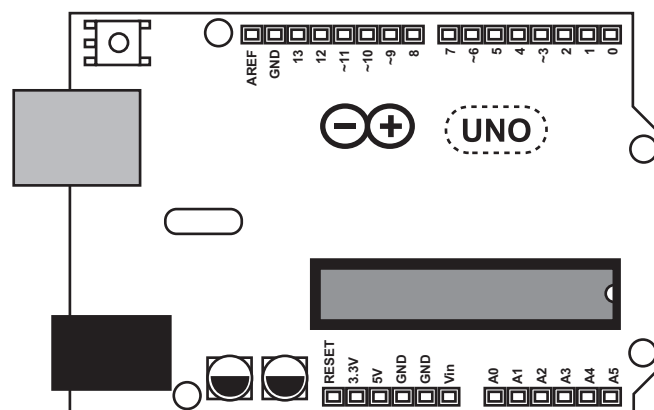


# PRÁCTICAS ARDUINO

## MICROLOG

### LOG 5041



MICRO-LOG TECNOLOGÍA Y SISTEMAS, S.L.

C/ Andrés Obispo, 37, 5º - 28043 Madrid  
Telf. 91 759 59 10  
pedidos@microlog.es  
www.microlog.es



## ÍNDICE

### A. Introducción

#### 1. Identificación de los materiales

### B. Instalación programas

#### 1. Instalar Arduino IDE en nuestro PC

##### 1.1. Instalación en Linux

##### 1.2. Instalación en Windows

#### 2. Instalar S4A en nuestro PC

#### 3. Instalar mBlock

##### 3.1. Instalación en Windows

### C. Teoría Básica

#### 1. Instrucciones básicas (S4A, Makeblock y Arduino IDE)

##### 1.1. Iniciación

##### 1.2. Bucles

##### 1.3. Espera un segundo

##### 1.4. Condicionales

##### 1.5. Instrucciones de operación

##### 1.6. Instrucciones de movimiento

##### 1.7. Control de servomotor

##### 1.8. Variables

#### 2. Entorno gráfico de programación

##### 2.1. S4A

##### 2.2. mBlock

##### 2.3. Arduino IDE

#### 3. Programar encendido y apagado de un diodo LED verde paso a paso en mBlock

##### 3.1 Montaje de componentes electrónicos

##### 3.2 Conectar la placa Arduino UNO

##### 3.3 Programar encendido y apagado de un diodo LED verde

## ÍNDICE

### D. Prácticas

1. Luz intermitente
2. Aviso para invidentes
3. Semáforo
4. Semáforo con aviso para invidentes
5. Semáforo con pulsador
6. Cruce de semáforos
7. Encendido nocturno (LDR)
8. Cadena de montaje
9. Ventilador automático
10. Diodo LED RGB
  - 10.1 Luz roja, verde y azul (RGB Ánodo Común)
  - 10.2 Luz roja, verde y azul (RGB Cátodo Común)
11. Arcoiris de luz (RGB)
  - 11.1 Arcoiris de luz (RGB Ánodo Común)
  - 11.2 Arcoiris de luz (RGB Cátodo Común)
12. Display de 7 segmentos
13. Control de display de 7 segmentos con el teclado
14. Detector de objetos (CNY70)
15. Movimiento de un servomotor 180°
16. Movimiento de un servomotor con un potenciómetro
17. Medidor de distancias (ULTRASÓNICO)
18. Control de motores con puente en H (C.I. L293D)
19. Conexión de Display LCD
20. Control de LEDs con mando a distancia
21. Motor paso a paso
22. Sensor de sonido
23. Sensor de línea
24. Módulo display de 4 dígitos
25. Módulo matriz de LEDs
26. Teclado numérico
27. Acceso por contraseña
28. Sensor de color
29. Módulo relé
30. Sensor acelerómetro
31. Sensor de temperatura y humedad
32. Sensor de alcohol
33. Sensor de caudal

Todas las prácticas están resueltas en la carpeta:  
entrenador\_arduino/PRÁCTICAS SENCILLAS

Para descargar la última versión, copiar el siguiente enlace:  
[www.microlog.es/entrenador\\_arduino.zip](http://www.microlog.es/entrenador_arduino.zip)

## **A. INTRODUCCIÓN**

Todo sistema robótico está formado por:

- Un sistema motriz encargado de generar los movimientos del robot.
- Un conjunto de sensores que ofrecerá al robot una visión del entorno que le rodea.
- Un sistema de control que analizará los datos recibidos a través de los sensores y decidirá cómo debe actuar el robot en consecuencia, mostrando así cierta autonomía e inteligencia.

Con un conjunto de materiales realizaremos pequeñas prácticas que nos ayudarán a comprender algunos aspectos relacionados con la robótica:

- Diseñar sencillos circuitos electrónicos.
- Realizar lecturas de datos a partir de diversos sensores.
- Realizar movimientos en motores que formarían el sistema motriz de cualquier robot.
- Realizar programas de control basados en Arduino.

Posteriormente estaremos capacitados para pasar al desarrollo de robots.

### **1. IDENTIFICACIÓN DE LOS MATERIALES**

- 1 Motor LOG 00
- 1 Miniservo LOG 06
- 1 Hélice tripala de plástico de 28 mm LOG 22
- 1 Pulsador para C.I. LOG 542
- 2 Diodo LED verde de 5 mm LOG 722
- 2 Diodo LED amarillo de 5 mm LOG 723
- 2 Diodo LED rojo de 5 mm LOG 724
- 1 Diodo RGB Cátodo Común de 5 mm LOG 729
- 1 Diodo RGB Ánodo Común de 5 mm LOG 730
- 1 LDR LOG 731
- 1 Potenciómetro 100K LOG 743
- 6 Resistencias de 220 ohmios LOG 748 220 (Rojo - Rojo - Marrón)
- 3 Resistencias de 470 ohmios LOG 748 470 (Amarillo - Morado - Marrón)
- 2 Resistencias de 1K ohmios LOG 748 1K (Marrón - Negro - Rojo)
- 1 Resistencias de 10K ohmios LOG 748 10K (Marrón - Negro - Naranja)
- 1 Transistor NPN BD547 LOG 751
- 1 Diodo de silicio LOG 760
- 1 Placa board de 400 contactos LOG 886
- 1 Sensor ultrasónico LOG 4046
- 1 Zumbador C.I. V LOG 7714
- 2 Conjunto de 10 Latiguillos LOG 9519
- 1 Cable USB LOG 4009
- 1 Arduino UNO LOG 4031

## **B. INSTALACIÓN DE PROGRAMAS**

### **1. INSTALAR ARDUINO IDE**

**Independientemente del lenguaje en el que vayamos a programar, Arduino IDE (processing) es indispensable que esté instalado en nuestro equipo.**

Primero conectamos la tarjeta controladora al ordenador a través del cable USB, y a continuación descargamos el software de programación desde la página [www.arduino.org](http://www.arduino.org). Dependiendo del sistema operativo que utilicemos procederemos de una de las siguientes maneras:

#### **1.1. INSTALACIÓN EN LINUX**

Nos hemos basado en la distribución Ubuntu para realizar esta guía de instalación, puesto que la mayoría de las distribuciones de Linux a nivel educativo se basan en ella.

Para trabajar con Arduino es preciso que nuestra distribución tenga instalados los siguientes paquetes además de Arduino IDE:

- sun's java runtime (jre)
- gcc-avr
- avr-libc
- binutils-avr

Para instalar estos paquetes se precisa la clave de administrador. Abrimos un terminal y ejecutamos las siguientes instrucciones:

```
$ sudo add-apt-repository ppa:arduino-ubuntu-team  
$ sudo apt-get update  
$ sudo apt-get install arduino
```

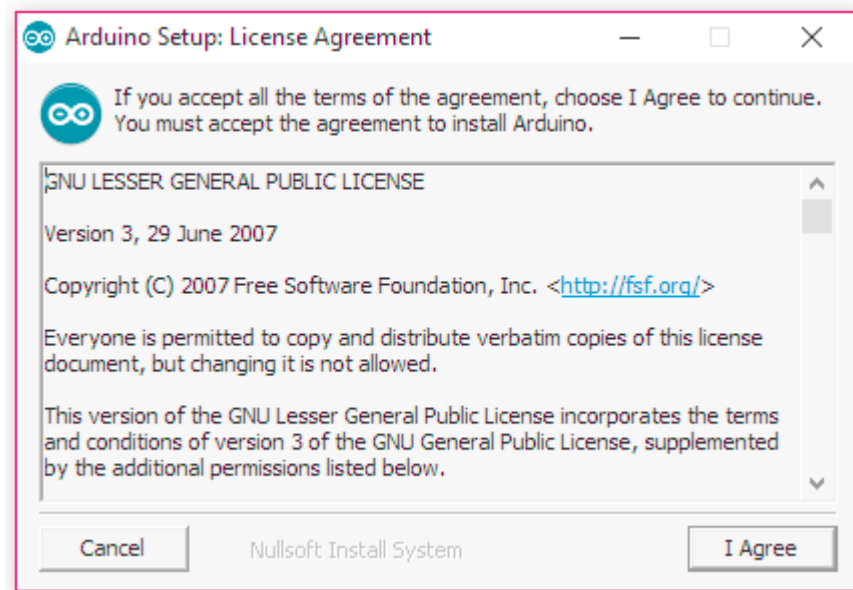
Para ejecutar la aplicación Arduino IDE basta con acceder al menú Aplicaciones > Programación > Arduino

## B. INSTALACIÓN DE PROGRAMAS

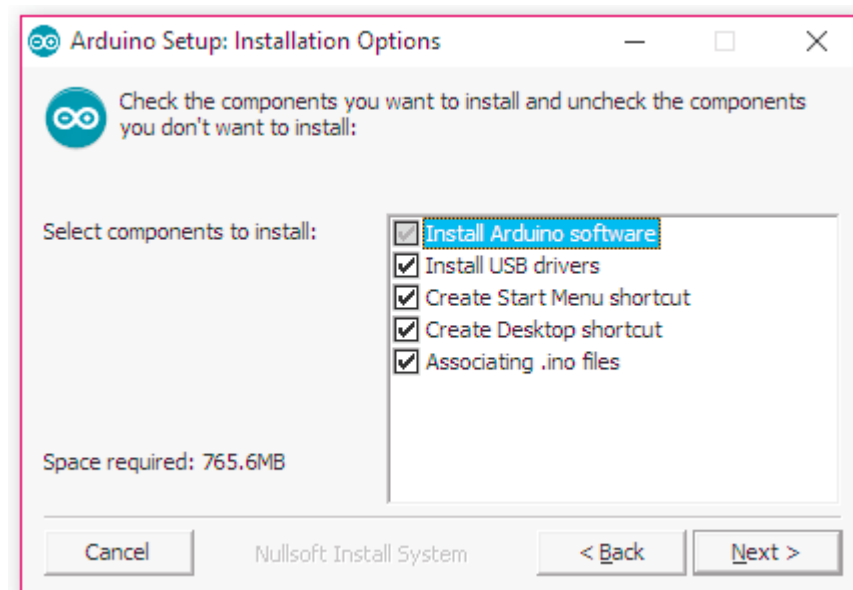
### 1.2. INSTALACIÓN EN WINDOWS

Descargamos el software de Arduino desde la sección "downloads" de [www.arduino.org](http://www.arduino.org) y procedemos a ejecutar el programa descargado.

- Aceptamos el acuerdo de licencia pulsando "I Agree".

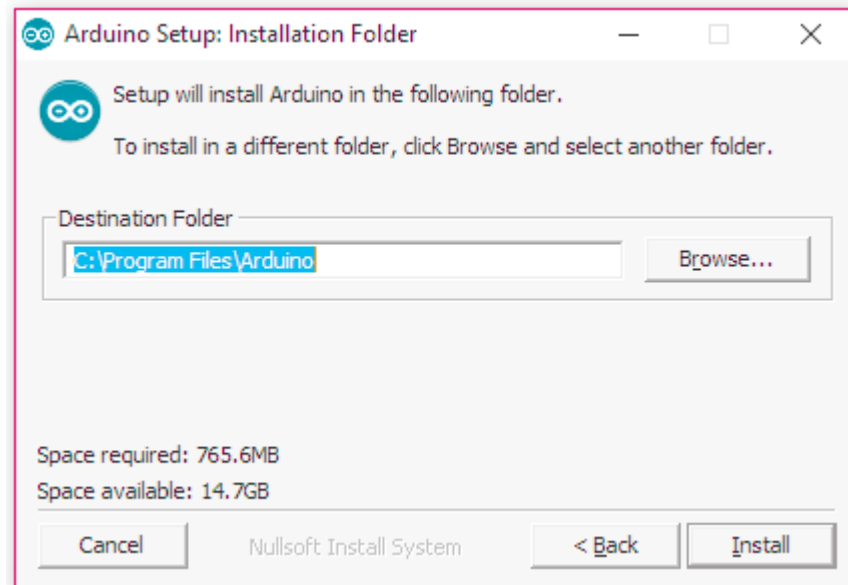


- Marcamos todos los componentes a instalar y pulsamos "Next".

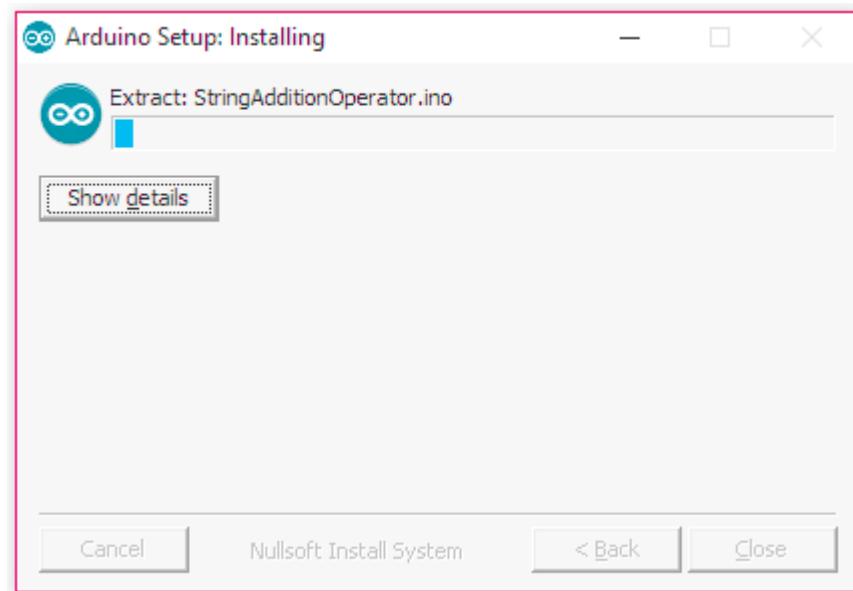


## B. INSTALACIÓN DE PROGRAMAS

- Seleccionamos la ruta donde se instalará el programa Arduino IDE y pulsamos “Install”.



- Comienza la instalación.



- Una vez finalizada la instalación pulsamos “Close”.



## B. INSTALACIÓN DE PROGRAMAS

En nuestro escritorio habrá el siguiente icono que dará acceso al sistema de programación Arduino IDE.



Al entrar en el programa veremos la siguiente pantalla.



En ella podemos crear nuestros propios programas introduciendo las líneas de código y posteriormente cargarlos en la tarjeta Arduino, utilizando la opción de menú *cargar* que se incluye en el menú *archivo*.

O bien, podemos cargar directamente en la tarjeta Arduino los ejemplos que podemos encontrar en la opción *ejemplos* del menú *archivos*.

## B. INSTALACIÓN DE PROGRAMAS

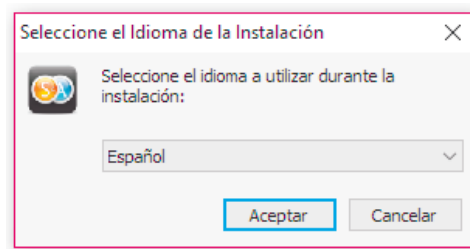
### 2 INSTALACIÓN DE SCRATCH (S4A)

Si el usuario se siente más cómodo con entornos de programación gráficos, es posible programar Arduino con una variante de SCRATCH, esta versión se llama S4A.

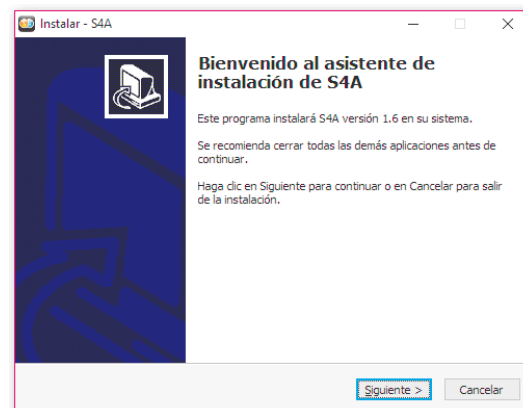
Para que funcione correctamente es preciso tener instalado el software Arduino IDE que hemos instalado en el capítulo anterior.

A continuación, descargamos S4A desde el siguiente enlace: <http://s4a.cat>

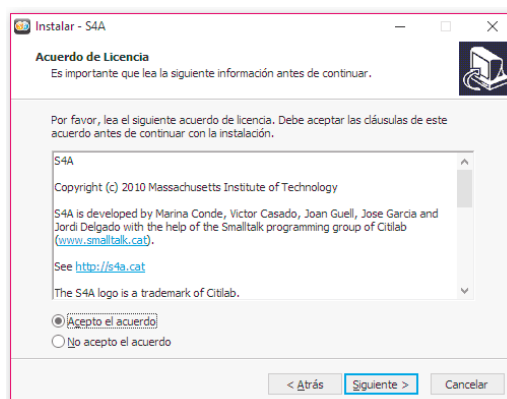
- Ejecutamos el instalador descargado y seleccionamos el idioma de instalación.



- Pulsamos en siguiente.

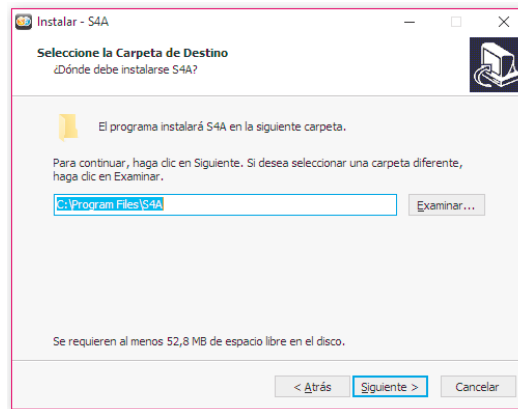


- Aceptamos el acuerdo de licencia y pulsamos siguiente.

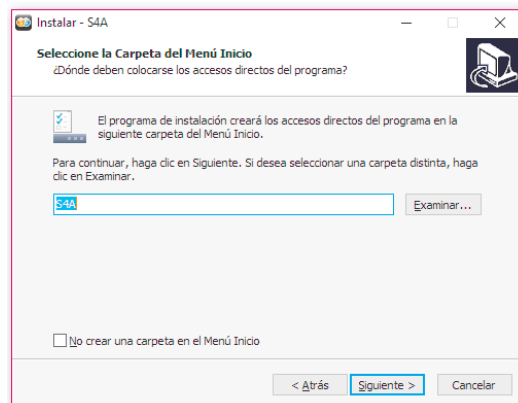


## B. INSTALACIÓN DE PROGRAMAS

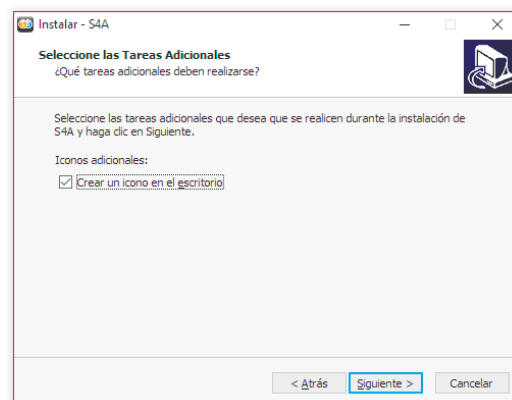
- Seleccionamos la carpeta de destino.



- Seleccionamos el destino de los accesos directos.

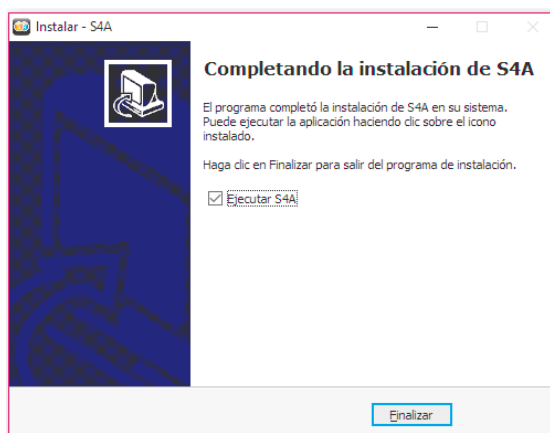
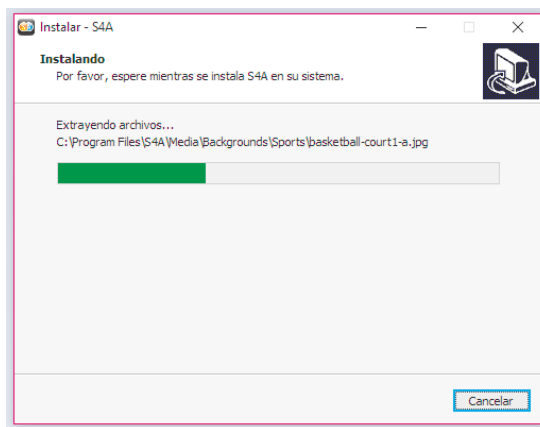
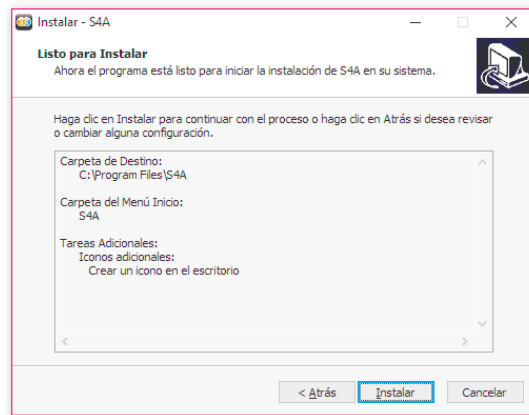


- Indicamos si queremos crear un acceso directo en el escritorio.



## B. INSTALACIÓN DE PROGRAMAS

- Y pulsamos instalar.



## **B. INSTALACIÓN DE PROGRAMAS**

### **3 INSTALACIÓN DE MBLOCK**

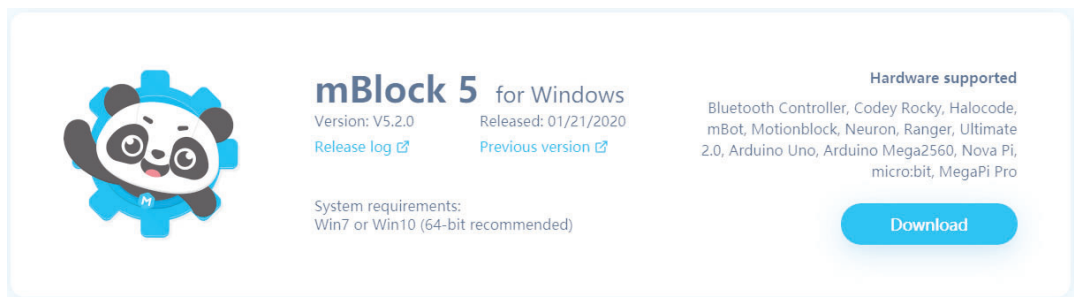
Descargar la versión más actualizada del software de instalación tanto para Linux como para Windows:

<http://www.mblock.cc/download>

- Pulsamos sobre "windows".



- Seleccionamos mBlock 5 y pinchamos en Download.



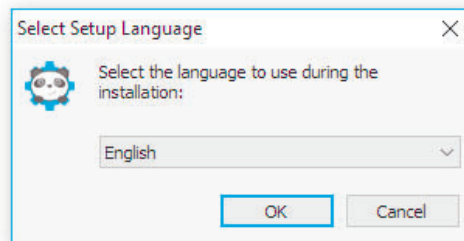
## B. INSTALACIÓN DE PROGRAMAS

### 3.1. INSTALACIÓN EN WINDOWS

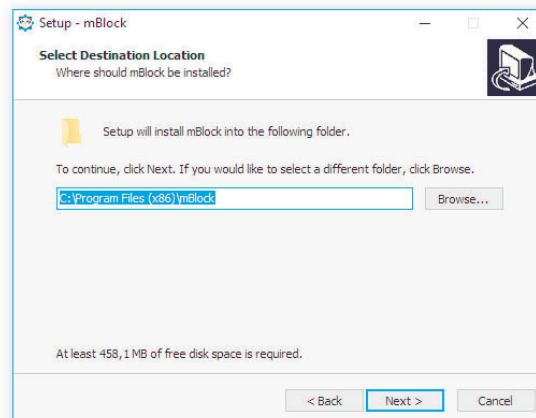
Hacemos doble clic sobre el archivo V5.2.0.exe para abrir la aplicación y comenzar la instalación.



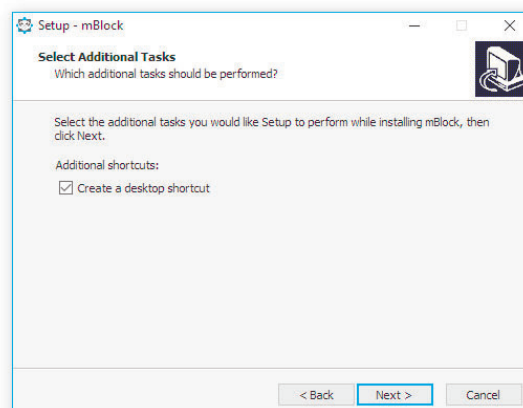
- Seleccionamos el idioma "English" y pulsamos sobre el botón "OK".



- Seleccionamos la carpeta donde queremos instalar el software o dejar la carpeta que viene por defecto. Pulsar sobre el botón "Next".

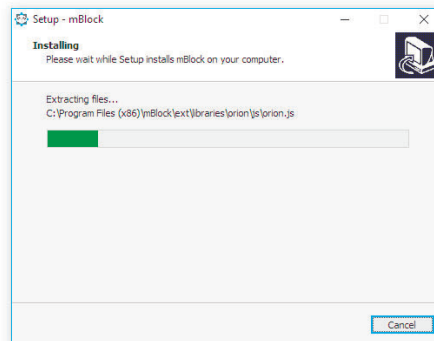
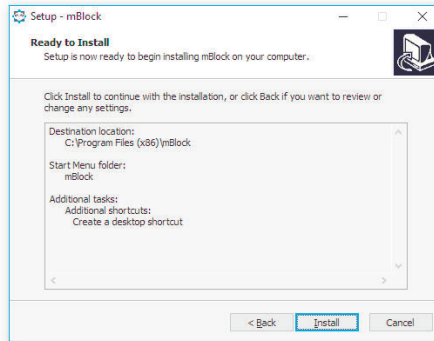


- Hacemos clic sobre "Create a desktop shortcut" si queremos crear un acceso directo al programa en el escritorio.

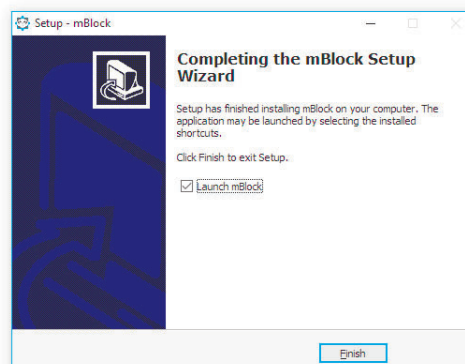


## B. INSTALACIÓN DE PROGRAMAS

- Revisamos que todo está correcto y pulsamos sobre el botón "Next".



- Seleccionamos si queremos ejecutar ahora el programa o no, y pulsar sobre el botón "Finish".

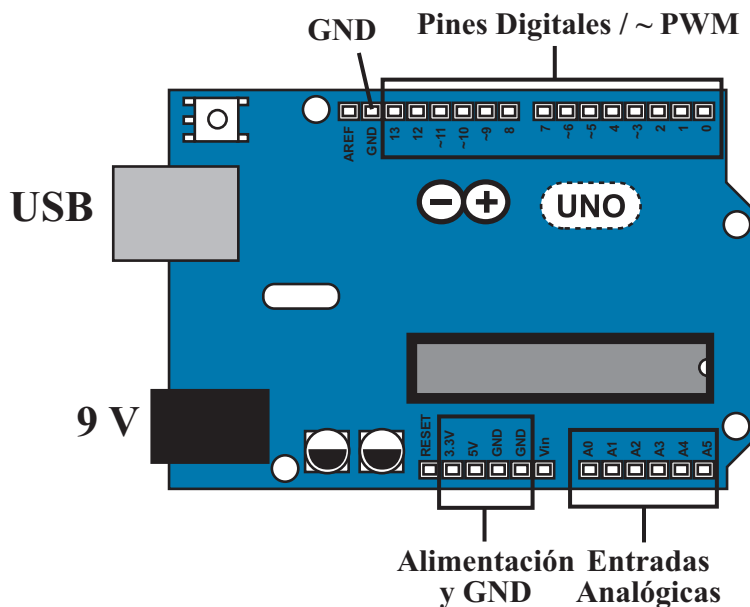


## C. TEORÍA BÁSICA

Para que el robot adquiriera cierta inteligencia es muy importante saber elaborar la correcta programación del sistema de control.

Arduino UNO es una placa electrónica que podemos utilizar como sistema de control de nuestros robots gracias a su microcontrolador ATmega328. Contiene la siguiente distribución de pines:

- 14 pines digitales que pueden ser configurados como entradas o como salidas, 6 de las cuales (serigrafiadas con el símbolo ~) pueden utilizarse tanto como señales digitales PWM o como salidas analógicas.
- 6 entradas analógicas serigrafiadas desde A0 hasta A5.
- 3 pines GND - toma de tierra.
- 1 Pin de alimentación a 5V.
- 1 Pin de alimentación a 3.3V.
- Conector de alimentación de la tarjeta a 9V.
- Conector de alimentación y programación puerto USB.





## C. TEORÍA BÁSICA

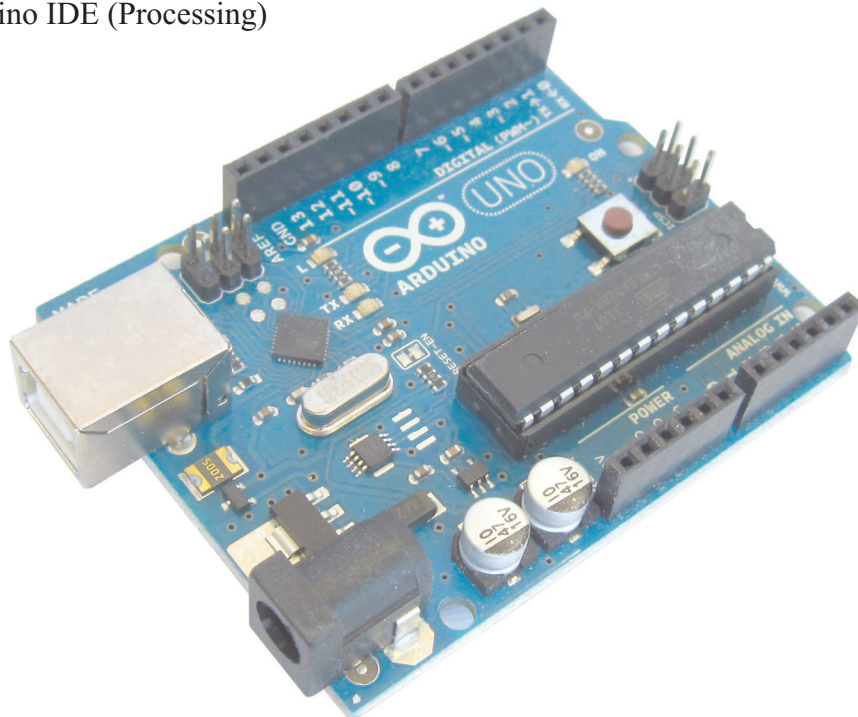
Una ventaja del uso de las tarjetas Arduino UNO es la posibilidad de programar usando lenguajes técnicos basados en la introducción de líneas de código o usando lenguajes gráficos basados en el uso de bloques, más sencillos y fáciles de comprender para usuarios principiantes y niños.

