en caso de que exista aceleración. La sensibilidad del acelerómetro ADXL335 expresada en mV/g, cuyo valor es de 330, permitirá obtener la relación entre el voltaje y la aceleración. En este caso g se refiere a la aceleración de la gravedad.

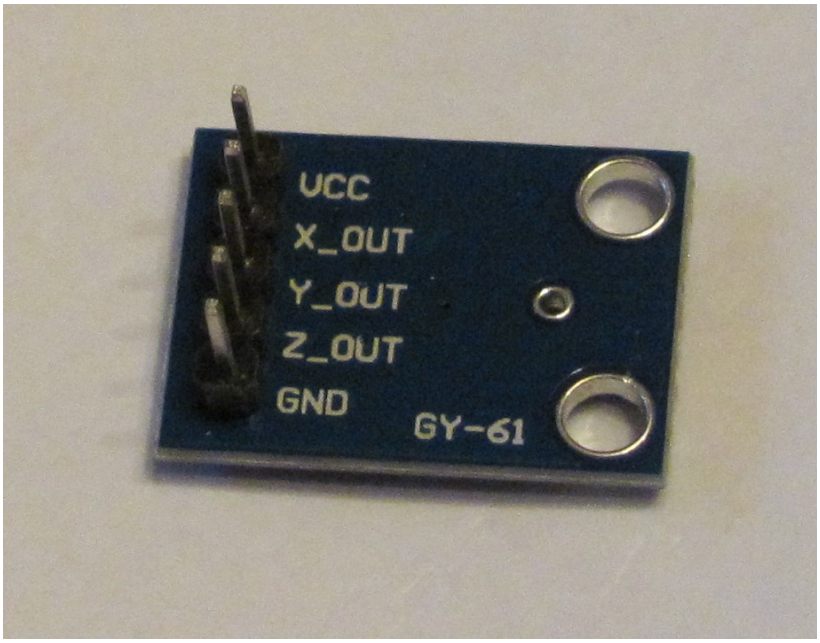


Fig.3.46. Sensor de aceleración ADXL335

3.8.1. Proyecto 13a: Calibración del acelerómetro

Una vez obtenidos los voltajes VoltajeX0, VoltajeY0 y VoltajeZ0 con el sensor en reposo podremos calibrar el acelerómetro. Para ello, conectamos el sensor a la placa Arduino siguiendo el circuito Fig.3.47, tal y como muestra el montaje de la Fig.3.48. Después de conectar la placa Arduino al PC, subimos el sketch del Prog.3.14. Como resultado en el monitor de serie obtenemos

la tabla de la Fig.3.49, que conduce a los valores más frecuentes: VoltajeX0 = 1475 mV, VoltajeY0 = 1577 mV , VoltajeZ0 = 1372 mV. Valiéndonos de un multímetro podemos comprobar que el resultado de estos voltajes es el correcto (Fig.3.50).

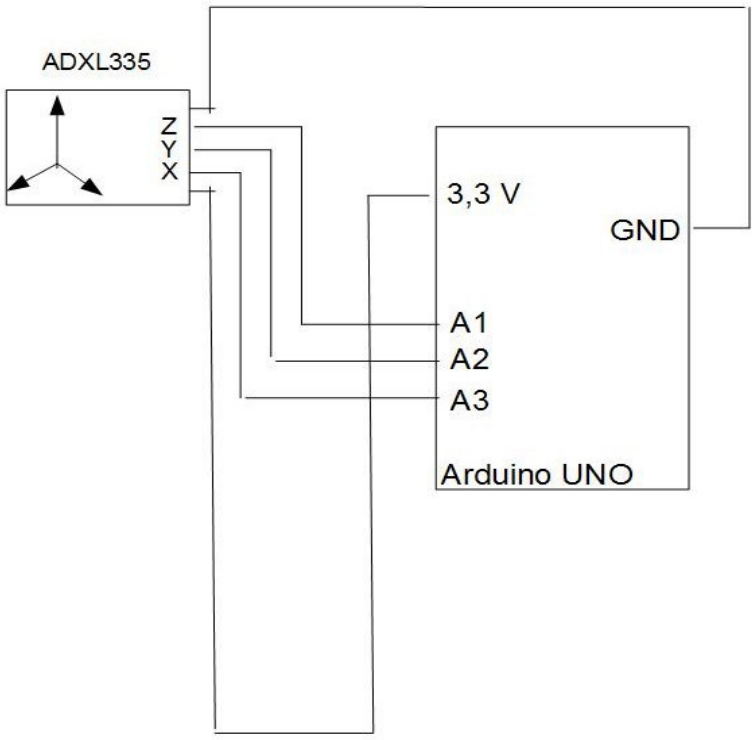


Fig.3.47. Circuito para la calibración del acelerómetro

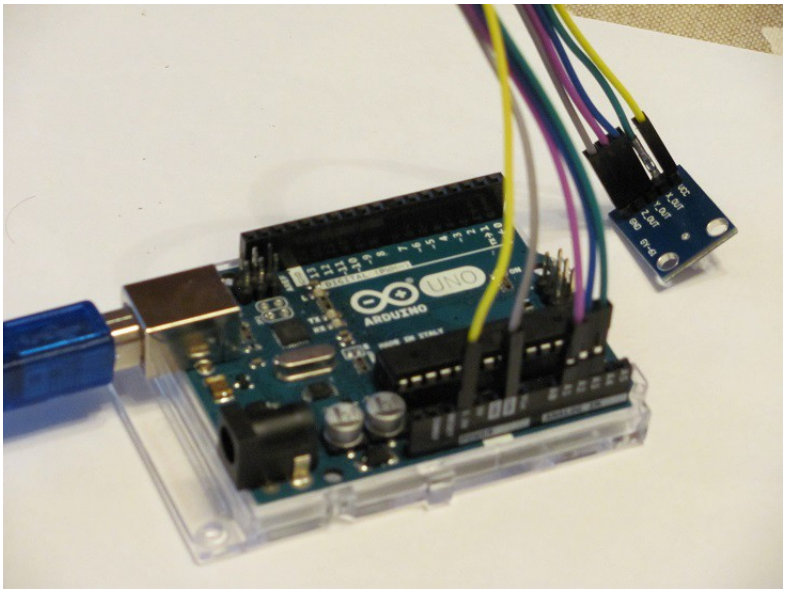


Fig.3.48. Montaje del circuito de la Fig.3.47

Prog.3.14. Sketch para calibrar el sensor de aceleración

```
/*
Proyecto 13a

Calibramos el acelerómetro
*/

// Usamos los pin analógicos A1, A2 y A3 para leer los voltajes
const int PinX = A3;
const int PinY = A2;
const int PinZ = A1;
int Medida = 1;

void setup()
{
  Serial.begin(9600);
  Serial.print("Medida      ");
  Serial.print("VoltajeX     ");
  Serial.print("VoltajeY     ");
  Serial.println("VoltajeZ     ");
  Serial.print("------      ");
  Serial.print("------      ");
  Serial.print("------      ");
  Serial.println("------     ");
}

void loop()
{
  float VoltajeX;
  float VoltajeY;
  float VoltajeZ;
  VoltajeX = LeerVoltaje(PinX);
  VoltajeY = LeerVoltaje(PinY);
  VoltajeZ = LeerVoltaje(PinZ);
  // Se mandan los datos de Aruino al monitor de serie

    Serial.print("  ");
    Serial.print(Medida);
    Serial.print("          ");
    Serial.print(VoltajeX);
    Serial.print("      ");
    Serial.print(VoltajeY);
    Serial.print("      ");
    Serial.println(VoltajeZ);
    Medida++;
    delay(1000);

}

float LeerVoltaje(int Pin)
{
  // Definimos la función LeerVoltaje()
  return (analogRead(Pin) * 1000. * 5. / 1024.);
}
```

