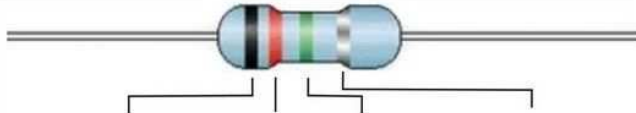


PRÁCTICA 0. Introducción al Arduino UNO

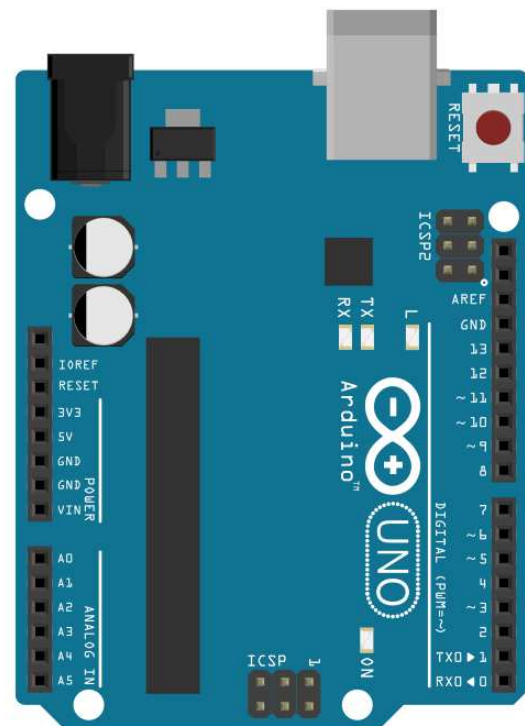
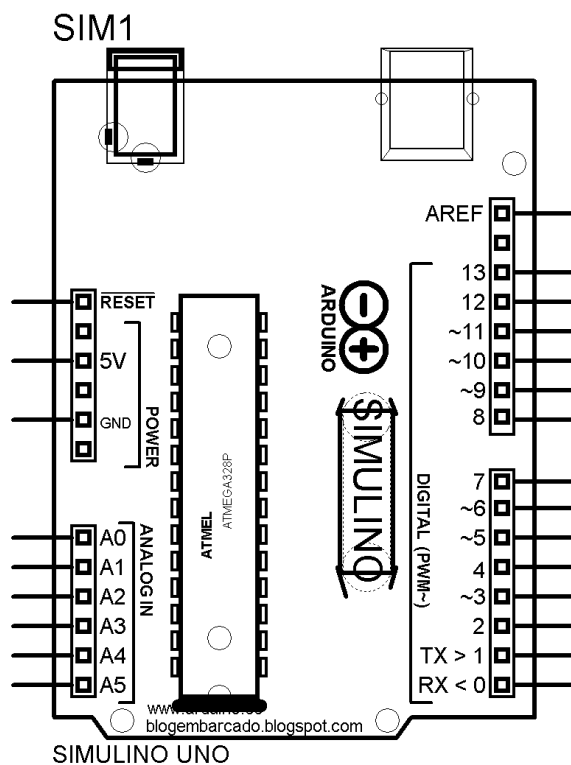
Explicación. Conocer las partes de la placa Arduino UNO. Conocer la placa protoboard. Conocer el código de colores de una resistencia. Conocer cómo funciona un LED. Instalar Arduino UNO.



Color	1ra. Banda	2da. Banda	3ra. Banda Multiplicador	Tolerancia %
Negro	0	0	x1	
Cafe	1	1	x10	
Rojo	2	2	x100	2%
Naranja	3	3	x1000	
Amarillo	4	4	x10000	
Verde	5	5	x100000	
Azul	6	6	x1000000	
Violeta	7	7	x10000000	
Gris	8	8	x100000000	
Blanco	9	9	x1000000000	
				Dorado 5%
				Plata 10%

Circuitos Básicos

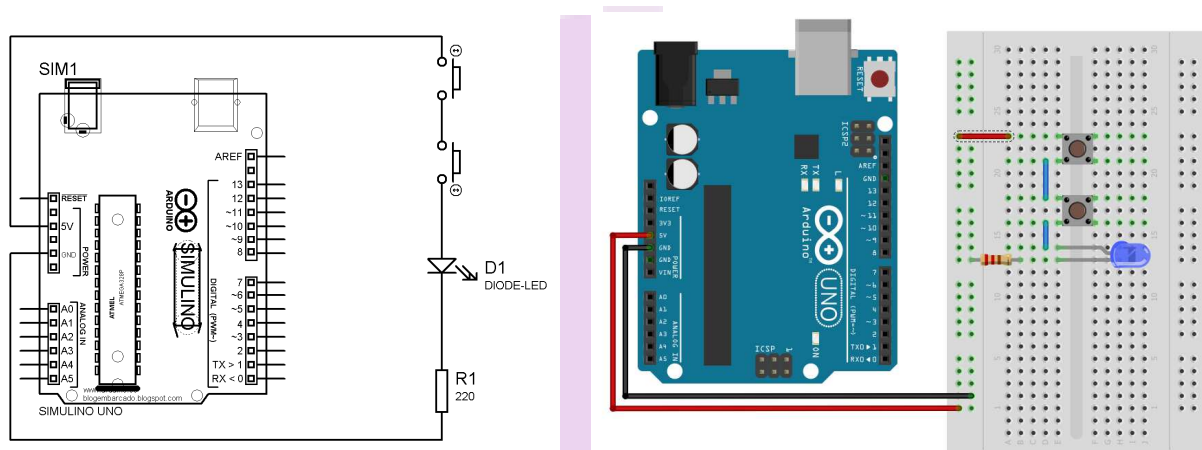
Por www.areatecnologia.com



PRÁCTICA 1. Usar Arduino como una pila de 5 V

Materiales. 2 pulsadores, 1 led y 1 resistencia de 220 Ω (rojo-rojo-marrón).

Explicación. Debemos conseguir que el led se encienda mientras estemos accionando los dos pulsadores a la vez. Para ello utilizaremos Arduino simplemente como una pila.

Esquemas eléctricos

Ampliación. ¿Cómo harías que el led se encienda mientras al menos uno de los dos pulsadores esté accionado?

OIKOS

PRÁCTICA 2. Encender un LED de forma intermitente

Materiales. 1 led y 1 resistencia de 220 Ω (rojo-rojo-marrón).

Explicación. Debemos conseguir que un led se encienda de forma intermitente de la siguiente manera: en cada ciclo, el encendido durará medio segundo y el apagado un segundo. Para ello, conectaremos el ánodo del led al pin 13. El cátodo del led irá a una resistencia de 220 Ω para proteger el led. La resistencia irá a tierra. Hecho esto, ya sabemos que cuando el pin 13, que es digital, esté en alto (5 V), el led se encenderá y cuando el pin 13 esté en bajo (0 V), el led se apagará.

Esquema de un sketch. Dividimos cada sketch en tres partes:

1. Declaración de variables y constantes globales.

Por ejemplo, si escribimos `const int led=13;` estamos declarando una constante entera llamada `led` de valor 13. Por ser una constante no podremos modificar su valor durante el sketch.

Por ejemplo, si escribimos `int pepito=12;` estamos declarando una variable entera llamada `pepito` a la que inicialmente le hemos asignado el valor de 12. No es obligatorio asignar un valor inicial a una variable.

2. `void setup() {...}`

Esta parte del sketch va entre llaves y se ejecuta una única vez. Aquí es donde decidiremos qué pines de Arduino vamos a utilizar y si van a ser de entrada o salida. También podemos inicializar los que creamos conveniente.

Los pines digitales pueden ser de entrada o de salida y son: 0, 1, 2, ..., 12 y 13. Los pines digitales solo pueden en alto o HIGH (5 V) y en bajo o LOW (0 V). Los pines analógicos siempre son de entrada y son: A0, A1, ..., A4 y A5. Los pines analógicos pueden tomar cualquier valor entre 0 V y 5 V.

Si escribimos `pinMode(12,OUTPUT);` estamos declarando el pin digital 12 como salida. Por tanto, Arduino va a poder poner este pin en HIGH o en LOW según se lo indiquemos en el sketch.

Si escribimos `digitalWrite(12,HIGH);` Arduino pondrá el pin de salida 12 en alto.

Si escribimos `digitalWrite(12,LOW);` Arduino pondrá el pin de salida 12 en bajo.

Si escribimos `pinMode(9,INPUT);` estamos declarando el pin digital 9 como entrada. Por tanto, Arduino va a poder leer si este pin está en alto o en bajo cuando se lo indiquemos. De momento, no vamos a necesitar leer pines; ni digitales ni analógicos.

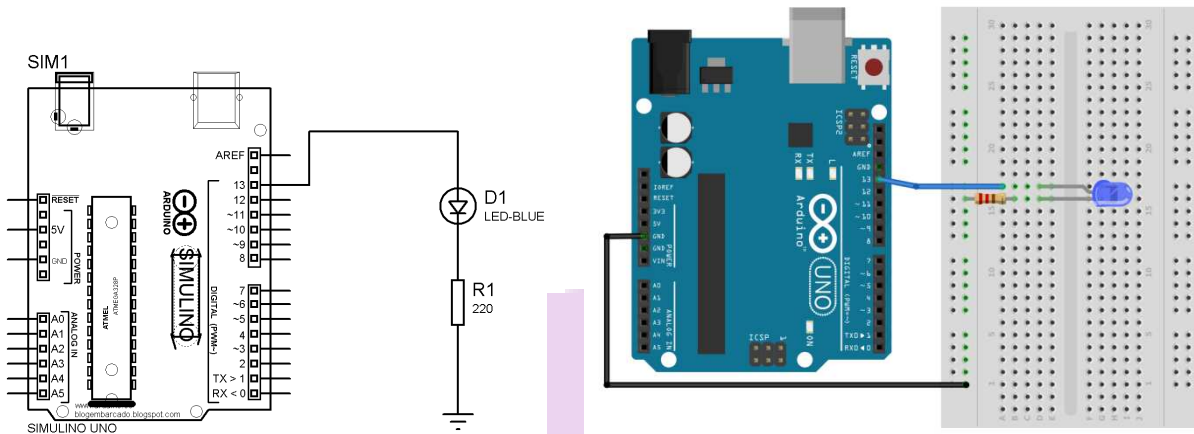
3. `void loop() {...}`

Esta parte del sketch va entre llaves y se ejecuta constantemente; esto es, después de ejecutarse la última línea se volverá a ejecutar la primera y se repetirá el bucle.

Si escribimos `delay(500);` la ejecución del programa se detendrá durante 500 ms; esto es, medio segundo. Así, la función `delay` detiene la ejecución del programa el tiempo que le indiquemos entre paréntesis en milisegundos.

Si en una línea escribimos `//`, entonces a continuación, en esa línea, podemos poner un comentario.

Esquemas eléctricos



Sketch

```
const int led=13; //conectaré el ánodo del LED al pin 13

void setup() {
  pinMode(led,OUTPUT); //declaro el pin al que conectaré el LED como de salida
}

void loop() {
  digitalWrite(led,HIGH); //pongo el pin al que conectaré en LED en alto ...
  delay(500);             //durante medio segundo
  digitalWrite(led,LOW);  //pongo el pin al que conectaré el LED en bajo...
  delay(1000);            //durante un segundo
}
```

Ampliación. ¿Cómo harías para que el encendido fuese de 1 s y el apagado de 0,3 s?

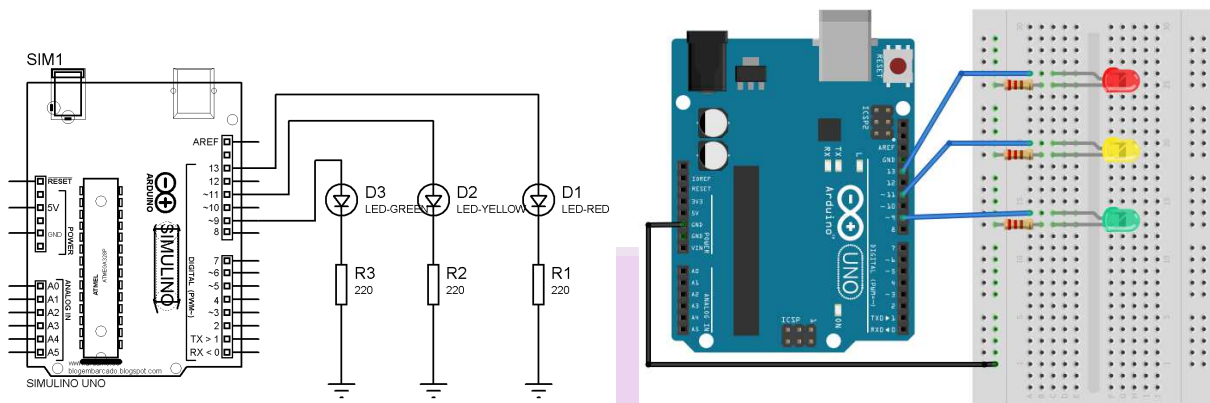
PRÁCTICA 3. El semáforo

Materiales. 1 led rojo, 1 led amarillo, 1 led verde y 3 resistencias de $220\ \Omega$ (rojo-rojo-marrón).

Explicación. Debemos conseguir que los tres leds se enciendan en bucle alternativamente, como si de un semáforo se tratase, según esta secuencia: rojo (2 s), verde (2 s), amarillo (0,5 s). Para ello, conectaremos el ánodo del led rojo al pin 13, el ánodo del led amarillo al pin 11 y el ánodo del led verde al pin 9.



Esquemas eléctricos



Sketch

```
const int rojo=13;           //el ánodo del led rojo irá al pin 13
const int amarillo=11;       //el ánodo del led amarillo irá al pin 11
const int verde=9;          //el ánodo del led verde irá al pin 9

void setup() {
  pinMode(rojo,OUTPUT);      //el pin al que va el ánodo del rojo es de salida
  pinMode(amarillo,OUTPUT);  //el pin al que va el ánodo del amar es de salida
  pinMode(verde,OUTPUT);     //el pin al que va el ánodo del verde es de salida
  digitalWrite(rojo,LOW);    //apagamos el led rojo por si estuviera encendido
  digitalWrite(amarillo,LOW); //apagamos el led amar por si estuviera encendido
  digitalWrite(verde,LOW);   //apagamos el led verde por si estuviera encendido
}

void loop() {
  digitalWrite(rojo,HIGH);   //encendemos el led rojo...
  delay(2000);               //durante 2 s...
  digitalWrite(rojo,LOW);    //y lo apagamos
  digitalWrite(verde,HIGH);  //encendemos el led verde...
  delay(2000);               //durante 2 s...
  digitalWrite(verde, LOW);  //y lo apagamos
  digitalWrite(amarillo,HIGH); //encendemos el led amarillo...
  delay(500);                //durante 0,5 s...
  digitalWrite(amarillo,LOW); //y lo apagamos
}
```

Ampliación. ¿Cómo harías para que el rojo permaneciese encendido 3 s, el verde 2 s y el amarillo 0,3 s?

PRÁCTICA 4. El semáforo cont. (condicional IF)

Materiales. 1 led rojo, 1 led amarillo, 1 led verde y 3 resistencias de 220 Ω (rojo-rojo-marrón).

Explicación. Con el mismo circuito físico que en la práctica anterior, deberemos cambiar el sketch de Arduino para que funcione de la siguiente manera. La primera vez que se ejecute el void loop(), los tres leds seguirán la secuencia: verde (1 s), amarillo (1 s) y rojo (1 s). Cada vez que se ejecute el bucle, el tiempo que permanecerá encendido cada led disminuirá en 0,1 s. Cuando cada led permanezca encendido solo 0,1 s, el retardo permanecerá con ese valor de 0,1 s, no disminuyendo más.

Vamos a declarar una variable entera a la que llamaremos `retardo`, cuyo valor inicial será 1000 para que la primera vez que se ejecute el void loop() cada led aguante encendido 1 s. Así, usaremos la variable `retardo` para decirle a los leds cuánto tiempo han de permanecer encendidos.

En el void loop() tenemos que conseguir que si el valor de la variable `retardo` es mayor de 100, entonces el valor de `retardo` para el siguiente loop disminuirá en 100. Para disminuir la variable `retardo` en 100 escribiremos:

```
retardo = retardo-100;
```

con esto estamos diciendo que el nuevo valor de `retardo` va a ser el antiguo valor de `retardo` menos 100. Así, ese `=` no lo tenemos que tomar como un igual usual de matemáticas, sino como un operador de asignación, pues le asignamos a la variable el valor que tenía anteriormente menos 100.

También, en el void loop() tenemos que hacer que si `retardo` no vale más de 100, entonces no se disminuya el valor de `retardo` para el siguiente loop. Para ello emplearemos el condicional `if`. Veamos cómo funciona. Su sintaxis es:

```
if(condición){  sentencias  }
```

Si lo que escribamos en condición es cierto, entonces se ejecutarán las sentencias que escribamos entre llaves. Si condición es falso, entonces no ejecutará las sentencias que escribamos entre llaves.

```
if(condición){  sentencias1  }  
else {  sentencias2  }
```

Si lo que escribamos en condición es cierto, entonces se ejecutarán las sentencias1 que escribamos entre llaves. Si condición es falso, entonces se ejecutarán las sentencias2 que escribamos entre llaves.

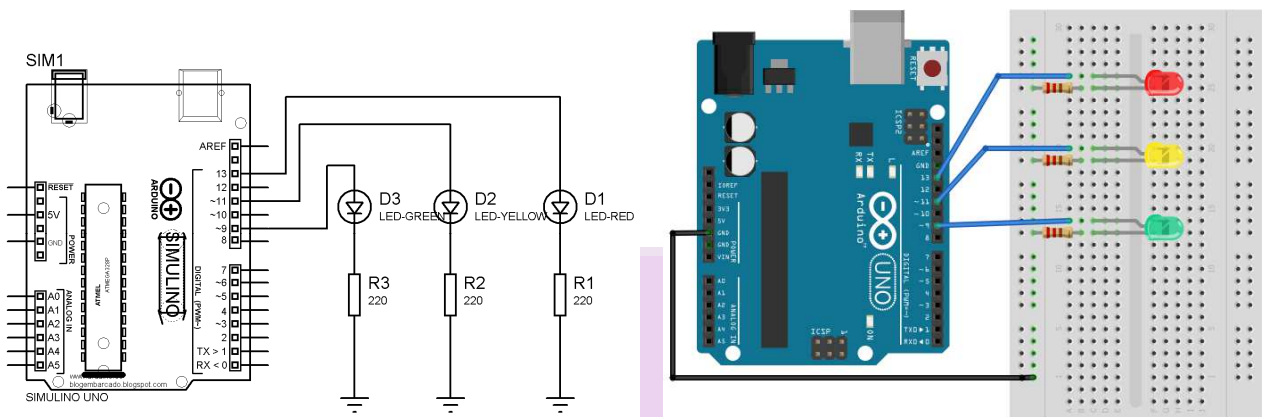
```
if(condición1){  sentencias1  }  
else if (condición2){  sentencias2  }  
else if (condición3){  sentencias3  }  
...  
else if (condiciónN){  sentenciasN  }  
else {  sentenciasN+1  }
```

Si lo que escribamos en condición1 es cierto, entonces se ejecutarán las sentencias1 que escribamos entre llaves. En caso contrario, si condición2 es cierto, entonces se ejecutarán las sentencias2 que escribamos entre llaves. En caso contrario, si condición3 es cierto, entonces se ejecutarán las sentencias3 que escribamos entre llaves. Así, hasta condiciónN. Si condición N también es falso, entonces se ejecutarán las sentenciasN+1 que escribamos entre llaves.

