Proyecto de robótica educativa

Inicio:

Hardware

- ✓ **Controladores**
- ✓ **Drivers**
- ✓ **Sensores**
- ✓ **Actuadores**

Software

- ✓ **Arduino**
  - ✓ **Lenguaje C adaptado**

Controladores

MICROCONTROLADORES PIC

El nombre verdadero de este microcontrolador es PICmicro (Peripheral Interface Controller), conocido bajo el nombre PIC. Su primer antecesor fue creado en 1975 por la compañía General Instruments. Este chip denominado PIC1650 fue diseñado para propósitos completamente diferentes. Diez años más tarde, al añadir una memoria EEPROM, este circuito se convirtió en un verdadero microcontrolador PIC. Hace unos pocos años la compañía Microchip Technology fabricó la 5 billonésima muestra.

Familia	ROM [Kbytes]	RAM [bytes]	Pines	Frecuencia de reloj. [MHz]	Entradas A/D	Resolución del convertidor A/D	Comparadores	Temporizadores de 8/16 bits	Comunicación serial	Salidas PWM	Otros
Arquitectura de la gama baja de 8 bits, palabra de instrucción de 12 bits											
PIC10FXXX	0.375 - 0.75	16 - 24	6 - 8	4 - 8	0 - 2	8	0 - 1	1 x 8	-	-	-
PIC12FXXX	0.75 - 1.5	25 - 38	8	4 - 8	0 - 3	8	0 - 1	1 x 8	-	-	EEPROM
PIC16FXXX	0.75 - 3	25 - 134	14 - 44	20	0 - 3	8	0 - 2	1 x 8	-	-	EEPROM
PIC16HVXXX	1.5	25	18 - 20	20	-	-	-	1 x 8	-	-	Vdd = 15V
Arquitectura de la gama media de 8 bits, palabra de instrucción de 14 bits											
PIC12FXXX	1.75 - 3.5	64 - 128	8	20	0 - 4	10	1	1 - 2 x 8 1 x 16	-	0 - 1	EEPROM
PIC12HVXXX	1.75	64	8	20	0 - 4	10	1	1 - 2 x 8 1 x 16	-	0 - 1	-
PIC16FXXX	1.75 - 14	64 - 368	14 - 64	20	0 - 13	8 or 10	0 - 2	1 - 2 x 8 1 x 16	USART I2C SPI	0 - 3	-
PIC16HVXXX	1.75 - 3.5	64 - 128	14 - 20	20	0 - 12	10	2	2 x 8 1 x 16	USART I2C SPI	-	-
Arquitectura de la gama alta de 8 bits, palabra de instrucción de 16 bits											
PIC18FXXX	4 - 128	256 - 3936	18 - 80	32 - 48	4 - 16	10 or 12	0 - 3	0 - 2 x 8 2 - 3 x 16	USB2.0 CAN2.0 USART I2C SPI	0 - 5	-
PIC18FXXJXX	8 - 128	1024 - 3936	28 - 100	40 - 48	10 - 16	10	2	0 - 2 x 8 2 - 3 x 16	USB2.0 USART Ethernet I2C SPI	2 - 5	-
PIC18FXXKXX	8 - 64	768 - 3936	28 - 44	64	10 - 13	10	2	1 x 8 3 x 16	USART I2C SPI	2	-

Todos los microcontroladores PIC utilizan una arquitectura Harvard, lo que quiere decir que su memoria de programa está conectada a la CPU por más de 8 líneas. Hay microcontroladores de 12, 14 y 16 bits,