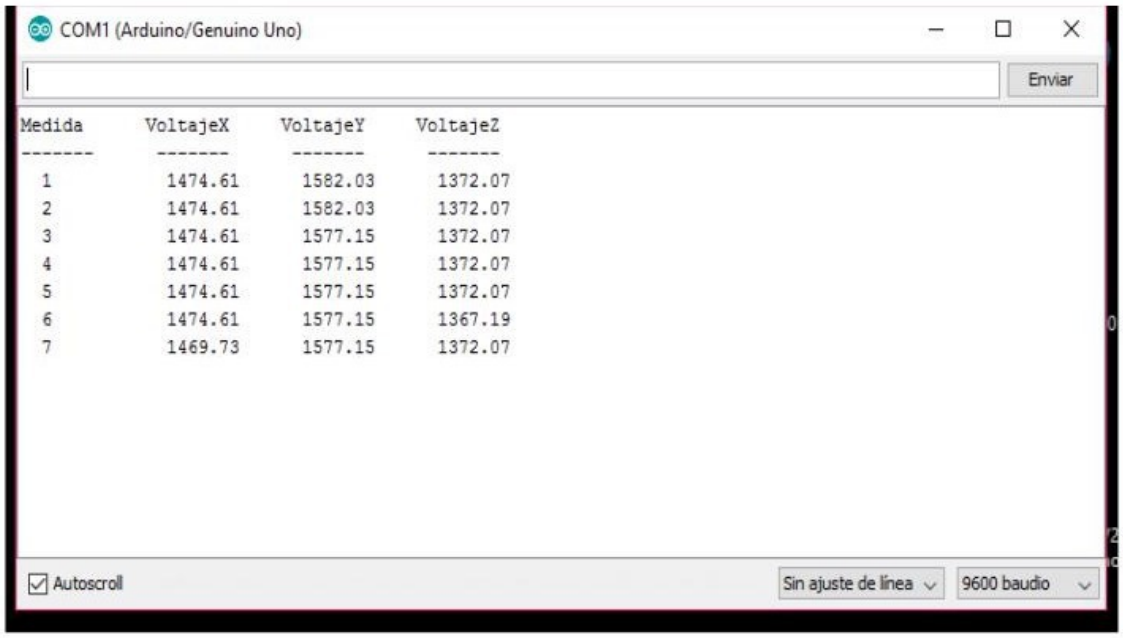


Fig.3.49. Valores del VoltajeX0, VoltajeY0 y VoltajeZ0 medidos en la calibración

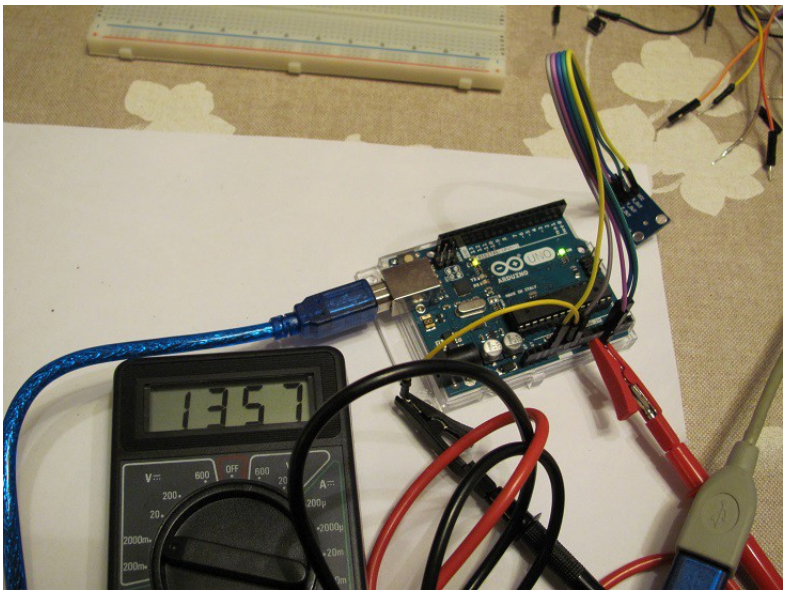


Fig.3.50. Comprobación de que VoltajeZ0 es correcto

3.8.2. Proyecto 13b: Medida de la aceleración

Para medir aceleraciones con el sensor sólo nos resta calcular la aceleración a partir del voltaje. Utilizando el mismo circuito y montaje de las Figs.3.47 y 3.48, podemos valernos de la función CalculaAcelera () en el sketch del Prog.3.15 para calcular la aceleración. En este caso, el monitor de serie mostrará la aceleración en m/s^2 (Fig.3.51). No hay que olvidar mover bruscamente el sensor para percibir aceleraciones.

Prog.3.15. Sketch para el cálculo de la aceleración

```
/*
Proyecto 13b

Medimos la aceleración
*/

// Usamos los pin analógicos A1, A2 y A3 para leer los voltajes
const int PinX = A3;
const int PinY = A2;
const int PinZ = A1;
// Valores del voltaje cuando el sensor esta en reposo
const float VoltajeX0 = 1474.;
const float VoltajeY0 = 1577.;
const float VoltajeZ0 = 1372.;
int Medida = 1;

void setup()
{
  Serial.begin(9600);
  Serial.print("Medida      ");
  Serial.print("AceleraX     ");
  Serial.print("AceleraY     ");
  Serial.println("AceleraZ     ");
  Serial.print("------      ");
  Serial.print("------      ");
  Serial.print("------      ");
  Serial.println("------     ");
}
```

```
void loop()
{
 float VoltajeX, VoltajeY, VoltajeZ;
 float VoltajeXf, VoltajeYf, VoltajeZf;
 float AceleraX, AceleraY, AceleraZ;
 VoltajeX = LeerVoltaje(PinX);
 VoltajeY = LeerVoltaje(PinY);
 VoltajeZ = LeerVoltaje(PinZ);
 VoltajeXf = VoltajeX - VoltajeX0;
 VoltajeYf = VoltajeY - VoltajeY0;
 VoltajeZf = VoltajeZ - VoltajeZ0;
 AceleraX = CalculaAcelera(VoltajeXf);
 AceleraY = CalculaAcelera(VoltajeYf);
 AceleraZ = CalculaAcelera(VoltajeZf);
 // Se mandan los datos de Aruino al monitor de serie

   Serial.print("  ");
   Serial.print(Medida);
   Serial.print("          ");
   Serial.print(AceleraX);
   Serial.print("       ");
   Serial.print(AceleraY);
   Serial.print("       ");
   Serial.println(AceleraZ);
   Medida++;
   delay(1000);

}

float LeerVoltaje(int Pin)
{
 // Definimos la función LeerVoltaje()
 return (analogRead(Pin) * 1000. * 5. / 1024.);
}

float CalculaAcelera(float V)
// Calcula la aceleración en m/s2 a partir del voltaje
{
 return (V / 330. * 9.8);
}
```

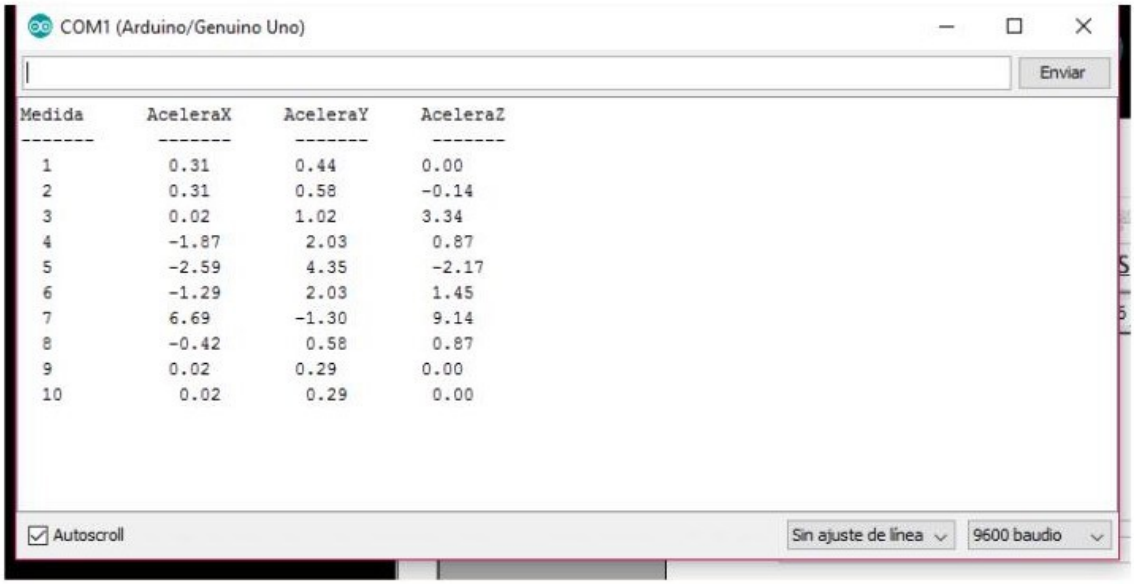


Fig.3.51. Aceleraciones en m/s^2 cuando el sensor se somete a un movimiento brusco de arriba a abajo

Capítulo 4

Zumbadores

En este capítulo mostraremos aplicaciones del buzzer o zumbador de tipo piezoeléctrico. Un zumbador es un aparato que emite señales audibles repetidamente. Puede ser de tipo electromecánico (cuyo componente activo es un electroimán) o piezoeléctrico. Entre los zumbadores piezoeléctricos podemos distinguir los activos y los pasivos. Los activos se caracterizan por poseer una fuente oscilante en su seno, la cual les permite seguir sonando siempre que se mantenga la corriente. Los pasivos, por el contrario, no contienen esta fuente oscilante, por lo que es necesario suministrar una sucesión de ondas cuadradas, como el proporcionado por los pines digitales PWM de la placa Arduino, para que continúe sonando.

