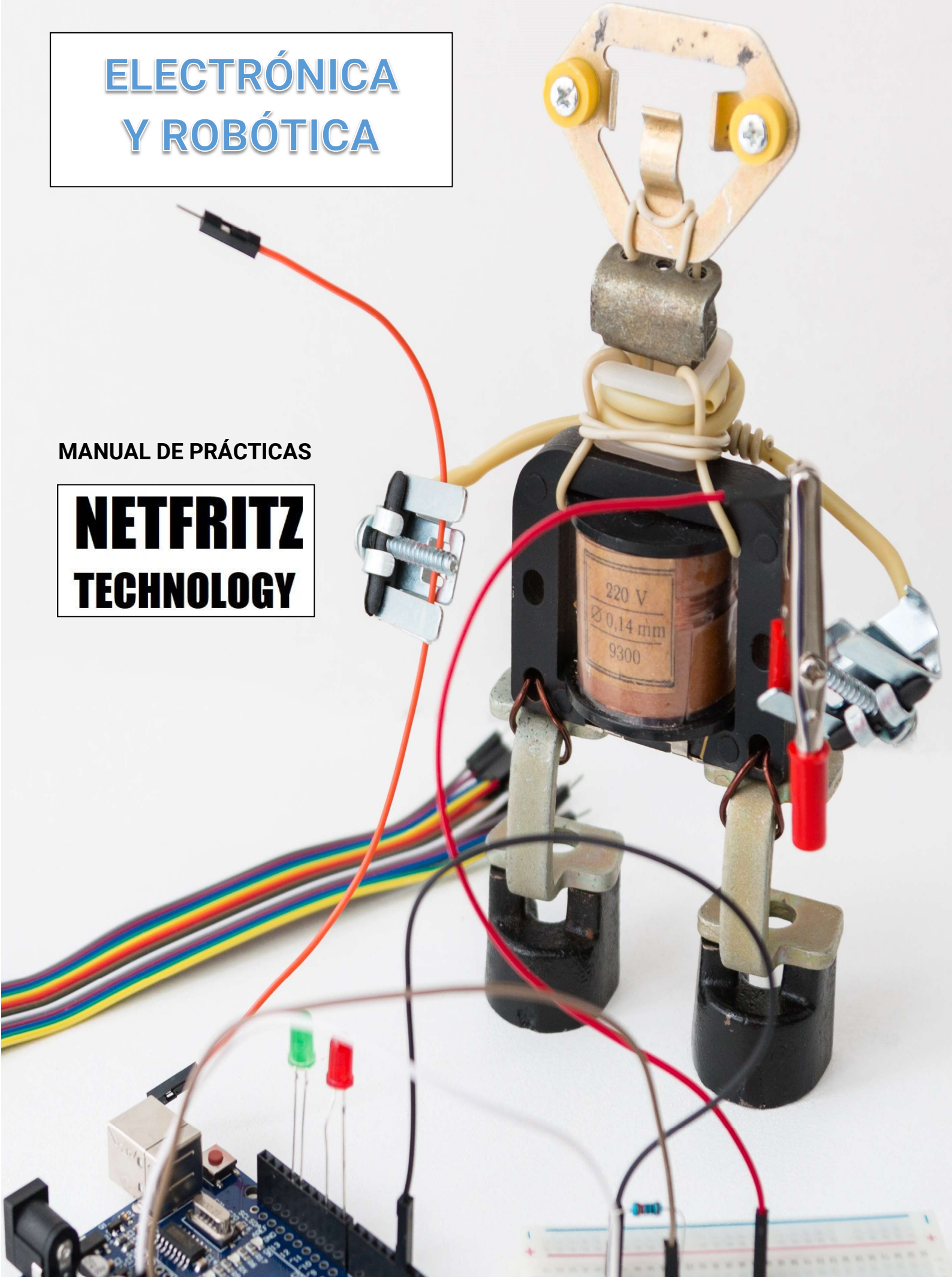
ELECTRÓNICA Y ROBÓTICA

MANUAL DE PRÁCTICAS

NETFRITZ TECHNOLOGY



TECNOLOGÍA SIGLO XXI

JUSTIFICACIÓN: Debido a la velocidad con la que surgen y evolucionan las nuevas tecnologías digitales, resulta imposible actualizarlas en los currículos con la agilidad que se requiere. Por este motivo se incorpora un listado de actividades prácticas y seminarios en las asignaturas de Tecnología, donde los alumnos recibirán y aplicarán estos nuevos conocimientos.

BLOQUE I: HARDWARE, SOFTWARE, FIRMWARE, REDES, PROGRAMACIÓN, RESPONSIVE

1. HARDWARE: montaje de un ordenador/PC, componentes, diagnóstico y reparación.
2. SOFTWARE: formateo de disco duro (HDD), instalación Sistema Operativo, Drivers, MS-DOS.
3. FIRMWARE: Placa Base, configuración de BIOS. Flashear BIOS, Firmwares binarios.
4. RED LOCAL: configuración TCP/UDP, grupos de trabajo, IP interna/pública. Servidores DNS. Dispositivos.
5. RED GLOBAL: Servidores Web en Internet, Arquitectura LAMP (Linux, Apache, MySQL, PHP).
6. PROGRAMACIÓN (I) WEB: HTML, CSS, Otros (JavaScript, Flash, Ajax…).
7. PROGRAMACIÓN (II) RESPONSIVE: HTML5, CSS3, adaptación a dispositivos táctiles y móviles.
8. *SEMINARIO (I): ORDENADORES CUÁNTICOS. LA 2ª REVOLUCIÓN CUÁNTICA.*

BLOQUE II: APLICACIONES DE ESCRITORIO, APP TELÉFONOS MÓVILES

9. APLICACIONES DE OFICINA: Office, OpenOffice.
10. APLICACIONES DE DIBUJO: Paint, Gimp, Corel Draw.
11. APLICACIONES DE ANIMACIÓN DE CONTENIDOS: Articulate Storyline.
12. OTRAS APLICACIONES: multimedia (audio/vídeo), compresores, conversores, códigos QR, API.
13. PROGRAMACIÓN (III) MOBILE, ANDROID STUDIO: Desarrollo APK Play Store de Google.
14. PROGRAMACIÓN (IV) MOBILE, XCODE: Lenguaje Swift. Desarrollo APP Store de Apple.
15. *SEMINARIO (II): OPEN SOURCE. CÓDIGO ABIERTO. LICENCIA GPL. SOFTWARE LIBRE.*

BLOQUE III: CIBERSEGURIDAD

16. AUDITORÍA WIFI: chequeo de la seguridad de redes inalámbricas.
17. INCIBE: prácticas recomendadas en materia de ciberseguridad.
18. CÓDIGOS MALICIOSOS: Virus, Gusanos y Troyanos. Builder RAT, Editor Hexadecimal.
19. SEGURIDAD EN SERVIDORES: puertos, HTTP y HTTPS, Certificado SSL, Sistemas SPF y DKIM.
20. *SEMINARIO (III): DEEP WEB. INTERNET PROFUNDO. LA WEB OSCURA. TOR.*

BLOQUE IV: ELECTRÓNICA Y ROBÓTICA

21. PRÁCTICAS DE ELECTRÓNICA Y ROBÓTICA

21. ELECTRÓNICA Y ROBÓTICA

DESCARGA LA GUÍA COMPLETA DE LAS PRÁCTICAS DE ELECTRÓNICA Y ROBÓTICA (PDF).

https://netfritz-technology.com/robotics/GUIA-TECH-ELECTRONICA-ROBOTICA.pdf

NIVEL 0: ELECTRÓNICA BÁSICA.

01– Introducción a los conceptos básicos. 3
02– Instrumentos de Medida: Óhmetros, Voltímetros, Amperímetros y Osciloscopios. 7
03– Montaje y circuito de 1 LED. 10
04– Montaje y circuito de 2 LEDS en serie. 15
05– Montaje y circuito de 2 LEDS en paralelo. 17

NIVEL 1: LEDS, POTENCIÓMETROS, FOTORRESISTENCIAS, TRANSISTORES, ZUMBADORES, PULSADORES.

06– Componentes Nivel 1. 22
07– Placa Microcontroladora. Lenguaje C++. 23
08– Encender y apagar 1 LED. 28
09– El Semáforo. 31
10– El Coche Fantástico. 35
11– El LED RGB. 41
12– El Potenciómetro. 46
13– La Fotorresistencia. 50
14– El Transistor. 54
15– El Zumbador. Star Wars. 57
16– El Pulsador. La pistola de Rayos Cósmicos. 63

NIVEL 2: SENSORES, MOTORES, DISPLAYS, LCD.

17– Componentes Nivel 2. 68
18– Sensor de proximidad. Aparcando el coche. 69
19– Sensor de temperatura. El termómetro digital. 71
20– El Reloj-Digital. 75
21– Registros de desplazamientos. 79
22– Pantalla LCD. 82
23– Control de motores con un transistor MOSFET. 84
24– Control de motores mediante un Puente en H. 86
25– Control de servos de modelismo. 88
26– Sensores ópticos reflexivos y de transmisión. 90

NIVEL 3: MÓDULOS DE EXPANSIÓN, ROBOTS.

27– Motor DC controlado por bluetooth. 92
28– Juego del Pong + Juego del Ping-Pong. 95
29– El Fórmula 1 teledirigido. Controlando remotamente el coche con el teléfono móvil. 106
30– El Robot Cuadrúpedo controlado por wifi. 114

// ELECTRÓNICA BÁSICA

//01 NIVEL 0: INTRODUCCIÓN A LOS CONCEPTOS BÁSICOS.

ELECTRÓNICA

La electrónica es una de las tres ramas principales que forman la robótica: electrónica, mecánica y programación.

LED

- Un LED es un diodo emisor de luz. Es un componente electrónico que al atravesarle una corriente eléctrica emite una luz de una longitud de onda (λ) concreta (color).

- Está formado por un cuerpo de plástico transparente o de color, y dos patas de distinta longitud. Dentro del cuerpo se encuentra el semiconductor al que están conectadas las patas o terminales.

- La pata más larga se corresponde con el ánodo (polo positivo), y la más corta con el cátodo (polo negativo). Hay que respetar esta polaridad para que el led emita luz.

INTENSIDAD DE CORRIENTE

El LED necesita un suministro de energía (intensidad de corriente) para su funcionamiento, y para ello usamos pilas colocadas dentro del portapilas. Éste alimenta todo nuestro circuito con dos cables: uno de color rojo que corresponde con el positivo, y otro negro correspondiente al negativo.

VOLTAJE

El voltaje de cada pila alcalina AA es de 1.5 voltios. Cada led, dependiendo de su color, tiene un voltaje diferente de funcionamiento, que puede variar entre los 1.6V y los 3V. Para reducir el voltaje entregado por el portapilas al de funcionamiento de nuestro LED, necesitaremos una resistencia.

RESISTENCIA

Una resistencia eléctrica es un componente que se opone al paso de la corriente eléctrica a través de ella. Dejan pasar la corriente en función de su valor, que se mide en Ohmios (Ω). Cuanto mayor sea la resistencia, menos corriente dejará pasar.

LEY DE OHM

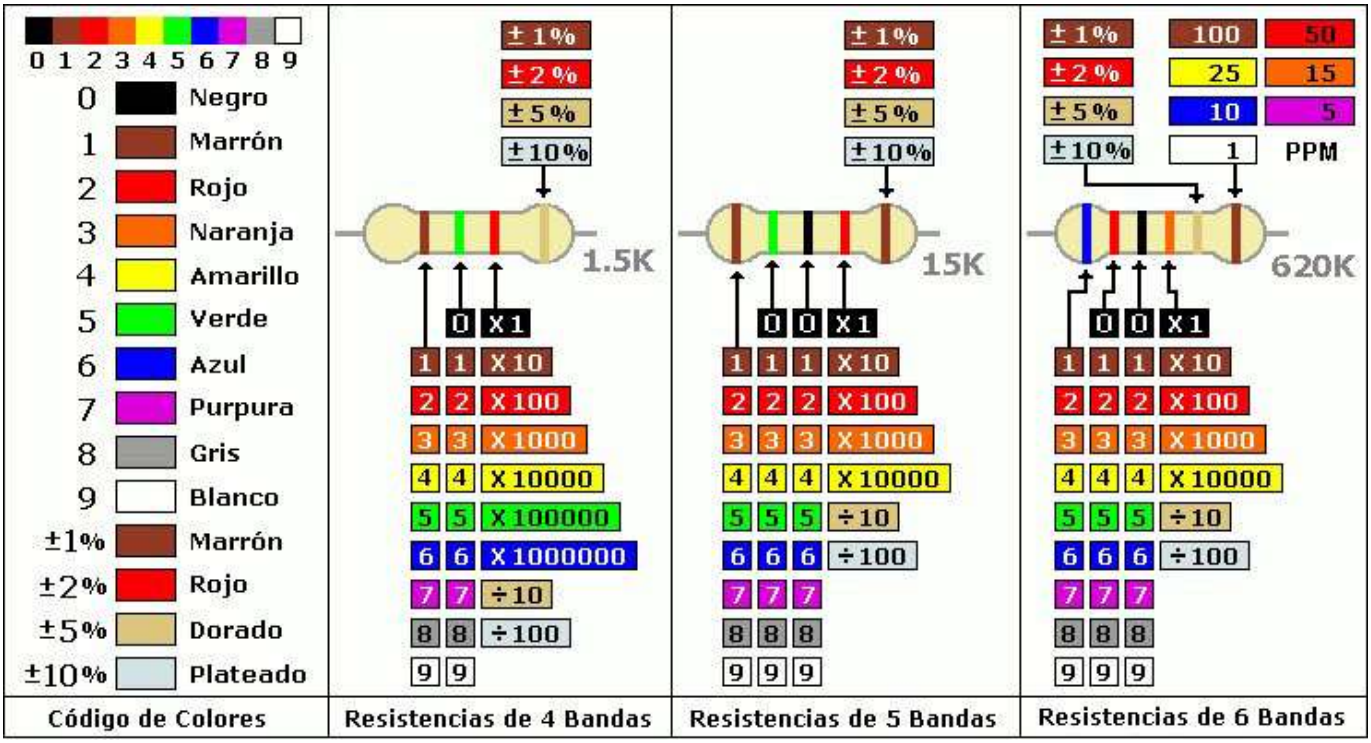
Para establecer la relación entre la intensidad de corriente (I), el voltaje (V) y la resistencia (R), utilizamos la Ley de Ohm:

$$I = V/R$$

Por ejemplo, para un LED rojo que funcione a 1.8V y tenga un consumo (intensidad) de 8mA, que esté alimentado por dos pilas alcalinas (3V), necesitaremos una resistencia de unos 150 Ω (Ohmios).

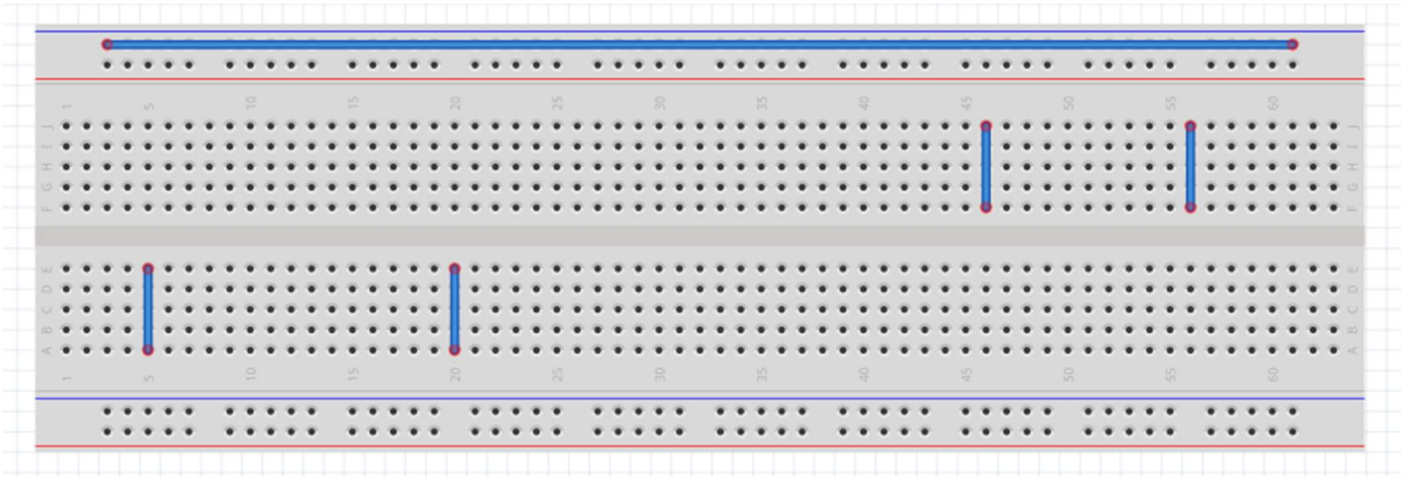
$$R = V / I = (3 - 1.8) / 0.008 = 150\ \Omega$$

Las resistencias utilizan un sistema de codificación basado en colores:



PLACA DE PROTOTIPOS

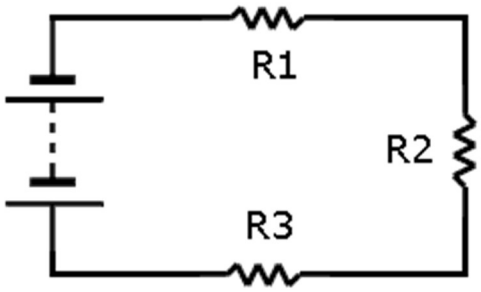
Para montar los circuitos utilizamos una Placa de Prototipos (placa de conexiones electrónicas), que nos ayuda a hacer conexiones rápidamente. Internamente están diseñadas de manera que ya tengan contactos conectados entre sí. Las filas superiores e inferiores de alimentación, marcadas en azul y rojo, están conectadas cada fila individualmente, mientras que las columnas del centro de la placa están conectadas de cinco en cinco contactos.



CIIRCUITOS EN SERIE Y EN PARALELO

CIRCUITOS EN SERIE

Los circuitos en serie se caracterizan por tener las resistencias conectadas en la misma línea existente entre los extremos de la batería o la pila, es decir, situados uno a continuación del otro. Por tanto, la corriente fluye por cada resistor uno tras otro. Si ponemos un ejemplo utilizando las centrales hidráulicas, podemos decir que dos depósitos de agua están conectados en serie si la salida de uno de ellos se conecta a la entrada del segundo. Otro ejemplo donde aparece la conexión en serie puede ser las baterías eléctricas, ya que están formadas por varias pilas que se encuentran conectadas en serie para alcanzar el voltaje necesario.

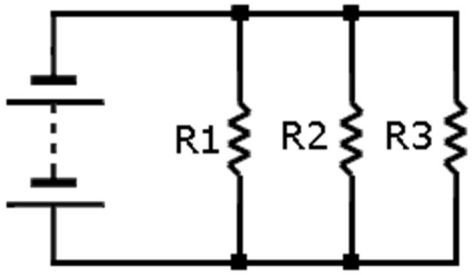


CIRCUITO SERIE

En un circuito en serie, la intensidad de corriente (I) que circula por todos los componentes del circuito, es la misma. El voltaje (V) de funcionamiento de varios LEDS en un circuito en serie, será la suma del voltaje de cada uno de ellos, de modo que se cumple que V = V1 + V2 + V3.

CIRCUITOS EN PARALELO

Los circuitos en paralelo se caracterizan por tener conectadas varias vías alineadas paralelamente entre sí, de tal forma que cada vía tiene una resistencia y estas vías están conectadas por puntos comunes, tal y como podemos apreciar en la siguiente imagen.



CIRCUITO PARALELO

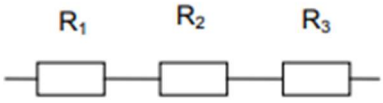
En un circuito en paralelo, hay componentes o conjuntos de componentes a los que se aplica el mismo voltaje. La intensidad (I) que genera el portapilas, se repartirá en I1, I2 e I3, de modo que se cumpla que I = I1 + I2 + I3.

ASOCIACIÓN DE RESISTENCIAS

Existen tres tipos de asociación:

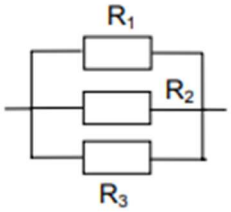
a. SERIE.

$$R_T = R_1 + R_2 + R_3 + \ldots$$

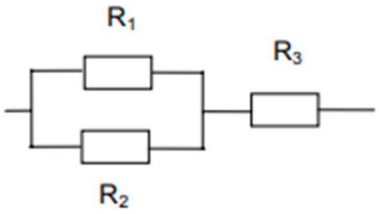


b. PARALELO.

$$\frac{1}{R_T} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \ldots$$

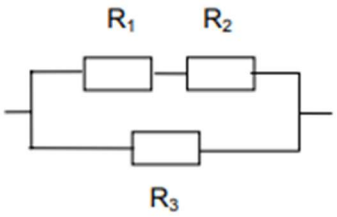


c. MIXTO.



Pasos a seguir:

1.- Paralelo R_1 II R_2 = Ra
2.- Serie Ra + R_3



Pasos a seguir:

1.- Serie: $R_1 + R_2$ = Rb
2.- Paralelo: Rb II R_3

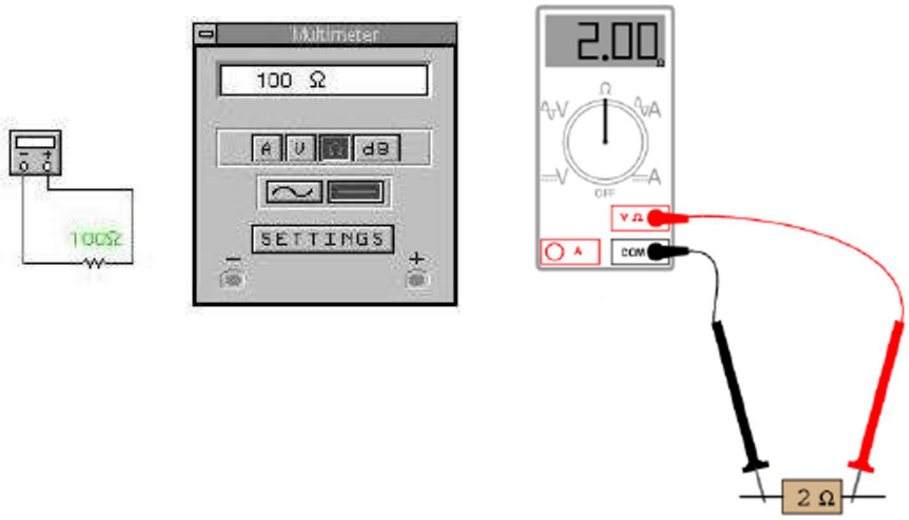
//02 NIVEL 0: MEDIDAS DE TENSIÓN, INTENSIDAD Y RESISTENCIA

Instrumentos de Medida: Óhmetros, Voltímetros, Amperímetros y Ocsiloscopios.

Para medir las tres magnitudes eléctricas se emplean distintos aparatos de medida y para cada uno de ellos hay que tener en cuenta ciertas consideraciones.

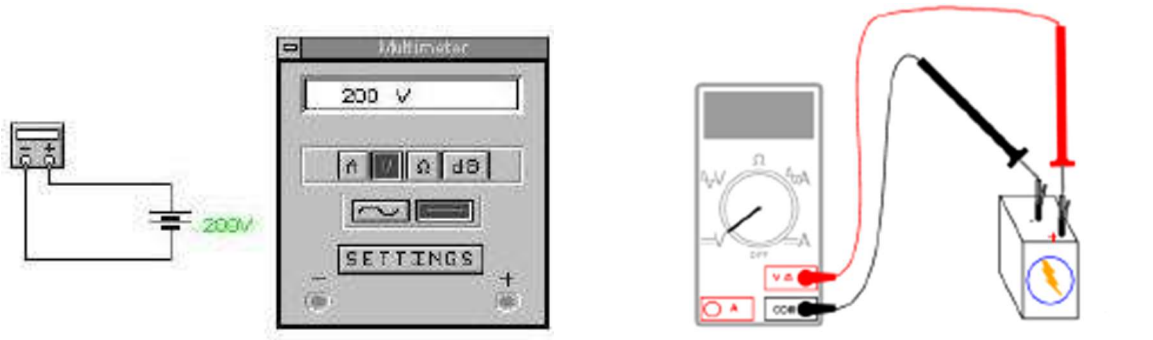
Los diferentes modos de funcionamiento de un multímetro nos permitirán utilizarlo como Óhmetro (resistencias), Voltímetro (voltajes) o Amperímetro (intensidades de corriente).

Medida de la resistencia.
La resistencia se mide con un óhmetro, y se conecta entre los dos extremos de la resistencia a medir, estando ésta desconectada del circuito eléctrico.

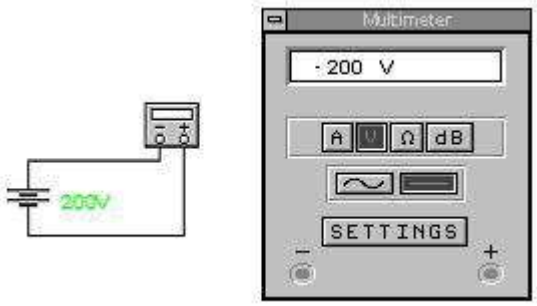


Medida de la tensión (voltaje).
La tensión se mide con un voltímetro y se conecta en paralelo a los dos puntos donde se desea medir la tensión.

El terminal positivo del voltímetro se conecta al terminal positivo de la tensión.



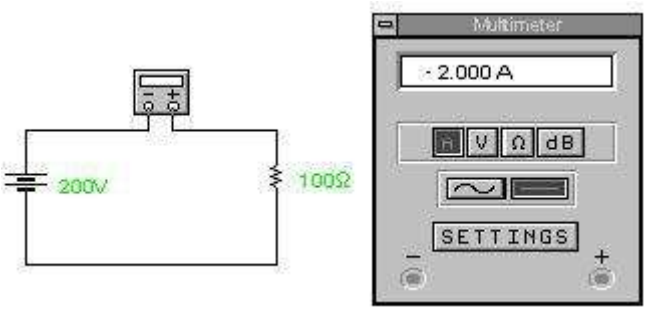
Si la conexión se realiza al revés la medida es de signo negativo.



Medida de la intensidad.

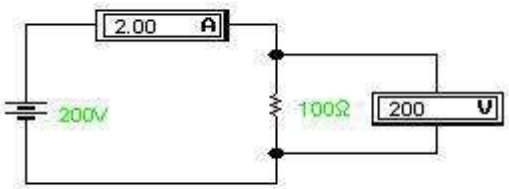
La intensidad se mide con un amperímetro que se intercala en serie en el circuito donde se quiere medir la intensidad.

Aquí también hay que tener en cuenta la polaridad de la conexión.

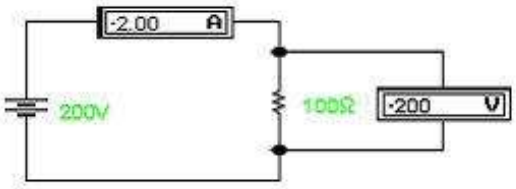


En el siguiente circuito se ha medido la tensión e intensidad.

a) Conexión correcta de polaridades:



b) Conexión incorrecta de polaridades:



Consideraciones importantes a tener en cuenta.

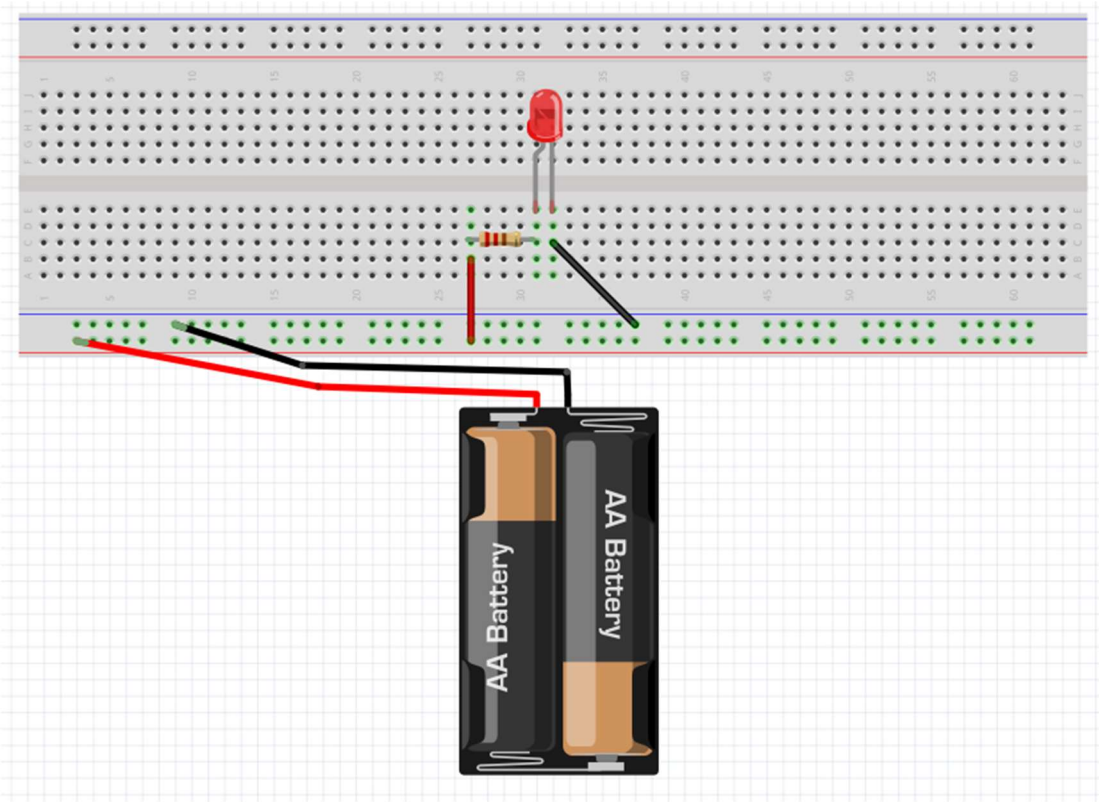
- Para medir la resistencia de un circuito eléctrico, se tiene que realizar sin tensión, si no es así el óhmetro puede estropearse.
- Para medir la tensión, el voltímetro se conecta en paralelo. Un voltímetro tiene una resistencia interna muy grande (en teoría infinita). Si éste se conecta en serie la resistencia del circuito será infinita y no circulará intensidad. No se corre riesgo de estropear el voltímetro, pero la medida será incorrecta.
- Para medir la intensidad, el amperímetro se conecta en serie. Un amperímetro tiene una resistencia interna muy pequeña (en teoría cero). Si éste se conecta en paralelo, la intensidad que circulará por el amperímetro será muy elevada (en teoría infinita), realmente lo que estamos haciendo es un cortocircuito. El amperímetro corre un serio riesgo de estropearse. La mayoría de los amperímetros llevan incorporado un fusible para protegerlos, aun así, si la intensidad del cortocircuito es muy elevada el amperímetro puede quedar inservible.

EL OSCILOSCOPIO

- Un osciloscopio es un instrumento de visualización electrónico para la representación gráfica de señales eléctricas que pueden variar en el tiempo. Es muy usado en electrónica de señal, frecuentemente junto a un analizador de espectro.
- Presenta los valores de las señales eléctricas en forma de coordenadas en una pantalla, en la que normalmente el eje X (horizontal) representa tiempos y el eje Y (vertical) representa tensiones. La imagen así obtenida se denomina oscilograma. Suelen incluir otra entrada, llamada "eje THRASHER" o "Cilindro de Wehnelt" que controla la luminosidad del haz, permitiendo resaltar o apagar algunos segmentos de la traza.
- Los osciloscopios, clasificados según su funcionamiento interno, pueden ser tanto analógicos como digitales, siendo el resultado mostrado idéntico en cualquiera de los dos casos, en teoría.

//03 1 LED

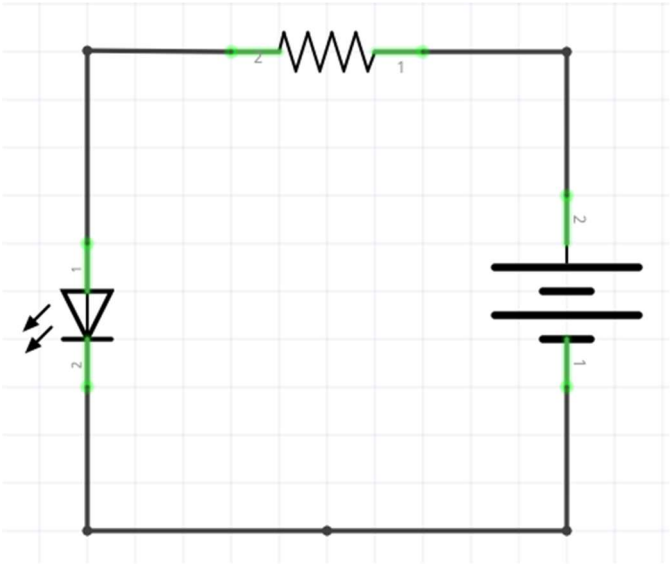
MONTAJE



MATERIALES

– Fuente de Tensión de 4.5voltios (3 pilas alcalinas de 1.5v cada una, tipo AA)
– LED Rojo: 10-20mA, 1.8-2-2v
– Resistencia de 330 Ω
– 4 cables M-M

CIRCUITO



CALCULA Y MIDE

Razonamiento para calcular la Resistencia que necesitamos:
– Queremos encender un LED que necesita una corriente de aproximadamente entre 10mA (0.01A) y 20mA para funcionar.
– El voltaje suministrado por la pila es de 4.5v, y el LED tiene un consumo de 1.8v, por lo tanto, los 2.7v de exceso deben consumirse en la Resistencia cuyo valor buscamos:

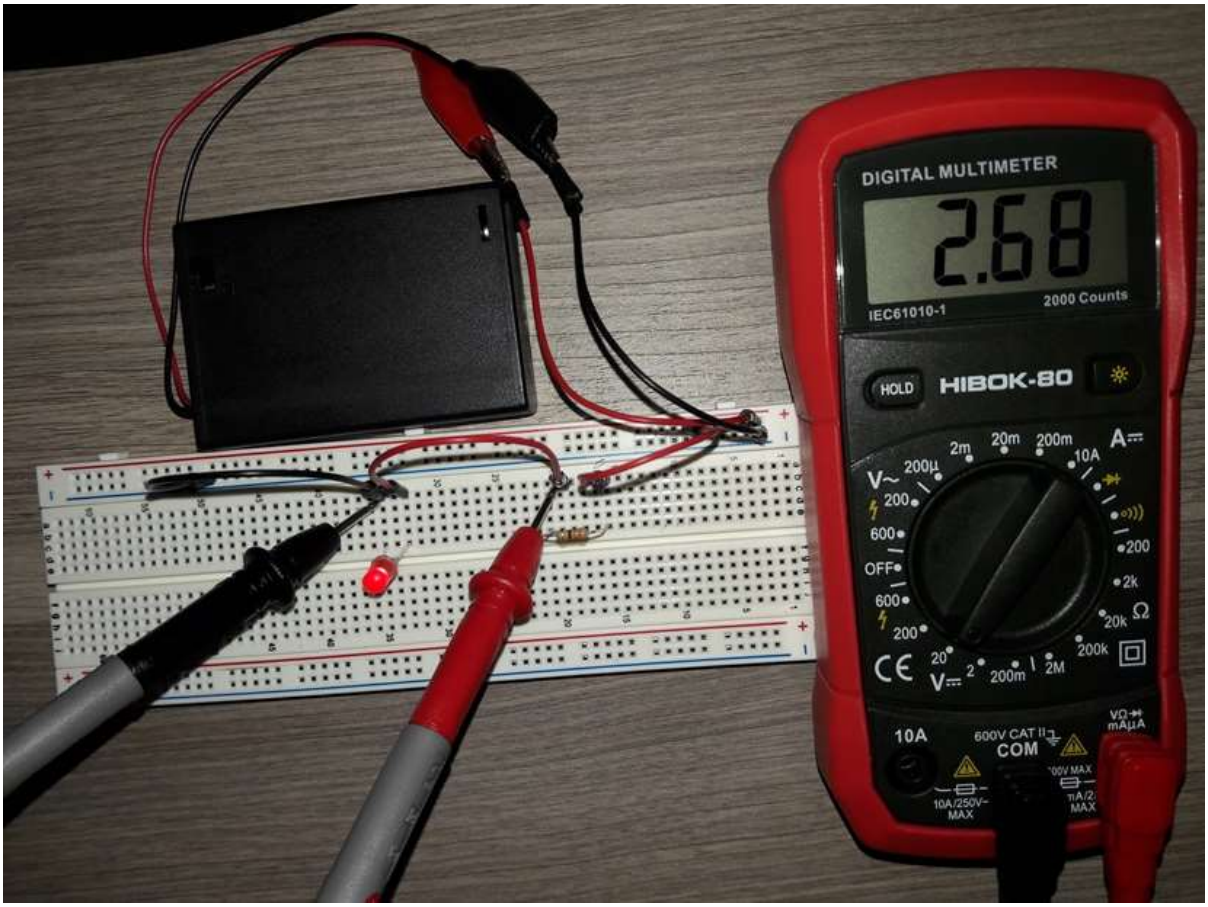
R= V / I = (4.5 – 1.8) / 0.01A = 270 Ω

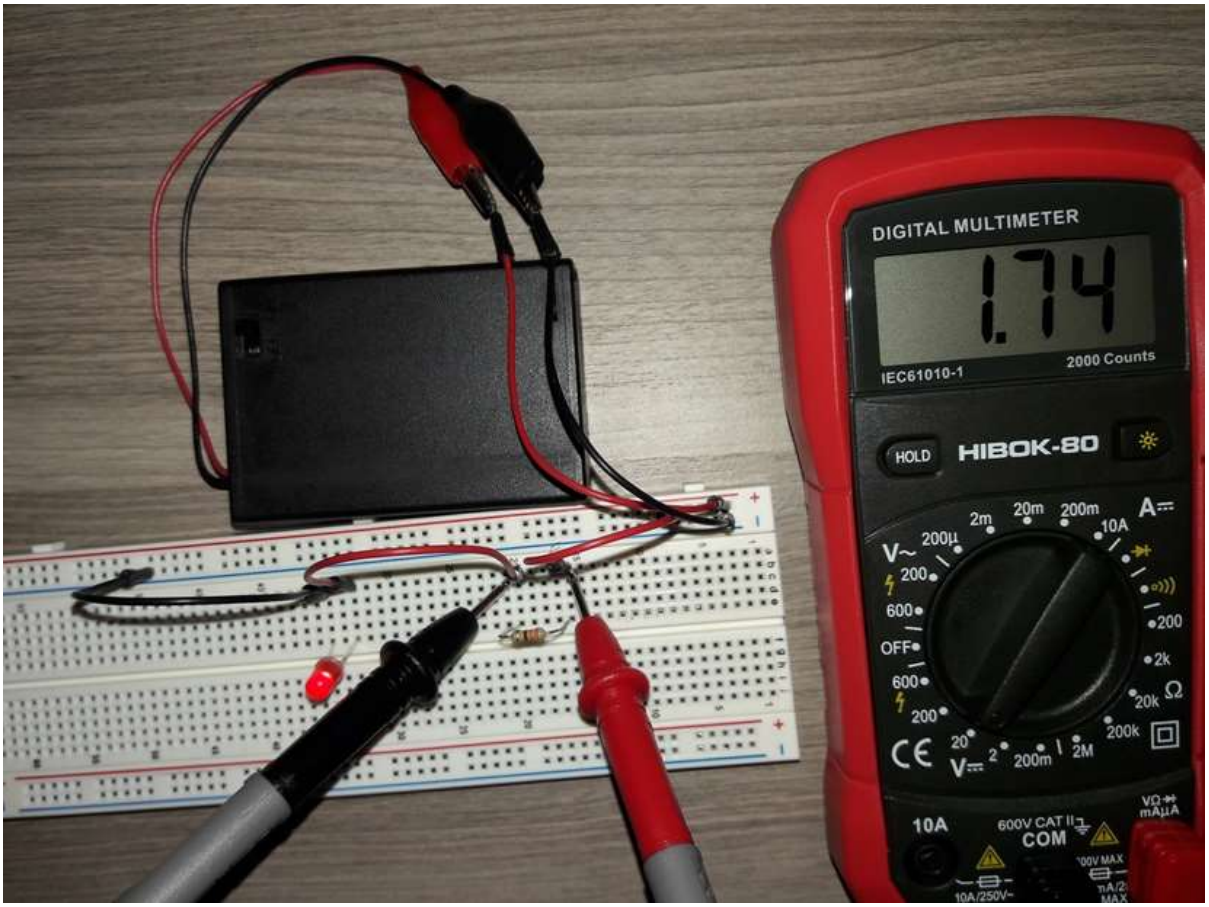
Cogemos la más parecida, en este caso 330 Ω.