Principalmente utilizaremos los siguientes lenguajes:

- **S4A:** Es un lenguaje basado en bloques, supone eliminar fallos de sintaxis durante la elaboración de los programas, ya que programar con estos sistemas es como realizar puzzles. Muy utilizado en Primaria y Secundaria para acercar la programación a los niños de una forma muy intuitiva, pero pierde sentido en la elaboración de robots independientes (autónomos) porque la placa no tiene autonomía cuando se programa con S4A, precisando estar conectada siempre al PC.
- **mBLOCK:** Es un lenguaje basado en el uso de bloques, lo que facilita notablemente el proceso de desarrollo de los programas. La ventaja que ofrece frente a S4A es que mantiene el carácter autónomo de la placa, siendo una buena alternativa para el diseño de robots.
- **Arduino IDE (Processing):** Es un lenguaje basado en líneas de código, cuya gramática supone cierta complejidad.

A lo largo de este manual, el alumno aprenderá a instalar y utilizar los diferentes sistemas de programación que puede emplear para programar su robot y posteriormente utilizará el lenguaje que mejor se adapte a su nivel y proyecto.

- S4A (Scratch)
- mBLOCK (Scratch)
- Arduino IDE (Processing)



## C. TEORÍA BÁSICA

### 1. INSTRUCCIONES BÁSICAS

#### 1.1. INICIALIZACIÓN

Todo programa debe inicializarse:

En **Scratch** comienza con un clic de ratón sobre el icono de la bandera, y se paraliza pulsando sobre el icono del círculo rojo.



Esta instrucción será la primera que incluyamos en nuestro proyecto. El resto de instrucciones se irán agregando bajo esta. Si nos fijamos en el dibujo, en la parte inferior de la instrucción encontraremos una pequeña pestaña. Cuando acercamos otra instrucción hacia la pestaña, aparece una línea blanca. Esa línea blanca nos indica que la instrucción se va a añadir al programa en cuanto soltemos el botón del ratón.

- En **Scratch** pulsamos en el bloque de "Control" y arrastramos "al presionar ":



- En **mBlock** pulsamos en el bloque de "Eventos" y arrastramos "al presionar ":



- En **Arduino IDE** tenemos que escribir:

```
void setup () {
}
```



## C. TEORÍA BÁSICA

### 1.2. BUCLES

Las instrucciones de tipo bucle son aquellas que suponen la repetición de una secuencia de instrucciones. Pueden ser bucles de tipo infinito, o bucles que se ejecutan un determinado número de veces.

Genera un bucle infinito dentro del programa. La secuencia de instrucciones que incluyamos dentro de esta instrucción se ejecutará continuamente, sin final.

- En **Scratch** o **mBlock** pulsamos en el bloque de "Control" y arrastramos "por siempre":



- En **Arduino IDE** tenemos que escribir:

```
void loop () {  
}
```

Ejecuta un determinado número de veces las instrucciones contenidas dentro del bucle.

- En **Scratch** o **mBlock** pulsamos en el bloque de "Control" y arrastramos "repetir 10":



- En **Arduino IDE** tenemos que escribir:

```
void loop () {  
    for (int i = 0; i < 10; i++){  
        ...  
    }  
}
```

## C. TEORÍA BÁSICA

### 1.3. ESPERAR UN SEGUNDO

Genera una pausa de un segundo durante la ejecución del programa.

- En **Scratch** o **mBlock** pulsamos en el bloque de "Control" y arrastramos "esperar 1 segundos":



- En **Arduino IDE** tenemos que escribir:

```
void loop () {  
    delay (1000);  
}
```

### 1.4. CONDICIONALES

Son instrucciones en las que se evalúa una condición, y en función de si ésta es cierta o falsa, se procede a ejecutar unas instrucciones u otras.

Sólo se ejecutan las instrucciones contenidas dentro del "si" en caso de que sea cierta la condición que se incluye junto al "si". Esta condición normalmente será una comparación de tipo numérico.

- En **Scratch** o **mBlock** pulsamos en el bloque de "Control" y arrastramos "si <>":



- En **Arduino IDE** tenemos que escribir:

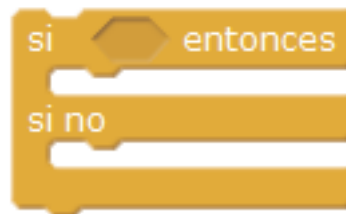
```
void loop () {  
    if (condición){  
        Instrucciones;  
    }  
}
```

Sentencia condicional, que ejecuta un conjunto de instrucciones u otro en función de si se da la condición indicada. Como ejemplo, podríamos programar una comparación de dos números que nos muestre en pantalla cuál es el mayor:

## C. TEORÍA BÁSICA

En este caso, disponemos de dos juegos de instrucciones. Las instrucciones que introducimos bajo el "sí" se ejecutarán en caso de que la condición sea cierta. En caso de que sea falsa, se ejecutarán las instrucciones bajo el "sí no".

- En **Scratch** o **mBlock** pulsamos en el bloque de "Control" y arrastramos "si <>":



- En **Arduino IDE** tenemos que escribir:

```
if(A>B){
    Serial.println('A es mayor que B');
}
else{
    Serial.println('B es mayor que A');
}
```

### 1.5. INSTRUCCIONES DE OPERACIÓN (OPERADORES)

En esta sección encontraremos instrucciones que utilizaremos para realizar operaciones matemáticas y comparaciones de tipo lógico para su uso en condiciones.

Las instrucciones redondeadas son operaciones matemáticas.

- En **Scratch** o **mBlock** pulsamos en el bloque de "Operadores" y arrastramos por ejemplo "\_ + \_":



Las instrucciones con los bordes puntiagudos son comparaciones que utilizaremos para generar condiciones en bucles e instrucciones condicionales. Pueden enlazarse varias condiciones con operadores de tipo:

- y: deben cumplirse ambas condiciones
- o: se cumple alguna de las condiciones
- no: no se da la condición



- Ejemplo en **Arduino IDE**:

```
void loop () {
    for (int i = 0; i < 10; i++){
        ...
    }
}
```

## C. TEORÍA BÁSICA

### 1.6. INSTRUCCIONES DE MOVIMIENTO

En esta sección se incluyen las instrucciones propias de **Arduino UNO**.

En **S4A** se ha eliminado el concepto de pines y estados (high / low) y se utiliza un lenguaje más natural (encendido/apagado) para acceder a cada una de las características de la tarjeta controladora.

#### Encender y apagar un pin digital:

Con estas instrucciones encendemos o apagamos cualquier operador conectado a los pines digitales.

- En **Scratch** pulsamos en el bloque de "Movimiento" y arrastramos "digital **13** encendido" o "digital **13** apagado", podemos sustituir el 13 por 12, 11 o 10. El resto de pines digitales de la tarjeta controladora no podemos utilizarlos con estas dos instrucciones. Han sido reservados para otros usos que veremos más adelante.



- En **mBlock** pulsamos en el bloque de "Robots" y arrastramos "fijar salida pin digital **9** a **ALTO**", podemos sustituir el 9 por el número de salida deseada y ALTO por BAJO, dependiendo si queremos activar o desactivar el pin.



- En **Arduino IDE** tenemos que escribir:

```
digitalWrite(nº_de_pin, estado);
```

- Activa el operador que tengamos conectado a un determinado pin:

```
digitalWrite(nº_de_pin, HIGH)
```

- Desactiva el operador que tengamos conectado a un determinado pin:

```
digitalWrite(nº_de_pin, LOW)
```

## C. TEORÍA BÁSICA

- Ejemplo:

En la función setup hemos definido el pin número 13 como un pin de salida al que podremos conectar físicamente operadores de salida como un led o un zumbador.

En la función loop introducimos las instrucciones de nuestro programa:

```
void setup() {
    pinMode(13, OUTPUT); //Declaramos el pin 13 de salida
}

void loop() {
    digitalWrite(13, HIGH); //Activar la salida 13
    delay(1000);           //Esperar 1 segundo
    digitalWrite(13, LOW); //Desactivar la salida 13
    delay(1000);           //Esperar 1 segundo
}
```

Así provocamos una intermitencia del dispositivo conectado al pin 13, encendiendo y apagando dicho operador con intervalos de 1 segundo de espera..

### Encender y apagar un pin analógico

Con esta instrucción modulamos la señal de un operador conectado a los pines PWM pudiendo enviar un valor desde 0 hasta 255.

- En **Scratch** pulsando en el bloque de "Movimiento" y arrastrando "analógico 9 valor 255", podemos sustituir el 9 por 6 o 5 y 255 por un valor entre 0 y 255. El resto de pines analógicos de la tarjeta controladora no podemos utilizarlos con esta instrucción. Han sido reservados para otros usos que veremos más adelante.



- En **mBlock** pulsando en el bloque de "Robots" y arrastrando "fijar pin pwm 5 a 0", podemos sustituir el 9 por el número de salida PWM deseada y 0 por un valor entre 0 y 255, dependiendo de cuanta intensidad queramos obtener.



## C. TEORÍA BÁSICA

- En **Arduino IDE** tenemos que escribir:

**analogWrite(nº\_de\_pin, valor)**

Si conectamos un led al pin número 9 de la placa y realizamos el siguiente programa, veremos que el led se enciende con diferente intensidad de luz en función del valor introducido en la señal analogWrite:

```
void setup() {  
    pinMode(9, OUTPUT); // Declara el pin de Salida  
}  
  
void loop() {  
    analogWrite(9, 0); // Pin desactivado  
  
    delay(1000);  
    analogWrite(9, 50);  
    delay(1000);  
    analogWrite(9, 100);  
    delay(1000);  
    analogWrite(9, 150); // Pin activado a media potencia  
    delay(1000);  
    analogWrite(9, 200);  
    delay(1000);  
    analogWrite(9, 250); // Pin activado al máximo  
    delay(1000);  
}
```



## C. TEORÍA BÁSICA

### 1.7. CONTROL DE SERVOMOTORES

Con las siguientes instrucciones, podremos controlar el movimiento de un servomotor.

- En **Scratch** pulsamos en el bloque de "Movimiento" y aparecen 3 bloques para controlar servomotores "motor 8 apagado", "motor 8 dirección **horario**" y "motor 8 ángulo **180**":
- Con la instrucción "motor 8 apagado" paramos cualquier movimiento del servomotor, si pinchamos en el 8 podemos cambiar el pin por 4 , 7 u 8.



- Con la instrucción "motor 8 dirección **horario**" activamos el servomotor, girando en sentido horario, si pinchamos en el 8 podemos cambiar el pin por 4 , 7 u 8; y pinchando en horario, cambiamos el sentido de giro a antihorario.



- Con la instrucción "motor 8 ángulo **180**" colocamos el servo en el ángulo 180. Si pinchamos en el 8, podemos cambiar el pin por 4 , 7 u 8 y pinchando en el 180 podemos escribir el ángulo deseado entre 0 y 180.



- En **mBlock** pulsamos en el bloque de "Robots" y arrastramos "fijar ángulo del pin 9 del servo a 90", colocamos el servo en el ángulo 90. Si pinchamos en el 9 podemos sustituir el 9 por el número de salida deseada y pinchando en el 90 podemos escribir el ángulo deseado entre 0 y 180.



- En **Arduino IDE** tenemos que incluir la librería de "servo.h" para que nos reconozca las instrucciones para servomotores:

```
#include <servo.h> // Libreria para servomotores

void setup() {
  myservo.attach(9); // Indicamos el pin del servomotor
}

void loop() {
  myservo.write(90); // Colocamos el servo en 90°
}
```

## C. TEORÍA BÁSICA

### 1.8. VARIABLES

Una variable es un espacio donde guardar un dato. Podríamos definirla como una caja a la que ponemos un nombre y donde guardamos un dato. Cuando queremos acceder al dato, le llamamos por el nombre asignado.

Dispone de un juego de instrucciones que nos permite crear variables, almacenar por ejemplo los datos proporcionados por un sensor, fijar el valor de la variable a un determinado dato, realizar operaciones sobre la variable y mostrar el contenido de la variable.

- En **Scratch** pulsamos en el bloque de "Variables", pichamos en "Nueva variable" y escribimos por ejemplo "Dato" y nos aparecen los siguientes bloques:



- En **mBlock** pulsamos en el bloque de "Datos y Bloques", pichamos en "Crear una variable" y escribimos por ejemplo "Dato" y nos aparecen los siguientes bloques:



## **C. TEORÍA BÁSICA**

- En **Arduino IDE**, utilizamos la instrucción "int" para definir una variable, ponemos el siguiente código una línea por encima de void setup:

```
Int nombre_de_variable = dato;
```

"Dato" será el número o carácter que emplearemos como dato inicial en la variable:

```
int Dato;  
  
void setup() {  
}  
  
void loop() {  
}
```

## C. TEORÍA BÁSICA

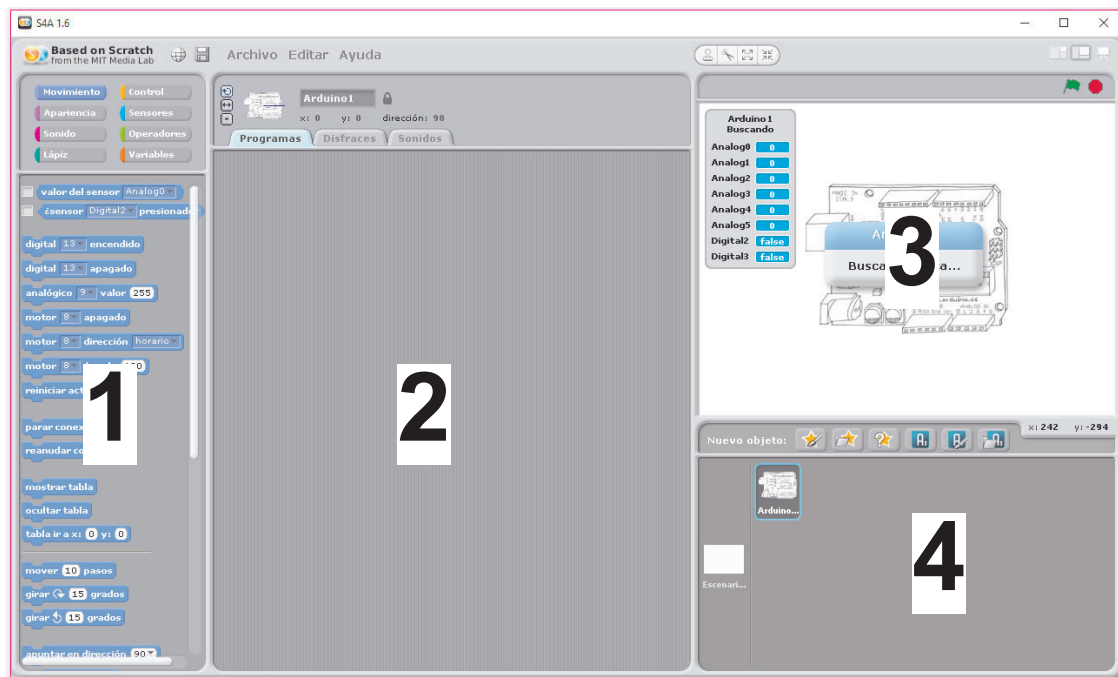
### 2. ENTORNO GRÁFICO DE PROGRAMACIÓN

#### 2.1. S4A

**S4A** es una modificación del software libre Scratch que nos permite crear programas para Arduino UNO, pero teniendo en cuenta que los proyectos siempre serán dependientes de su conexión con el PC.

Está basado en programación por bloques y sus instrucciones han sido diseñadas con un lenguaje natural, eliminando términos técnicos y empleando una terminología más natural. Así se facilita el acceso a la programación en niveles educativos básicos.

El entorno de programación se divide en varias zonas.



Zona 1:

Conjuntos de instrucciones. Pulsando en cada uno de los botones de la zona superior, se despliega todo el catálogo de instrucciones de la categoría seleccionada.

Zona 2:

Aquí estructuramos nuestro programa arrastrando instrucciones desde la zona 1.

Zona 3:

Escenario, donde normalmente veríamos el resultado de nuestro programa cuando éste es virtual. En el caso del trabajo con Arduino UNO, lo normal es olvidarnos de personajes y escenarios ya que buscamos ver el resultado directamente en el proyecto que se ha realizado físicamente.

Zona 4:

Personajes que van a intervenir en nuestro programa. Trabajando con Arduino UNO, lo más habitual es que nuestro único personaje sea la propia tarjeta controladora.

## C. TEORÍA BÁSICA

### 2.1.1 CATÁLOGO DE INSTRUCCIONES

Las instrucciones se clasifican por colores y en las siguientes categorías:

- Movimiento (azul oscuro): Conjunto de instrucciones relacionadas con el control de los pines de la tarjeta Arduino, así como control del movimiento de cualquier personaje que tengamos dentro del escenario de representación.

#### Movimiento

- Apariencia (morado): Instrucciones orientadas a modificar el aspecto de los personajes de nuestra aplicación. Para el caso de Arduino, es un conjunto de instrucciones que apenas se utilizan .

#### Apariencia

- Sonido (rosa): Conjunto de instrucciones relacionadas con la elaboración de aplicaciones musicales, emitiendo sonidos y notas musicales.

#### Sonido

- Lápiz (verde oscuro): Scratch nos ofrece la posibilidad de que los personajes dejen un rastro durante sus movimientos por el escenario, como si arrastrase un lápiz durante su trayectoria. Este rastro se genera con las instrucciones que podremos encontrar en esta sección.

#### Lápiz

- Control (Yema): Las instrucciones incluidas en esta sección son imprescindibles para crear la lógica de nuestros programas. Incluyen condicionales, bucles y llamadas de acción.

#### Control

- Sensores (azul): Instrucciones de interacción con el ratón, el teclado, sonidos y los personajes.

#### Sensores

- Operadores (verde claro): Operaciones matemáticas, lógicas y con cadenas de texto.

#### Operadores

- Variables (naranja): Instrucciones para el almacenamiento y gestión de datos..

#### Variables

## **C. TEORÍA BÁSICA**

### **2.1.2. PRIMEROS PASOS CON S4A**

Para que **S4A** reconozca la tarjeta Arduino debemos seguir estos pasos:

- Abrir el archivo S4AFireware15.ino. Al abrirlo, el sistema Arduino IDE nos indica que debe estar ubicado en una carpeta concreta, aceptamos y automáticamente creará la carpeta y abrirá el archivo.
- Pinchar en la pestaña “Herramientas” y comprobar que la tarjeta seleccionada es Arduino UNO.
- Pinchar en la pestaña “Herramientas” y comprobar que está seleccionado el puerto correcto como indicamos en el apartado anterior.
- Pulsar en cargar y grabar el firmware en la tarjeta Arduino.
- Abrimos S4A y si el proceso se ha hecho correctamente, aparecerá el mensaje “Buscando placa” y en unos segundos, cuando la tarjeta se haya detectado, el mensaje desaparecerá.

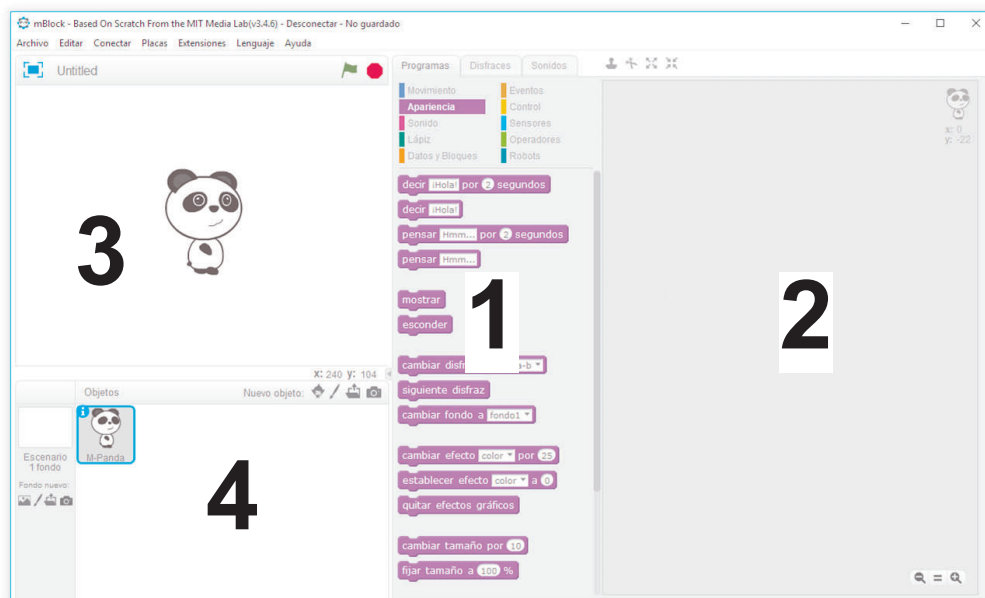
## C. TEORÍA BÁSICA

### 2.2. MBLOCK

MBLOCK es una modificación del software libre Scratch que nos permite crear programas para Arduino UNO, a diferencia de S4A, los proyectos se pueden cargar a la tarjeta, permitiendo la desconexión con el PC.

Está basado en programación por bloques y sus instrucciones han sido diseñadas con un lenguaje natural, eliminando términos técnicos y empleando una terminología más natural. Así se facilita el acceso a la programación en niveles educativos básicos.

El entorno de programación se divide en varias zonas.



Zona 1:

Conjuntos de instrucciones. Pulsando en cada uno de los botones de la zona superior. En esta zona 1 se despliega todo el catálogo de instrucciones de la categoría seleccionada.

Zona 2:

Aquí estructuramos nuestro programa arrastrando instrucciones desde la zona 1.

Zona 3:

Escenario, donde normalmente veríamos el resultado de nuestro programa cuando éste es virtual. En el caso del trabajo con Arduino UNO, lo normal es olvidarnos de personajes y escenarios ya que buscamos ver el resultado directamente en el proyecto que se ha realizado físicamente.

Zona 4:

Personajes que van a intervenir en nuestro programa. Trabajando con Arduino UNO, lo más habitual es que nuestro único personaje sea la propia tarjeta controladora.

## C. TEORÍA BÁSICA

### 2.2.1. CATÁLOGO DE INSTRUCCIONES

Las instrucciones se clasifican por colores y en las siguientes categorías:

- Movimiento (azul oscuro): Conjunto de instrucciones relacionadas con el control de los pines de la tarjeta Arduino, así como control del movimiento de cualquier personaje que tengamos dentro del escenario de representación.

#### Movimiento

- Apariencia (morado): Instrucciones orientadas a modificar el aspecto de los personajes de nuestra aplicación. Para el caso de Arduino, es un conjunto de instrucciones que apenas se utilizan .

#### Apariencia

- Sonido (rosa): Conjunto de instrucciones relacionadas con la elaboración de aplicaciones musicales, emitiendo sonidos y notas musicales.

#### Sonido

- Lápiz (verde oscuro): Scratch nos ofrece la posibilidad de que los personajes dejen un rastro durante sus movimientos por el escenario, como si arrastrasen un lápiz durante su trayectoria. Este rastro se genera con las instrucciones que podremos encontrar en esta sección.

#### Lápiz

- Datos y bloques (naranja): Instrucciones para el almacenamiento y gestión de datos.

#### Datos y Bloques

- Eventos (marrón): Las instrucciones incluidas en esta sección son imprescindibles para inicializar nuestro programa, llamadas de acción.

#### Eventos

- Control (yema): Las instrucciones incluidas en esta sección son muy importantes para crear la lógica de nuestros programas (condicionales, bucles).

#### Control

- Sensores (azul): Instrucciones de interacción con el ratón, el teclado, sonidos y los personajes.

#### Sensores

- Operadores (verde claro): Operaciones matemáticas, lógicas y con cadenas de texto.

#### Operadores

- Robots (turquesa): Instrucciones básicas para controlar distintos sensores o módulos.

#### Robots



## C. TEORÍA BÁSICA

### 2.2.2. PRIMEROS PASOS CON MBLOCK

Una vez instalado, abrir "mBlock":

- Pinchar en la pestaña "Placas", y pinchar donde pone "Arduino UNO".
- Pinchar en la pestaña "Conectar", seleccionar donde pone "Puerto serie" y pinchar donde pone "COM\*Numero\*".

Para trabajar directamente desde el PC de forma no autónoma, primero hay que cargar el Firmware.

- Pinchar en la pestaña "Conectar", seleccionar donde pone "Actualizar Firmware".
- Una vez cargado, ya podremos trabajar usando la instrucción "al presionar ".



Para trabajar sin conexión al PC de forma autónoma:

- Pinchar en las instrucciones "Robots" y arrastrar "Programa de Arduino".



- Pinchar encima de la instrucción "Programa de Arduino", ahora se ha ocultado la zona 3 y 4, y nos aparece la zona 5. Tenemos dos botones en la zona 5:
  - **Atrás:** Volver a visualizar las zonas 1 y 2.
  - **Subir a Arduino:** Guardar el programa en nuestra placa controladora para poder funcionar de forma autónoma.

## **C. TEORÍA BÁSICA**

### **2.3. ARDUINO IDE**

Al instalar Arduino IDE, los drivers de la tarjeta quedan instalados en el PC. Para comprobar su correcto funcionamiento conectamos la tarjeta al ordenador a través del cable USB. Automáticamente el ordenador nos reconocerá la tarjeta.

Para introducir en ella los programas que hagamos, necesitamos saber qué puerto COM ha asignado el sistema operativo a la tarjeta:

- Pulsamos con el botón derecho en el icono de “equipo” en el panel de control de Windows.
- Pulsamos propiedades.
- Pulsamos administrador de dispositivos.
- Y desplegamos el menú “puertos com y lpt”.
- En la lista desplegable veremos que uno de los puertos está asociado a la tarjeta Arduino UNO. Debemos recordar que número de puerto se ha asociado a la tarjeta para proceder a la carga de programas correctamente.
- Si la tarjeta Arduino UNO no aparece en el listado de puertos, el ordenador no está reconociendo correctamente la tarjeta.

#### **2.3.1. PRIMEROS PASOS CON ARDUINO IDE**

Para cargar un programa en la tarjeta desde Arduino IDE.

- Abrimos el programa.
- Seleccionamos del menú herramientas > puerto serial, el puerto en el que se instaló la tarjeta Arduino UNO.
- Seleccionamos del menú herramientas > tarjeta, el modelo de la tarjeta, en nuestro caso Arduino UNO.
- Seleccionamos un programa de ejemplo para probar la conectividad de la tarjeta. En el menú "archivo" seleccionamos ejemplos > 01. Basics > Blink.
- Seleccionamos del menú archivo la opción cargar, y el programa queda guardado en la tarjeta. A partir de ahora, si desconectamos la tarjeta del PC, ésta ejecutará continuamente el programa que hemos instalado en ella, basta con conectar una pila de 9V a la entrada de corriente.

Si la carga del programa se ha realizado correctamente, observaremos que parpadea un led naranja dentro de la tarjeta Arduino. Este LED está asociado al pin 13, nos mostrará la actividad de ese pin siempre que se active/desactive.

## C. TEORÍA BÁSICA

### 2.3.2. SINTAXIS DE PROCESSING

Es importante tener claros las siguientes reglas del lenguaje:

- Las instrucciones son sensibles a las mayúsculas y minúsculas. Es decir, si una instrucción está definida dentro de processing con una letra en mayúscula y las demás en minúscula, debemos respetarlo pues en caso contrario no la reconocerá y nos devolverá un mensaje de error.
- Toda línea termina en ; (punto y coma) excepto llaves, bucles y condicionales.
- Las asignaciones numéricas se realizan con el operador = y las comparaciones con == (doble igual).

Para familiarizarnos con la sintaxis vamos a realizar un repaso por las instrucciones más habituales con sencillos ejemplos.

### 2.3.3. ESTRUCTURA DE UN PROGRAMA PARA ARDUINO

Todo programa para Arduino realizado con processing consta de la siguiente estructura:

```
void setup() {  
    Instrucción;  
    Instrucción;  
    ...  
}  
  
void loop() {  
    Instrucción;  
    Instrucción;  
    ...  
}
```

Setup y loop son instrucciones del sistema que utilizaremos siempre:

- En **setup** configuramos los pines de la tarjeta controladora que vamos a utilizar en nuestro programa, pudiendo ser de entrada o de salida:

```
pinMode(nº_de_pin, INPUT); //Para definir el pin como entrada  
pinMode(nº_de_pin, OUTPUT); //Para definir el pin como salida
```

- En **loop** vamos a introducir el programa que queremos ejecutar dentro de la placa. Este programa se ejecutará continuamente como si de un bucle infinito se tratase.

## **C. TEORÍA BÁSICA**

### **2.3.4. INSTRUCCIONES PROPIAS DE PROCESSING**

A continuación se detallan algunas instrucciones de processing que serán de utilidad en la elaboración de nuestros programas:

#### **Visualización de datos en pantalla**

La visualización de datos en pantalla resulta de utilidad para localizar posibles errores en la lógica de nuestro programa. En processing esta tarea se realiza a través del Monitor serial.

Para ello primero hay que configurar el puerto serie introduciendo la línea.

```
Serial.begin(9600);
```

Dentro de la función setup.

Para visualizar un dato en la pantalla utilizamos la instrucción.

```
Serial.println(dato);
```

Dentro de la función loop.

Para poder ver los datos que se están enviando a través de println tenemos que abrir el monitor serial: menú herramientas > monitor serial.

Veamos un ejemplo:

```
void setup() {  
    Serial.begin(9600);  
}  
  
void loop() {  
    Serial.println("hola mundo");  
}
```

Si abrimos el monitor serial, veremos que nos muestra en pantalla la frase "hola mundo" continuamente.

#### **Pausas y retardos**

```
delay(Número_de_milisegundos)
```

Genera una pausa dentro del programa de un determinado tiempo indicado en milisegundos.

## C. TEORÍA BÁSICA

### Lectura de un PIN digital

#### **digitalRead(nº\_de\_pin)**

Nos permite leer los datos aportados por un operador conectado a un pin digital. Supongamos que conectamos un pulsador al pin 12. Definiendo ese pin como entrada podemos leer cuándo ha sido pulsado.

Veamos un ejemplo: Encendemos el led del pin 13 siempre que se presione el pulsador del pin 12.

```
void setup() {
    pinMode(12, INPUT);
    pinMode(13, OUTPUT);
}

void loop() {
    if (digitalRead(12)==HIGH){
        digitalWrite(13, HIGH);
    }
    Else{
        digitalWrite(13, LOW);
    }
}
```

Conectamos al pin 12 un pulsador y configuramos el pin como entrada (input) para realizar la lectura de las pulsaciones.

En la función loop indicamos que si la lectura del pulsador nos devuelve el valor HIGH es porque éste fue presionado. En este caso introducimos el valor HIGH en el pin 13, activando su led. En caso contrario, el pin 12 nos ha devuelto el valor LOW (no presionado), así pues, procedemos desactivando el led del pin 13 (LOW).

Observar que en la comparación hemos utilizado el operador == (doble igual). Si hubiésemos introducido un solo símbolo = realizaría una asignación, generando siempre una afirmación que no siempre será cierta.

Otra forma de escribir este mismo programa sería la siguiente:

```
void setup() {
    pinMode(12, INPUT);
    pinMode(13, OUTPUT);
}

void loop() {
    digitalWrite(13, digitalRead(12));
}
```

Configuramos el pin 13 con el mismo estado que el pin 12.

Si presionamos el pulsador, nos genera un estado HIGH que introducimos directamente en el pin 13, activando el led.