4.1. PROYECTO 14: EXPERIMENTOS CON EL ZUMBADOR

A simple vista podemos distinguir los zumbadores activos de los pasivos fijándonos en sus reversos. Como muestra la Fig.4.1, la placa que rodea las patas de un zumbador pasivo es de color verde, mientras que la que rodea uno activo es de color negro. Ambos tienen grabado un signo "+" en el anverso, indicando la pata que es de polaridad positiva. Con ayuda de un multímetro podemos también identificar cual de los zumbadores es activo y cual es pasivo. Si la resistencia entre ambas patas es de unos 15 Ω (Fig.4.2), se trata de un zumbador pasivo. Si por el contrario, la resistencia es mayor que varios cientos de ohmios, se trata de un zumbador activo.

El uso de un zumbador en montajes electrónicos es idéntico al de un LED intermitente, en el que se enciende y apaga el componente sucesivas veces. Únicamente hay que tener en cuenta que la frecuencia de encendido y apagado es mucho mayor en un zumbador.

Fig.4.1. Zumbadores activo (en el centro) y pasivo (a la izquierda)

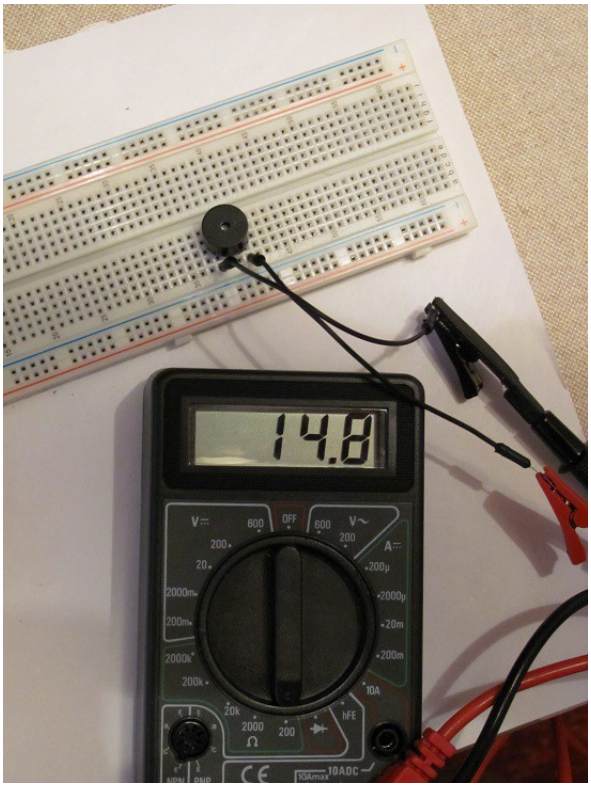


Fig.4.2. Resistencia de un zumbador pasivo

4.1.1. Proyecto 14a: Zumbido de un zumbador pasivo. Uso de un potenciómetro para cambiar la frecuencia

Las Figs. 4.3 y 4.4 muestran el circuito y el montaje de un zumbador al pin digital 11 de la placa Arduino. Todo ello es idéntico al Proyecto 1 de intermitencia de un LED, salvo que se sustituye el LED por el zumbador. Vemos también que el sketch (Prog.4.1) es muy similar al del Proyecto 1, únicamente las funciones delay() entre activación y desactivación del zumbador son mucho más rápidas, de 1 ms. Para mayor comodidad hemos definido la función zumbar(), la cual permite cambiar los delay(), o lo que es lo mismo la frecuencia de vibración del zumbador. Podemos para ello emplear un potenciómetro como muestran las Figs.4.5 y 4.6 y el sketch 4.2. En este caso, utilizamos, para modificar la frecuencia de la función zumbar(), la escala de 0 a 1023 del potenciómetro.

4.1.2. Proyecto 14b: Intento de cambiar el volumen de un zumbador pasivo con PWM. Generación de pitidos de una duración determinada. Notas musicales.

El circuito y el montaje son idénticos a los de la Figs.4.3 y 4.4. En este caso, utilizaremos modulación de impulsos en anchura en el sketch (Prog.4.3) para tratar de cambiar paulatinamente el volumen del zumbador. Como vemos es análogo al Proyecto 2a, en el que se empleaba esta misma técnica para disminuir progresivamente la intensidad luminosa de un LED. No obstante, podemos comprobar que el experimento no da el resultado esperado. El volumen es el mismo, salvo al final,

cuando la variable `a` del bucle se hace cero. Es decir, para `a = 0`, en `analog(PinZumba, a)`, el zumbador deja de sonar. Este hecho nos permite crear pitidos `bip()` de una determinada duración y simular, por ejemplo, un despertador, como muestra Prog.4.4.

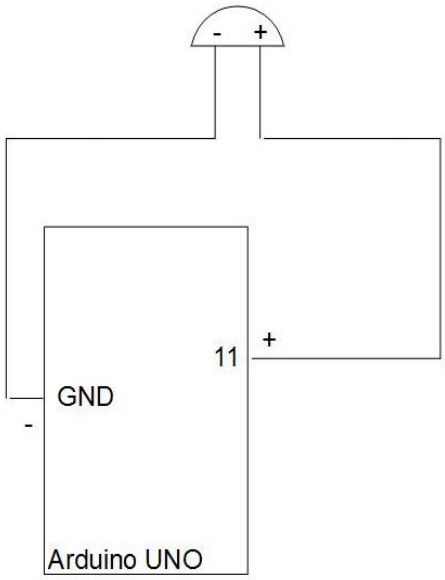


Fig.4.3. Circuito que une un zumbador a la placa Arduino

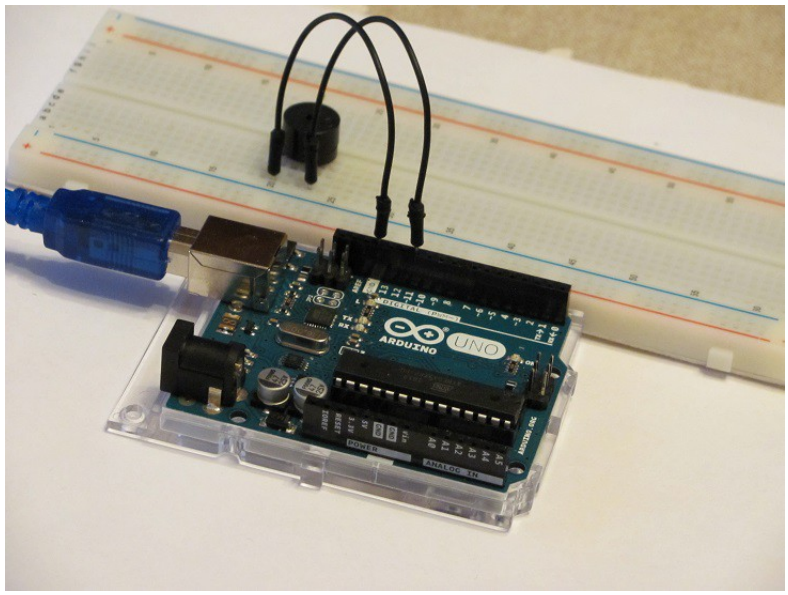


Fig.4.4. Montaje que une un zumbador a la placa Arduino

Prog.4.1. Sketch que hace vibrar un zumbador pasivo

```
/*
Proyecto 14a_1

Zumbido del zumbador
*/

const int PinZumba =11;
int Frecuencia = 1;

void setup()
{
  pinMode(PinZumba, OUTPUT);
}

void loop()
{
  zumbar(Frecuencia);
}

void zumbar(int pausa)
{
  digitalWrite(PinZumba, HIGH);       // Activa el zumbador
  delay(pausa);                       // Espera pausa ms encendido
  digitalWrite(PinZumba, LOW);        // Desactiva el zumbador
  delay(pausa);                       // Espera pausa ms apagado
}
```