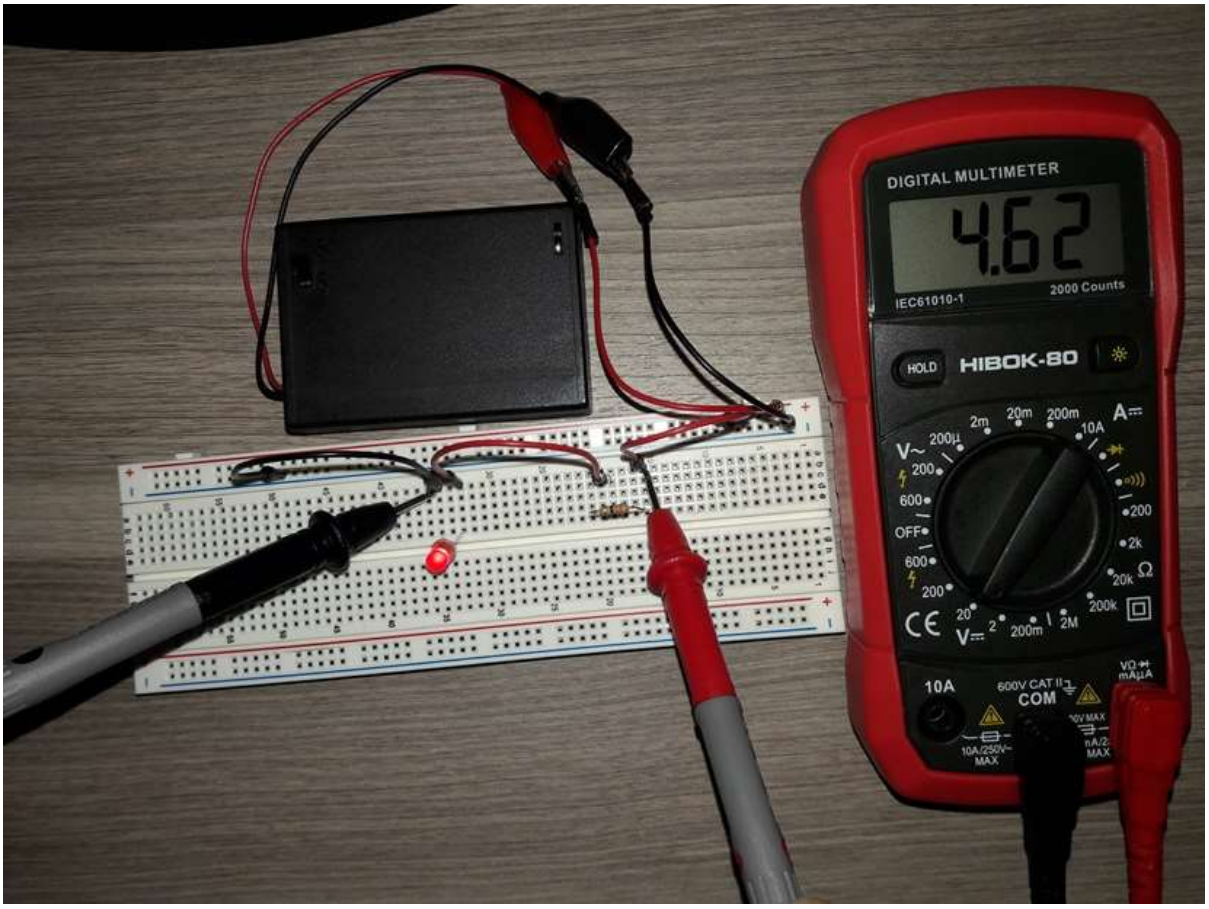
Calcula y mide:
Con los datos anteriores, prueba a realizar los cálculos analíticos y a medir con el voltímetro la diferencia de potencial de cada componente.
Solución: La intensidad de corriente I, es la misma por toda la malla.

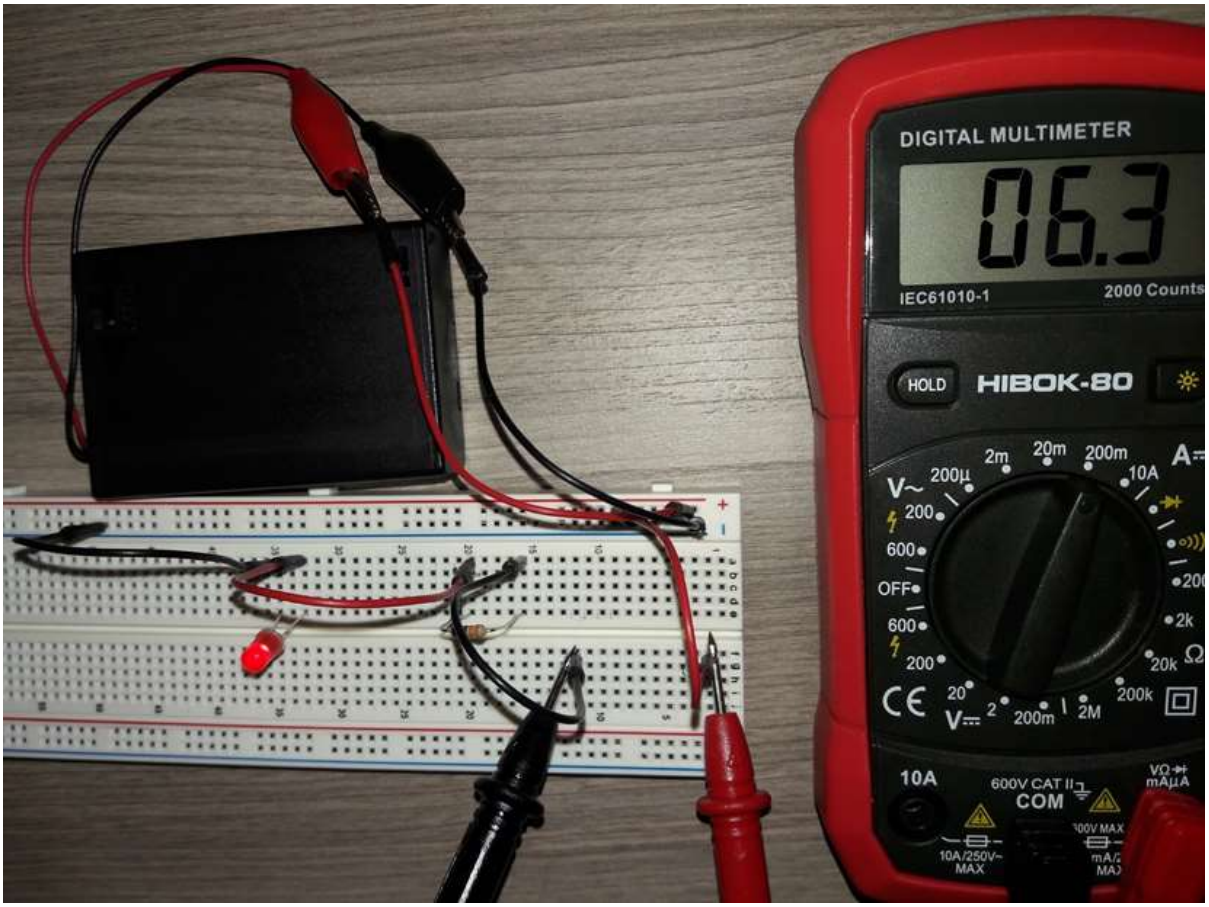
I = 8mA
V-resistencia = 2.7v
V-LED = 1.8v
V-Total = 4.5v

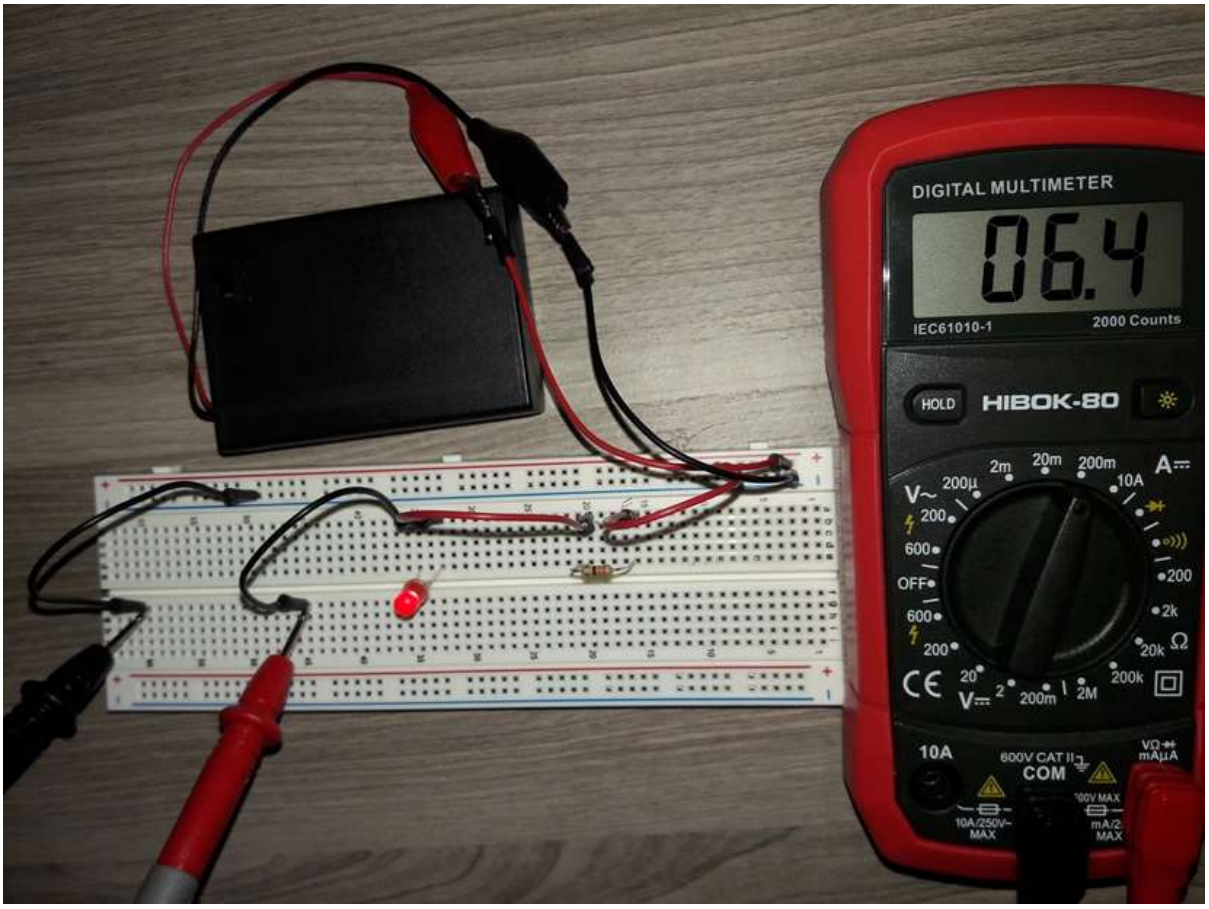
Utilizando el voltímetro conectado en paralelo con el circuito, podremos medir la diferencia de potencial (voltaje, tensión) en la resistencia, en el LED y en todo el circuito (fotos 1, 2 y 3). Dado que nuestro montaje es un circuito en serie, el potencial total será la suma de todos los potenciales. La intensidad de corriente I que mediremos con el multímetro en modo amperímetro conectado en serie con el circuito, sin embargo, será la misma por toda la malla (fotos 4 y 5).




DIGITAL MULTIMETER
1.74
IEC61010-1
2000 Counts
HOLD
HIBOK-80
10A
600V CAT II
COM

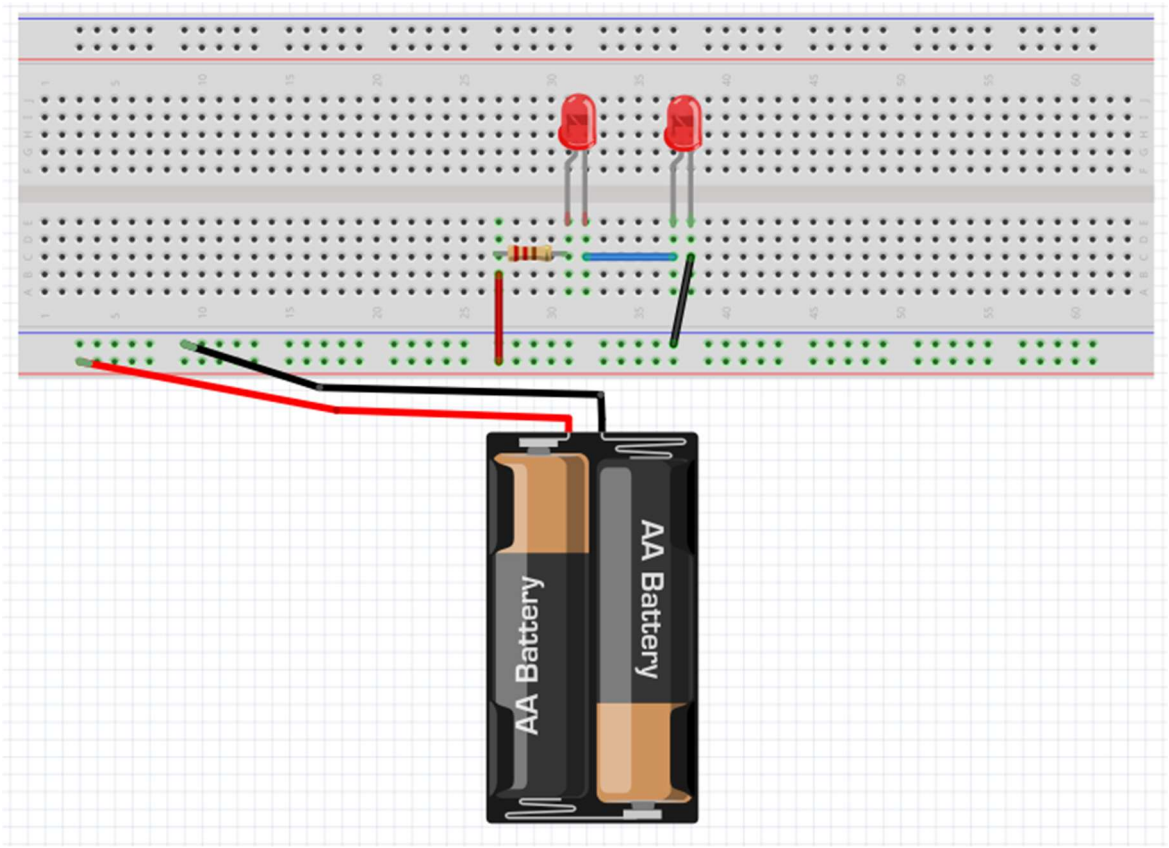

DIGITAL MULTIMETER
4.62
IEC61010-1
2000 Counts
HOLD
HIBOK-80
10A
600V CAT II
COM


DIGITAL MULTIMETER
06.3
IEC61010-1
2000 Counts
HOLD
HIBOK-80
10A
600V CAT II
COM


DIGITAL MULTIMETER
06.4
IEC61010-1
2000 Counts
HOLD
HIBOK-80
10A
600V CAT II
COM

//04 2 LEDS EN SERIE

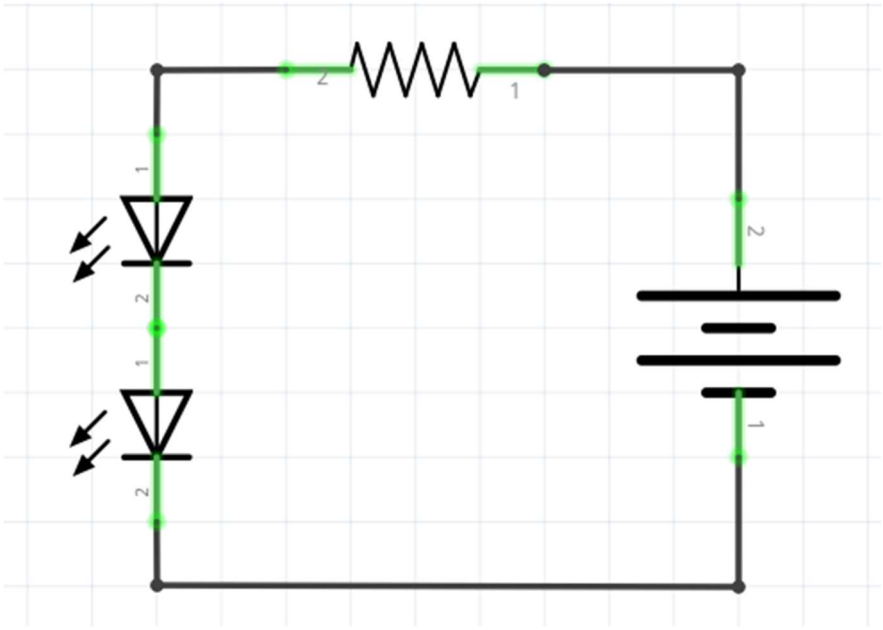
MONTAJE



MATERIALES

– Fuente de Tensión de 4.5voltios (3 pilas alcalinas de 1.5v cada una, tipo AA)
– 2 LEDS Rojos: 10-20mA, 1.8-2-2v
– 1 Resistencia de 100 Ω
– 5 cables M-M

CIRCUITO



CALCULA Y MIDE

Razonamiento para calcular la Resistencia que necesitamos:
– Queremos encender dos LEDS en serie que necesitan una corriente de aproximadamente entre 10mA (0.01A) y 20mA para funcionar.
– El voltaje suministrado por la pila es de 4.5v, y cada LED tiene un consumo de 1.8v, por lo tanto, los 0.9v de exceso deben consumirse en la Resistencia cuyo valor buscamos:

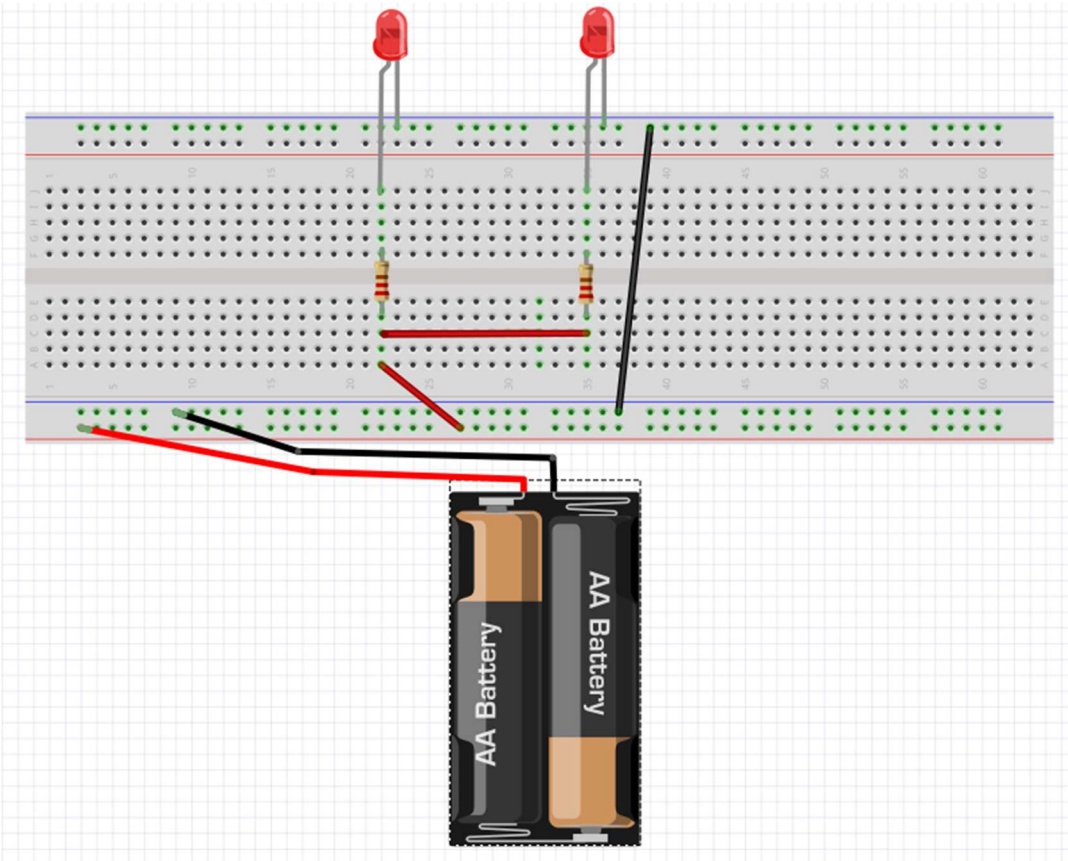
R= V / I = (4.5 – 1.8 – 1.8) / 0.01A = 90 Ω

Cogemos la más parecida, en este caso 100 Ω.

Calcula y mide:
Con los datos anteriores, prueba a realizar los cálculos analíticos y a medir con el voltímetro la diferencia de potencial de cada componente:
Solución: La intensidad de corriente I, es la misma por toda la malla.
I = 9mA
V-resistencia = 0.9v
V-LED1 = 1.8v
V-LED2 = 1.8v
V-Total = 4.5v

//05 2 LEDS EN PARALELO

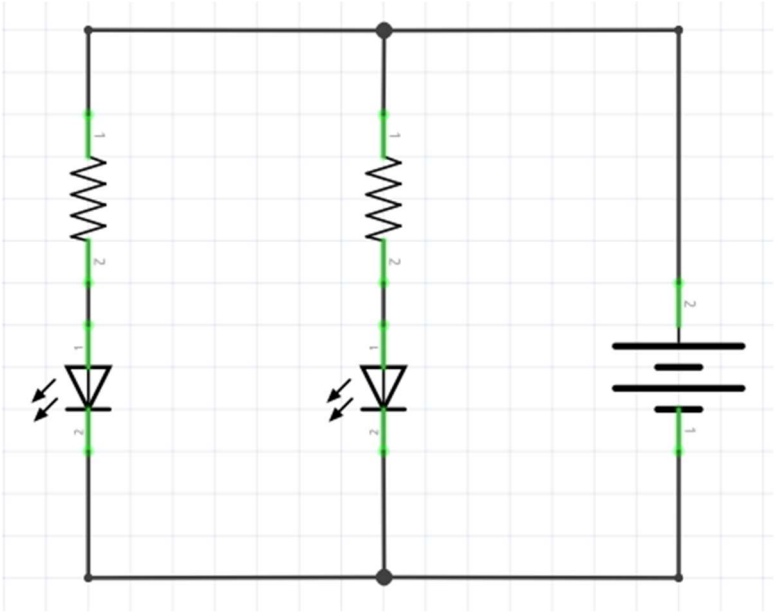
MONTAJE



MATERIALES

– Fuente de Tensión de 4.5voltios (3 pilas alcalinas de 1.5v cada una, tipo AA)
– 2 LEDS Rojos: 10-20mA, 1.8-2-2v
– 2 Resistencias de 330 Ω
– 5 cables M-M

CIRCUITO



CALCULA Y MIDE

Razonamiento para calcular las corrientes de cada rama:
– La corriente I cuando llega a un nudo, se reparte por los diferentes cables. El camino por el que menos resistencia encuentre, será por el que se vaya la mayor parte de la corriente.
– Para calcular la corriente I que hay entrando en el nudo, hallamos la resistencia equivalente del circuito, en este caso, sumándolas en paralelo:

1/R= 1/330 + 1/330 de donde la resistencia equivalente R = 165 Ω

– Conocida la resistencia equivalente del circuito y el voltaje que suministra la fuente de tensión (pila), podemos obtener la corriente I que entra en el nudo, y que se reparte posteriormente por las dos ramas.

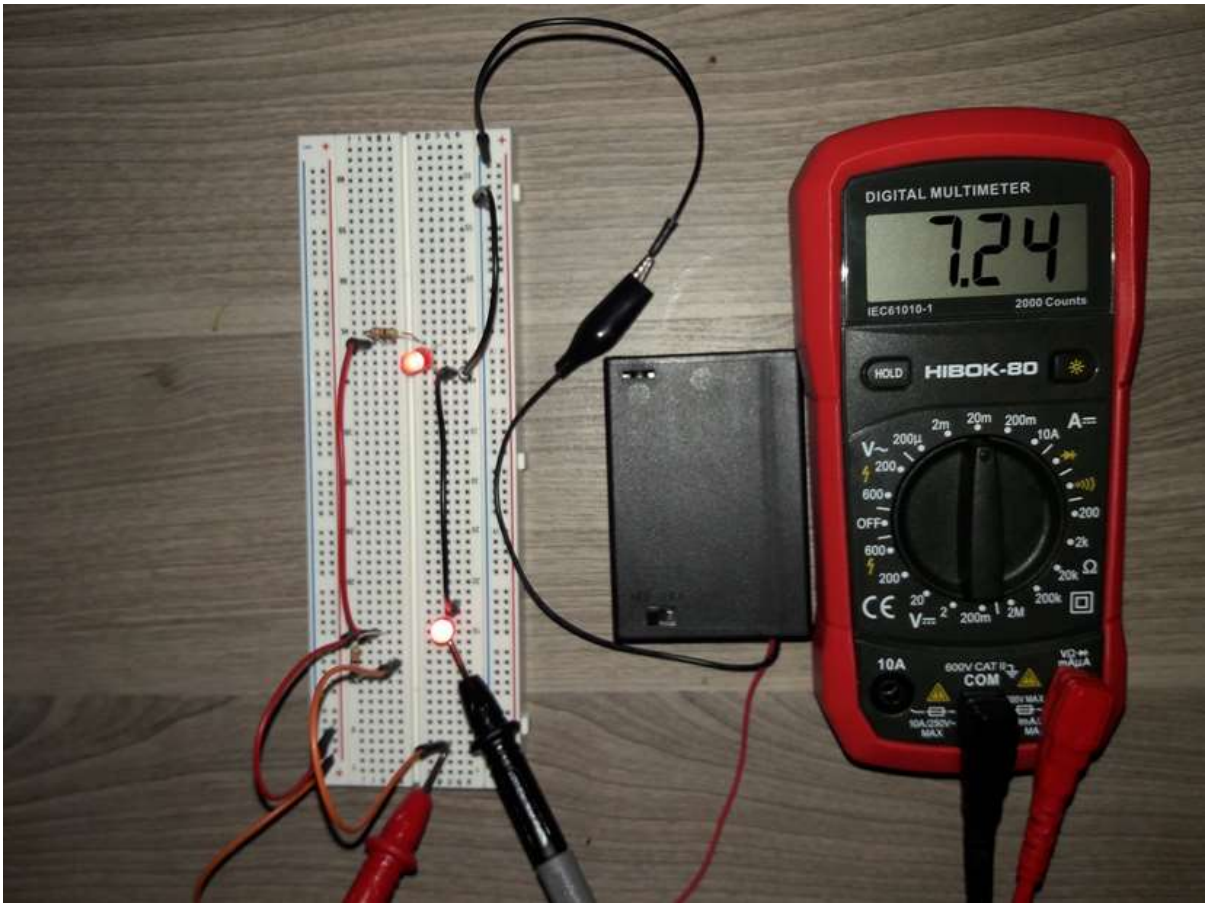
I= (4.5 – 1.8) / 165 = 2.7v / 165Ω = 16mA

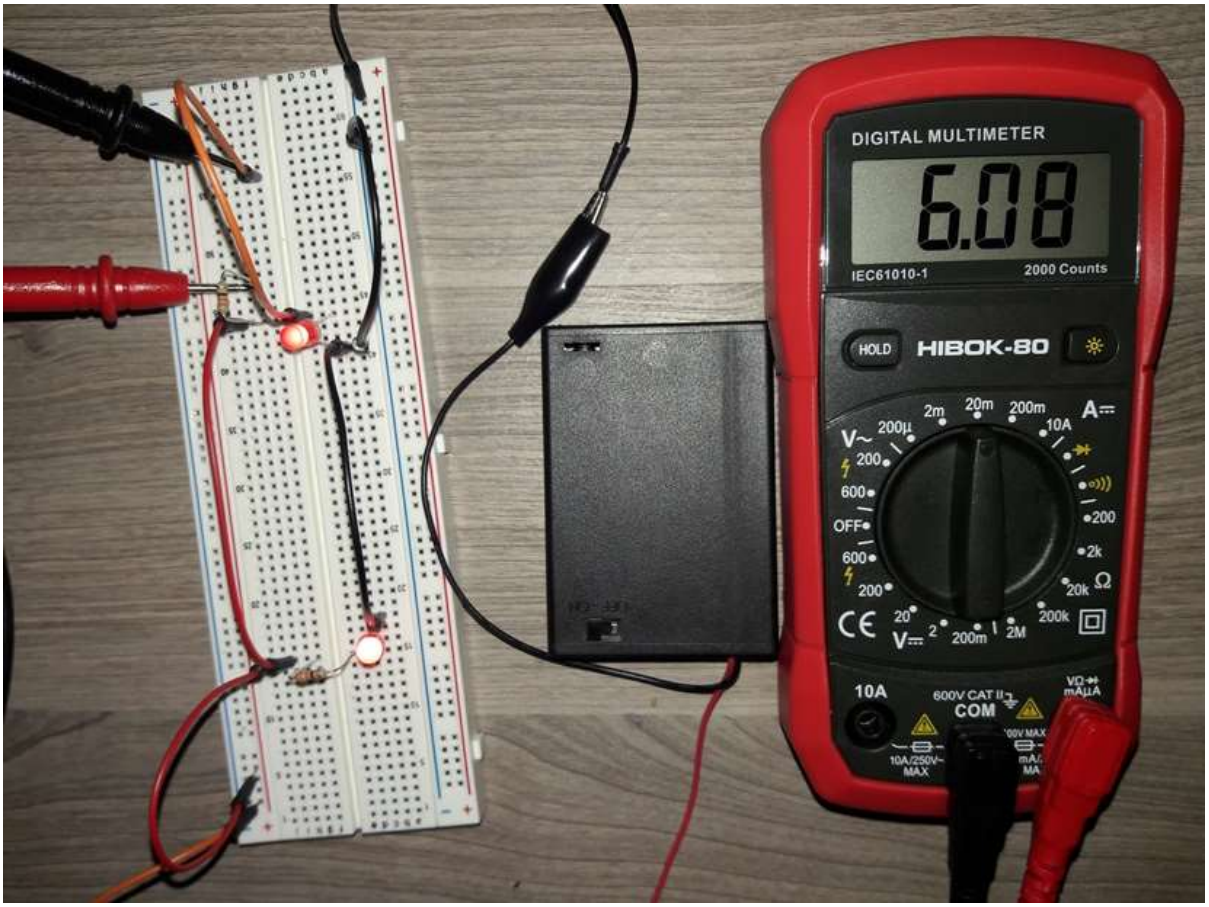
– Esta corriente I se reparte dependiendo de la resistencia que encuentra en cada una de las ramas.
– Dado que ambas ramas tienen la misma resistencia, la corriente se distribuye por igual en ambas.