Si por el contrario nos genera un estado LOW porque el pulsador no fue presionado, guarda este estado en el pin 13, apagando el led.

## C. TEORÍA BÁSICA

### Lectura de un PIN analógico

#### **analogRead(nº\_de\_pin)**

Nos permite leer los datos aportados por un operador conectado a un pin analógico (A0 – A5). Supongamos que conectamos una resistencia de luz LDR al pin A0, podemos realizar lecturas de ese pin y ver la cantidad de luz detectada. Este pin no requiere de configuración porque los pines desde A0 hasta A5 son todo entradas analógicas.

Veamos un ejemplo: vamos a encender el led del pin 13 siempre que se detecte poca luz (supongamos para un valor de la LDR inferior a 50) a través de una LDR conectada al pin A0.

```
void setup() {  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    if (analogRead(0)<50){  
        digitalWrite(13, HIGH);  
    }  
    Else{  
        digitalWrite(13, LOW);  
    }  
}
```

### 2.3.5. DECLARACIÓN DE LIBRERÍAS

Algunos operadores contienen librerías de instrucciones específicas que facilitan su programación como es el caso de los servomotores, sensor ultrasonido, display, LCD...

Para incluir esas librerías introducimos una línea al principio del archivo de nuestro programa (sería la primera línea del archivo) como la siguiente:

```
#include <libreria.h>
```

Donde la palabra librería será sustituido por el nombre de la librería en cuestión a utilizar.

Una vez incluida esa línea podremos utilizar dentro de nuestro programa cualquier instrucción que se incluya en la librería.

A lo largo de las prácticas incluidas en el manual, conoceremos diferentes librerías que podemos utilizar con los operadores mencionados.

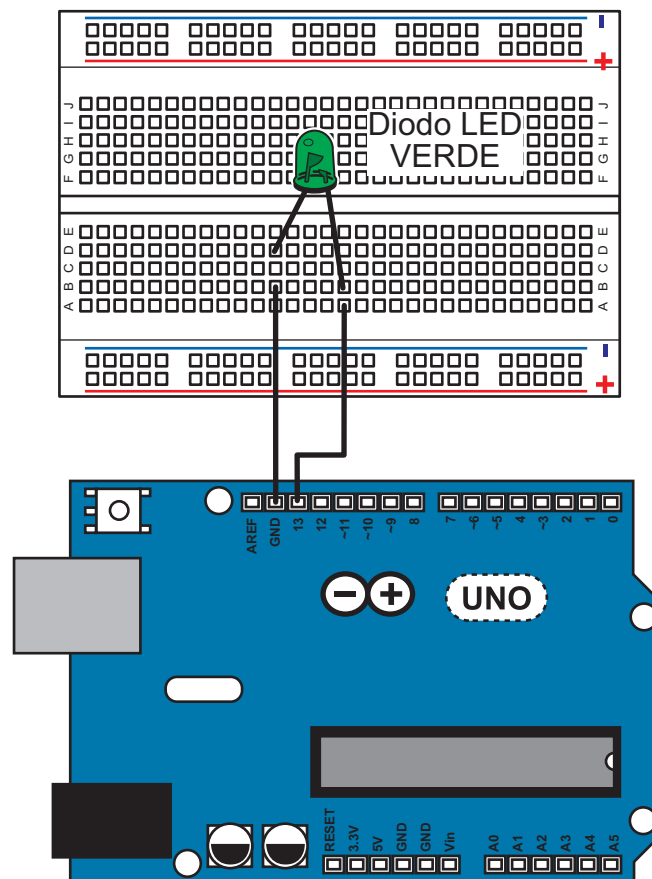
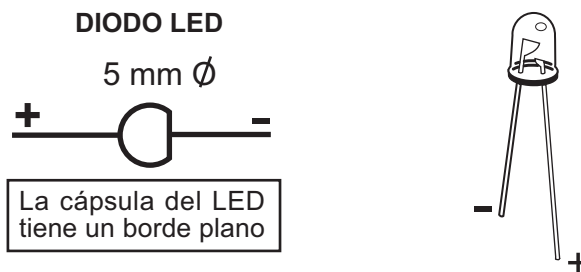
## C. TEORÍA BÁSICA

### 3. PROGRAMAR ENCENDIDO Y APAGADO DE UN DIODO LED PASO A PASO EN mBLOCK

#### 3.1. MONTAJE DE COMPONENTES ELECTRÓNICOS

Utilizar dos latiguillos cocodrilo-board macho para conectar el negativo del minikit LED verde al pin GND de la placa y el positivo del minikit al pin 13 de la placa.

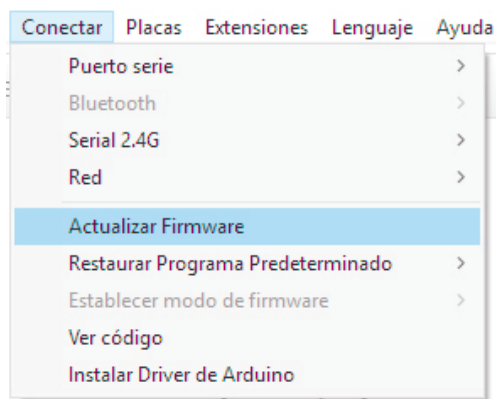
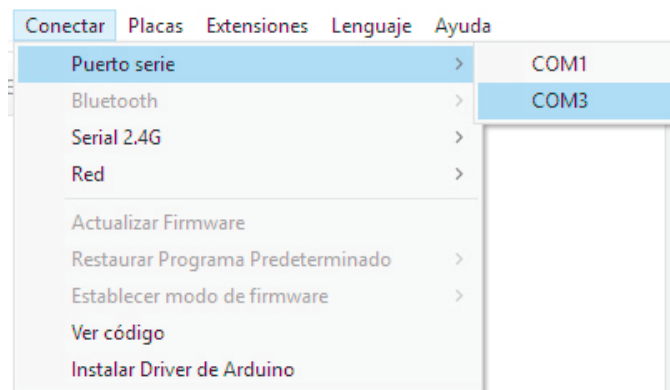
NOTA: *El diodo LED no requiere de resistencia, porque el pin 13 tiene incorporada una resistencia para su propio diodo LED.*



## C. TEORÍA BÁSICA

### 3.2. CONECTAR LA PLACA ARDUINO UNO

- Pinchar en la pestaña "Placas" y pinchar donde pone "Arduino UNO".
- Pinchar en la pestaña "Conectar", seleccionar donde pone "Puerto serie" y pinchar donde pone "COM\*Numero\*" (a cada placa se le asigna automáticamente un puerto).
- Cargar firmware pinchando en la pestaña "Conectar", seleccionar donde pone "Actualizar Firmware".





## C. TEORÍA BÁSICA

### 3.3. PROGRAMAR ENCENDIDO Y APAGADO DE UN DIODO LED VERDE

- Pinchar en la pestaña "Eventos" y arrastrar el bloque "al presionar".
- Pinchar en la pestaña "Robots" y arrastrar el bloque "fijar salida pin digital 13 a ALTO".
- Pinchar en la pestaña "Control" y arrastrar el bloque "esperar 1 segundo".
- Pinchar en la pestaña "Robots" y arrastrar el bloque "fijar salida pin digital 13 a BAJO".



- Pinchar en la bandera verde que aparece a la izquierda, en la zona del escenario.
- Comprobar que el diodo LED verde se enciende y apaga 1 vez.

## D. PRÁCTICAS

### 1. LUZ INTERMITENTE

#### Introducción

Sencillo proyecto consistente en un LED que parpadea continuamente.

Para ello debemos conectar un LED al pin 12 de la tarjeta controladora.

El LED es un operador con una determinada polaridad que debemos respetar. Así pues, conectaremos el LED siguiendo estas indicaciones:

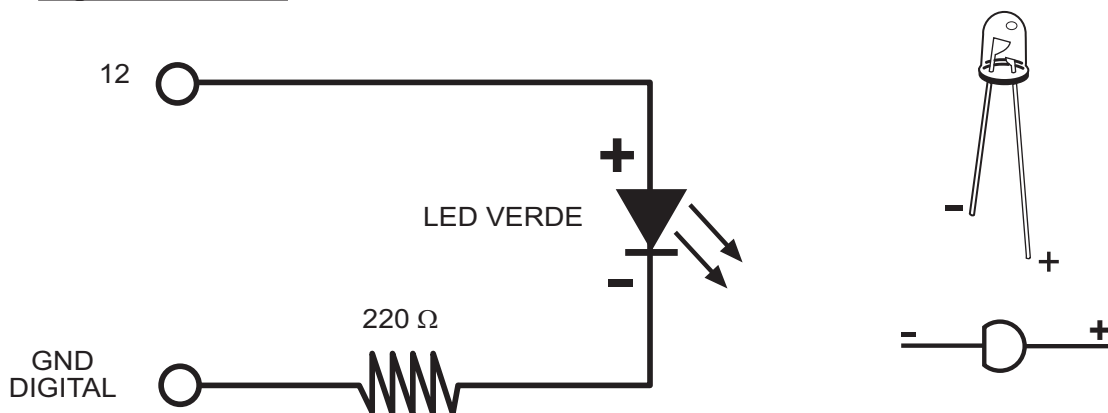
- La pata del LED larga (+) al pin 12 de la placa
- La pata del LED corta (-) se conecta a la resistencia
- La resistencia se conecta al pin GND de la placa

*Nota: El pin 13 del Arduino incluye una resistencia, por lo que se podría conectar directamente el diodo LED sin incluir la resistencia.*

#### Lista de materiales

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Led verde LOG 722
- 1 Resistencia de 220 ohmios (rojo-rojo-marrón) LOG 748 220
- 2 Latiguillos board macho-macho LOG 9519

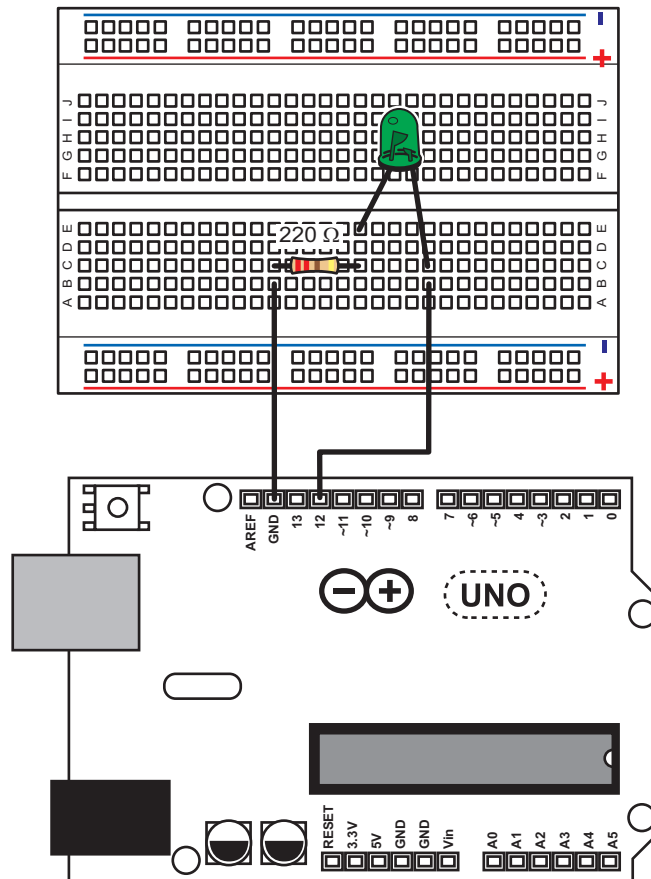
#### Esquema Eléctrico



## D. PRÁCTICAS

### Montaje en placa board

- Pin GND a la resistencia
- Pin 12 al diodo LED



### Esquema del programa

```

Inicio de programa
  Bucle infinito
    activa salida 12 (enciende el LED)
    pausa
    desactiva salida 12 (apaga el LED)
    pausa
  Fin del bucle
fin de programa
  
```

## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



### Programa con processing

```

int LED = 12;

void setup() {
  pinMode(LED, OUTPUT); //Se configura el pin 12 como salida
}

void loop() {
  digitalWrite(LED, HIGH);    // Se enciende el LED
  delay(1000);               // Se espera unos segundos
  digitalWrite(LED, LOW);    // Se apaga el LED
  delay(1000);               // Se espera unos segundos
}

```

## **D. PRÁCTICAS**

### **2. AVISO PARA INVIDENTES**

#### **Introducción**

El circuito consiste en conectar un zumbador al pin 12 de la tarjeta controladora.

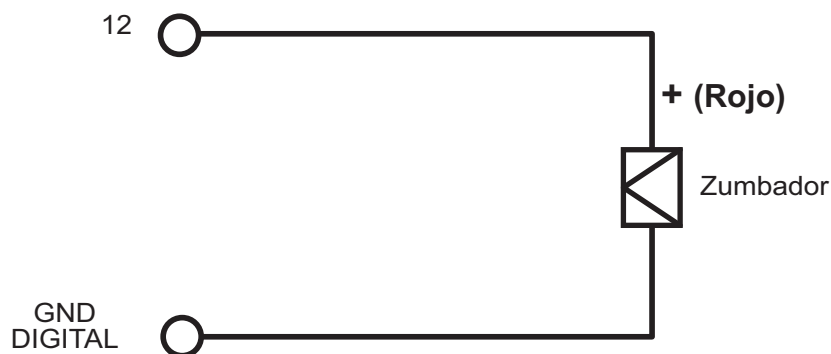
El zumbador es un operador con una determinada polaridad que debemos respetar. Así pues, conectaremos el zumbador siguiendo estas indicaciones:

- La pata larga del zumbador (+) al pin 12 de la placa
- La pata corta del zumbador (-) al pin GND de la placa

#### **Lista de materiales**

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Zumbador activo LOG 7714
- 2 Latiguillos board macho-macho LOG 9519

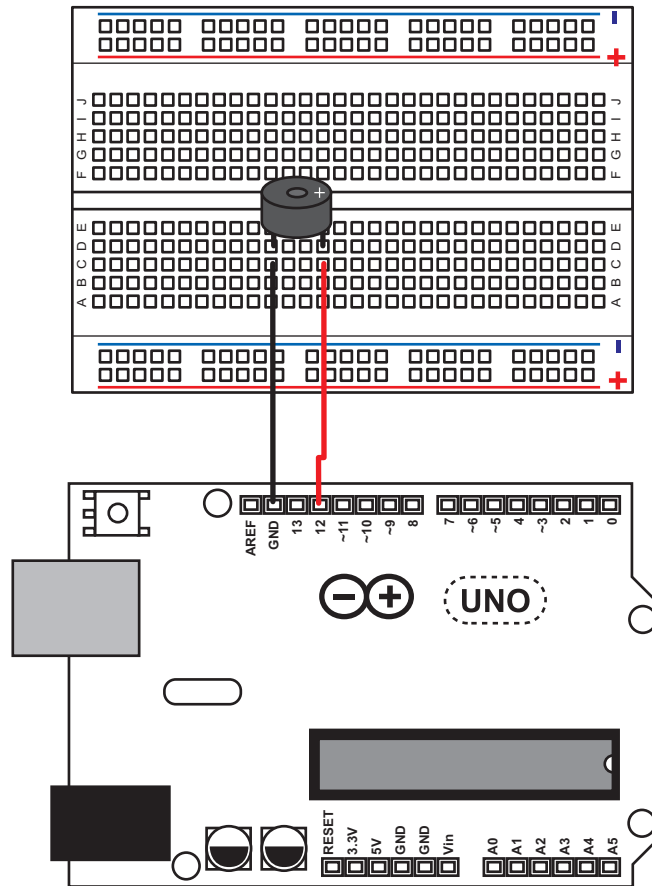
#### **Esquema Eléctrico**



## D. PRÁCTICAS

### Montaje en placa board

- Pin GND a la pata corta
- Pin 12 a la pata larga +



### Esquema del programa

```

Inicio de programa
  Bucle infinito
    activa salida 12 (enciende el Zumbador)
    pausa
    desactiva salida 12 (apaga el Zumbador)
    pausa
  Fin del bucle
fin de programa
  
```

## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



### Programa con processing

```

int Zumbador = 12;

void setup() {
  pinMode(Zumbador, OUTPUT); //Se configura el pin 12 como salida
}

void loop() {
  digitalWrite(Zumbador, HIGH);    // Se enciende el Zumbador
  delay(1000);                     // Se espera unos segundos
  digitalWrite(Zumbador, LOW);     // Se apaga el Zumbador
  delay(1000);                     // Se espera unos segundos
}

```

## D. PRÁCTICAS

### 3. SEMÁFORO

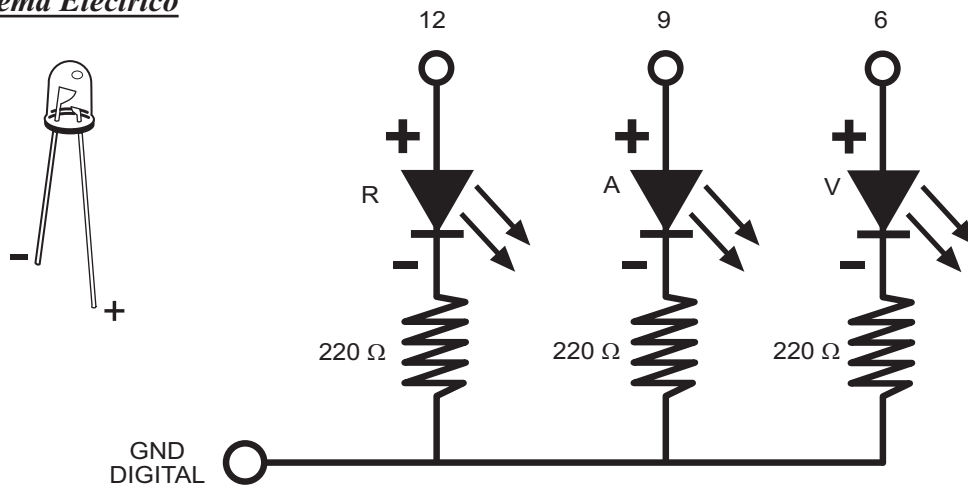
#### Introducción

Construcción de un semáforo de 3 colores para vehículos con tres diodos LED (verde, amarillo y rojo).

#### Lista de materiales

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Led rojo LOG 724
- 1 Led amarillo LOG 723
- 1 Led verde LOG 722
- 3 Resistencias de 220 ohmios (rojo-rojo-marrón) LOG 748 220
- 4 Latiguillos board macho-macho LOG 9519

#### Esquema Eléctrico

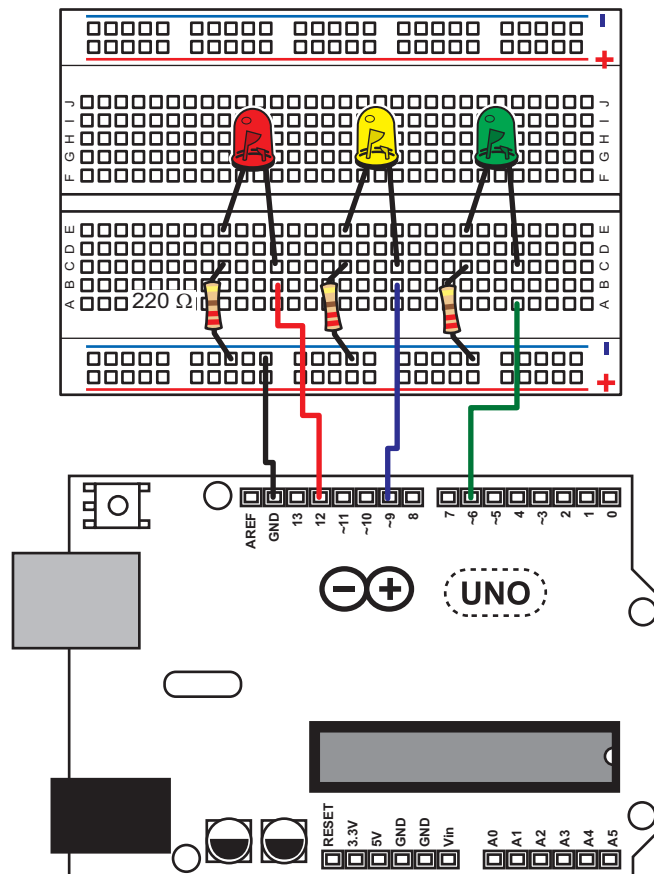




## D. PRÁCTICAS

### Montaje en placa board

- Pin 6 al diodo LED verde
- Pin 9 al diodo LED amarillo
- Pin 12 al diodo LED rojo
- Pin GND a la regleta de conexiones
- De la regleta de conexiones a las resistencias



### Esquema del programa

```

Inicio de programa
  bucle infinito
    Activamos la salida 12    (enciende el LED rojo)
    Desactivamos la salida 9  (apaga el LED amarillo)
    Desactivamos la salida 6  (apaga el LED verde)
    pausa
    Desactivamos la salida 12 (apaga el LED rojo)
    Desactivamos la salida 9  (apaga el LED amarillo)
    Activamos la salida 6     (enciende el LED amarillo)
    pausa
    Desactivamos la salida 12 (apaga el LED rojo)
    Activamos la salida 9     (enciende el LED verde)
    Desactivamos la salida 6  (apaga el LED verde)
    pausa
  fin del bucle
fin
  
```

## D. PRÁCTICAS

### Programa con Scratch



### Programa con processing

```

int LEDR = 12;
int LEDA = 9;
int LEDV = 6;

void setup() {
  pinMode(LEDR, OUTPUT);
  pinMode(LEDA, OUTPUT);
  pinMode(LEDV, OUTPUT);
}

void loop() {
  digitalWrite(LEDR, HIGH); // Semáforo en rojo
  digitalWrite(LEDA, LOW);
  digitalWrite(LEDV, LOW);
  delay(4000);
  digitalWrite(LEDR, LOW); // semáforo en verde
  digitalWrite(LEDA, LOW);
  digitalWrite(LEDV, HIGH);
  delay(4000);
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDA, HIGH);
  digitalWrite(LEDV, LOW); // semáforo en amarillo
  delay(800);
}

```

## D. PRÁCTICAS

### Programa con Makeblock



## D. PRÁCTICAS

### 4. SEMÁFORO CON AVISO PARA INIDENTES

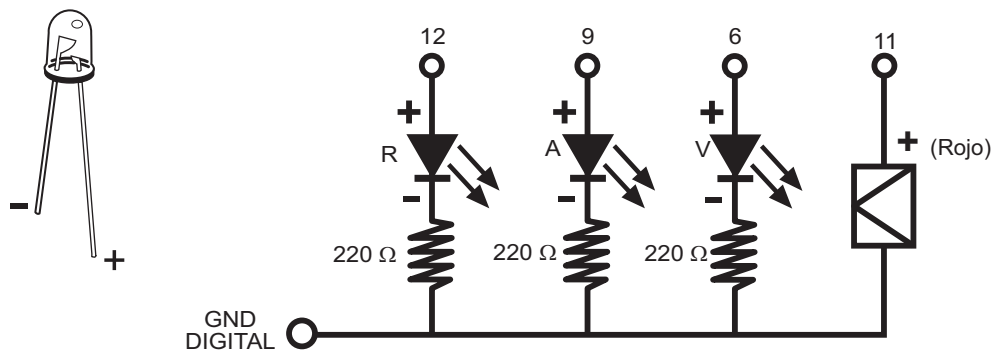
#### Introducción

Construcción de un semáforo de 3 colores para vehículos, con una señal acústica que avisa cuando el semáforo está rojo, para que los peatones invidentes puedan cruzar la calzada.

#### Lista de materiales

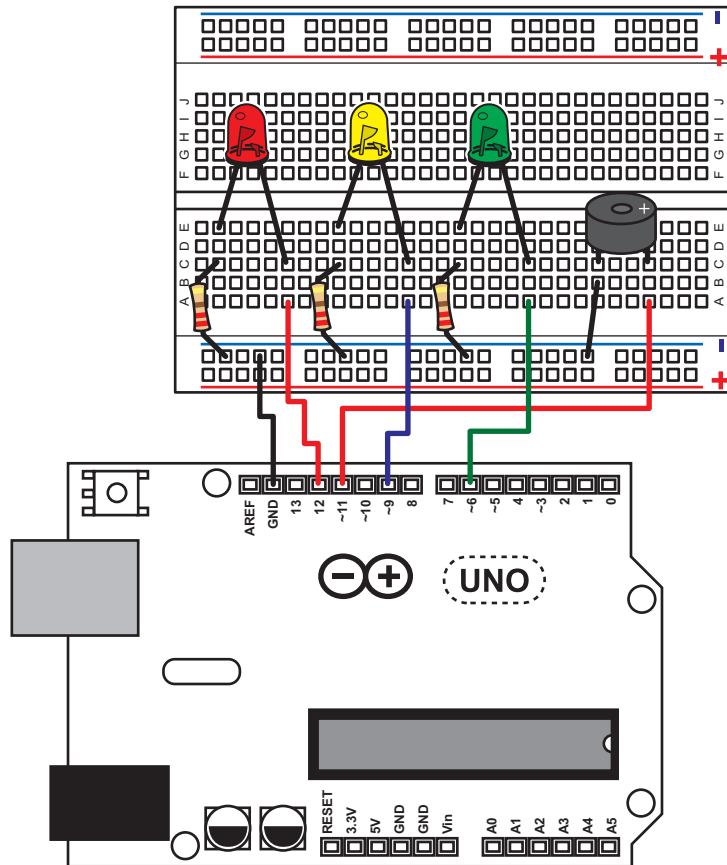
- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Led rojo LOG 724
- 1 Led amarillo LOG 723
- 1 Led verde LOG 722
- 3 Resistencias de 220 ohmios (rojo-rojo-marrón) LOG 748 220
- 1 Zumbador activo LOG 7714
- 6 Latiguillos board macho-macho LOG 9519

#### Esquema Eléctrico



## D. PRÁCTICAS

### Montaje con placa board



### Esquema del programa

Inicio de programa  
bucle infinito

|                        |                            |
|------------------------|----------------------------|
| Activa la salida 12    | (enciende el LED rojo)     |
| Activa la salida 11    | (enciende el zumbador)     |
| Desactiva la salida 9  | (apaga el LED amarillo)    |
| Desactiva la salida 6  | (apaga el LED verde)       |
| Espera 1 minuto        |                            |
| Desactiva la salida 12 | (apaga el LED rojo)        |
| Desactiva la salida 11 | (apaga el zumbador)        |
| Desactiva la salida 9  | (apaga el LED amarillo)    |
| Activa la salida 6     | (enciende el LED verde)    |
| Espera 1 minuto        |                            |
| Desactiva la salida 12 | (apaga el LED rojo)        |
| Desactiva la salida 11 | (apaga el zumbador)        |
| Activa la salida 9     | (enciende el LED amarillo) |
| Desactiva la salida 6  | (apaga el LED verde)       |
| Espera 10 segundos     |                            |

fin del bucle  
fin

## D. PRÁCTICAS

### Programa con Scratch



## D. PRÁCTICAS

### Programa con Makeblock



## D. PRÁCTICAS

### Programa con processing

```
int LEDR = 12;
int LEDA = 9;
int LEDV = 6;
int zumbador = 11;

void setup() {
  pinMode(LEDR, OUTPUT);
  pinMode(LEDA, OUTPUT);
  pinMode(LEDV, OUTPUT);
  pinMode(zumbador, OUTPUT);
}

void loop() {
  digitalWrite(LEDR, HIGH);      // Semáforo en rojo
  digitalWrite(zumbador, HIGH); // Aviso a invidentes
  digitalWrite(LEDA, LOW);
  digitalWrite(LEDV, LOW);
  delay(4000);
  digitalWrite(LEDR, LOW);      // Semáforo verde
  digitalWrite(zumbador, LOW);  // Aviso a invidentes
  digitalWrite(LEDA, LOW);
  digitalWrite(LEDV, HIGH);
  delay(4000);
  digitalWrite(LEDR, LOW);      // Semáforo ámbar
  digitalWrite(zumbador, LOW);
  digitalWrite(LEDA, HIGH);
  digitalWrite(LEDV, LOW);
  delay(800);
}
```



## D. PRÁCTICAS

### 5. SEMÁFORO CON PULSADOR

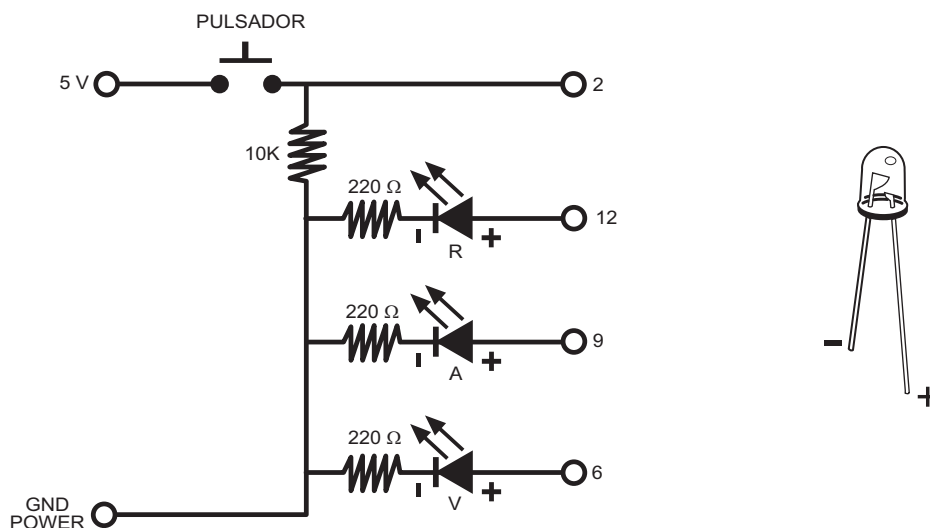
#### Introducción

Construcción de un semáforo de 3 colores para vehículos que siempre está en verde a no ser que un peatón pulse un botón, en cuyo caso realiza el ciclo ámbar-rojo para que el peatón pueda cruzar la calzada.