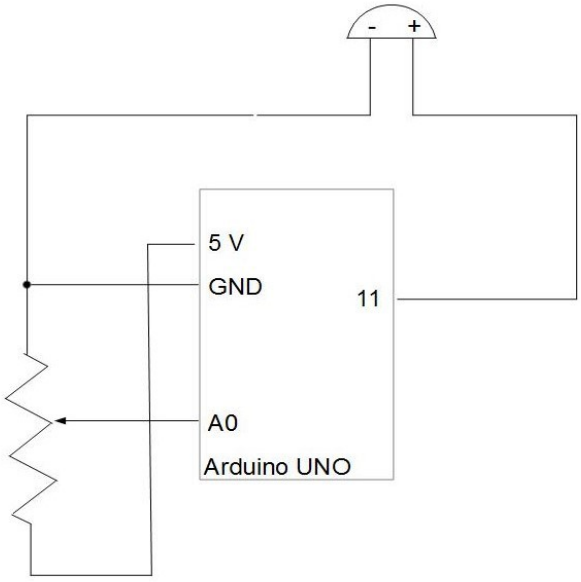


Fig.4.5. Circuito que cambia la frecuencia de un zumbador con un potenciómetro

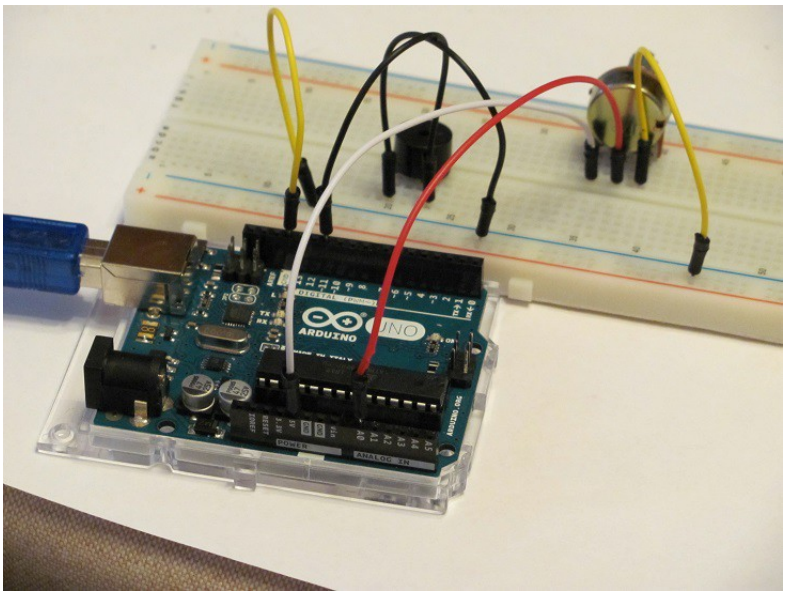


Fig.4.6. Montaje para cambiar la frecuencia de un zumbador con un potenciómetro

Prog.4.2. Sketch que cambia la frecuencia de vibración de un zumbador con ayuda de un potenciómetro

```
/*
Proyecto 14a_2

Zumbador que va cambiando su frecuencia
 con ayuda de un potenciómetro
*/

const int PinAnalogico = A0;
const int Pin = 11;
int ValorEntrada = 0;
int Frecuencia = 0;


void setup()
{
  pinMode(Pin, OUTPUT);
}

void loop()
{
 // Pierde y gana frecuencia según el potenciómetro
 // lee el valor del potenciómetro
  ValorEntrada = analogRead(PinAnalogico);
  Frecuencia = ValorEntrada;
  zumbar(Frecuencia);
}

void zumbar(int pausa)
{
  digitalWrite(Pin, HIGH);      // Activa el zumbador
  delay(pausa);                  // Espera pausa ms encendido
  digitalWrite(Pin, LOW);       // Desactiva el zumbador
  delay(pausa);                 // Espera pausa ms apagado
}
```

Prog.4.3. Intento de cambiar el volumen del zumbador con PWM

```
/*
Proyecto 14b_1

Intento de cambiar el volumen del zumbador mediante PWM
*/

const int PinZumba = 11;

void setup ()
{
  pinMode(PinZumba, OUTPUT);
}

void loop()
{
  for (int a=0; a<=255;a++)
  {
    analogWrite(PinZumba, a);
    delay(20);
  }
    for (int a=255; a>=0;a--)
  {
    analogWrite(PinZumba, a);
    delay(20);
  }
}
```

Prog.4.4. Simulando los pitidos de un despertador

```
/*
Proyecto 14b_2

Creando pitidos como en un despertador
*/

const int PinZumba = 11;

void setup ()
{
  pinMode(PinZumba, OUTPUT);
}

void loop()
{
  bip(200);       // pita 200 ms
  nbip(200);      // no pita 200 ms
}

void bip(int duracion)
{
  analogWrite(PinZumba, 10);
  delay(duracion);
}

void nbip(int duracion)
{
  analogWrite(PinZumba, 0);
  delay(duracion);
}
```

Para articular notas musicales no nos basta con la función bip(). Además de la duración debemos ser capaces de modificar la frecuencia. La generación de notas musicales se realiza con la función tone(Pin, frecuencia, duracion), que incorpora Arduino en su compilador. El Sketch Prog.4.5 muestra como se puede repetir una y otra vez la escala musical con un zumbador pasivo. La variable de cadena Notas[] indica la frecuencia de las notas, mientras que la variable Duracion[] denota su duración. Asignamos el valor 1000/4 si son negras, 1000/8 si son corcheas, 1000/16 si son semicorcheas, etc. Después de cada nota se produce un silencio que es un 30% de la duración de la nota. Esto nos permite separar unas notas de otras.

Prog.4.5. Creando notas musicales con el zumbador

```
/*
Proyecto 14b_3

Creando la escala de notas musicales
*/

const int PinZumba = 11;
int NumeroNotas = 8;
int Notas[] = {261, 277, 294, 311, 330, 349, 370, 392};
int Duracion[] = {4, 4, 4, 4, 4, 4, 4, 4};

void setup ()
{
  pinMode(PinZumba, OUTPUT);
}

void loop()
{
  int DuracionNota;
  int SilencioEntreNotas;
  for (int a = 0; a < NumeroNotas; a++)
  {
    // Establece la duración de cada nota
    // Negra 1000/4, corchea 1000/8
    DuracionNota = 1000 / Duracion [a];
    tone(PinZumba, Notas[a], DuracionNota);
    // Para distinguir las notas las separa por un silencio
    // que es el 30% de la duración de la nota
    SilencioEntreNotas = DuracionNota * 1.30;
    delay(SilencioEntreNotas);
  }
}
```

Capítulo 5

Motores

El *Sunfounder Super Kit V2.0 for Arduino* y el *Kuman Super Starter Kit* contienen dos tipos de motores: uno típico de corriente continua y otro lento paso a paso (Fig.5.1). En el de corriente continua se puede regular su velocidad o número de rotaciones completas por minuto (RPM) y su sentido de giro. El motor paso a paso considera una sucesión de movimientos angulares discretos de magnitud uniforme que se pueden programar. En este capítulo mostraremos experimentos con cada uno de ellos.

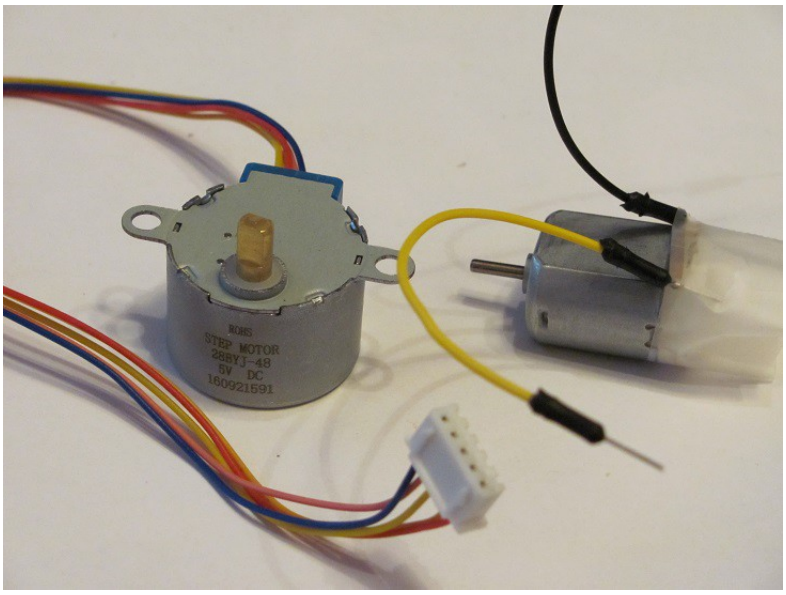


Fig.5.1. Motor de corriente continua (a la derecha) y de paso a paso (a la izquierda)

5.1. PROYECTO 15: VARIAR LA VELOCIDAD DE UN MOTOR DE CORRIENTE CONTINUA

El motor de corriente continua del *Sunfounder Super Kit V2.0 for Arduino* tiene dos pines a los que hay que soldar sendos cables de conexión. Sin embargo, para realizar pruebas, basta con insertar dos cables y añadir cinta aislante o celofán para evitar que escapen (Fig.5.1).