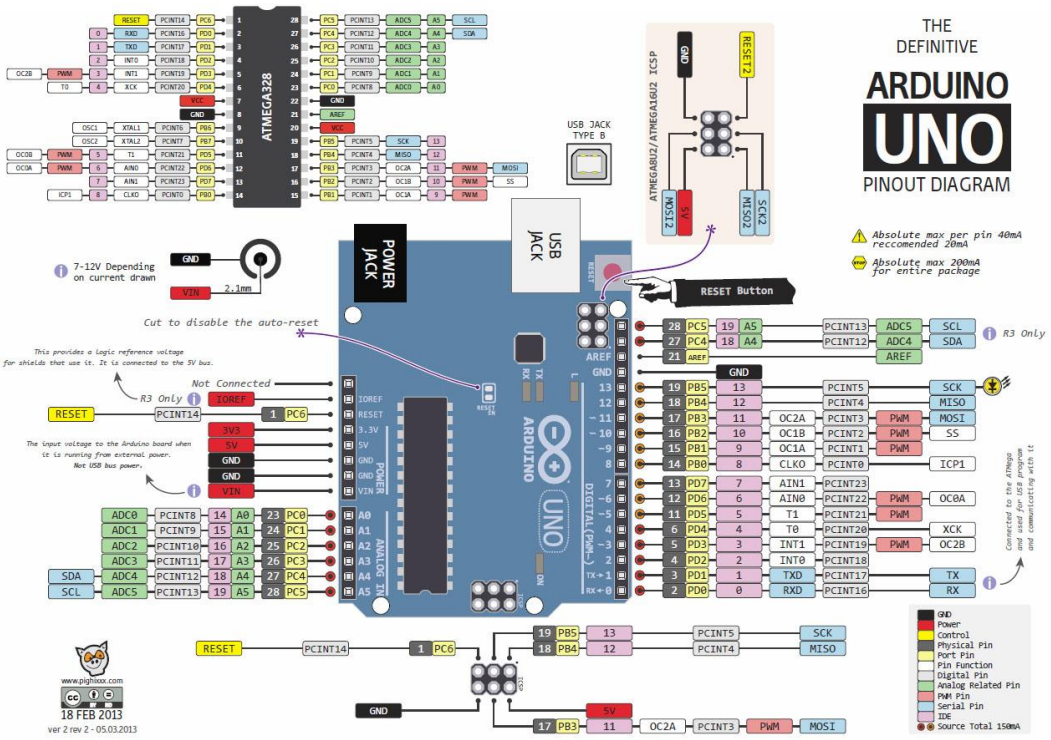
dependiendo de la anchura del bus. La tabla anterior muestra las características principales de estas tres categorías.
Como se puede ver en la tabla de la página anterior, salvo “los monstruos de 16 bits” PIC 24FXXX y PIC 24HXXX – todos los microcontroladores tienen la arquitectura Harvard de 8 bits y pertenecen a una de las tres grandes grupos. Por eso, dependiendo del tamaño de palabra de programa existen la primera, la segunda y la tercera categoría de microcontroladores, es decir Microcontroladores de 12, 14 o 16 bits. Puesto que disponen del núcleo similar de 8 bits, todos utilizan el mismo juego de instrucciones y el “esqueleto” básico de hardware conectado a más o menos unidades periféricas.