#### Lista de materiales

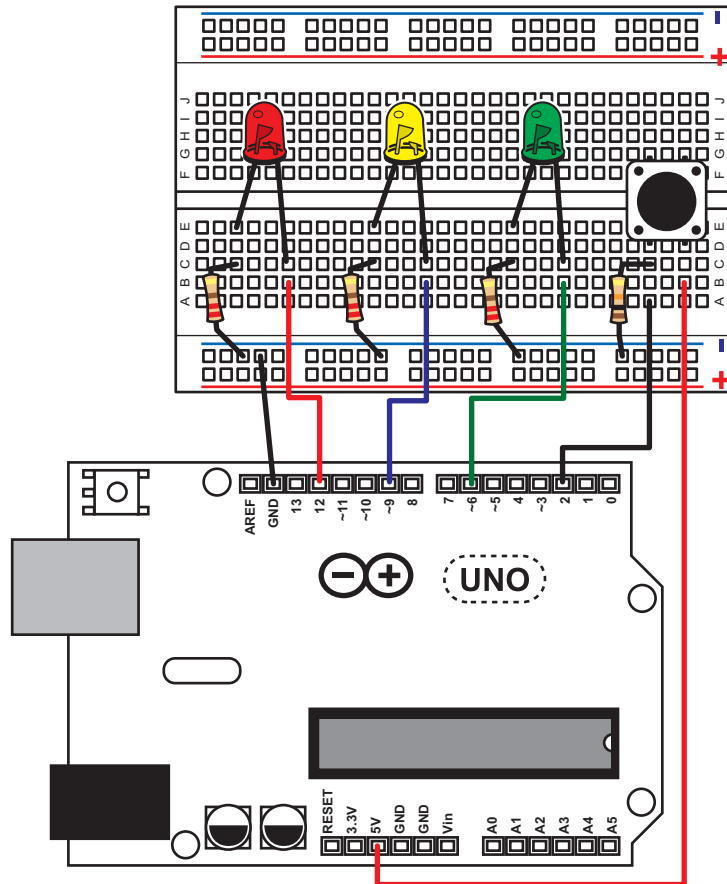
- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Led rojo LOG 724
- 1 Led amarillo LOG 723
- 1 Led verde LOG 722
- 3 Resistencias de 220 ohmios (rojo-rojo-marrón) LOG 748 220
- 1 Resistencia de 10K ohmios (marrón-negro-naranja) LOG 748 10K
- 1 Pulsador LOG 542
- 6 Latiguillos board macho-macho LOG 9519

#### Esquema Eléctrico



## D. PRÁCTICAS

### Montaje con placa board



### Esquema del programa

```

Inicio de programa
  bucle infinito
    Activa la salida 6           (enciende el LED verde)
    Si esta activa la salida 2   (si pulsamos el pulsador)
      desactiva la salida 6     (apaga el LED verde)
      activa la salida 9        (enciende el LED amarillo)
      espera 10 segundos
      desactiva la salida 9     (apaga el LED amarillo)
      activa la salida 12       (enciende el LED rojo)
      espera 1 minuto
      desactiva la salida 12    (apaga el LED rojo)
    Fin del si
  fin del bucle
fin
  
```

## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



## D. PRÁCTICAS

### Programa con processing

```
int LEDR=12;
int LEDA=9;
int LEDV=6;
int pulsador=2;

void setup()
{
  pinMode(LEDV,OUTPUT);
  pinMode(LEDV,OUTPUT);
  pinMode(LEDA,OUTPUT);
  pinMode(pulsador,INPUT);
}

void loop()
{
  digitalWrite(LEDV,HIGH);          //LED verde encendido
  if (digitalRead(pulsador) == HIGH) //lee pulsador
  {
    digitalWrite(LEDV,LOW); //LED verde apagado
    digitalWrite(LEDA,HIGH); //LED amarillo encendido
    delay(400);
    digitalWrite(LEDA,LOW); //LED amarillo apagado
    digitalWrite(LEDV,HIGH); //LED rojo encendido
    delay(4000);
    digitalWrite(LEDV,LOW); //LED rojo apagado
  }
}
```

## D. PRÁCTICAS

### 6. CRUCE DE SEMÁFOROS

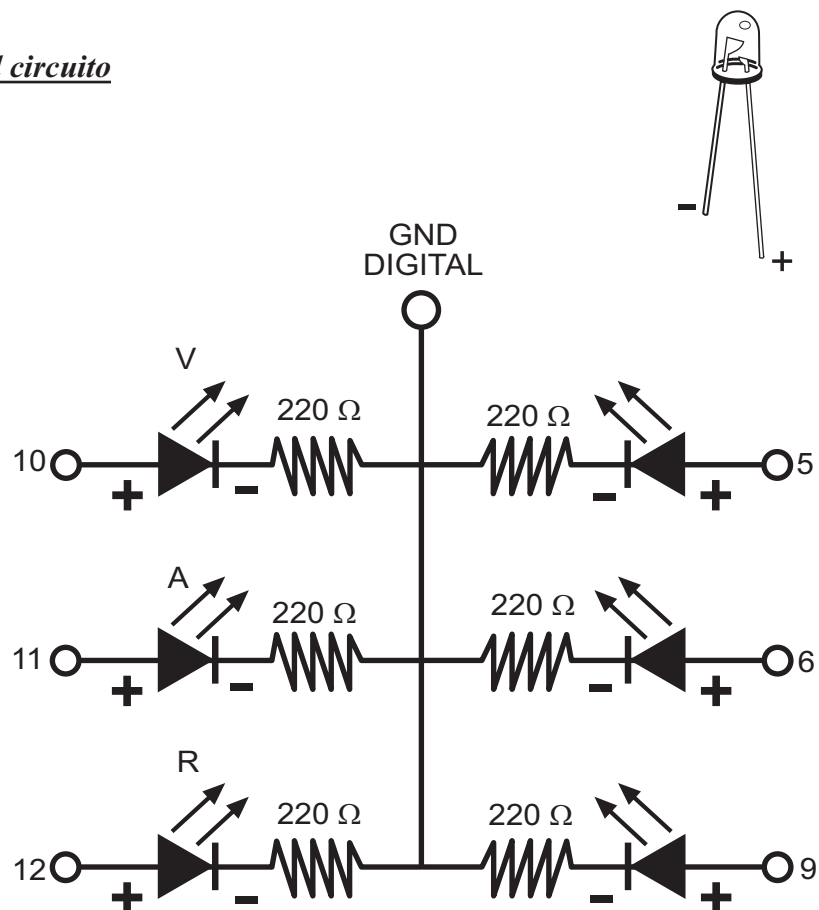
#### Introducción

Simulación de un cruce de semáforos de 3 colores para vehículos.

#### Lista de materiales

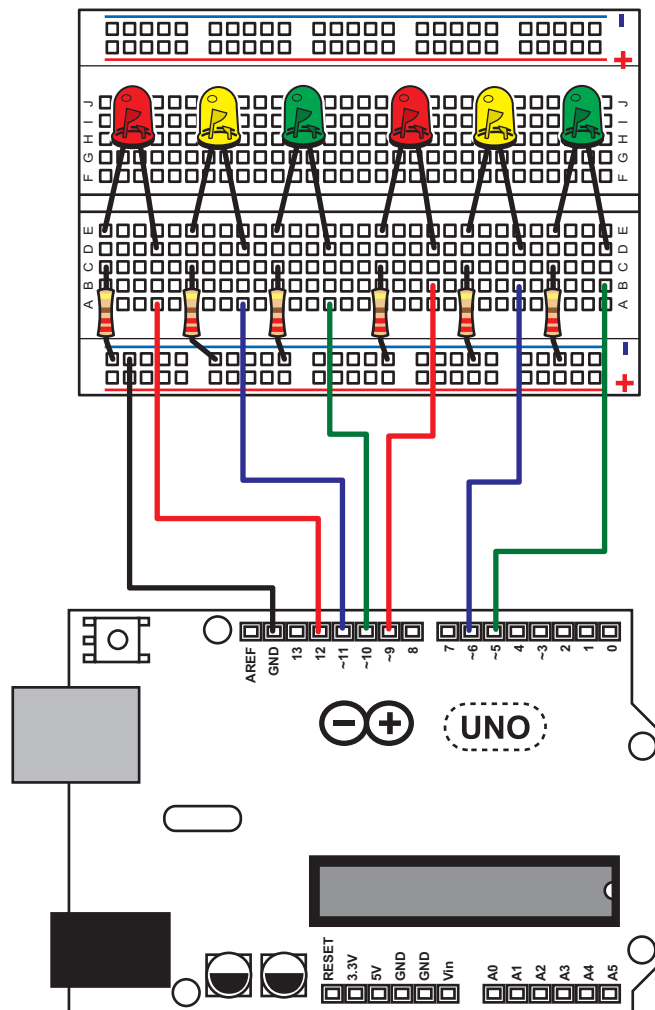
- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 6 Resistencias de 220 ohmios (rojo-rojo-marrón) LOG 748 220
- 2 Led rojo LOG 724
- 2 Led amarillo LOG 723
- 2 Led verde LOG 722
- 7 Latiguillos board macho-macho LOG 9519

#### Esquema del circuito



## D. PRÁCTICAS

### Montaje en placa board



## D. PRÁCTICAS

### Programa con Scratch

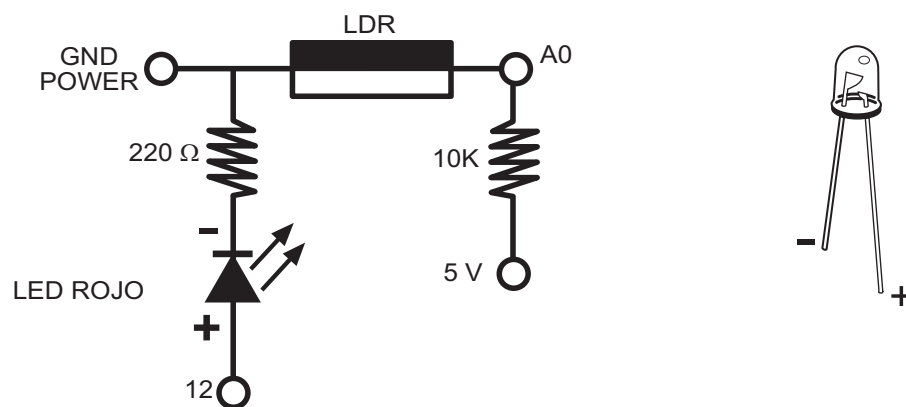


## D. PRÁCTICAS

### Programa con Makeblock

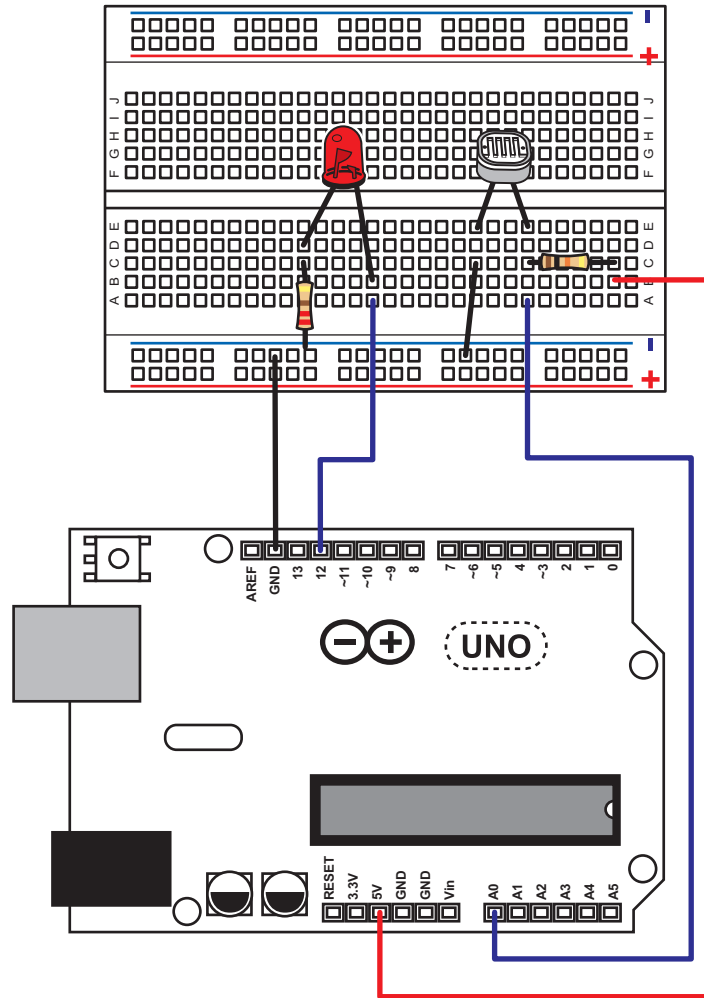






## D. PRÁCTICAS

### Montaje en placa board



### Esquema del programa

Inicio de programa

bucle infinito

Si valor de entrada analogica A0 < 200 (si el valor de la LDR < 200)  
Activa la salida 12 (enciende el LED rojo)

Si no

Desactiva la salida 12 (apaga el LED rojo)

fin del bucle

fin

## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



## **D. PRÁCTICAS**

### **Programa con processing**

```
int sensorPin = A0;
int LEDPin = 12;
int sensorValue = 0;

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

void loop() {
  sensorValue = analogRead(sensorPin);
  if (sensorValue > 200){           // Si LDR < 200
    digitalWrite(LEDPin, LOW);} // Enciende el LED
  else{
    digitalWrite(LEDPin, HIGH);}  // Apaga el LED
}
```

## **D. PRÁCTICAS**

### **8. CADENA DE MONTAJE**

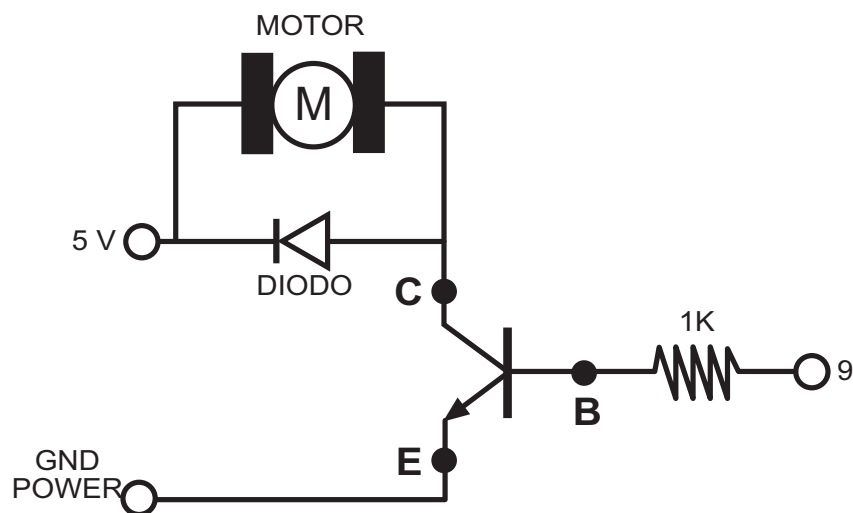
#### **Introducción**

Construcción consistente en un motor que realiza el movimiento de avance y pausa alterno de una cadena de montaje.

#### **Lista de materiales**

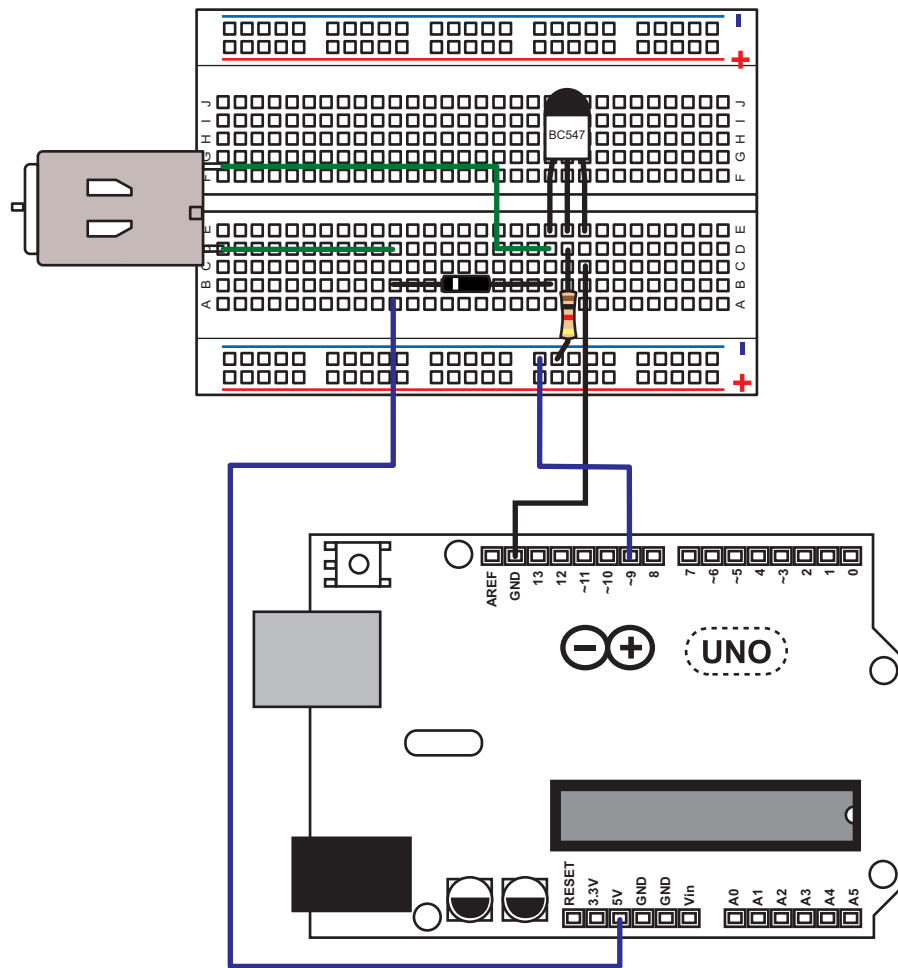
- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Motor LOG 00
- 1 Resistencia de 1K ohmios (marrón-negro-rojo) LOG 748 1K
- 1 Diodo LOG 760
- 1 Transistor LOG 751
- 5 Latiguillos board macho-macho LOG 9519

#### **Esquema del circuito**



## D. PRÁCTICAS

### Montaje en placa board



### Esquema del programa

```

Inicio de programa
  bucle infinito
    Desactiva la salida 9 // Para el motor
    Espera
    Activa la salida 9    // Acciona el motor
    Espera
  fin del bucle
fin
  
```

## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



### Programa con processing

```

int LEDPin = 9;

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

void loop()
{
  digitalWrite(LEDPin, LOW); // Para el motor
  delay(100);
  digitalWrite(LEDPin, HIGH); // Activa el motor
  delay(100);
}

```

## D. PRÁCTICAS

### 9. VENTILADOR AUTOMÁTICO

#### Introducción

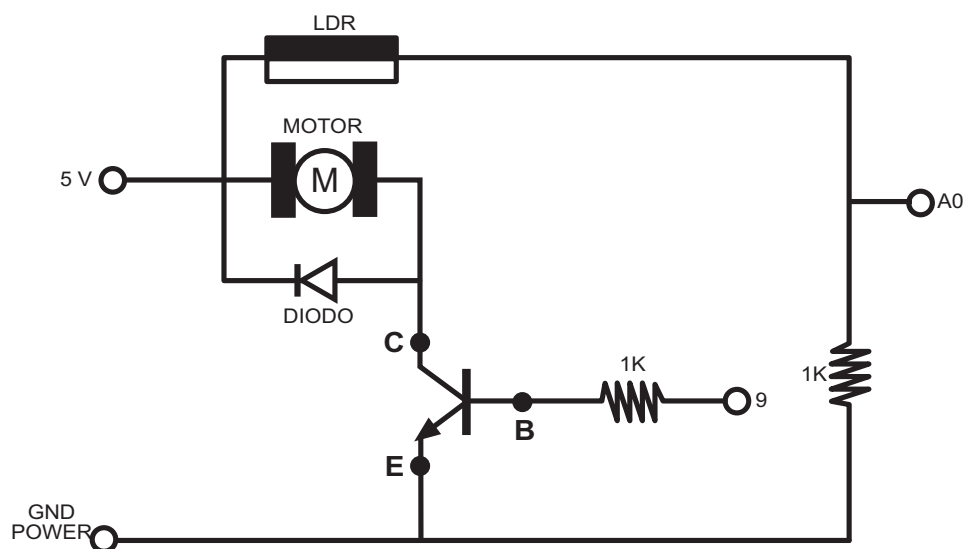
Construcción consistente en un motor que se activa ante la falta de luz.

Utilizamos una LDR para realizar la medición de luz.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 LDR LOG 731
- 2 Resistencias de 1K (marrón-negro-rojo) LOG 748 1K
- 1 Diodo LOG 760
- 1 Transistor LOG 751
- 1 Motor LOG 00
- 1 Hélice LOG 22
- 7 Latiguillos board macho-macho LOG 9519

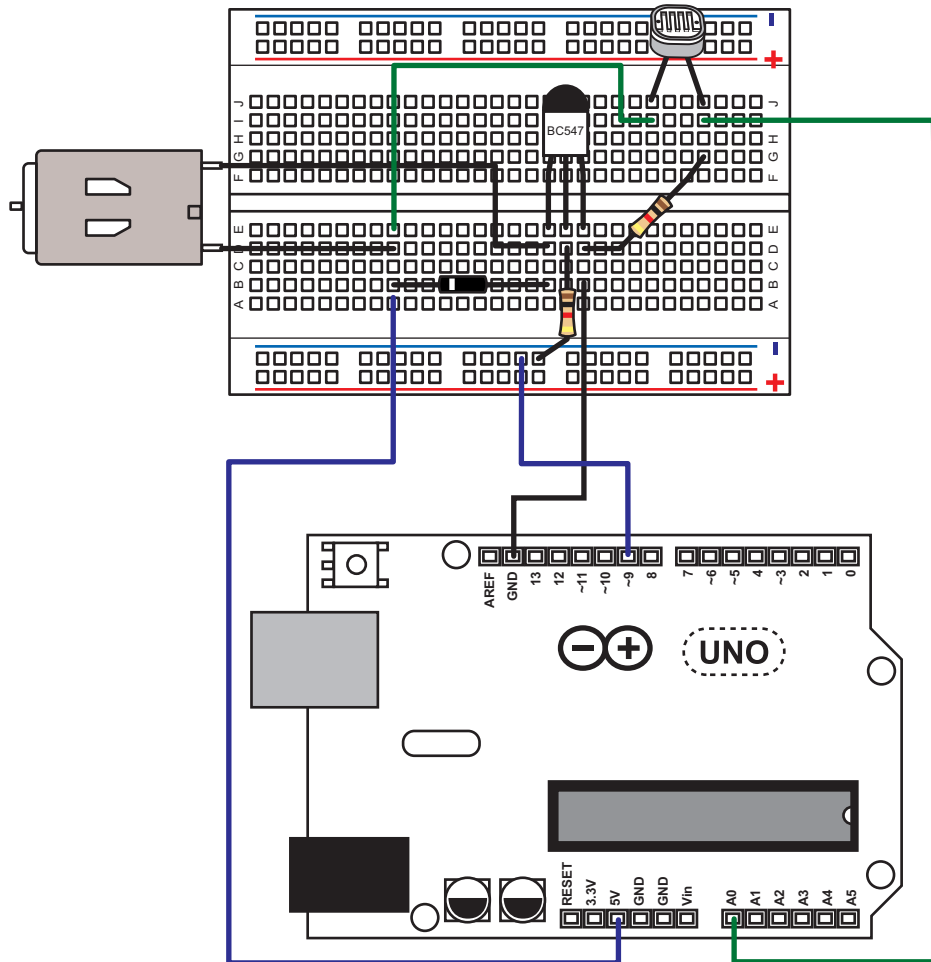
#### Esquema eléctrico





## D. PRÁCTICAS

### Montaje en placa board



### Esquema del programa

```

Inicio de programa
  bucle infinito
    Si valor de entrada analogica > 200 (Si el valor de LDR >200)
      Desactiva la salida 9      (Para motor)
    Si no
      Activa la salida 9          (Activa motor)
  fin del bucle
fin
  
```

## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



### Programa con processing

```
int sensorPin = A0;
int LEDPin = 9;
int sensorValue = 0;

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

void loop() {
  sensorValue = analogRead(sensorPin); // Si LDR > 200
  if (sensorValue > 200){
    digitalWrite(LEDPin, LOW); // Apaga el motor
  }
  else{
    digitalWrite(LEDPin, HIGH); // Activa motor
  }
}
```

## D. PRÁCTICAS

### 10.1 LUZ ROJA, VERDE Y AZUL (RGB Ánodo Común)

#### Introducción

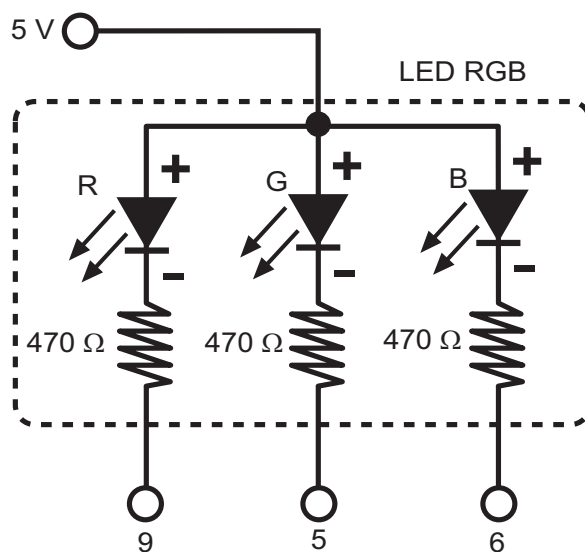
Un LED RGB permite visualizar una luz de diferente color en función de la alimentación de cada una de sus patas.

En esta práctica vamos a visualizar los colores rojo, verde y azul, alimentando para cada color, una sola parte gracias al uso de pins PWM de la placa Arduino.

#### Lista de materiales

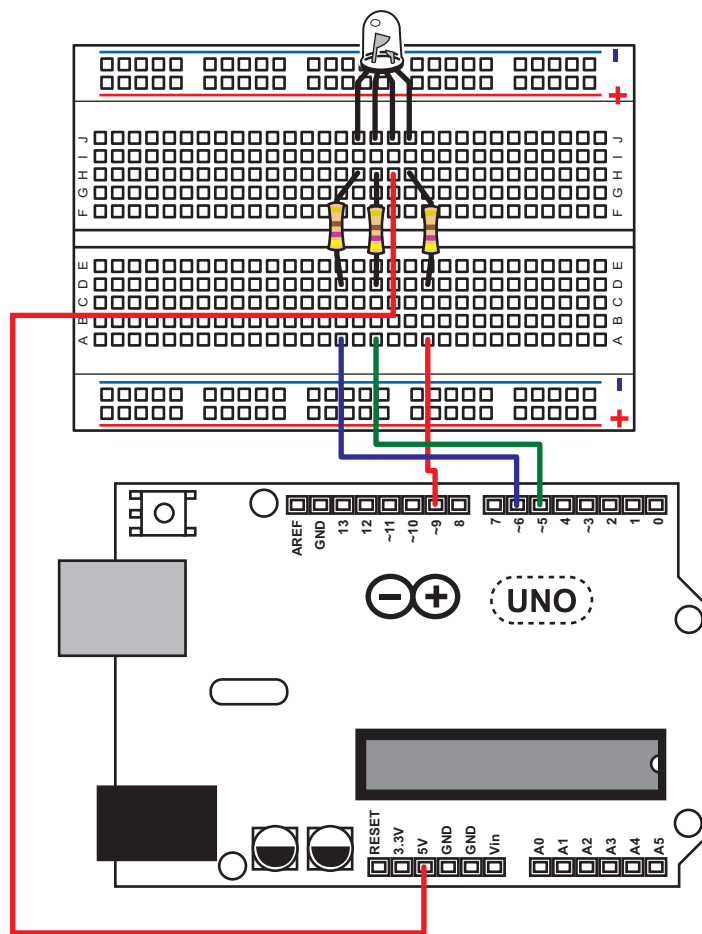
- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 3 Resistencias de 470 ohm (amarillo-morado-marrón) LOG 748 470
- 1 Led RGB Ánodo Común LOG 730
- 4 Latiguillos board macho-macho LOG 9519

#### Esquema del circuito



## D. PRÁCTICAS

### Montaje en placa board



### Esquema del programa

Inicio de programa

Bucle infinito

Activa la salida 5 a 0

Activa la salida 6 a 255

Activa la salida 9 a 255

Espera

Activa la salida 5 a 255

Activa la salida 6 a 0

Activa la salida 9 a 255

Espera

Activa la salida 5 a 255

Activa la salida 6 a 255

Activa la salida 9 a 0

Espera

Fin del bucle

## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



## **D. PRÁCTICAS**

### **Programa con processing**

```
int LED1 = 5;  
int LED2 = 6;  
int LED3 = 9;  
int i=0;
```

```
void setup() {  
  pinMode(LED1, OUTPUT);  
  pinMode(LED2, OUTPUT);  
  pinMode(LED3, OUTPUT);  
}
```

```
void loop() {  
  analogWrite(LED1,0);  
  analogWrite(LED2,255);  
  analogWrite(LED3,255);  
  delay(1000);  
  analogWrite(LED1,255);  
  analogWrite(LED2,0);  
  analogWrite(LED3,255);  
  delay(1000);  
  analogWrite(LED1,255);  
  analogWrite(LED2,255);  
  analogWrite(LED3,0);  
  delay(1000);  
}
```

## D. PRÁCTICAS

### 10.2 LUZ ROJA, VERDE Y AZUL (RGB Cátodo Común)

#### Introducción

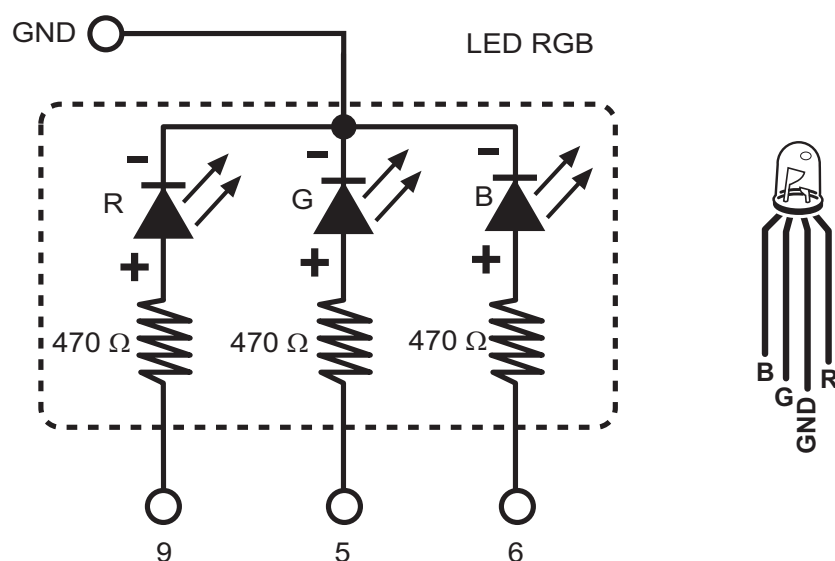
Un LED RGB permite visualizar una luz de diferente color en función de la alimentación de cada una de sus patas.

En esta práctica vamos a visualizar los colores rojo, verde y azul, alimentando para cada color, una sola parte gracias al uso de pins PWM de la placa Arduino.