Un motor de corriente continua se compone de un imán fijo denominado imán estátor y de una bobina móvil, que gira en torno a un eje, denominada rotor. Cuando el campo magnético generado por la bobina del rotor se opone al del imán estátor, se crea un par de fuerzas que hace

girar el rotor. Para cambiar el sentido de rotación del rotor, basta con invertir la polaridad de la fuente de alimentación.

5.1.1. Proyecto 15a: Variando la velocidad del motor de corriente continua

A la hora de variar la velocidad del motor, nuestro primer pensamiento sería conectar uno de los polos del motor a un pin digital con PWM y el otro polo a tierra. De esta forma, como ya hemos realizado en otras ocasiones, con la función analog() podemos ir proporcionando voltajes mayores o menores hasta obtener la velocidad deseada. El problema de proceder de esta forma es que el motor consume mucha más energía que un LED o un zumbador, y la potencia suministrada por el pin digital es insuficiente. Los pines digitales de Arduino proporcionan un máximo de 40 mA, lo que multiplicado por los 5 V de tensión, genera una potencia de 0,2 W, suficiente para encender un LED o activar un zumbador, pero insuficiente para mover un motor. ¿Por qué no entonces conectar un polo del motor a la fuente de 5 V de Arduino y la otra a tierra? De esta forma obtendríamos la potencia suficiente, pero no conseguiríamos modificar la velocidad del motor. La solución al problema que se nos plantea está en combinar ambas formas de proceder, utilizando para ello un transistor.

El transistor es un dispositivo electrónico capaz de suministrar una señal de salida en respuesta a una señal de entrada. Es decir, como respuesta de una señal pequeña proporcionada por el pin digital de Arduino, obtenemos una señal de intensidad de corriente suficiente como para mover el motor. El esquema del circuito sería como muestra la Fig.5.2.

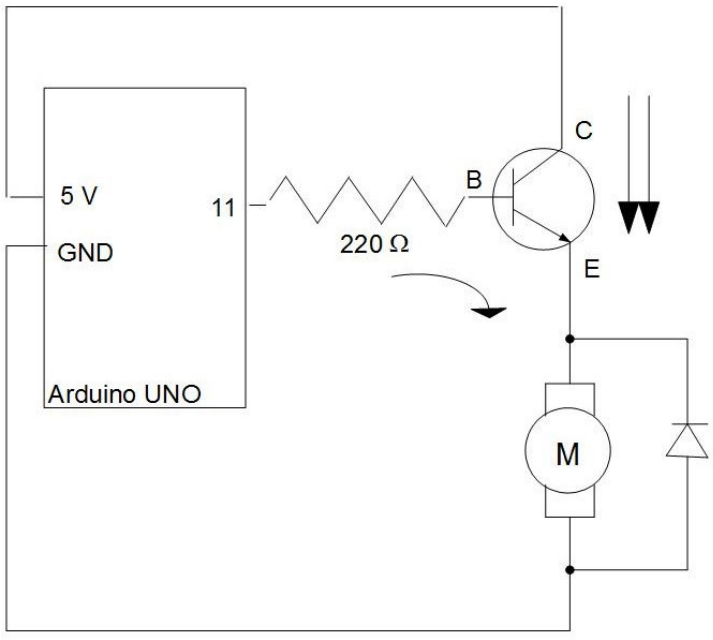


Fig.5.2. Circuito que ilustra como se utiliza un transistor para regular la corriente suministrada al motor

El transistor mostrado en el circuito de la Fig.5.2 es de unión bipolar npn y cuenta con tres patas y una cabeza de plástico. Mirando al transistor con la parte plana de la cabeza hacia arriba

(Fig.5.3), la pata central se denomina base (B), la de la izquierda emisor (E) y la de la derecha colector (C). El transistor lleva serigrafiado en la parte plana de la cabeza el código s8050, lo que viene a indicar que se trata de una unión bipolar npn. Las flechas en la Fig.5.2 indican en que dirección se mueve la corriente. La corriente en el emisor es la suma de la corriente en el colector mas la corriente en la base.

El diodo rectificador de la Fig.5.4 tiene como misión evitar que las corrientes en sentido inverso, originadas al apagar el motor, puedan dañar el transistor o incluso la placa Arduino. El *Sunfounder Super Kit V2.0 for Arduino* contiene cuatro diodos rectificadores 1N4007. La banda gris en uno de sus extremos denota el polo negativo.

Tras realizar el montaje de la Fig.5.5 y programar el sketch Prog.5.1 vemos como el motor va acelerando hasta alcanzar una velocidad máxima, para luego frenarse lentamente y volver a la posición de reposo. A continuación, vuelve a repetir el movimiento de aceleración positiva y negativa sucesivas veces.

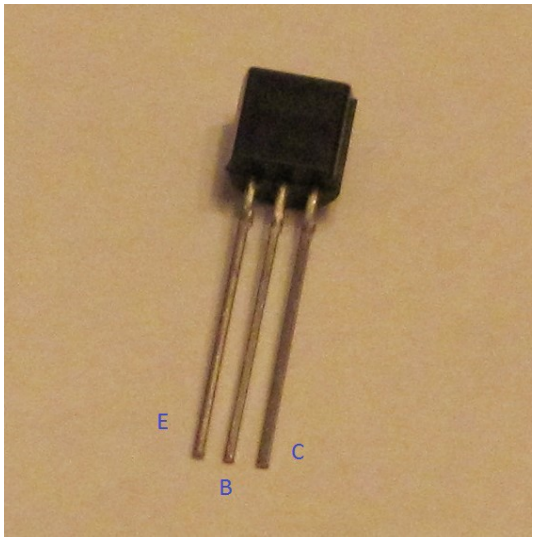


Fig.5.3. Transistor s8050 incluido en el Sunfounder Super Kit V2.0 for Arduino

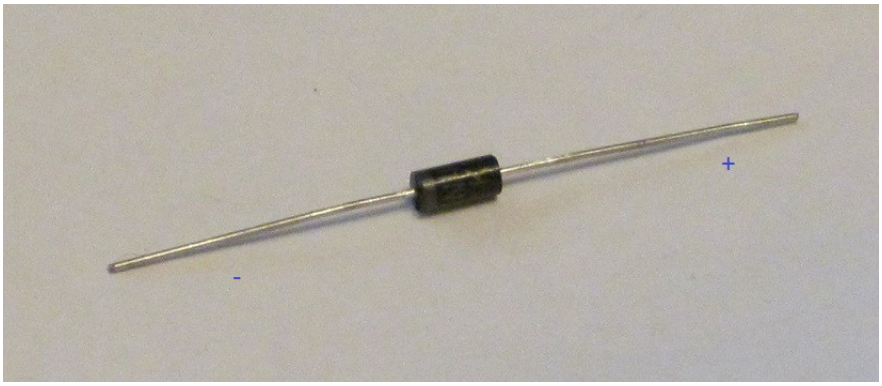


Fig.5.4. Diodo rectificador IN4007 incluido en el Sunfounder Super Kit V2.0 for Arduino

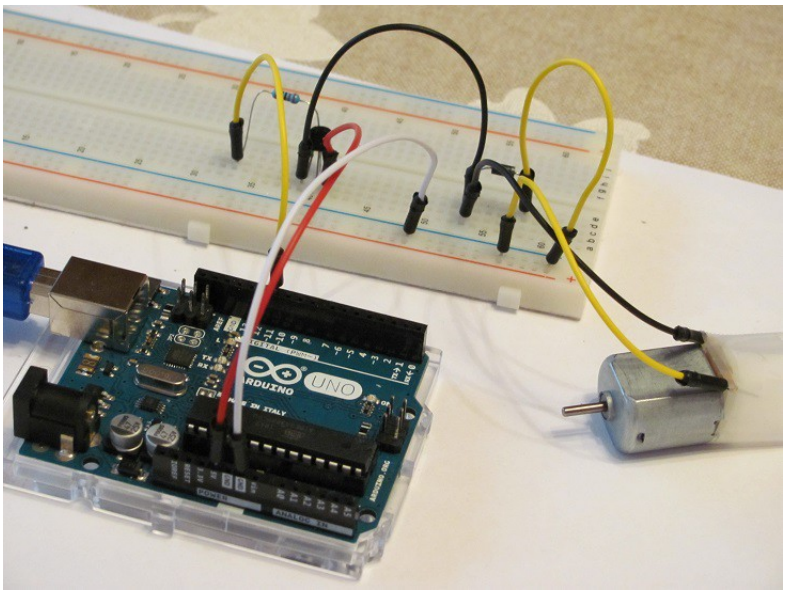


Fig.5.5. Montaje para conseguir un movimiento de aceleración positiva y negativa del motor de corriente continua