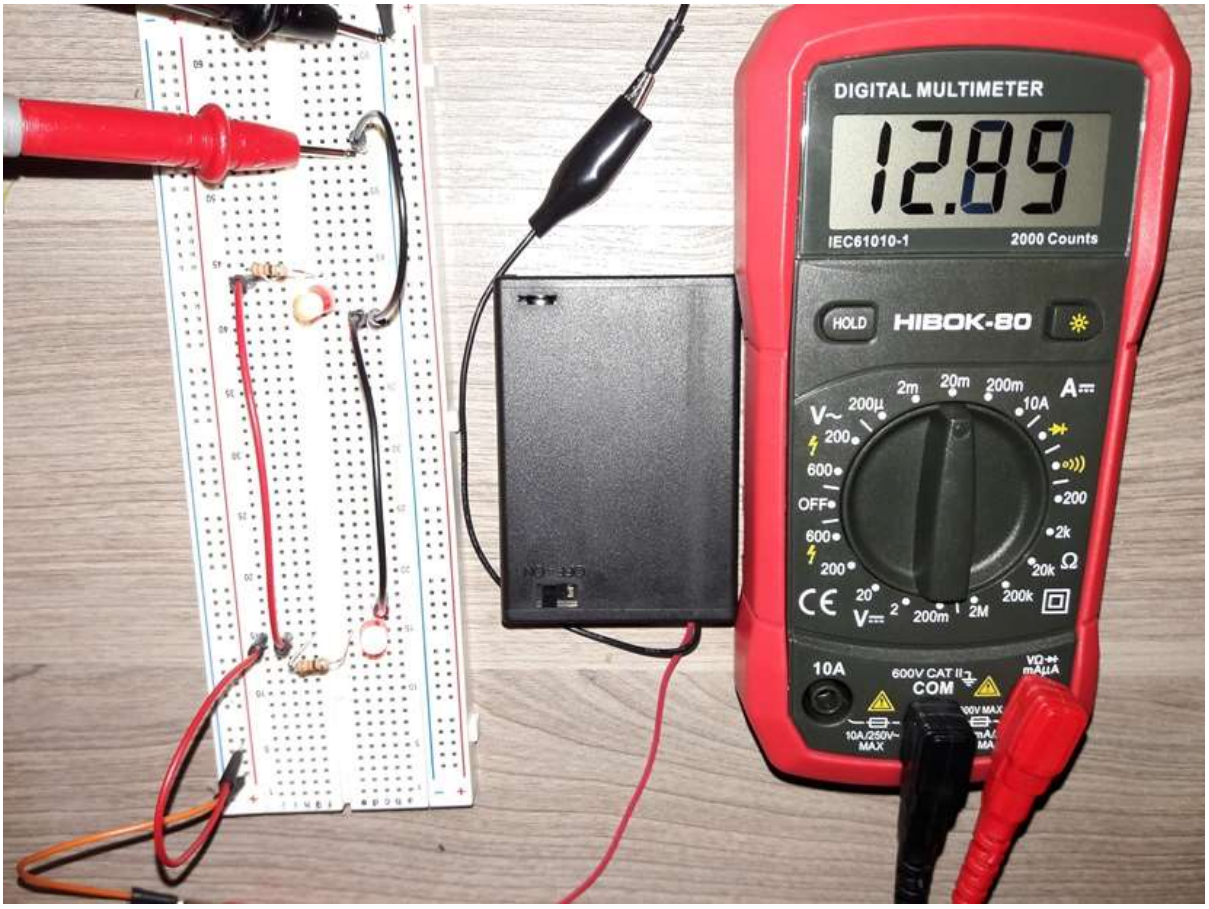
Calcula y mide:
Comprueba que la diferencia de potencial entre dos puntos es la misma, independientemente del camino seguido, y para ello:
Mide la diferencia de potencial desde la pata positiva de la primera resistencia, hasta la negativa también del primer diodo.
Mide la diferencia de potencial desde la pata positiva de la segunda resistencia, hasta la negativa también del segundo diodo.
Solución: La diferencia de potencial (voltaje) es la misma.

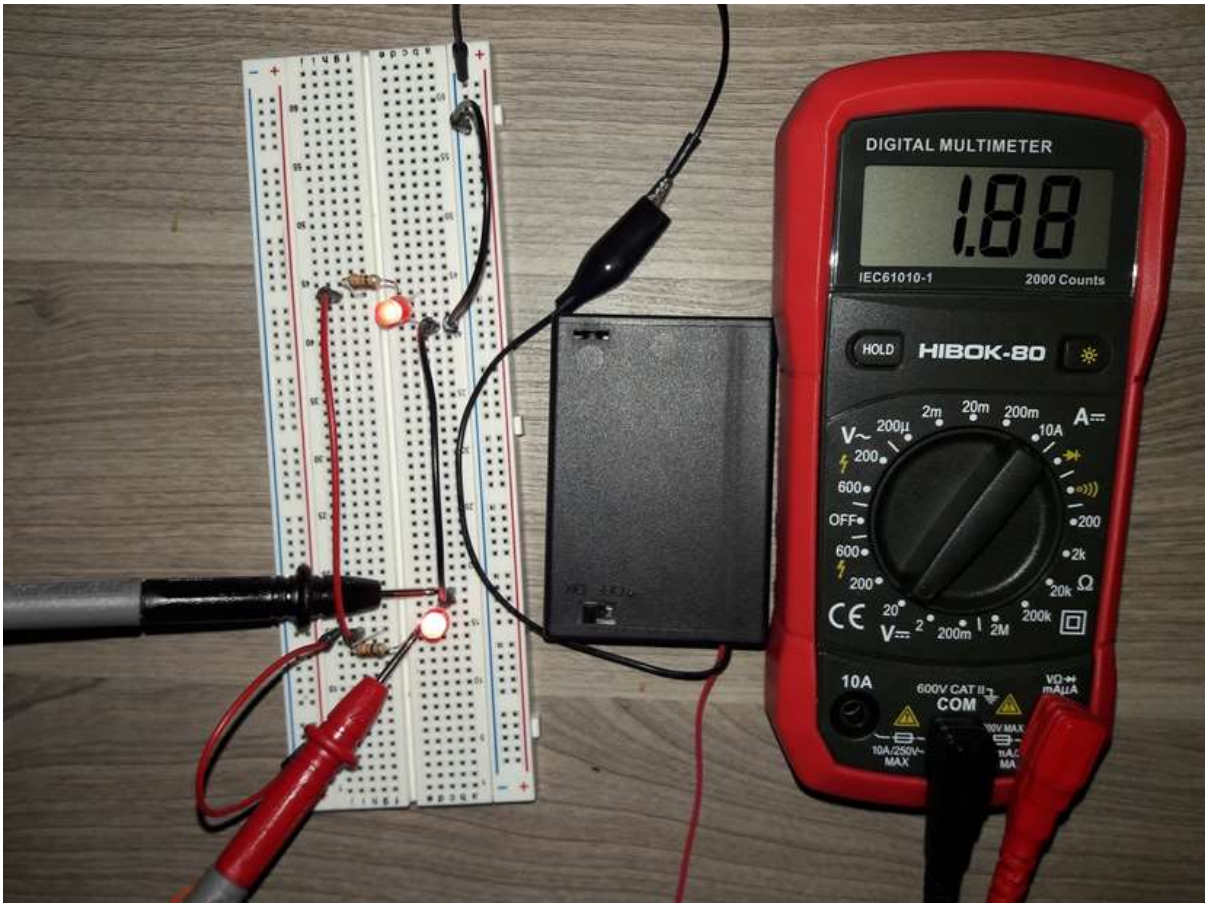
I1 = V/R1 = 2.7v / 330 Ω = 8mA I2 = V/R2 = 2.7v / 330 Ω = 8mA I = V/R = 2.7v / 165 Ω = 16mA
V-resistencia1 = I1 * R1 = 8mA * 330 Ω = 2.7v V-LED1 = 1.8v V-Total-resistencia1-LED1 = 4.5v
V-resistencia2 = I2 * R2 = 8mA * 330 Ω = 2.7v V-LED2 = 1.8v V-Total-resistencia2-LED2 = 4.5v

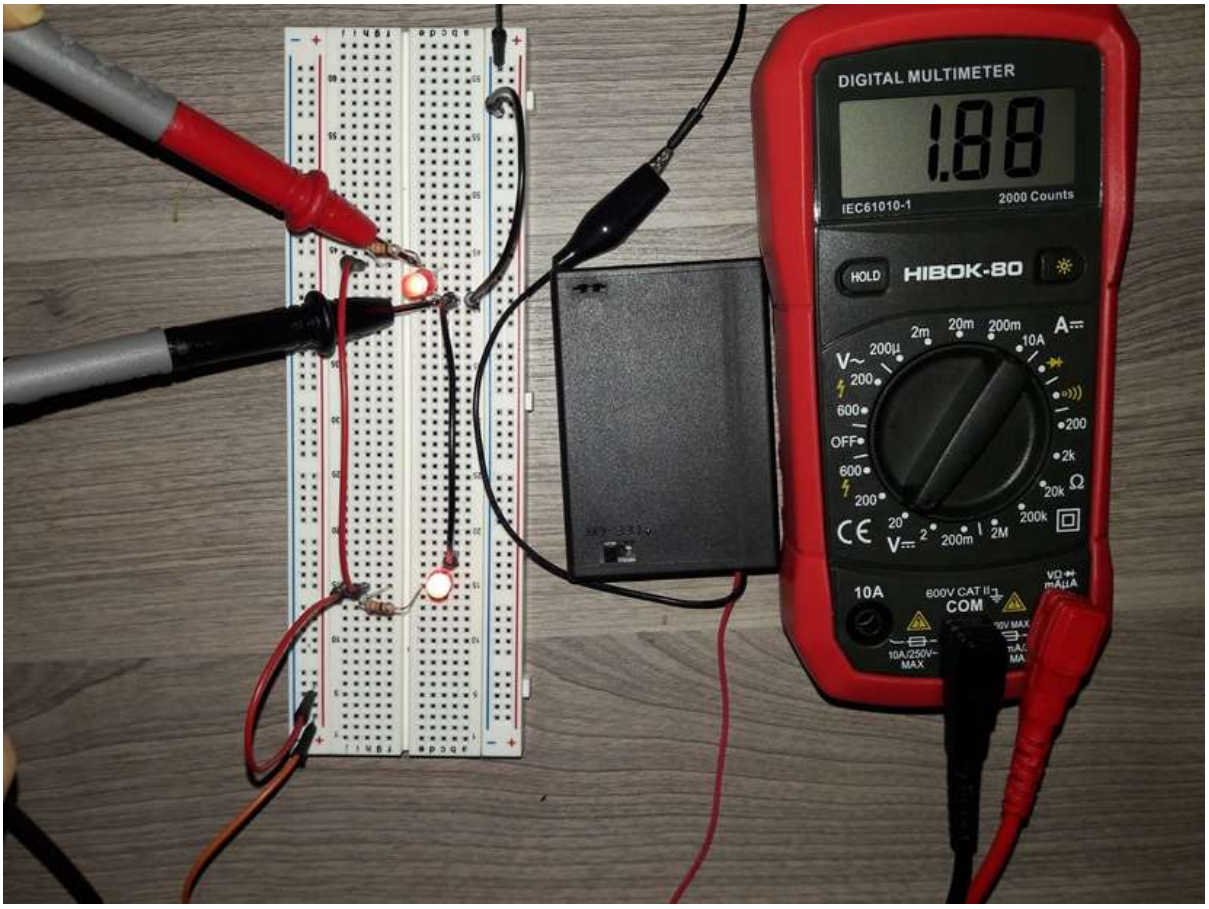
Utilizando el amperímetro en serie con el circuito, podremos medir la intensidad de corriente I antes del nudo y en cada una de las ramas (fotos 1, 2 y 3). Con el modo voltímetro del multímetro en paralelo con el circuito, podremos confirmar que la diferencia de potencial (voltaje) entre dos puntos, no depende del camino seguido (fotos 4 y 5).




DIGITAL MULTIMETER
6.08
IEC61010-1
2000 Counts
HOLD
HIBOK-80
OFF
10A
600V CAT II
COM

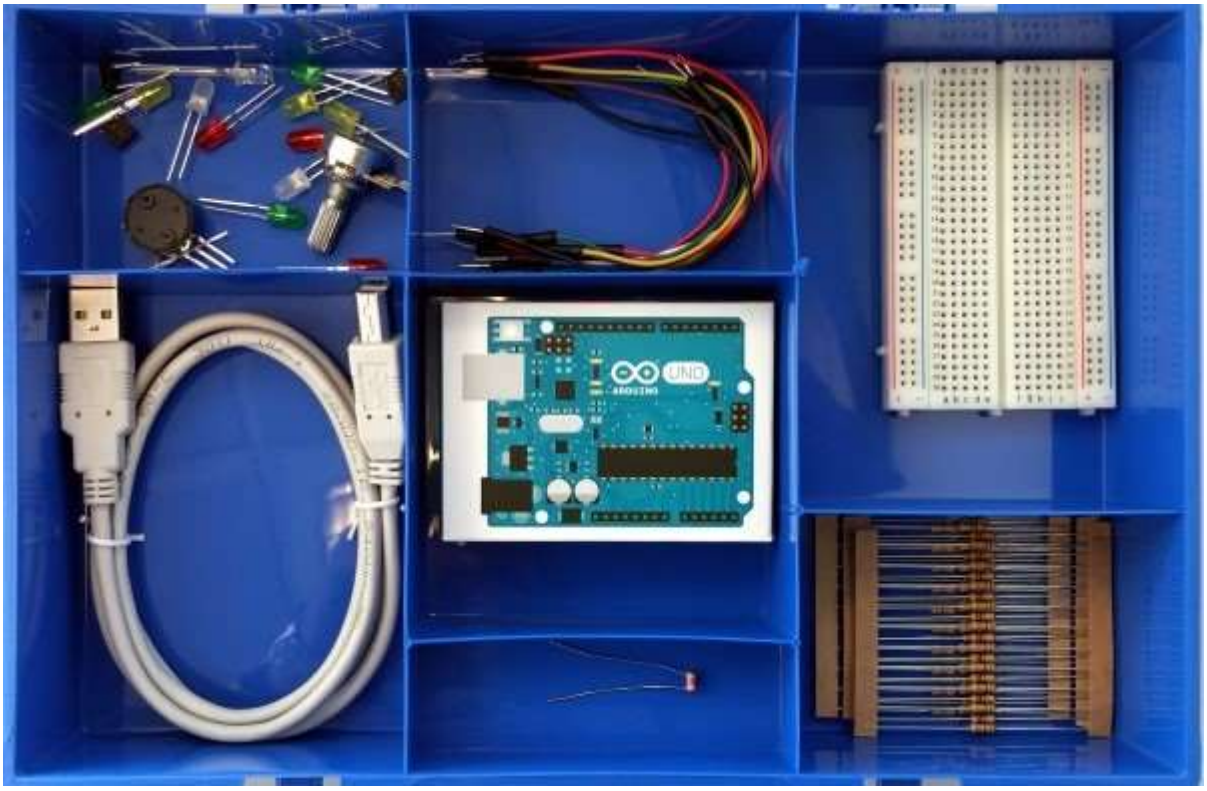

DIGITAL MULTIMETER
12.89
IEC61010-1
2000 Counts
HOLD
HIBOK-80
OFF
10A
600V CAT II
COM


DIGITAL MULTIMETER
1.88
IEC61010-1
2000 Counts
HOLD
HIBOK-80
OFF
10A
600V CAT II
COM

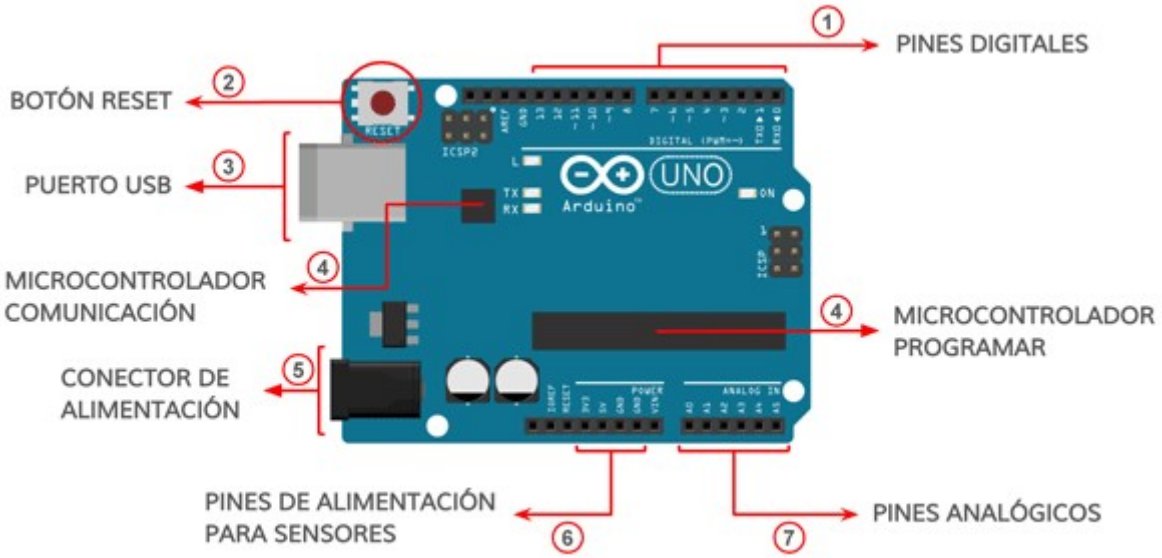

DIGITAL MULTIMETER
1.88
IEC61010-1
2000 Counts
HOLD
HIBOK-80
OFF
10A
600V CAT II
COM

// MONTAJES Y CÓDIGOS

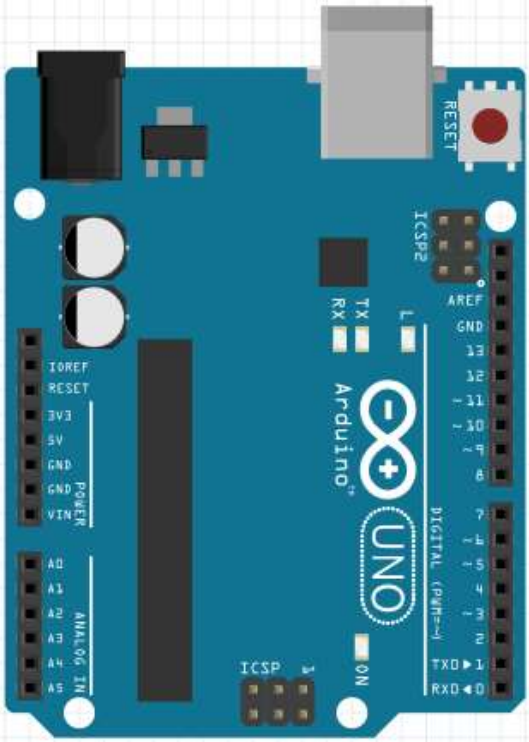
//06 NIVEL 1: COMPONENTES.



//07 NIVEL 1: PLACA MICROCONTROLADORA. LENGUAJE C++.



MCU : Atmega 328
Input voltage : 7V-12V
Operating voltage : 5V
CPU Speed : 16MHZ
Analog In/Out : 6/0
Digital IO/PWM :14/6
EEPROM : 1KB
SRAM : 2KB
Flash : 32KB
UART : 1
USB : Regular



ARDUINO PIN		MICROCONTROLLER PIN
0	-	PD0(RXD)
1	-	PD1(TXD)
2	-	PD2(INT0)
3	-	PD3(INT1)
4	-	PD4
5	-	PD5
6	-	PD6
7	-	PD7
8	-	PB0
9	-	PB1
10	-	PB2(SS')
11	-	PB3(MOSI)
12	-	PB4(MISO)
13	-	PB5(SCK)
A0	-	PC0
A1	-	PC1
A2	-	PC2
A3	-	PC3
A4	-	PC4(SDA)
A5	-	PC5(SCL)

Programación en Arduino

La programación en Arduino es muy similar a la de muchos otros lenguajes de programación, esencialmente al lenguaje C, del cual hereda muchas funcionalidades.

Para programar siempre se sigue la estructura de un algoritmo, pero ...

¿Qué es un algoritmo?

Un algoritmo es un conjunto de instrucciones detalladas paso a paso para resolver un problema o completar una tarea. Los algoritmos pueden ser cosas muy cotidianas como, una receta para cocinar, el método usado en resolver una suma, o el proceso de doblar una camisa, para todas estas actividades implícitamente realizamos pasos que hemos aprendido con el pasar de los años. En Arduino, escribiremos algoritmos que le indicarán al microcontrolador cómo realizar una tarea. Por ejemplo, para hacer parpadear un led.

Esta tarea posee los pasos siguientes.

- Encender el led
- Esperar
- Apagar el led
- Esperar

Para que este conjunto de pasos se ejecuta continuamente, anidamos los pasos en un bucle o ciclo, un bucle realiza la repetición de un conjunto de instrucciones infinitamente, hasta cumplir una condición o definiendo previamente un número de veces.

En la programación de microcontroladores es normal que el programa principal se repita infinitamente hasta que se resetee el programa o se desconecte el microcontrolador.

Variables

Una variable es la manera de codificar, representar y almacenar un valor o un valor dentro de un programa y así facilitar su manipulación. Las variables son almacenadas en la memoria del microcontrolador. Para que Arduino consiga realizar las tareas que le asignamos necesita el procesamiento e interpretación de estos datos.

```
byte  a = 5;
```

Hagamos una abstracción.

Consideremos a las variables como los cajones de un armario, cada uno tiene una etiqueta describiendo el contenido, acompañado de un tipo que indicará el número de cajones necesarios para almacenar un contenido y dentro se encuentra el valor de la variable como el contenido del cajón.

Tipos de dato en Arduino

Arduino considera los tipos de datos siguientes:

Entre los más importantes y usados tenemos:

- El tipo boolean que indica verdadero o falso.
- El tipo int para almacenar números enteros.
- El tipo float para números con decimales, no enteros
- El tipo string que es una secuencia y agrupación de datos de tipo char. Se utiliza para almacenar cadenas de texto.
- El array, que es una colección de datos de un mismo tipo.

Array

Un array es una colección de variables de un mismo tipo, que comparten la misma etiqueta o nombre.

Declaración

La sintaxis para declarar un array es simple solo agregamos corchetes al nombre de la variable, tener en cuenta que este array no posee un tamaño determinado.

Variable

```
int Pin;
```

```
int Pin = 5;
```

Array

```
int Pins[];
```

Si queremos que este array tenga un número determinado de elementos colocamos tal número dentro de los corchetes.

```
int Pins[3];
```

Para agregar elementos en la declaración del array, los colocamos entre llaves separados por comas.

```
int Pins[3] = {1,2,3};
```

Para acceder a cada uno de los elementos del array. sólo hay que colocar el índice de este entre los corchetes, recordar que el primer índice del array es 0.

```
Pins[índice];
```

Funciones

Una función es un bloque de código que tiene un conjunto de instrucciones que realizan una tarea específica. Estas instrucciones son ejecutadas cuando la función es invocada dentro de un programa principal.

Poseen la siguiente estructura

```
type name (parámetros)
{
        Instrucción 1;
        Instrucción 2;
}
```

El tipo de dato es el valor que devolverá la función a un programa principal, por ejemplo 'int' se utilizará cuando la función devuelve un dato numérico de tipo entero.

Si la función no devuelve ningún valor entonces se colocará delante la palabra “void”, que significa “vacío”.

Después se escribe el nombre de la función o etiqueta con el que se identifica y con el cual será invocada dentro del programa principal.

Entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función. Un parámetro es información que se brinda a la función para que pueda ser procesada.

Los parámetros son declarados con el tipo de dato al que pertenecen y son separados con comas.

Todas las instrucciones pertenecientes a la función son declaradas dentro de dos corchetes que delimitan su alcance.

Si la función retorna un valor al programa principal, se utiliza la palabra clave return.

```
int suma (int a, int b)
{
        return a+b;
}
```

Sintaxis básica de un programa en Arduino La estructura básica de un programa en Arduino es bastante simple y se compone de un mínimo de dos partes. La de configuraciones y el programa principal.

La primera función es void setup() es la parte encargada de las configuraciones. Es la primera función a ejecutar al correr el programa en el microcontrolador, se ejecuta sólo una vez, y se utiliza para configurar o inicializar los pines en un modo de trabajo específico, de entrada o salida, además de la configuración de la comunicación y otras.

```
void setup ()
{
        Configuración 1;
        Instrucción 1;
}
```

La segunda es void loop(), como mencionamos antes la repetición constante de un grupo de instrucciones es fundamental en la programación en microcontroladores. void loop() contiene las instrucciones que se ejecutaran continuamente como, lectura de entradas, señales de salida, etc. Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte de las tareas.

```
void loop ()
{
        Instrucción 2;
        Instrucción 3;
}
```

Sintaxis

Algunos elementos de sintaxis a tener en cuenta:

Llaves

Las llaves sirven para definir el inicio y el final de un bloque de instrucciones. Se utilizan para los bloques de programación setup(), loop(), funciones de usuario, etc.

```
type function()
{
        instrucciones;
}
```

Punto y coma

El punto y coma ";" se utiliza para separar instrucciones y así poder ser interpretadas y compiladas. Muchas veces olvidamos de colocarlos y esto nos dará error al momento de compilar el programa.

```
int x = 13;
```

Línea de comentarios

Una línea de comentario empieza con // y terminan con la siguiente línea de código, las líneas de comentarios son ignoradas por el programa y no ocupan espacio en la memoria.

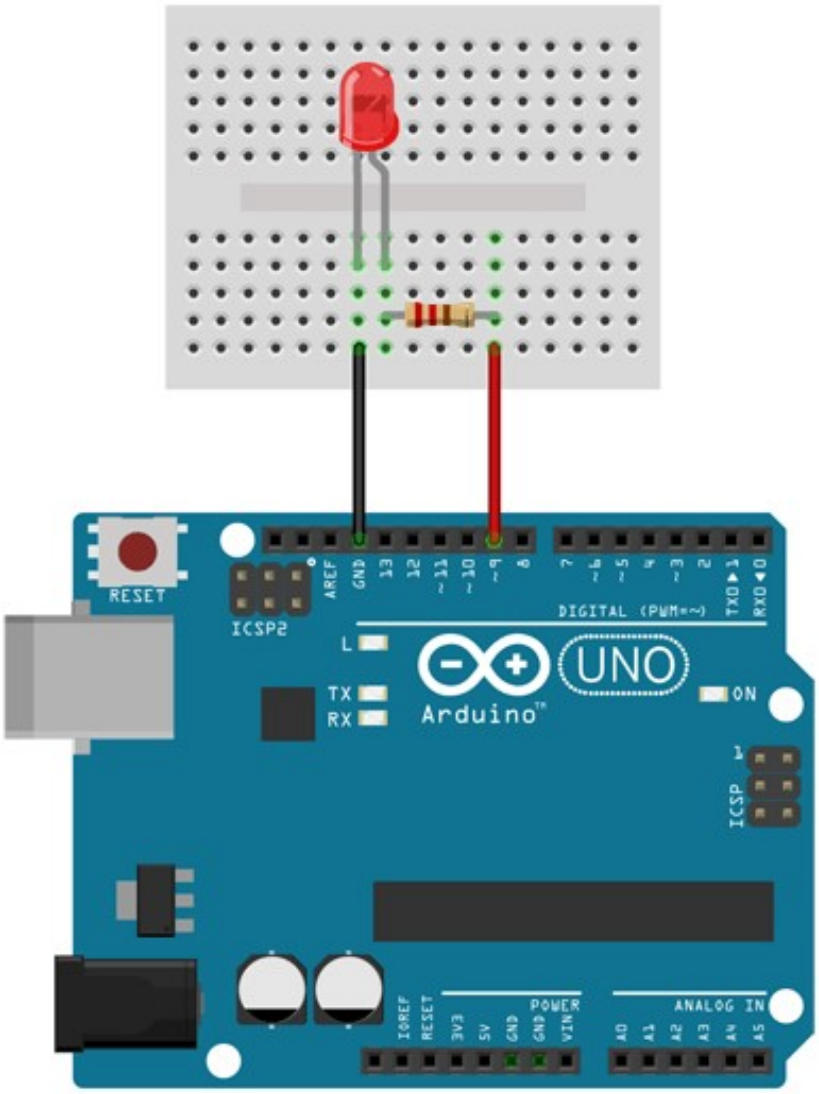
```
// Esto es un comentario
```

Una línea de comentario se utiliza a menudo después de una instrucción, para proporcionar más información acerca de lo que hace ésta o para recordarla más adelante.

// NIVEL 1: LEDS, POTENCIÓMETROS, FOTORRESISTENCIAS, TRANSISTORES, ZUMBADORES, PULSADORES.

//08 ENCENDER Y APAGAR UN LED.

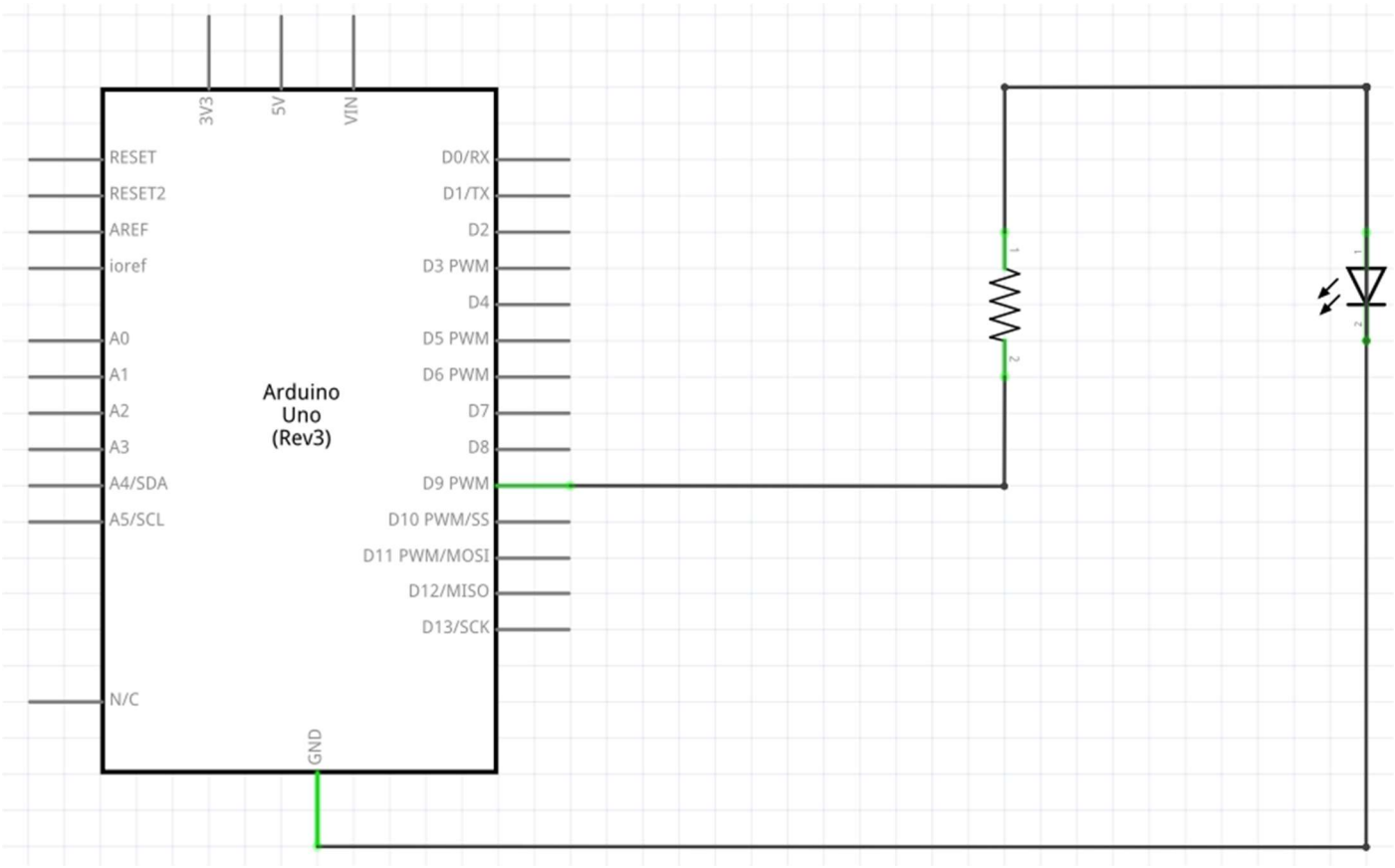
MONTAJE



MATERIALES

– Un cable USB tipo A-B
– Placa microcontroladora
– Placa de prototipos
– LED rojo de 5mm
– 1 Resistencia de 330 Ohmios (naranja, naranja, marrón)
– 2 cables M-M

CIRCUITO



CALCULA Y MIDE

Razonamiento para calcular la Resistencia que necesitamos:
– Queremos encender un LED que necesita una corriente de aproximadamente entre 10mA (0.01A) y 20mA para funcionar.
– El voltaje suministrado por la placa Arduino es de 5v, y el LED tiene un consumo de 1.8v, por lo tanto, los 3.2v de exceso deben consumirse en la Resistencia cuyo valor buscamos:

R= V / I = (5 – 1.8) / 0.01A = 320 Ω

Cogemos la más parecida, en este caso 330 Ω.

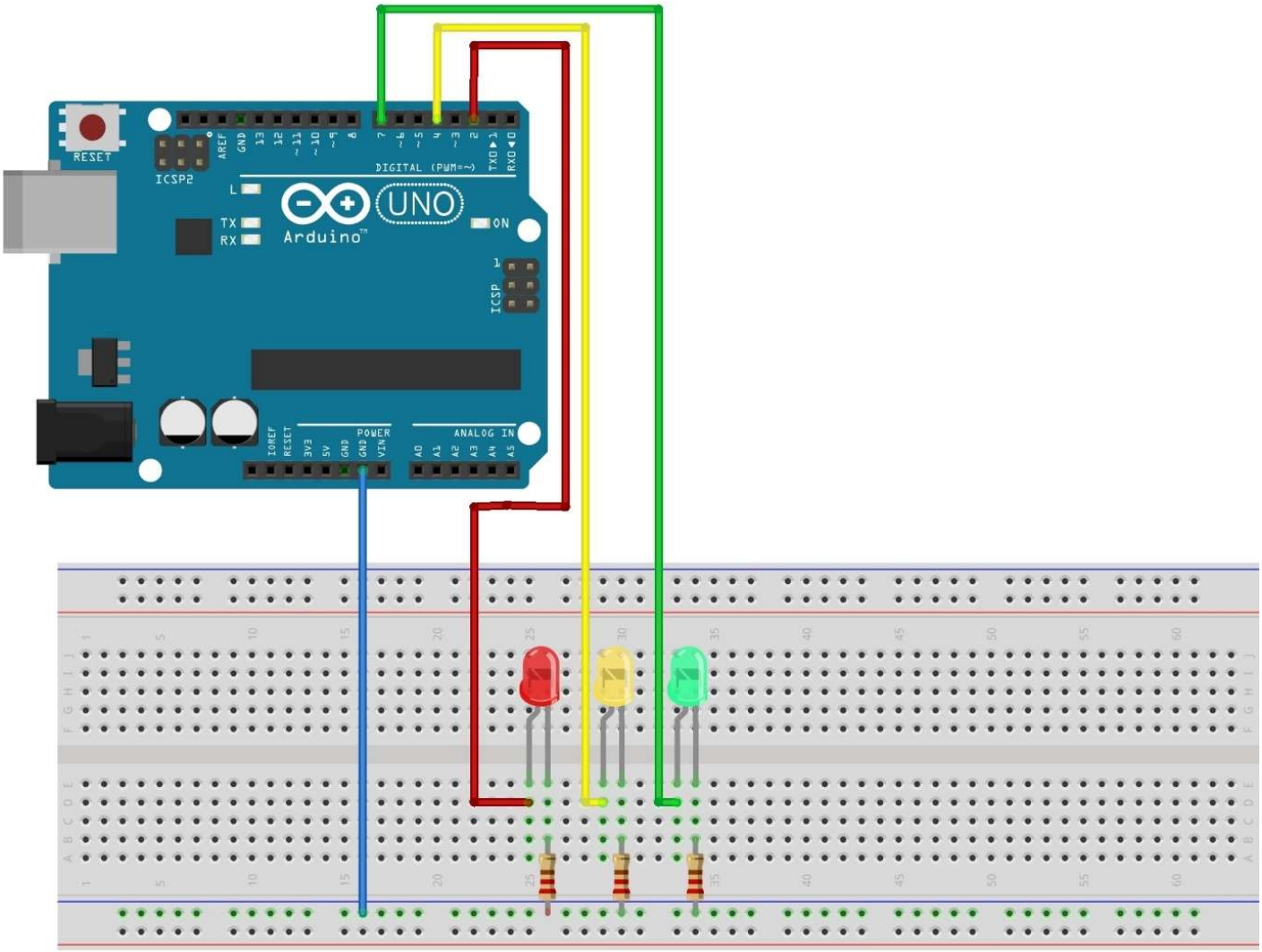
Calcula y mide:
Con los datos anteriores, prueba a realizar los cálculos analíticos y a medir con el voltímetro la diferencia de potencial de cada componente.
Solución: La intensidad de corriente I, es la misma por toda la malla.
I = 9.7mA
V-resistencia = 3.2v
V-LED = 1.8v
V-Total = 5v

CÓDIGO

```
const int ledPIN = 9;

void setup() {
  Serial.begin(9600);    //iniciar puerto serie
  pinMode(ledPIN , OUTPUT);  //definir pin como salida
}

void loop(){
  digitalWrite(ledPIN , HIGH);   // poner el Pin en HIGH
  delay(1000);                   // esperar un segundo
  digitalWrite(ledPIN , LOW);    // poner el Pin en LOW
  delay(1000);                   // esperar un segundo
}
```

//09 EL SEMÁFORO.