#### Lista de materiales

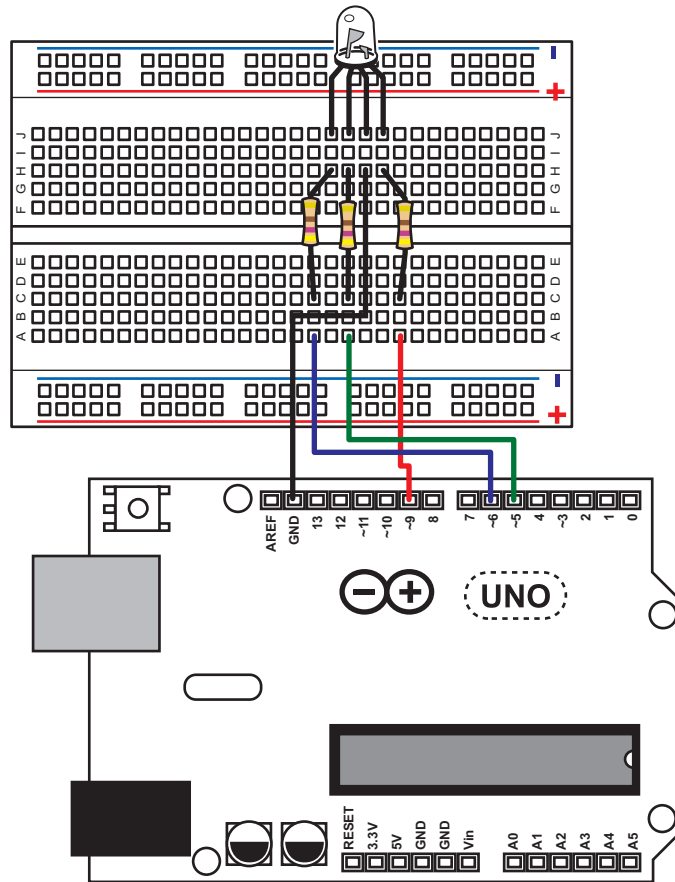
- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 3 Resistencias de 470 ohm (amarillo-morado-marrón) LOG 748 470
- 1 LED RGB Cátodo Común LOG 729
- 4 Latiguillos board macho-macho LOG 9519

#### Esquema del circuito



## D. PRÁCTICAS

### Montaje en placa board



### Esquema del programa

Inicio de programa

Bucle infinito

Activa la salida 5 a 255

Activa la salida 6 a 0

Activa la salida 9 a 0

Espera

Activa la salida 5 a 0

Activa la salida 6 a 255

Activa la salida 9 a 0

Espera

Activa la salida 5 a 0

Activa la salida 6 a 0

Activa la salida 9 a 255

Espera

Fin del bucle

Fin



## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



## D. PRÁCTICAS

### Programa con processing

```
int LED1 = 5;  
int LED2 = 6;  
int LED3 = 9;  
int i=0;
```

```
void setup() {  
  pinMode(LED1, OUTPUT);  
  pinMode(LED2, OUTPUT);  
  pinMode(LED3, OUTPUT);  
}
```

```
void loop() {  
  analogWrite(LED1,255);  
  analogWrite(LED2,0);  
  analogWrite(LED3,0);  
  Delay (1000);  
  analogWrite(LED1,0);  
  analogWrite(LED2,255);  
  analogWrite(LED3,0);  
  Delay (1000);  
  analogWrite(LED1,0);  
  analogWrite(LED2,0);  
  analogWrite(LED3,255);  
  Delay (1000);  
}
```

## D. PRÁCTICAS

### 11.1 ARCOIRIS DE LUZ (RGB Ánodo Común)

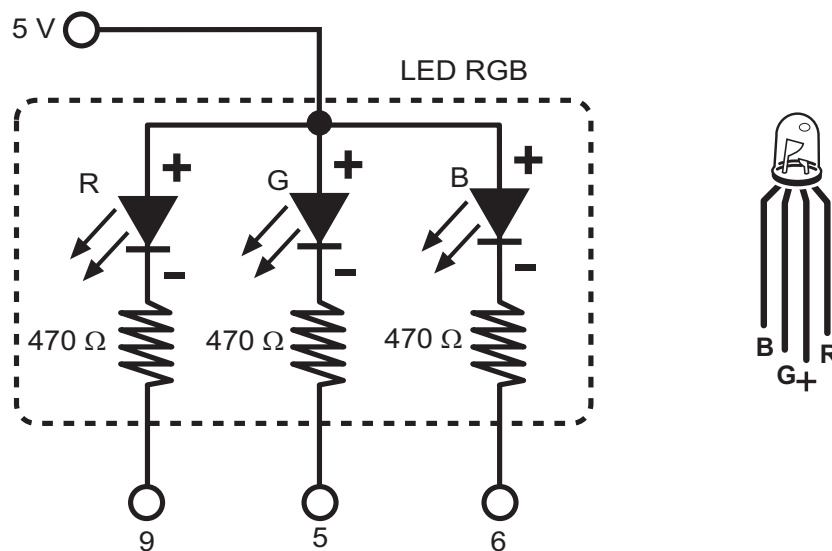
#### Introducción

En esta práctica el LED nos mostrará toda la gama de colores que sus características le permitan. Para ello empleamos los pines PWM de la placa Arduino y realizamos diferentes combinaciones de tensión en cada una de las patas del LED RGB.

#### Lista de materiales

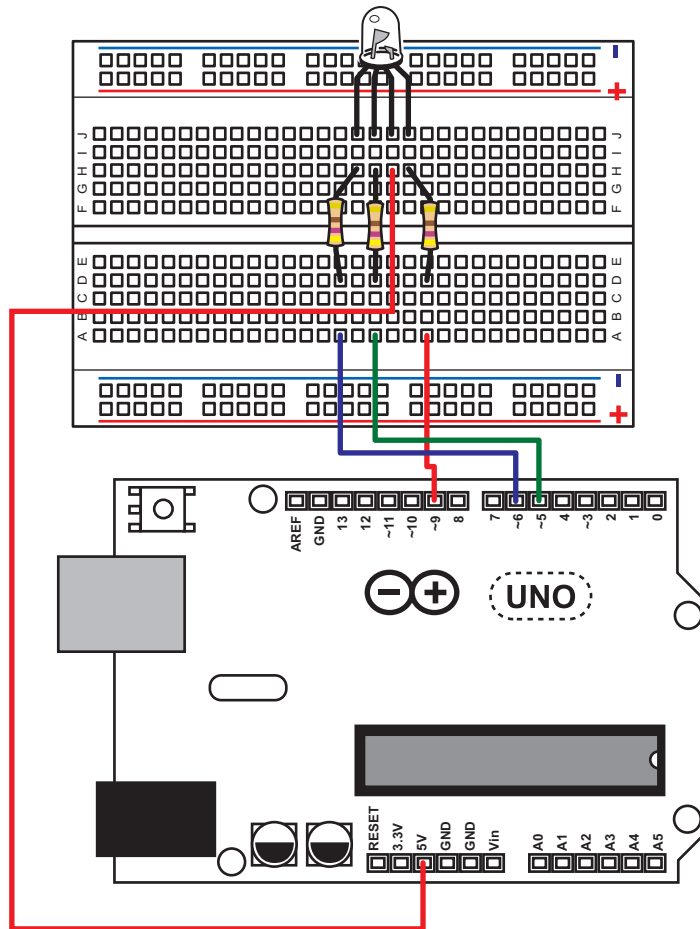
- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 3 Resistencias de 470 ohm (amarillo-morado-marrón) LOG 748 470
- 1 LED RGB Ánodo Común LOG 730
- 4 Latiguillos board macho-macho LOG 9519

#### Esquema del circuito



## D. PRÁCTICAS

### Montaje en placa board



### Esquema del programa

Inicio del programa

Bucle infinito

Salida 5 varía de 255 a 0

Salida 6 varía de 0 a 255

Salida 5 varía de 0 a 255

Salida 9 varía de 255 a 0

Salida 6 varía de 255 a 0

Salida 9 varía de 0 a 255

Fin del bucle

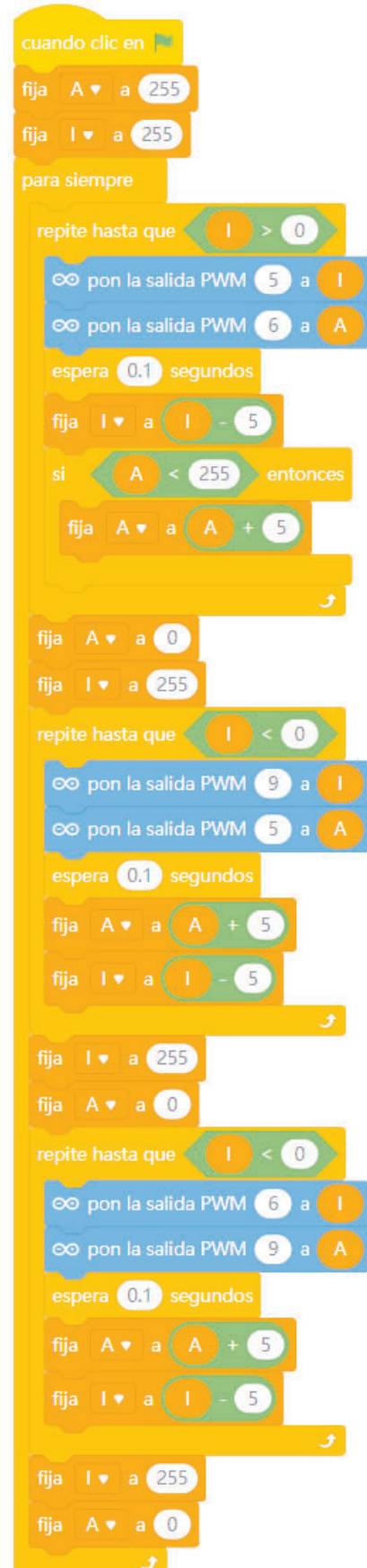
Fin

## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



## D. PRÁCTICAS

### Programa con processing

```
int i=255;
int a=255;
void setup() {
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(9, OUTPUT);
  analogWrite(5, 255);
  analogWrite(6, 255);
  analogWrite(9, 255);
}

void loop() {
  while (i<0){
    analogWrite(5, i);
    analogWrite(6, a);
    delay(100);
    i=i-5;
    if(a<255)
    { a=a+5;}
  }
  i=255;
  a=0;
  while (i>0){
    analogWrite(9, i);
    analogWrite(5, a);
    delay(100);
    a=a+5;
    i=i-5;
  }
  i=255;
  a=0;
  while (i>0){
    analogWrite(6, i);
    analogWrite(9, a);
    delay(100);
    i=i-5;
    a=a+5;
  }
  i=255;
  a=0;
}
```

## D. PRÁCTICAS

### 11.2 ARCOIRIS DE LUZ (RGB Cátodo Común)

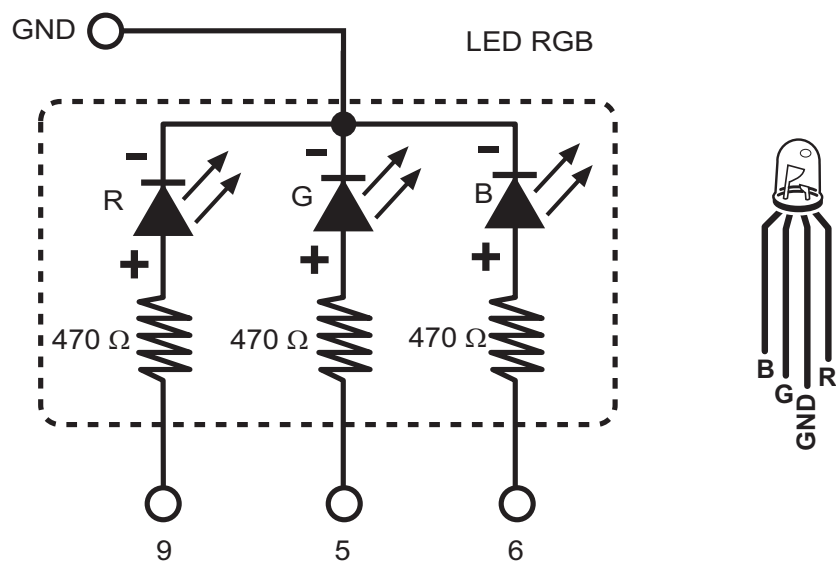
#### Introducción

En esta práctica el LED nos mostrará toda la gama de colores que sus características le permitan. Para ello empleamos los pines PWM de la placa Arduino y realizamos diferentes combinaciones de tensión en cada una de las patas del LED RGB.

#### Lista de materiales

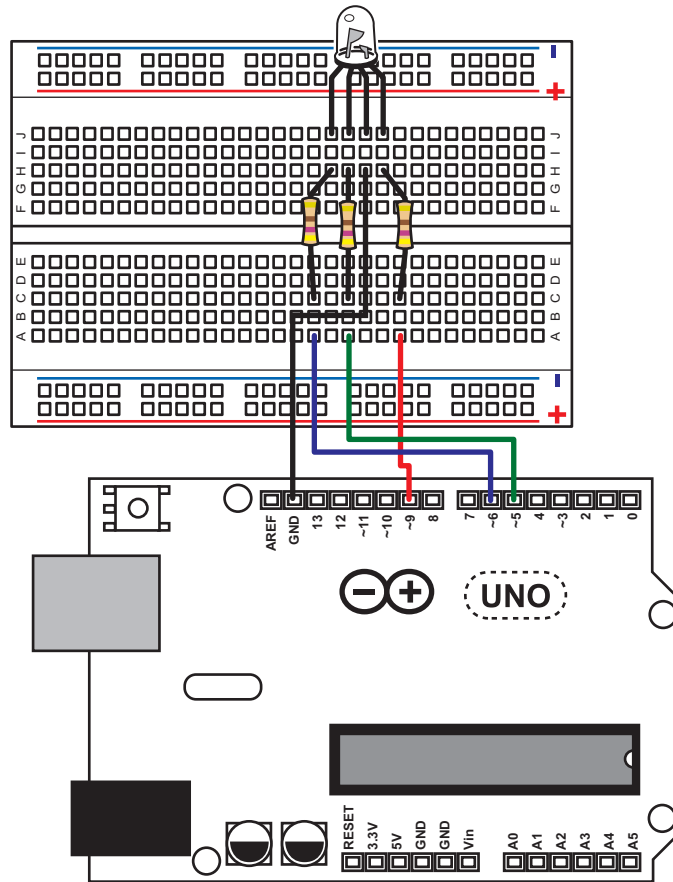
- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 3 Resistencias de 470 ohm (amarillo-morado-marrón) LOG 748 470
- 1 LED RGB Cátodo común LOG 729
- 4 Latiguillos board macho-macho LOG 9519

#### Esquema del circuito



## D. PRÁCTICAS

### Montaje en placa board



### Esquema del programa

```

Inicio del programa
  Bucle infinito
    Salida 5 varía de 0 a 255
    Salida 9 varía de 0 a 255
    Salida 5 varia de 255 a 0
    Salida 9 varía de 255 a 0
    Salida 6 varia de 0 a 255
    Salida 6 varia de 255 a 0
  Fin del bucle
Fin
  
```



## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



## **D. PRÁCTICAS**

### **Programa con processing**

```
int i=0;
void setup() {
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(9, OUTPUT);
}

void loop() {
  while (i<250){
    analogWrite(5, i);
    delay(100);
    i=i+10;
  }
  i=0;
  while (i<250){
    analogWrite(9, i);
    analogWrite(5, (250-i));
    delay(100);
    i=i+10;
  }
  i=0;
  while (i<250){
    analogWrite(6, i);
    analogWrite(9, (250-i));
    delay(100);
    i=i+10;
  }
  i=0;
  while (i<250){
    analogWrite(5, i);
    analogWrite(6, (250-i));
    delay(100);
    i=i+10;
  }
  i=0;
}
```

## D. PRÁCTICAS

### 12. DISPLAY DE 7 SEGMENTOS

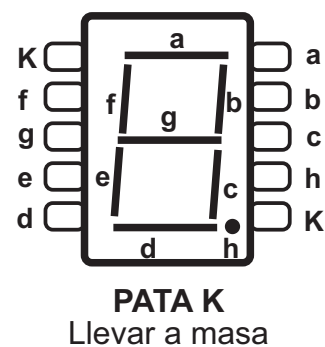
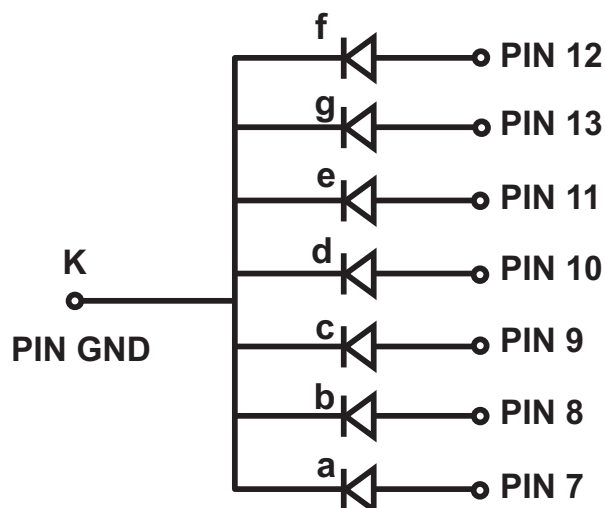
#### Introducción

En esta práctica vamos a visualizar los dígitos del 0 al 9 a través de un display de 7 segmentos.

#### Lista de materiales

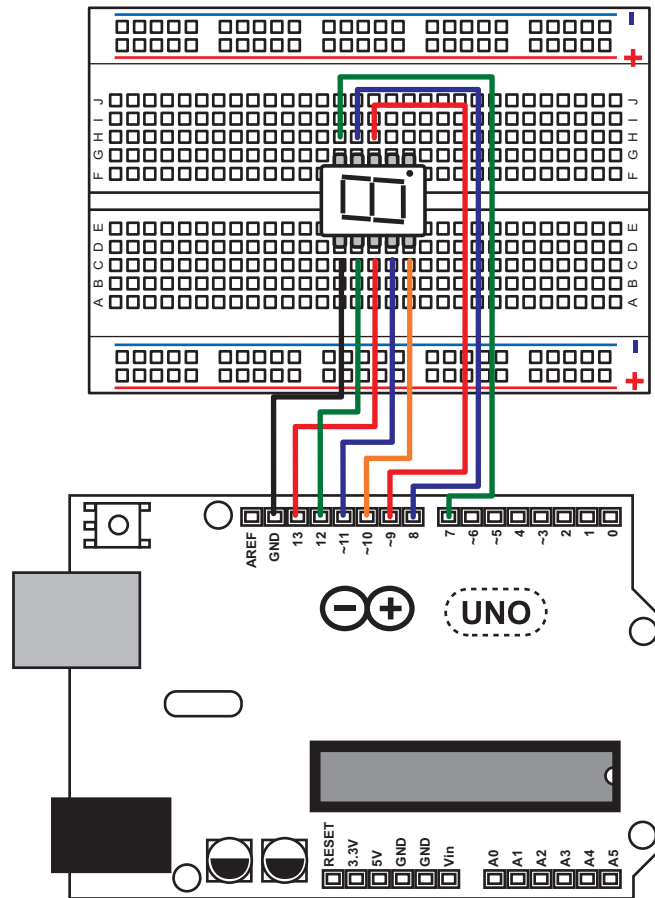
- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Display de 7 segmentos LOG 831CC
- 8 Latiguillos board macho-macho LOG 9519

#### Esquema del circuito



## D. PRÁCTICAS

### Montaje en placa board



### Esquema del programa

```

Inicio de programa
  Bucle infinito
    Visualiza 0
    pausa
    Visualiza 1
    pausa
    Visualiza 2
    pausa
    Visualiza 3
    pausa
    Visualiza 4
    pausa
    Visualiza 5
    pausa
    Visualiza 6
    pausa
    Visualiza 7
    pausa
    Visualiza 8
    pausa
    Visualiza 9
    pausa
  Fin del bucle
Fin programa
  
```

## D. PRÁCTICAS

### Funcion Display

Cada segmento ha sido asignado a un pin de la placa Arduino. La función display recibe la combinación de unos y ceros que nos indica que segmentos deben ser encendidos o apagados para visualizar el número en cuestión. El objetivo de esta función es enviar a cada pin el estado del segmento que tiene asociado para terminar generando un dígito en el display físico.

NOTA: Los pines digitales de Arduino pueden recibir las etiquetas "HIGH/LOW" o "1/0" teniendo el mismo resultado "encendido/apagado".

### Programa con processing

```
int pausa=1000;           // Variable que define el intervalo de tiempo entre cada dígito

void setup()
{
  pinMode(7, OUTPUT); // Asignación de las salidas digitales
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
}

void display (int a, int b, int c, int d, int e, int f, int g) // Funcion del display
{
  digitalWrite (7,a); //Se reciben 7 variables y se asignan a cada una de las salidas
  digitalWrite (8,b);
  digitalWrite (9,c);
  digitalWrite (10,d);
  digitalWrite (11,e);
  digitalWrite (12,f);
  digitalWrite (13,g);
}

void loop() // Dependiendo de cada dígito, se envía a la función display los estados (0 y 1) a cada uno
de los segmentos
{
  display (1,1,1,1,1,0); //escribe 0
  delay(pausa);
  display (0,1,1,0,0,0,0); //escribe 1
  delay(pausa);
  display (1,1,0,1,1,0,1); //escribe 2
  delay(pausa);
  display (1,1,1,1,0,0,1); //escribe 3
  delay(pausa);
  display (0,1,1,0,0,1,1); //escribe 4
  delay(pausa);
  display (1,0,1,1,0,1,1); //escribe 5
  delay(pausa);
  display (1,0,1,1,1,1,1); //escribe 6
  delay(pausa);
  display (1,1,1,0,0,0,0); //escribe 7
  delay(pausa);
  display (1,1,1,1,1,1,1); //escribe 8
  delay(pausa);
  display (1,1,1,0,0,1,1); //escribe 9
  delay(pausa);
}
```

| CIFRA | a | b | c | d | e | f | g |
|-------|---|---|---|---|---|---|---|
| 1     | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2     | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3     | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4     | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 5     | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6     | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7     | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8     | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9     | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0     | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

## D. PRÁCTICAS

### 13. DISPLAY DE 7 SEGMENTOS CON EL TECLADO

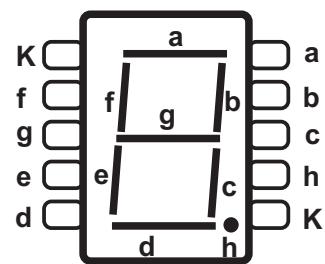
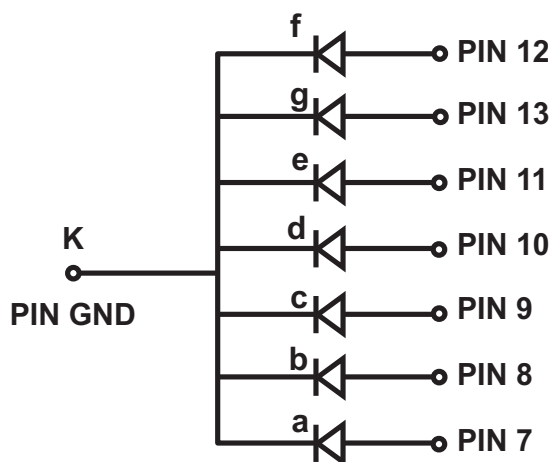
#### Introducción

En esta ocasión el display va a visualizar los dígitos que previamente se introducen por teclado (0 al 9).

#### Lista de materiales

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Display de 7 segmentos LOG 831CC
- 8 Latiguillos board macho-macho LOG 9519

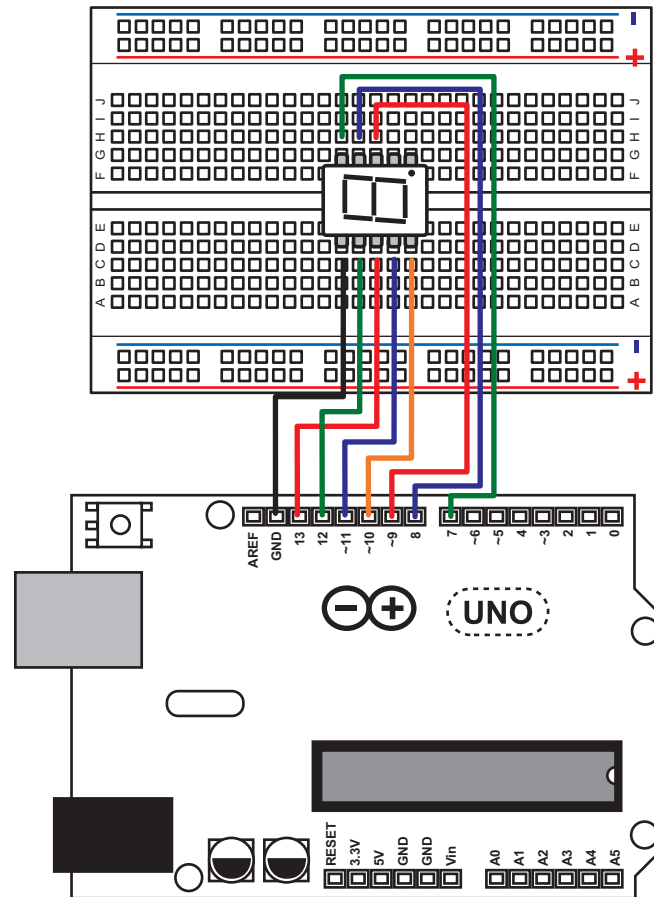
#### Esquema del circuito



**PATA K**  
Llevar a masa

## D. PRÁCTICAS

### Montaje en placa board



### Esquema del programa

```

Inicio de programa
  Bucle infinito
    Mientras dato<0 o dato>9
      pedir dato por teclado
    Visualizar dato display
  Inicializar dato
Fin programa
  
```

## **D. PRÁCTICAS**

### **Programa con processing**

```
int pausa=1000; // Variable que define el intervalo de tiempo entre cada digito
int dato=11;
void setup()
{
  pinMode(7, OUTPUT); // Asignación de las salidas digitales
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  Serial.begin(9600);
}

void display (int a, int b, int c, int d, int e, int f, int g)
// Funcion del display
{
  digitalWrite (7,a); //Se reciben 7 variables y se asignan a cada una de las salidas
  digitalWrite (8,b);
  digitalWrite (9,c);
  digitalWrite (10,d);
  digitalWrite (11,e);
  digitalWrite (12,f);
  digitalWrite (13,g);
}
```

SIGUE →



## D. PRÁCTICAS

### Programa con processing (continuación)

```

void loop() //Funcion principal
// Dependiendo de cada dígito, se envía a la función display los estados (0 y 1) a cada uno
de los segmentos
{
  while(dato<48 or dato>57){
    dato=Serial.read();
  }
  Serial.println(dato);
  Serial.println(char(dato));

  switch (char(dato)) {
    case '0':
      display (1,1,1,1,1,0); //escribe 0
      Serial.println(char(dato));
      break;
    case '1':
      display (0,1,1,0,0,0); //escribe 1
      Serial.println(char(dato));
      break;
    case '2':
      display (1,1,0,1,1,0); //escribe 2
      Serial.println(char(dato));
      break;
    case '3':
      display (1,1,1,1,0,0); //escribe 3
      break;
    case '4':
      display (0,1,1,0,0,1); //escribe 4
      break;
    case '5':
      display (1,0,1,1,0,1); //escribe 5
      break;
    case '6':
      display (1,0,1,1,1,1); //escribe 6
      break;
    case '7':
      display (1,1,1,0,0,0); //escribe 7
      break;
    case '8':
      display (1,1,1,1,1,1); //escribe 8
      break;
    case '9':
      display (1,1,1,0,0,1); //escribe 9
      break;
  }

  delay(pausa);
  display (0,0,0,0,0,0);
  dato=11;
}

```

| CIFRA | a | b | c | d | e | f | g |
|-------|---|---|---|---|---|---|---|
| 1     | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2     | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3     | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4     | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 5     | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6     | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7     | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8     | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9     | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0     | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

## D. PRÁCTICAS

### 14. DETECTOR DE OBJETOS (CNY70)

#### Introducción

Utilizamos la capacidad del octoacoplador CNY70 para diferenciar cambios de luz reflejada. Vamos a realizar una práctica donde el octoacoplador funciona como detector de presencia. Si detecta un objeto frente a él activa un LED.

#### Lista de materiales

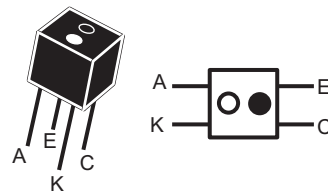
- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Resistencia de 220 ohm (rojo-rojo-marrón) LOG 748 220
- 1 Resistencia de 47K ohm (amarillo - morado - naranja) LOG 748 47K
- 1 Led verde LOG 722
- 1 Octoacoplador CNY70 LOG 837
- 5 Latiguillos board macho-macho LOG 9519

#### Esquema del circuito

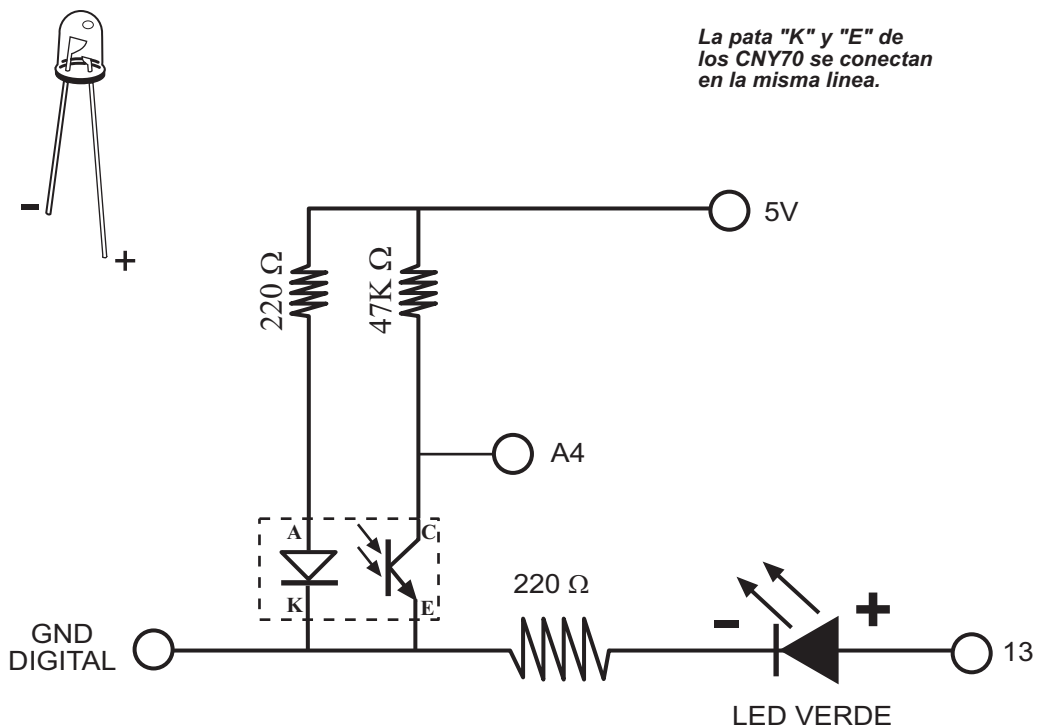
##### RESISTENCIAS

220  $\Omega$ : Rojo - Rojo - Marrón  
47K  $\Omega$ : Amarillo - Morado - Naranja

##### OPTOACOPLADOR CNY70 LOG 837

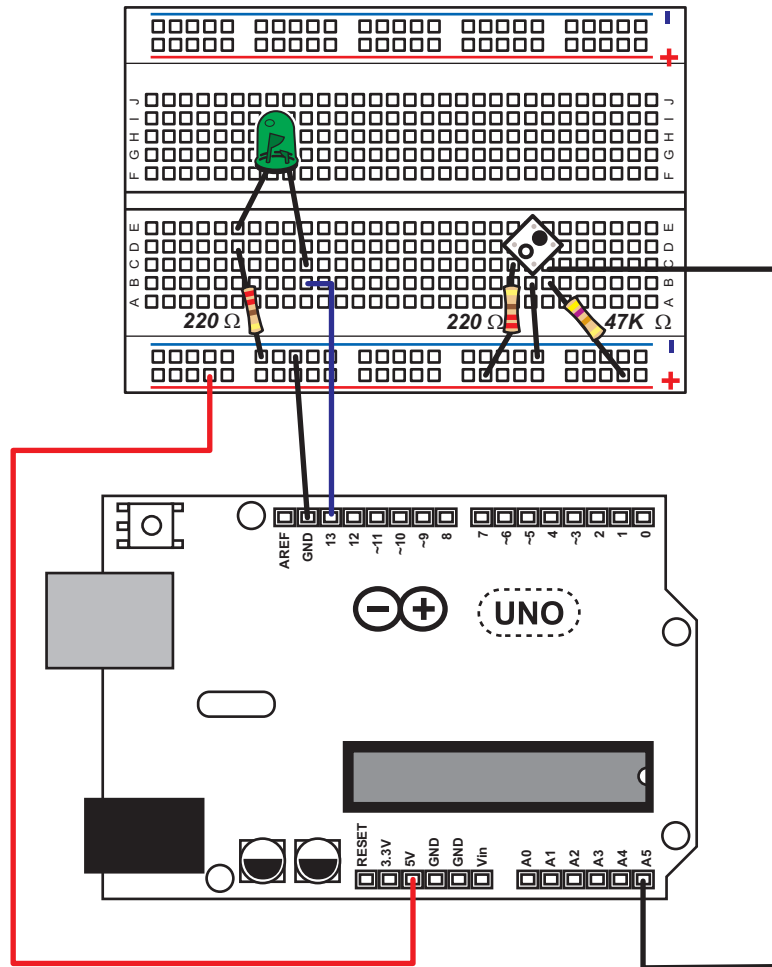


La pata "K" y "E" de los CNY70 se conectan en la misma línea.