Prog.5.1. Sketch para conseguir un movimiento acelerado y desacelerado del motor de corriente continua

```
/*
Proyecto 15a

Cambia la velocidad de un motor de corriente continua
*/

const int PinControl =11;

void setup()
{
  pinMode(PinControl, OUTPUT);
}

void loop()
{
  // Movimiento acelerado de reposo a una velocidad máxima
  for (int a = 0; a < 255; a++)
  {
    analogWrite(PinControl, a);
    delay(20);
  }
 // Movimiento de aceleración negativa de velocidad máxima a cero
  for (int b = 254; b >= 0; b--)
  {
    analogWrite(PinControl, b);
    delay(20);
  }
}
```

5.1.2. Proyecto 15a: Variando la velocidad del motor con ayuda de un potenciómetro

Podemos añadir un potenciómetro que regule la velocidad del motor. Para ello proponemos el circuito de la Fig.5.6, el montaje en la placa de pruebas de la Fig.5.7 y el sketch de Prog.5.2.

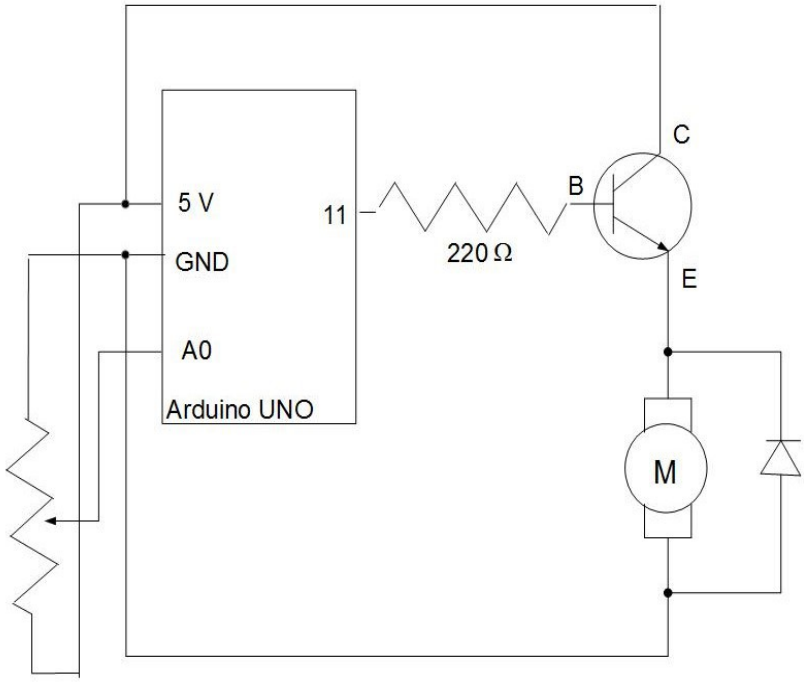


Fig.5.6. Circuito con potenciómetro que regula la velocidad del motor de corriente continua

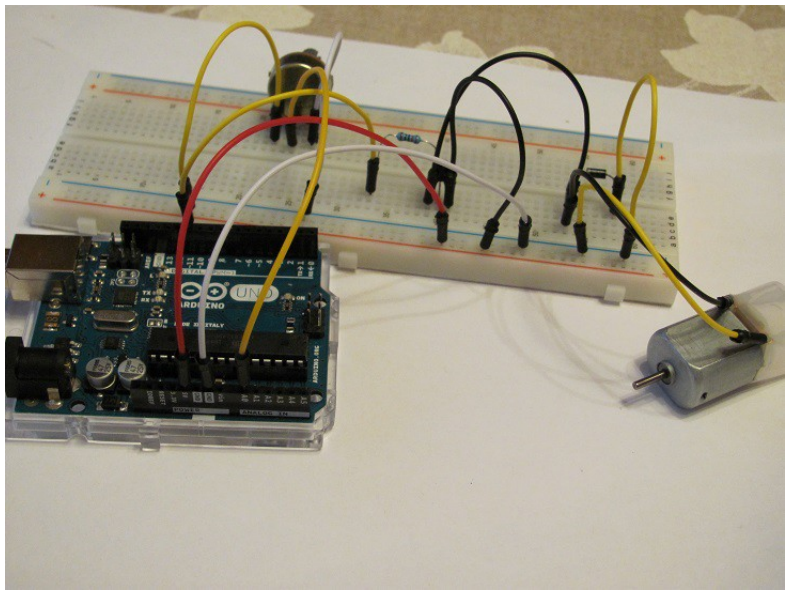


Fig.5.7. Montaje del Proyecto 15b

Prog.5.2. Sketch del Proyecto 15b

```
/*
Proyecto 15b

Cambia la velocidad de un motor con ayuda de un potenciómetro
*/

const int PinAnalogico = A0;
const int PinControl = 11;
int ValorEntrada = 0;
int ValorSalida = 0;

void setup()
{
  pinMode(PinControl, OUTPUT);
}

void loop()
{
  ValorEntrada = analogRead(PinAnalogico);
  ValorSalida = map(ValorEntrada,0,1023,0,255);
  analogWrite(PinControl,ValorSalida);
}
```

5.2. PROYECTO 16: CAMBIANDO EL SENTIDO Y LA VELOCIDAD DE UN MOTOR DE CORRIENTE CONTINUA CON L293D

Ya hemos visto que para modificar el sentido de rotación del motor basta con cambiar la polaridad del motor. Sin embargo, surge la pregunta de como hacerlo sin conectar y desconectar el circuito. Para ello es necesario montar un puente H. Se trata de un circuito electrónico que permite rotar al motor en ambos sentidos. No vamos a entrar en particularidades sobre este tipo de circuitos, sólo diremos que el chip L293D de Texas Instruments está preparado para cumplir con esta misión, entre otras varias. La Fig.5.8 muestra el anverso de este chip, en el que se han numerado las patillas de 1 a 16. La función de cada una de ellas se especifica en la Tabla 5.1. El chip L293D está preparado para aceptar la conexión a dos motores. En este caso, vamos a manejar un único motor de corriente continua. Por tanto, las patillas 2 y 7 se conectarán a sendos pines digitales de la placa Arduino; mientras que, las patillas 3 y 6 se conectarán a ambos polos del motor.

Las Figs.5.9 y 5.10 muestran el circuito y montaje realizados para modificar el sentido de rotación y la velocidad en un motor de corriente continua. En el sketch de Prog.5.3 aceleraremos y desaceleraremos el motor, como se llevó a cabo en el Proyecto 15a, pero esta vez, haciéndolo rotar en el sentido de las agujas del reloj y en sentido contrario, alternativamente.

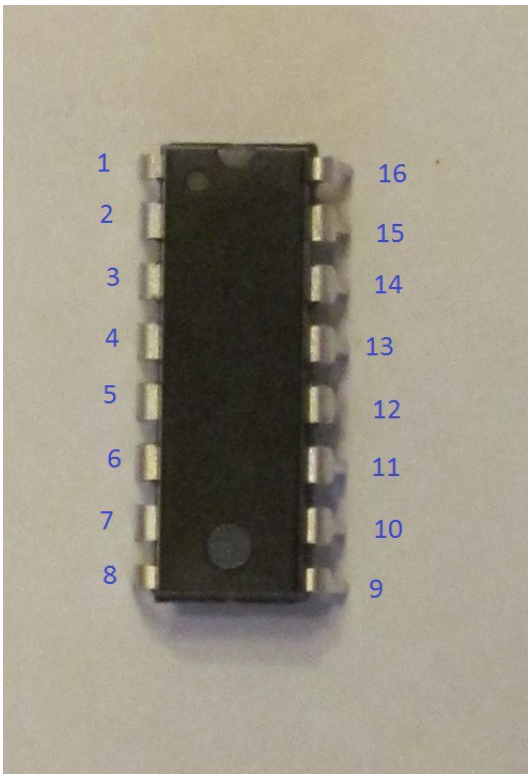


Fig.5.8. Chip L293D con cada una de sus patillas numeradas

Tabla 3.1. Función de cada una de las patillas según su numeración

Números	Código	Función
1	1,2EN	Activar los canales 1 y 2 del controlador (motor 1)
2,7	<1:2>A	Entradas del controlador (motor 1)
3,6	<1:2>Y	Salidas del controlador (motor 1)
10,15	<3:4>A	Entradas del controlador (motor 2)
11 14	<3:4>Y	Salidas del controlador (motor 2)
9	3,4EN	Activar los canales 3 y 4 del controlador (motor 2)
4,5	GND	Conexión a tierra (motor 1)
12,13	GND	Conexión a tierra (motor 2)
16	Vcc1	Alimentación de 5V para el propio chip
8	Vcc2	Alimentación del motor