Los operadores lógicos para las condiciones son:

< (menor) <= (menor o igual) > (mayor) >= (mayor igual) == (igual) != (distinto)
&& (operador lógico Y) || (operador lógico O) ! (operador NO)

Esquemas eléctricos



Sketch.

```
const int rojo=13;           //el ánodo del led rojo irá al pin 13
const int amarillo=11;       //el ánodo del led amarillo irá al pin 11
const int verde=9;          //el ánodo del led verde irá al pin 9
int retardo=1000;           //cada led estará encendido inicialmente 1 s

void setup() {
  pinMode(rojo,OUTPUT);      //el pin al que va el ánodo del rojo es de salida
  pinMode(amarillo,OUTPUT);  //el pin al que va el ánodo del amar es de salida
  pinMode(verde,OUTPUT);     //el pin al que va el ánodo del verde es de salida
  digitalWrite(rojo,LOW);    //apagamos el led rojo por si estuviera encendido
  digitalWrite(amarillo,LOW); //apagamos el led amar por si estuviera encendido
  digitalWrite(verde,LOW);   //apagamos el led verde por si estuviera encendido
}

void loop() {
  digitalWrite(rojo,HIGH);    //encendemos el led rojo...
  delay(retardo);             //durante el valor de retardo en ms...
  digitalWrite(rojo,LOW);     //y lo apagamos
  digitalWrite(verde,HIGH);   //encendemos el led verde...
  delay(retardo);             //durante el valor de retardo en ms...
  digitalWrite(verde, LOW);   //y lo apagamos
  digitalWrite(amarillo,HIGH); //encendemos el led amarillo...
  delay(retardo);             //durante 0,5 s...
  digitalWrite(amarillo,LOW); //y lo apagamos
  if(retardo>100){
    retardo=retardo-100;
  }
}
```

Ampliación. ¿Cómo harías para que después de hacer el ciclo de retardo de 0,2 s haga el de 0,1 s cinco veces y luego se apague el semáforo definitivamente?

PRÁCTICA 5. Coche fantástico (bucle FOR)

Materiales. 7 leds rojos y resistencias de 220 Ω (rojo-rojo-marrón).

Explicación. Se pretende que los 7 leds se vayan encendiendo y apagando de forma cíclica, como en el coche fantástico, según esta secuencia: 1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2. El led 1 estará conectado al pin 1, el led 2 al pin 2 y así sucesivamente. La duración de encendido de cada led será de 0,1 s. Si bien el sketch se puede realizar sin usar la sentencia FOR, nosotros nos aprovecharemos de esta sentencia para que el sketch ocupe bastantes menos líneas de código.

Declararemos una variable a la que llamaremos *i* como entera; esta variable será usada en los bucles for. Si escribimos:

```
for(i=1;i<=6;i++){  sentencias  }
```

sucede lo siguiente: el *i*=1 dentro de los paréntesis del for significa que inicialmente la variable *i* vale 1. Después se comprueba si la condición *i*<=6 es cierta o no. En caso de no ser cierta, no se ejecutarán las sentencias que hay entre llaves y se saldrá del bucle for. Como en nuestro caso sí es cierta, pues *i* vale 1, se ejecutarán las sentencias entre llaves. Después se ejecutará la sentencia *i*++ de dentro del paréntesis del for. En este lenguaje, al igual que en C o en C++, escribir *i*++ es equivalente a escribir *i*=*i*+1. Por tanto, al finalizar de ejecutar las sentencias del for entre llaves la variable *i* se incrementará en 1, esto es, pasará a valer 2. Ahora se volverá a comprobar si la condición *i*<=6 es cierta o no. Como sí es cierta, pues *i* vale 2, se ejecutarán las sentencias entre llaves y, después, se ejecutará la sentencia *i*++; esto es, *i* pasará a valer 3. En definitiva, lo que tendremos es que el grupo de sentencias entre llaves se ejecutarán un total de seis veces; en la primera *i* vale 1, en la segunda *i* vale 2, ..., en la sexta *i* vale 6; después de la sexta pasada *i* valdrá 7 con lo que nos saldremos del bucle for.

Si escribimos:

```
for(i=5;i>=2;i--){  sentencias  }
```

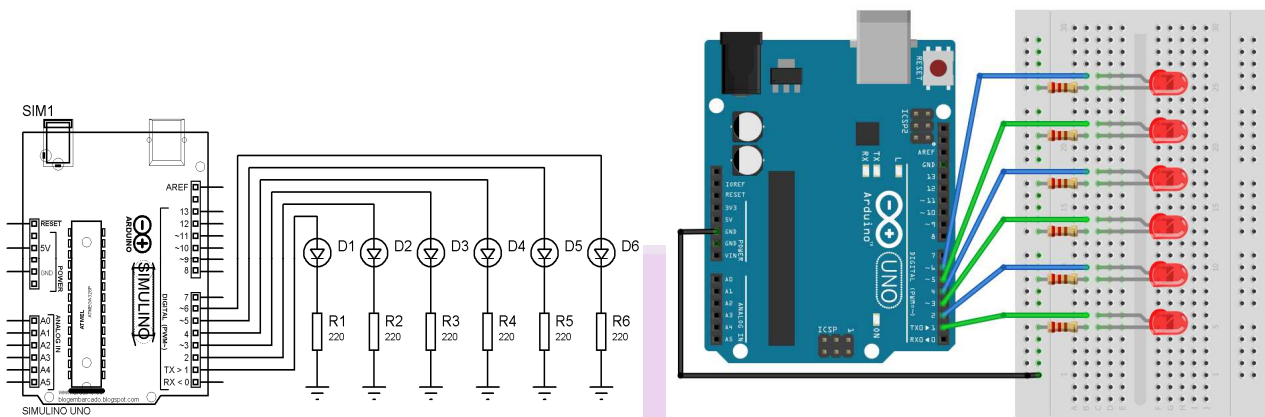
en la primera pasada del bucle *i* vale 5, en la segunda *i* vale 4, en la tercera *i* vale 3, en la cuarta *i* vale 2 y después nos saldremos del bucle.

Si escribimos:

```
for(i=1;i<=6;i++){  
    digitalWrite(i,HIGH);  
    delay(100);  
    digitalWrite(i,LOW);  
}
```

en la primera pasada del bucle el pin 1 se pone en alto, se espera 0,1 s y el pin 1 se pone en bajo. En la segunda pasada el pin 2 se pone en alto, se espera 0,1 s y el pin 2 se pone en bajo. Así, hasta la sexta pasada, donde el pin 6 se pone en alto, se espera 0,1 s y el pin 6 se pone en bajo. Después salimos del bucle.

Esquemas eléctricos



Sketch

```
int i; //esta variable valdrá para numerar los pines y los leds,...
      //pues al pin 1 conectaremos el ánodo del led 1 y así.

void setup() {
  for(i=1;i<=6;i++){          //vamos a realizar este bucle 6 veces
    pinMode(i,OUTPUT);        //el pin i es de salida
    digitalWrite(i,LOW);       //apagamos el led i en caso de estar encendido
  } //al terminar el bucle hemos declarado los pines del 1 al 6 ...
  //como de salida y hemos apagado los seis leds
}

void loop() {
  for(i=1;i<=6;i++){
    digitalWrite(i,HIGH); //encendemos el led i ...
    delay(100);           //durante 0,1 s ...
    digitalWrite(i,LOW);  //y luego lo apagamos
  } // al terminar el bucle hemos hecho la secuencia de leds 1, 2, ..., 6
  for(i=5;i>=2;i--){
    digitalWrite(i,HIGH); //encendemos el led i ...
    delay(100);           //durante 0,1 s ...
    digitalWrite(i,LOW);  //y luego lo apagamos
  } //al terminar el bucle hemos hecho la secuencia de leds 5, 4, 3, 2.
}
```

Ampliación. ¿Cómo harías para que se encendieran alternativamente los pines pares y los impares?

PRÁCTICA 6. Led a distinta intensidad (función analogWrite)

Materiales. 1 led rojo y 1 resistencia de 220 Ω (rojo-rojo-marrón).

Explicación. Se pretende que un led pase de no lucir nada hasta lucir al máximo, luego volverá a ir luciendo cada vez menos hasta que se pague y se vuelva a repetir el ciclo. Los tiempos que tarde en pasar de apagado a totalmente encendido y viceversa son irrelevantes siempre que lo podamos percibir claramente.

Para ello debemos conectar el ánodo del led a uno de los siguientes pines digitales: 3, 5, 6, 9, 10 o 11. Estos pines son PWM~, lo que significa que aunque están bien a 0 V bien a 5 V por ser digitales, pueden simular voltajes analógicos intermedios. Por ejemplo, si en la mitad del ciclo está en alto y la otra mitad está en bajo, entonces se comporta en término medio como si fuese una salida de 2,5 V.

Conectaremos el ánodo del led al pin 11, que es PWM~. Tendremos que declarar este pin como de salida en el setup().

Si escribimos:

```
analogWrite(11, 0);
```

el pin 11 se pondrá a 0 voltios.

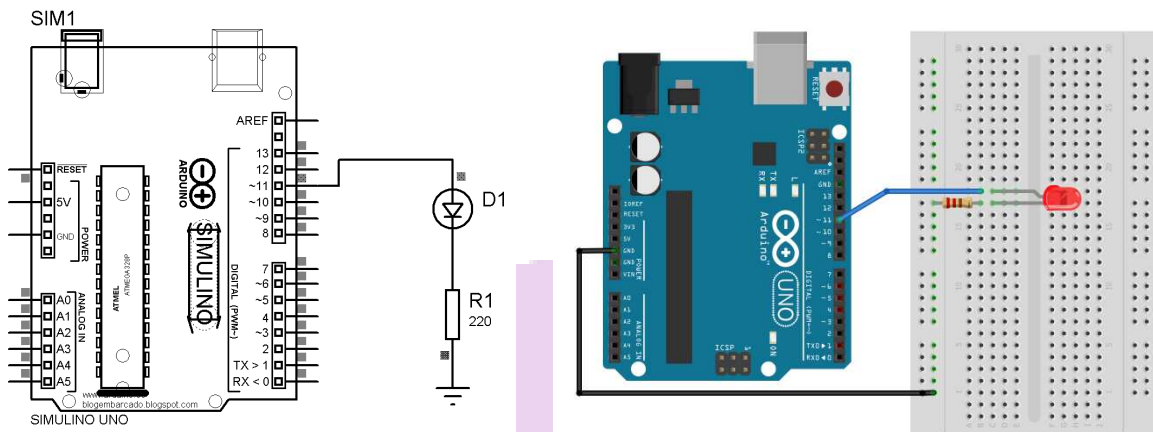
Si escribimos:

```
analogWrite(11, 255);
```

el pin 11 se pondrá a 5 voltios. Así, el primer argumento de analogWrite es el pin PWM~ en el que queremos escribir. El segundo argumento debe valer entre 0 y 255; 0 para 0 V y 255 para 5 V. Si quisiéramos 2,5 V escribiríamos 127.5.

Como pista, para que el led se encienda poco a poco haremos un bucle for con la variable i en el cual la variable i valga inicialmente 0 y vaya aumentando hasta 255. Dentro de las llaves del bucle escribiremos la sentencia `analogWrite(11, i);` y le daremos un delay de 10 ms porque si no, encendería demasiado rápido. Después habrá que hacer otro bucle for en el cual la variable i vaya desde 254 hasta 1.

Esquemas eléctricos



Sketch

```
const int led = 11; //conectaré el ánodo del led al pin 11, que es PWM~
int i; //usaremos la variable i para contar

void setup() {
  pinMode(led, OUTPUT); //el pin al que conecto el ánodo del led es de salida
  analogWrite(led, 0); //apagamos el led inicialmente
}

void loop() {
  for(i=0;i<=255;i++){
    analogWrite(led,i); //escribimos en el led el valor i
    delay(10); //sin este retraso no nos daríamos cuenta de que el led ...
                //se está encendiendo paulatinamente
  } //al término del bucle el led pasa de estar apagado a totalmente encendido
  for(i=254;i>=1;i--){
    analogWrite(led,i);
    delay(10);
  } //al término del bucle el led pasa de estar casi totalmente encendido ...
    //a estar casi totalmente apagado
}
```

Ampliación. ¿Cómo harías para que una vez que esté totalmente encendido espere medio segundo antes de empezar a apagarse y lo mismo para cuando esté totalmente apagado?

PRÁCTICA 7. Control de la luminosidad de un LED (función analogRead)

Materiales. 1 led rojo, 1 potenciómetro 10 k Ω , 1 resistencia de 220 Ω (rojo-rojo-marrón).

Explicación. Se pretende controlar la luminosidad de un led mediante un potenciómetro usando Arduino. Si bien esto se puede hacer sin Arduino, al incorporarlo vamos a poder saber qué voltaje le llega al led en cada momento. La idea es que vamos a leer el voltaje de la pata central del potenciómetro, incluso podemos sacar por pantalla dicho voltaje, y vamos a darle ese voltaje al led.

Como la luminosidad del led va a ser variable, conectaremos su ánodo a un pin digital tipo PWM~, por ejemplo el pin 11. Recordemos que si queremos escribir un voltaje en el pin PWM~ 11, primero lo declararemos como pin de salida en el setup() y después escribiremos:

```
analogWrite(11,brillo);
```

donde la variable entera brillo puede valer entre 0 y 255.

Ya sabemos que los pines analógicos son A0, A1, ..., A5 y que todos los pines analógicos son de lectura, por lo que no hace falta declararlos en el setup(). Lo que vamos a aprender ahora es cómo leer el voltaje de un pin analógico. Para ello usaremos la función analogRead. La sentencia:

```
analogRead(A0);
```

lee el voltaje del pin analógico A0 con la siguiente salvedad; el valor de `analogRead(A0)` es 0 si A0 está a 0 V y es 1023 si A0 está a 5 V. Si A0 está a 2,5 voltios, entonces valdrá 512,5. Por tanto, `analogRead(A0)`; nos va a dar un valor entre 0 y 1023.

Así, si queremos pasar el voltaje del pin analógico (de lectura) A0 al pin digital PWM~ (de escritura) 11 de manera sencilla, escribiremos:

```
brillo=analogRead(A0)/4;
```

```
analogWrite(11,brillo);
```

pues 255 es muy parecido a 1023/4.

Como la luminosidad del led se controla mediante el potenciómetro, deberemos saber el voltaje de la pata central del potenciómetro. Por tanto, uno de los extremos del potenciómetro irá a 5 V, el otro extremo irá a 0 V (da igual cual) y la pata central irá conectada a un pin analógico, por ejemplo el pin A0, para poder leer su voltaje.

Para poder usar la pantalla del serial, en el `void setup()` debemos escribir la sentencia:

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

Aquí, el 9600 significa que la velocidad de transferencia es de 9600 baudios, esto es, 9600 bit/s, que es la velocidad adecuada entre Arduino y el serial o monitor serie.

Para ver por el serial el valor de la variable brillo seguido de un salto de línea escribiremos:

```
Serial.println(brillo);
```

Si no queremos el salto de línea escribiremos:

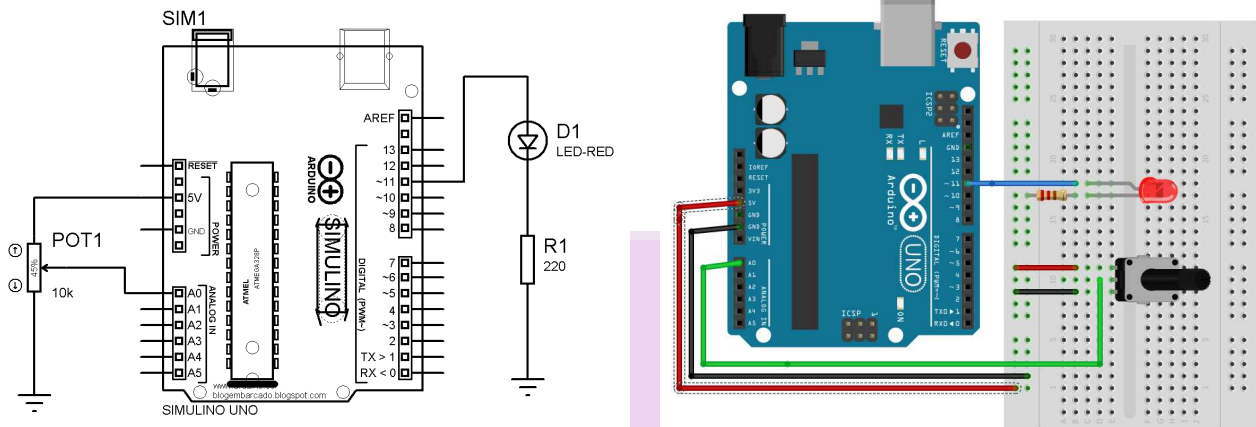
```
Serial.print(brillo);
```

Si queremos que en el serial ponga: "El valor de la variable brillo es:" seguido del valor de la variable brillo y luego un salto de línea, escribiremos:

```
Serial.print("El valor de la variable brillo es: ");
```

```
Serial.println(brillo);
```

Esquemas eléctricos



Sketch

```
const int led = 11; //el ánodo del led al pin 11
const int pot = A0; //el terminal medio del potenciómetro al pin A0
int brillo; // nos dará el brillo del led (entre 0 y 255)

void setup() {
  Serial.begin(9600); //esta sentencia es necesaria para usar la pantalla
  pinMode(11, OUTPUT); //el pin al que conecto el ánodo del led es de salida
  //no es necesario declarar que el pin A0 es de entrada...
  //pues los pines analógicos son de entrada automáticamente
}

void loop() {
  brillo=analogRead(pot)/4; //convertimos el valor de lectura del potenc...
                          //(entre 0 y 1023) en el valor de escritura analógica...
                          //en el led (entre 0 y 255) y lo guardamos en brillo
  analogWrite(led,brillo); //encendemos el led con la luminosidad brillo
  Serial.println(brillo);  //sacamos por la pantalla el valor de brillo...
  delay(100);             //cada 0,1 s
}
```

Ampliación. ¿Cómo harías para que el led no tenga luminosidad variable sino que en el primer tercio del rango del potenciómetro el led esté apagado, en el segundo encendido y en el tercero otra vez apagado?

Práctica 8. Control de un led RGB

Materiales. 1 led RGB, 3 potenciómetros de 10 k Ω , 3 resistencias de 220 Ω (rojo-rojo-marrón).

Explicación. Esta práctica es parecida a la anterior, pero multiplicada por tres. Usaremos un led RGB, lo que nos permitirá obtener cualquier color en el led. Así, lo que pretendemos es poner en el led RGB el color que queramos, controlado mediante tres potenciómetros.

Un led RGB, en realidad, son tres leds: uno rojo (R), uno verde (G) y uno azul (B), que son los tres colores primarios luz. Mezclando estos tres colores primarios, podemos obtener cualquier otro color. El modelo de led RGB con el que se ha hecho esta práctica tiene 4 patas en este orden: ánodo R (rojo), cátodo común, ánodo B (azul) y ánodo G (verde). Esto puede variar de unos modelos a otros.