## D. PRÁCTICAS

### Montaje en placa board



### Esquema del programa

```

Inicio de programa
  Bucle infinito
    Lectura del CNY70
    Si CNY70<400
      Enciende el LED
    Sino
      Apaga LED
  Fin del programa
  
```

## D. PRÁCTICAS

### Programa con Makeblock



### Programa con processing

```

int LED = 13;
int sensorValue=0; //El sensor CNY70 LOG 837

void setup() {
  pinMode(LED, OUTPUT); //Inicializacion de pin LED LOG 722 de salida
  Serial.begin(9600);
}

void loop() {

  sensorValue = analogRead(A5); //Leemos el sensor CNY70 LOG 837
  Serial.println(sensorValue); // Imprimimos en puerto serie para depurar
  if (analogRead(A5)<400) //si sensor LOG 837 destecta objeto
  {
    digitalWrite(LED, HIGH); // activamos el LED verde LOG 722
  }
  else
  {
    digitalWrite(LED, LOW); // desctivamos el LED verde LOG 722
  }
}
  
```

## **D. PRÁCTICAS**

### **15. MOVIMIENTO DE UN SERVOMOTOR 180°**

#### **Introducción**

Un servomotor es un motor de precisión cuyo movimiento se gestiona indicando ángulos de posicionamiento.

En esta práctica el usuario introduce un valor de 0 a 180 y el servo se posiciona en ese ángulo.

#### **Lista de materiales**

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Miniservo LOG 06
- 3 Latiguillos board macho-macho LOG 9519

#### **Esquema del circuito**

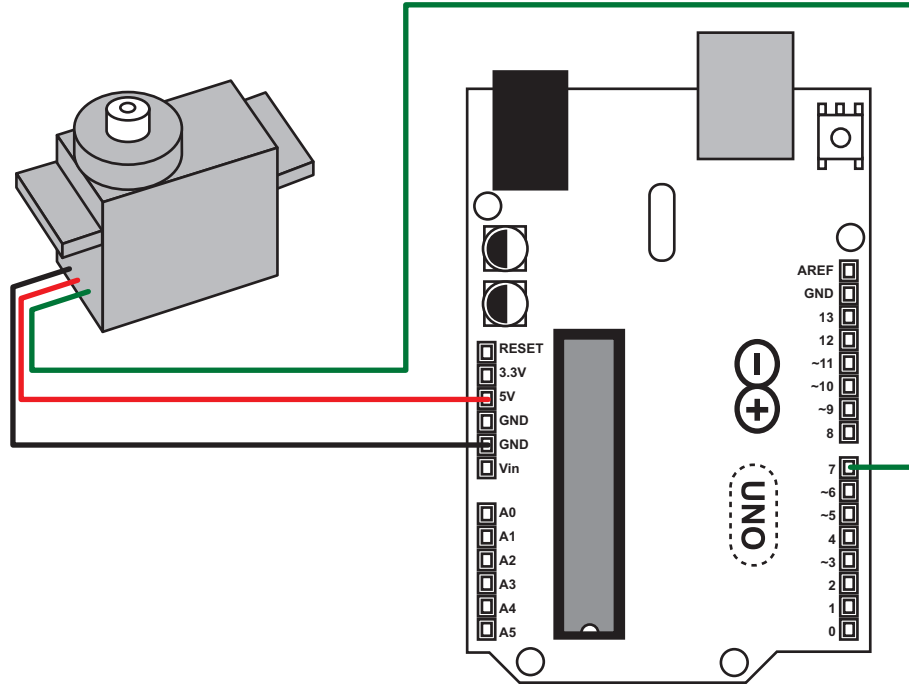
- Cable naranja – salida 7
- Cable marrón – GND
- Cable rojo – 5V

#### **Esquema del programa**

```
Por siempre
    Pedir ángulo
    Si ángulo >= 0 y <= 180
        Posicionar motor en ese ángulo
Fin
```

## D. PRÁCTICAS

### Montaje en placa board



## D. PRÁCTICAS

### Programa con Scratch



### Programa con Makeblock



### Programa con processing

```
#include <Servo.h>
int input;

void setup() {
  Serial.begin(9600);
  myservo.attach(7);
}

void loop() {
  if (Serial.available()>0){
    input=Serial.parseInt();
    if (input>0 && input<181){
      myservo.write(input);
      delay(15)
    }
  }
}
```

## **D. PRÁCTICAS**

### **16. MOVIMIENTO DE UN SERVO CON UN POTENCIÓMETRO**

#### **Introducción**

Conectar un servomotor a la tarjeta Arduino UNO y realizar un programa para controlar el giro del motor con un potenciómetro.

Conectamos el potenciómetro a la entrada analógica 0 y el servomotor al pin digital 8.

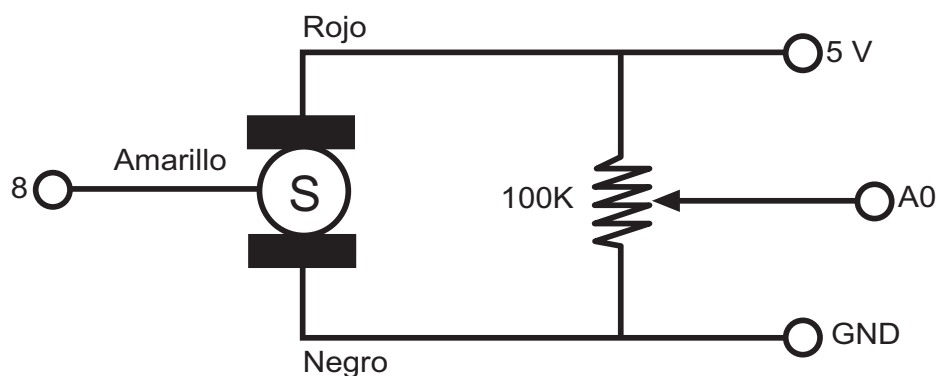
Introducimos en el editor de programas Arduino el código indicado en el apartado 6 y cargamos el programa en la tarjeta Arduino UNO.

Con un destornillador plano giramos el potenciómetro y observamos cómo el motor gira a la vez que variamos el potenciómetro y se alterna el sentido de giro en función del sentido del destornillador.

#### **Lista de materiales**

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Potenciómetro 100K LOG 743
- 1 Miniservo LOG 06
- 6 Latiguillos board macho-macho LOG 9519

#### **Esquema del circuito**



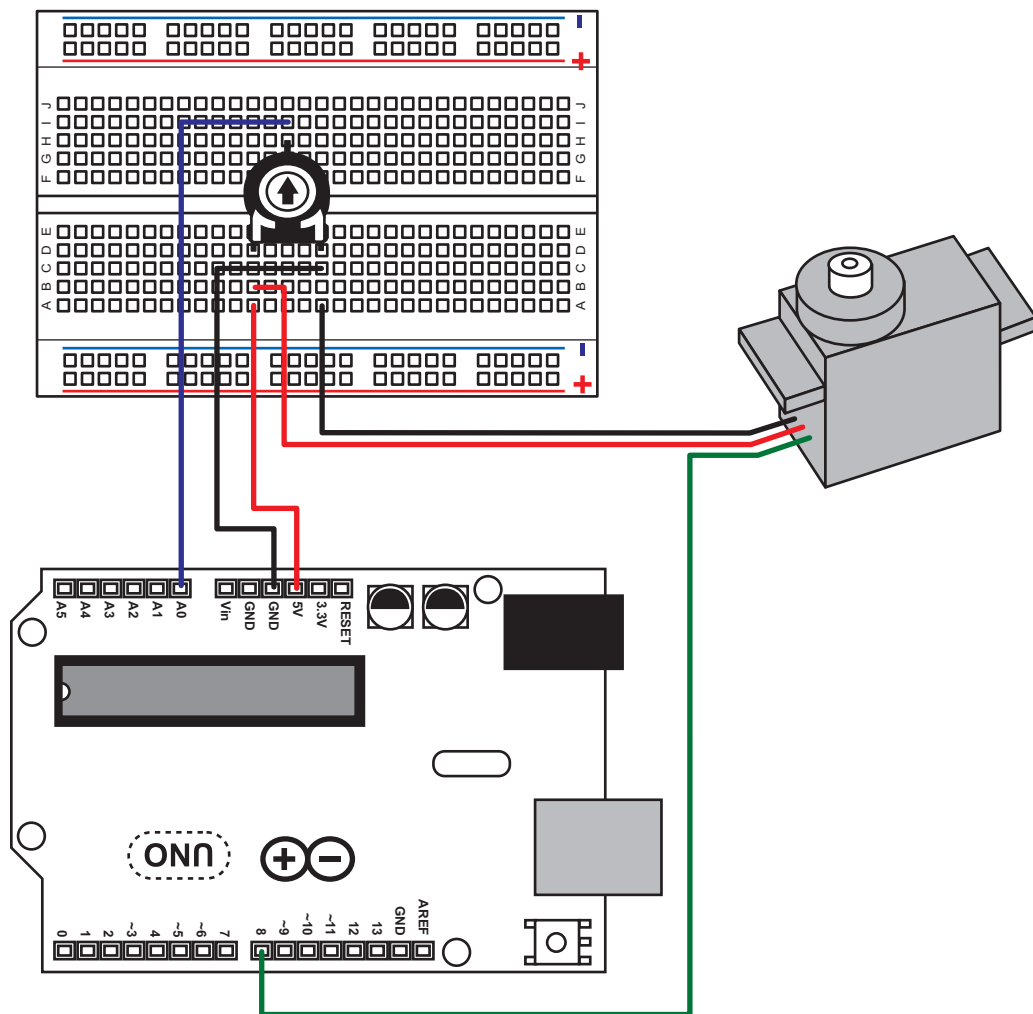


## D. PRÁCTICAS

### Montaje en placa board

Realizamos las siguientes conexiones:

- Conectamos la pata central del potenciómetro a la entrada analógica A0.
- Conectamos las patas traseras a 5V y GND de la tarjeta Arduino.
- Conectamos el cable rojo del motor a 5V de la tarjeta Arduino.
- Conectamos el cable marrón del motor a GND de la tarjeta Arduino.
- Conectamos el cable amarillo del motor al pin digital 8 de la tarjeta Arduino.
- Conectamos la tarjeta al puerto USB del ordenador.



### Esquema del programa

```

Inicio de programa
  Bucle infinito
    Lectura del potenciómetro
    Posicionamiento motor
    espera
  Fin del programa
  
```

## D. PRÁCTICAS

### Programa con processing

```
#include <Servo.h>
```

```
Servo myservo;  
int potpin=0;  
int val;
```

```
void setup()  
{  
  myservo.attach(8);  
}
```

```
void loop()  
{  
  val=analogRead(potpin);  
  val=map(val, 0, 1023, 0, 179);  
  myservor.write(val);  
  delay(15);  
}
```

### Programa con Scratch



### Programa con Makeblock



## **D. PRÁCTICAS**

### **Pruebas**

Al inicio el motor estará parado. Con un destornillador giramos el regulador del potenciómetro hacia la derecha. Observamos que mientras giramos el potenciómetro el motor se mueve.

Cuando dejamos de girar el regulador del potenciómetro el motor se para.

Con el destornillador giramos el regulador en sentido contrario y observamos que el motor cambia el sentido de giro y se activa mientras giramos el regulador.

Cuando dejamos de girar el regulador del potenciómetro el motor se para.

## **D. PRÁCTICAS**

### **17. MEDIDOR DE DISTANCIAS**

#### **Introducción**

Un sensor ultrasónico permite medir la distancia al objeto más cercano.

En esta práctica realizaremos un programa para medir estas distancias.

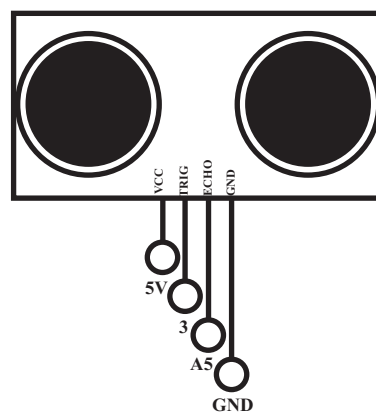
Mide la distancia enviando un pulso sonoro, que rebota en una superficie y capta cuanto tiempo ha tardado en rebotar en la superficie.

#### **Lista de materiales**

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Sensor ultrasónico LOG 4046
- 4 Latiguillos board macho-macho LOG 9519

#### **Conexión del ultrasónico**

- Vcc al pin de 5 V
- Trig al pin 12
- Echo al pin 11
- GND al pin GND



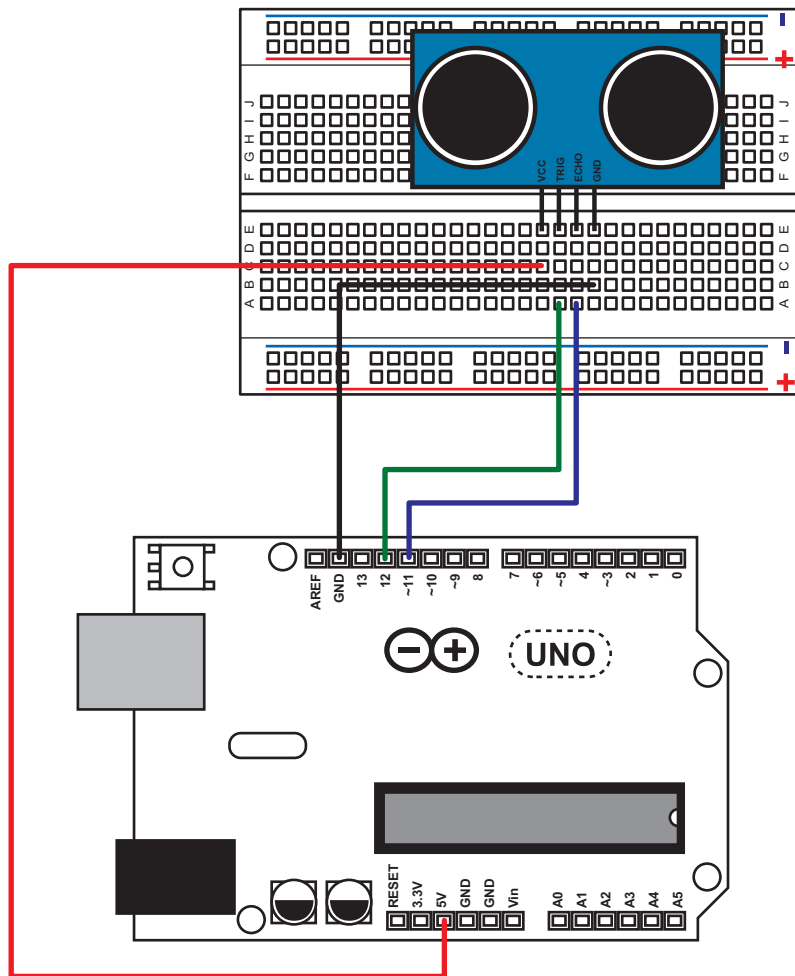
## D. PRÁCTICAS

### Montaje en placa board

Copiar la carpeta NewPing de "entrenador\_Arduino.zip", en la carpeta "libraries" que está dentro de la carpeta de Arduino.

Realizamos las siguientes conexiones:

- Conectamos "Vcc" a la salida 5 V de Arduino.
- Conectamos "Trig" al pin digital 12.
- Conectamos "Echo" al pin digital 11.
- Conectamos "Gnd" al pin GND de la tarjeta Arduino.



### Esquema del programa

```

Inicio de programa
  Bucle infinito
    Envio pulso de luz
    Lectura de la distancia
    Visualización distancia
  Fin programa
  
```

## D. PRÁCTICAS

### Programa con Scratch

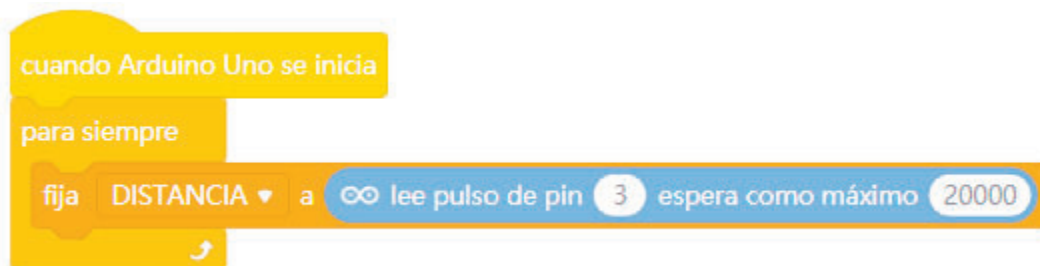
Para trabajar con Scratch, el sensor ultrasónico lleva asociado un firmware propio (S4Afirmware14\_distancia.ino). Este firmware funciona con la versión de S4A 1.5; ambos archivos se incluyen en el archivo descargable. En este firmware se han asignado los pines 10 y A5 al sensor ultrasónico, de tal forma que el pin "trig" se conectará al pin 10 y el pin "Echo" al A5. Cuando realizamos nuestro programa con S4A, la lectura del pin A5 se corresponderá con la distancia al objeto más cercano detectable.

Realizamos las siguientes conexiones:

- Conectamos "Vcc" a la salida 5 V de Arduino.
- Conectamos "Trig" al pin digital 3.
- Conectamos "Echo" al pin digital A5.
- Conectamos "Gnd" al pin GND de la tarjeta Arduino.



### Programa con Makeblock



### Programa con processing

```
#include <NewPing.h>
```

```
#define TRIGGER_PIN 3
```

```
#define ECHO_PIN A5
```

```
#define MAX_DISTANCE 200
```

```
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
```

```
void setup() {
```

```
    Serial.begin(115200); // Cambiar la velocidad de lectura de monitor serie a 115200
```

```
}
```

```
void loop() {
```

```
    delay(50);
```

```
    unsigned int uS = sonar.ping();
```

```
    Serial.print("Ping: ");
```

```
    Serial.print(uS / US_ROUNDTRIP_CM);
```

```
    Serial.println("cm");
```

```
}
```

### Pruebas

Abrir el "Serial monitor" de Arduino, comprobar que la velocidad esta en 115200 baudios. Acercar o alejar un objeto para que nos muestre la distancia a la que se encuentra.

## **D. PRÁCTICAS**

### **18. CONTROL DE MOTORES CON PUENTE EN H**

#### **Introducción**

Cuando conectamos un motor a la tarjeta Arduino UNO, las instrucciones `digitalWrite` / `analogWrite` nos permiten activar o parar el motor, pero no nos permiten el cambio de giro del motor.

Para poder realizar cambios de giro que permitan al coche invertir su marcha es preciso el uso de dispositivos como un puente en H.

Un puente en H consiste en 4 interruptores que nos permiten controlar la polaridad de la corriente que llega a los motores conectados al C.I. Así en función de esta polaridad, podemos controlar el sentido de giro de los motores.

Para poder controlar la velocidad de giro de los motores, se conectan a las salidas 5, 6, 9 y 10 de Arduino, que permiten modular la señal (PWM).

El programa a realizar consistirá en probar los diferentes modos de avance del coche:

- De frente
- Atrás
- Derecha
- Izquierda
- Parar
- Acelerar
- Frenar

#### **Lista de materiales**

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 2 Motores LOG 00
- 4 Condensadores 100 nF LOG 770
- 1 Condensador 100  $\mu$ F LOG 773
- 1 Controlador de motores L293D LOG 874
- 20 Latiguillos board macho-macho LOG 9519

## D. PRÁCTICAS

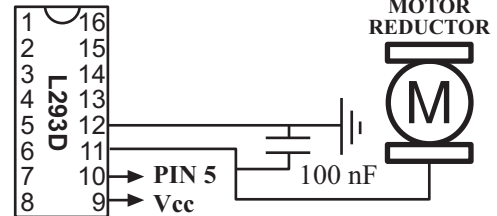
### Esquema del circuito

El circuito se plantea en 4 fases:

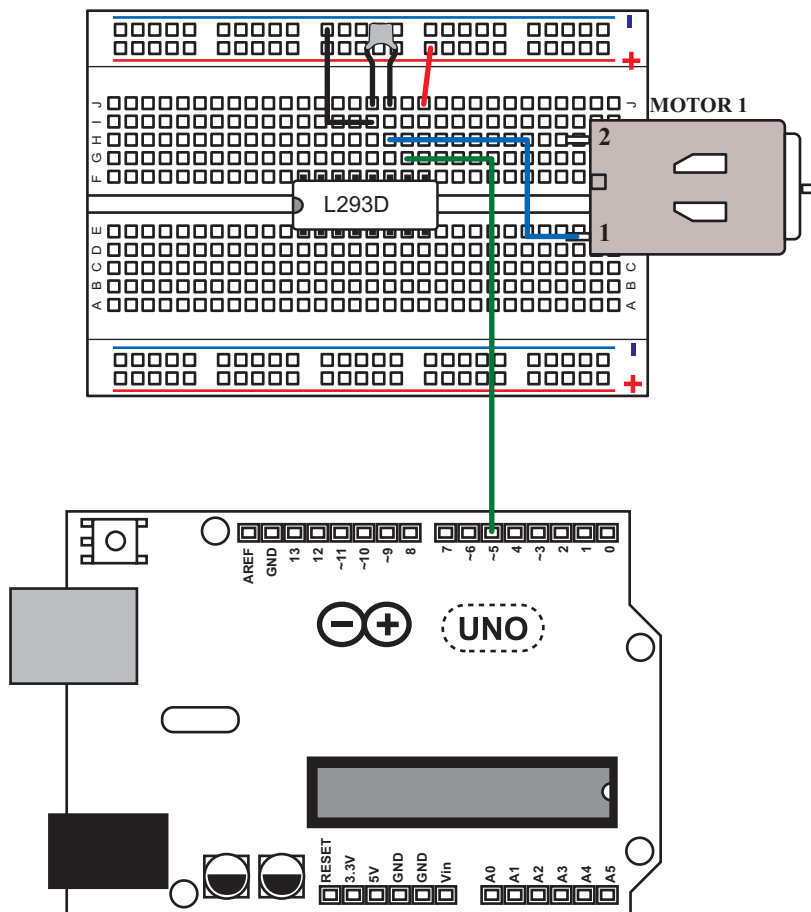
#### 1. AVANCE PARCIAL DEL MOTOR 1

- Conectar el terminal 1 del motor 1 al pin 5 (avance).
- Conectar el condensador de 100 nF a la salida 11 y 12 del L293D, para evitar picos de tensión en el circuito que puedan bloquear la tarjeta Arduino UNO.
- Conectar la salida 10 del L293D al PIN 5 de la tarjeta Arduino UNO.
- Conectar la salida 9 del L293D a la línea positiva de la placa board.

PUENTE EN H  
L293D LOG 874



### Montaje en placa board





## D. PRÁCTICAS

### Esquema del circuito

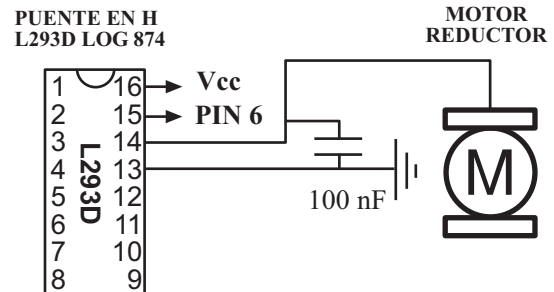
#### 2. RETROCESO PARCIAL DEL MOTOR 1

- Conectar el terminal 2 del motor 1 al pin 6 (retroceso).

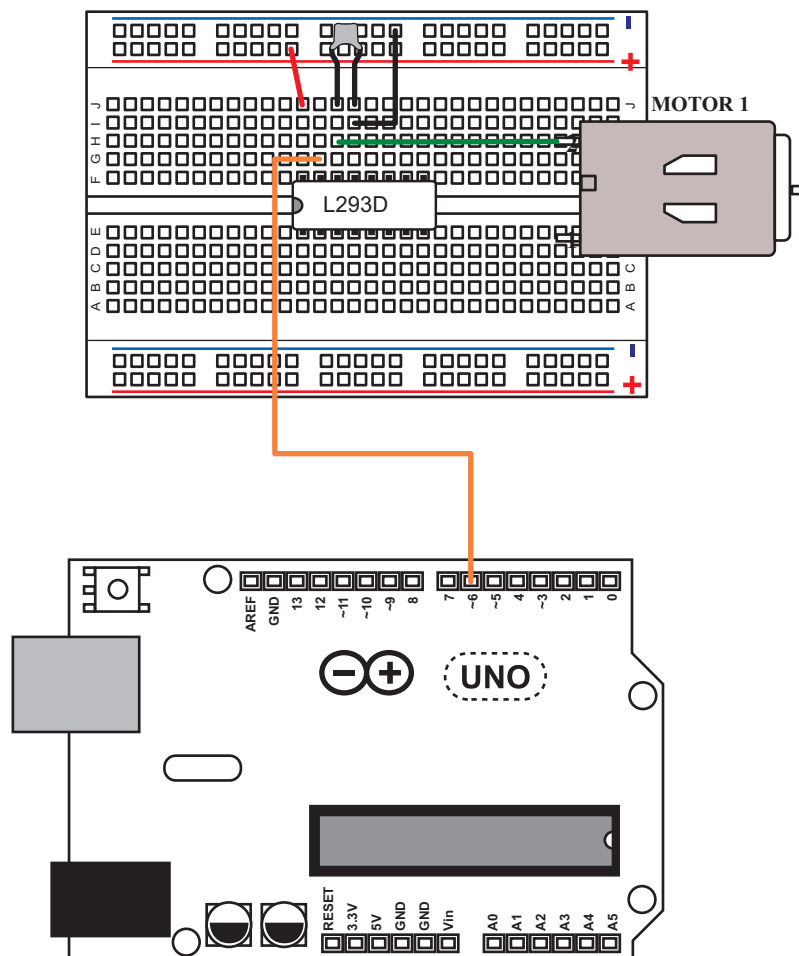
- Conectar el condensador de 100 nF a la salida 13 y 14 del L293D, para evitar picos de tensión en el circuito, que puedan bloquear la tarjeta Arduino UNO.

- Conectar la salida 15 del L293D al PIN 6 de la tarjeta Arduino UNO.

- Conectar la salida 16 del L293D a la línea positiva de la placa board.



### Montaje en placa board



## D. PRÁCTICAS

### Esquema del circuito

#### 3. AVANCE PARCIAL DEL MOTOR 2

- Conectar el terminal 3 del motor 2 al pin 9 (avance).

- Conectar el condensador de 100 nF a la salida 5 y 6 del L293D, para evitar picos de tensión en el circuito, que puedan bloquear la tarjeta Arduino UNO.

- Conectar la salida 7 del L293D al PIN 9 de la tarjeta Arduino UNO.

- Conectar la salida 8 del L293D a la línea positiva de la placa board.

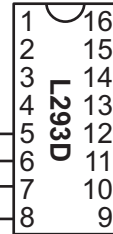
MOTOR  
REDUCTOR



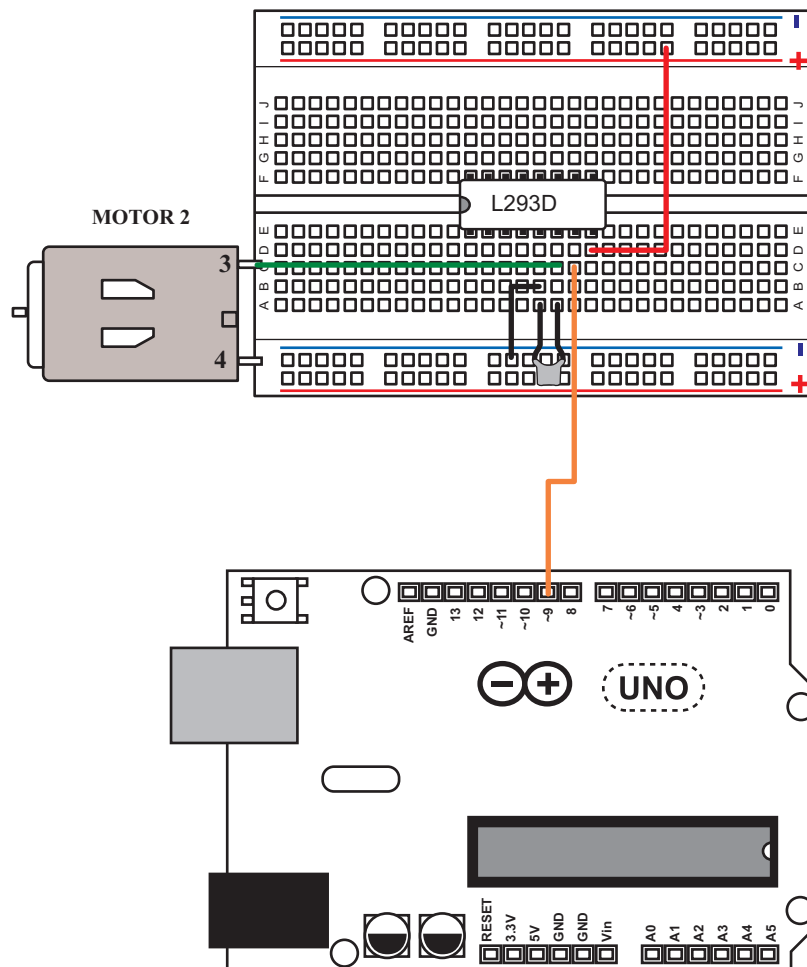
100 nF

PIN 9  
Vcc

PUENTE EN H  
L293D LOG 874



### Montaje en placa board



### Esquema del circuito

Diagrama de conexión para el motor reductor. Se muestra un motor reductor (MOTOR REDUCTOR) conectado a un puente en H L293D LOG 874. El motor está conectado a los pines 1 y 2 del puente. Un condensador de 100 nF está conectado entre los pines 1 y 2. El pín 10 del puente está conectado a Vcc. El pín 8 del puente está conectado a tierra.

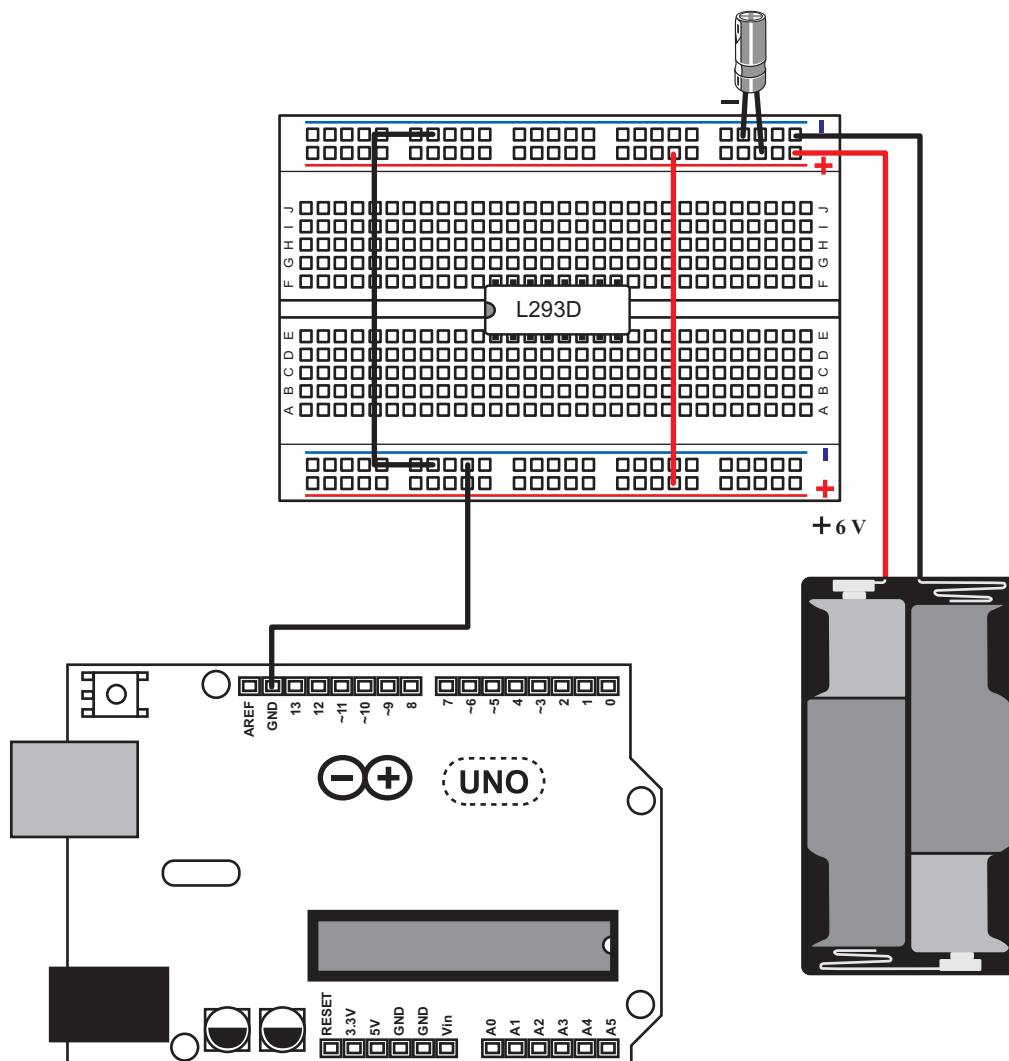
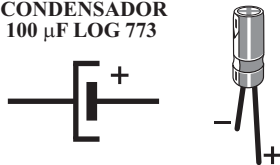
## D. PRÁCTICAS

### Montaje en placa board

#### ALIMENTACIÓN DE LOS MOTORES.

- Conectar el portapilas de 6 V a la placa board, el cable negro a la línea de negativos y conectar un latiguillo haciendo un puente entre líneas de negativo.
- Conectar el cable rojo al interruptor, un latiguillo del interruptor a la línea de positivos y otro latiguillo haciendo un puente entre líneas de positivo.
- Conectar el condensador de 100  $\mu$ F entre la línea de positivo y negativo, para evitar caídas de tensión que bloqueen la tarjeta Arduino (respetar la polaridad del condensador).