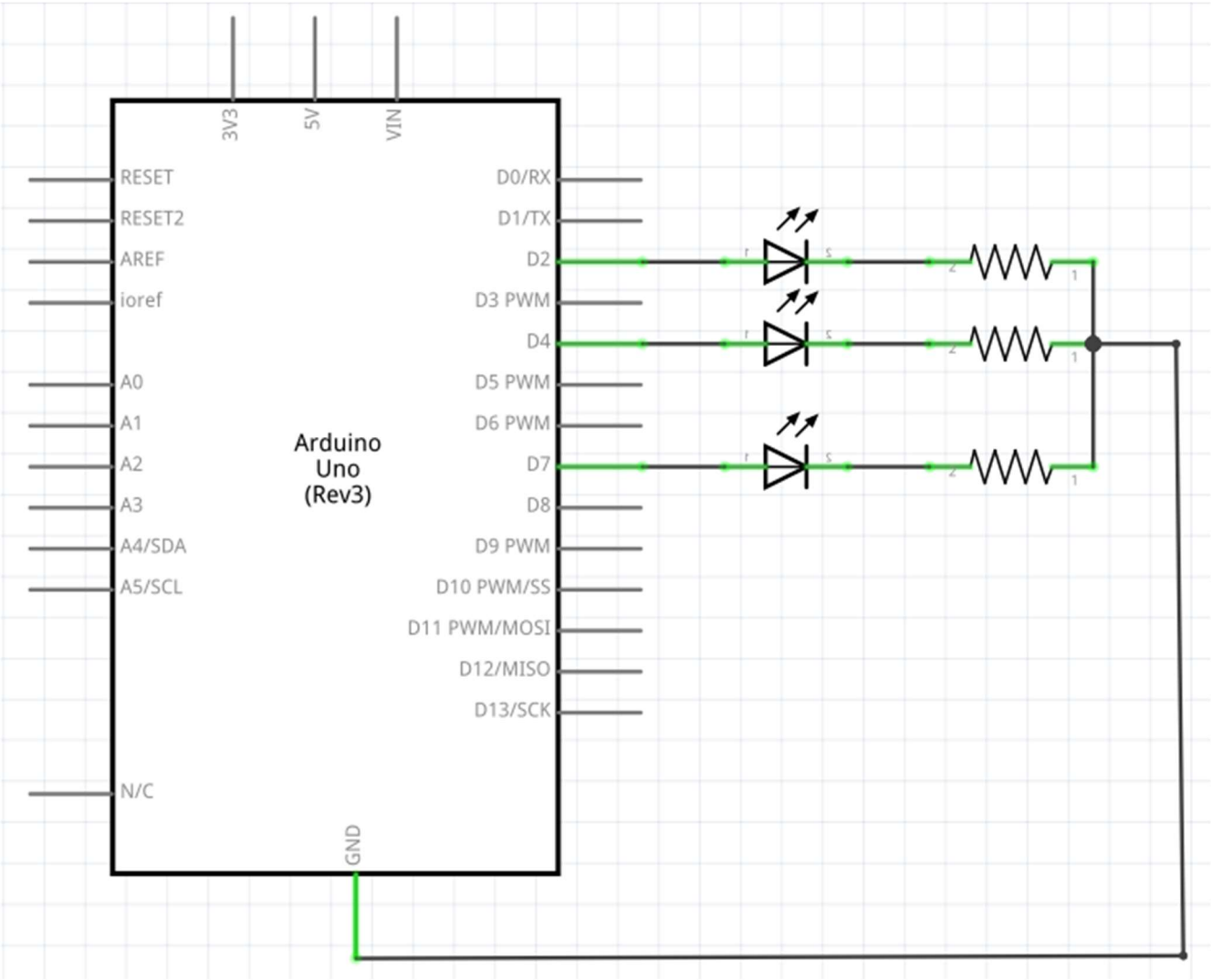
MONTAJE



MATERIALES

– Un cable USB tipo A-B
– Placa microcontroladora
– Placa de prototipos
– LED rojo de 5mm
– LED amarillo de 5mm
– LED verde de 5mm
– 3 Resistencias de 330 Ohmios (naranja, naranja, marrón)
– 4 cables M-M

CIRCUITO



CALCULA Y MIDE

Razonamiento para calcular las corrientes de cada rama:
– La corriente I cuando llega a un nudo, se reparte por los diferentes cables. El camino por el que menos resistencia encuentre, será por el que se vaya la mayor parte de la corriente.
– Para calcular la corriente I que hay entrando en el nudo, hallamos la resistencia equivalente del circuito, en este caso, sumándolas en paralelo:

1/R= 1/330 + 1/330 + 1/330

de donde la resistencia equivalente R = 110 Ω

– Conocida la resistencia equivalente del circuito y el voltaje que suministra la placa microcontroladora (Arduino), podemos obtener la corriente I que entra en el nudo, y que se reparte posteriormente por las 3 ramas.

I= (5 – 1.8) / 110 = 3.2v / 110 Ω = 29.1mA

– Esta corriente I se reparte dependiendo de la resistencia que encuentra en cada una de las ramas.
– Dado que todas las ramas tienen la misma resistencia, la corriente se distribuye por igual en cada una.

Calcula y mide:
Comprueba que la diferencia de potencial entre dos puntos es la misma, independientemente del camino seguido, y para ello:
Mide la diferencia de potencial desde la pata positiva de una resistencia, hasta la negativa del diodo de su misma rama.
Mide la esta misma diferencia de potencial en cada una de las ramas.
Solución: La diferencia de potencial (voltaje) es la misma.

I = V/R = 3.2v / 110 Ω = 29.1mA
I1 = V/R1 = 3.2v / 330 Ω = 9.7mA
I2 = V/R2 = 3.2v / 330 Ω = 9.7mA
I3 = V/R3 = 3.2v / 330 Ω = 9.7mA

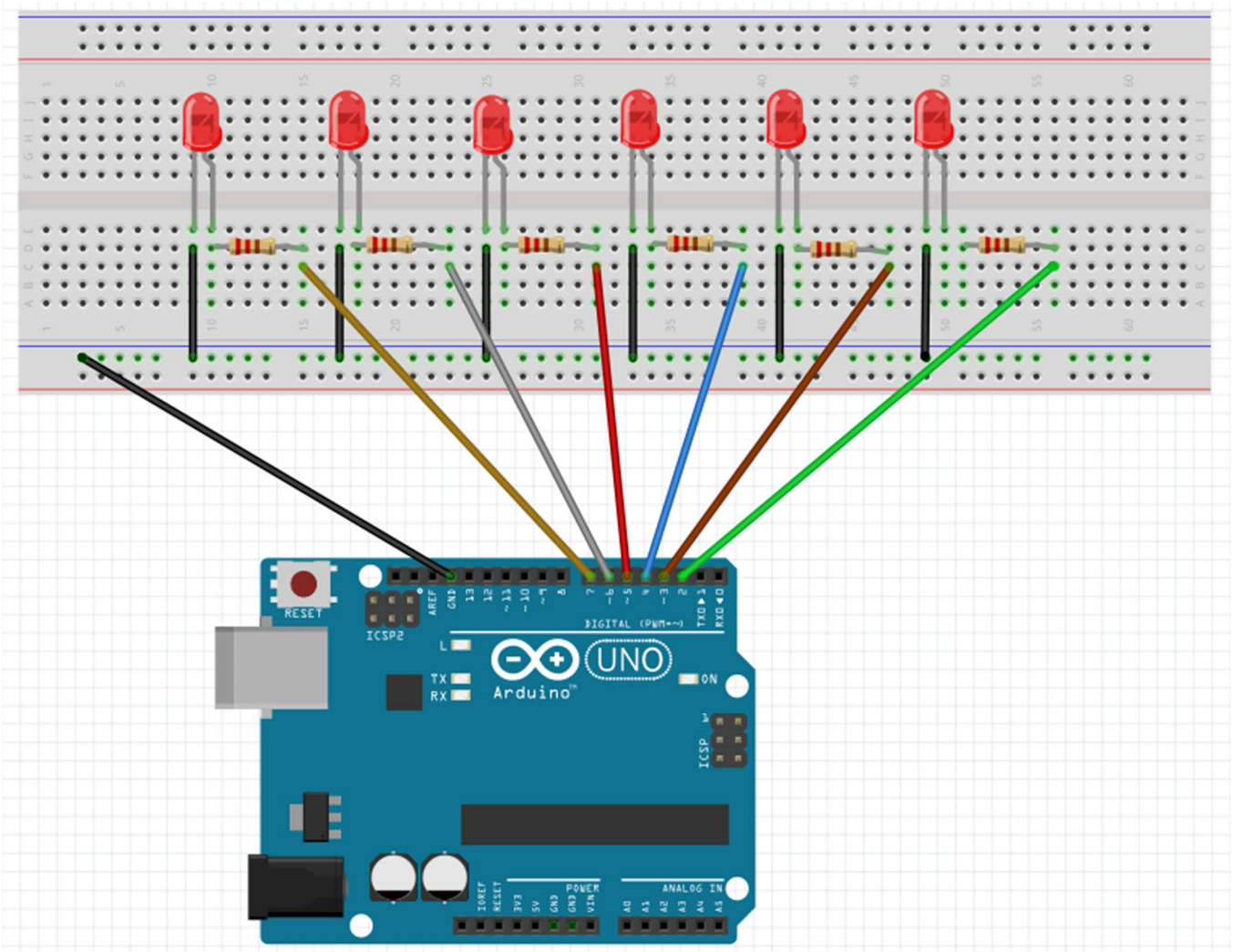
V-resistencia1 = I1 * R1 = 9.7mA * 330 Ω = 3.2v V-LED1 = 1.8v
V-resistencia2 = I2 * R2 = 9.7mA * 330 Ω = 3.2v V-LED2 = 1.8v
V-resistencia3 = I3 * R3 = 9.7mA * 330 Ω = 3.2v V-LED3 = 1.8v
V-Total-cualquier-rama = 5v

CÓDIGO

```
//** Definiciones **//
int rojo=2;      //definimos el valor del pin para el led rojo
int amarillo=4;  //definimos el valor del pin para el led amarillo
int verde=7;     //definimos el valor del pin para el led verde

//** Programa **//

void setup() {
  pinMode(verde,OUTPUT);   //declaramos el pin verde como salida
  pinMode(amarillo,OUTPUT);//declaramos el pin amarillo como salida
  pinMode(rojo,OUTPUT);    //declaramos el pin rojo como salida
}

void loop() {
 digitalWrite(verde,HIGH); //encendemos el led verde
 delay(2000);              //esperamos 2 segundos
 digitalWrite(verde,LOW);  //apagamos el led verde
 delay(500);               //esperamos medio segundo

 digitalWrite(amarillo,HIGH); //encendemos el led amarillo
 delay(2000);                 //esperamos 2 segundos
 digitalWrite(amarillo,LOW);  //apagamos el led amarillo
 delay(500);                  //esperamos medio segundo

 digitalWrite(rojo,HIGH); //encendemos el led rojo
 delay(2000);              //esperamos 2 segundos
 digitalWrite(rojo,LOW);  //apagamos el led rojo
 delay(500);               //esperamos medio segundo
}
```

//10 EL COCHE FANTÁSTICO.

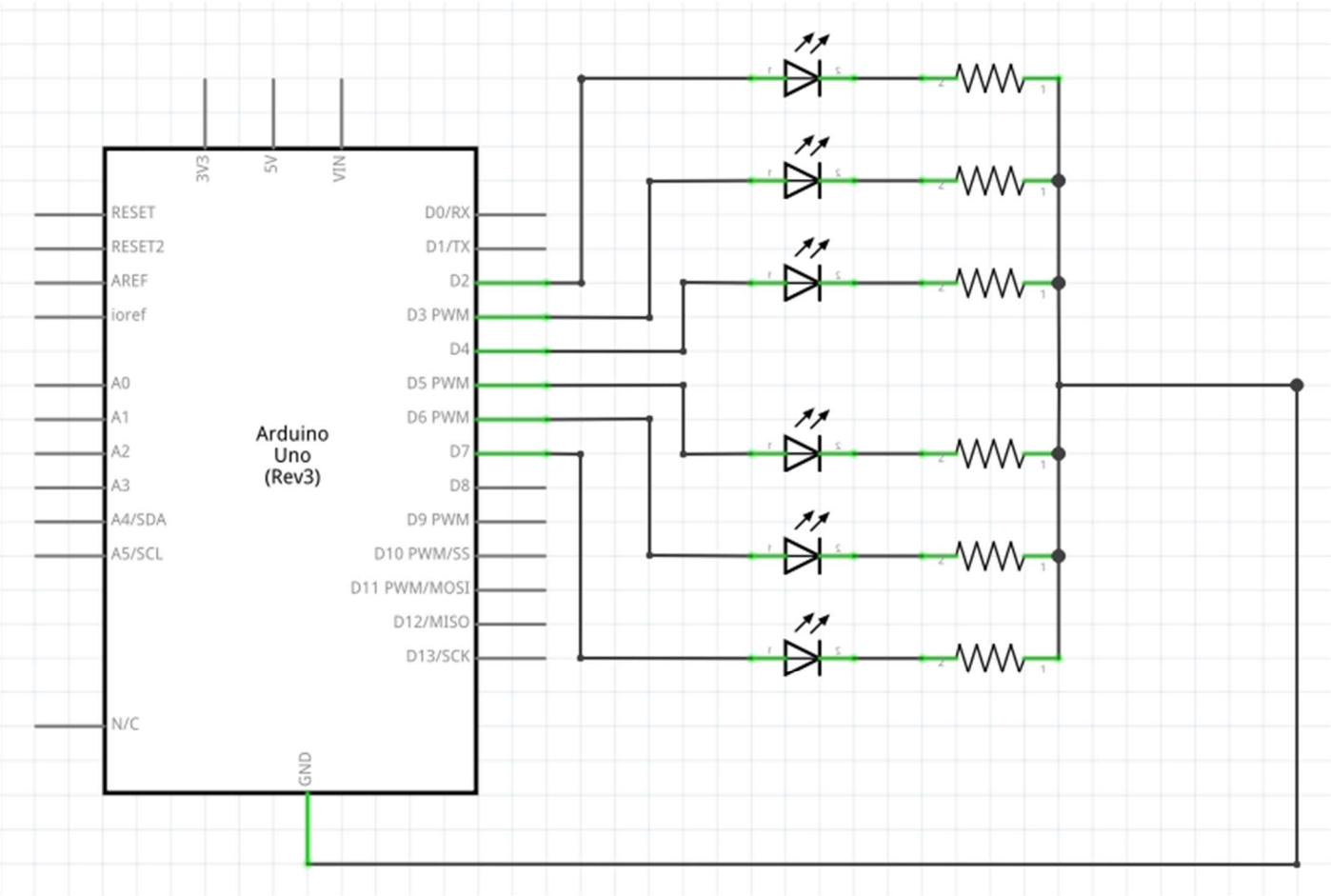
MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- 6 LEDS rojos de 5mm
- 6 Resistencias de 330 Ohmios (naranja, naranja, marrón)
- 13 cables M-M

CIRCUITO



CALCULA Y MIDE

Razonamiento para calcular las corrientes de cada rama:
– La corriente I cuando llega a un nudo, se reparte por los diferentes cables. El camino por el que menos resistencia encuentre, será por el que se vaya la mayor parte de la corriente.
– Para calcular la corriente I que hay entrando en el nudo, hallamos la resistencia equivalente del circuito, en este caso, sumándolas en paralelo:

1/R= 1/330 + 1/330 + 1/330 + 1/330 + 1/330 + 1/330 de donde la resistencia equivalente R = 55 Ω

– Conocida la resistencia equivalente del circuito y el voltaje que suministra la placa microcontroladora (Arduino), podemos obtener la corriente I que entra en el nudo, y que se reparte posteriormente por las 6 ramas.

I= (5 – 1.8) / 55 = 3.2v / 55 Ω = 58.2mA

– Esta corriente I se reparte dependiendo de la resistencia que encuentra en cada una de las ramas.
– Dado que las ramas tienen la misma resistencia, la corriente se distribuye por igual en cada una.

Calcula y mide:
Comprueba que la diferencia de potencial entre dos puntos es la misma, independientemente del camino seguido, y para ello:
Mide la diferencia de potencial desde la pata positiva de una resistencia, hasta la negativa del diodo de su misma rama.
Mide la esta misma diferencia de potencial en cada una de las ramas.
Solución: La diferencia de potencial (voltaje) es la misma.

I = V/R = 3.2v / 55 Ω = 58.2mA
I1 = V/R1 = 3.2v / 330 Ω = 9.7mA
I2 = V/R2 = 3.2v / 330 Ω = 9.7mA
I3 = V/R3 = 3.2v / 330 Ω = 9.7mA
I4 = V/R4 = 3.2v / 330 Ω = 9.7mA
I5 = V/R5 = 3.2v / 330 Ω = 9.7mA
I6 = V/R6 = 3.2v / 330 Ω = 9.7mA

V-resistencia1 = I1 * R1 = 9.7mA * 330 Ω = 3.2v V-LED1 = 1.8v
V-resistencia2 = I2 * R2 = 9.7mA * 330 Ω = 3.2v V-LED2 = 1.8v
V-resistencia3 = I3 * R3 = 9.7mA * 330 Ω = 3.2v V-LED3 = 1.8v
V-resistencia4 = I4 * R4 = 9.7mA * 330 Ω = 3.2v V-LED4 = 1.8v
V-resistencia5 = I5 * R5 = 9.7mA * 330 Ω = 3.2v V-LED5 = 1.8v
V-resistencia6 = I6 * R6 = 9.7mA * 330 Ω = 3.2v V-LED6 = 1.8v
V-Total-cualquier-rama = 5v

CÓDIGO

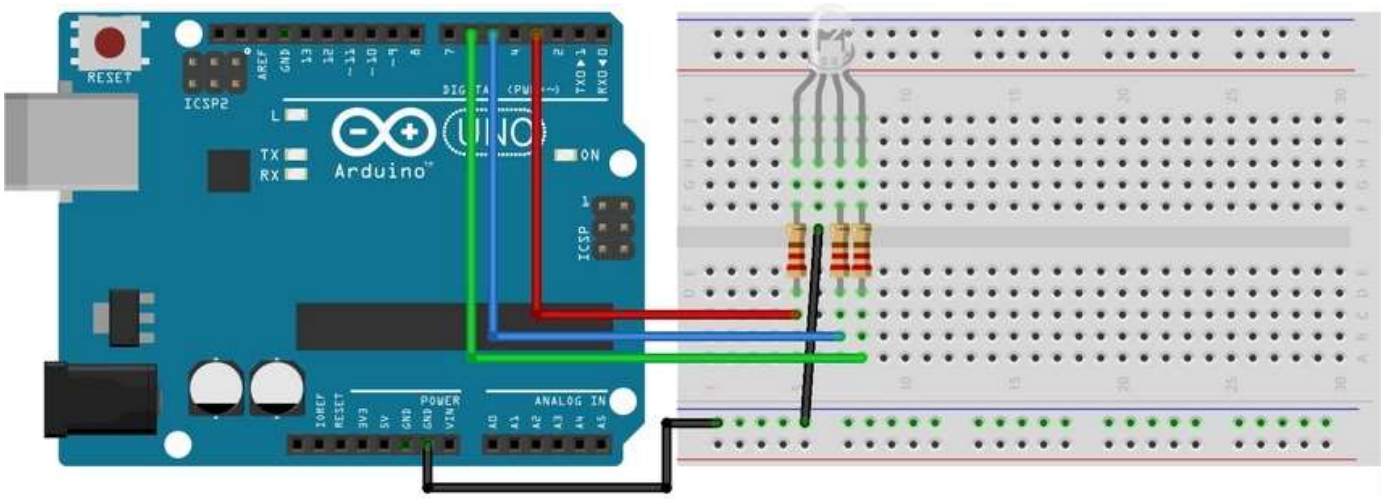
```
int pin2 = 2;
int pin3 = 3;
int pin4 = 4;
int pin5 = 5;
int pin6 = 6;
int pin7 = 7;
int timer = 50;

void setup(){
  Serial.begin(9600);
  pinMode(pin2, OUTPUT);
  pinMode(pin3, OUTPUT);
  pinMode(pin4, OUTPUT);
  pinMode(pin5, OUTPUT);
  pinMode(pin6, OUTPUT);
  pinMode(pin7, OUTPUT);
}

void loop() {

   //timer = analogRead(A0);  //El valor leido por analog read es el temporizador

   digitalWrite(pin2, HIGH);
   Serial.println("Enciendo primer led");
   delay(timer);
   digitalWrite(pin2, LOW);
   delay(timer);

   digitalWrite(pin3, HIGH);
   Serial.println("Enciendo segundo led");
   delay(timer);
   digitalWrite(pin3, LOW);
   delay(timer);

   digitalWrite(pin4, HIGH);
```

```
  Serial.println("Enciendo tercer led");
  delay(timer);
  digitalWrite(pin4, LOW);
  delay(timer);

  digitalWrite(pin5, HIGH);
  Serial.println("Enciendo cuarto led");
  delay(timer);
  digitalWrite(pin5, LOW);
  delay(timer);

  digitalWrite(pin6, HIGH);
  Serial.println("Enciendo quinto led");
  delay(timer);
  digitalWrite(pin6, LOW);
  delay(timer);

  digitalWrite(pin7, HIGH);
  Serial.println("Enciendo quinto led");
  delay(timer);
  digitalWrite(pin7, LOW);
  delay(timer);

  digitalWrite(pin6, HIGH);
  Serial.println("Enciendo quinto led");
  delay(timer);
  digitalWrite(pin6, LOW);
  delay(timer);

  digitalWrite(pin5, HIGH);
  Serial.println("Enciendo cuarto led");
  delay(timer);
  digitalWrite(pin5, LOW);
  delay(timer);

  digitalWrite(pin4, HIGH);
  Serial.println("Enciendo tercer led");
  delay(timer);
```

```
    digitalWrite(pin4, LOW);
    delay(timer);

    digitalWrite(pin3, HIGH);
    Serial.println("Enciendo segundo led");
    delay(timer);
    digitalWrite(pin3, LOW);
    delay(timer);
}
```

//11 EL LED RGB.

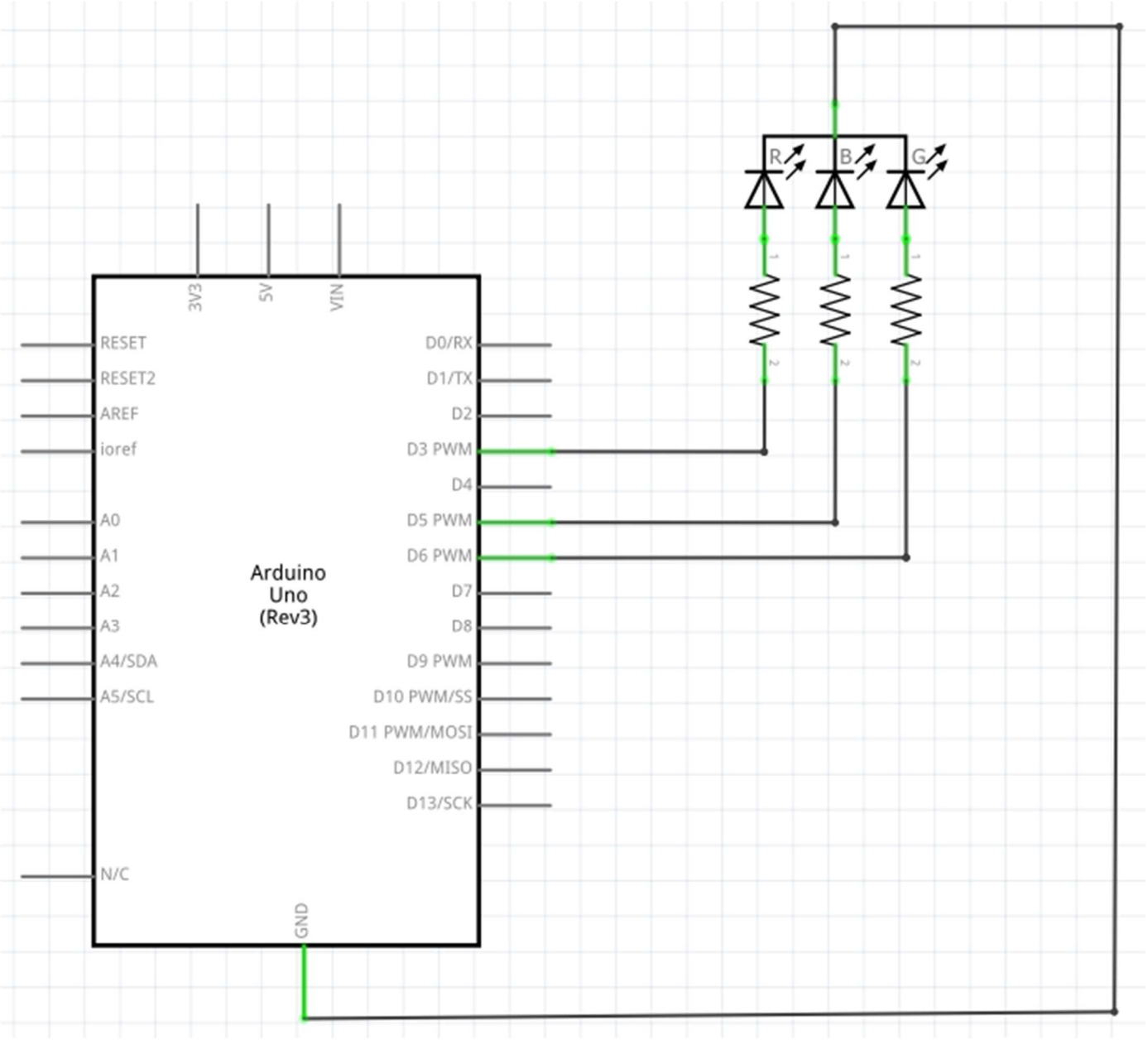
MONTAJE



MATERIALES

– Un cable USB tipo A-B
– Placa microcontroladora
– Placa de prototipos
– LED RGB de 5mm
– 3 Resistencias de 330 Ohmios (naranja, naranja, marrón)
– 4 cables M-M

CIRCUITO



CALCULA Y MIDE

El LED RGB:
– Dispone en su interior de tres LEDS individuales (rojo, verde y azul – RGB) que podemos encender de forma independiente o mediante cualquier combinación de ellos.
– Tiene 4 terminales, siendo el más largo de ellos el punto común a los tres LEDS, y que por tanto irá conectado a la tierra (potencial 0) de nuestro circuito.
– Su circuito es el mismo paralelo que resolvimos para la práctica del Semáforo, por lo que aprovechamos los cálculos y razonamientos de la misma.

Razonamiento para calcular las corrientes de cada rama:
– La corriente I cuando llega a un nudo, se reparte por los diferentes cables. El camino por el que menos resistencia encuentre, será por el que se vaya la mayor parte de la corriente.
– Para calcular la corriente I que hay entrando en el nudo, hallamos la resistencia equivalente del circuito, en este caso, sumándolas en paralelo:

1/R= 1/330 + 1/330 + 1/330 de donde la resistencia equivalente R = 110 Ω

– Conocida la resistencia equivalente del circuito y el voltaje que suministra la placa microcontroladora (Arduino), podemos obtener la corriente I que entra en el nudo, y que se reparte posteriormente por las 3 ramas.

I= (5 – 1.8) / 110 = 3.2v / 110 Ω = 29.1mA

– Esta corriente I se reparte dependiendo de la resistencia que encuentra en cada una de las ramas.
– Dado que las ramas tienen la misma resistencia, la corriente se distribuye por igual en cada una.

Calcula y mide:
Comprueba que la diferencia de potencial entre dos puntos es la misma, independientemente del camino seguido, y para ello:
Mide la diferencia de potencial desde la pata positiva de una resistencia, hasta la negativa del diodo de su misma rama. Mide la esta misma diferencia de potencial en cada una de las ramas.
Solución: La diferencia de potencial (voltaje) es la misma.

I = V/R = 3.2v / 110 Ω = 29.1mA
I1 = V/R1 = 3.2v / 330 Ω = 9.7mA
I2 = V/R2 = 3.2v / 330 Ω = 9.7mA
I3 = V/R3 = 3.2v / 330 Ω = 9.7mA
V-resistencia1 = I1 * R1 = 9.7mA * 330 Ω = 3.2v V-LED1 = 1.8v
V-resistencia2 = I2 * R2 = 9.7mA * 330 Ω = 3.2v V-LED2 = 1.8v
V-resistencia3 = I3 * R3 = 9.7mA * 330 Ω = 3.2v V-LED3 = 1.8v
V-Total-cualquier-rama = 5v

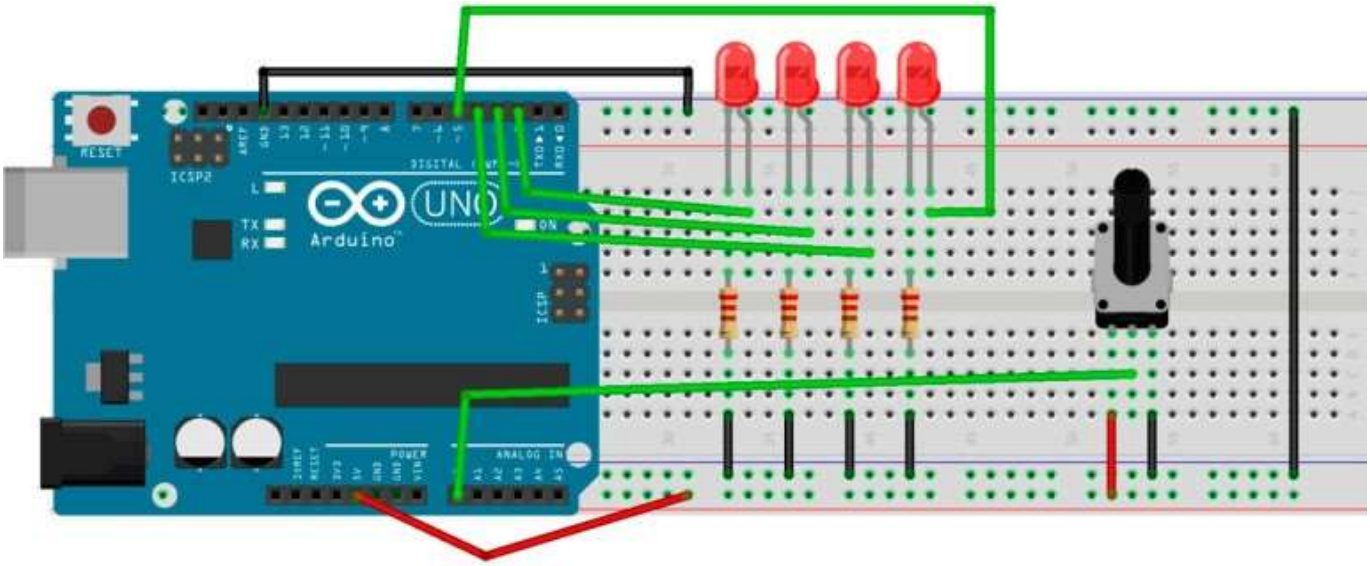
CÓDIGO

```
/*-----Declaracion de variables para cada color R G B-----*/
int rled = 3; // Pin PWN 3 para led rojo
int bled = 5; // Pin PWM 5 para led azul
int gled = 6; // Pin PWM 6 para led verde

/*----Declaracion de variables auxiliares-----*/
int i; // Variable para ciclos repetitivos
int repeat = 5; // Variables para cantidad limite de repeticiones

void setup() {
/*----- Se inicializan pines PWM como salida*/
  pinMode(rled, OUTPUT);
  pinMode(bled, OUTPUT);
  pinMode(gled, OUTPUT);
}

void loop() {
 for(i=0; i < repeat; i++) //Se repite la ejecucion de la funcion rgbon() "repeat" veces
    rgbon();
 delay(1000); //Se espera 1 segundo
 colors('y'); //Se enciende el LED en color amarillo (y de yellow)
 delay(1000);
 colors('o'); //Se enciende el LED en color naranja (o de orange)
 delay(1000);
 colors('p'); //Se enciende el LED en color rosado (p de pink)
 delay(1000);
}
/*-----Funcion para mostrar colores principales cada 500 ms-----*/
void rgbon(){
  analogWrite(rled,255); // Se enciende color rojo
  delay(500);            // Se esperan 500 ms
  analogWrite(rled,0);   // Se apaga color rojo
  analogWrite(bled,255); // Se enciende color azul
  delay(500);            // Se esperan 500 ms
  analogWrite(bled,0);   // Se apaga color azul
```

```
  analogWrite(gled,255); // Se enciende color verde
  delay(500);            // Se esperan 500 ms
  analogWrite(gled,0);   // Se apaga color verde
}
/*-----Funcion que permite escoger entre color amarillo, naranja o rosado-----*/
void colors(char color){ //La funcion recibe un parametro que se guarda en variable color

  switch(color){ //Se compara variable color con dato guardado
    case 'y': analogWrite(rled,255); // Si color == 'y' se enciende color amarillo
            analogWrite(gled,255); // Mezclando r = 255 / g = 255 / b = 0
            analogWrite(bled,0);
    break;
    case 'o': analogWrite(rled,255); // Si color == 'o' se enciende color naranja
            analogWrite(gled,180); // Mezclando r = 255 / g = 180 / b = 0
            analogWrite(bled,0);
    break;
    case 'p': analogWrite(rled,255); // Si color == 'p' se enciende color rosado
          analogWrite(gled,0);   // Mezclando r = 255 / g = 0 / b = 255
            analogWrite(bled,255);
    break;
  }
}
```

//12 EL POTENCIÓMETRO.

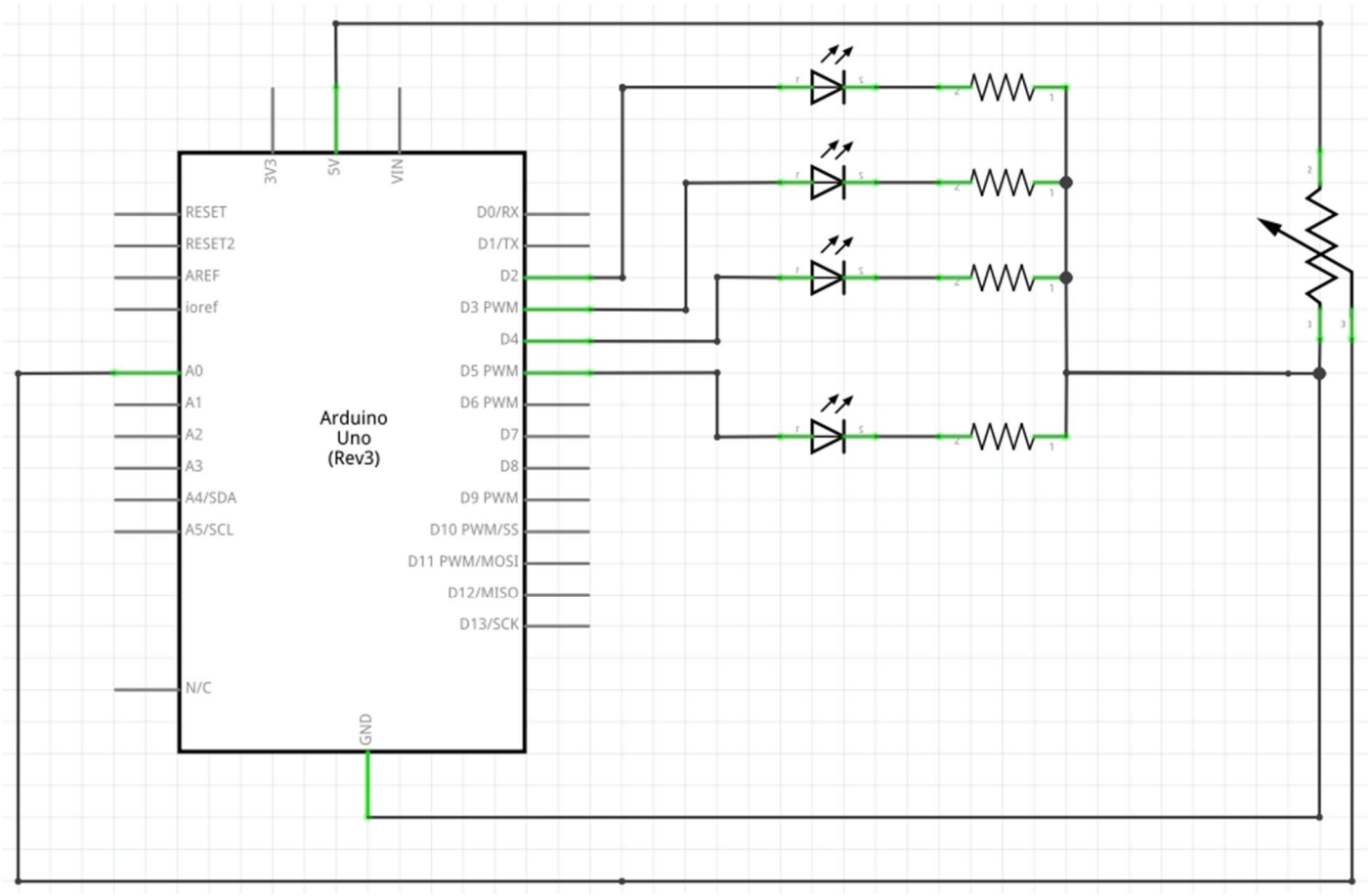
MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- Potenciómetro circular de 10K
- 4 LEDS rojos de 5mm
- 4 Resistencias de 330 Ohmios (naranja, naranja, marrón)
- 14 cables M-M

CIRCUITO



CALCULA Y MIDE

El Potenciómetro:

– Para controlar el encendido de LEDS y regular su brillo o luminosidad, utilizaremos un potenciómetro (dispositivo de entrada) y LEDS (dispositivos de salida). La información del dispositivo de entada (sensor) nos servirá para regular de forma continua la luminosidad del dispositivo de salida (actuador).