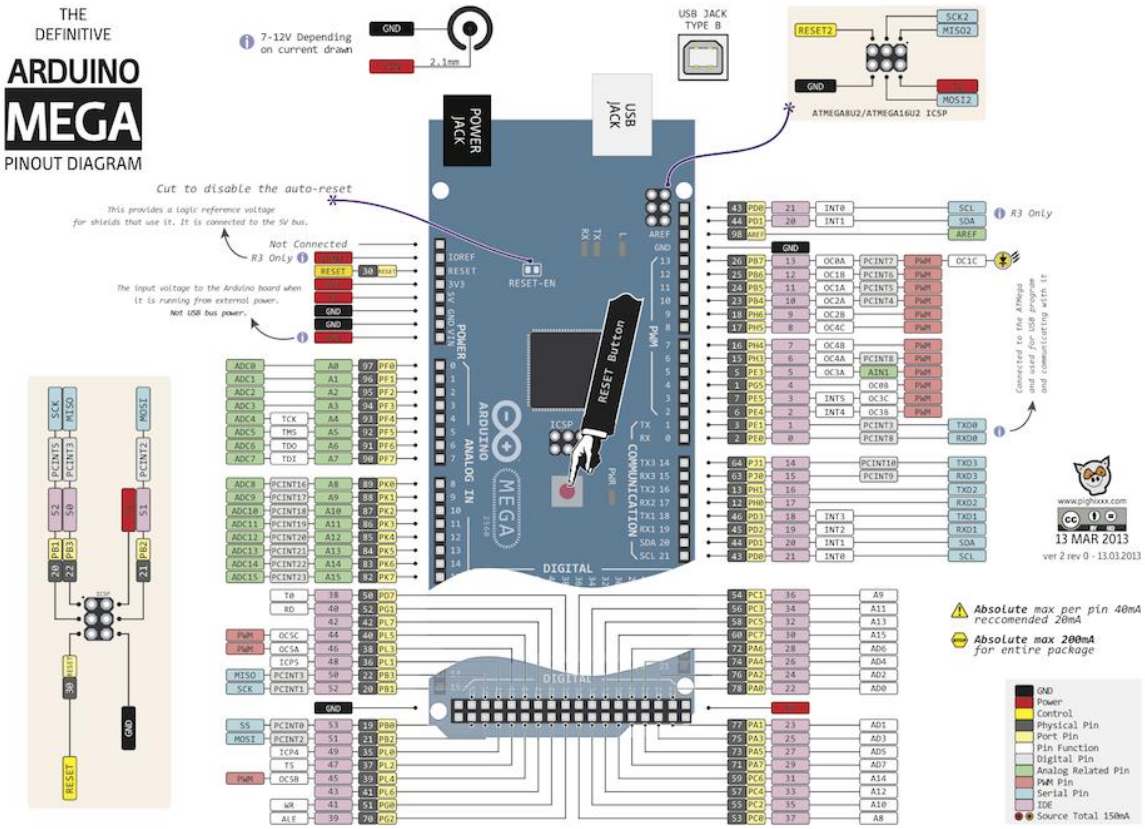
RASPBERRY PI

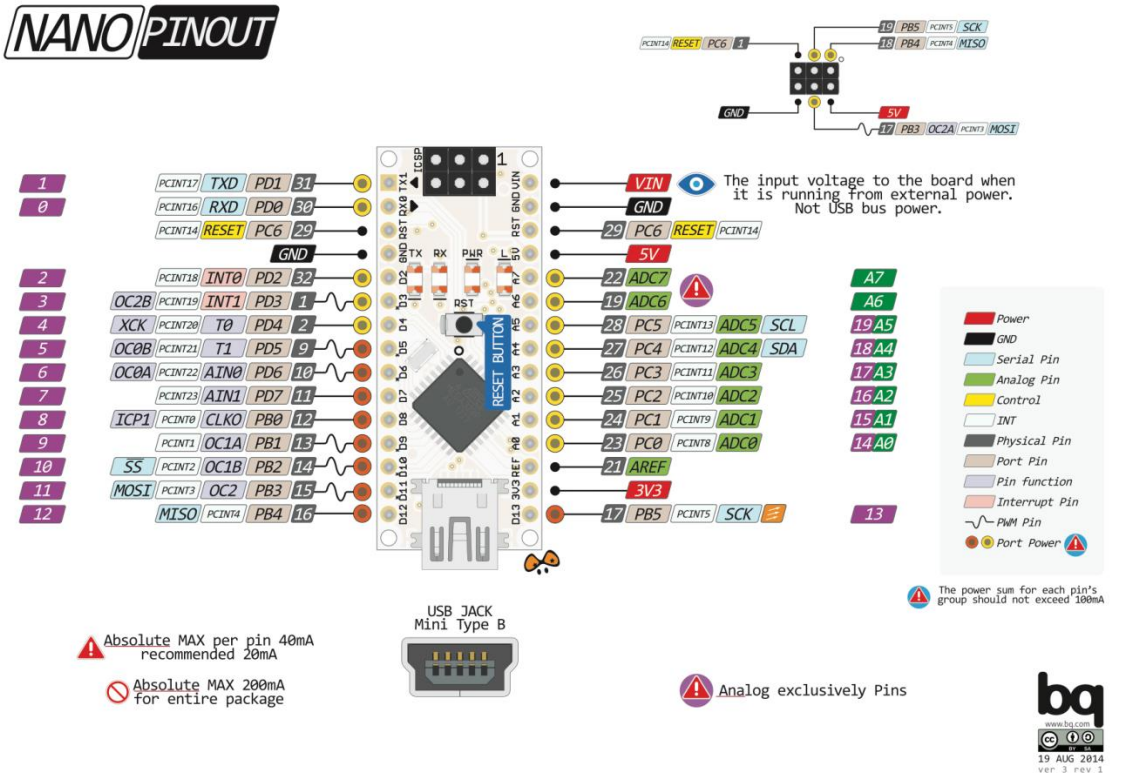
Los Mini PC´s son siempre una buena opción para disfrutar de toda la potencia de un ordenador pero recurriendo a un tamaño compacto. Los podemos utilizar como servidor de contenidos, conectados al televisor y, por supuesto, como ordenador al uso. Raspberry Pi es uno de los productos más populares para estos fines, tanto por su atractivo precio como por las enormes opciones que trae consigo. En 2012 llamó la atención de decenas de miles de entusiastas, tras emerger como un ordenador de bajo coste que pudiese llegar al mayor número de usuarios posible, y gracias a la amplia comunidad que aporta valor a este proyecto, existen usos tan diversos como sencillos de implementar con unos pocos conocimientos.
Pero, ¿qué es Raspberry Pi? En realidad, se trata de una diminuta placa base de 85 x 54 milímetros (un poco más grande que una cajetilla de tabaco) en el que se aloja un chip Broadcom BCM2835 con procesador ARM hasta a 1 GHz de velocidad, GPU Video Core IV y hasta 512 Mbyte de memoria RAM. En cuanto a su precio, suele estar por debajo de los 40 euros, una de las razones que explica su popularidad. De hecho, a finales de 2013 se superaron ya las dos millones de unidades vendidas en todo el mundo.

Arduino