CONDENSADOR  
100  $\mu$ F LOG 773



## D. PRÁCTICAS

### Montaje en Makeblock



## D. PRÁCTICAS

### Programa con processing

```
int motor1Avance = 6; // usamos un pin de salida al motor
int motor1Atras = 5; // usamos un pin de salida al motor
int motor2Avance = 10; // usamos un pin de salida al motor
int motor2Atras = 9; // usamos un pin de salida al motor

void setup()
{
  pinMode(motor1Avance, OUTPUT); //definimos los pines para los motores como salidas
  pinMode(motor1Atras, OUTPUT);
  pinMode(motor2Avance, OUTPUT);
  pinMode(motor2Atras, OUTPUT);
  analogWrite(motor1Avance, 0); //La plataforma móvil LOG 4080 comienza en estado parada
  analogWrite(motor1Atras, 0);
  analogWrite(motor2Avance, 0);
  analogWrite(motor2Atras, 0);
}

void loop() {

  Serial.println("ADELANTE");
  analogWrite(motor1Avance, 255); //el coche LOG 4205 avanza en línea recta
  analogWrite(motor1Atras, 0);
  analogWrite(motor2Avance, 217);
  analogWrite(motor2Atras, 0);
  delay(5000);

  Serial.println("ATRAS"); //corresponde con el botón retroceder del LOG 4000
  analogWrite(motor1Avance, 0); // la plataforma LOG 4080 retrocede
  analogWrite(motor1Atras, 255);
  analogWrite(motor2Avance, 0);
  analogWrite(motor2Atras, 217);
  delay(5000);

  Serial.println("IZQUIERDA"); //corresponde con el botón izda del LOG 4000
  analogWrite(motor1Avance, 200); // La plataforma LOG 4080 gira a la izquierda
  analogWrite(motor1Atras, 0);
  analogWrite(motor2Avance, 100);
  analogWrite(motor2Atras, 0);
  delay(5000);

  Serial.println("DERECHA"); //corresponde con el botón decha del LOG 4000
  analogWrite(motor1Avance, 100); // La plataforma LOG 4080 gira a la derecha
  analogWrite(motor1Atras, 0);
  analogWrite(motor2Avance, 200);
  analogWrite(motor2Atras, 0);
  delay(5000);

  Serial.println("PARAR"); //corresponde con el botón de parar del LOG 4000
  analogWrite(motor1Avance, 0); // paramos la plataforma LOG 4080
  analogWrite(motor1Atras, 0);
  analogWrite(motor2Avance, 0);
  analogWrite(motor2Atras, 0);

  for (int i=0; i <= 255; i++){
    analogWrite(motor1Avance, i); //el coche LOG 4205 acelera en línea recta
    analogWrite(motor1Atras, 0);
    analogWrite(motor2Avance, i);
    analogWrite(motor2Atras, 0);
    delay(10);
  }

  for (int i=255; i >0 255; i--){
    analogWrite(motor1Avance, i); //el coche LOG 4205 frena en línea recta
    analogWrite(motor1Atras, 0);
    analogWrite(motor2Avance, i);
    analogWrite(motor2Atras, 0);
    delay(10);
  }
}
```

## D. PRÁCTICAS

### 19. CONEXIÓN DISPLAY LCD

#### Introducción

Visualización de un texto en un display LCD.

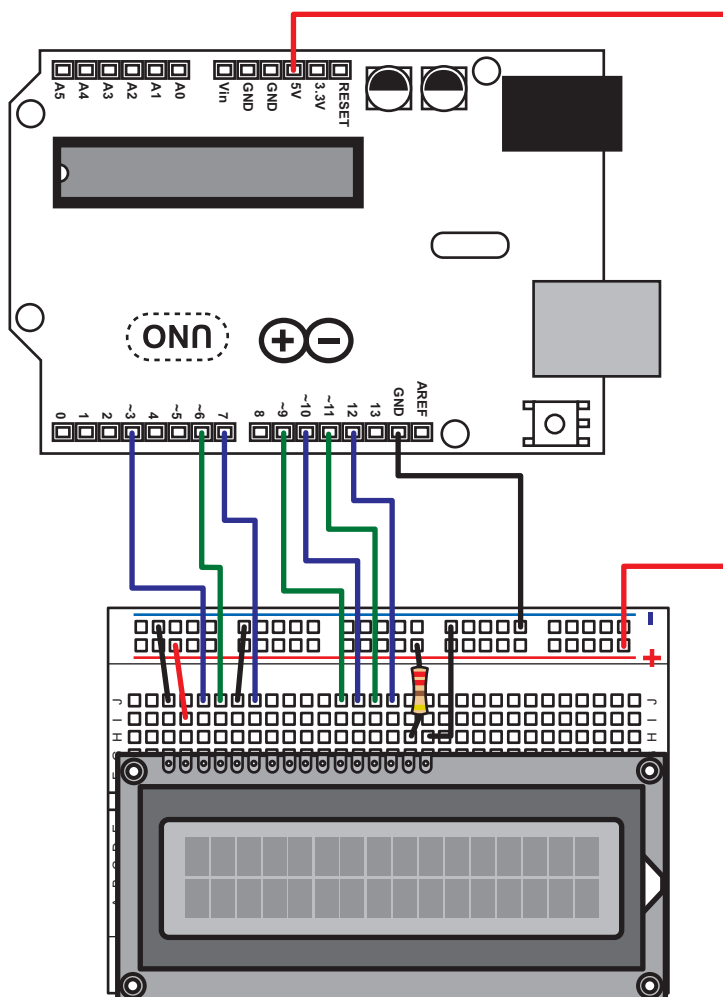
#### Lista de materiales

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Display LCD LOG 4069
- 1 Resistencia de 220 ohmios (rojo-rojo-marrón) LOG 748 220
- 13 Latiguillos board macho-macho LOG 9519

#### Montaje en placa board

Realizamos las siguientes conexiones:

- Conectamos la salida 5 V de Arduino a la línea "+" (positiva).
- Conectamos un pin GND de Arduino a la línea "-" (negativa).



**NOTA:**  
El display requiere soldar torretas a los pines de salida para conectar a la placa board

## **D. PRÁCTICAS**

### **Programa con processing**

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(6, 7, 9, 10, 11, 12);

void setup() {
    lcd.begin(16, 2);
    analogWrite(3,100);
}

void loop() {
    lcd.print("M");
    delay(1000);
    lcd.print("I");
    delay(1000);
    lcd.print("C");
    delay(1000);
    lcd.print("R");
    delay(1000);
    lcd.print("O");
    delay(1000);
    lcd.print("L");
    delay(1000);
    lcd.print("O");
    delay(1000);
    lcd.print("G");
    delay(1000);
    for(int i=0; i<=3; i++){
        lcd.clear();
        delay(1000);
        lcd.print("MICROLOG");
        delay(1000);
    }
    lcd.clear();
}
```



## D. PRÁCTICAS

### 20. CONTROL DE LEDS CON MANDO A DISTANCIA

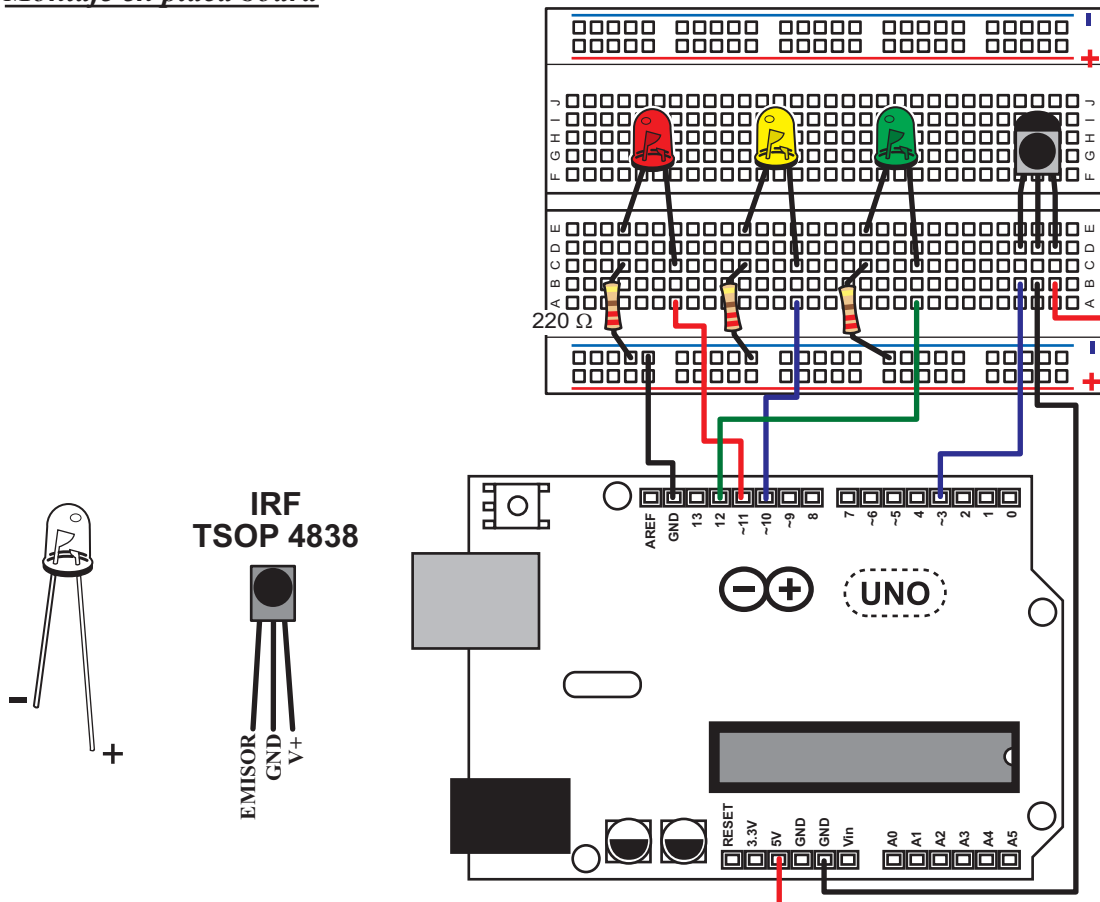
#### Introducción

Con ayuda de un mando a distancia por infrarrojos LOG 4000, cuya señal vamos a recibir con un TSOP LOG 835, la placa Arduino encenderá y apagará 3 leds en función del botón del mando pulsado por el usuario.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Placa board LOG 886
- 1 Led rojo LOG 724
- 1 Led amarillo LOG 723
- 1 Led verde LOG 722
- 3 Resistencias de 220 ohmios (rojo-rojo-marrón) LOG 748 220
- 1 Sensor TSOP LOG 835
- 1 Mando a distancia LOG 4000
- 7 Latiguillos board macho-macho LOG 9519

#### Montaje en placa board



## D. PRÁCTICAS

### Programa con processing

```
#include <IRremote.h>

int RECV_PIN2 = 3;
int led_verde = 12;
int led_rojo = 11;
int led_amarillo = 10;
int e_verde=0;
int e_rojo=0;
int e_amarillo=0;

IRrecv irrecv(RECV_PIN2);

decode_results results;

void setup()
{
    Serial.begin(9600);
    irrecv.enableIRIn();           // activamos el sensor TSOP LOG 835
    pinMode(led_verde, OUTPUT);    // definimos los pines para los leds como salidas
    pinMode(led_rojo, OUTPUT);
    pinMode(led_amarillo, OUTPUT);
}

void loop() {
    if (irrecv.decode(&results)) {
        Serial.println(results.value, HEX);
        int value = results.value;
        Serial.println(value);
        Serial.println("leyendo");
        irrecv.resume();           // Recibimos un dato del mando LOG 4000

        if (value==2295){           // si el dato recibido es 2295, corresponde a la flecha avance - enciendo led verde
            if(e_verde==0){         // enciendo led verde
                digitalWrite(led_verde, HIGH);
                e_verde=1;
                Serial.println("enciendo led verde");
            }
            else
            {
                Serial.println("apago led verde");
                digitalWrite(led_verde, LOW);
                e_verde=0;
            }
        }

        if (value==20655){ // si el dato recibido por el LOG 835 es 20655
            if(e_rojo==0){ // enciendo led rojo
                digitalWrite(led_rojo, HIGH);
                e_rojo=1;
                Serial.println("enciendo led rojo");
            }
            else
            {
                Serial.println("apago led rojo");
                digitalWrite(led_rojo, LOW);
                e_rojo=0;
            }
        }

        if (value==2041){ // si el dato recibido por el LOG 835 es 20655
            if(e_amarillo==0){ // enciendo led amarillo
                digitalWrite(led_amarillo, HIGH);
                e_amarillo=1;
                Serial.println("enciendo led amarillo");
            }
            else
            {
                Serial.println("apago led amarillo");
                digitalWrite(led_amarillo, LOW);
                e_amarillo=0;
            }
        }

        irrecv.resume(); // recibe un dato del emisor de infrarrojo
    }

    delay(100);
}
```

## **D. PRÁCTICAS**

### **21. MOTOR PASO A PASO**

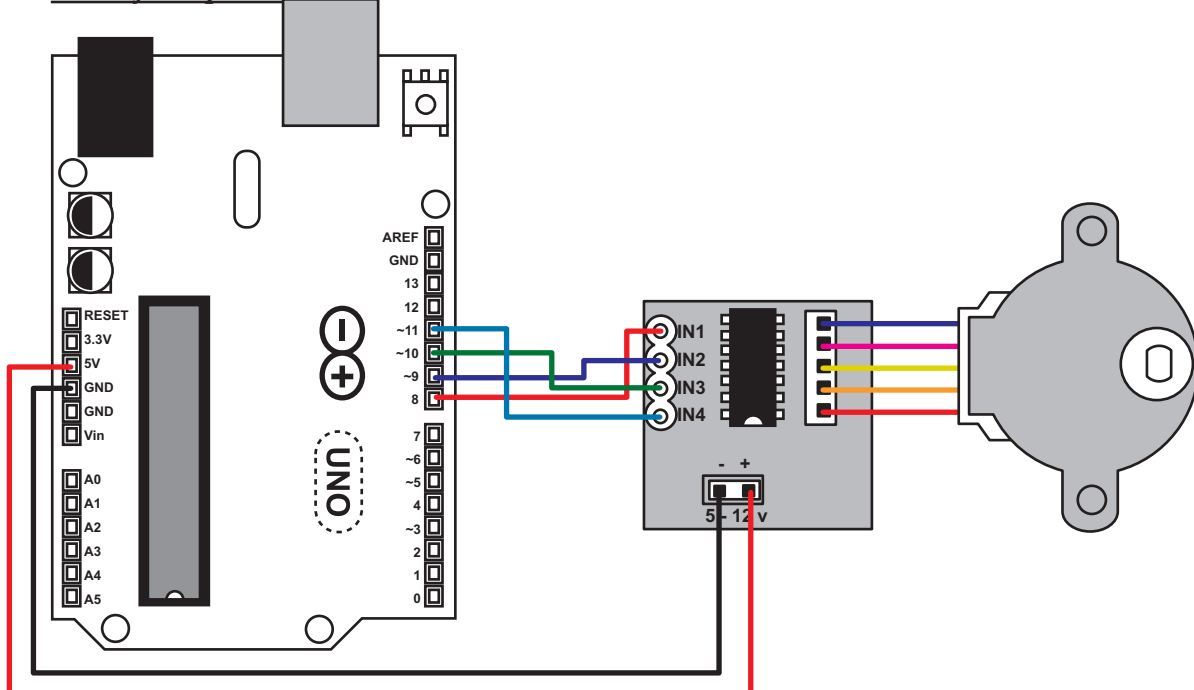
#### **Introducción**

Realizamos movimiento de avance con pausa sobre un motor paso a paso con controlador.

#### **Lista de materiales**

- 1 Tarjeta Arduino LOG 4031
- 1 Cable USB LOG 4009
- 1 Motor paso a paso y controlador LOG 14P
- 6 Latiguillos board macho-macho LOG 9519

#### **Montaje en placa board**



#### **Programa con processing**

```
#include <Stepper.h>
#define STEPS 200

Stepper stepper(STEPS, 8, 9, 10, 11);

void setup(){
    stepper.setSpeed(60);
}

void loop(){
    stepper.step(100);
    delay(1000);
}
```

## 22. SENSOR DE SÓNIDO

Sencillo proyecto capaz de detectar las variaciones de sonido de una estancia.

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Placa board LOG 886
- 1 Diodo LED Verde LOG 722
- 1 Diodo LED Rojo LOG 724
- 1 Resistencia de 220 ohmios (rojo-rojo-marrón) LOG 748 220
- 1 Sensor de sonido LOG 8457
- 6 Latiguillos board macho-macho LOG 7519

## D. PRÁCTICAS

### *Programa con processing*

//Por debajo del umbral de sonido se enciende el LED verde  
//Por encima del umbral de sonido se enciende el LED rojo

```
void setup() {  
    pinMode(13, OUTPUT); //LED verde  
    pinMode(8, OUTPUT);  // LED Rojo  
}  
  
void loop() {  
    if (analogRead(A4)>600) //Umbral marcado en 600  
    {  
        digitalWrite(8, HIGH);  
        digitalWrite(13, LOW);  
        delay(100);  
    }  
    else  
    {  
        digitalWrite(13, HIGH);  
        digitalWrite(8, LOW);  
        delay(100);  
    }  
}
```

## D. PRÁCTICAS

## Introducción

## Lista de materiales

### Montaje en placa board

## D. PRÁCTICAS

### Programa con processing

```
void setup() {  
  
    pinMode(13, OUTPUT); //Declaramos PIN 13 de salida - LED VERDE  
    pinMode(8, OUTPUT);  //Declaramos PIN 8 de salida - LED ROJO  
    Serial.begin(9600);  
}  
  
void loop() {  
    // Leemos la señal del sensor de línea  
    Serial.println(analogRead(A4));  
    // Imprimimos por el monitor serie la lectura del sensor  
    Serial.println(A4);  
  
    if(analogRead(A4)<600) // Si el sensor está por debajo de 600  
    {  
        digitalWrite(13, HIGH); //Encendemos LED Verde  
        digitalWrite(8, LOW); //Apagamos LED ROJO  
        //Imprimimos que esta en blanco  
        Serial.println("Línea Blanca");  
    }  
    else // Sino, es que esta por encima de 600  
    {  
        digitalWrite(13, LOW); //Apagamos LED Verde  
        digitalWrite(8, HIGH); //Encendemos LED Rojo  
        //Imprimimos que esta sobre línea negra  
        Serial.println("Línea Negra");  
    }  
}
```

## D. PRÁCTICAS

### 24. MÓDULO DISPLAY DE 4 DÍGITOS

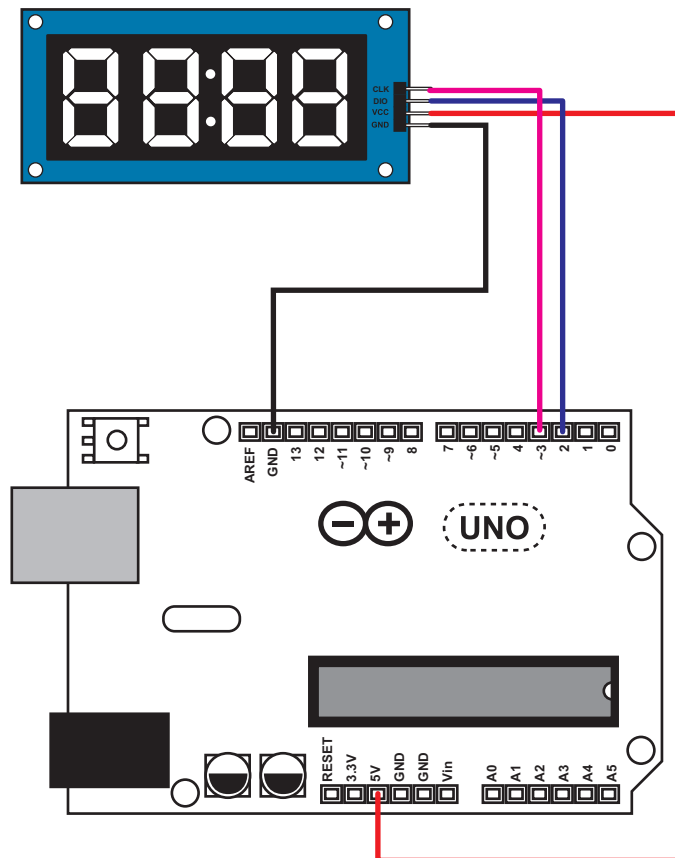
#### Introducción

Sencillo proyecto de como se imprimen números en el display.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Módulo display de 4 dígitos LOG 8454
- 4 Latiguillos board hembra-macho LOG 9518

#### Montaje en placa board





## D. PRÁCTICAS

### Programa con processing

```
#include <TM1637.h> //Incluimos libreria del DigitalTube

//Definimos un par de pines para la señal CLK y DIO
#define CLK A5 //Salida analógica y comunicación A5
#define DIO A4 //Salida analógica y comunicación A4

TM1637 display1(CLK,DIO); //Ahora tenemos que crear una instancia del
tipo TM1637 que llamaremos Display1
int8_t digitos[] = {0,0,0,0} ; //Pasemos los dígitos a representar
en un array de 4 byte y los inicializamos a 0

void setup() {
    display1.set(7); // Intensidad de luz del 0 al 7
    display1.init(); // inicializar la librería y el display
}

void loop() {
    digitos[0]=0;
    digitos[1]=1;
    digitos[2]=2;
    digitos[3]=3;
    display1.display(digitos); //Se imprime los digitos
    delay(1000);
    digitos[0]=4;
    digitos[1]=5;
    digitos[2]=6;
    digitos[3]=7;
    display1.display(digitos);
    delay(1000);
}
```

## D. PRÁCTICAS

### 25. MÓDULO MATRIZ DE LEDS

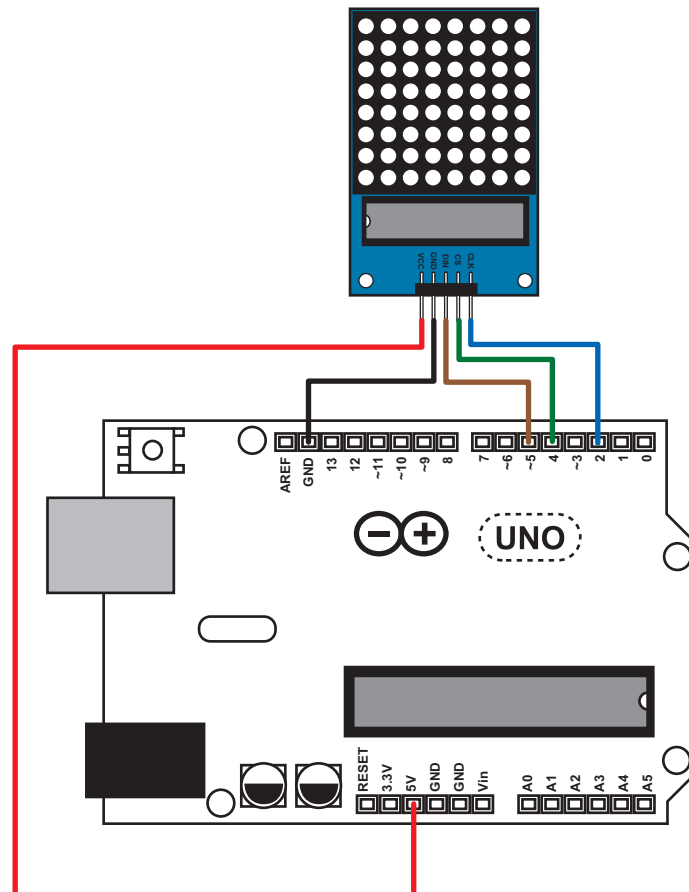
#### Introducción

Sencillo proyecto capaz de detectar las variaciones de sonido de una estancia.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Módulo Matriz de LEDs LOG 8467
- 5 Latiguillos board macho-macho LOG 7519

#### Montaje en placa board



## D. PRÁCTICAS

### Programa con processing

```
#include "LedControl.h"

LedControl lc=LedControl(5, 4, 2, 1); // DIN, CLK, CS, NRDEV

//Una variable para esperar antes de actualizar la pantalla.
unsigned long delaytime = 50;

void setup() {
//Obtenga la cantidad de dispositivos que hemos "creado" con Ledcontrol.
  int devices=lc.getDeviceCount();

// Inicializa todos los dispositivos en un bucle.
  for(int address=0;address<devices;address++) {
    // El MAX72XX IC está en modo de suspensión al inicio.
    lc.shutdown(address,false);
    // Establecer el brillo a un nivel medio..
    lc.setIntensity(address,8);
    // Borrar la matriz de puntos (pantalla clara).
    lc.clearDisplay(address);
  }
}

void loop() {
// Leer la cantidad de dispositivos.
  int devices=lc.getDeviceCount();

// Deje que los LED se enciendan uno por uno.
  for(int row=0;row<8;row++) {
    for(int col=0;col<8;col++) {
      for(int address=0;address<devices;address++) {
        lc.setLed(address,row,col,true); delay(delaytime);
      }
    }
  }
// Apague los LED de a uno por vez.
  for(int row=0;row<8;row++) {
    for(int col=0;col<8;col++) {
      for(int address=0;address<devices;address++) {
        lc.setLed(address,row,col,false); delay(delaytime);
      }
    }
  }
}
```

## D. PRÁCTICAS

### 26. TECLADO NUMÉRICO

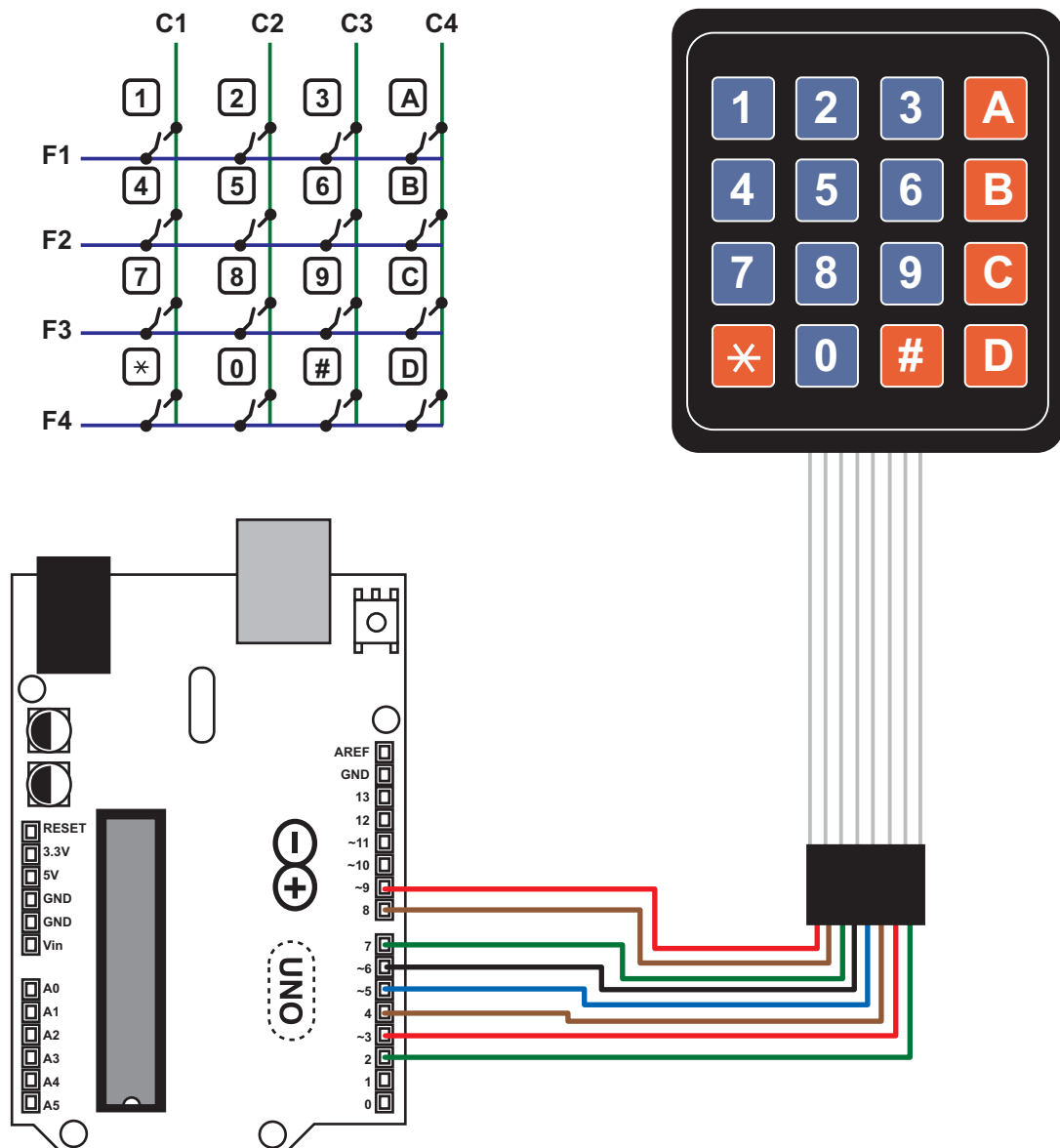
#### Introducción

Sencillo proyecto para aprender el funcionamiento de un teclado matricial.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Teclado matricial LOG
- 8 Latiguillos board macho-macho LOG 7519

#### Montaje en placa board



## D. PRÁCTICAS

### Programa con processing

```
#include <Keypad.h>           // importa libreria Keypad

const byte FILAS = 4; // define numero de filas
const byte COLUMNAS = 4; // define numero de columnas
char keys[FILAS][COLUMNAS] = { // define la distribucion de teclas
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};

// pines del Arduino correspondientes a las filas
byte pinesFilas[FILAS] = {9,8,7,6};

// pines del Arduino correspondientes a las columnas
byte pinesColumnas[COLUMNAS] = {5,4,3,2};