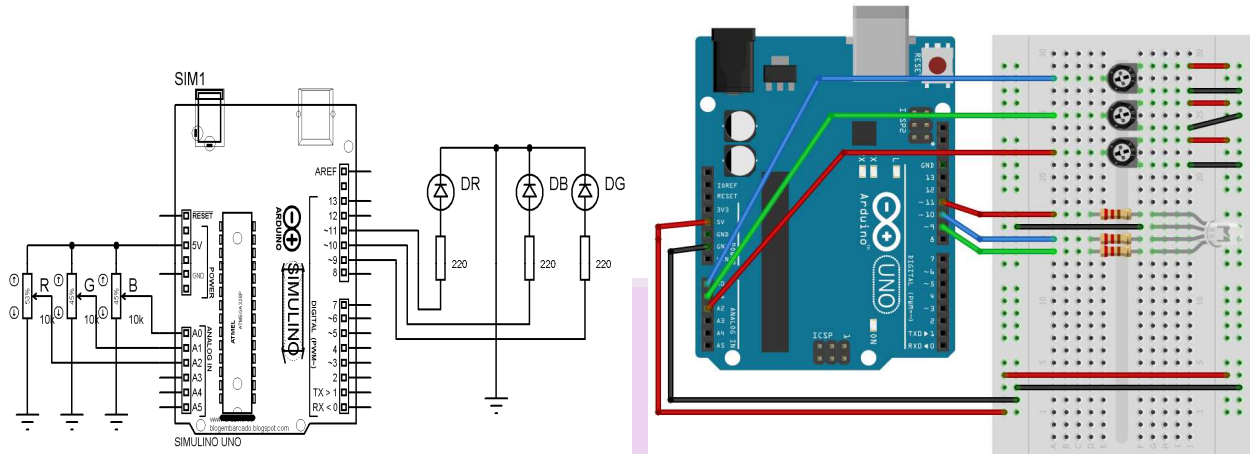
Se pretende controlar el color global del led. Por ejemplo, el rojo puro tiene un valor RGB: R=255, G=0; B=0. El verde puro: R=0, G=255, B=0. El logo de Oikos Matematikón: R=103, G=22, B=116. Debemos conseguir controlar el led para conseguir los siguientes colores: rojo, verde, azul y logo Oikos Matematikón y logo Arduino.

Como pistas, en cuanto al led, debemos conectar cada ánodo a un pin digital tipo PWM~; por ejemplo, la pata R al pin 11, la pata G al 9 y la pata B al 10. En cuanto a los potenciómetros debemos conectar cada terminal medio a un pin analógico; por ejemplo, el potenciómetro que controle el color rojo al pin A2, el que controle el verde al A1 y el que controle el azul al A0.



OIKOS

Esquemas eléctricos



Sketch

```
const int ledR = 11; //el ánodo rojo del led al pin 11
const int ledG = 9;  //el ánodo verde del led al pin 9
const int ledB = 10; //el ánodo azul del led al pin 10
const int potR = A2; //el terminal medio del poten rojo al pin A2
const int potG = A1; //el terminal medio del poten verde al pin A1
const int potB = A0; //el terminal medio del poten azul al pin A0
int brilloR;          //nos dará la cantidad de rojo (entre 0 y 255)
int brilloG;          //nos dará la cantidad de verde (entre 0 y 255)
int brilloB;          //nos dará la cantidad de azul (entre 0 y 255)

void setup() {
  Serial.begin(9600); //esta sentencia es necesaria para usar la pantalla
  pinMode(ledR, OUTPUT);
  pinMode(ledG, OUTPUT);
  pinMode(ledB, OUTPUT);
}

void loop() {
  brilloR=analogRead(potR)/4;
  brilloG=analogRead(potG)/4;
  brilloB=analogRead(potB)/4;
  analogWrite(ledR,brilloR);
  analogWrite(ledG,brilloG);
  analogWrite(ledB,brilloB);
  Serial.print("R: ");
  Serial.print(brilloR);
  Serial.print(", G: ");
  Serial.print(brilloG);
  Serial.print(", B: ");
  Serial.println(brilloB);
  delay(100);
}
```

Ampliación. ¿Cómo conseguirías el color del logo de Arduino. Primero hay que adivinar el código RGB del logo de Arduino. Para ello, podemos usar Paint. Abrimos Paint y pegamos el logo de Arduino. Pinchamos en el cuentagotas o selector de color y luego pinchamos en el color del logo de Arduino. Después pinchamos en Editar colores y ahí nos aparecerá su código RGB.

Práctica 9. Luminosidad de led aleatoria (función RANDOM)

Materiales. 1 led amarillo, 1 led rojo, 2 resistencias de $220\ \Omega$ (rojo-rojo-marrón).

Explicación. En esta práctica vamos a hacer que la luminosidad con la que lucen los dos led sea aleatoria, asemejándose a la luz de una vela. Para ello, usaremos la función random, cuya sintaxis es:

`random(mín, máx);`

donde `mín` es el valor mínimo que puede tomar nuestro número aleatorio y `máx` es el número máximo que puede valer nuestro número aleatorio.

Existe otra sintaxis, que es:

`random(máx);`

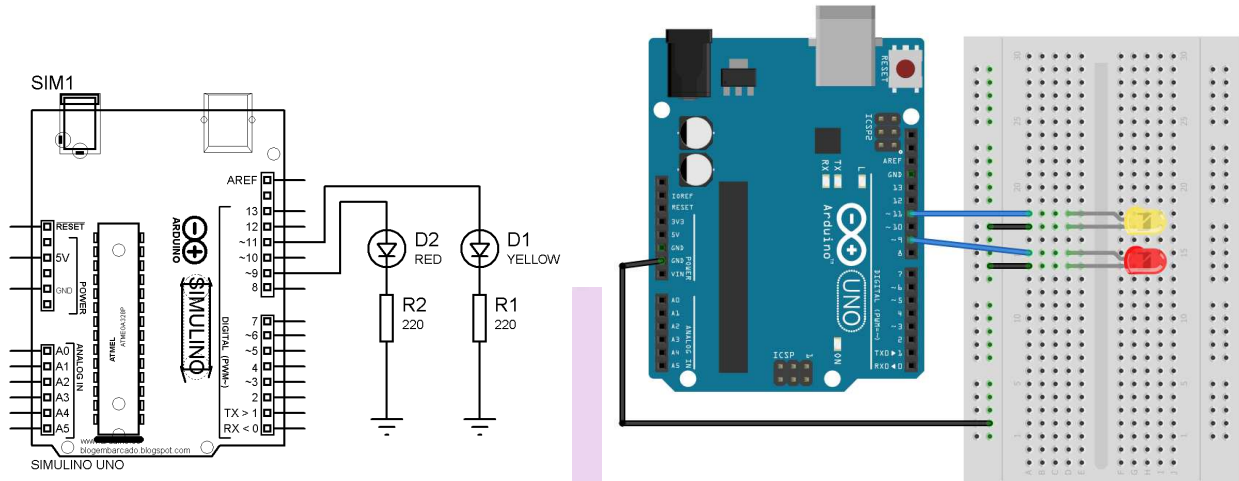
en este caso, el valor mínimo que puede tomar la función es 0.

Deberemos conectar el ánodo de cada led a un pin PWM~; por ejemplo, el ánodo del amarillo al 11 y el ánodo del rojo al 9.

Los números aleatorios valdrán entre 0 y 255. En cada loop generaremos dos números aleatorios; uno para el led amarillo y otro para el led rojo. Escribiremos en el pin de cada uno de esos led un `analogWrite` con el valor aleatorio generado.

OIKOS

Esquemas eléctricos



Sketch

```
const int ledAm = 11; //tiene que ser PWM~
const int ledRo = 9; //tiene que ser PWM~

void setup() {
  pinMode(ledAm, OUTPUT);
  pinMode(ledRo, OUTPUT);
}

void loop() {
  analogWrite(ledAm,random(0,255));
  analogWrite(ledRo,random(0,255));
  delay(200);
}
```

Ampliación. ¿Cómo modificarías el sketch para poder visualizar también por el serial los dos números aleatorios generados en cada loop?

OIKOS

Práctica 10. Control de un piezo (función TONE)

Materiales. 1 piezo.

Explicación. Pretendemos controlar tanto el tono que suena en el piezo, como los sonidos y los silencios. En particular, queremos que el piezo haga una escala con las notas: do, re, mi, fa, sol, la, si. Para ello, tendremos que consultar en internet la frecuencia de cada nota.

El piezo que vamos a usar no tiene polaridad y no necesita de una resistencia que lo proteja, por lo que la práctica será muy sencilla. Uno de los terminales del piezo irá a un pin digital, por ejemplo, el 8. El otro terminal del piezo irá a tierra.

La función tone funciona así:

`tone(arg1, arg2, arg3);`

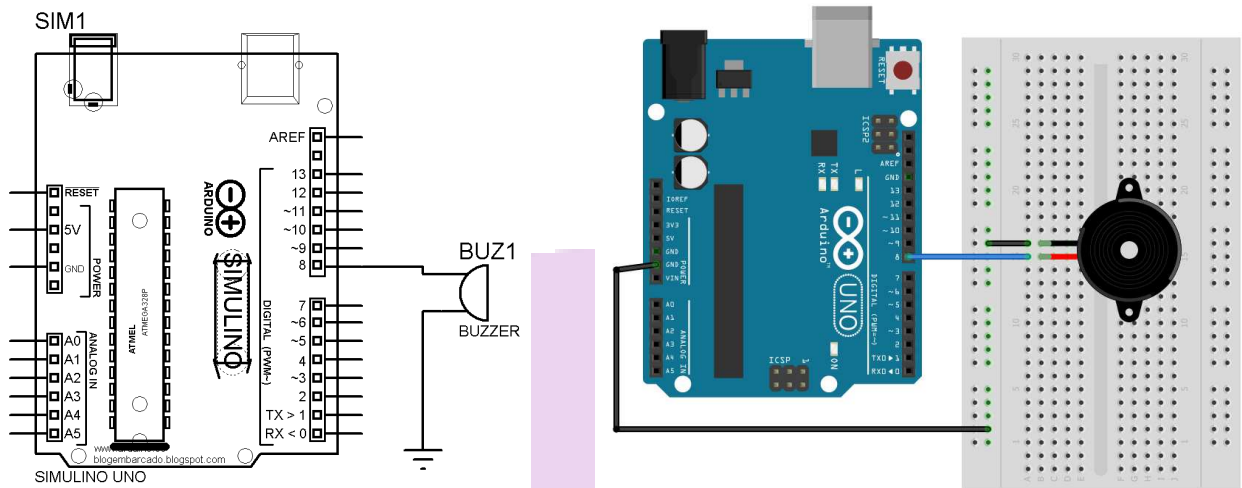
donde, `arg1` es el pin digital al que conectamos uno de los terminales del piezo,
`arg2` es la frecuencia en Hz de la nota musical que queremos que suene,
`arg3` es el tiempo en milisegundos que queremos que suene.

También tenemos la siguiente opción:

`tone(arg1, arg2);`

en este caso, al no estar el argumento del tiempo, la nota no parará hasta que escribamos la instrucción:
`notone(arg1);`

Esquemas eléctricos



Sketch

```
const int piezo = 8;

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

void loop() {
  tone(piezo,261.63,100); //Do durante 0,1 s
  delay(500);
  tone(piezo,293.66,100); //Re durante 0,1 s
  delay(500);
  tone(piezo,329.63,100); //Mi durante 0,1 s
  delay(500);
  tone(piezo,349.23,100); //Fa durante 0,1 s
  delay(500);
  tone(piezo,392.00,100); //Sol durante 0,1 s
  delay(500);
  tone(piezo,440.00,100); //La durante 0,1 s
  delay(500);
  tone(piezo,493.88,100); //Si durante 0,1 s
  delay(500);
}
```

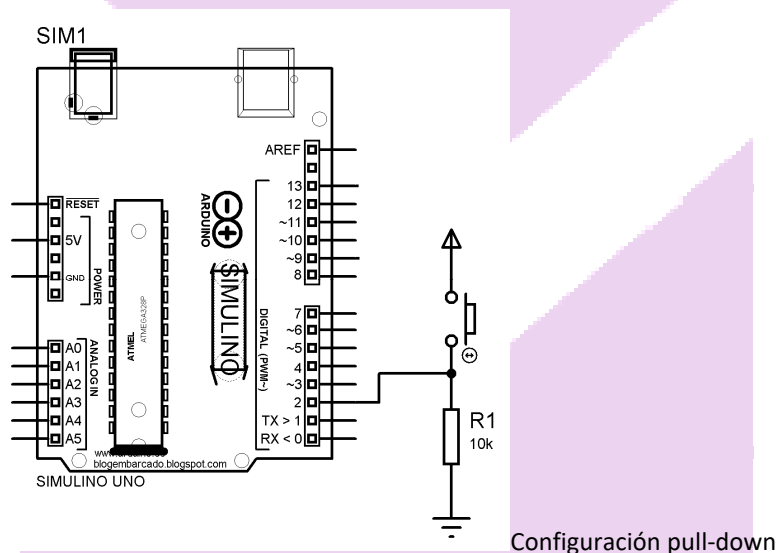
Ampliación. ¿Cómo harías para que el piezo tocara alguna melodía?

Práctica 11. Activar la alarma sonora y visual (config. pull-down)

Materiales. 1 piezo, 1 led rojo, 1 resistencia de $220\ \Omega$ (rojo-rojo-marrón), 1 resistencia de $10\ k\Omega$ (marrón-negro-rojo)

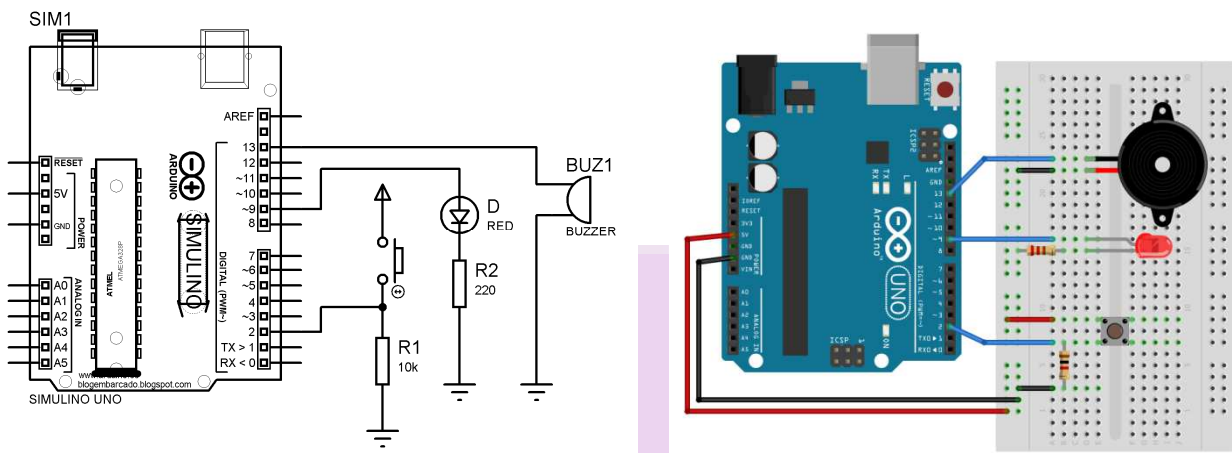
Explicación. Pretendemos que al pulsar instantáneamente un pulsador comience a sonar alternativamente el piezo y a encenderse alternativamente el led. Cuando el piezo suene el led se enciende y cuando el piezo no suene el led se apaga. Cada ciclo de piezo será así: el piezo suena a una frecuencia de $220\ Hz$ durante $0,1\ s$, luego deja de sonar durante $0,4\ s$. Como vemos, cada ciclo dura $0,5\ s$. Al terminar diez ciclos, tanto el piezo como el led se apagan indefinidamente.

Para ello, conectaremos uno de los terminales del pulsador a $5\ V$; el otro terminal del pulsador irá a un pin digital, por ejemplo el 2, (que declaramos como de lectura) y también a una resistencia de $10\ k\Omega$. El otro extremo de esta resistencia irá a tierra (si no estuviera esta resistencia, al accionar el pulsador provocaríamos un cortocircuito). Esta configuración del pulsador se conoce como pull-down. Mientras no lo pulsemos el terminal del pulsador conectado a la resistencia estará a $0\ V$; mientras lo pulsemos dicho terminal estará a $5\ V$.



En cada loop nos preguntaremos si el pulsador está accionado (función IF); si lo está, entonces tendrán lugar los diez ciclos tal como se han descrito (función FOR), para lo que tendremos que conectar un extremo del piezo a un pin digital, por ejemplo el 13, y el otro a tierra, el ánodo del led a otro pin digital, por ejemplo el 9, y el cátodo a tierra protegido por la resistencia de $220\ \Omega$.

Esquemas eléctricos



Sketch

```
const int piezo = 13; //piezo al pin 13
const int led = 9;    //ánodo del led al pin 9
const int puls = 2;   //pulsador al pin 2

void setup() {
  pinMode(piezo, OUTPUT);
  pinMode(led, OUTPUT);
  pinMode(puls, INPUT);
}

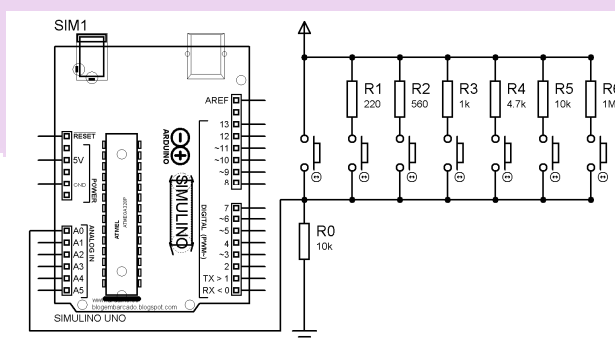
void loop() {
  if (digitalRead(puls)==HIGH){
    for(int i=1;i<=10;i++){
      tone(piezo,220);
      digitalWrite(led,HIGH);
      delay(100);
      noTone(piezo);
      digitalWrite(led,LOW);
      delay(400);
    }
  }
}
```

Ampliación. ¿Cómo harías para que en vez de 10 ciclos fuesen 15 ciclos y cada ciclo durase 20 ms menos?

Práctica 12. Piano 1ª parte (lectura analógica)

Materiales. 1 piezo (se usará en la 2ª parte), 7 pulsadores, 1 resistencia de $220\ \Omega$ (rojo-rojo-marrón), 1 de $560\ \Omega$ (verde-azul-marrón), 1 de $1\ \text{k}\Omega$ (marrón-negro-rojo), 1 de $4,7\ \text{k}\Omega$ (amarillo-violeta-rojo), 2 de $10\ \text{k}\Omega$ (marrón-negro-rojo) y 1 de $1\ \text{M}\Omega$ (marrón-negro-verde).