Arduino es una plataforma de prototipos electrónica de código abierto (open-source) basada en hardware y software flexibles y fáciles de usar. Está pensado para artistas, diseñadores, como hobby y para cualquiera interesado en crear objetos o entornos interactivos.
Arduino puede sentir el entorno mediante la recepción de entradas desde una variedad de sensores y puede afectar a su alrededor mediante el control de luces, motores y otros artefactos. El microcontrolador de la placa se programa usando el *Arduino Programming Language* (basado en Wiring) y el *Arduino Development Environment* (basado en Processing). Los proyectos de Arduino pueden ser autónomos o se pueden comunicar con software en ejecución en un ordenador (por ejemplo con *Flash, Processing, MaxMSP*, etc.).
Las placas se pueden ensamblar a mano o encargarlas pre ensambladas; el software se puede descargar gratuitamente. Los diseños de referencia del hardware (archivos CAD) están disponibles bajo licencia open-source, por lo que eres libre de adaptarlas a tus necesidades.
Arduino recibió una mención honorífica en la sección *Digital Communities* del *Ars Electrónica Prix* en 2006




THE DEFINITIVE
ARDUINO MEGA
PINOUT DIAGRAM
7-12V Depending on current drawn
USB JACK TYPE B
ATMEGA8U2/ATMEGA16U2 ICSP
Cut to disable the auto-reset
Not Connected
R3 Only
RESET Button
RESET-EN
POWER JACK
USB JACK
ANALOG IN
DIGITAL
COMMUNICATION
PWM
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
13 MAR 2013
ver 2 rev 0 - 13.03.2013