// crea objeto
Keypad teclado = Keypad(makeKeymap(keys), pinesFilas, pinesColumnas,
FILAS, COLUMNAS);

char TECLA; // almacena la tecla presionada

void setup()
{
    Serial.begin(9600); // inicializa comunicacion serie
}

void loop(){
    // obtiene tecla presionada y asigna a variable
    TECLA = teclado.getKey();
    if (TECLA) // comprueba que se haya presionado una tecla
    {
        Serial.print(TECLA); // envia a monitor serial la tecla presionada
    }
}
```

## D. PRÁCTICAS

### 27. ACCESO POR CONTRASEÑA

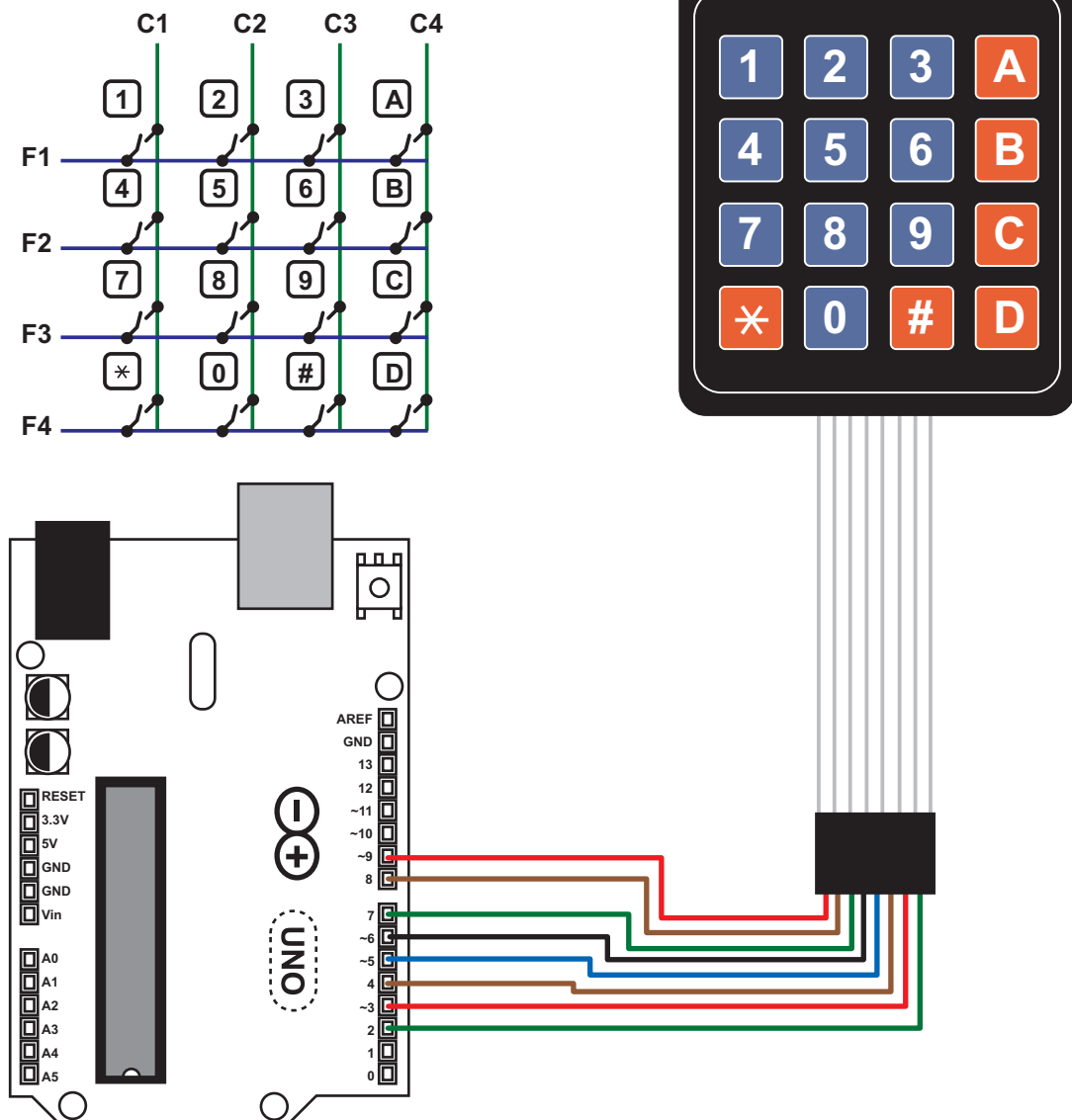
#### Introducción

Sencillo proyecto capaz de detectar las variaciones de sonido de una estancia.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Teclado matricial LOG
- 8 Latiguillos board macho-macho LOG 7519

#### Montaje en placa board



## D. PRÁCTICAS

### *Programa con processing*

```
#include <Keypad.h>           // importa libreria Keypad

const byte FILAS = 4; //define numero de filas
const byte COLUMNAS = 4; //define numero de columnas
char keys[FILAS][COLUMNAS] = { //define la distribucion de teclas
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};

//pines del Arduino correspondientes a las filas
byte pinesFilas[FILAS] = {9,8,7,6};

//pines del Arduino correspondientes a las columnas
byte pinesColumnas[COLUMNAS] = {5,4,3,2};

//crea objeto
Keypad teclado = Keypad(makeKeymap(keys), pinesFilas, pinesColumnas,
FILAS, COLUMNAS);

char TECLA; //almacena la tecla presionada
char CLAVE[5]; //almacena en un array 6 digitos ingresados
char CLAVE_MAESTRA[5] = "1234"; //almacena en un array la contraseña
byte INDICE = 0; //indice del array

void setup()
{
    Serial.begin(9600); //inicializa comunicacion serie
}

void loop(){
    //obtiene tecla presionada y asigna a variable
    TECLA = teclado.getKey();
    if (TECLA) //comprueba que se haya presionado una tecla
    {
        CLAVE[INDICE] = TECLA; //almacena en array la tecla presionada
        INDICE++; //incrementa indice en uno
        Serial.print(TECLA); //envia a monitor serial la tecla presionada
    }

    if(INDICE == 4) //si ya se almacenaron los 4 digitos
    {
        //compara la clave ingresada con la clave correcta
        if(!strcmp(CLAVE, CLAVE_MAESTRA))
            //Si es correcta imprime en monitor serie "Correcta"
            Serial.println(" Correcta");
        else
            //Si no es correcta imprime en monitor serie "Incorrecta"
            Serial.println(" Incorrecta");
        INDICE = 0;
    }
}
```

## D. PRÁCTICAS

### 28. SENSOR DE COLOR

#### Introducción

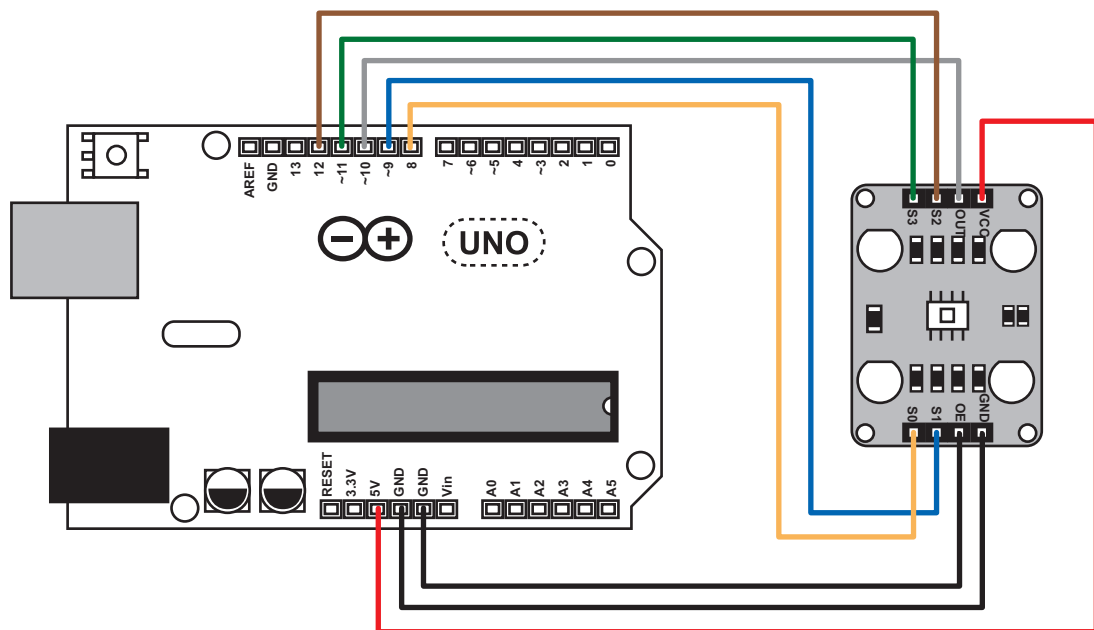
Sencillo práctica para identificar el color de un objeto entre Rojo, Verde y Azul.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Sensor de color LOG 8474
- 8 Latiguillos board macho-macho LOG 7519

#### Montaje en placa board

| Sensor | Arduino |
|--------|---------|
| VCC    | 5V      |
| GND    | GND     |
| S0     | 8       |
| S1     | 9       |
| S2     | 12      |
| S3     | 11      |
| OUT    | 10      |
| OE     | GND     |





## D. PRÁCTICAS

### *Programa con processing*

```
/* TCS230 sensor de color */

const int s0 = 8;
const int s1 = 9;
const int s2 = 12;
const int s3 = 11;
const int out = 10;

// Variables para los colores RGB
int rojo = 0;
int verde = 0;
int azul = 0;

void setup()
{
  Serial.begin(9600);
  pinMode(s0, OUTPUT);
  pinMode(s1, OUTPUT);
  pinMode(s2, OUTPUT);
  pinMode(s3, OUTPUT);
  pinMode(out, INPUT);
  digitalWrite(s0, HIGH);
  digitalWrite(s1, HIGH);
}

void loop()
{
  color();
  /*Serial.print("Rojo:");
  Serial.print(rojo, DEC);
  Serial.print(" Verde: ");
  Serial.print(verde, DEC);
  Serial.print(" Azul: ");
  Serial.print(azul, DEC); */

  if (rojo < azul && rojo < verde && rojo < 20)
  {
    Serial.println("Color Rojo");
  }
  else if (azul < rojo && azul < verde)
  {
    Serial.println("Color Azul");
  }
  else if (verde < rojo && verde < azul)
  {
    Serial.println("Color Verde");
  }
  else{
    Serial.println("ERROR");
  }
  delay(300);
}

void color()
{
  digitalWrite(s2, LOW);
  digitalWrite(s3, LOW);
  //count OUT, pRed, RED
  rojo = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
  digitalWrite(s3, HIGH);
  //count OUT, pBLUE, BLUE
  azul = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
  digitalWrite(s2, HIGH);
  //count OUT, pGreen, GREEN
  verde = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
}
```

## D. PRÁCTICAS

### 29. MÓDULO RELÉ

#### Introducción

Activar un relé a través del monitor serie para encender o apagar un operador.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Módulo relé LOG 8468
- 3 Latiguillos board macho-macho LOG 7519

#### Montaje en placa board

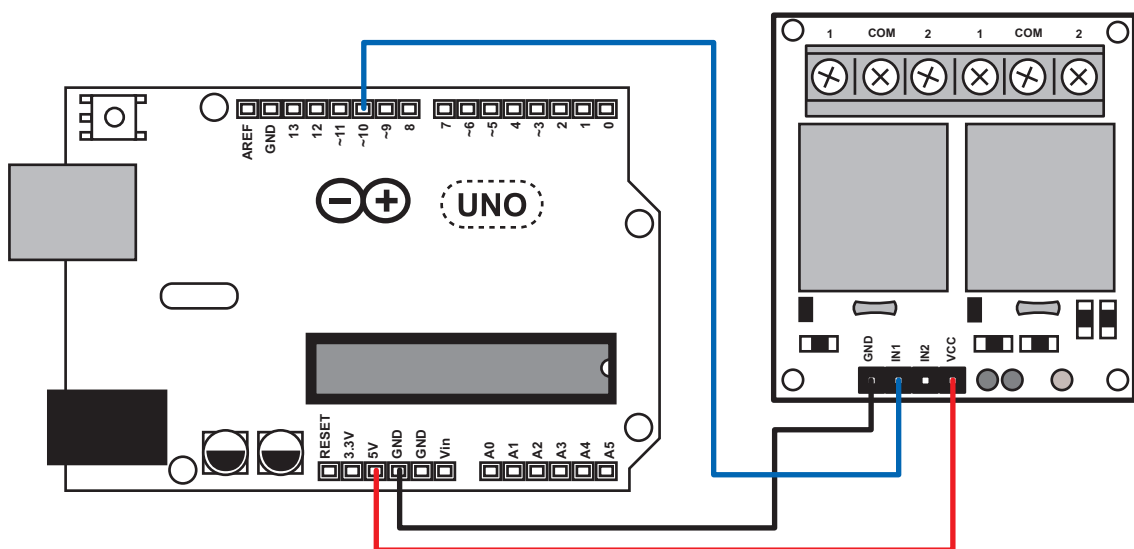
##### Módulo Relé:

Conexión con el Arduino:

VCC - PIN 5V  
GND - PIN GND  
IN1 - PIN 10

Conexión con el relé:

1 - Normalmente cerrado  
COM - Común  
2 - Normalmente abierto



## D. PRÁCTICAS

### Programa con processing

```
//Variable donde almacena el dato de entrada por el puerto serie
int opcion;

void setup() {
    Serial.begin(9600);
    pinMode(10, OUTPUT); //Pin 10 de salida
}

void loop() {
    //Si introducimos un valor mayor que 0
    if (Serial.available()>0){
        opcion=Serial.read();
        if(opcion=='1') { // Si es un 1 abre el circuito
            digitalWrite(10, LOW);
            Serial.println("OFF");
        }
        if(opcion=='2') { // Si es un 2 cierra el circuito
            digitalWrite(10, HIGH);
            Serial.println("ON");
        }
    }
}
```

## D. PRÁCTICAS

### 30. SENSOR ACCELERÓMETRO

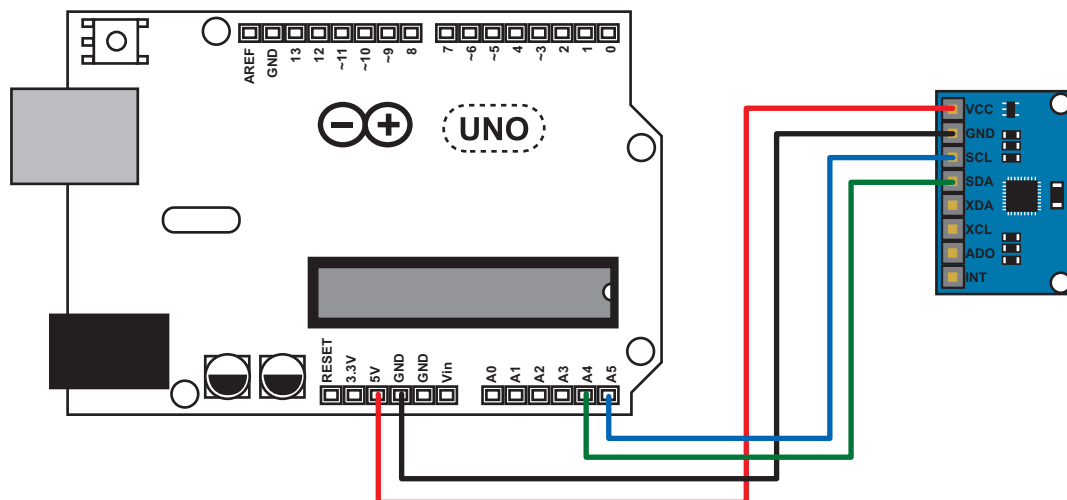
#### Introducción

Sencillo práctica para identificar el funcionamiento y visualizar los valores.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Sensor acelerómetro LOG 8445
- 4 Latiguillos board macho-macho LOG 7519

#### Montaje en placa board



## D. PRÁCTICAS

### Programa con processing

```

/*ACELERÓMETRO MPU6050*/

#include<Wire.h>
const int MPU_addr=0x68; // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup(){
    Wire.begin();
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x6B); // PWR_MGMT_1 register
    Wire.write(0); // set to zero (wakes up the MPU-6050)
    Wire.endTransmission(true);
    Serial.begin(9600);
}
void loop(){
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H)
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_addr,14,true); // request a total of 14
registers
    AcX=Wire.read()<<8|Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C
(ACCEL_XOUT_L)
    AcY=Wire.read()<<8|Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E
(ACCEL_YOUT_L)
    AcZ=Wire.read()<<8|Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40
(ACCEL_ZOUT_L)
    Tmp=Wire.read()<<8|Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
    GyX=Wire.read()<<8|Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44
(GYRO_XOUT_L)
    GyY=Wire.read()<<8|Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46
(GYRO_YOUT_L)
    GyZ=Wire.read()<<8|Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48
(GYRO_ZOUT_L)
    Serial.print("AcX = ");
    Serial.print(AcX);
    Serial.print(" | AcY = ");
    Serial.print(AcY);
    Serial.print(" | AcZ = ");
    Serial.print(AcZ);
    Serial.print(" | Tmp = ");
    Serial.print(Tmp/340.00+36.53); //equation for temperature in
degrees C from datasheet
    Serial.print(" | GyX = ");
    Serial.print(GyX);
    Serial.print(" | GyY = ");
    Serial.print(GyY);
    Serial.print(" | GyZ = ");
    Serial.println(GyZ);
    delay(333);
}

```

## D. PRÁCTICAS

### 31. SENSOR TEMPERATURA Y HUMEDAD

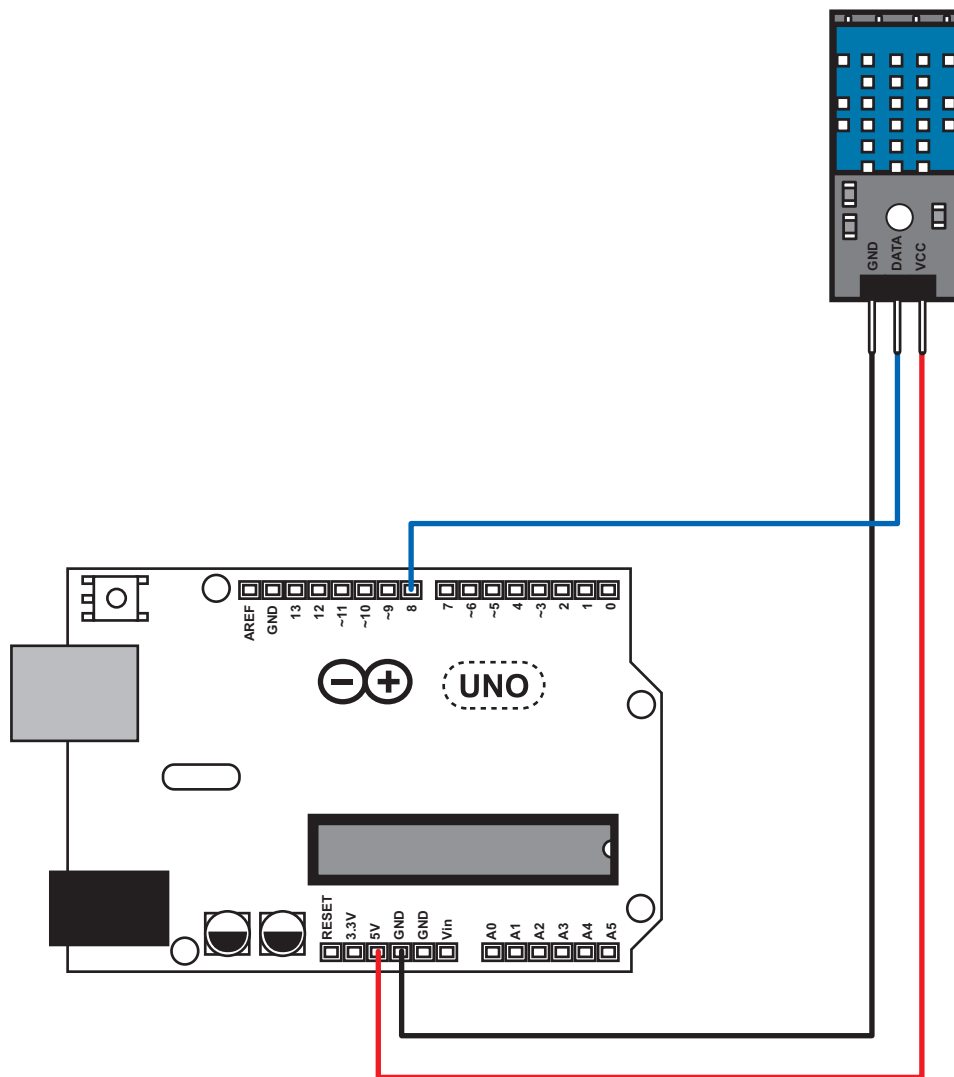
#### Introducción

Sencillo práctica para recoger valores de la temperatura y humedad.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Sensor de temperatura y humedad LOG 8452
- 3 Latiguillos board macho-macho LOG 7519

#### Montaje en placa board



## D. PRÁCTICAS

### Programa con processing

```
/* DHT11 sensor de TEMPERATURA Y HUMEDAD */

//Instalar la Librerias "DHT sensor library" y "Adafruit unified sensor"
#include <DHT.h>
#include <DHT_U.h>

int SENSOR = 8;      //Pin DATA de DHT11 a pin digital 8
int TEMPERATURA;
int HUMEDAD;

DHT dht(SENSOR, DHT11); //Creacion del objeto

void setup(){
    Serial.begin(9600); //Inicializacion de monitor serial
    dht.begin();        //Inicializacion de sensor
}

void loop(){
    //Obtencion de valor de temperatura
    TEMPERATURA = dht.readTemperature();
    //Obtencion de valor de humedad
    HUMEDAD = dht.readHumidity();
    //Escritura en monitor serial de los valores
    Serial.print("Temperatura: ");
    Serial.print(TEMPERATURA);
    Serial.print(" Humedad: ");
    Serial.println(HUMEDAD);
    delay(500);
}
```

## D. PRÁCTICAS

### 32. SENSOR DE ALCOHOL

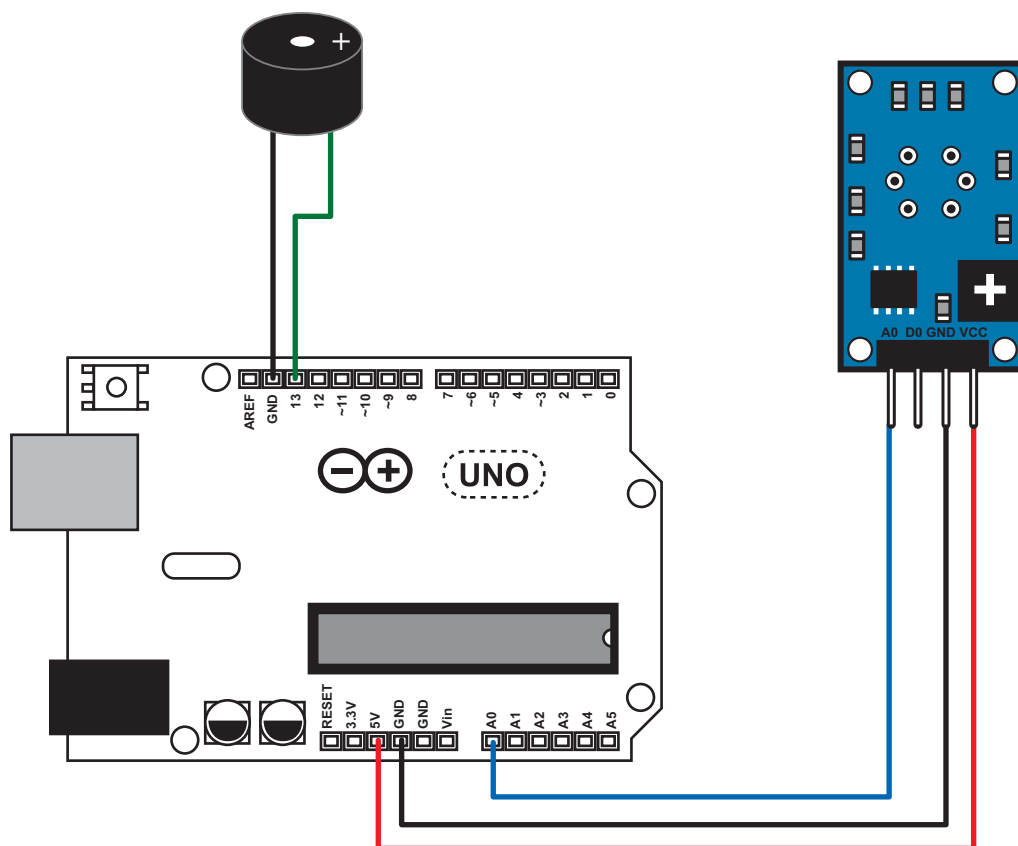
#### Introducción

Sencillo práctica para simular un alcoholímetro, cuando superamos un nivel de se dispara la alarma.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Sensor de alcohol LOG
- 5 Latiguillos board macho-macho LOG 7519

#### Montaje en placa board





## D. PRÁCTICAS

### ***Programa con processing***

```
/* Alarma de alcohol MQ-3*/

// Fijar el valor limite en el que se activa la alarma
int valor_limite= 400;

// Fije el valor despues de visualizar el nivel con el Monitor Serial
float valor_alcohol;

void setup() {
    Serial.begin(9600);
    pinMode(13,OUTPUT); // Pin 13 como salida para el Zumbador
}

void loop() {
    valor_alcohol=analogRead(A0);
    //Muestra por el puerto serie el valor leido del Sensor MQ3
    Serial.println(valor_alcohol);
    float porcentaje=(valor_alcohol/10000); //Calcula el porcentaje
    // Muestra por el puerto serie el porcentaje
    Serial.print(porcentaje);
    Serial.println("%");
    Serial.println(' ');

    // Si la medida de gas es mayor de valor limite
    if(valor_alcohol > valor_limite){
        // Enciende el Zumbador conectado al Pin 13
        digitalWrite(13, HIGH);
    }
    else{ // Si es menor del valor limite apaga el Zumbador
        digitalWrite(13, LOW);
    }
    delay (300);
}
```

## D. PRÁCTICAS

### 33. SENSOR DE CAUDAL

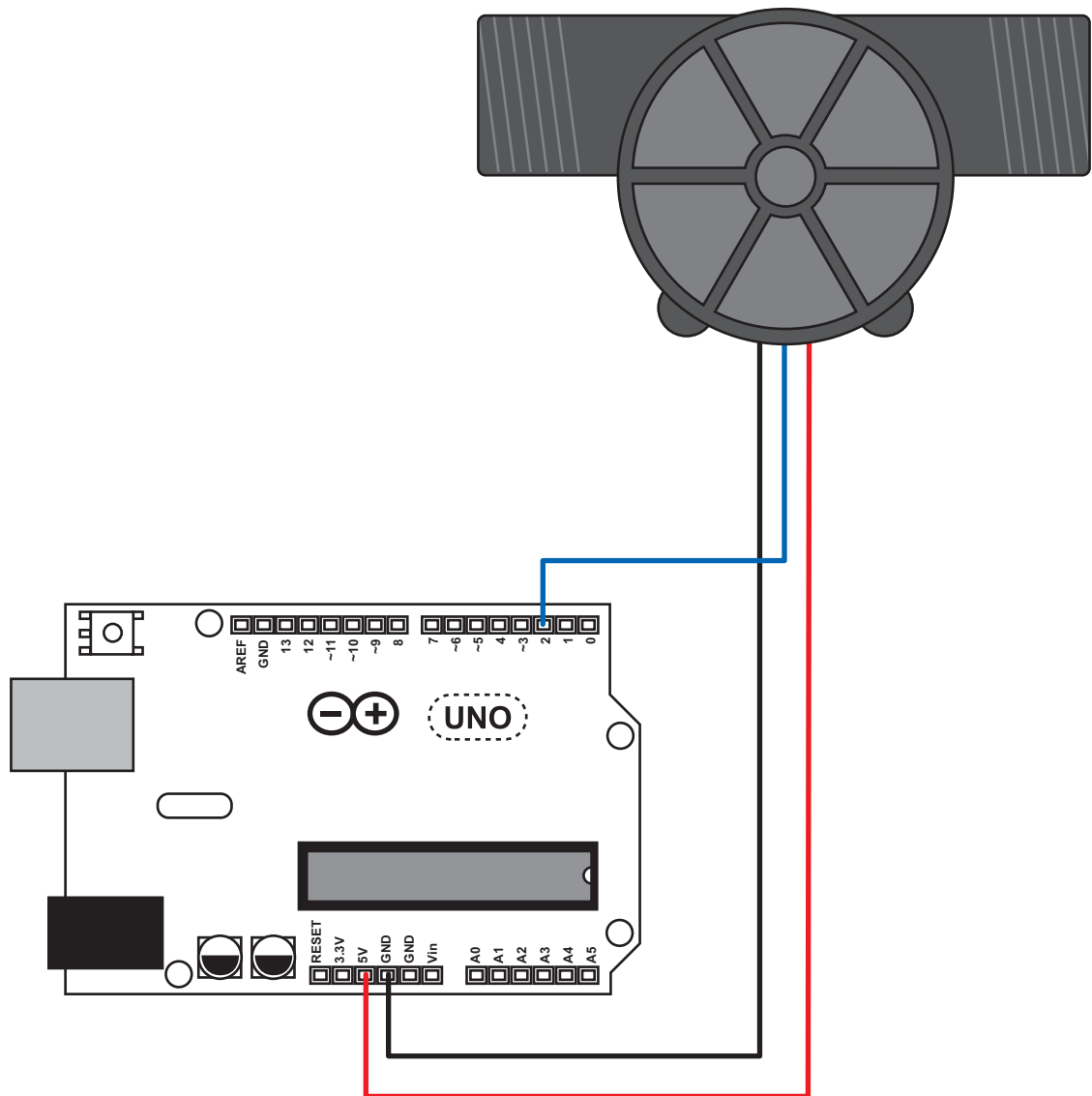
#### Introducción

Sencillo práctica para medir los Hz y el caudal.

#### Lista de materiales

- 1 Tarjeta Arduino LOG 8431
- 1 Cable USB LOG 7509
- 1 Sensor de caudal LOG
- 3 Latiguillos board macho-macho LOG 7519

#### Montaje en placa board



## D. PRÁCTICAS

### Programa con processing

```
/* Sensor de Caudal YF-S201*/

const int sensorPin = 2; //Sensor conectado en el pin 2
volatile int NumPulsos; //variable para la cantidad de pulsos recibidos
const float factorK = 7.5; //para convertir de frecuencia a caudal

const int medicion_intervalo = 2500;

//---Función que se ejecuta sin interrupción---
void ContarPulsos()
{
    NumPulsos++;
}

float ObtenerFrecuencia()
{
    NumPulsos = 0;    //Ponemos a 0 el número de pulsos

    interrupts(); //Habilitamos las interrupciones
    delay(medicion_intervalo); //muestra de 1 segundo
    noInterrupts(); //Deshabilitamos las interrupciones

    //Hz(pulsos por segundo)
    return (float)NumPulsos * 1000 / medicion_intervalo; }

void setup()
{
    Serial.begin(9600);
    attachInterrupt(digitalPinToInterrupt(sensorPin), ContarPulsos,
    RISING);
}

void loop()
{
    //obtenemos la frecuencia de los pulsos en Hz
    float frecuencia = ObtenerFrecuencia();
    //calculamos el caudal en L/m
    float caudal_Lmin = frecuencia / factorK;

    //-----Enviamos por el puerto serie-----
    Serial.print("Frecuencia: ");
    Serial.print(frecuencia, 0);
    Serial.print(" (Hz)\tCaudal: ");
    Serial.print(caudal_Lmin, 3);
    Serial.println(" (L/min)");
}
```



**MICRO-LOG TECNOLOGÍA Y SISTEMAS, S.L.**

C/ Andrés Obispo, 37, 5º - 28043 Madrid

Telf. 91 759 59 10

[pedidos@microlog.es](mailto:pedidos@microlog.es)

[www.microlog.es](http://www.microlog.es)