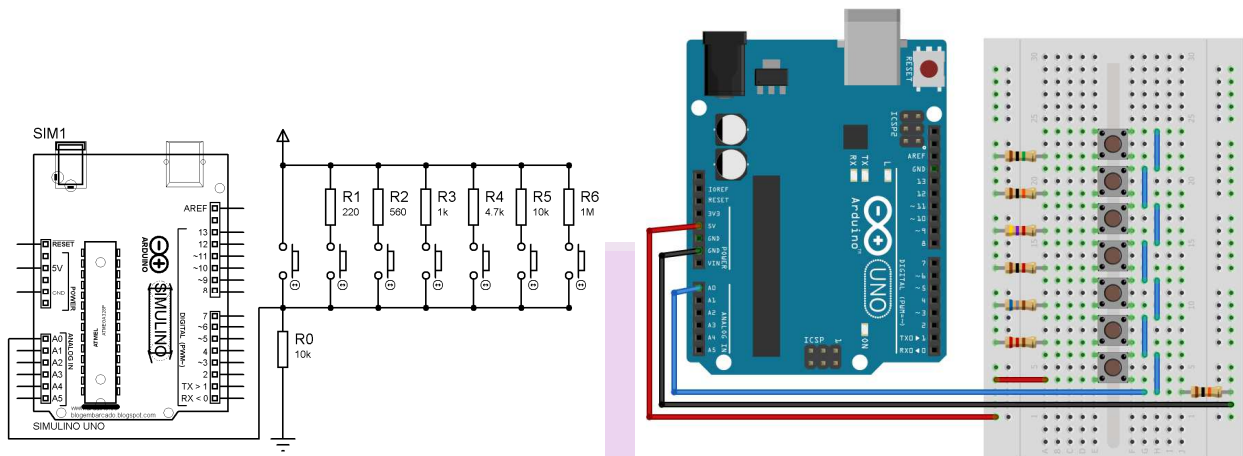
Explicación. Pretendemos hacer un piano con las siete notas musicales entre esta práctica y la siguiente. Cada tecla será un pulsador. Para conseguirlo, cada pulsador se conecta de la siguiente manera: un extremo a una resistencia (cada pulsador a una resistencia de distinto valor) y de ésta a $5\ \text{V}$; el otro extremo se conectará por un lado al pin analógico A0 y por otro a una resistencia fija común de $10\ \text{k}\Omega$ y de ésta a $0\ \text{V}$. El primer pulsador no tendrá resistencia, el segundo la de $220\ \Omega$, el tercero la de $560\ \Omega$, el cuarto la de $1\ \text{k}\Omega$, el quinto la de $4,7\ \text{k}\Omega$, el sexto la de $10\ \text{k}\Omega$ y el séptimo la de $1\ \text{M}\Omega$.



En la primera parte de la práctica, vamos a conectar los siete pulsadores y a leer por el serial el valor de A0 (entre 0 y 1023) según qué pulsador estemos apretando. Hemos conectado resistencias de distinto valor a cada pulsador para que las lecturas de A0 al apretar cada pulsador sean distintas, pues, obviamente, cuanto menor sea la resistencia del pulsador, mayor será el valor de lectura de A0.

Deberemos anotar estos valores que hemos leído, pues en la segunda parte de la práctica, los necesitaremos para que cada pulsador haga sonar en el piezo con una nota musical distinta.

Circuitos eléctricos



Sketch

```
int lectura;

void setup() {
  Serial.begin(9600);
}

void loop() {
  lectura=analogRead(A0);
  if(lectura>3){
    Serial.println(lectura);
    delay(500);
  } //se hace el if para evitar que salgan en el serial ...
    //las lecturas de no apretar ningún pulsador, pues ...
    //si no apretamos ningún pulsador la lectura será 0 ...
    //o muy parecido, mientras que si lo pulsamos será al menos 7.
}
```

Solución a la lectura de valores. Una vez realizada esta primera parte, los valores típicos de lectura que obtendremos serán bastante parecidos a los siguientes: 1023, 1003-1002, 971-970, 932-930, 697-693, 509-508 y 9-7.

Ampliación. ¿Cómo harías para calcular estos valores de lectura teóricamente, usando la ley de Ohm?

Práctica 13. Piano 2ª parte (función IF ELSE)

Materiales. Solo hay que añadir el piezo a la práctica anterior. 1 piezo (se usará en la 2ª parte), 7 pulsadores, 1 resistencia de 220 Ω (rojo-rojo-marrón), 1 de 560 Ω (verde-azul-marrón), 1 de 1 k Ω (marrón-negro-rojo), 1 de 4,7 k Ω (amarillo-violeta-rojo), 2 de 10 k Ω (marrón-negro-rojo) y 1 de 1 M Ω (marrón-negro-verde).

Explicación. Una vez hecha la primera parte, vamos a terminar el piano. Debemos conseguir que al apretar el primer pulsador (el que no tiene resistencia) suene un DO (261,63 Hz), al apretar el segundo un RE (293,66 Hz), al apretar el tercero un MI (329,63 Hz), con el cuarto un FA (349,23 Hz), con el quinto un SOL (392,00 Hz), con el sexto un LA (440 Hz) y con el séptimo un SI (493,88 Hz).

Conectaremos uno de los extremos del piezo al pin 13.

Para programar el sketch usaremos las sentencias if – else if – else, que pasamos a recordar.

```
if (condición1) {  sentencias1  }  
else if (condición2) {  sentencias2  }  
else if (condición3) {  sentencias3  }  
...  
else if (condiciónN) {  sentenciasN  }  
else {  sentenciasN+1  }
```

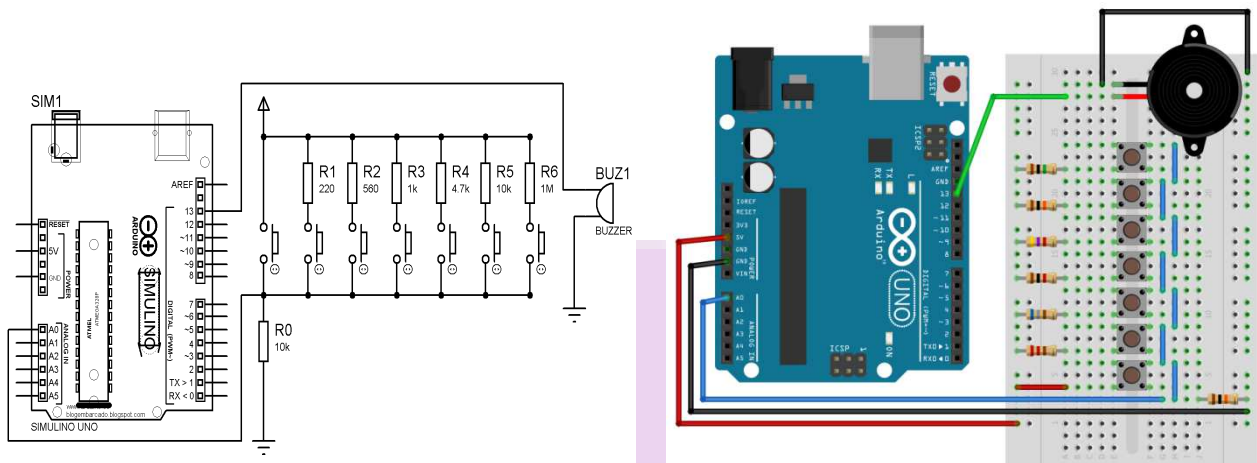
Si lo que escribamos en condición1 es cierto, entonces se ejecutarán las sentencias1 que escribamos entre llaves. En caso contrario, si condición2 es cierto, entonces se ejecutarán las sentencias2 que escribamos entre llaves. En caso contrario, si condición3 es cierto, entonces se ejecutarán las sentencias3 que escribamos entre llaves. Así, hasta condiciónN. Si condición N también es falso, entonces se ejecutarán las sentenciasN+1 que escribamos entre llaves.

Los operadores lógicos para las condiciones son:

< (menor) <= (menor o igual) > (mayor) >= (mayor igual) == (igual) != (distinto)
&& (operador lógico Y) || (operador lógico O) ! (operador NO)

Hecha la primera parte de la práctica, nos habrán salido unos valores de lectura de A0 parecidos a los siguientes: 1023, 1003-1002, 971-970, 932-930, 697-693, 509-508 y 9-7.

Circuitos eléctricos



Sketch

```

const int piezo = 13; //el piezo al pin 13
int lectura;

void setup() {
  //Serial.begin(9600);
  pinMode(piezo,OUTPUT);
}

void loop() {
  lectura=analogRead(A0);
  if(lectura>1010){          //1010 está entre 1003 y 1023
    tone(piezo,261.63,100); //Do durante 0,1 s
  }
  else if (lectura>990){    //990 está entre 971 y 1002
    tone(piezo,293.66,100); //Re durante 0,1 s
  }
  else if (lectura>950){    //950 está entre 932 y 970
    tone(piezo,329.63,100); //Mi durante 0,1 s
  }
  else if (lectura>910){    //910 está entre 697 y 930
    tone(piezo,349.23,100); //Fa durante 0,1 s
  }
  else if (lectura>600){    //600 está entre 509 y 693
    tone(piezo,392.00,100); //Sol durante 0,1 s
  }
  else if (lectura>400){    //400 está entre 9 y 508
    tone(piezo,440.00,100); //La durante 0,1 s
  }
  else if (lectura>4){      //4 es mayor que 0 y menor que 7
    tone(piezo,493.88,100); //Si durante 0,1 s
  }
  else {
    noTone(piezo);
  }
}

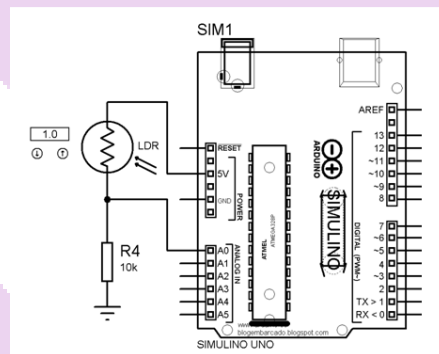
```

Práctica 14. Farola (resistencia LDR)

Materiales. 1 resistencia LDR, 1 resistencia de 10 k Ω (marrón-negro-rojo), 3 leds rojos y 3 resistencias de 220 Ω (rojo-rojo-marrón).

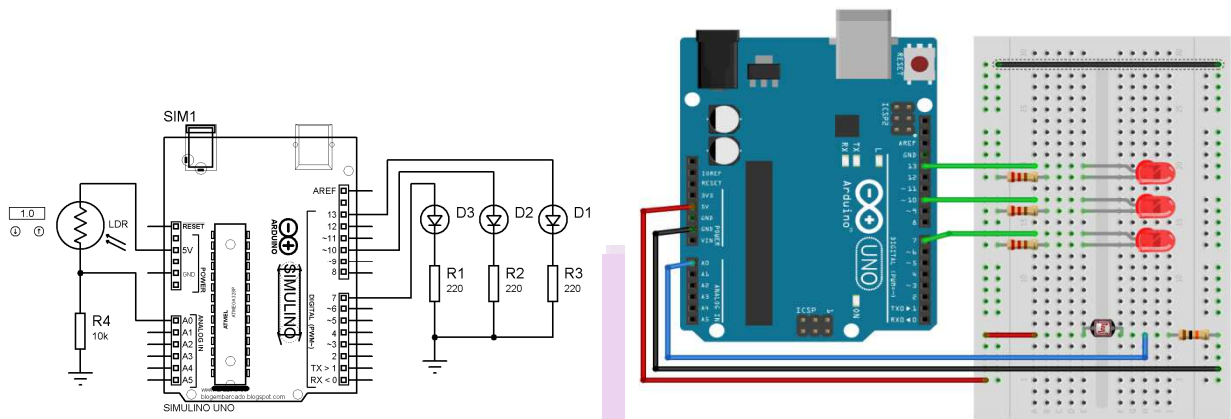
Explicación. Pretendemos que cuando haya muy poca luz se enciendan los tres leds, que cuando haya poca luz se enciendan dos leds, que cuando haya un poco más de luz se encienda solo uno y cuando haya mucha luz no se encienda ninguno. Para ellos necesitaremos una resistencia LDR, pues su valor óhmico depende de la luz que incida sobre ella.

Una resistencia LDR tiene menor valor óhmico cuanto más luz le llegue; su valor puede ir de unos 50 Ω (mucha luz) a varios megaohmios (muy poca luz). Una LDR no tiene polaridad. En una primera parte de la práctica habrá que calibrar esto. Conectaremos un extremo de la LDR a 5V y el otro extremo, tanto a la entrada analógica A0 como a una resistencia de 10 k Ω que estará conectada a tierra. La información de la luz nos la dará la lectura de A0: valores cercanos a 1023 significan mucha luz y valores cercanos a 0 muy poca luz.



Hecha la calibración nos puede dar algo así: por debajo de 50 los tres leds, por debajo de 200 dos leds, por debajo de 500 un led. En el resto de la práctica debemos conectar los tres leds convenientemente de forma que el conjunto funcione como se especifica en el primer párrafo de la explicación. El ánodo del primer led irá al pin 13, el ánodo del segundo led al pin 10 y el ánodo del tercer led al pin 7. Cuando haya muy poca luz se encienden los tres leds. Cuando haya algo más de luz el tercer led se apaga. Cuando haya más luz el segundo led también se apaga. Cuando haya mucha luz el primer led también se apaga.

Esquemas eléctricos



Sketch

```

const int led1 = 13;
const int led2 = 10;
const int led3 = 7;
int lectura;

void setup() {
  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  pinMode(led3,OUTPUT);
}

void loop() {
  lectura=analogRead(A0);
  if (lectura<50){
    digitalWrite(led1,HIGH);
    digitalWrite(led2,HIGH);
    digitalWrite(led3,HIGH);
  }
  else if (lectura<200){
    digitalWrite(led1,HIGH);
    digitalWrite(led2,HIGH);
    digitalWrite(led3,LOW);
  }
  else if (lectura<400){
    digitalWrite(led1,HIGH);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
  }
  else {
    digitalWrite(led1,LOW);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
  }
  delay(500);
}

```

Práctica 15. Theremin óptico (función MAP)

Materiales. 1 resistencia LDR, 1 resistencia de 10 k Ω (marrón-negro-rojo), 1 piezo.

Explicación. Pretendemos que la frecuencia del piezo varíe con la luz; de esta manera, al acercar las manos a la LDR variará la frecuencia del piezo. Cuando no tapemos nada la LDR, el piezo no debe sonar.

La conexión de la LDR es idéntica a la práctica anterior. La cuestión ahora es que debemos transformar los valores de lectura del pin analógico A0 (desde 0 hasta 1023) en frecuencia del piezo. Para eso usaremos la función MAP

La sintaxis de la función map es:

```
map(x, x0, x1, y0, y1);
```

Imaginemos la recta que pasa por los puntos (x0, y0) y (x1, y1), entonces `map(x, x0, x1, y0, y1)` será el valor de la ordenada de la recta cuando la abscisa vale x.

Vamos a conectar uno de los extremos del piezo al pin 13.

Debemos calibrar los posibles valores que toma la entrada A0 en función de la proximidad de nuestras manos. Una posibilidad de calibración es un valor mínimo de 250 (cuando nuestras manos tapan mucho la LDR) y un valor de 650 cuando no tapamos la LDR.

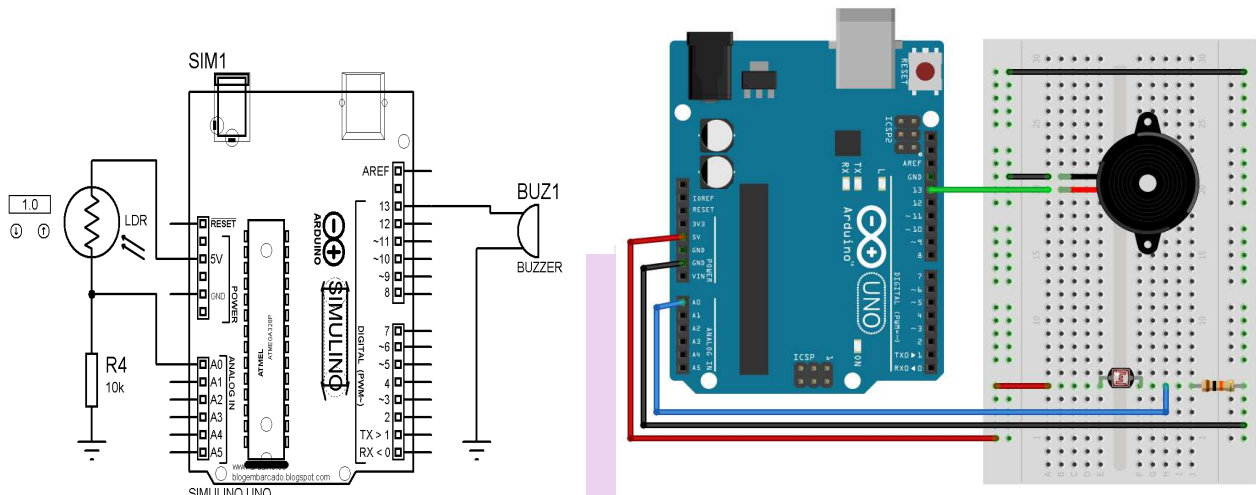
Debemos elegir las frecuencias mínima y máxima que tocará nuestro piezo. Podemos elegir, por ejemplo, una frecuencia mínima de 50 Hz y una frecuencia máxima de 1000 Hz.

Así, podremos escribir:

```
lectura=analogRead(A0);  
frecuencia = map(lectura,250,650,50,1000);
```

donde `lectura` es la variable que lee el pin A0 y `frecuencia` es la variable que determinará la frecuencia a la que debe sonar el piezo.

Esquemas eléctricos



Sketch

```
const int piezo = 13; //piezo al pin 13
const int lectBaja = 250; //lectura de A0 al tapar mucho la ldr
const int lectAlta = 650; //lectura de A0 al no tapar nada la ldr
const int frecBaja = 50; //menor frecuencia a la que sonará
const int frecAlta = 1000; //mayor frecuencia a la que sonará
int lectura; //lectura de A0 de cada instante
int frecuencia; //frec. a la que sonará el piezo

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

void loop() {
  lectura=analogRead(A0);
  frecuencia = map(lectura,lectBaja,lectAlta,frecBaja,frecAlta);
  if (lectura<lectAlta){
    tone(piezo,frecuencia,100);
  } //gracias a este if el piezo no suena si no se tapa nada la ldr
}
```

OIKOS

Práctica 16. Medidor de temperatura (dispositivo TMP36)

Materiales. 1 sensor TMP36, 3 leds rojos y 3 resistencias de 220 Ω (rojo-rojo-marrón).

Explicación. Pretendemos que según el calor que desprenda tu mano se enciendan uno, dos o tres leds. Más leds encendidos significarán más “pasión”. Para ello contamos con el sensor de temperatura TMP36.

Veamos cómo funciona el TMP36. Si lo miramos por su parte plana, el terminal izquierdo se conecta a 5 V, el terminal derecho se conecta a 0 V y el terminal central a un pin analógico; por ejemplo, al A0. El voltaje del terminal central estará comprendido entre 0 V y 5 V y será menor cuanto mayor sea la temperatura.



El TMP36 funciona entre -50°C y 125°C . No es muy preciso ya que tiene una incertidumbre de $\pm 1^{\circ}\text{C}$, pero nos sirve. La función que relaciona el voltaje de salida del TMP36 con la temperatura se debe calcular a partir de estas pistas:

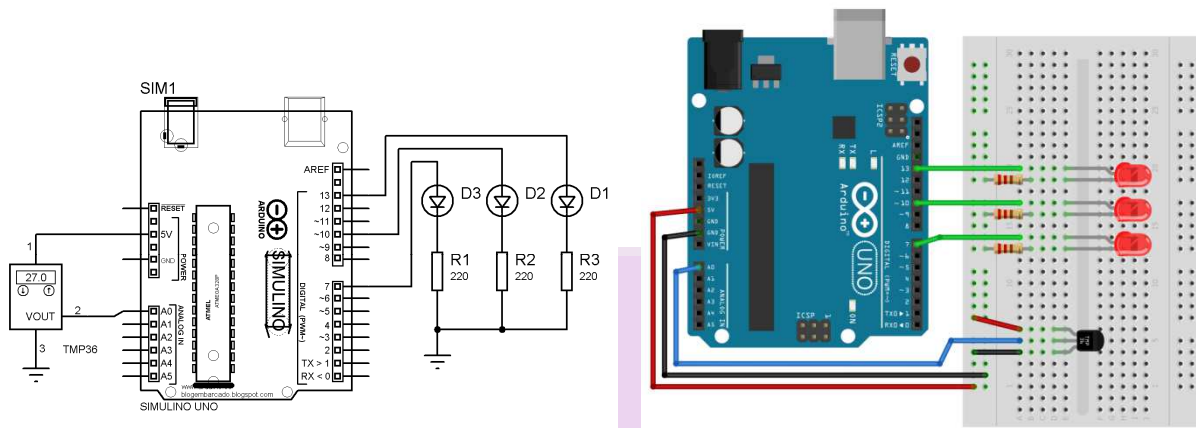
- El voltaje de salida del TMP36 vale 0 V cuando la temperatura es de -50°C .
- Cada vez que la temperatura aumenta un grado el voltaje de salida aumenta 10 mV.