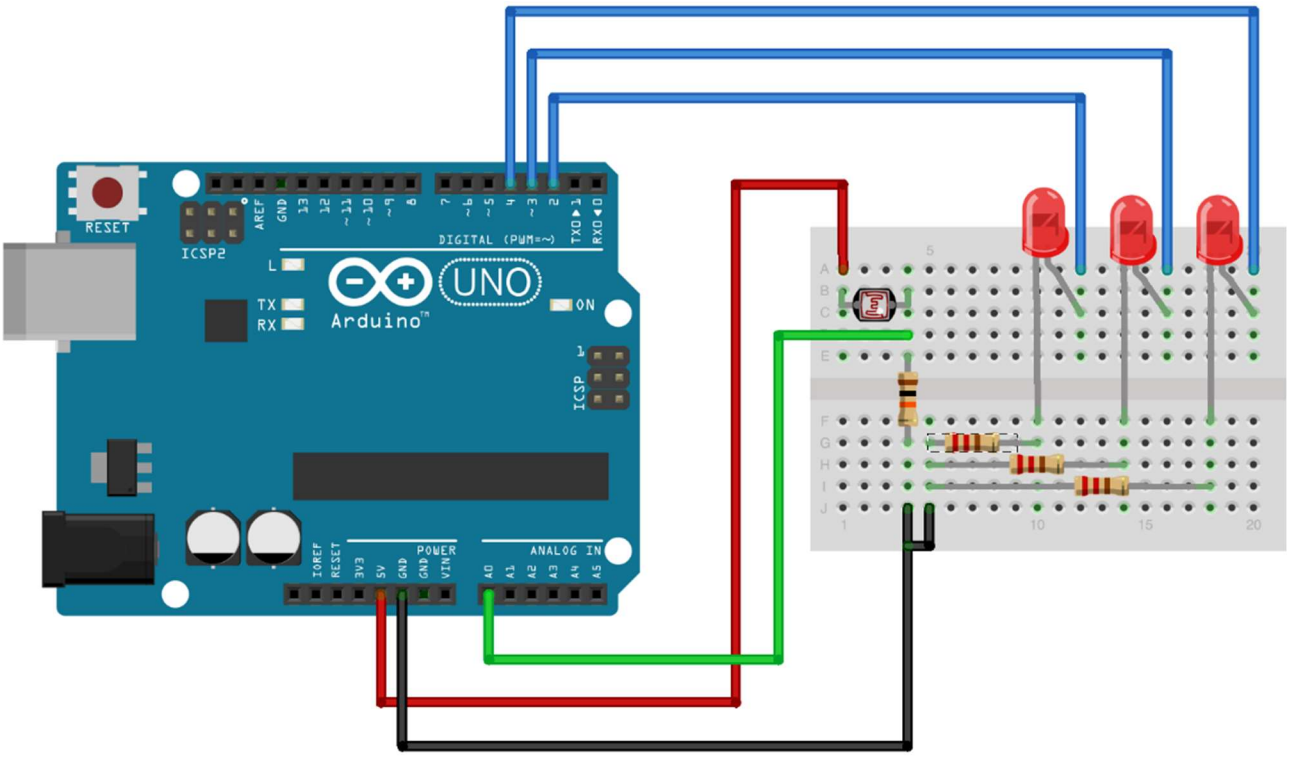
CÓDIGO

```
long valor;

//Declaramos los pins de los LEDs
int LED_1 = 2;
int LED_2 = 3;
int LED_3 = 4;
int LED_4 = 5;

void setup() {
  //Inicializamos la comunicación serial
  Serial.begin(9600);

  //Escribimos por el monitor serie mensaje de inicio
  Serial.println("Inicio de sketch - valores del potenciometro");
}

void loop() {
  // leemos del pin A0 valor
  valor = analogRead(A0);

  //Imprimimos por el monitor serie
  Serial.print("El valor es = ");
  Serial.println(valor);

  if(valor >= 0 && valor <=255)
  {
      digitalWrite(LED_1, HIGH);
      digitalWrite(LED_2, LOW);
      digitalWrite(LED_3, LOW);
      digitalWrite(LED_4, LOW);
  }

  if (valor >= 256 && valor <=511)
  {
      digitalWrite(LED_1, LOW);
```

```
      digitalWrite(LED_2, HIGH);
      digitalWrite(LED_3, LOW);
      digitalWrite(LED_4, LOW);
  }

  if (valor >= 512 && valor <=767)
  {
      digitalWrite(LED_1, LOW);
      digitalWrite(LED_2, LOW);
      digitalWrite(LED_3, HIGH);
      digitalWrite(LED_4, LOW);
    }

  if (valor >= 768 && valor <=1023)
  {
      digitalWrite(LED_1, LOW);
      digitalWrite(LED_2, LOW);
      digitalWrite(LED_3, LOW);
      digitalWrite(LED_4, HIGH);
   }
}
```

//13 LA FOTORRESISTENCIA.

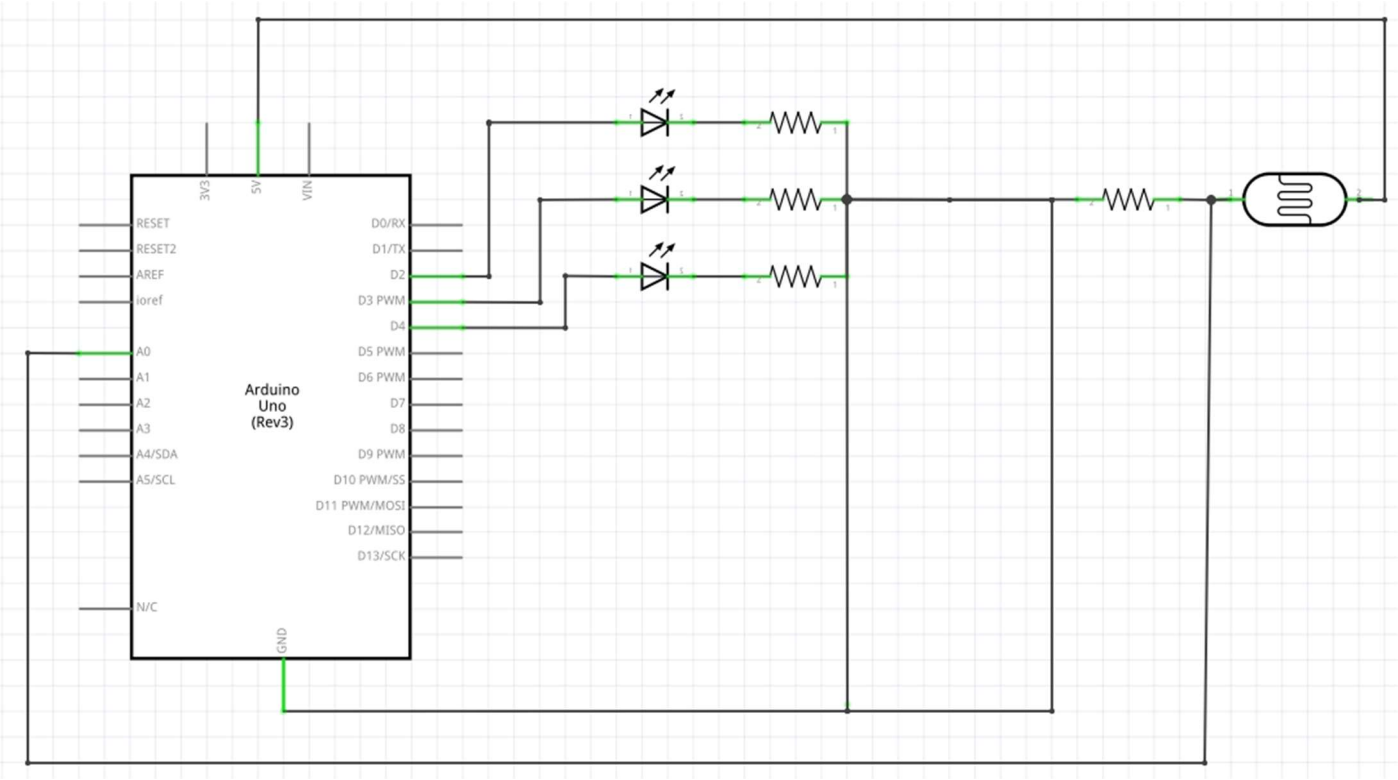
MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- LDR GL5516
- 3 LEDS rojos de 5mm
- 3 Resistencias de 330 Ohmios (naranja, naranja, marrón)
- 1 Resistencia de 10K (marrón, negro, naranja)
- 7 cables M-M

CIRCUITO



CALCULA Y MIDE

La Fotorresistencia (LDR):
- El LDR es una resistencia cuyo valor óhmico disminuye según aumenta la cantidad de luz que recibe.
- Necesitamos un mecanismo para medir las variaciones de la resistencia del circuito, y para ello se suele usar el circuito conocido como divisor de tensión: conectamos el LDR a otra resistencia, y medimos la salida en medio de estas dos resistencias. Cuando disminuya la resistencia del LDR, disminuirá este voltaje, y viceversa.

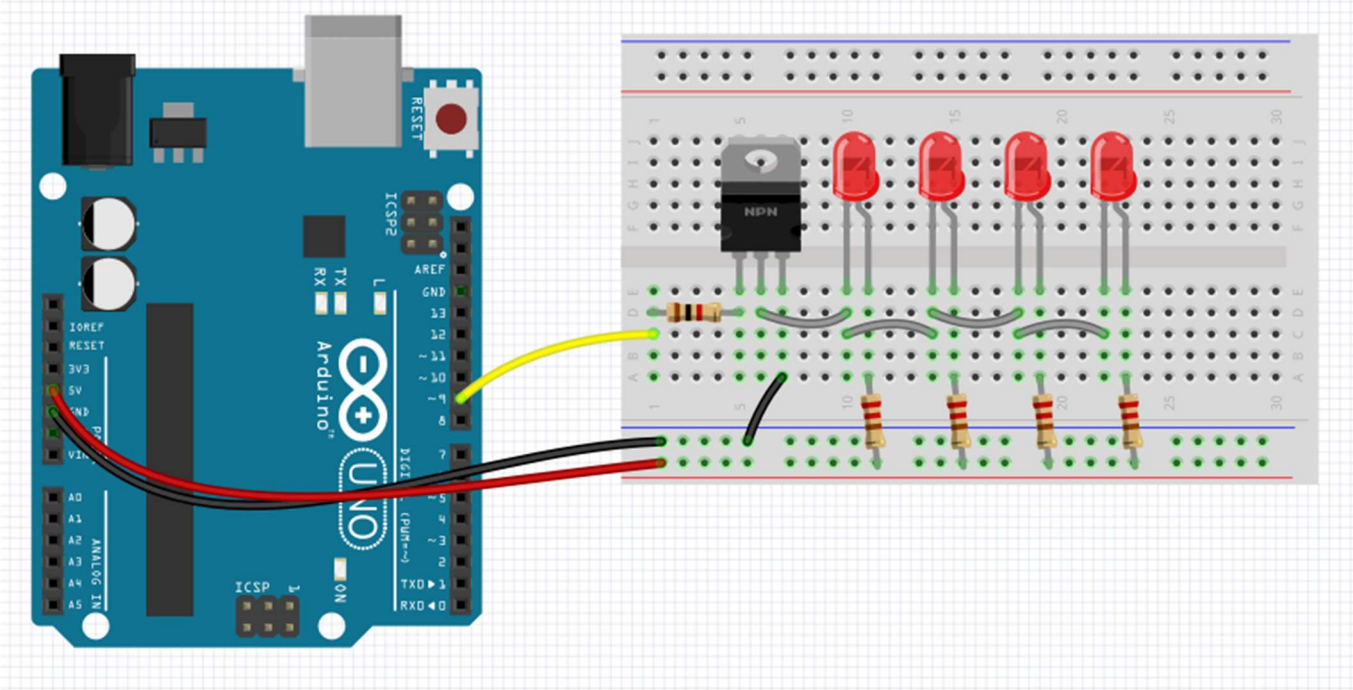
CÓDIGO

```
// Pin donde se conectan los leds
int pinLed1 = 2;
int pinLed2 = 3;
int pinLed3 = 4;
// Pin analogico de entrada para el LDR
int pinLDR = 0;

// Variable donde se almacena el valor del LDR
int valorLDR = 0;

void setup()
{
  // Configuramos como salidas los pines donde se conectan los led
  pinMode(pinLed1, OUTPUT);
  pinMode(pinLed2, OUTPUT);
  pinMode(pinLed3, OUTPUT);

  //  Configurar el puerto serial
  Serial.begin(9600);
}

void loop()
{
  // Apagar todos los leds siempre que se inicia el ciclo
  digitalWrite(pinLed1, LOW);
  digitalWrite(pinLed2, LOW);
  digitalWrite(pinLed3, LOW);

  // Guardamos el valor leido del ADC en una variable
  // El valor leido por el ADC (voltaje) aumenta de manera directamente proporcional
  // con respecto a la luz percibida por el LDR
  valorLDR= analogRead(pinLDR);

  // Devolver el valor leido a nuestro monitor serial en el IDE de Arduino
  Serial.println(valorLDR);
```

```
  // Encender los leds apropiados de acuerdo al valor de ADC
  if(valorLDR > 256)
  {
    digitalWrite(pinLed1, HIGH);
  }
  if(valorLDR > 512)
  {
    digitalWrite(pinLed2, HIGH);
  }
  if(valorLDR > 768)
  {
    digitalWrite(pinLed3, HIGH);
  }
  // Esperar unos milisegundos antes de actualizar
  delay(200);
}
```

//14 EL TRANSISTOR.

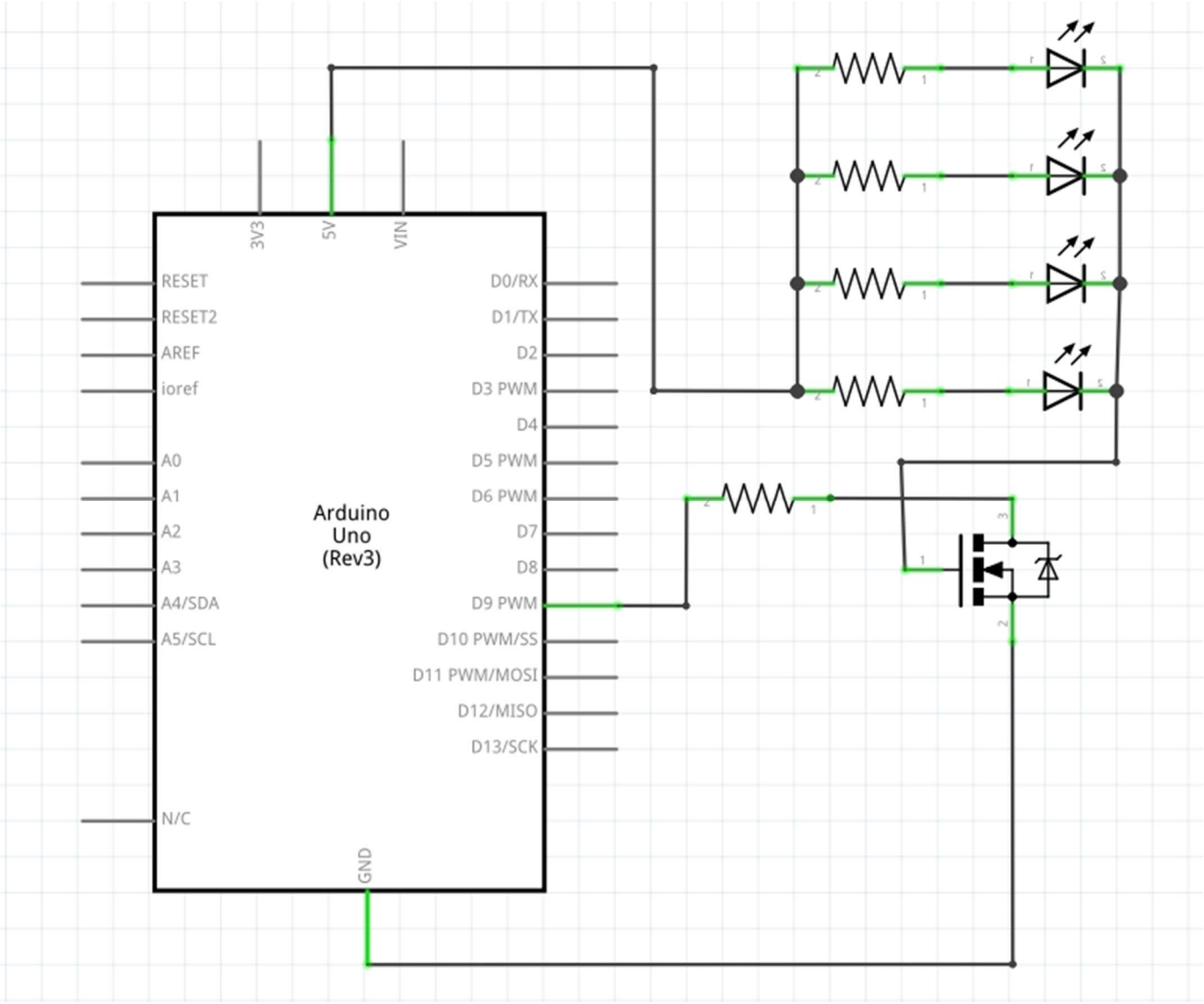
MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- TRANSISTOR MOSFET IRF540
- 4 LEDS rojos de 5mm
- 4 Resistencias de 330 Ohmios (naranja, naranja, marrón)
- 1 Resistencia de 10K (marrón, negro, naranja)
- 8 cables M-M

CIRCUITO



CALCULA Y MIDE

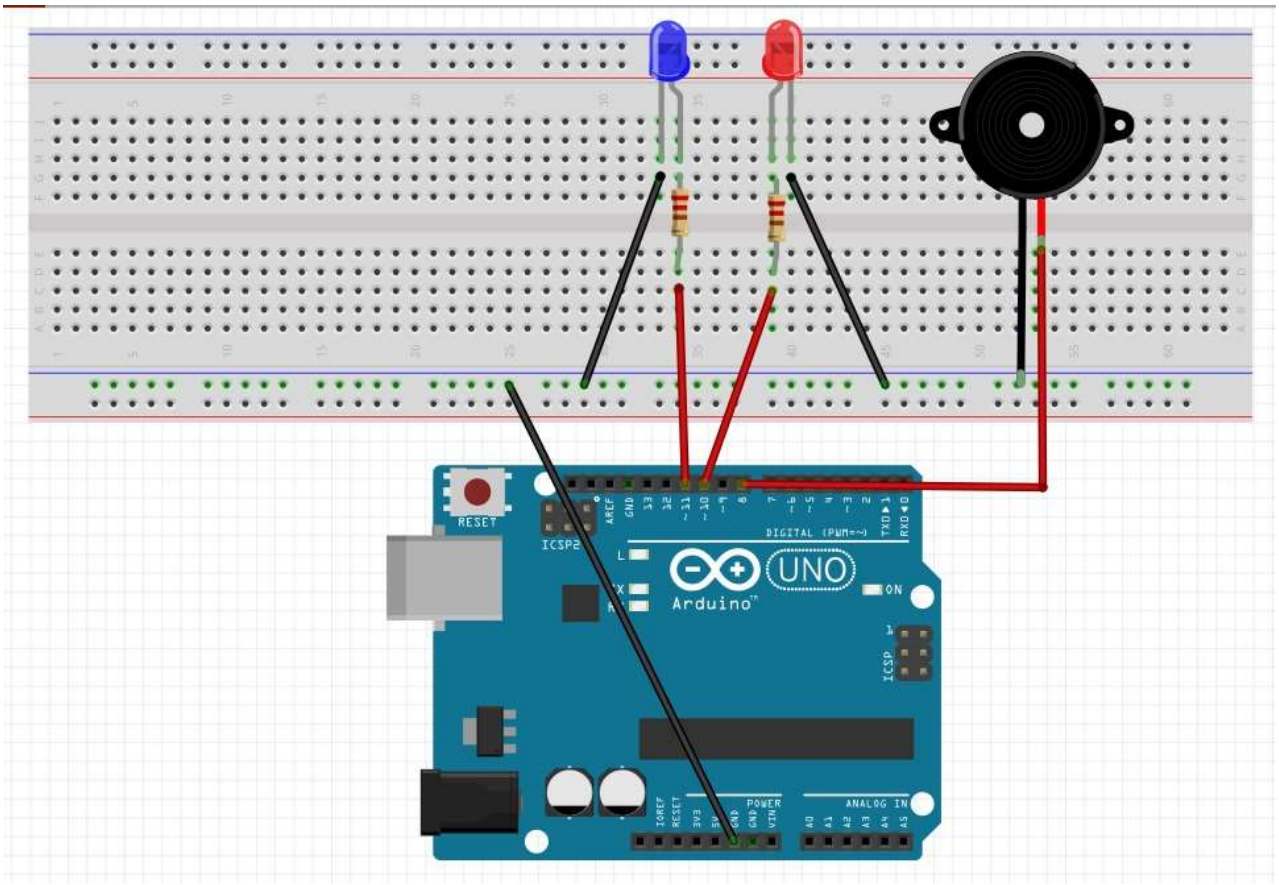
Transistor:
- Cuando queremos controlar actuadores (dispositivos de salida) de mayor potencia, necesitamos interponer entre la salida de la placa Arduino y la carga, un dispositivo llamado driver.
- Un ejemplo de driver/controlador de actuadores es el transistor MOSFET.

CÓDIGO

```
const int transistorPin = 9;    // connected to the base of the transistor


 void setup() {
   // set  the transistor pin as output:
   pinMode(transistorPin, OUTPUT);
 }

 void loop() {
   digitalWrite(transistorPin, HIGH);
   delay(1000);
   digitalWrite(transistorPin, LOW);
   delay(1000);
 }
```

//15 BANDA SONORA DE STAR WARS CON UN ZUMBADOR.

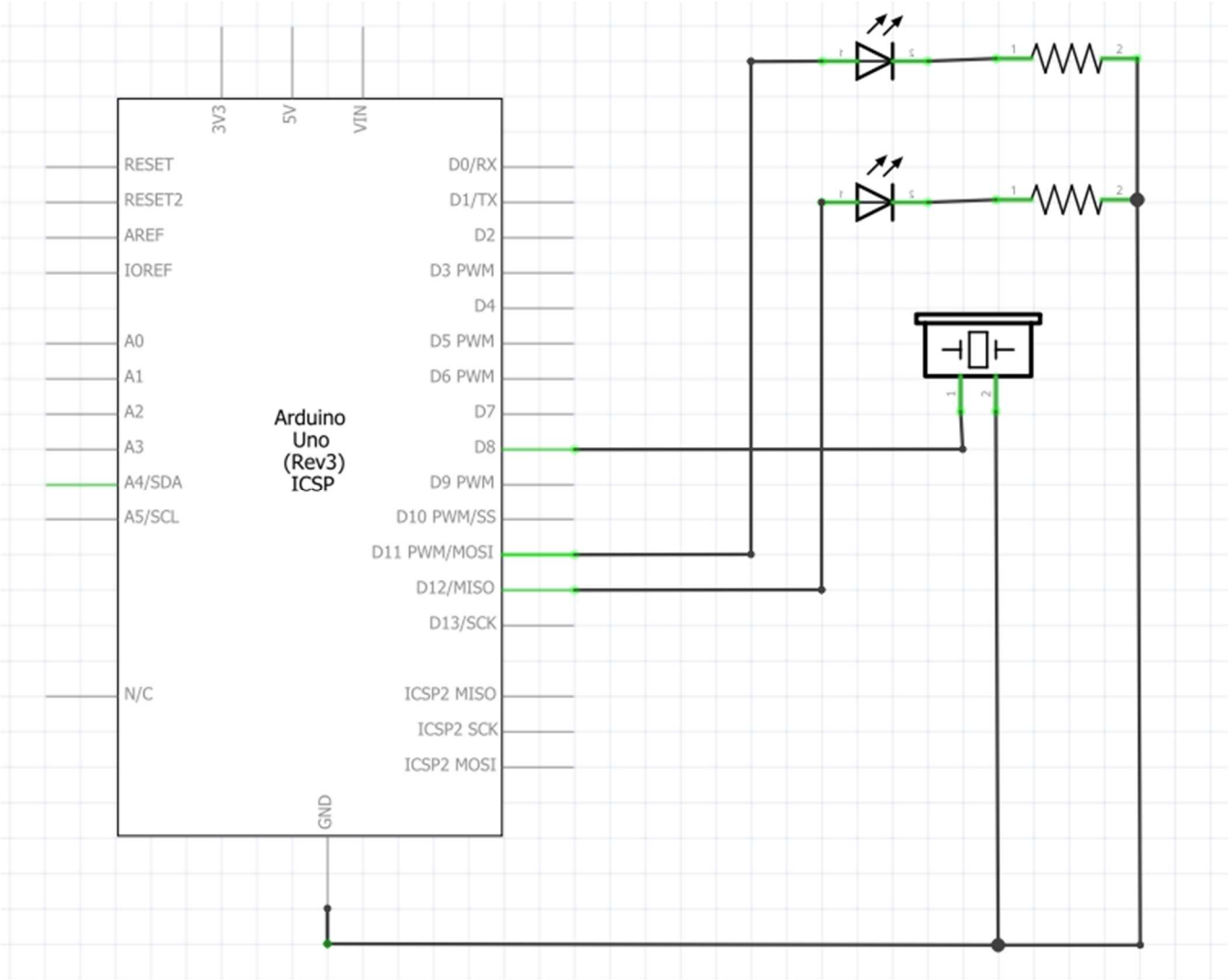
MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- ZUMBADOR PIEZOELÉCTRICO
- 2 LEDS de 5mm
- 2 Resistencias de 330 Ohmios (naranja, naranja, marrón)
- 7 cables M-M

CIRCUITO



CALCULA Y MIDE

El Zumbador piezoeléctrico:

- Cuando alimentamos un zumbador activo, éste emite una frecuencia característica. Por el contrario, un zumbador pasivo se comporta como un altavoz, emitiendo el sonido correspondiente a la frecuencia de la señal que se le aplica.
- Un zumbador activo consta de dos patas y un cuerpo donde se encuentran el oscilador y la membrana de emisión. Este tipo de zumbadores tiene polaridad: la pata positiva está marcada con un "+" en la parte superior del mismo.

CÓDIGO

```
// Declaración de las frecuencias de las notas musicales
const float c = 261.63; // Do (Octava 0)
const float d = 293.66; // Re (Octava 0)
const float e = 329.63; // Mi (Octava 0)
const float f = 349.23; // Fa (Octava 0)
const float g = 392.00; // Sol (Octava 0)
const float gS = 415.30; // Sol# (Octava 0)
const float a = 440.00; // La (Octava 0)
const float b = 466.16; // La# (Octava 0)
const float cH = 523.25; // Do (Octava 1)
const float cSH = 554.37; // Do# (Octava 1)
const float dH = 587.33; // Re (Octava 1)
const float dSH = 622.25; // Re# (Octava 1)
const float eH = 659.26; // Mi (Octava 1)
const float fH = 698.46; // Fa (Octava 1)
const float fSH = 739.99; // Fa# (Octava 1)
const float gH = 783.99; // Sol (Octava 1)
const float gSH = 830.61; // Sol# (Octava 1)
const float aH = 880.00; // La (Octava 1)
const int zumbador = 8; // Pin digital para el zumbador
const int LED_1 = 11; // Pin digital para el LED 1
const int LED_2 = 12; // Pin digital para el LED 2
int contador = 0;
void setup()
{
pinMode(zumbador, OUTPUT); // Pin digital 8 como salida
pinMode(LED_1, OUTPUT); // Pin digital 12 como salida
pinMode(LED_2, OUTPUT); // Pin digital 13 como salida
}
void loop()
{
// Suena la primera sección
primeraSeccion();
// Suena la segunda sección
segundaSeccion();
```

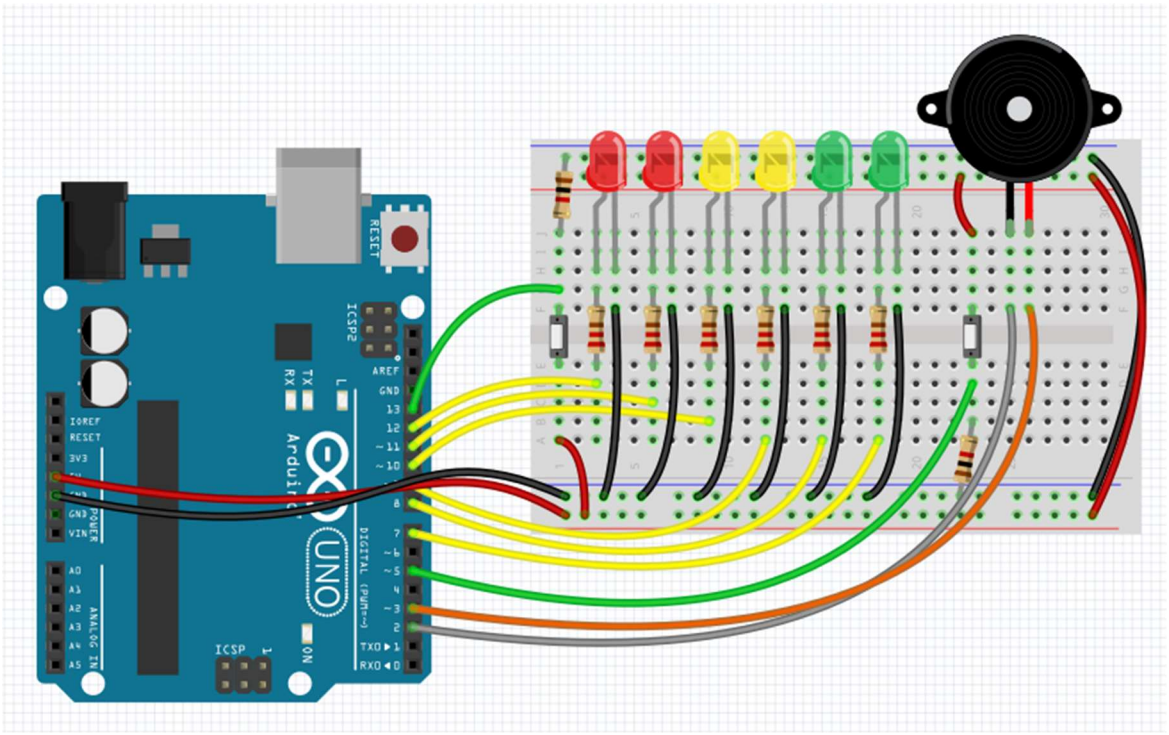
```
// Variante 1
tono(f, 250);
tono(gS, 500);
tono(f, 350);
tono(a, 125);
tono(cH, 500);
tono(a, 375);
tono(cH, 125);
tono(eH, 650);
delay(500);
// Se repite la segunda sección
segundaSeccion();
// Variante 2
tono(f, 250);
tono(gS, 500);
tono(f, 375);
tono(cH, 125);
tono(a, 500);
tono(f, 375);
tono(cH, 125);
tono(a, 650);
delay(650);
}
// Función que ejecuta cada tono
void tono(int frecuencia, int duracion)
{
// Suena el tono en el zumbador
tone(zumbador, frecuencia, duracion);
// Se enciende LED_1 o LED_2 alternamente cada vez que suena un nuevo tono
if(contador % 2 == 0)
{
digitalWrite(LED_1, HIGH);
delay(duracion);
digitalWrite(LED_1, LOW);
}
else
{
digitalWrite(LED_2, HIGH);
```

```
delay(duracion);
digitalWrite(LED_2, LOW);
}
// Para de sonar el tono en el zumbador
noTone(zumbador);
delay(50);
// Se incrementa el contador
contador++;
}
// Función de la primera sección
void primeraSeccion()
{
tono(a, 500);
tono(a, 500);
tono(a, 500);
tono(f, 350);
tono(cH, 150);
tono(a, 500);
tono(f, 350);
tono(cH, 150);
tono(a, 650);
delay(500);
tono(eH, 500);
tono(eH, 500);
tono(eH, 500);
tono(fH, 350);
tono(cH, 150);
tono(gS, 500);
tono(f, 350);
tono(cH, 150);
tono(a, 650);
delay(500);
}
// Función de la segunda sección
void segundaSeccion()
{
tono(aH, 500);
tono(a, 300);
```

```
tono(a, 150);
tono(aH, 500);
tono(gSH, 325);
tono(gH, 175);
tono(fSH, 125);
tono(fH, 125);
tono(fSH, 250);
delay(325);
tono(a, 250);
tono(dSH, 500);
tono(dH, 325);
tono(cSH, 175);
tono(cH, 125);
tono(b, 125);
tono(cH, 250);
delay(350);
}
```

//16 EL PULSADOR. LA PISTOLA DE RAYOS CÓSMICOS.

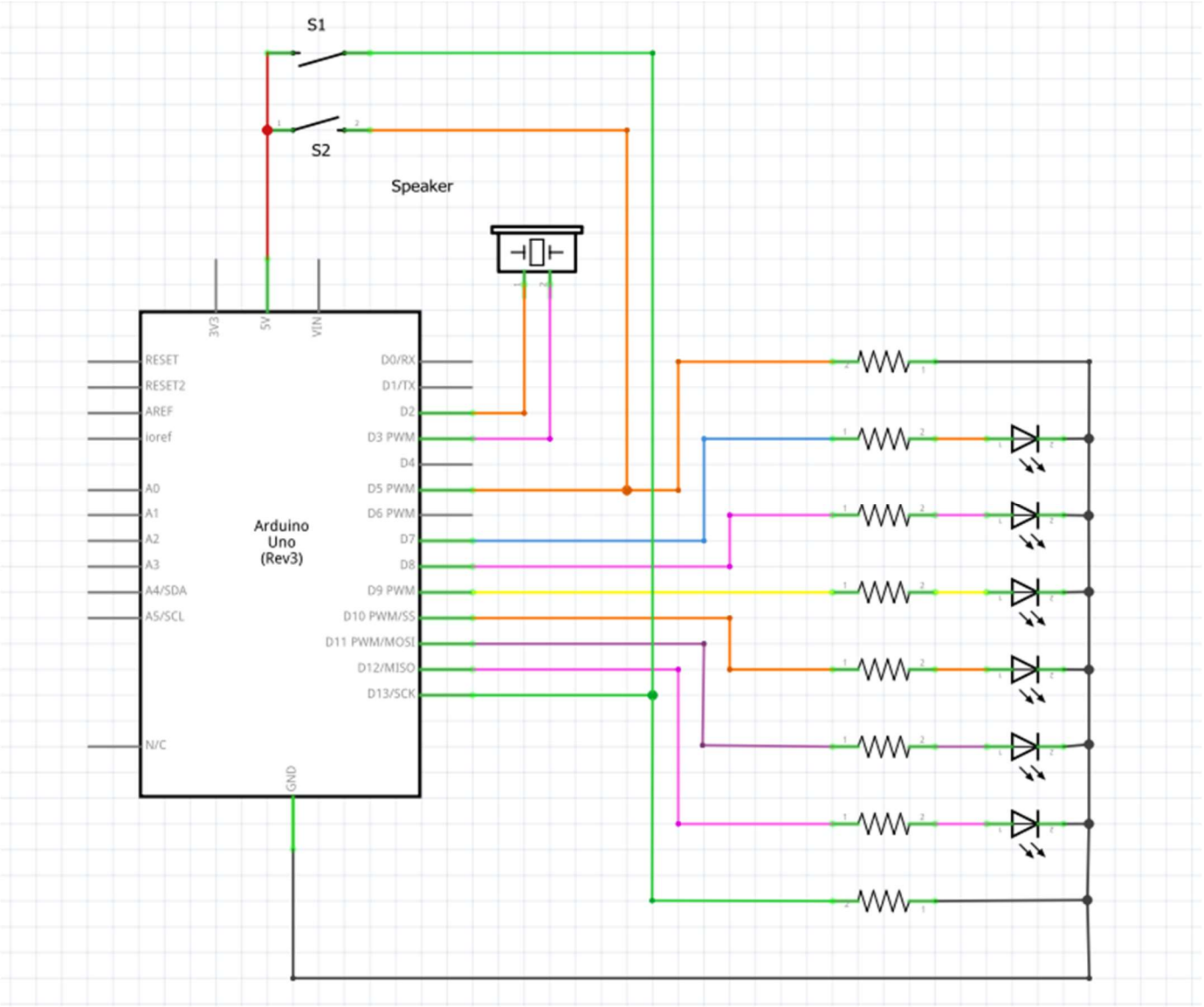
MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- ZUMBADOR PIEZOELÉCTRICO
- 2 PULSADORES para placa (2 terminales)
- 2 LEDS rojos de 5mm
- 2 LEDS amarillos de 5mm
- 2 LEDS verdes de 5mm
- 6 Resistencias de 330 Ohmios (naranja, naranja, marrón)
- 2 Resistencias de 10K (marrón, negro, naranja)
- 22 cables M-M

CIRCUITO



CALCULA Y MIDE

El Pulsador:
- El pulsador, al igual que el interruptor, es un elemento que nos permite cerrar y abrir el circuito eléctrico. A diferencia del interruptor, sólo mantiene una posición estable (normalmente la de circuito abierto). Una vez que lo pulsamos, cambia de posición, y la mantiene mientras apliquemos la fuerza.
- Tiene dos patas y un pequeño accionador para cerrar el circuito al pulsar.