NANO PINOUT
The input voltage to the board when it is running from external power. Not USB bus power.
RESET BUTTON
USB JACK Mini Type B
Absolute MAX per pin 40mA recommended 20mA
Absolute MAX 200mA for entire package
Analog exclusively Pins
The power sum for each pin's group should not exceed 100mA
19 AUG 2014
ver 3 rev 1

Drivers

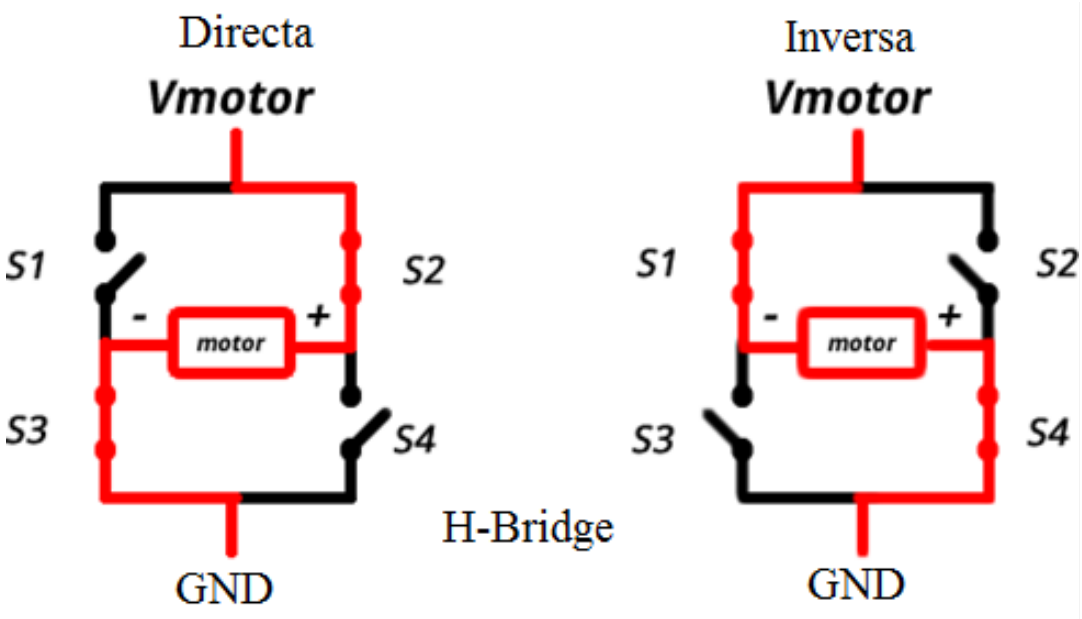
Esquemas, Motores DC y puente H

Hemos visto por encima las características básicas de los motores de corriente continua con escobillas (Porque hay varios tipos además de este), pero vamos a entrar un poco más en detalle de que cosas debes considerar cuando vayas a elegir uno de estos motores.
Cuando compres un motor de corriente continua debes fijarte en tres cosas básicamente:

Voltaje de funcionamiento*: Que especifica los rangos de tensión entre los que el motor funcionará correctamente. Por ejemplo los más usados se mueven entre 3 y 6V, pero no es raro encontrar de 9, 12 y 24V*
Revoluciones por minuto*: Que indica la velocidad que puede alcanzar a las distintas tensiones y que se suele consignar como Rpm. Típicamente en motores de Hobby andaremos entre 200 y 2.000 rpm*
Par motor*: O potencia que el motor puede desarrollar y que indica la carga útil que puede mover. En inglés se llama Torque.*

En cuanto a la rotación del motor, porque este es un tema de mediana complejidad, ya que para ello necesitamos **invertir la polaridad de la tensión** en bornes del motor, y esto lo no podemos hacer usando solamente nuestro **Arduino** *(Porque Arduino puede proporcionar +5V pero no -5V ni la corriente que consume un motor DC con carga).*