Con esas dos pistas debemos sacar la expresión de la temperatura (en grados centígrados) en función del voltaje (en voltios).

Declaremos una variable llamada `lectura` que leerá el pin A0 (valor entre 0 y 1023). Declaremos una variable en coma flotante llamada `voltaje` que dará el voltaje en voltios del pin A0 (valor entre 0 y 5). Declaremos una variable en coma flotante llamada `temperatura` que dará la temperatura en grados centígrados.

Ahora explicaremos a parte de la práctica correspondiente a la iluminación de leds. Esta parte puede variar sus valores de temperatura de referencia en función de la temperatura que haga en la habitación. Cuando la temperatura esté por encima de 35°C se encenderán los tres leds. Cuando la temperatura esté entre 30°C y 34°C se encenderán dos leds. Cuando la temperatura esté entre 25°C y 29°C se encenderá un led. Cuando la temperatura esté por debajo de 25°C no se encenderá ningún led.

Esquemas eléctricos



Sketch

```

const int led1 = 13;
const int led2 = 10;
const int led3 = 7;
const int temp1 = 35;
const int temp2 = 30;
const int temp3 = 25;
int lectura;
float voltaje;
float temperatura;

void setup() {
  Serial.begin(9600);
  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  pinMode(led3,OUTPUT);
}

void loop() {
  lectura=analogRead(A0);
  voltaje=lectura*5.0/1023.0;
  temperatura=100*voltaje-50;
  Serial.println(voltaje);
  if (temperatura>=temp1){
    digitalWrite(led1,HIGH);
    digitalWrite(led2,HIGH);
    digitalWrite(led3,HIGH);
  }
  else if (lectura>=temp2){
    digitalWrite(led1,HIGH);
    digitalWrite(led2,HIGH);
    digitalWrite(led3,LOW);
  }
  else if (lectura>=temp3){
    digitalWrite(led1,HIGH);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
  }
  else {
    digitalWrite(led1,LOW);
    digitalWrite(led2,LOW);
    digitalWrite(led3,LOW);
  }
  delay(1000);
}

```

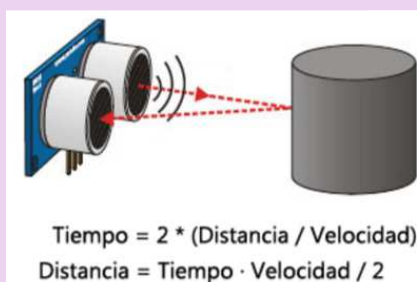

PRÁCTICA 17. Control del sensor de ultrasonidos I

Materiales. 1 sensor de ultrasonidos HC-SR04.

Explicación. En esta práctica vamos a aprender a medir la distancia desde un sensor a un objeto. Para ello, usaremos el sensor de ultrasonidos HC-SR04. Se pide que por el serial salga la distancia, en cm, entre el sensor de ultrasonidos y el primer objeto que se interponga con el sensor.

El rango de medición del HC-SR04 está entre los dos 2 cm y los 500 cm; fuera de estos límites no mide con exactitud. La resolución es de 0,3 cm.

La idea básica es que el sensor lanza un ultrasonido que chocará con el objeto regresando al sensor. Sabiendo el tiempo que dura este proceso y la velocidad del sonido (340 m/s), podremos calcular la distancia a la que se encuentra el objeto.



El HC-SR04 tiene cuatro patas. La primera, la de la izquierda, hay que conectarla a 5 V. La cuarta pata, la de la derecha, a 0 V. La segunda pata se llama trigger y la conectaremos al pin 3, que declaramos de salida. Para que el sensor envíe el ultrasonido, debemos poner la pata trigger en alto durante exactamente 10 μ s, esto es, 0,01 ms. La tercera pata se llama echo y la conectaremos al pin 2, que declaramos de entrada. Una vez que la onda ha regresado al sensor, la pata echo se pondrá en alto exactamente el tiempo que haya durado el proceso de envío y recepción del ultrasonido.

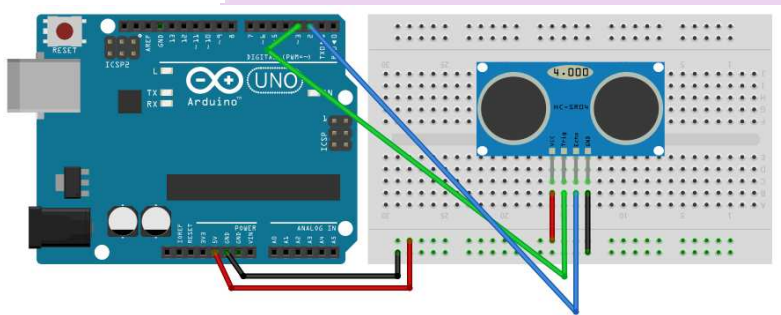
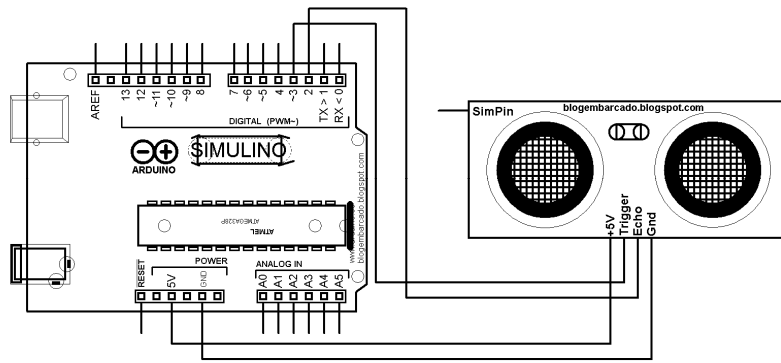
Tenemos la siguiente fórmula: $(\text{distancia_al_objeto}) = (\text{tiempo_ida_y_vuelta}) / 2 \cdot (\text{velocidad_sonido})$

La instrucción `pulseIn(pecho, HIGH)` nos dará el tiempo en microsegundos que el pin que hemos llamado `pecho` está en HIGH; esto es, el $(\text{tiempo_ida_y_vuelta})$ en microsegundos. Así, si a la variable `duracion` le asignamos el valor `pulseIn(pecho, HIGH)`
`duracion = pulseIn(pecho, HIGH);`
 tendremos que `duracion` es el $(\text{tiempo_ida_y_vuelta})$ en microsegundos.

La velocidad del sonido es 340 m/s. Podéis comprobar que es lo mismo que 0.034 cm/ μ s. Así, si a la variable `distancia` le asignamos el valor `duracion/2*0.034`
`distancia = duracion/2*0.034;`
 tendremos que `distancia` es la distancia al objeto en centímetros.

Si `distancia` es mayor o igual que 500 o menor o igual que 0, sabemos que la medida no es fiable. Por tanto, en estos casos saldrá por el serial "-----", en vez de el valor de `distancia`.

Esquemas eléctricos



Sketch

```
const int pecho = 2;           //la pata echo del CI va al pin 2
const int ptrig = 3;           //la pata trigger del CI va al pin 3
int duracion, distancia;       //para calcular distancia

void setup() {
  Serial.begin(9600);           //inicia el puerto serial
  pinMode(pecho, INPUT);        //define el pin 2 como entrada (pecho)
  pinMode(ptrig, OUTPUT);       //define el pin 3 como salida (ptrig)
}

void loop() {
  digitalWrite(ptrig, LOW);     //al comenzar el loop el trigger debe estar en LOW
  delay(10);
  digitalWrite(ptrig, HIGH);    //genera el pulso de trigger por 0,01 ms
  delay(0.01);
  digitalWrite(ptrig, LOW);
  duracion = pulseIn(pecho, HIGH); //tiempo de ida y vuelta en microsegundos
  distancia = (duracion/2)*0.034; //distancia al objeto en centimetros
  delay(10);                    //para darle tiempo
  if ((distancia >= 500) || (distancia <= 0)) { //si la distancia es mayor de 5 m
    Serial.println("-----"); //... o menor o igual a 0 no mide nada
  }
  else{
    Serial.print(distancia);
    Serial.println(" cm");
  }
  delay(100); //para que los datos por el serial no aparezcan tan deprisa
}
```

Ampliación. ¿Cómo harías para que cuando la distancia fuese menor o igual a 10 cm también se encendiera un led?

PRÁCTICA 18. Control del sensor de ultrasonidos II

Materiales. 1 sensor de ultrasonidos HC-SR04, 1 piezo.

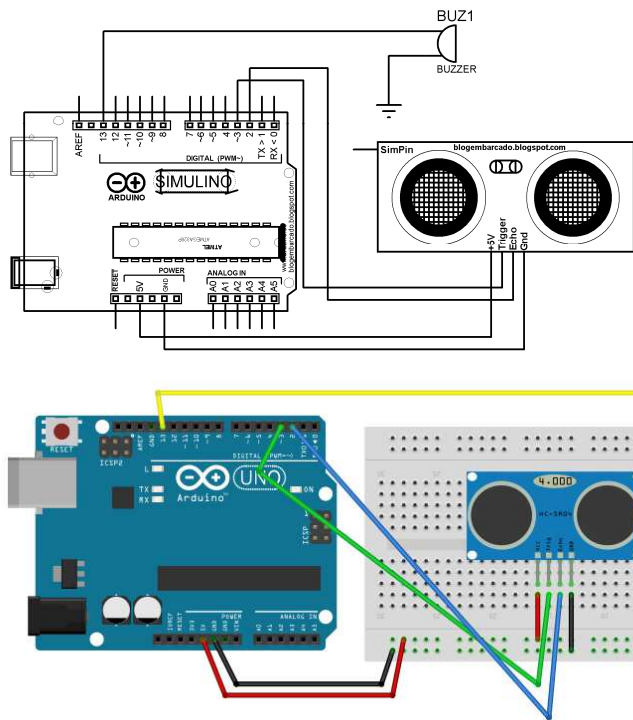
Explicación. Esta práctica es una continuación de la anterior. Aquí vamos a simular la alarma que tienen los coches en su parte trasera cuando perciben un objeto cerca. Seguiremos usando nuestro sensor de ultrasonidos HC-SR04.

Lo que se pretende es que cuando un objeto esté a una distancia de 20 cm empiece a sonar el piezo intermitentemente con una frecuencia de 261.63 Hz durante 50 ms y que pause durante 600 ms. Cuanto más cerca esté el objeto más cortas serán las pausas, de manera que cuando el objeto esté a 2 cm la pausa sea tan solo de 50 ms. Así, para calcular los retardos usaremos la función `map` verificando que cuando la distancia sea de 20 cm la pausa sea de 600 ms y cuando la distancia sea de 2 cm la pausa sea de 50 ms.

También se pide dicha distancia por la pantalla del serial como en la práctica pasada.



Esquemas eléctricos



Sketch

```
const int pecho = 2;           //la pata echo del CI va al pin 2
const int ptrig = 3;           //la pata trigger del CI va al pin 3
const int piezo = 13;          //el piezo al pin 13
int duracion, distancia;       //para calcular distancia

void setup() {
  Serial.begin(9600);           //inicia el puerto serial
  pinMode(pecho, INPUT);        //define el pin 2 como entrada (pecho)
  pinMode(ptrig, OUTPUT);       //define el pin 3 como salida (ptrig)
}

void loop() {
  digitalWrite(ptrig, LOW);     //al comenzar el loop el trigger debe estar en LOW
  delay(2);
  digitalWrite(ptrig, HIGH);    //genera el pulso de trigger por 0,01 ms
  delay(0.01);
  digitalWrite(ptrig, LOW);
  duracion = pulseIn(pecho, HIGH); //tiempo de ida y vuelta en microsegundos
  distancia = (duracion/2)*0.034; //distancia al objeto en centímetros
  delay(2);                     //para darle tiempo
  if ((distancia >= 500) || (distancia < 0)) { //si la distancia es mayor de 5 m
    Serial.println("-----");           //... o menor a 0 no mide nada
  }
  else{
    Serial.print(distancia);
    Serial.println(" cm");
    if(distancia<=20){
      tone(piezo,261.63,50);
      delay(map(distancia,2,20,50,600));
    }
  }
}
```

Ampliación. ¿Cómo harías para que cuando la distancia fuese menor o igual a 10 cm también se encendiera un led?

PRÁCTICA 19. Control de un servo I (librería Servo.h)

Materiales. 1 servo, 2 condensadores electrolíticos de 100 μ F, 1 potenciómetro de 10 k Ω .

Explicación. Pretendemos controlar un servo a través de un potenciómetro, pues la posición del servo dependerá de la posición del potenciómetro; el potenciómetro irá al pin analógico A0.

Un servo es una especie de motor que puede girar un ángulo de entre 0° y 180° con mucha precisión; de hecho, le podemos decir exactamente en qué posición debe estar a través del ángulo. Los servos son fundamentales en robótica. Para controlar un servo con Arduino necesitamos cargar la librería `Servo.h`, pues esta librería contiene instrucciones relativas a los servos. De un servo salen tres terminales: el terminal rojo debe ir a 5 V, el terminal negro debe ir a 0 V y el terminal blanco debe ir a un pin digital de Arduino, por ejemplo el pin 9.



Utilizaremos condensadores en paralelo con el servo porque los motores y los servos consumen bastante corriente al arrancar. También podemos poner un servo en paralelo con el potenciómetro. La función de un condensador es que no se produzcan diferencias muy bruscas de corriente, pues éstas pueden dañar los dispositivos. Mucho ojo porque los condensadores electrolíticos tienen polaridad, así que siempre conectaremos la pata larga al positivo y la corta al negativo.

Las librerías se cargan en las primeras líneas del sketch; así, para cargar la librería `Servo.h` escribiremos en la primera línea del sketch:

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

Una vez hayamos cargado la librería declararemos un objeto tipo Servo al que llamaremos `servito`. Para ello, en la segunda línea del sketch, escribiremos:

```
Servo servito;
```

Declararemos la variable entera `angulo`, que usaremos para decirle al servo su posición y declararemos la variable entera `lectura` que leerá el valor del pin analógico A0, al que conectaremos la pata central del potenciómetro.

Declararemos que hemos conectado su terminal blanco en el pin 9. Para ello, escribiremos en el void `setup()` la sentencia:

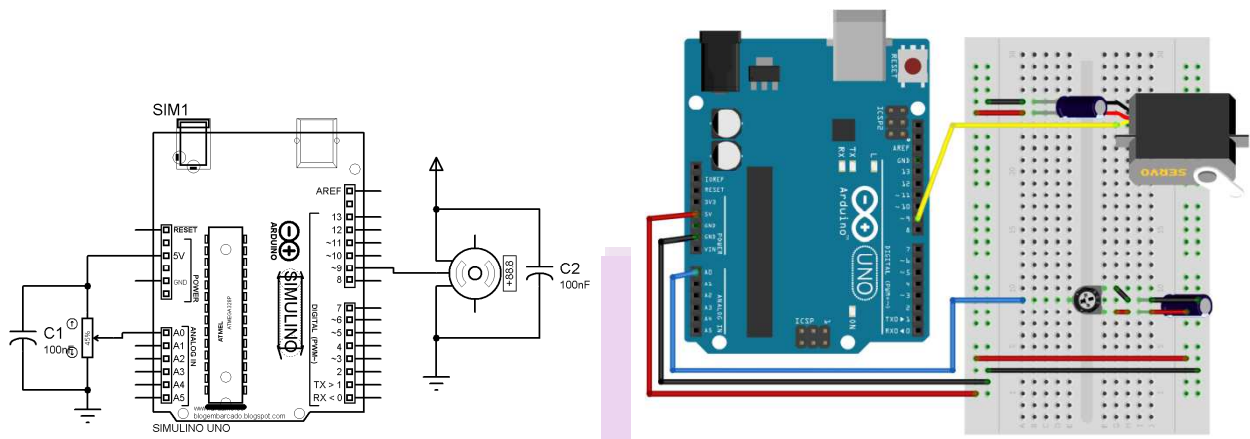
```
servito.attach(9);
```

Como hemos dicho, un servo puede tomar una posición de entre 0° y 180°. Si quisiésemos que tomara la posición de 150°, escribiríamos:

```
servito.write(150); //ordena a servito ponerse en la posición 150 grados  
delay(20); //para darle tiempo al servo a ponerse en la posición 150 grados
```

Haremos girar al servo un ángulo en función de la lectura analógica A0, que depende de la posición del potenciómetro; por tanto, necesitaremos usar la función `map` para pasar de la lectura de A0 (entre 0 y 1023) al ángulo que debe girar (entre 0 y 179).

Esquemas eléctricos



Sketch

```
#include <Servo.h> //Carga la librería para trabajar con servos
Servo servito;     //Declaramos un objeto tipo Servo llamado servito
int angulo;        //Esta variable almacena el ángulo que debe girar servito
int lectura;       //Esta variable almacena la lectura del pin A0

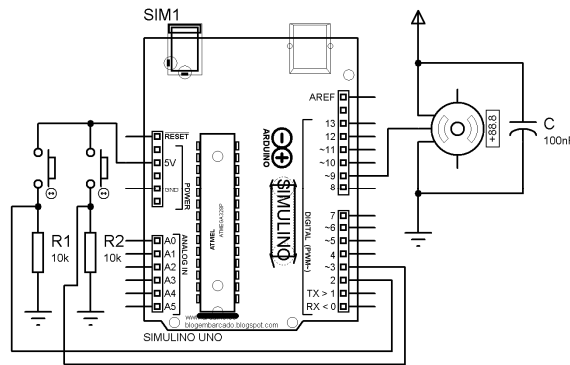
void setup() {
  servito.attach(9); //servito está conectado al pin 9
}

void loop() {
  lectura=analogRead(A0);
  angulo=map(lectura,0,1023,0,179); //calcula angulo a partir de lectura
  servito.write(angulo); //ordena a servito ponerse en la posición angulo
  delay(20); //para darle tiempo al servo a ponerse en la posición angulo
}
```

PRÁCTICA 20. Control de un servo 2

Materiales. 1 servo, 1 condensador electrolítico de 100 μ F, 2 pulsadores y 2 resistencias de 10 k Ω (marrón-negro-naranja).

Explicación. Pretendemos controlar un servo a través de dos pulsadores. Cuando pulsemos el primero pero no el segundo, el servo irá a la posición de la izquierda (0°). Cuando pulsemos el segundo pero no el primero, el servo irá a la posición de la derecha (180°). En caso contrario a los dos anteriores, el servo irá a la posición central (90°).