CÓDIGO

```
/*
Cosmic Raygun
*/
int speakerPin=3;
int speakerGndPin=2;
int fireBtn=5;
int reloadBtn=13;
int ledPins[]={7,8,9,10,11,12};
int ledCount=6;
int fireTime=300;
long fireTimer;
int reloadTime=1600;
long reloadTimer;
int myState=0;
void setup(){
  Serial.begin(57600);
  pinMode(speakerPin,OUTPUT);
  pinMode(fireBtn,INPUT);
  pinMode(reloadBtn,INPUT);
  for (int i=0; i<6; i++){
    pinMode(ledPins[i],OUTPUT);
  }
  analogWrite(speakerGndPin,120);
}
void fire(){
  tone(speakerPin,8000+(fireTime-((int)millis()-fireTimer)*40));
  delay(1);
  int myVal=(int)millis()-fireTimer;
  //ledCount=0;
  if (myVal>(fireTime/12)*6){
    ledCount=1;
  } else if (myVal>(fireTime/12)*5){
    ledCount=2;
  } else if (myVal>(fireTime/12)*4){
    ledCount=3;
```

```
  } else if (myVal>(fireTime/12)*3){
    ledCount=4;
  } else if (myVal>(fireTime/12)*2){
    ledCount=5;
  } else if (myVal>(fireTime/12)*1){
    ledCount=6;
  } }
void reload(){
  tone(speakerPin,6000+(reloadTime+((int)millis()-reloadTimer)*2));
  delay(1);
  Serial.print("\t");Serial.print(reloadTimer);Serial.print("\t");Serial.print((int)millis()-re
loadTimer);
  int myVal=(int)millis()-reloadTimer;
  if (myVal>reloadTime){
    ledCount=6;
  } else if (myVal>((reloadTime/6)*5)){
    ledCount=5;
  } else if (myVal>((reloadTime/6)*4)){
    ledCount=4;
  } else if (myVal>((reloadTime/6)*3)){
    ledCount=3;
  } else if (myVal>((reloadTime/6)*2)){
    ledCount=2;
  } else if (myVal>((reloadTime/6)*1)){
    ledCount=1;
  } }
void showMyLights(){
  for (int i=0; i<6; i++){
    digitalWrite(ledPins[i],LOW);
  }
  for (int i=0; i<ledCount; i++){
    digitalWrite(ledPins[i],HIGH);
  }}
void loop(){
  Serial.print(millis());
  showMyLights();
  switch (myState){
   case 0:          // unlocked and loaded
```

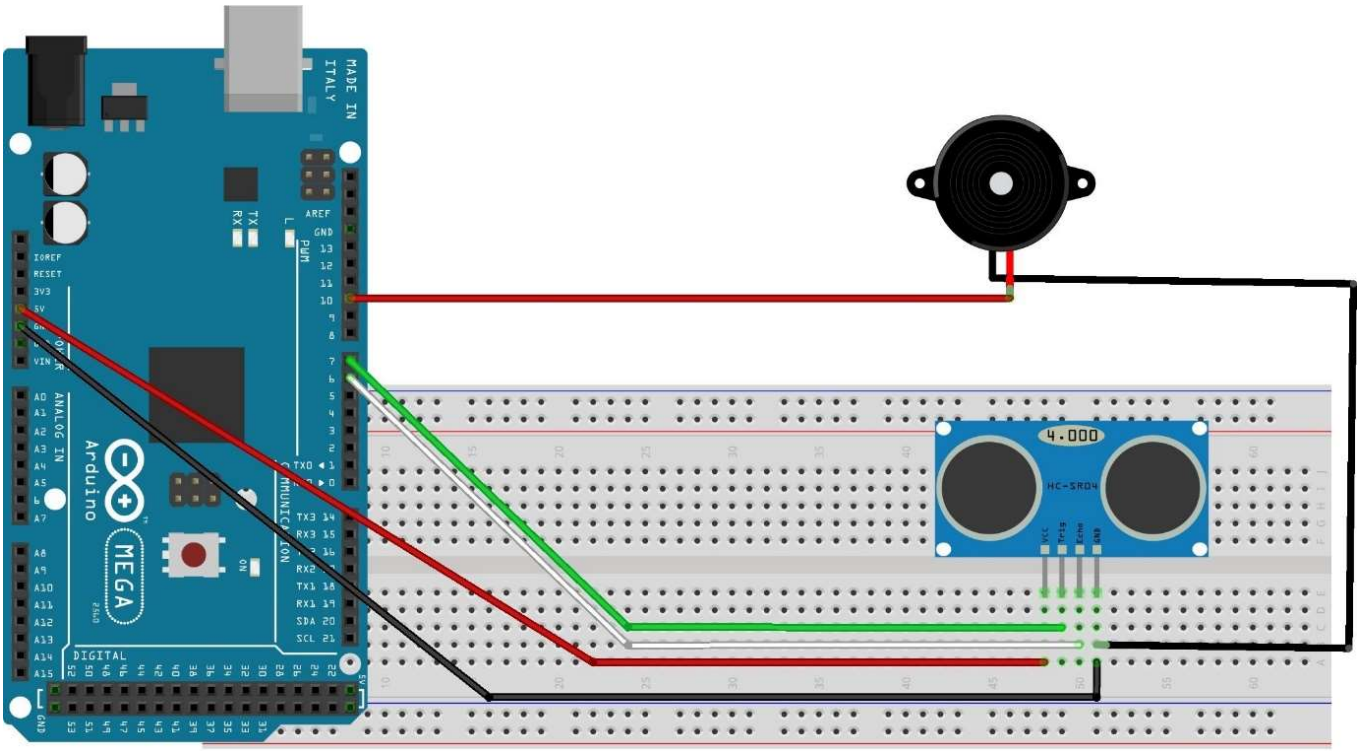
```
   {
    ledCount=6;
    noTone(speakerPin);
    if (digitalRead(fireBtn)){
      fireTimer=millis();
      myState=1;
    }    break;   }
   case 1:          // fire
    {
    if (fireTimer+fireTime>millis()){
      fire();
    } else {
      myState=2;
    }   break;    }
   case 2:          // unloaded
    {
      ledCount=0;
        noTone(speakerPin);
    if (digitalRead(reloadBtn)){
      reloadTimer=millis();
      myState=3;
    }      break;    }
   case 3:          // reloading
    {
    if (reloadTimer+reloadTime>millis()){
      reload();
    } else {
      myState=0;
    }    break;    }   }
  Serial.println("");
  delay(1);
}
```

//17 NIVEL 2: COMPONENTES.

// NIVEL 2: SENSORES, MOTORES, DISPLAYS, LCD.

//18 SENSOR DE PROXIMIDAD. APARCANDO EL COCHE.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- SENSOR DE DISTANCIA POR ULTRASONIDOS HC-SR04
- ZUMBADOR PIEZOELÉCTRICO
- 6 cables M-M

CALCULA Y MIDE

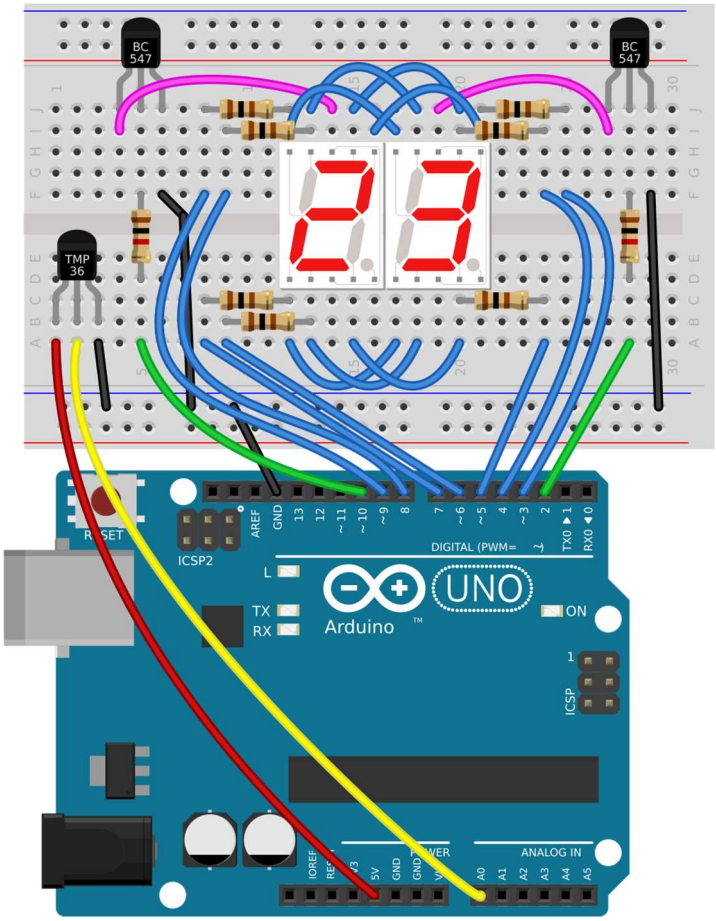
El sensor ultrasónico HC-SR04:
- El sensor emite ondas elásticas fuera del espectro audible (ultrasonidos de 40kHz), que se propagan por el aire a la velocidad del sonido (340m/s) rebotando en los objetos que encuentran a su paso.
- Dispone de un emisor (altavoz) y un receptor (micrófono), y tiene un ángulo de visión de 30º.
_ Conocido el tiempo de retardo de la onda rebotada y la velocidad a la que viaja ésta (la del sonido), podemos detectar objetos a una distancia determinada.

CÓDIGO

```
#define Pecho 6
#define Ptrig 7
#define buzzer 10  //define el puerto de salida del buzzer
#define minima 30  //define la distancia minima a la que el buzzer emite sonido
#define maxima 40 //define la distancia maxima a la que el buzzer emite sonido
long duracion, distancia;
void setup() {
  Serial.begin (9600);       // inicializa el puerto seria a 9600 baudios
  pinMode(Pecho, INPUT) ;     // define el pin 6 como entrada (echo)
  pinMode(Ptrig, OUTPUT) ;    // define el pin 7 como salida  (triger)
  pinMode(buzzer, OUTPUT) ;  // Define el pin 10 como salida
  }
void loop() {
  digitalWrite(Ptrig, LOW) ;
  delayMicroseconds(2) ;
  digitalWrite(Ptrig, HIGH) ;   // genera el pulso de triger por 10ms
  delayMicroseconds(10) ;
  digitalWrite(Ptrig, LOW) ;
  duracion = pulseIn(Pecho, HIGH) ;
  distancia = (duracion/2) / 29;            // calcula la distancia en centimetros
  if (distancia <= 0){                      // si la distancia es menor o igual a cero
    Serial.println("---" ) ;                  // no mide nada
  }
  else {
    Serial.print(distancia);           // envia el valor de la distancia por el puerto serial
   Serial.println("cm" ) ;              // le coloca a la distancia la unidad "cm"
  } delay (500);                      // espera 1/2 segundo y mide de nuevo
   if (distancia >= minima && distancia <= maxima){
 //si el objeto esta en el rango entre la minima y maxima distancia emite sonido
    analogWrite(buzzer,128) ; //emite sonido
    delay(500); //espera medio segundo
    analogWrite(buzzer,LOW) ; // no emite sonido
   }
}
```

//19 SENSOR DE TEMPERATURA. TERMÓMETRO DE AMBIENTE DIGITAL.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- 2 DISPLAYS led de 7 segmentos y cátodo común, tipo D5611
- SENSOR DE TEMPERATURA TMP36
- 2 TRANSISTORES BC547
- 7 Resistencias de 100 Ohmios (marrón, negro, marrón)
- 2 Resistencia de 1K (marrón, negro, rojo)
- 21 cables M-M

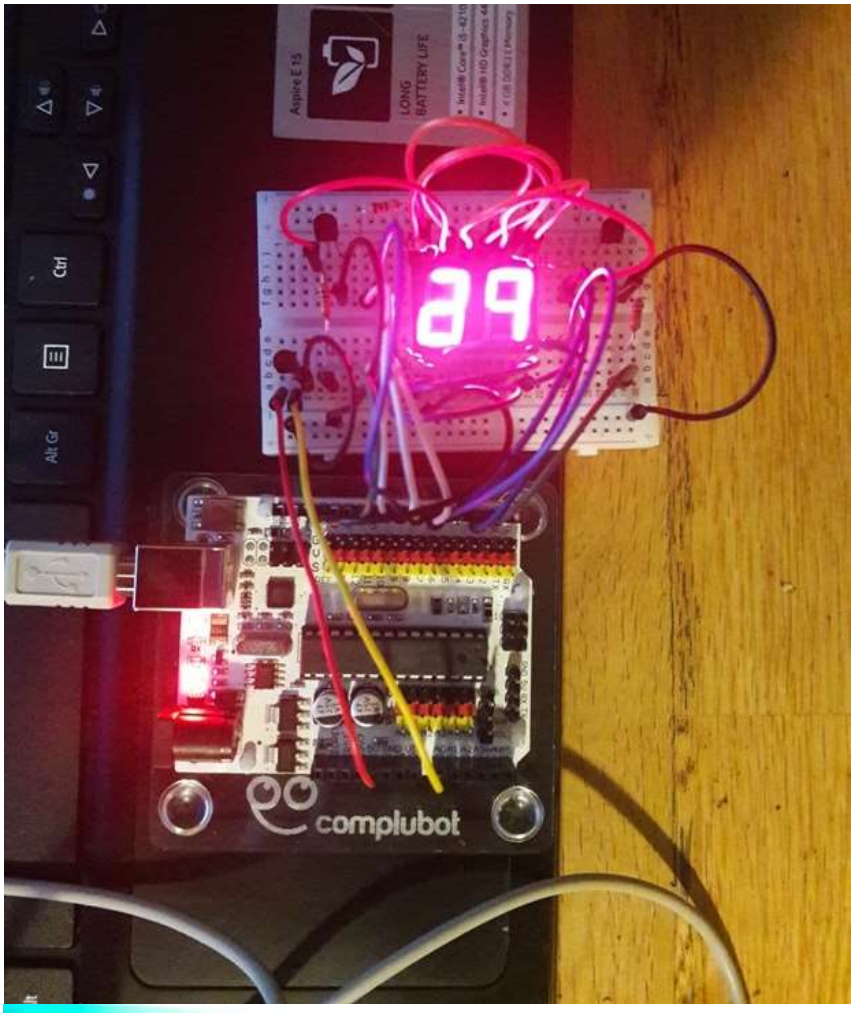
CALCULA Y MIDE

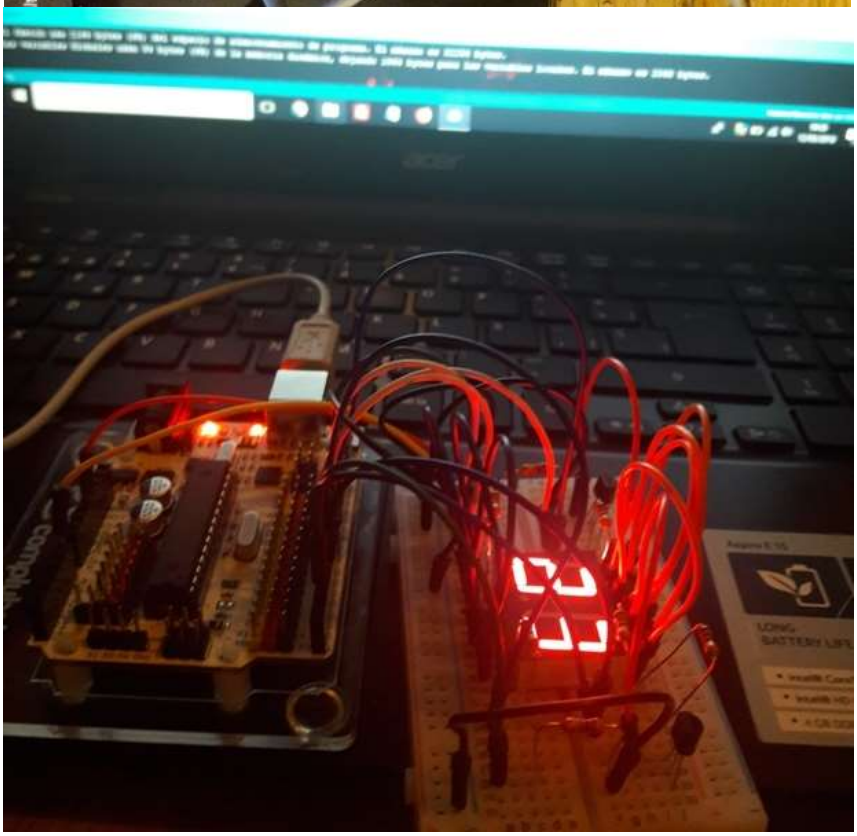
El sensor de temperatura TMP36:
- El TMP36 es un sensor de temperatura integrado que proporciona un voltaje de salida que varía de forma lineal con la temperatura. No requiere ningún tipo de calibración para proporcionar una precisión de +/-1ºC a +25ºC y de +/-2ºC en el rango completo de medida -20ºC a 125ºC.
- Este sensor proporciona un voltaje de salida de 750mV a 25ºC, con un factor de escala de 10mV/ºC.

CÓDIGO

```
// pines control display led

#define LCD_1 10
#define LCD_2 2

//pines conexion distintos segmentos del display led

#define SEG_A 4
#define SEG_B 3
#define SEG_C 5
#define SEG_D 6
#define SEG_E 7
#define SEG_F 8
#define SEG_G 9

//entrada analogica sensor temperatura

#define TMP36 0

// Matriz de codificacion

byte led_7s[10][7] = {
  {1, 1, 1, 1, 1, 1, 0}, // = 0
  {1, 0, 1, 0, 0, 0, 0}, // = 1
  {1, 1, 0, 1, 1, 0, 1}, // = 2
  {1, 1, 1, 1, 0, 0, 1}, // = 3
  {1, 0, 1, 0, 0, 1, 1}, // = 4
  {0, 1, 1, 1, 0, 1, 1}, // = 5
  {1, 1, 1, 1, 1, 0, 1}, // = 6
  {1, 1, 1, 0, 0, 0, 0}, // = 7
  {1, 1, 1, 1, 1, 1, 1}, // = 8
  {1, 1, 1, 0, 0, 1, 1}, // = 9
};
byte temperatura;
float voltaje;
```

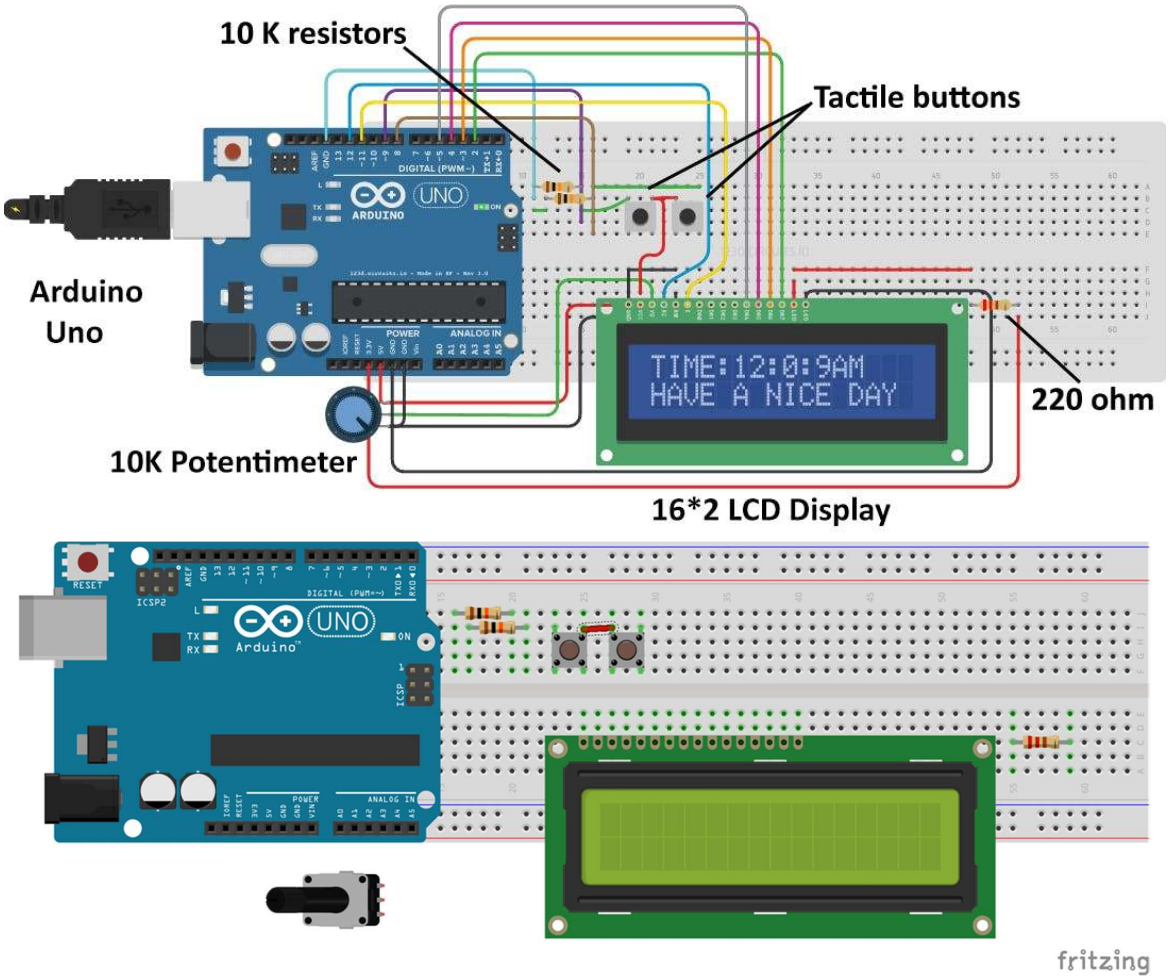
```
void setup()
{
  pinMode(LCD_2, OUTPUT);
  pinMode(LCD_1, OUTPUT);
  // config segmentos display
  pinMode(SEG_A, OUTPUT);
  pinMode(SEG_B, OUTPUT);
  pinMode(SEG_C, OUTPUT);
  pinMode(SEG_D, OUTPUT);
  pinMode(SEG_E, OUTPUT);
  pinMode(SEG_F, OUTPUT);
  pinMode(SEG_G, OUTPUT);
}
void loop()
{
  //lectura voltaje TMP36
  voltaje=analogRead(TMP36)*5.0/1024;
  //convertimos voltaje a temperatura: 10mC/ºC
  //desplazamiento 500mV
  temperatura =(voltaje*100.0)-50;
  //mostramos digito mas significativo
  visualiza_digi(LCD_1, temperatura/10);
  //mostramos digito menos significativo
  visualiza_digi(LCD_2, temperatura %10);
}
void visualiza_digi(byte pos, byte digi)
{
  byte segmento = SEG_B;
  for (byte contador = 0; contador <7; ++contador)
  {
    digitalWrite(segmento, led_7s[digi][contador]);
    ++segmento;
  }
  digitalWrite(pos, HIGH); //activamos digito
  delay(10); //espera visualizar digito
  digitalWrite(pos, LOW); // desactivamos digito
}
```


complubot



//20 EL RELOJ-DIGITAL.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- PANTALLA LCD 2x16, compatible con el controlador HD44780
- Potenciómetro 10K
- 2 PULSADORES para placa (2 terminales)
- 2 Resistencias de 10KΩ (marrón, negro, naranja)
- 1 Resistencia de 220Ω (rojo, rojo, marrón)
- 30 cables M-M

CALCULA Y MIDE

Reloj-digital:
- La pantalla LCD la utilizamos para mostrar la hora y fecha actual al igual que cualquier reloj digital tradicional.

LIBRERÍAS

Para cargar las librerías y utilizarlas, podemos subir los archivos a la carpeta libraries.

También podemos cargarlas desde un archivo zip o utilizar una ya existente, a través de la pestaña "Programa→incluir Librería"

Libraries

The Arduino environment can be extended through the use of libraries, just like most programming platforms. Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from Sketch > Import Library.
A number of libraries come installed with the IDE, but you can also download or create your own. See these instructionsfor details on installing libraries. There's also a tutorial on writing your own libraries. See the API Style Guide for information on making a good Arduino-style API for your library.

Standard Libraries

- EEPROM - reading and writing to "permanent" storage
- Ethernet / Ethernet 2 - for connecting to the internet using the Arduino Ethernet Shield, Arduino Ethernet Shield 2 and Arduino Leonardo ETH
- Firmata - for communicating with applications on the computer using a standard serial protocol.
- GSM - for connecting to a GSM/GRPS network with the GSM shield.
- LiquidCrystal - for controlling liquid crystal displays (LCDs)
- SD - for reading and writing SD cards
- Servo - for controlling servo motors
- SPI - for communicating with devices using the Serial Peripheral Interface (SPI) Bus
- SoftwareSerial - for serial communication on any digital pins. Version 1.0 and later of Arduino incorporate Mikal Hart's NewSoftSerial library as SoftwareSerial.
- Stepper - for controlling stepper motors
- TFT - for drawing text , images, and shapes on the Arduino TFT screen
- WiFi - for connecting to the internet using the Arduino WiFi shield
- Wire - Two Wire Interface (TWI/I2C) for sending and receiving data over a net of devices or sensors.

The Matrix and Sprite libraries are no longer part of the core distribution.

- 101 Only Libraries
- Due Only Libraries
- Due, Zero and MKR1000 Libraries
- Zero, MKRZERO and MKR1000 Libraries
- WiFi 101 and MKR1000 Library
- MKR FOX 1200 only Library
- MKR WAN 1300 only Library
- MKR GSM 1400 only Library
- Esplora Only Library
- Arduino Robot Library
- Yún devices Library
- USB Libraries (Leonardo, Micro, Due, Zero and Esplora)
- Contributed Libraries
- Sensing
- Displays and LEDs
- Audio and Waveforms
- Motors and PWM
- Timing

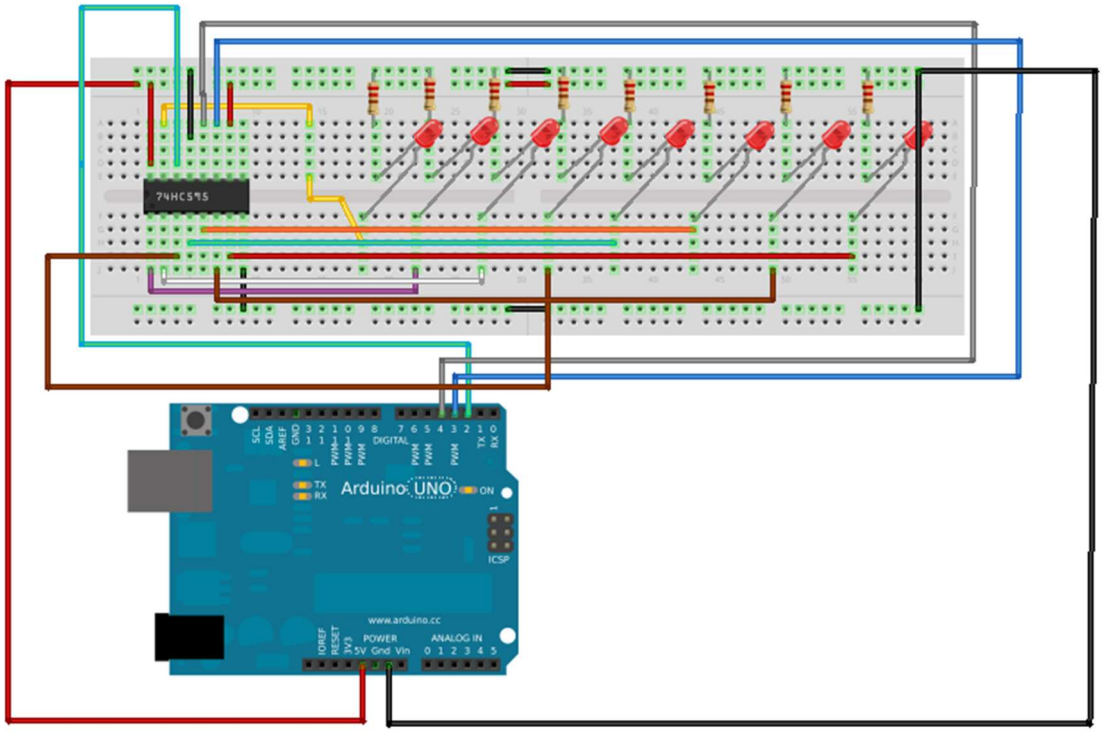
CÓDIGO

```
/*
Digital Winding Clock. No RTC.
Just set it to exact time using the push buttons each time you switch on the clock
*/
#include
LiquidCrystal
lcd(12,11,5,4,3,2);
int h=12;
int m;
int s;
int flag;
int TIME;
const int hs=8;
const int ms=9;
int state1;
int state2;
void setup()
{
 lcd.begin(16,2);
}
void loop()
{
lcd.setCursor(0,0);
s=s+1;
lcd.print("TIME:");
lcd.print(h);
lcd.print(":");
lcd.print(m);
lcd.print(":");
lcd.print(s);
if(flag<12)lcd.print("AM");
if(flag==12)lcd.print("PM");
if(flag>12)lcd.print("PM");
if(flag==24)flag=0;
delay(1000);
```

```
lcd.clear();
if(s==60){
 s=0;
 m=m+1;
}
if(m==60)
{
 m=0;
 h=h+1;
 flag=flag+1;
}
if(h==13)
{
 h=1;
}
lcd.setCursor(0,1);
lcd.print("HAVE A NICE DAY");
//-------Time
// setting-------//
state1=digitalRead(hs);
if(state1==1)
{
 h=h+1;
 flag=flag+1;
 if(flag<12)lcd.print("AM");
 if(flag==12)lcd.print("PM");
 if(flag>12)lcd.print("PM");
 if(flag==24)flag=0;
 if(h==13)h=1;
}
state2=digitalRead(ms);
if(state2==1){
 s=0;
 m=m+1;
}
}
```

//21 REGISTROS DE DESPLAZAMIENTOS.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- 74HC595
- 8 LEDS rojos de 3mm
- 8 Resistencias de 330 Ohmios (naranja, naranja, marrón)
- 25 cables M-M

CALCULA Y MIDE

Registros de desplazamiento:
- En ocasiones podemos necesitar más salidas digitales de las que tenga disponible la placa. Una solución es utilizar un registro de desplazamiento con entrada en serie y salida en paralelo, utilizando tan solo 3 salidas digitales de la placa y proporcionando 8 salidas digitales ampliables.
- Uno de los registros de desplazamiento más populares es el 74HC595.

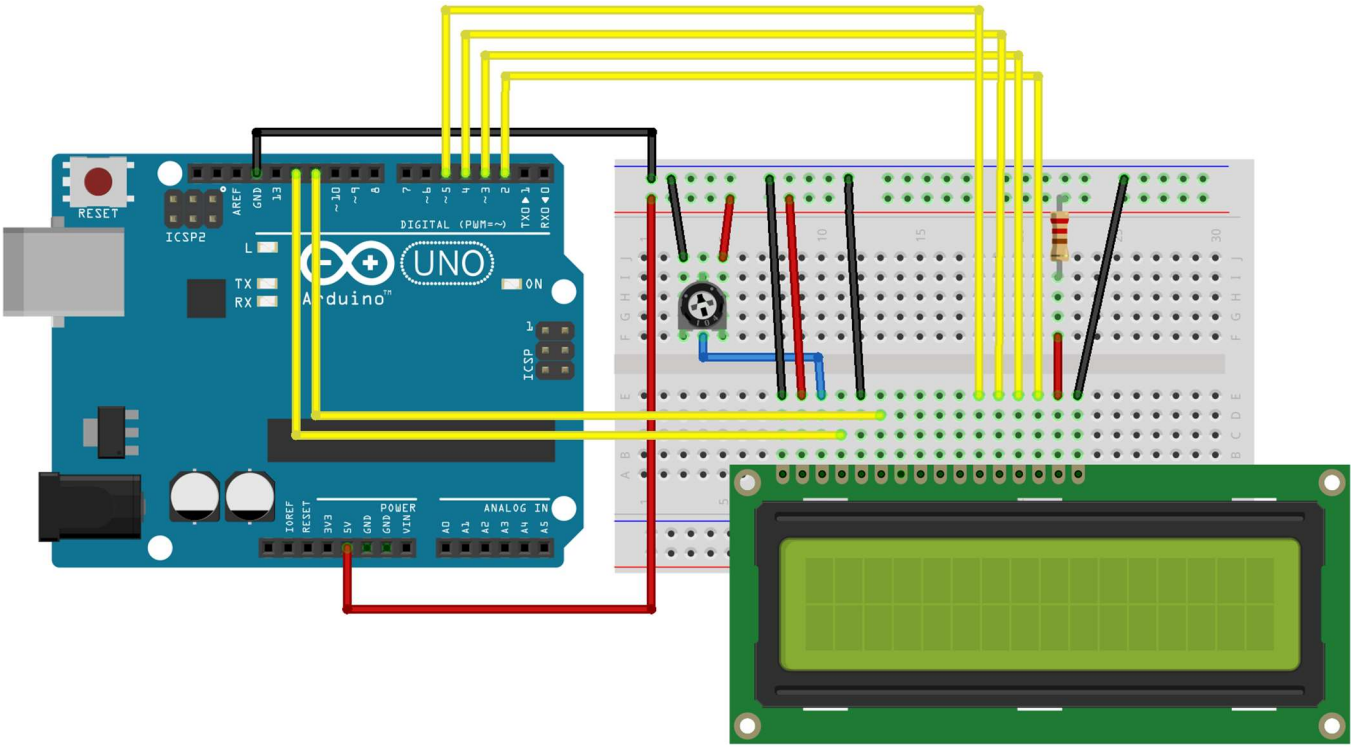
CÓDIGO

```
int ds = 2; //data
int shcp = 3; // clock
int stcp = 4; //latch
int data = 0;
//importante. Variables del shift register
boolean pinState[] = {0,0,0,0,0,0,0,0};  //Matriz que contiene los valores actuales
void setup() {
  pinMode(ds, OUTPUT);      //Establecer los pines a utilizar
  pinMode(shcp, OUTPUT);
  pinMode(stcp, OUTPUT);
  Serial.begin(9600); //Iniciar el monitor serie
  for(int x=0;x<8;x++)
  {    digital_Write(x,LOW);     //Apagar todos los pines
  }}
void loop() {
  if(Serial.available()>0)
  {
    String y= Serial.readString();  //Leer un valor entre 0 y 7
    int x = y.toInt();  //Se transforma el dato recibido en un entero
    if(x>=0 && x<8)
    {
      pinState[x]=!pinState[x];     //Matriz con nuestros valores
      digital_Write(x,pinState[x]); //Analogo a digitalWrite
      Serial.print("Pin: ");
      Serial.print(x);              // Variable que contiene el pin
      Serial.print(", Estado: ");
      Serial.println((pinState[x]==1 ? "ENCENDIDO" : "APAGADO"));
    }  }}
void digital_Write(int myPin, boolean myState) //Funcion principal
//Analogo a digitalWrite(), para el shift register.
//myState = HIGH o LOW
{ bitWrite(data,myPin,myState);  //Transformar nuestros datos
  shiftOut(ds, shcp, MSBFIRST, data);  //Enviar al integrado
  digitalWrite(stcp, HIGH);           // Procedimiento según el datasheet
  digitalWrite(stcp, LOW);            //
```

```
}
int ds = 2; //data
int shcp = 3; // clock
int stcp = 4; //latch
int data = 0;
//importante. Variables de el shift register
boolean pinState[] = {0,0,0,0,0,0,0,0};  //Matriz que contiene los valores actuales
void setup() {
  pinMode(ds, OUTPUT);      //Establecer los pines a utilizar
  pinMode(shcp, OUTPUT);
  pinMode(stcp, OUTPUT);
  Serial.begin(9600); //Iniciar el monitor serie
  for(int x=0;x<8;x++)
  { digital_Write(x,LOW);     //Apagar todos los pines
  }}
void loop() {
  if(Serial.available()>0)
  { String y= Serial.readString();  //Leer un valor entre 0 y 7
    int x = y.toInt();  //Se transforma el dato recibido en un entero
    if(x>=0 && x<8)
    {
      pinState[x]=!pinState[x];     //Matriz con nuestros valores
      digital_Write(x,pinState[x]); //Analogo a digitalWrite
      Serial.print("Pin: ");
      Serial.print(x);              // Variable que contiene el pin
      Serial.print(", Estado: ");
      Serial.println((pinState[x]==1 ? "ENCENDIDO" : "APAGADO"));
    }  }}
void digital_Write(int myPin, boolean myState) //Funcion principal
//Analogo a digitalWrite(), para el shift register.
//myState = HIGH o LOW
{
  bitWrite(data,myPin,myState);  //Transformar nuestros datos
  shiftOut(ds, shcp, MSBFIRST, data);  //Enviar al integrado
  digitalWrite(stcp, HIGH);           // Procedimiento según el datasheet
  digitalWrite(stcp, LOW);            //
}
```

//22 PANTALLA LCD.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- PANTALLA LCD 2x16, compatible con el controlador HD44780
- Potenciómetro circular de 10KΩ
- 1 Resistencia de 220Ω (rojo, rojo, marrón)
- 15 cables M-M

CALCULA Y MIDE

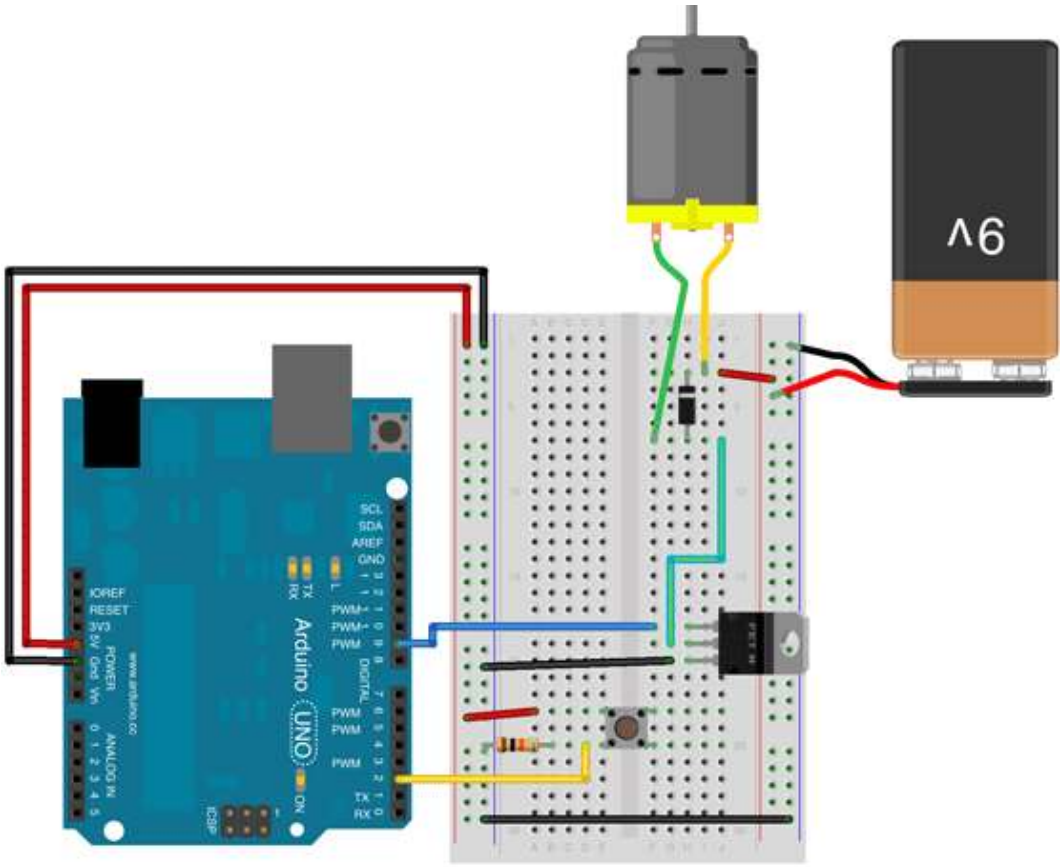
Pantalla LCD:
- El IDE de Arduino nos proporciona una librería llamada LiquidCrystal.h, mediante la que podemos manejar cualquier pantalla LCD que utilice como controlador el HD44780 de Hitachi.
- Aunque la pantalla LCD tiene un bus de datos de 8 bits, esta librería también nos permite comunicarnos mediante un bus de 4 bits, reduciendo mucho el número de terminales que usamos.