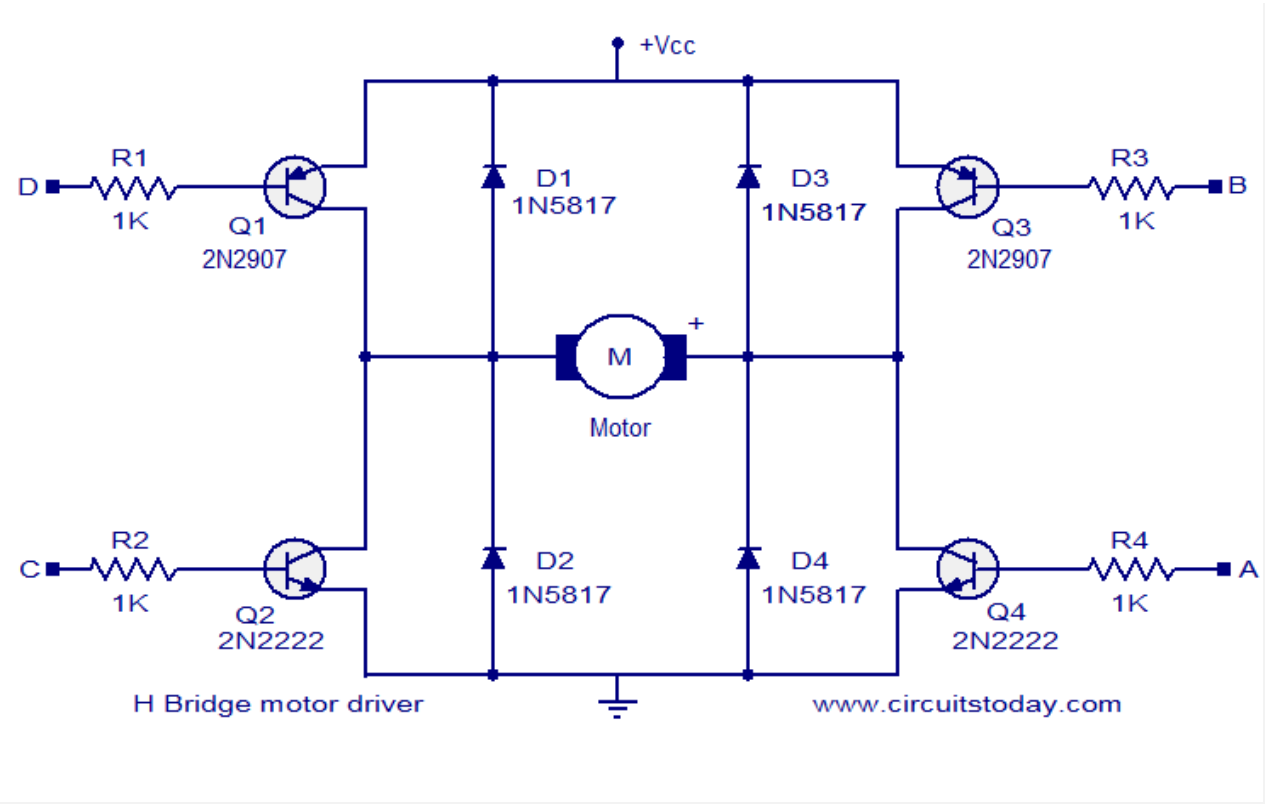
Para resolver este problema tenemos que usar un poco más de ingenio y diseñar un circuito que se llama puente H (**H-bridge**). Para comprenderlo, imaginemos el siguiente montaje a base de interruptores:



Cuando usamos los interruptores en la posición de la imagen izquierda el motor gira en un sentido que llamaremos directo. Pero si los colocamos en la posición de la derecha girará en sentido contrario, porque hemos invertido la polaridad de la tensión en las entradas del motor, y por tanto el sentido de giro.

A estos circuitos se les llama **H-bridge**, porque recuerdan vagamente a una H alrededor del motor.