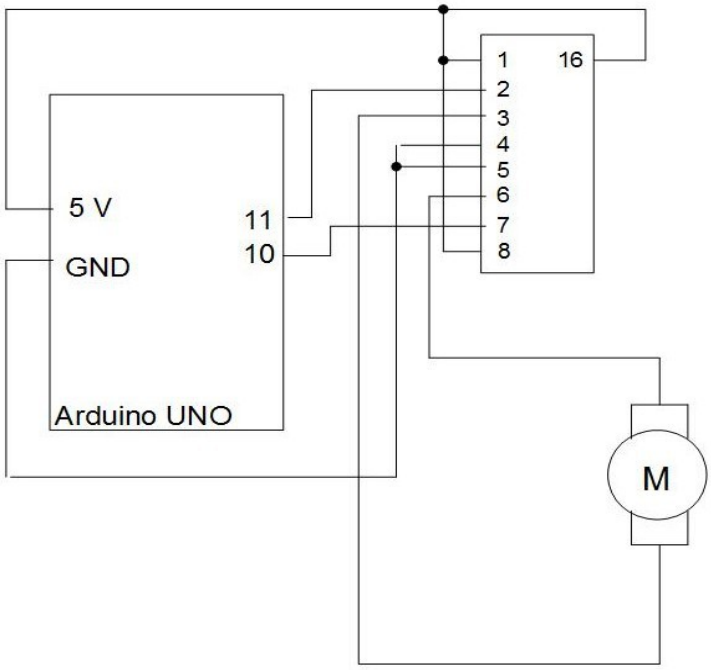


Fig.5.9. Circuito del Proyecto 16

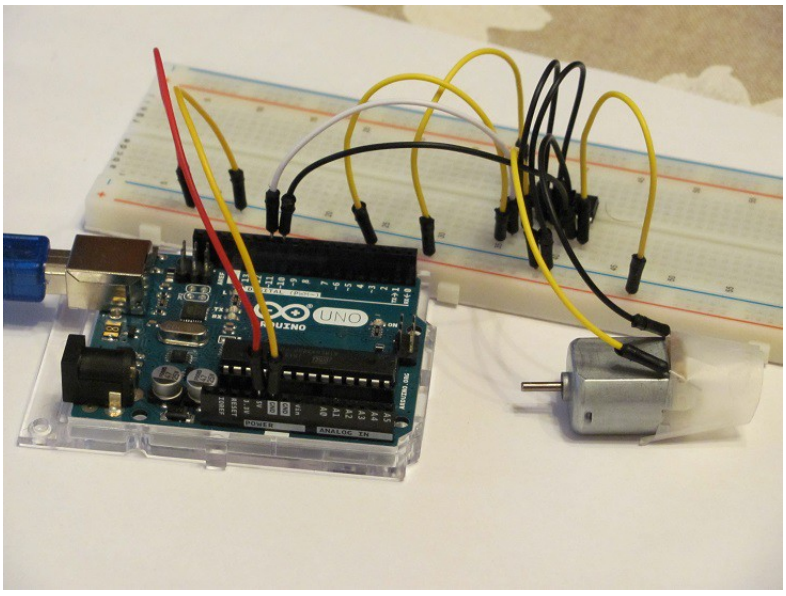


Fig.5.10. Circuito del Proyecto 16

Prog.5.3. Sketch del Proyecto 16

```
/*
Proyecto 16

Cambia la velocidad y el sentido de un motor de corriente continua
*/

const int PinControl1 = 11;
const int PinControl2 = 10;

void setup()
{
  pinMode(PinControl1, OUTPUT);
  pinMode(PinControl2, OUTPUT);
}

void loop()
{
  // Movimiento acelerado de reposo a una velocidad máxima
  // a favor de las agujas del reloj
  int a, b;
  for (a = 0; a < 255; a++)
  {
    favorReloj(a);
    delay(20);
  }
  // Movimiento de aceleración negativa de velocidad máxima a cero
  // a favor de las agujas del reloj
  for (b = 254; b >= 0; b--)
  {
    favorReloj(b);
    delay(20);
  }
  // Movimiento acelerado de reposo a una velocidad máxima
  // en contra de las agujas del reloj
  for (a = 0; a < 255; a++)
  {
    antiReloj(a);
    delay(20);
  }
  // Movimiento de aceleración negativa de velocidad máxima a cero
  // en contra de las agujas del reloj
  for (b = 254; b >= 0; b--)
  {
    antiReloj(b);
    delay(20);
  }
}
```

```
// Función que hace que gire el motor en sentido de las agujas del reloj
void favorReloj(int Velocidad)
{
  analogWrite(PinControl1,Velocidad);
  analogWrite(PinControl2,0);
}

// Función que hace que gire el motor en sentido contrario a las agujas del reloj
void antiReloj(int Velocidad)
{
  analogWrite(PinControl1,0);
  analogWrite(PinControl2,Velocidad);
}
```

5.3. PROYECTO 17: CONTROL DE VELOCIDAD EN UN MOTOR PASO A PASO 28BYJ-48

Como ya mencionamos anteriormente la principal característica de los motores paso a paso es su movimiento angular discreto en pasos uniformes. Exteriormente vemos que un motor paso a paso contiene 4, 5 o 6 cables de conexión, dependiendo del motor de que se trate. En nuestro caso, el motor 28BYJ-48 posee 5 cables de conexión de distinto color. Como puede verse en la Fig.5.11, de los 5 cables, cuatro pertenecen a los terminales (Naranja, Rosa, Amarillo y Azul) de 2 bobinados diferentes (en realidad 2 pares de bobinados conectados). El cable restante es de un terminal central común (Rojo). Este tipo de motor se denomina unipolar de 5 cables.

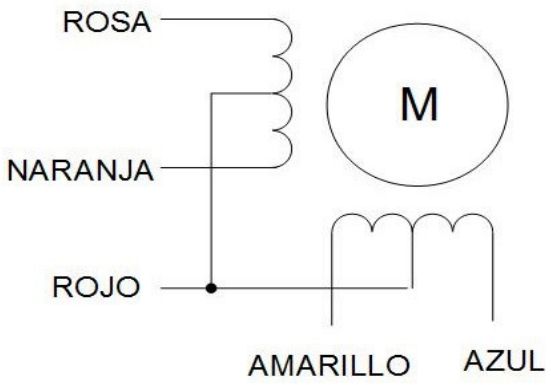


Fig.5.11. Esquema de los cinco terminales de color en el motor 28BYJ-48

La pregunta que surge ahora es cómo conectar esos cinco cables a la placa Arduino. Para conseguirlo contamos con la placa SBT0811 (Fig.5.12), la cual hace las veces de interfaz, puesto que convierte los 5 terminales del motor en 4 pines de salida, que se pueden conectar directamente a la placa Arduino. Gracias a esta placa, que integra el chip ULN2003, podemos modificar la velocidad de rotación del motor, contar el número de pasos e incluso modificar el sentido de rotación.

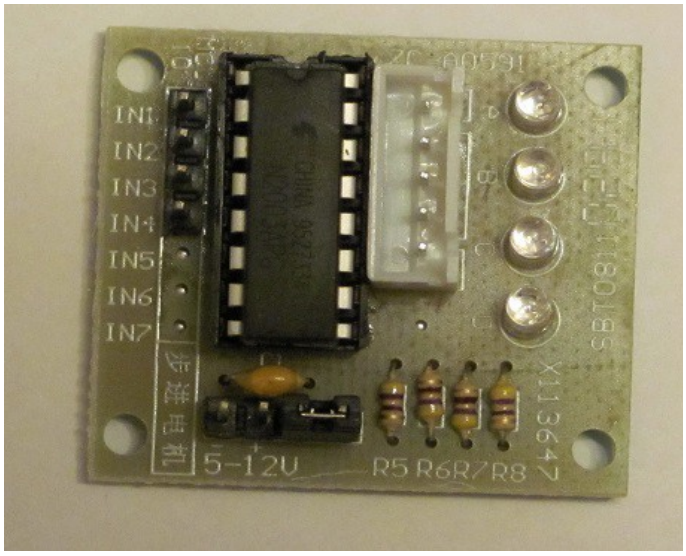


Fig.5.12. Placa SBT0811

La placa SBT0811 cuenta con dos pines (junto a los caracteres chinos) que permiten alimentar el motor directamente de la placa Arduino. El polo positivo (+) se conecta al pin de 5V y el negativo (-) a tierra (GND). A la hora de conectar los cinco cables del motor a la placa SBT0811, conviene observar que las conexiones del adaptador de plástico blanco no se encuentran en el centro, sino que están ligeramente a un lado. Esto evita cualquier confusión cuando se introduce en su alojamiento de plástico en la placa SBT0811. Las entradas IN1, IN2, IN3 e IN4 pueden conectarse a los pines digitales 8, 9, 10 y 11 de la placa Arduino como muestra el circuito de la Fig.5.13. Una vez realizado el montaje, éste debe quedar como muestra la Fig.5.14.