CÓDIGO

```
/*
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)
*/

// include the library code:
#include
// initialize the library by associating any needed LCD interface pin
// with the arduino pin number it is connected to
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // Turn off the display:
  lcd.noDisplay();
  delay(500);
  // Turn on the display:
  lcd.display();
  delay(500);
}
```

//23 CONTROL DE MOTORES CON UN TRANSISTOR MOSFET.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- MOTOR DE CC de 6 a 9v
- TRANSISTOR MOSFET IRL540
- 1 DIODO 1N4007
- 1 PULSADOR para placa (2 terminales)
- 1 Resistencia de 10K (marrón, negro, naranja)
- 11 cables M-M
- Portapilas tipo clip, para pilas de 9v
- 1 pila de 9v

CALCULA Y MIDE

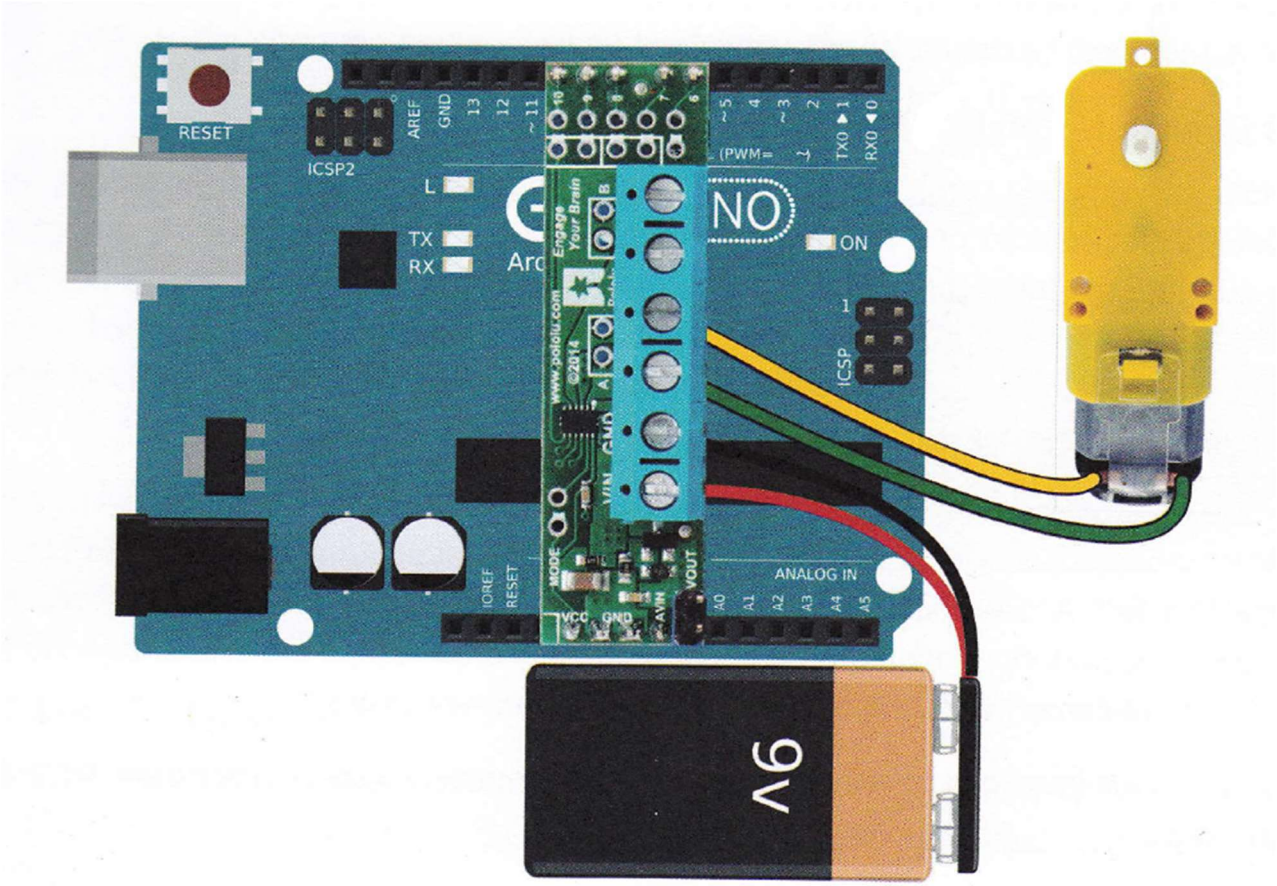
Transistor IRF540 vs IRL540:
- El transistor IRL540 (Logic-Level Gate Drive) está diseñado para funcionar con salidas lógicas de 5V y proporcionar mejores prestaciones y estabilidad que los de la serie IRF (estos se calientan más con la misma carga, por lo que resulta un problema cuando se manejan cargas de alto consumo, como por ejemplo motores).
- IRF540, para una V-puerta-surtidor de V, I=22A
- IRL540, para una V-puerta-surtidor de V, I=60A

CÓDIGO

```
// give a name to digital pin 2, which has a pushbutton attached
int pushButton = 2;
// the transistor which controls the motor will be attached to digital pin 9
int motorControl = 9;

// the setup routine runs once when you press reset:
void setup() {
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);

  // make the transistor's pin an output:
  pinMode(motorControl, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {

  // read the state of the button and check if it is pressed
  if(digitalRead(pushButton) == HIGH){
    // ramp up the motor speed
    for(int x = 0; x <= 255; x++){
      analogWrite(motorControl, x);
      delay(50);
    }

    // ramp down the motor speed
    for(int x = 255; x >= 0; x--){
      analogWrite(motorControl, x);
      delay(50);
    }
  }

  delay(1);        // delay in between reads for stability
}
```

//24 CONTROL DE MOTORES MEDIANTE UN PUENTE EN H.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- MÓDULO DRV8835 DE POLOLU (Ref.2511)
- MOTOR DE CC de 6 a 9v
- Portapilas tipo clip, para pilas de 9v
- 1 pila de 9v

CALCULA Y MIDE

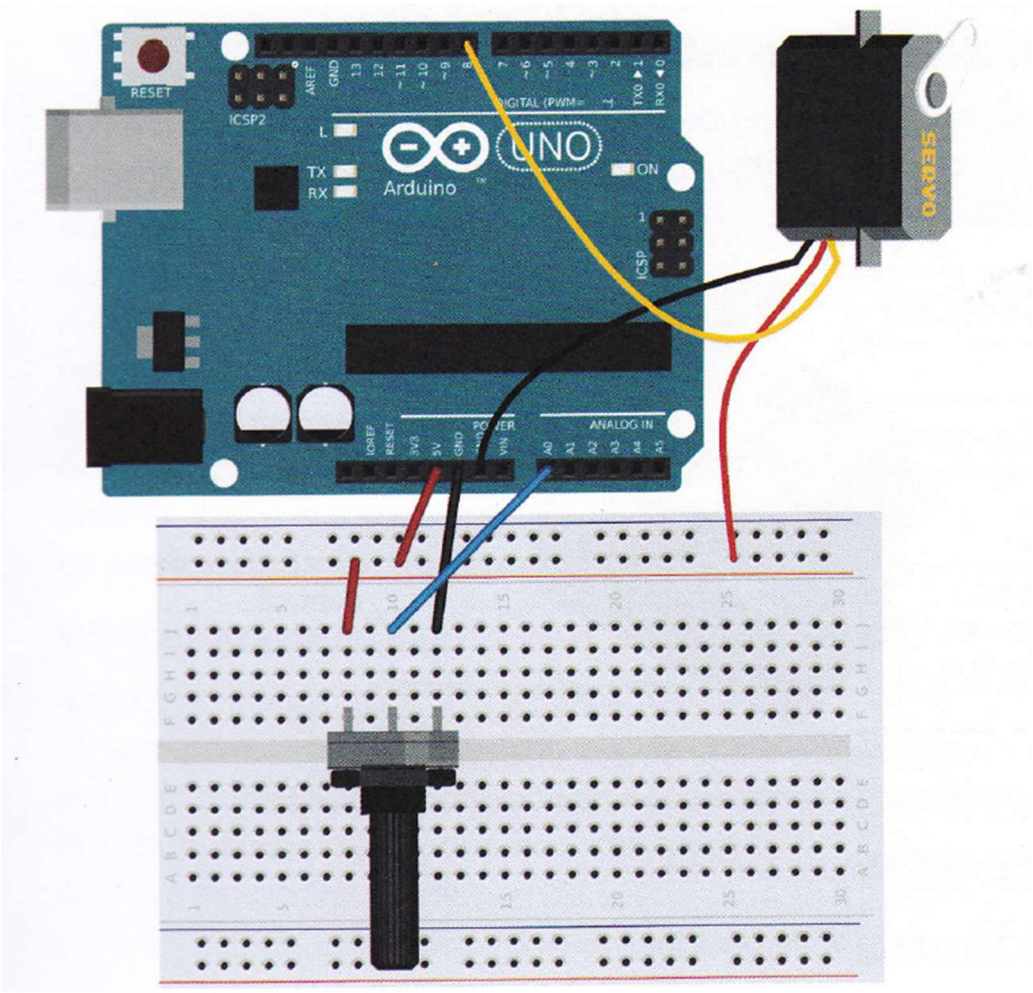
Puente en H:
- Con el transistor MOSFET no podemos invertir el sentido de giro del motor, y la forma de pararlo es poco eficaz, por esto usamos el controlador llamado Puente en H para superar dichas limitaciones. El que utilizaremos es un módulo que incorpora el DRV8835.
- El módulo DRV8835 permite controlar hasta dos motores DC. A diferencia de los shields de control de motores, este módulo es pequeño y ocupa tan solo 10 conexiones de la placa microcontroladora (Arduino).

CÓDIGO

```
#define PWM 9 //Pin de control PWM
#define DIR 7 //Pin de control de dirección
int i; //Variable auxiliar

void setup()
{
  pinMode(PWM,OUTPUT); //Configuramos pin PWM como salida
  pinMode(DIR,OUTPUT); //Configuramos pin DIR como salida
  }

void loop() {
  digitalWrite(DIR,LOW); //Motor en sentido horario
  analogWrite(PWM,0); //Motor parado

//Buble que incrementa la velocidad de giro del motor en una
//unidad cada 50 ms hasta llegar a su valor máximo (100%)
for(i=0; i<255; i++)
{
analogWrite(PWM,i); //Actualiza la velocidad
delay(50);
}

delay(2000); //Mantenemos el motor al 100% durante 2 segundos
digitalWrite(DIR,HIGH); //Motor en sentido antihorario
delay(2000); //Mantenemos el motor al -100% durante 2 segundos

//Buble que disminuye la velocidad de giro del motor en una
//unidad cada 50 ms hasta llegar a su valor mínimo (0%)
for(i=255; i>0; i--)
{
  analogWrite(PWM,i); //Actualiza la velocidad
  delay(50);
}
delay(2000); //Para el motor durante 2 segundos
}
```

//25 CONTROL DE SERVOS DE MODELISMO.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- SERVO DE MODELISMO
- Potenciómetro circular de 10K Ohmios
- 7 cables M-M

CALCULA Y MIDE

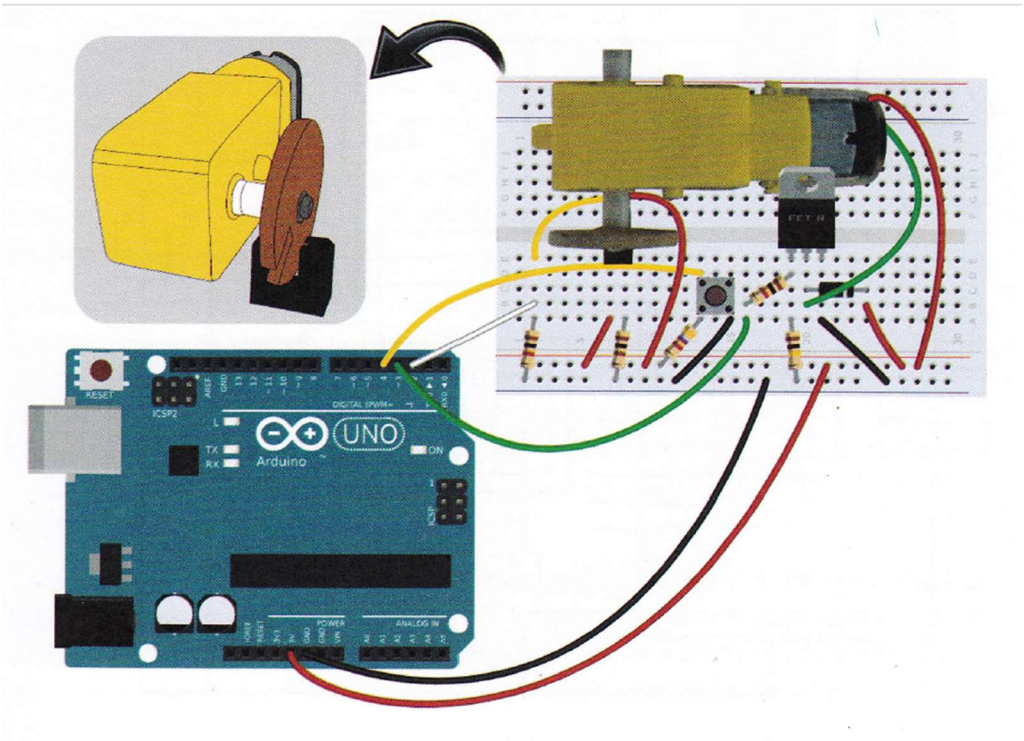
Servos de modelismo:
- Los servos de modelismo o de radio control (RC) son actuadores que permiten posicionar un eje de salida dentro de un determinado rango de posicionamiento angular (giro).
- El servo de RC es un sistema de control retroalimentado que dispone de motor de corriente continua con reductora, potenciómetro con sensor de posición y electrónica de control.
- La conexión entre un servo de RC y la placa microcontroladora (Arduino) se realiza con 3 cables: Rojo para alimentación positiva (4.5V-6.5V), naranja o blanco para la señal de control, y negro o marrón para alimentación negativa (GND).

CÓDIGO

```
// Controlamos la psición de un servo de RC en función
// de la posición del potenciómetro
#include  // Se incluye librería para controlar servos

// Pin de la placa para controlar el servo de RC
#define SERVO 8
// Pin de la placa para leer el potenciómetro
#define POT 0

// Creamos el objeto con el que se va a controlar el servo
Servo mi_servo;
// Variable donde se va a almacenar la posición del potenciómetro
int pos = 0;

void setup()
{
  // Inicialización del pin 8 como pin para controlar un servo
  mi_servo.attach(SERVO);
}

void loop()
{
// Lectura de la posición del potenciómetro
pos = analogRead(POT);

// Conversión de la lectura a grados para posicionar el servo
pos = map(pos, 0, 1023, 0, 180);

// Escritura en el pin del servo la posición
mi_servo.write(pos);

// Esperamos un poco el posicionamiento del servo de RC
delay(100);
}
```

//26 SENSORES ÓPTICOS REFLEXIVOS Y DE TRANSMISIÓN.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- MOTOR CON REDUCTORA Y DOBLE SALIDA DE EJE
- TRANSISTOR MOSFET IRL540
- 1 PULSADOR para placa (2 terminales)
- 1 Resistencia de 100 Ohmios (marrón, negro, marrón)
- 1 DIODO 1N4007
- 2 Resistencias de 1K (marrón, negro, rojo)
- 1 Resistencia de 4K7 (amarillo, morado, rojo)
- 1 Resistencia de 100K (marrón, negro, amarillo)
- SENSOR TCST1103
- 10 cables M-M
- Disco codificador
- Cinta adhesiva de doble cara

CALCULA Y MIDE

Sensor de transmisión TCST1103:
- Los sensores ópticos hacen uso de la forma en la que la luz interactúa con los objetos, por ejemplo, para detectarlos o conseguir alguna información sobre su color. Utilizaremos el sensor TCST1103.
- El lado del emisor está indicado por un pequeño chaflán. En ese lado podemos ver la letra “E” (cátodo del diodo emisor de infrarrojos) y el símbolo “+” (ánodo del diodo emisor de infrarrojos). En el lado del receptor, el contrario al del emisor, tenemos la letra “D” (emisor del fototransistor) y el símbolo “+” (colector del fototransistor).

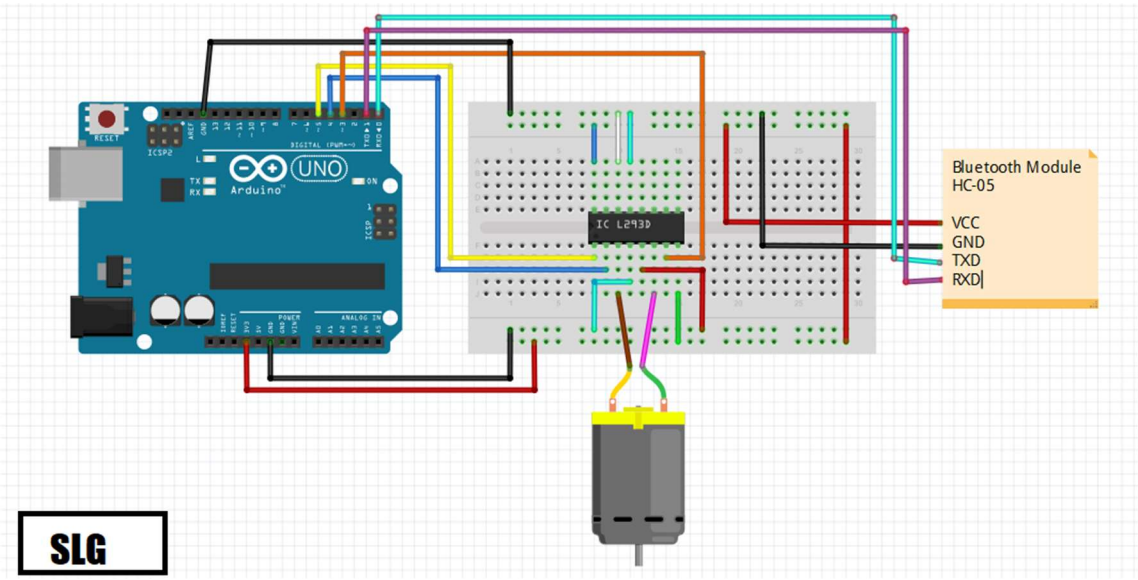
CÓDIGO

```
#define MOTOR 3 // Salida de control para el motor
#define RECEPTOR 2 // Receptor del sensor de transmisión
#define PULSADOR 4 // Pulsador de control
#define V_MOTOR 60 // Velocidad del motor
int estado = 0;

void setup()
{
  pinMode(MOTOR, OUTPUT);
  pinMode(RECEPTOR, INPUT);
}

void loop()
{
  if (digitalRead(PULSADOR) == HIGH)
  {
    estado = digitalRead(RECEPTOR);
  }
  else
  {
    estado = 0;
  }

  switch (estado)
  {
    case 0:
    analogWrite(MOTOR, V_MOTOR);
    break;
    case 1:
    default:
    analogWrite(MOTOR,0);
    break;
  }
}
```

// NIVEL 3: MÓDULOS DE EXPANSIÓN, ROBOTS.

//27 MOTOR DC CONTROLADO POR BLUETOOTH.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- Modulo Bluetooth HC-06 (Modulo o Shield)
- L293D (circuito integrado)
- 1 motor DC
- Dispositivo Móvil con Android
- Blueterm
- 20 cables M-M

CALCULA Y MIDE

Módulo Bluetooth HC-05 y HC-06:
- El Bluetooth es un estándar de comunicación inalámbrica que permite la transmisión de datos a través de radiofrecuencia en la banda de 2,4 GHz.
- El módulo Bluetooth HC-05 puede ser maestro/esclavo (master/slave);, el HC-06 sólo puede actuar como esclavo (slave). La diferencia entre maestro y esclavo es que en modo esclavo es el dispositivo quien se conecta al módulo, mientras que en modo maestro es el módulo quien se conecta con un dispositivo.
- Físicamente, los dos módulos son muy parecidos, solo varían algunas conexiones. Los pins que encontraremos son los siguientes:
-Vcc: Alimentación del módulo entre 3,6V y 6V.
-GND: La masa del módulo.
-TXD: Transmisión de datos.
-RXD: Recepción de datos a un voltaje de 3,3V.
-KEY: Poner a nivel alto para entrar en modo configuración del módulo (solo el modelo HC-05)
-STATE: Para conectar un led de salida para visualizar cuando se comuniquen datos.

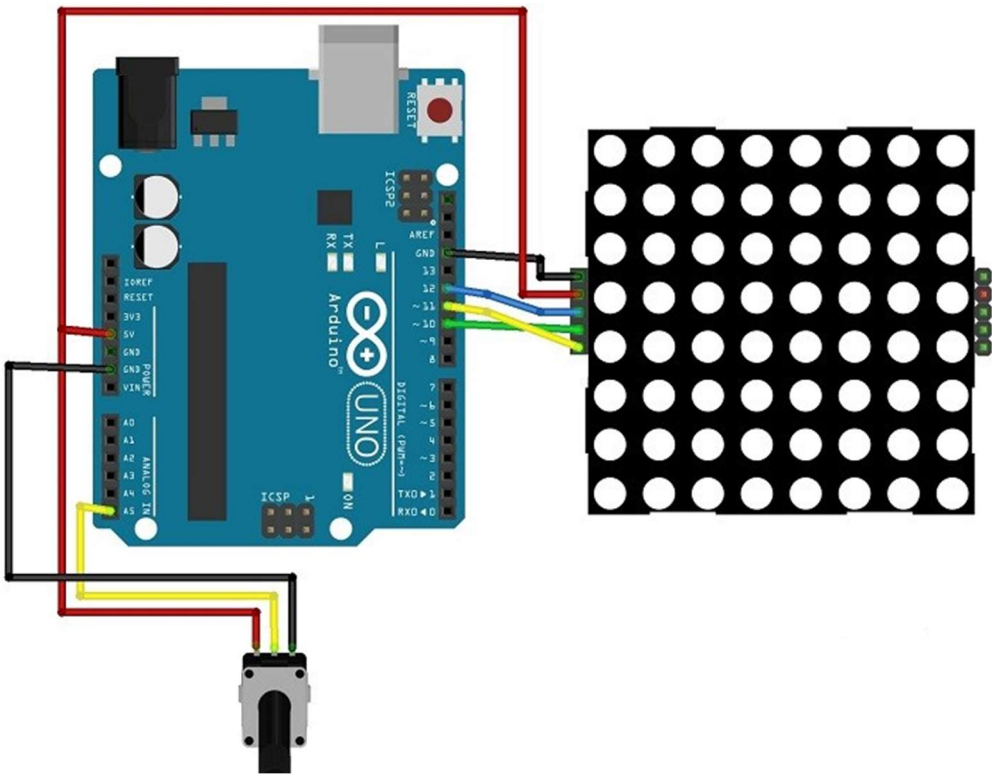
CÓDIGO

```
int motorPin1 = 3;           // Pin 2 to L293D IC
int motorPin2 = 4;           // Pin 7 to L293D IC
int enablePin = 5;           // Pin 1 to L293D IC
int state;
int flag=0;
void setup() {
// evaluate pins to be outputs
    pinMode(motorPin1, OUTPUT);
    pinMode(motorPin2, OUTPUT);
    pinMode(enablePin, OUTPUT);
// apply 5V on the enable Pin
    digitalWrite(enablePin, HIGH);
    Serial.begin(9600);
}
void loop() {
// send, read, and save data in "state"
    if(Serial.available() > 0){
      state = Serial.read();
      flag=0;
    }
// state = 0 "no rotation"
    if (state == '0') {
        digitalWrite(motorPin1, LOW);
        digitalWrite(motorPin2, LOW);
        if(flag == 0){
          Serial.println("Motor: off");
          flag=1;
        }
    }
// state = 1 "right dir. rotation
    else if (state == '1') {
        digitalWrite(motorPin1, LOW);
        digitalWrite(motorPin2, HIGH);
        if(flag == 0){
          Serial.println("Motor: right");
```

```
          flag=1;
        }
    }
// state = 2 "left dir. rotation
    else if (state == '2') {
        digitalWrite(motorPin1, HIGH);
        digitalWrite(motorPin2, LOW);
        if(flag == 0){
          Serial.println("Motor: left");
          flag=1;
        }
    }
}
```

//28 JUEGO DEL PONG.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- Matriz de LEDS de 8 x 8 con controlador MAX721
- Potenciómetro circular de 10K
- 8 cables M-M

CALCULA Y MIDE

Matriz de LEDs de 8 x 8 con controlador MAX721:
- Al igual que la pantalla de un ordenador o móvil, que están compuestas por píxeles, la matriz de LEDs 8×8 dispone de hasta 64 LEDs rojos con los que podemos dibujar casi cualquier cosa, de hecho tenemos un total de ¡18446744073709551616 combinaciones!
- El módulo de matriz LEDs se comunica a través del bus SPI (Serial Programming Interface) con Arduino, se trata de un protocolo de comunicación serie muy común entre dispositivo y controlador bidireccional que requiere de 4 líneas (una de habilitación o carga de datos, otra de reloj, una para los datos de entrada y otra para los datos de salida). No obstante, dado que la comunicación entre el Arduino y el módulo de matriz LEDs es unidireccional, sólo necesitaremos 3 líneas: la señal CS (Chip Select), a veces conocida por LOAD, permite habilitar el dispositivo para recibir información, la señal DIN es la entrada de datos (del módulo) y la señal CLK es la señal de reloj.
- Arduino dispone de un bus SPI, con lo que se suelen generar con instrucciones de escritura directamente en los pines de entrada o salida de Arduino.

CÓDIGO

```
#include "LedControl.h"
#include "Timer.h"
#define POTPIN A5 // Potentiometer
#define PADSIZE 3
#define BALL_DELAY 200
#define GAME_DELAY 10
#define BOUNCE_VERTICAL 1
#define BOUNCE_HORIZONTAL -1
#define NEW_GAME_ANIMATION_SPEED 50
#define HIT_NONE 0
#define HIT_CENTER 1
#define HIT_LEFT 2
#define HIT_RIGHT 3
//#define DEBUG 1
byte sad[] = {
B00000000,
B01000100,
B00010000,
B00010000,
B00000000,
B00111000,
B01000100,
B00000000
};
byte smile[] = {
B00000000,
B01000100,
B00010000,
B00010000,
B00010000,
B01000100,
B00111000,
B00000000
};
Timer timer;
```

```
LedControl lc = LedControl(12,11,10,1);
byte direction; // Wind rose, 0 is north
int xball;
int yball;
int yball_prev;
byte xpad;
int ball_timer;
void setSprite(byte *sprite){
    for(int r = 0; r < 8; r++){
        lc.setRow(0, r, sprite[r]);
    }
}
void newGame() {
    lc.clearDisplay(0);
    // initial position
    xball = random(1, 7);
    yball = 1;
    direction = random(3, 6); // Go south
    for(int r = 0; r < 8; r++){
        for(int c = 0; c < 8; c++){
            lc.setLed(0, r, c, HIGH);
            delay(NEW_GAME_ANIMATION_SPEED);
        }
    }
    setSprite(smile);
    delay(1500);
    lc.clearDisplay(0);
}
void setPad() {
    xpad = map(analogRead(POTPIN), 0, 1020, 8 - PADSIZE, 0);
}
void debug(const char* desc){
#ifdef DEBUG
    Serial.print(desc);
    Serial.print(" XY: ");
    Serial.print(xball);
    Serial.print(", ");
    Serial.print(yball);
```

```
    Serial.print(" XPAD: ");
    Serial.print(xpad);
    Serial.print(" DIR: ");
    Serial.println(direction);
#endif
}
int checkBounce() {
    if(!xball || !yball || xball == 7 || yball == 6){
        int bounce = (yball == 0 || yball == 6) ? BOUNCE_HORIZONTAL : BOUNCE_VERTICAL;
#ifdef DEBUG
        debug(bounce == BOUNCE_HORIZONTAL ? "HORIZONTAL" : "VERTICAL");
#endif
        return bounce;
    }
    return 0;
}
int getHit() {
    if(yball != 6 || xball < xpad || xball > xpad + PADSIZE){
        return HIT_NONE;
    }
    if(xball == xpad + PADSIZE / 2){
        return HIT_CENTER;
    }
    return xball < xpad + PADSIZE / 2 ? HIT_LEFT : HIT_RIGHT;
}
bool checkLoose() {
    return yball == 6 && getHit() == HIT_NONE;
}
void moveBall() {
    debug("MOVE");
    int bounce = checkBounce();
    if(bounce) {
        switch(direction){
            case 0:
                direction = 4;
            break;
            case 1:
                direction = (bounce == BOUNCE_VERTICAL) ? 7 : 3;
```

```
            break;
            case 2:
                direction = 6;
            break;
            case 6:
                direction = 2;
            break;
            case 7:
                direction = (bounce == BOUNCE_VERTICAL) ? 1 : 5;
            break;
            case 5:
                direction = (bounce == BOUNCE_VERTICAL) ? 3 : 7;
            break;
            case 3:
                direction = (bounce == BOUNCE_VERTICAL) ? 5 : 1;
            break;
            case 4:
                direction = 0;
            break;
        }
        debug("->");
    }
    // Check hit: modify direction is left or right
    switch(getHit()){
        case HIT_LEFT:
            if(direction == 0){
                direction =  7;
            } else if (direction == 1){
                direction = 0;
            }
        break;
        case HIT_RIGHT:
            if(direction == 0){
                direction = 1;
            } else if(direction == 7){
                direction = 0;
            }
        break;
```

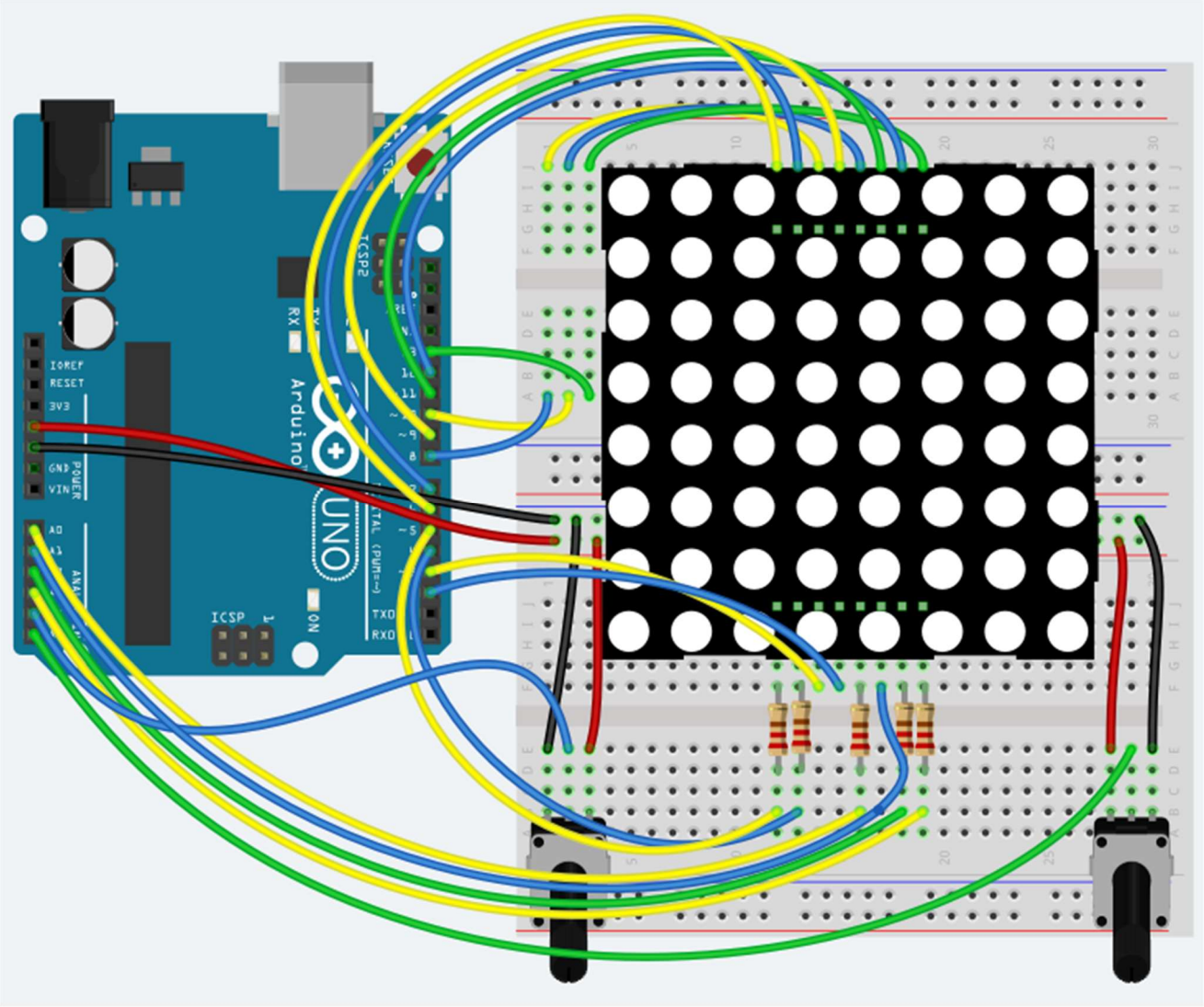
```
}

// Check orthogonal directions and borders ...
if((direction == 0 && xball == 0) || (direction == 4 && xball == 7)){
    direction++;
}
if(direction == 0 && xball == 7){
    direction = 7;
}
if(direction == 4 && xball == 0){
    direction = 3;
}
if(direction == 2 && yball == 0){
    direction = 3;
}
if(direction == 2 && yball == 6){
    direction = 1;
}
if(direction == 6 && yball == 0){
    direction = 5;
}
if(direction == 6 && yball == 6){
    direction = 7;
}
// "Corner" case
if(xball == 0 && yball == 0){
    direction = 3;
}
if(xball == 0 && yball == 6){
    direction = 1;
}
if(xball == 7 && yball == 6){
    direction = 7;
}
if(xball == 7 && yball == 0){
    direction = 5;
}
```

```
    yball_prev = yball;
    if(2 < direction && direction < 6) {
        yball++;
    } else if(direction != 6 && direction != 2) {
        yball--;
    }
    if(0 < direction && direction < 4) {
        xball++;
    } else if(direction != 0 && direction != 4) {
        xball--;
    }
    xball = max(0, min(7, xball));
    yball = max(0, min(6, yball));
    debug("AFTER MOVE");
}
void gameOver() {
    setSprite(sad);
    delay(1500);
    lc.clearDisplay(0);
}
void drawGame() {
    if(yball_prev != yball){
        lc.setRow(0, yball_prev, 0);
    }
    lc.setRow(0, yball, byte(1 << (xball)));
    byte padmap = byte(0xFF >> (8 - PADSIZE) << xpad) ;
#ifdef DEBUG
    //Serial.println(padmap, BIN);
#endif
    lc.setRow(0, 7, padmap);
}
void setup() {
  // The MAX72XX is in power-saving mode on startup,
  // we have to do a wakeup call
  pinMode(POTPIN, INPUT);
  lc.shutdown(0,false);
  // Set the brightness to a medium values
  lc.setIntensity(0, 8);
```

```
  // and clear the display
  lc.clearDisplay(0);
  randomSeed(analogRead(0));
#ifdef DEBUG
  Serial.begin(9600);
  Serial.println("Pong");
#endif
  newGame();
  ball_timer = timer.every(BALL_DELAY, moveBall);
}
void loop() {
    timer.update();
    // Move pad
    setPad();
#ifdef DEBUG
    Serial.println(xpad);
#endif
    // Update screen
    drawGame();
    if(checkLoose()) {
        debug("LOOSE");
        gameOver();
        newGame();
    }
    delay(GAME_DELAY);
}
```

//28B JUEGO DEL PING PONG.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Placa microcontroladora
- Placa de prototipos
- Matriz de LEDS de 8 x 8 sin controlador MAX721
- 2 Potenciómetros circulares de 10K
- 5 Resistencias de 220Ω (rojo, rojo, marrón)
- 27 cables M-M

CALCULA Y MIDE

- En esta variante se utilizan dos potenciómetros como controladores.
- Nos saltaremos el controlador MAX7219 y nos conectaremos directamente a la matriz, para ello tendremos que soldar los pines, como hiciéramos con la pantalla LCD para montar el Reloj digital.