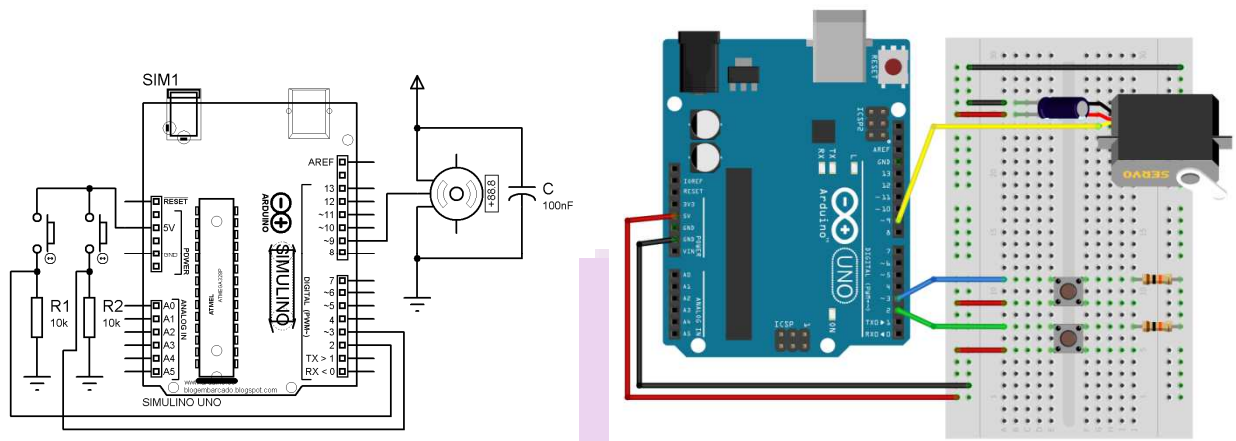


Si pulso el pulsador que va al pin 2 y no pulso el pulsador que va al pin 3, entonces el servo se pondrá en la posición izquierda, esto es, de 0°.

Si pulso el pulsador que va al pin 3 y no pulso el pulsador que va al pin 2, entonces el servo se pondrá en la posición derecha, esto es, de 180°.

En cualquier otro caso, el servo se pondrá en la posición central, esto es, de 90°.

Esquemas eléctricos



Sketch

```
#include <Servo.h>           //Carga la librería para trabajar con servos
const int pinIzdo = 2; //el pulsador que mueve a 0° va al pin de lectura 2
const int pinDcho = 3; //el pulsador que mueve a 180° va la pin de lectura 3
Servo servito;              //Declaramos un objeto tipo Servo llamado servito
int movIzdo=LOW; //Valdrá HIGH cuando apretemos el pulsador que mueve a 0°
int movDcho=LOW; //Valdrá HIGH cuando apretemos el pulsador que mueve a 180°

void setup() {
  servito.attach(9); //servito está conectado al pin 9
  pinMode(pinIzdo, INPUT);
  pinMode(pinDcho, INPUT);
}

void loop() {
  movIzdo=digitalRead(pinIzdo);
  movDcho=digitalRead(pinDcho);
  if ((movIzdo==HIGH)&&(movDcho==LOW)) {
    servito.write(0);
  }
  else if ((movIzdo==LOW)&&(movDcho==HIGH)) {
    servito.write(180);
  }
  else {
    servito.write(90);
  }
  delay(20);
}
```


PRÁCTICA 21. Control de un motor de cc (Puente H L298N)

Materiales. 1 motor de cc de 6 o 9 V, 1 puente H L298N, 1 potenciómetro de 10 k Ω , 1 pulsador, 1 resistencia de 10 k Ω (marrón-negro-naranja) y 1 batería de 6 o 9 V.

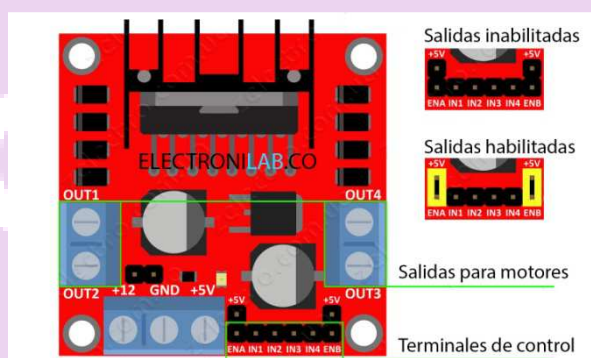
Explicación. Pretendemos controlar un motor de cc mediante un pulsador y un potenciómetro. Según apretamos o no el pulsador, el motor girará en un sentido o en otro. Con el potenciómetro le daremos más o menos voltaje al motor.

En las prácticas de los servos hemos utilizado condensadores en paralelo para que la intensidad no variase demasiado; así evitábamos que se pudiera dañar el Arduino. Como los servos han funcionado en vacío (no han movido nada pesado), esta solución ha sido suficiente. Si los servos tuvieran que hacer más fuerza o fuesen servos más grandes la solución de los condensadores sería insuficiente. Lo mismo sucede con los motores de cc. Los motores de cc no pueden ser alimentados directamente por Arduino, pues necesitan más intensidad de la que Arduino les puede dar. Si conectamos directamente un motor de cc a Arduino corremos el peligro de dañar Arduino e incluso el ordenador.

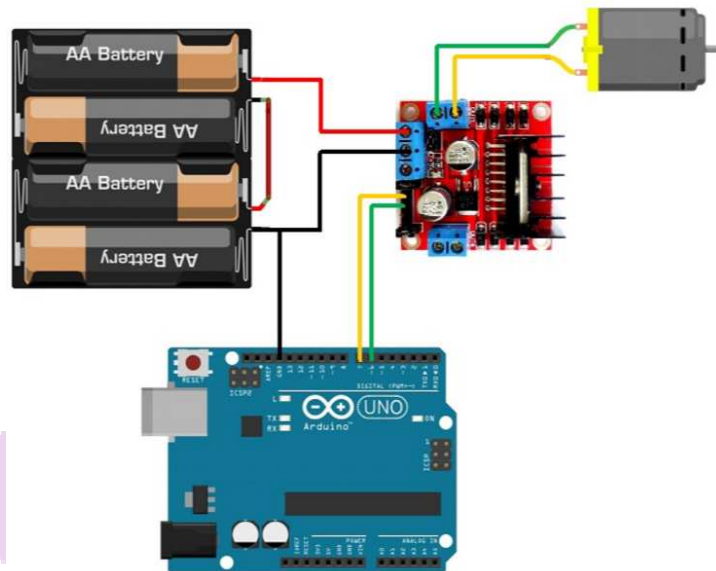
Para poder trabajar motores con Arduino utilizaremos el módulo L298N. Este módulo es alimentado externamente por una batería de entre 6 V y 12 V y permite controlar hasta dos motores de cc, de manera que podremos hacerlos girar en ambos sentidos. Utilizando los pines PWM~ de Arduino, podremos incluso controlar la velocidad de los dos motores.

Veamos el L298N. En la entrada donde pone +12 conectaremos el positivo de la batería. La entrada GND se pone a 0 V. Podemos conectar la entrada donde pone +5V a 5 V de Arduino, pero no es obligatorio. En las entradas OUT1 y OUT2 conectaremos los dos terminales del motor de cc. Las entradas IN1 e IN2 son las entradas de control e irán a dos pines PWM~; por ejemplo, IN1 al pin 10 e IN2 al pin 9. El módulo funciona así.

- Si IN1 está en HIGH, entonces OUT1 se pone al voltaje de la batería. Si IN1 está en LOW, entonces OUT1 se pone a 0 V.
- Si IN2 está en HIGH, entonces OUT2 se pone al voltaje de la batería. Si IN2 está en LOW, entonces OUT2 se pone a 0 V.



De esta manera, si IN1 está en HIGH e IN2 está en LOW, entonces el motor gira en un sentido. Si IN1 está en LOW e IN2 está en HIGH, entonces el motor gira en sentido contrario. Si ambos están en HIGH o ambos están en LOW, entonces el motor no gira. Como hemos conectado IN1 e IN2 a dos pines PWM~, podremos también controlar la velocidad del motor de cc.



El dibujo de arriba muestra cómo conectarlo. Pero **¡OJO!**, en el dibujo la entrada IN1 (amarilla) va al pin 7 y la entrada IN2 (verde) va al pin 6. Nosotros hemos dicho que la IN1 va al pin 10 e IN2 va al pin 9 para que ambas sean PWM~.

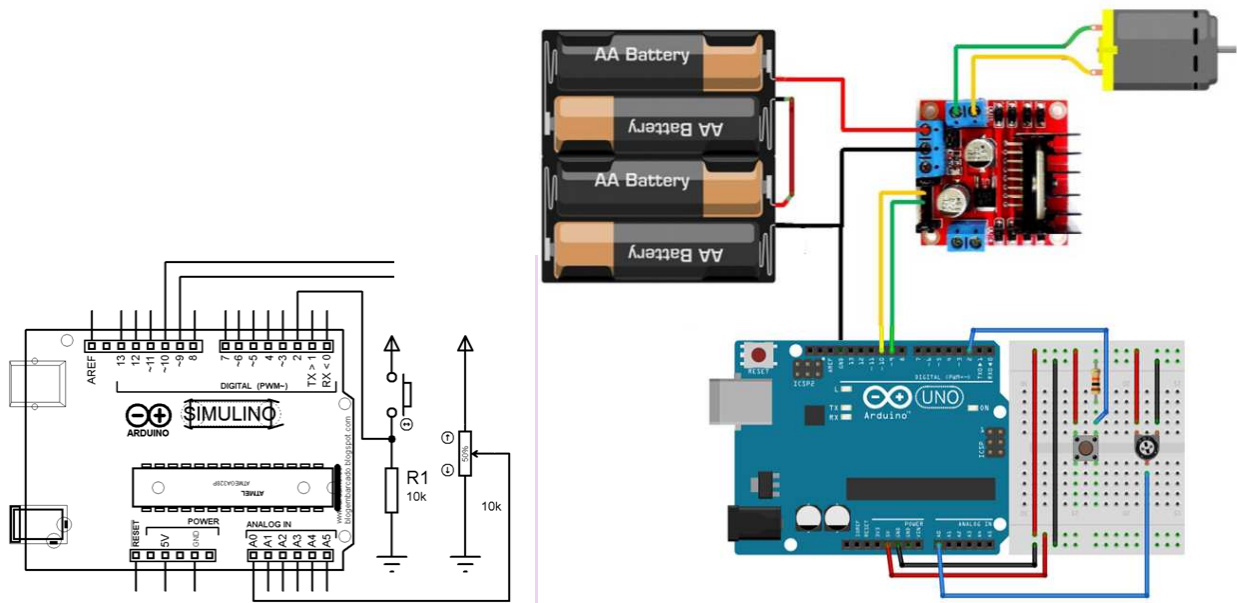
Para el resto de la práctica, conectaremos la pata central del potenciómetro al pin analógico A0. Declaramos una variable entera a la que llamaremos `vel` que valdrá entre 0 y 255, pues estos son los valores que podemos escribir en los pines 9 y 10 mediante `analogWrite`. La lectura del pin A0 va desde 0 hasta 1023; sin embargo, los valores que puede tomar `vel` van del 0 al 255, por lo que tendremos que escribir:

```
vel=analogRead(A0)/4;
```

El pulsador irá conectado al pin 2, que declaramos de entrada, y a la resistencia, de forma que si pulso el pulsador, el pin 2 se pone a 5 V y si no lo pulso, el pin 2 se pone a 0 V.

Tendremos que escribir un condicional de manera que si el pin 2 está en LOW, entonces en el pin 9 se escriba `vel` y en el pin 10 se escriba 0 para que el motor gire en un sentido. Por el contrario, si el pin 2 está en HIGH, entonces en pin 9 se escribirá 0 y en el pin 10 se escribirá `vel` para que el motor gire en el sentido contrario.

Esquemas eléctricos



Sketch

```

const int sentido = 2; //el pulsador va al pin de lectura 2
const int derA = 9;    //pata 7 del CI al pin PWM~ 9
const int derB = 10;   //pata 2 del CI al pin PWM~ 10
int vel;               //a mayor valor mayor velocidad (entre 0 y 255)

void setup() {
  pinMode(sentido, INPUT);
  pinMode(derA, OUTPUT);
  pinMode(derB, OUTPUT);
}

void loop() {
  vel=analogRead(A0)/4; //vel vale entre 0 (0/4) y 255 (1023/4)
  if (digitalRead(sentido)==LOW){
    analogWrite(derA,vel);
    analogWrite(derB,0);
  }
  else {
    analogWrite(derA,0);
    analogWrite(derB,vel);
  }
}

```

PRÁCTICA 21. Casa domótica (módulo Bluetooth HC-06)

Materiales. 2 leds rojos, 1 resistencia de 220 Ω (rojo-rojo-marrón), 1 servo motor, 1 condensador electrolítico de 100 μF y 1 módulo Bluetooth HC-06.

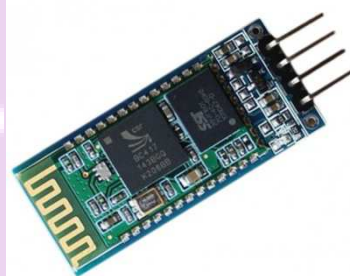
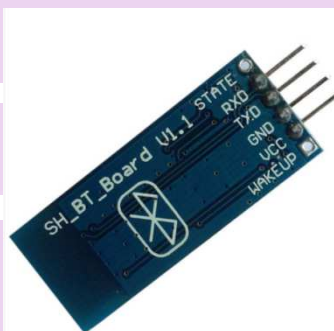
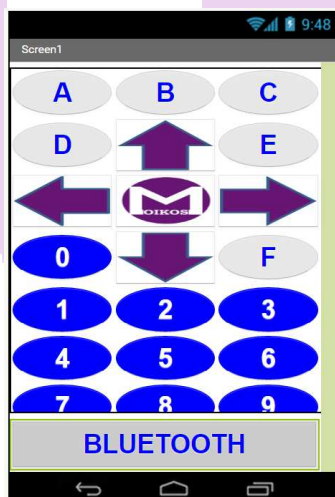
Explicación. Pretendemos controlar dos leds y un servo mediante el móvil gracias a una App que se puede descargar del blog y al módulo Bluetooth HC-06.

Uno de los leds tendrá su ánodo en el pin 13, de manera que solo controlaremos que se encienda o se apague. Hasta ahora no lo hemos dicho, pero podemos conectar el ánodo de un led al pin 13 y el cátodo directamente a GND, pues el pin 13 da menos de 5 V, por lo que no corremos el riesgo de perjudicar al led. Para encender este led presionaremos instantáneamente el botón D de la App. Para apagarlo una vez que esté encendido, volveremos a presionar instantáneamente el botón D de la App.

El segundo led tendrá su ánodo en el pin 11, que es PWM~, de manera que además de encenderlo o apagarlo podremos variar su intensidad. La intensidad la controlaremos con los botones flecha arriba \uparrow y flecha abajo \downarrow .

El servomotor ira al pin 7. Si presionamos instantáneamente el botón flecha izquierda \leftarrow , el motor se pondrá en posición izquierda, esto es, 180°. Si presionamos instantáneamente el botón flecha derecha \rightarrow , el motor se pondrá en posición derecha, esto es, 0°. Si presionamos instantáneamente el botón logo OIKOS, el motor se pondrá en posición central, esto es, 90°.

Veamos cómo funciona el **módulo Bluetooth HC-06**. Este módulo tiene cuatro patas. La pata RX debe ir al pin TX de Arduino (pin digital 1). La pata TX debe ir al pin RX de Arduino (pin digital 0). Así, estas patas van intercambiadas con el Arduino para poder establecer la conexión. La pata GND va a 0 V y la pata VCC va a 5 V.



Es importante cargar el sketch en Arduino antes de conectar las patas RX y TX del módulo HC-06, pues si cargamos el sketch con estas patas conectadas nos dará error. Así, primero cargaremos el sketch en Arduino y luego conectaremos el módulo HC-06 a Arduino. Si tenemos que modificar el sketch y volverlo a cargar en Arduino, antes de cargarlo deberemos desconectar el módulo de Arduino.

Veamos cómo funciona la **App**. Es una App sencillísima. Cada vez que apretemos un botón de la App, ésta mandará, vía Bluetooth, un carácter al serial. La equivalencia entre botones de la App y caracteres es la siguiente:

botón	A	B	C	D	E	F	↑	←	logo	→	↓	0	1	2	3	4	5	6	7	8	9
carácter	A	B	C	D	E	F	a	b	c	d	e	0	1	2	3	4	5	6	7	8	9

Una vez conectado el módulo HC-06 a Arduino pero antes de que móvil y módulo estén conectados vía Bluetooth, el led interior que lleva el módulo estará parpadeando. Para conectar el móvil al módulo HC-06 mediante Bluetooth, dentro de la App, presionamos en Bluetooth. Cuando móvil y módulo estén conectados vía Bluetooth, el led del módulo lucirá sin parpadear.

El conjunto debe funcionar de la siguiente manera:

- Inicialmente, el led del pin 13 estará apagado. Si en la App pulsamos D y el led del pin 13 está apagado, entonces dicho led se encenderá. Si en la App pulsamos D y el led del pin 13 está encendido, entonces dicho led se apagará.
- Inicialmente, el led del pin 11 estará apagado. Cada vez que en la App pulsemos el botón ↑, el led del pin 11 lucirá un poco más. Cada vez que en la App pulsemos el botón ↓ el led del pin 11 lucirá un poco menos.
- Inicialmente, el servo estará en la posición de 90°. Si en la App pulsamos el botón ←, entonces el servo se pondrá en la posición 180°. Si en la App pulsamos el botón →, entonces el servo se pondrá en la posición 0°.

Damos las siguientes pistas para el **sketch**.

Declararemos una variable tipo carácter o entera, que llamaremos `lectura`. En el void `loop()` esta variable recogerá el carácter que la App mande al serial mediante el grupo de sentencias:

```
if (Serial.available()>0){ //Si se ha mandado algo al serial...
  lectura=Serial.read(); //lectura leerá lo que se ha mandado
}
else{ //Si no se ha mandado algo al serial...
  lectura=' '; //lectura vale el carácter espacio en blanco
}
```

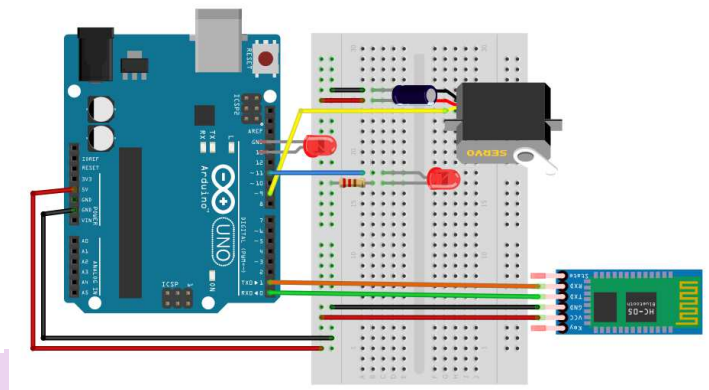
Declararemos una variable de tipo entero, que llamaremos `brillo`, y que inicialmente valdrá 0. En el void `loop()` haremos, mediante la función `analogWrite`, que el led de intensidad variable luzca con el valor de `brillo`. Si en la App apretamos el botón ↑ y el valor de `brillo` es menor o igual a 225, entonces incrementaremos el valor de `brillo` en 25. Si en la App apretamos el botón ↓ y el valor de `brillo` es mayor o igual a 25, entonces decrementaremos el valor de `brillo` en 25. Así los valores que tomará `brillo` estarán entre 0 y 250 y serán múltiplos de 25.