Como sketch (Prog.5.4) se propone que el motor paso a paso avance 10 pasos a favor de las agujas del reloj y realice 5 pasos en contra de las agujas del reloj, para luego repetir los pasos una y otra vez. Es importante tener en cuenta que el motor tiene una reductora de 1/64, por lo que puede realizar hasta 64 × 8 = 512 medios pasos por vuelta. En el sketch utilizamos el valor de 64 medios pasos por vuelta. En el monitor de serie aparece el sentido de giro y el número de pasos realizados.

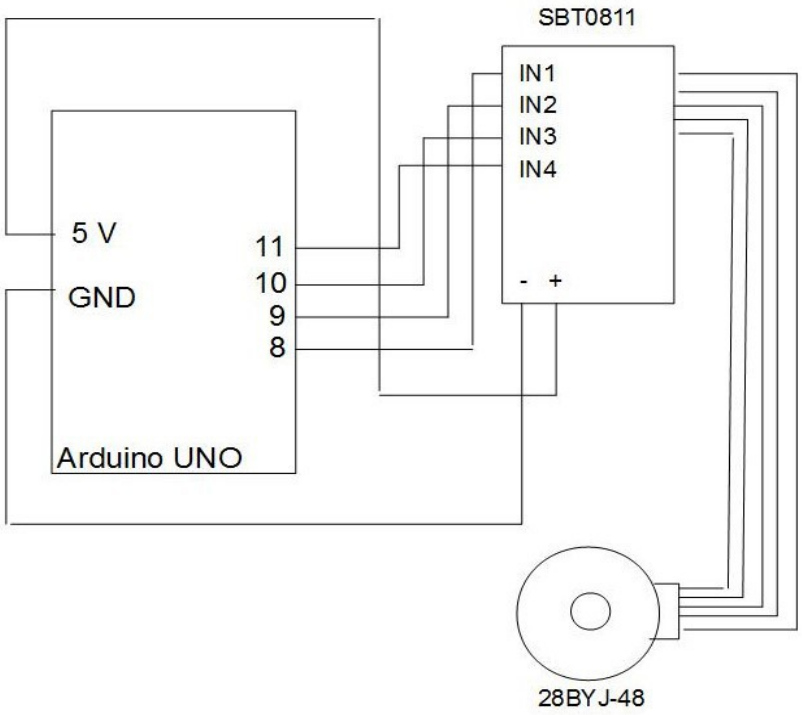


Fig.5.13. Circuito del Proyecto 17

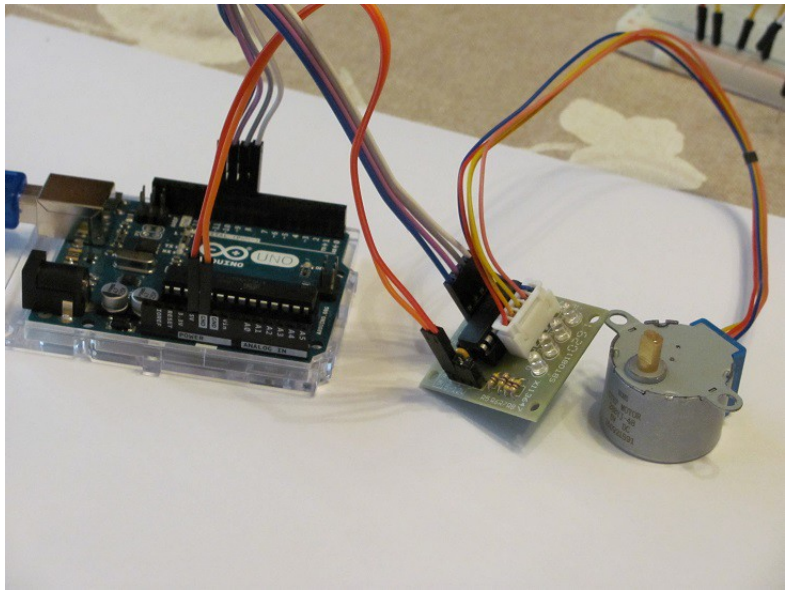


Fig.5.14. Montaje del Proyecto 17

Prog.5.4. Sketch del Proyecto 17

```
/*
Proyecto 17

Motor paso a paso que da 10 pasos en un sentido y 5 en el contrario
*/

#include <Stepper.h>

// Hacemos que de 64 medios pasos por vuelta
const int PasosPorVuelta = 32;
// Contaremos cada uno de los pasos
int Paso = 0;
// Comienza la conexión entre el motor paso a paso y los pines 8, 9, 10 y 11
Stepper miMotor(PasosPorVuelta, 8, 9, 10, 11);

void setup()
{
 // Fijamos una velocidad de 100 RPM
 miMotor.setSpeed(100);
 // Abrimos el monitor de serie
 Serial.begin(9600);
}

void loop()
{
 int a;
 // El motor realiza 10 pasos a favor de las agujas del reloj
 for (a = 1; a <= 10; a++)
 {
   miMotor.step(PasosPorVuelta);
   delay(500);
   Paso++;
   Serial.print("a favor     ");
   Serial.println(Paso);
 }

 // El motor realiza 5 pasos en contra de las agujas del reloj
 for (a = 1; a <= 5; a++)
 {
   miMotor.step(-PasosPorVuelta);
   delay(500);
   Paso++;
   Serial.print("contrario   ");
   Serial.println(Paso);
 }
}
```

CONCEPTOS BÁSICOS DE ARDUINO

Agustín Grau Carles

Este manual desarrolla completamente 33 experimentos con la placa Arduino, utilizando componentes electrónicos del *Sunfounder Super Kit V2.0 for Arduino* y del *Kuman Super Starter Kit.* El primer capítulo muestra brevemente todos aquellos aspectos imprescindibles antes de comenzar a trabajar con Arduino. El segundo capítulo esta dedicado a los LED convencionales y RBG, intercalando experimentos en los que se utiliza el potenciómetro y los botones. El tercer capítulo considera la calibración y estudio de sensores térmicos, de intensidad lumínica, de infrarrojos, de contacto, de llama, de aceleración y de proximidad. El cuarto capítulo se centra en el estudio de los zumbadores. Por último, el quinto capítulo contempla experimentos con motores de corriente continua y de paso a paso. Todos los experimentos contienen una fotografía del montaje, un esquema del circuito y el sketch correspondiente en C.

e-GO! Library Español

COMO LO VIO EN TV

¡1250 LIBROS PARA LLEVAR EN SU BOLSILLO!

La velocidad, comodidad y movilidad son suyas. El e-GO! Library Español es una forma innovadora para tener y mantener un suministro fresco y abundante de grandes títulos. Es el mejor entretenimiento y fácil de obtener. El e-GO! Library Español es una unidad flash de memoria USB que pone a miles de los mejores libros de la actualidad su bolsillo!

Cargue su Kindle, iPad, Nook, o cualquier dispositivo con una variedad de ficción y no ficción. En su tiempo libre, elija entre sus temas, títulos y autores independientes favoritos y categorías como: romance, ciencia ficción, misterios, finanzas, biografías, negocios y muchos más.

- ✓ **1,000 LIBROS** indipendientes más populares
- ✓ **BONO**- 250 títulos clásicos
- ✓ **CONTENIDO ÚNICO** / Autores independientes
- ✓ **LLAVE USB PRECARGADA** de 4GB
- ✓ **SIRVE CON TODOS** los lectores y dispositivos
- ✓ **IDEAL** para viajar
- ✓ **AHORRA** innumberables horas de Descargas
- ✓ **EL REGALO** Perfecto

VER MÁS