CÓDIGO

```
int rowPins[8] = {9,3,2,12,15,11,7,6};          // matrix rows connected to the Arduino
int colPins[8] = {13,8,17,10,5,16,4,14};        // matrix columns connected to the Arduino


int pot1Pin=18;                                 // declaring the pin for player 1's potentiomete
r
int pot2Pin=19;                                 // declaring the pin for player 2's potentiomete
r


int image[8][8]={                               // clear
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0}
};

int death[8][8]={                               // all on
{1,1,1,1,1,1,1,1},
{1,1,1,1,1,1,1,1},
{1,1,1,1,1,1,1,1},
{1,1,1,1,1,1,1,1},
{1,1,1,1,1,1,1,1},
{1,1,1,1,1,1,1,1},
{1,1,1,1,1,1,1,1},
{1,1,1,1,1,1,1,1}
};

int death2[8][8]={                              // skull
{0,1,1,1,1,1,0,0},
{1,1,1,1,1,1,1,0},
{1,0,0,1,0,0,1,0},
{1,1,1,1,1,1,1,0},
{0,1,1,1,1,1,0,0},
```

```
{0,1,0,1,0,1,0,0},
{0,1,0,1,0,1,0,0},
{0,0,0,0,0,0,0,0}
};

int blank[8][8]={                              // all off
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0}
};

long theTimer;                                          // timer variable

int gameState=0;                                        // storing the game status
int animations=300;                                     // storing the duration of the images
between the games

float ballXSpeed = 1;                                   // storing the x acceleration
float ballXPosition=4;                                  // storing the ball position as number

float ballYSpeed = 0;                                   // storing the y acceleration
float ballYPosition=4;                                  // storing the ball position as number

int imageYPosition;                                     // storing the image position as numbe
r
int imageXPosition;                                     // storing the image position as numbe
r

int player1Position=3;                                  // storing the paddle positoin of play
er 1
int player2Position=3;                                  // storing the paddle positoin of play
er 2

int gameSpeed;                                          // storing the current game speed
```

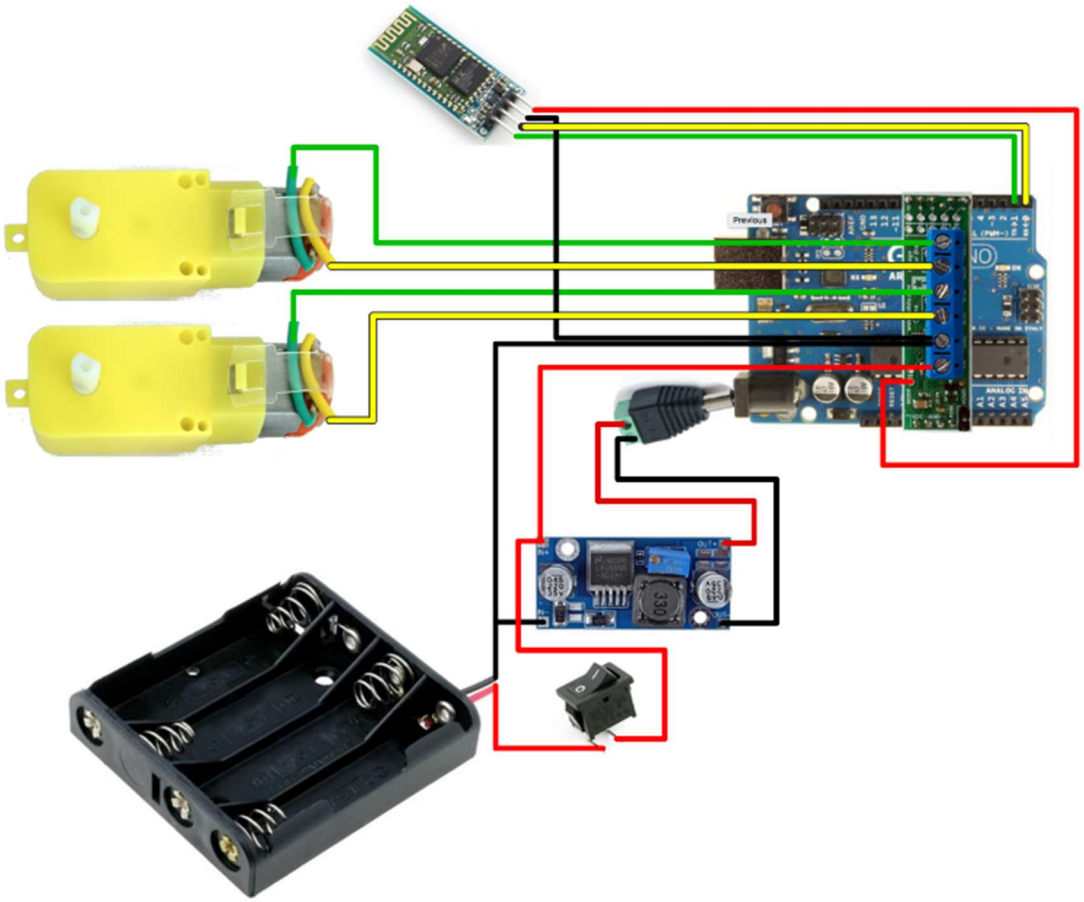
```
void setup(){
  for (int i=0; i<8; i++){                              // all pins are outputs
    pinMode(rowPins[i],OUTPUT);
    pinMode(colPins[i],OUTPUT);
  }
}


void draw(){
  for (int y=0; y<8; y++){                              // rowwise
    for (int x=0; x<8; x++){                            // from left to right, entries are che
cked
      if (image[x][y]==1){                              // if entry equals 1
        digitalWrite(colPins[x],HIGH);                  // the column pin is switched on
      } else {                                          // else
        digitalWrite(colPins[x],LOW);                   // the column pin is switched off
      }
    }
    digitalWrite(rowPins[y],LOW);                       // switch the row pin to LOW (because
it is the cathod of the LED LOW menas ON)
    delayMicroseconds(1000);                            // stop the program for 1 seconds
    digitalWrite(rowPins[y],HIGH);                      // switch the row pin to HIGH (what me
ans OFF)
  }
}


void update(){
  switch (gameState) {                                  // switching game mode (called state m
achine)
  case 0:                                               // new game
    memcpy(image,blank,sizeof(blank));                  // clear screen
    gameSpeed=300;                                      // set the game speed
    ballXPosition=3;                                    // set ball position
    ballYPosition=3;                                    // set ball position
    ballYSpeed=0;                                       // ball should fly straight
    if (random(0,2)>0){                                 // but randomly left or right
      ballXSpeed=1;
    } else {
      ballXSpeed=-1;
    }
```

```
    theTimer=millis();
    gameState=1;
    break;
  case 1:                                          // game active
    image[player1Position][0]=0;                   // paddle player 1 clear old position
    image[player1Position+1][0]=0;                 // paddle player 1 clear old position
    image[player2Position][7]=0;                   // paddle player 2 clear old position
    image[player2Position+1][7]=0;                 // paddle player 2 clear old position

    player1Position=map(analogRead(pot1Pin),0,1023,0,6); // reading the position of player 1
    player2Position=map(analogRead(pot2Pin),0,1023,0,6); // reading the position of player 2

    image[player1Position][0]=1;                   // paddle player 1 display
    image[player1Position+1][0]=1;                 // paddle player 1 display
    image[player2Position][7]=1;                   // paddle player 2 display
    image[player2Position+1][7]=1;                 // paddle player 2 display

    if (millis()>theTimer+gameSpeed){              // timer for game speed
      if (gameSpeed>50) gameSpeed-=3;              // accelerate game
      theTimer=millis();                           // set new timer
      image[imageYPosition][imageXPosition]=0;     // overwrite old position
      ballXPosition+=ballXSpeed;                   // update position
      ballYPosition+=ballYSpeed;                   // update position

      if (ballYPosition>=7) ballYSpeed*=-1;        // collision bottom border
      if (ballYPosition<=0) ballYSpeed*=-1;        // collision top border

      ballYPosition=constrain(ballYPosition,0,7);  // constrain values between 0 and 7
      ballXPosition=constrain(ballXPosition,0,7);  // constrain values between 0 and 7
      imageYPosition=round(ballYPosition);
      imageXPosition=round(ballXPosition);

      if ((ballXPosition>=6)&&(image[imageYPosition][7]==1)) {  // if ball hits a paddle
        ballXSpeed*=-1;                            // reflect the ball
        ballYSpeed=random(-2,3);                   // random reflection angle
      }
      if ((ballXPosition<=1)&&(image[imageYPosition][0]==1)) {  // if ball hits a paddle
        ballXSpeed*=-1;                            // reflect the ball
```

```
        ballYSpeed=random(-2,3);                    // random reflection angle
      }
      if (ballXPosition>=7){                        // ball out
        gameState=2;                                // change status to lost game
        theTimer=millis();                          // new timer is set
      }

      if (ballXPosition<=0){                        // ball out
        gameState=2;                                // change status to lost game
        theTimer=millis();                          // new timer is set
      }

      image[imageYPosition][imageXPosition]=1;      // set new image position

    }
    break;
  case 2:                                           // game was lost
    if (millis()>theTimer+gameSpeed){               // wait for a short time
      theTimer=millis();
      gameState=3;                                  // game state to lost game display
    }
    break;
  case 3:                                           // lost game display
    memcpy(image,death,sizeof(death));              // show image of lost game, memcpy i
s a function copying one array to another
    if (millis()>theTimer+animations){              // wait again
      gameState=0;                                  // game state change to »start a gam
e«
    }
    break;
  }
}


void loop(){
  update();                                         // all calculations are in the updat
e method
  draw();                                           // all display methods are in here
}
```

//29 FÓRMULA 1 TELEDIRIGIDO, CONTROLADO REMOTAMENTE CON EL TELÉFONO MÓVIL.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Una placa controladora
- Chasis
- Dos motores de corriente continua de doble eje
- Un driver de motores de doble canal DRV8835
- Un interruptor
- Dos sensores de contacto (finales de carrera), que funcionan como bumpers
- Un sensor de ultrasonidos HC-SR04
- Un módulo Bluetooth HC-05
- Un portapilas 5 x AA
- Un conector Jack de alimentación 5,5 x 2,1
- Dos ruedas para motor con eje en D
- Una rueda loca
- 7 cables macho-macho
- 20 cables hembra-hembra
- Dos gomas elásticas
- Tornillería

CALCULA Y MIDE

Conexiones con la placa Arduino:

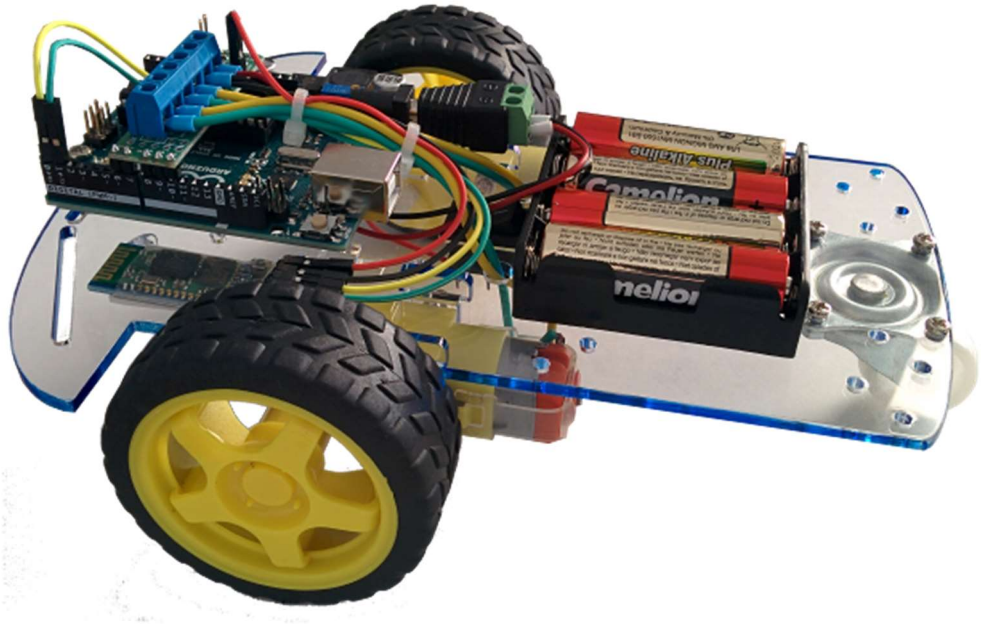
Pin	Dispositivo --> Color del cable
D0	Transmisión de datos Bluetooth (*) --> Blanco
D1	Recepción de datos Bluetooth (*) --> Gris
D2	Bumper derecho --> Verde
D3	Bumper izquierdo --> Azul
D7	DIR M1 --> Conexión directa al driver de motores
D8	DIR M2 --> Conexión directa al driver de motores
D9	PWM Motor 1 (derecha) --> Conexión directa al driver de motores
D10	PWM Motor 2 (Izquierda) --> Conexión directa al driver de motores
D12	Echo del sensor de distancia --> Blanco
D13	Trig del sensor de distancia --> Amarillo.

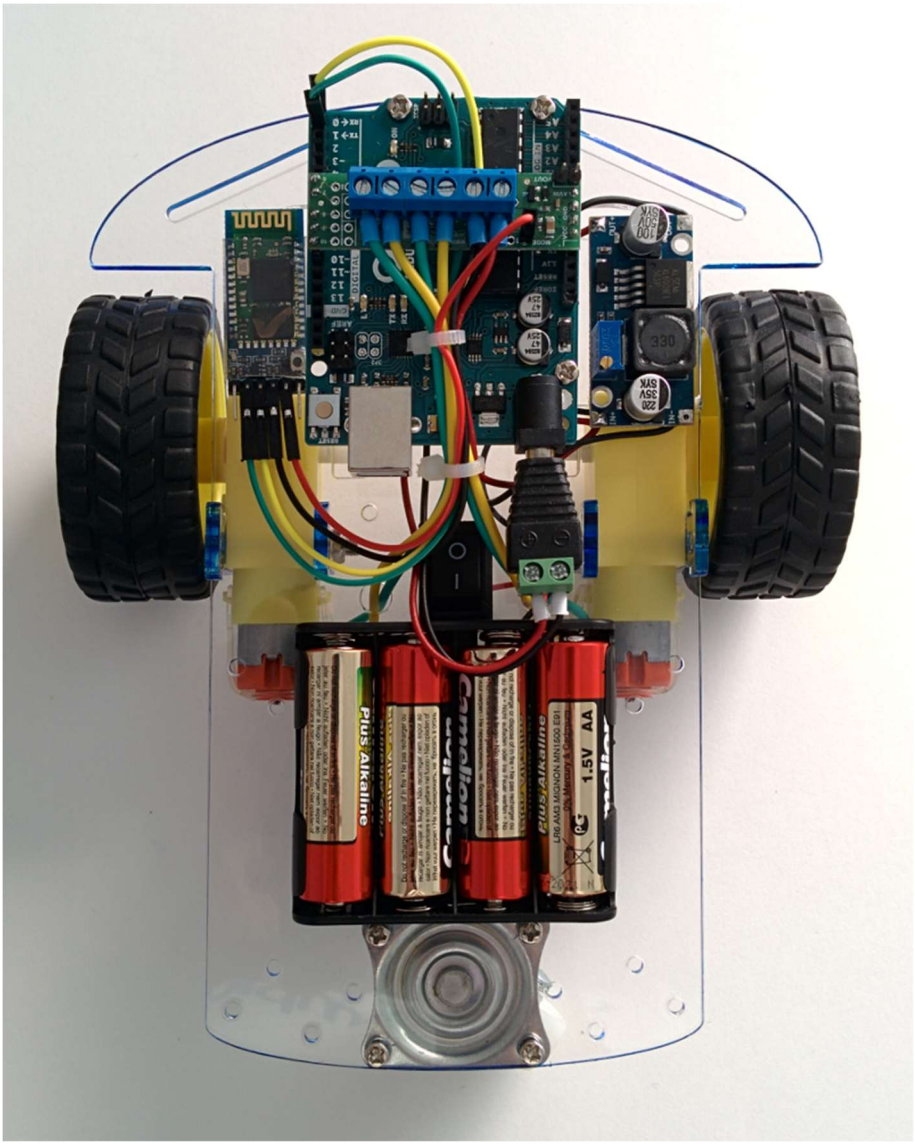
CÓDIGO

```
/*
  Este ejemplo permite controlar el robot de forma remota con un
  smartphone o una tablet con Android.

  En función del movimiento que se seleccione en la aplicación,
  se entra en un caso de la máquina de estados, lo que lleva a diferentes movimientos del robot
.
*/

#define M1_DIR 7             //Pin de Arduino que controla la dirección del motor (DIGITAL)
#define M1_PWM 9             //Pin de Arduino que controla la velocidad (PWM)
#define M2_DIR 8             //Pin de Arduino que controla la dirección del motor (DIGITAL)
#define M2_PWM 10            //Pin de Arduino que controla la velocidad (PWM)

char estado = 'c';            // Estado con el que se inicia el robot: parado
char anterior;                // Variable para coservar el estado anterior de la máquina de est
ados
char memo;                    // Variable para memorizar el último estado diferente

void setup()
{
  //Declaración como salida de todos los pines que se
  //necesitan para controlar los motores del robot
  pinMode(M1_DIR, OUTPUT);
  pinMode(M1_PWM, OUTPUT);
  pinMode(M2_DIR, OUTPUT);
```

```
  pinMode(M2_PWM, OUTPUT);

  Serial.begin(9600);        // Inicia el puerto serial para comunicación con el Bluetooth
}

void loop()
{
  if (Serial.available() > 0) // lee el bluetooth y almacena el carácter en "estado"
  {
    // Almacenamiento del último estado utilizado
    // Adquisición del nuevo estado
    // Memorización del último estado diferente utilizado
    anterior = estado;
    estado = Serial.read();
    if (estado != anterior)
    {
      memo = anterior;
      }
  }

  switch (estado)
  {
    case 'a':    // Botón desplazar al Frente
      // Los dos motores avanzan hacia delante
      digitalWrite(M1_DIR, LOW);
      analogWrite(M1_PWM, 255);
      digitalWrite(M2_DIR, HIGH);
      analogWrite(M2_PWM, 255);
      break;

    case 'b':    // Botón IZQ
      // Realiza giro
      digitalWrite(M1_DIR, LOW);  //Motor uno avanza
      analogWrite(M1_PWM, 255);
      digitalWrite(M2_DIR, LOW);  //Motor dos retrocede
      analogWrite(M2_PWM, 255);
      delay (100);
      estado = memo;
```

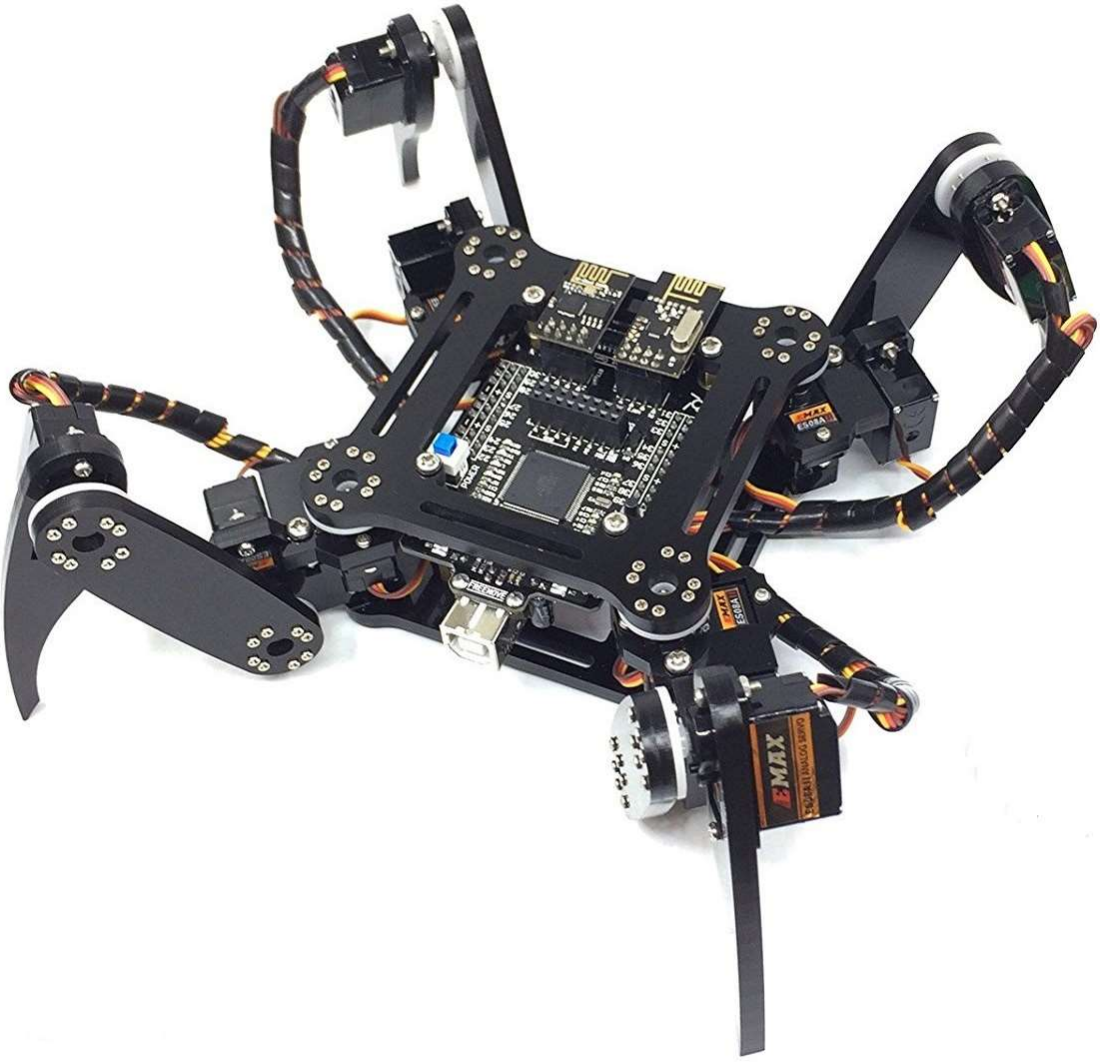
```
      break;

    case 'c':    // Botón Parar
      // Los dos motores se detienen
      digitalWrite(M1_DIR, LOW);
      analogWrite(M1_PWM, 0);
      digitalWrite(M2_DIR, LOW);
      analogWrite(M2_PWM, 0);
      break;

    case 'd':    // Botón DCHA
      // Realiza giro
      digitalWrite(M1_DIR, HIGH); //Motor uno retrocede
      analogWrite(M1_PWM, 255);
      digitalWrite(M2_DIR, HIGH); //Motor dos avanza
      analogWrite(M2_PWM, 255);
      delay (100);
      estado = memo;
      break;

    case 'e':    // Botón Reversa
      //Los dos motores retroceden
      digitalWrite(M1_DIR, HIGH);
      analogWrite(M1_PWM, 255);
      digitalWrite(M2_DIR, LOW);
      analogWrite(M2_PWM, 255);
      break;

    case 'g':   //A la espera, no hace nada
      break;
  }
}
```




Plus Alkaline
1.5V AA

//30 EL ROBOT CUADRÚPEDO CONTROLADO POR WIFI.

MONTAJE



MATERIALES

- Un cable USB tipo A-B
- Una placa controladora
- Piezas mecánicas y acrílicas
- Gráfico de calibración
- Controlador
- Módulo WiFi ESP8266
- 12 x EMAX ES08AII Servo
- Cable Tidy 50cm
- Soporte de batería de 9V
- Módulo 2 x NRF24L01

CALCULA Y MIDE

Chip ESP8266:
- El ESP8266 es un chip de bajo costo Wi-Fi con una pila TCP/IP completa y un microcontrolador.
- Este pequeño módulo permite a otros microcontroladores conectarse a una red inalámbrica Wi-Fi y realizar conexiones simples con TCP/IP usando comandos al estilo Hayes.
- El ESP8285 es como un ESP8266 pero con 1 MB de memoria flash interna, para permitir a dispositivos de un chip conexiones de Wi-Fi.

CÓDIGO

Custom Programming

You can also use the library (FNQR) we provide to program the robot.

First, include FNQR library in Arduino sketch.

```
1  #include <FNQR.h>
```

Then define a robot object.

```
2  FNQR robot;
```

In function setup(), start the robot.

```
3  void setup() {
4    robot.Start();
5  }
```

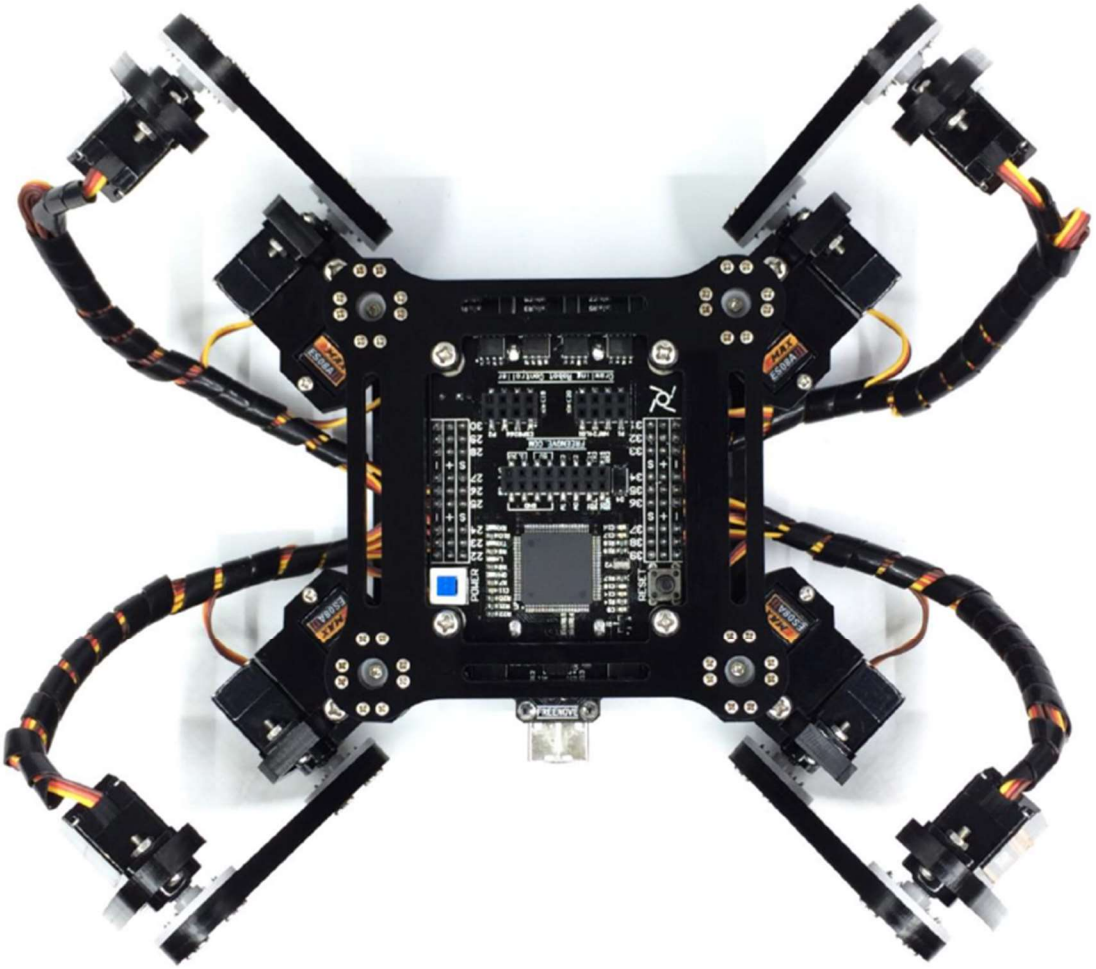
Now, you can directly use the following code in function loop() to control the robot.

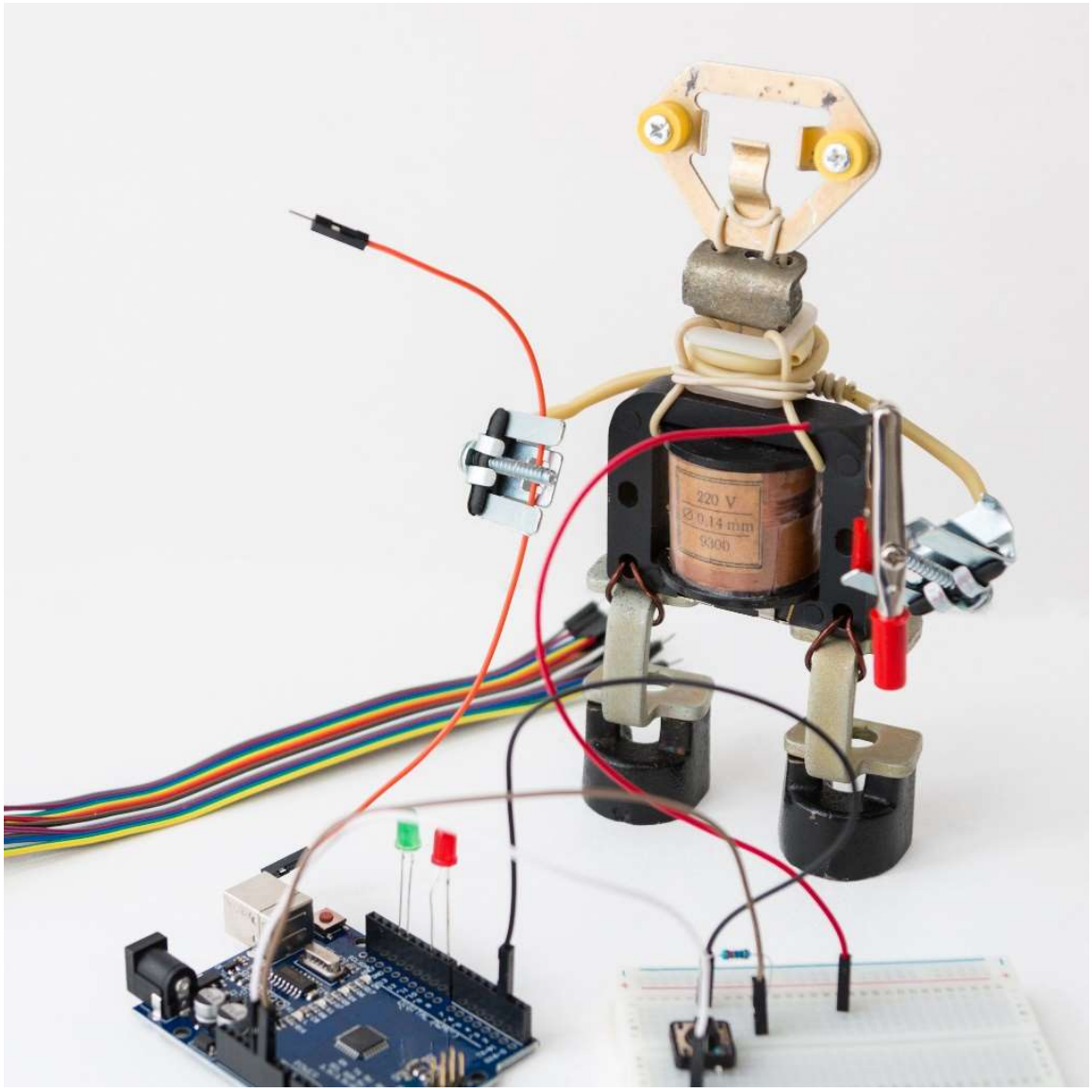
```
6    robot.CrawlForward();
7    robot.CrawlBackward();
8    robot.TurnLeft();
9    robot.TurnRight();
10   robot.SleepMode();
11   robot.ActiveMode();
12   robot.SwitchMode();
13   robot.MoveBody(float x, float y, float z);
14   robot.RotateBody(float x, float y, float z, float angle);
```

There are some examples in"File" > "Examples" > "FNQR" > "Examples ". You can open and upload them to learn how to use FNQR library to control the robot.

You can also open the FNQR.h file of FNQR library to view the specific usage, this file has detailed comments.

If you upload the code written by yourself, the robot will be controlled only by your code, and you will not be able to use the computer, Android device and remote control to control it. You can reupload the default sketch to restore.





FIN

"Si he visto más lejos, ha sido por subirme a hombros de gigantes"

(Isaac Newton)