Declararemos una variable de tipo entero, que llamaremos `encendido`, y que inicialmente valdrá 0. Esta variable hará que si en la App pulsamos el botón D y `encendido` vale 0, entonces el led del pin 13 se encenderá y `encendido` pasará a valer 1. Si en la App pulsamos el botón D y `encendido` vale 1, entonces el led del pin 13 se apagará y `encendido` pasará a valer 0.

Declararemos un objeto tipo servo al que llamaremos `servito`. Recordar que antes habrá que cargar la librería `Servo.h` mediante: `#include <Servo.h>`. Este objeto irá al pin 7 de Arduino, por lo que en el `setup()` escribiremos: `servito.attach(7);`

Recordar inicializar el serial en el `setup()` mediante: `Serial.begin(9600);`

Esquema eléctrico



Sketch

```
#include <Servo.h> //Carga la librería para trabajar con servos
const int led1=13;
const int led2=11;
Servo servito;      //Declaramos un objeto tipo Servo llamado servito
char lectura;
int encendido=0; //el carácter G no se puede conseguir con la App
int brillo=0;

void setup() {
  Serial.begin(9600);
  servito.attach(7); //servito está conectado al pin 7
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  digitalWrite(led1, LOW);
  analogWrite(led2, 0);
  servito.write(90);
}

void loop() {
  analogWrite(led2,brillo);
  if (Serial.available()>0){
    lectura=Serial.read();
  }
  else{
    //mientras no introduzcamos otro caracter, ...
    lectura=' '; //lectura vale un espacio en blanco
  }
  if (lectura=='D'){
    if (encendido==0){
      digitalWrite(led1,HIGH);
    }
    if(encendido==1){
      digitalWrite(led1,LOW);
    }
    encendido=1-encendido;
  }
  if (lectura=='b'){
    servito.write(180);
  }
  if (lectura=='c'){
    servito.write(90);
  }
  if (lectura=='d'){
    servito.write(0);
  }
  if ((lectura=='a')&&(brillo<=225)){
    brillo=brillo+25;
  }
  if ((lectura=='e')&&(brillo>=25)){
    brillo=brillo-25;
  }
  delay(100);
}
```


PRÁCTICA 22. Coche controlado por Bluetooth I

Materiales para el conjunto de prácticas. 1 placa Arduino, 2 motores con reductora, 2 ruedas cuyo eje enganche bien con el eje de los motores, una rueda loca, un interruptor, un chasis para el coche, un puente H L298N, un módulo Bluetooth HC-06, un sensor de ultrasonidos HC-SR04, una placa board de 400 puntos, un led rojo, una buena pila alcalina de 9 V, cuatro cables de puente hembra-macho, cables de conexión, unas cuantas gomas elásticas.

Nota. En esta práctica 22 no usaremos el led rojo ni el módulo Bluetooth ni el sensor de ultrasonidos.

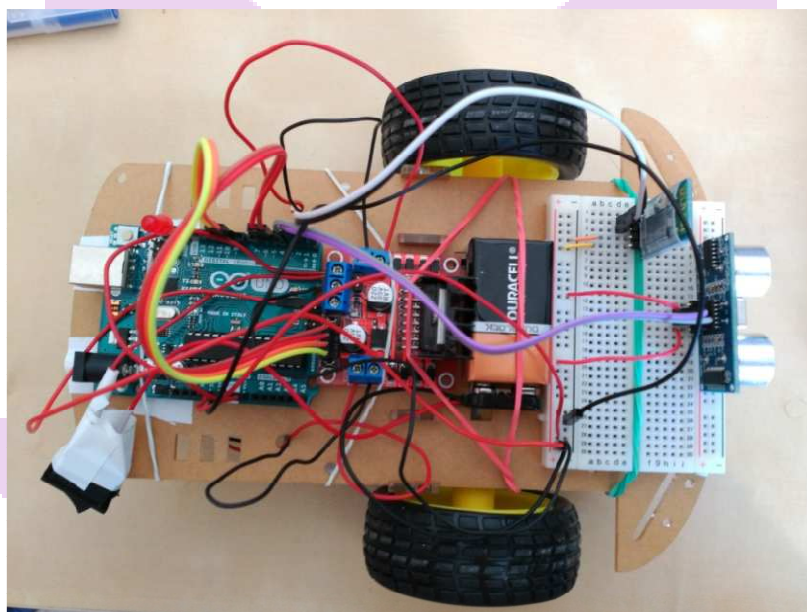
Explicación. En esta primera parte, pretendemos controlar un coche con dos motores mediante el monitor serial. La placa Arduino estará conectada al ordenador.

Lo primero será construir la **parte mecánica** del coche. Esto es, motores a las ruedas y al chasis y rueda loca. Una opción bastante económica es comprar en Amazon el siguiente kit:

[SODIAL\(R\) Kit Chasis WST inteligente de motores Robot Car velocidad Encoder Caja de batería](#)

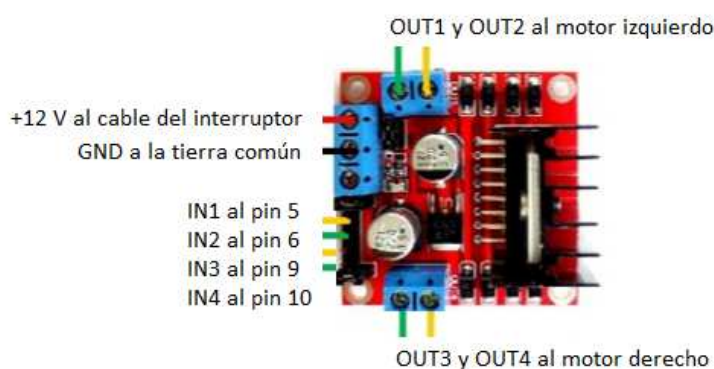
Por menos de 10 € nos trae dos motores con reductora, el chasis, las dos ruedas que van a los motores, una rueda loca, un interruptor y los tornillos y tuercas necesarios para montar la parte mecánica del coche.

Construida la parte mecánica del coche continuamos, vamos con la **posición de cada componente**. Fijaremos los componentes al chasis de forma sencilla; mediante gomas elásticas y/o cinta aislante. En la parte delantera estará la placa board, más atrás la pila, más atrás el puente H y finalmente la placa Arduino. El puente H se colocará de forma que OUT1 y OUT2 estén a la izquierda y OUT3 y OUT4 estén a la derecha; así, las aletas del puente H tocarán con la pila y las entradas de +12, GND, +5V, IN1, IN2, IN3 e IN4 del puente H tocarán con Arduino.



Vamos con las **conexiones de la pila**. Como pila, recomiendo una pila Duracell Duralock de 9 V. Otras pilas me han dado problemas. El positivo de la pila irá al interruptor. Del interruptor saldrá otro cable que irá al +12V del puente H. El negativo de la pila irá a la tierra común en la placa board. Es importante que cuando esté todo montado todas las tierras coincidan; así todos los GNDs tanto del Arduino, como de la pila, del puente H, del módulo Bluetooth y del sensor de ultrasonidos deben ser la misma tierra en la placa board.

Vamos con las **conexiones del puente H**. Los dos terminales del motor de la derecha se conectarán a OUT3 y OUT4 del puente H, que estarán también a la derecha; de momento no importa dónde conectemos cada uno de los dos cables, pues si fuese necesario, los intercambiaremos después. Los dos terminales del motor de la izquierda se conectarán a OUT1 y OUT2 del puente H, que estarán también a la izquierda; de momento da igual cómo conectemos cada uno de los dos cables, pues si fuese necesario, los intercambiaremos después. El +12V del puente H irá al interruptor, de manera que cuando el interruptor esté cerrado le lleguen los 9 V de la pila y cuando esté abierto no. El GND del puente H irá a la tierra común de la placa board. Para los pines IN1 a IN4 utilizaremos los cuatro cables hembra-macho que tenemos. El IN1 del puente H irá al pin 5 de Arduino. El IN2 del puente H irá al pin 6 de Arduino. El IN3 del puente H irá al pin 9 de Arduino. El IN4 del puente H irá al pin 10 de Arduino. Notar que los pines 5, 6, 9 y 10 son todos PWM~; gracias a esto podremos controlar la velocidad del coche más adelante.



Vamos con las **conexiones de Arduino**. Además de las conexiones que ya hemos hecho en los pines 5, 6, 9 y 10, no debemos olvidar conectar el GND de Arduino a la tierra común en la placa board. También debemos conectar el 5V de Arduino a la placa board; lo llamaremos el 5 V común.

Vamos a hacer un **sketch previo** para asegurarnos de que los cables de los dos motores a los puertos OUT del puente H están conectados correctamente. Vamos a poner el pin 5 en HIGH y el pin 6 en LOW. Si los cables del motor izquierdo están correctamente conectados en OUT1 y OUT2, la rueda izquierda debe girar de forma que el coche avance. En caso de que no sea así, intercambiaremos los cables del motor izquierdo en los puertos OUT1 y OUT2. Vamos a poner el pin 9 en HIGH y el pin 10 en LOW. Si los cables del motor derecho están correctamente conectados en OUT3 y OUT4, la rueda derecha debe girar de forma que el coche avance. En caso de que no sea así, intercambiaremos los cables del motor derecho en los puertos OUT3 y OUT4. Hasta aquí, hemos asegurado que si pin 5 está en HIGH y pin 6 está en LOW, entonces la rueda izquierda gira hacia adelante. También, si pin 9 está en HIGH y pin 10 está en LOW, entonces la rueda derecha gira hacia adelante.

Por otro lado, para que el coche avance, tanto la rueda izquierda como la derecha deben girar hacia adelante. Para que el coche gire a la izquierda, la rueda izquierda debe estar parada y la rueda derecha debe girar adelante. Para que el coche gire a la derecha, la rueda izquierda debe girar adelante y la rueda derecha debe parar. Para que el coche retroceda, ambas ruedas deben girar hacia atrás. Así, obtenemos la siguiente tabla.

	Motor izquierdo	Motor derecho	izqA(5) – izqB(6)	derA(9) – derB(10)
Avance	adelante	adelante	HIGH – LOW	HIGH – LOW
Giro izquierda	paro	adelante	LOW – LOW	HIGH – LOW
Paro	paro	paro	LOW – LOW	LOW – LOW
Giro derecha	adelante	paro	HIGH – LOW	LOW – LOW
Retroceso	atrás	atrás	LOW – HIGH	LOW – HIGH

Ahora veremos **cómo funcionará el coche**. Los caracteres válidos que podemos introducir para hacer funcionar el coche serán: 'a', 'b', 'c', 'd' y 'e'. Cuando en el serial introduzcamos el carácter:

- 'a' el coche avanzará y seguirá haciéndolo hasta que introduzcamos otro carácter válido.
- 'b' el coche girará a la izquierda y seguirá haciéndolo hasta que introduzcamos otro carácter válido.
- 'c' el coche parará y seguirá haciéndolo hasta que introduzcamos otro carácter válido.
- 'd' el coche girará a la derecha y seguirá haciéndolo hasta que introduzcamos otro carácter válido.
- 'e' el coche retrocederá y seguirá haciéndolo hasta que introduzcamos otro carácter válido.

Ya tenemos todo lo necesario para escribir nuestro sketch. La variable entera `estado` será la encargada de almacenar cómo debe rodar el coche por lo que podrá tomar como valores los caracteres 'a', 'b', 'c', 'd' y 'e'; bueno más bien, por tratarse de una variable entera, el código ASCII de estos caracteres. La cuestión es que si introducimos en el serial el carácter 'a', entonces `estado` almacenará el carácter 'a', pero además, `estado` debe seguir almacenando el carácter 'a' en todos los loops siguientes mientras no introduzcamos un nuevo carácter válido en el serial. Para conseguir esto, en el void loop() debemos escribir:

```
if(Serial.available()>0){ //si hay algo en el serial, lo lee...
    estado = Serial.read(); //y lo almacena en estado
} //de esta manera, mientras no escribamos nada nuevo en el serial...
//el valor de la variable estado no cambia
```

Intenta escribir tú mismo el sketch. En la siguiente página está la solución.



OIKOS

Sketch

```
const int derA = 9;           //IN3 del puente H al pin PWM~ 9
const int derB = 10;          //IN4 del puente H al pin PWM~ 10
const int izqA = 5;           //IN1 del puente H al pin PWM~ 5
const int izqB = 6;           //IN2 del puente H al pin PWM~ 6
const int vel = 100;          //este valor debe valer entre 0 y 255. Con 100 la velo-
                                cidad va bien
int estado='c';               //inicialmente el estado es parado

void setup() {
  Serial.begin(9600);
  pinMode(derA, OUTPUT);
  pinMode(derB, OUTPUT);
  pinMode(izqA, OUTPUT);
  pinMode(izqB, OUTPUT);
}

void loop() {

  if(Serial.available()>0){    //si hay algo en el serial, lo lee...
    estado = Serial.read();    //y lo almacena en estado
  } //de esta manera, mientras no escribamos nada nuevo en el serial...
    //el valor de estado no cambia

  //Hacia adelante
  if (estado=='a'){
    Serial.println("Adelante");
    analogWrite(izqA,vel);
    analogWrite(izqB,0);
    analogWrite(derA,vel);
    analogWrite(derB,0);
  }

  //Giro a la izquierda
  if (estado=='b'){
    Serial.println("Giro izquierda");
    analogWrite(izqA,0);
    analogWrite(izqB,0);
    analogWrite(derA,vel);
    analogWrite(derB,0);
  }

  //Paro
  if (estado=='c'){
    Serial.println("Paro");
    analogWrite(izqA,0);
    analogWrite(izqB,0);
    analogWrite(derA,0);
    analogWrite(derB,0);
  }

  //Giro a la derecha
  if (estado=='d'){
    Serial.println("Giro derecha");
    analogWrite(izqA,vel);
    analogWrite(izqB,0);
    analogWrite(derA,0);
    analogWrite(derB,0);
  }

  //Marcha atrás
  if (estado=='e'){
    Serial.println("Marcha atras");
    analogWrite(izqA,0);
    analogWrite(izqB,vel);
    analogWrite(derA,0);
    analogWrite(derB,vel);
  }
}
```

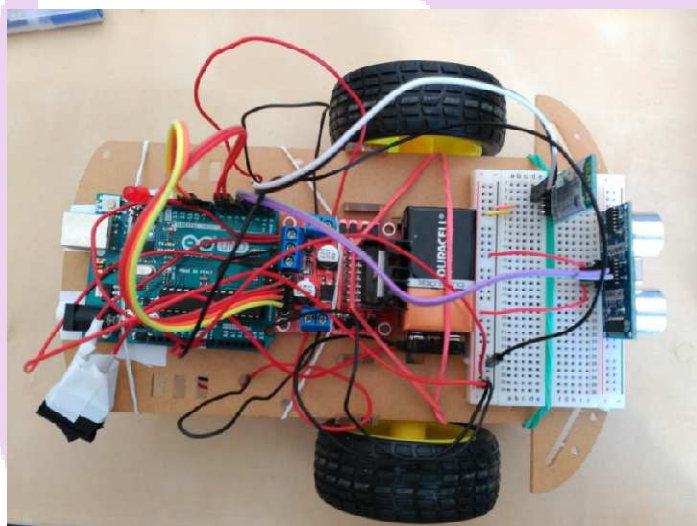
PRÁCTICA 23. Coche controlado por Bluetooth II

Materiales para el conjunto de prácticas. 1 placa Arduino, 2 motores con reductora, 2 ruedas cuyo eje enganche bien con el eje de los motores, una rueda loca, un interruptor, un chasis para el coche, un puente H L298N, un módulo Bluetooth HC-06, un sensor de ultrasonidos HC-SR04, una placa board de 400 puntos, un led rojo, una buena pila alcalina de 9 V, cuatro cables de puente hembra-macho, cables de conexión, unas cuantas gomas elásticas.

Nota. En esta práctica no usaremos el módulo Bluetooth.

Explicación. En esta segunda parte, pretendemos la siguiente ampliación: que el coche tenga una marcha automática, esto es, que pueda evitar los obstáculos que se encuentre a su paso. Para ello, necesitaremos añadir un sensor de ultrasonidos HC-SR04, que ya hemos estudiado en prácticas anteriores. Seguimos controlando el coche mediante el monitor serial, de manera que la placa Arduino estará conectada al ordenador.

La posición del **sensor de ultrasonidos** viene dada en la imagen. El salida Echo del sensor irá al pin 2 de Arduino y la salida Trigger del sensor irá al pin 3 de Arduino. El Vcc del sensor irá al 5V común de la placa board y el GND del sensor irá a la tierra común en la placa board.



El ánodo del **led rojo** irá al pin 13 y el cátodo al GND de Arduino que está al lado del pin 13 (recordar que si conectamos un led al pin 13 no es necesaria una resistencia de protección).

Queremos que el coche funcione de forma automática cuando la variable `estado` tome el valor 'A' (recordar que 'A' y 'a' son distintas). Además, deberemos medir la distancia del sensor al obstáculo utilizando las variables enteras `duracion` y `distancia` como ya vimos en prácticas anteriores. La variable `distancia` almacenará la distancia en cm entre el sensor y el obstáculo.

Si `distancia` es menor o igual a 15 y mayor o igual a 2, entonces:

- Se encenderá el led rojo.
- El coche se parará el coche durante 0,2 s.
- El coche retrocederá durante 0,5 s.
- El coche girará a la derecha durante 1,1 s.
- Se apagará el led rojo.

En caso contrario, el coche avanzará hacia adelante.

Ya puedes intentar hacer el nuevo sketch a partir del sketch de la práctica anterior.

Sketch (se resaltan las nuevas líneas de código)

```
const int derA = 9;           //IN3 del puente H al pin PWM~ 9
const int derB = 10;          //IN4 del puente H al pin PWM~ 10
const int izqA = 5;           //IN1 del puente H al pin PWM~ 5
const int izqB = 6;           //IN2 del puente H al pin PWM~ 6
const int pecho = 2;          //define el pin 2 como (pecho) para el ultrasonido
const int ptrig = 3;          //define el pin 3 como (ptrig) para el ultrasonido
const int vel = 100;          //entre 0 y 255. Con 100 va bien.
int estado='c';               //inicialmente el estado es parado
int duracion, distancia;      //para calcular distancia del sensor al obstáculo
```

```
void setup() {
  Serial.begin(9600);
  pinMode(derA, OUTPUT);
  pinMode(derB, OUTPUT);
  pinMode(izqA, OUTPUT);
  pinMode(izqB, OUTPUT);
  pinMode(pecho, INPUT);
  pinMode(ptrig, OUTPUT);
  pinMode(13,OUTPUT);          //el led rojo al pin 13
}

void loop() {
  if(Serial.available()>0){    //si hay algo en el serial, lo lee...
    estado = Serial.read();    //y lo almacena en estado
  } //de esta manera, mientras no escribamos nada nuevo en el serial...
    //el valor de estado no cambia

  //Hacia adelante
  if (estado=='a'){
    Serial.println("Adelante");
    analogWrite(izqA,vel);
    analogWrite(izqB,0);
    analogWrite(derA,vel);
    analogWrite(derB,0);
  }

  //Giro a la izquierda
  if (estado=='b'){
    Serial.println("Giro izquierda");
    analogWrite(izqA,0);
    analogWrite(izqB,0);
    analogWrite(derA,vel);
    analogWrite(derB,0);
  }

  //Paro
  if (estado=='c'){
    Serial.println("Paro");
    analogWrite(izqA,0);
    analogWrite(izqB,0);
    analogWrite(derA,0);
    analogWrite(derB,0);
  }

  //Giro a la derecha
  if (estado=='d'){
    Serial.println("Giro derecha");
    analogWrite(izqA,vel);
    analogWrite(izqB,0);
    analogWrite(derA,0);
    analogWrite(derB,0);
  }
}
```

```
//Marcha atrás
if (estado=='e'){
  Serial.println("Marcha atras");
  analogWrite(izqA,0);
  analogWrite(izqB,vel);
  analogWrite(derA,0);
  analogWrite(derB,vel);
}
```

```
//Automático
if (estado == 'A'){ //se mueve evitando obstáculos
  Serial.println("automático");
  digitalWrite(ptrig, HIGH); //genera el pulso de trigger por 10us
  delay(0.01);
  digitalWrite(ptrig, LOW);
  duracion = pulseIn(pecho, HIGH); //lee el tiempo del Echo
  distancia = (duracion/2)*0.034; //calcula la distancia en cm
  delay(10);
  if (distancia <= 15 && distancia >=2){ //si la distancia es menor de 15 cm
    digitalWrite(13,HIGH); //enciende LED
    analogWrite(derB, 0); //parar durante 200 ms
    analogWrite(izqB, 0);
    analogWrite(derA, 0);
    analogWrite(izqA, 0);
    delay (200);
    analogWrite(derB, vel); //atrás durante 500 ms
    analogWrite(izqB, vel);
    delay(500);
    analogWrite(derB, 0); //girar derecha durante 1100 ms
    analogWrite(izqB, 0);
    analogWrite(derA, 0);
    analogWrite(izqA, vel);
    delay(1100);
    digitalWrite(13,LOW);
  }
  else{ //si no hay obstáculos, hacia adelante
    analogWrite(derB, 0);
    analogWrite(izqB, 0);
    analogWrite(derA, vel);
    analogWrite(izqA, vel);
  }
}
```

```
}
```



OIKOS

PRÁCTICA 24. Coche controlado por Bluetooth III

Materiales para el conjunto de prácticas. 1 placa Arduino, 2 motores con reductora, 2 ruedas cuyo eje enganche bien con el eje de los motores, una rueda loca, un interruptor, un chasis para el coche, un puente H L298N, un módulo Bluetooth HC-06, un sensor de ultrasonidos HC-SR04, una placa board de 400 puntos, un led rojo, una buena pila alcalina de 9 V, cuatro cables de puente hembra-macho, cables de conexión, unas cuantas gomas elásticas.

Nota. En esta práctica no usaremos el módulo Bluetooth.

Explicación. En esta tercera parte, pretendemos la siguiente ampliación: que el coche tenga 10 velocidades, de la 0, que será parado a la 9 que será la máxima. Esta ampliación será posible gracias a que los pines 5, 6, 9 y 10 que hemos usado son PWM. Seguimos controlando el coche mediante el monitor serial, de manera que la placa Arduino estará conectada al ordenador. En esta práctica no hay que añadir ningún componente o conexión, pues toda la ampliación recae en el sketch.

Para que el coche vaya a la velocidad 0, en el serial pondremos '0'. Para que el coche vaya a la velocidad 1, en el serial pondremos '1'. Y seguimos así hasta la velocidad 9, donde pondremos '9'.

La velocidad se controla mediante la variable `vel`, cuyo valor valdrá entre 0 y 255 y que ya ha sido declarada como una constante entera en las prácticas anteriores. Así, ahora será declarada como variable entera de valor inicial 113. El porqué del 113 es porque de las 10 velocidades es la que más se acerca al valor 100 que ha tomado `vel` en las anteriores prácticas.

Es claro que a la velocidad 0 le corresponde un valor de `vel` de 0 y a la velocidad 9 le corresponde un valor de `vel` de 255. Haciendo pruebas con distintos valores de `vel`, he comprobado que para que el coche pueda andar es necesario como mínimo un valor de `vel` de 60. Así, a la velocidad 1 le corresponderá un valor de `vel` de 60. El resto de velocidades se calculará de manera proporcional, teniendo en cuenta que a la velocidad 1 le corresponde un `vel` de 60 y a la velocidad 9 un `vel` de 255. Por tanto, para el cálculo de `vel` las velocidades de la 1 a la 9 utilizaremos la función `map`. Es fácil comprobar que a la velocidad 4 le corresponde un `vel` de 113; así que inicialmente la velocidad con la que parte el coche es la velocidad 4.

Un apunte sobre el código ASCII que puede ser útil para programar. El código ASCII del carácter '0' no es 0 sino 48. El del carácter '1' es 49, el del carácter '2' es 50 y así hasta el carácter '9' cuyo valor es 57. Esto significa que cuando en el serial escribo el carácter '0', la variable lectura no toma el valor 0 sino el 48. Lo anterior debo tenerlo en cuenta a la hora de escribir la sentencia con la función `map` que me dará el valor de `vel` para las velocidades de la 1 a la 9.

No hace falta decir más para hacer las ampliaciones al sketch de la práctica anterior, de manera que ya podemos lanzarnos a intentarlo. Sin embargo, puede que nos encontremos con el siguiente problema a la hora de programar. Puesto que para decirle a Arduino la velocidad debemos escribir en el serial un carácter entre '0' y '9', dicho carácter va a ser almacenado en la variable `estado`. El problema es que si `estado` pasa a valer entre '0' y '9', entonces `estado` deja de valer alguno de los caracteres 'a', 'b', 'c', 'd' o 'e' y, tal como está el sketch, perdemos la información sobre si el coche estaba yendo hacia adelante, girando a la derecha, parado, girando a la izquierda o yendo hacia atrás. Si el lector quiere solucionarlo por sí mismo, que no continúe leyendo y vaya directo a escribir su sketch.

Para solucionar este problema, he declarado una variable entera que he llamado `estado_anterior` y que inicialmente también vale 'c'. Al comienzo de cada loop, si hay un nuevo carácter escrito en el serial, antes de que la variable `estado` tome el valor del nuevo carácter, la variable `estado_anterior` tomará el valor de la variable `estado`. En el caso de que ese nuevo carácter esté entre '0' y '9', programaremos para que `vel` tome el valor que le corresponde y, después, asignaremos a la variable `estado` el valor de la variable `estado_anterior`, para que el coche siga haciendo el tipo de movimiento que estaba haciendo antes introducir el carácter numérico entre '0' y '9'.

Sketch (se resaltan las nuevas líneas de código)

```
const int derA = 9;           //IN3 del puente H al pin PWM~ 9
const int derB = 10;          //IN4 del puente H al pin PWM~ 10
const int izqA = 5;           //IN1 del puente H al pin PWM~ 5
const int izqB = 6;           //IN2 del puente H al pin PWM~ 6
const int pecho = 2;          //define el pin 2 como (pecho) para el ultrasonido
const int ptrig = 3;          //define el pin 3 como (ptrig) para el ultrasonido
int vel = 113;                //entre 0 y 255. 113 corresponde a la marcha 4 de 0 a 9.
int estado='c';              //inicialmente el estado es parado
int estado_anterior='c';     //esta variable nos servirá para controlar la velocidad
int duracion, distancia;     //para calcular distancia

void setup() {
  Serial.begin(9600);
  pinMode(derA, OUTPUT);
  pinMode(derB, OUTPUT);
  pinMode(izqA, OUTPUT);
  pinMode(izqB, OUTPUT);
  pinMode(pecho, INPUT);
  pinMode(ptrig, OUTPUT);
  pinMode(13,OUTPUT);
}

void loop() {
  if(Serial.available()>0){ //si hay algo en el serial, lo lee...
    estado_anterior=estado;
    estado = Serial.read(); //y lo almacena en estado
  }

  //Hacia adelante
  if (estado=='a'){
    Serial.println("Adelante");
    analogWrite(izqA,vel);
    analogWrite(izqB,0);
    analogWrite(derA,vel);
    analogWrite(derB,0);
  }

  //Giro a la izquierda
  if (estado=='b'){
    Serial.println("Giro izquierda");
    analogWrite(izqA,0);
    analogWrite(izqB,0);
    analogWrite(derA,vel);
    analogWrite(derB,0);
  }

  //Paro
  if (estado=='c'){
    Serial.println("Paro");
    analogWrite(izqA,0);
    analogWrite(izqB,0);
    analogWrite(derA,0);
    analogWrite(derB,0);
  }

  //Giro a la derecha
  if (estado=='d'){
    Serial.println("Giro derecha");
    analogWrite(izqA,vel);
    analogWrite(izqB,0);
    analogWrite(derA,0);
    analogWrite(derB,0);
  }
}
```

```
//Marcha atrás
if (estado=='e'){
  Serial.println("Marcha atras");
  analogWrite(izqA,0);
  analogWrite(izqB,vel);
  analogWrite(derA,0);
  analogWrite(derB,vel);
}
```

```
//Velocidad
//Importante: el código ASCII del caracter '0' es 48, el del '1' es 49...
//el del '2' es 50, ... el del '9' es 57
if ((estado>='0') && (estado<='9')){
  estado=estado-48; //pasamos el valor ASCII al valor numérico
  if (estado==0){
    vel=0;
  }
  else{
    vel=map(estado,1,9,60,255);
  }
  Serial.println(vel);
  estado=estado_anterior;//devolvemos a estado su valor antes de pulsar el...
} //... el botón numérico
```

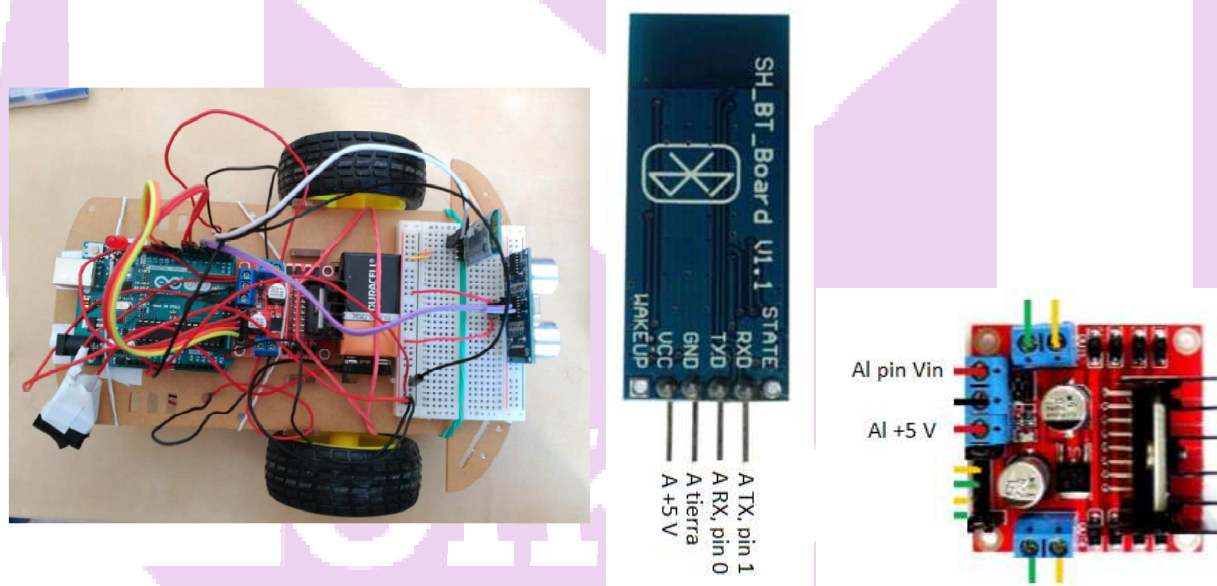
```
//Automático
if (estado == 'A'){ //botón A, se mueve evitando obstáculos
  Serial.println("automático");
  digitalWrite(ptrig, HIGH); //genera el pulso de trigger por 10us
  delay(0.01);
  digitalWrite(ptrig, LOW);
  duracion = pulseIn(pecho, HIGH); //lee el tiempo del Echo
  distancia = (duracion/2)*0.034; //calcula la distancia en cm
  delay(10);
  if (distancia <= 15 && distancia >=2){ //si la distancia es menor de 15 cm
    digitalWrite(13,HIGH); //enciende LED
    analogWrite(derB, 0); //parar los motores por 200 ms
    analogWrite(izqB, 0);
    analogWrite(derA, 0);
    analogWrite(izqA, 0);
    delay(200);
    analogWrite(derB, vel); //atrás durante 500 ms
    analogWrite(izqB, vel);
    delay(500);
    analogWrite(derB, 0); //girar derecha durante 1100 ms
    analogWrite(izqB, 0);
    analogWrite(derA, 0);
    analogWrite(izqA, vel);
    delay(1100);
    digitalWrite(13,LOW);
  }
  else{ //si no hay obstaculos hacia adelante
    analogWrite(derB, 0);
    analogWrite(izqB, 0);
    analogWrite(derA, vel);
    analogWrite(izqA, vel);
  }
}
}
```


PRÁCTICA 25. Coche controlado por Bluetooth IV

Materiales para el conjunto de prácticas. 1 placa Arduino, 2 motores con reductora, 2 ruedas cuyo eje enganche bien con el eje de los motores, una rueda loca, un interruptor, un chasis para el coche, un puente H L298N, un módulo Bluetooth HC-06, un sensor de ultrasonidos HC-SR04, una placa board de 400 puntos, un led rojo, una buena pila alcalina de 9 V, cuatro cables de puente hembra-macho, cables de conexión, unas cuantas gomas elásticas.

Explicación. En esta cuarta parte, pretendemos controlar el coche con el móvil, vía Bluetooth con la App Oikos_Car, que os pasaré por correo electrónico. Para hacer esta práctica no hay que tocar una sola línea de código en el sketch respecto de la práctica anterior. En esta práctica ya no vamos a controlar el coche mediante el monitor serial, sino por Bluetooth, por lo que necesitaremos un módulo Bluetooth HC-06. Además, como el coche se va a desplazar, Arduino no se va a poder alimentar por el puerto USB del ordenador, sino que se alimentará con la misma pila que alimenta a los dos motores.

El **módulo Bluetooth** va conectado a la placa board, como se ve en la figura. La pata +5 V va a 5 V común de la placa board. La pata GND va a la tierra común de la placa board. La pata TX va al pin RX de Arduino, esto es, al pin 0. La pata RX va al pin TX de Arduino, esto es, al pin 1. Así, las conexiones entre módulo y Arduino van intercambiadas. Recordamos que mientras las patas TX y RX del módulo estén conectadas a los pines 0 y 1 de Arduino no podemos descargar el sketch en Arduino, pues nos dará error. Por tanto, cada vez que queramos cargar un sketch en Arduino debemos desconectar los pines 0 y 1 de Arduino, después cargamos el sketch en Arduino y, finalmente, volvemos a conectar estos dos pines al módulo Bluetooth.



Veamos la nueva conexión que debemos añadir en **Arduino**. Además de los pines 0 y 1 ya vistos, necesitamos alimentar Arduino con la pila. Recordemos que el positivo de la pila iba a un interruptor y del interruptor salía un cable al +12 V del puente H. Lo que debemos hacer es añadir un cable que vaya del +12 V del puente H al pin Vin de Arduino. Este pin se encuentra en la zona Power de Arduino, entre un pin GND y el pin analógico A0. De esta manera, cuando cerremos el interruptor, Arduino será alimentado por la pila.

En la figura de arriba vemos las dos conexiones que debemos añadir en el **puente H**. Además del cable que acabamos de añadir desde el +12 V del puente H al pin Vin de Arduino, vamos a añadir una conexión más. Conectaremos el +5 V del puente H al 5V común de la placa board. Veamos por qué. A través de la pila, Arduino debe alimentarse a sí mismo, al led, al sensor de ultrasonidos y al módulo Bluetooth. Puede suceder que con la energía que recibe de la pila, Arduino no pueda alimentar a todo. Por suerte, la salida +5 V del puente H nos proporciona 5 V que vienen directamente de la pila; por ello, conectaremos el +5 V del

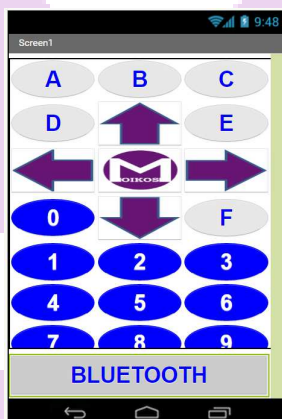
punto H con el 5V común de placa board, para asegurar que todos los componentes reciban su alimentación.

Repasemos las **conexiones**:

- Al **5V común de la placa board** están conectados: el 5V de Arduino, el +5 V del puente H, el +5 V del módulo Bluetooth, el Vcc del sensor de ultrasonidos.
- A la **tierra común de la placa board** están conectados: el negativo de la pila, el GND de Arduino, el GND del puente H, el GND del módulo Bluetooth y el GND del sensor de ultrasonidos.
- En **Arduino** están utilizados los pines: 5V, GND al lado de 5V, pin 0, pin 1, pin 2, pin 3, pin 5, pin 6, pin 9, pin 10, pin 13 y GND al lado del pin 13.
- En el **puente H** está todo conectado, saliendo del +12 V dos conexiones: una de la que recibirá la energía de la pila cuando el interruptor esté cerrado y otra que servirá para alimentar a Arduino cuando el interruptor esté cerrado. El GND va a la tierra común de la placa board. El +5 V va al 5V común de la placa board. OUT1 y OUT2 alimentan el motor izquierdo. OUT3 y OUT4 alimentan el motor derecho. IN1 va al pin 5. IN2 va al pin 6. IN3 va al pin 9. IN4 va al pin 10.
- En el **módulo Bluetooth** están las cuatro patas conectadas. +5V va al 5V común de la placa board. GND va a la tierra común de la placa board. RX va al pin TX. TX va al pin RX.
- En el **sensor de ultrasonidos** están conectadas las cuatro patas. Vcc va al 5V común de la placa board. GND va a la tierra común de la placa board. Echo va al pin 2. Trig va al pin 3.
- El ánodo del **led rojo** va al pin 13 y el cátodo del led rojo al GND que está al lado del pin 13.

Veamos cómo funciona la **App**. Es una App sencillísima. Cada vez que apretemos un botón de la App, ésta mandará, vía Bluetooth, un carácter al serial. La equivalencia entre botones de la App y caracteres es la siguiente:

botón	A	B	C	D	E	F	↑	←	logo	→	↓	0	1	2	3	4	5	6	7	8	9
carácter	A	B	C	D	E	F	a	b	c	d	e	0	1	2	3	4	5	6	7	8	9



Una vez conectado el módulo Bluetooth a Arduino pero antes de que móvil y módulo estén conectados vía Bluetooth, el led interior que lleva el módulo estará parpadeando. Para conectar el móvil al módulo HC-06 mediante Bluetooth, dentro de la App, presionamos en BLUETOOTH. Cuando móvil y módulo estén conectados vía Bluetooth, el led del módulo lucirá sin parpadear.

El funcionamiento será como sigue. Presionando los botones numéricos elegimos la velocidad del coche. Con las flechas le decimos si tiene que avanzar, girar a la izquierda, girar a la derecha o retroceder. Con el logo de Oikos Matematikón el coche para. Si pulsamos la A, el coche se pone en automático. No debemos pulsar los botones B, C, D, E, F, pues no tienen asignada ninguna tarea.

Ampliación. ¿Qué líneas añadirías al sketch para que si alguien pulsa cualquiera de los botones B, C, D, E o F no suceda nada?