Claro que invertir el giro mediante interruptores es poco práctico, y como la idea es ser electrónicos (*Sin carcajadas, por favor*), vamos a ver cómo podemos hacer la misma función usando electrónica y que no tengamos que conmutar manualmente, sino mediante señales eléctricas. Un típico **H-Bridge** seria parecido a este circuito:



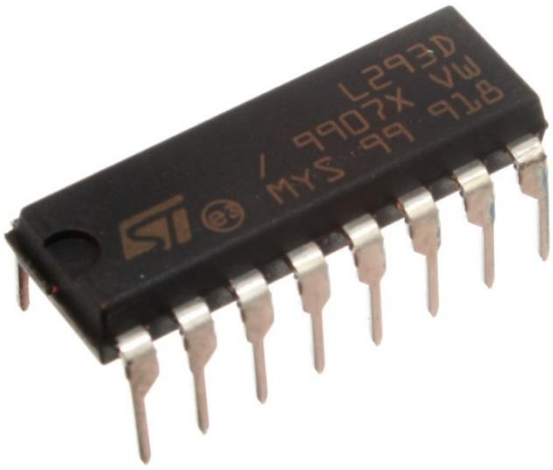
Los diodos, son para protección de la descarga inductiva del motor y proteger los transistores. Jugando con la tensión en los pines A; B; C; D podemos conseguir que los transistores entren en corte o saturación y disponemos de un medio electrónico para abrir o cerrar el equivalente a los interruptores mediante transistores.

*Observa que dos transistores son unos **PNP** y otros dos **NPN** para jugar con la polaridad de la tensión. El campo magnético del rotor almacena energía, que puede ser importante, y que cuando cortamos la alimentación debe ser liberada en forma de corriente eléctrica, y adecuadamente dirigida por los diodos para impedir daños en los transistores o en la fuente de alimentación*

Disponemos de varias versiones de circuitos **H-bridge** dependiendo de la tensión y la intensidad que se debe conmutar.
Nosotros hoy vamos a utiliza un integrado barato, probado y fácil de encontrar que incluye dos H**-bridge** y que nos sirve para pequeños motores de corriente continua. Se llama **L293D**.

EL H BRIDGE L293D

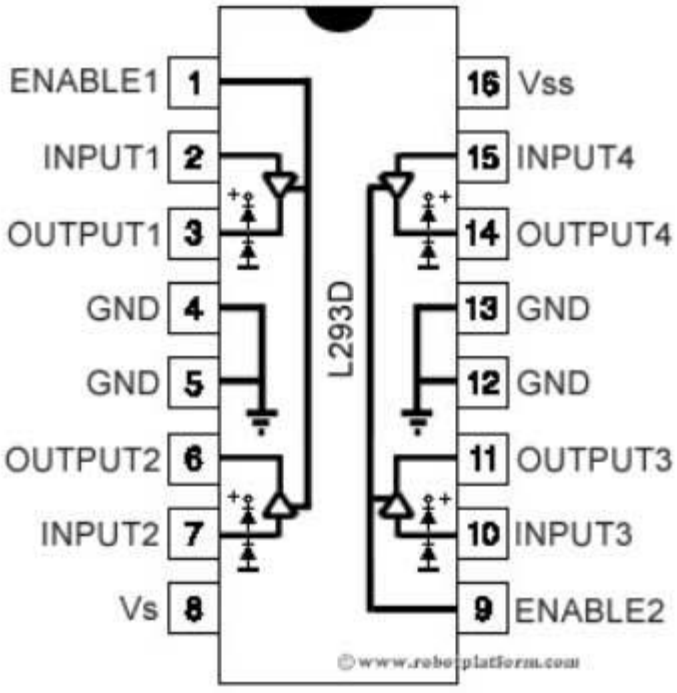
El **L293D**, es un pequeño integrado que incluye dos puentes **H-Bridge** que podemos usar para gobernar simultáneamente dos motores CC.



Si buscamos el datasheet del **L293D**, veremos que aunque el funciona a 5V internamente, puede conmutar tensiones de hasta 36V para los motores.

Asegurarte de que no sobrepasar la tensión máxima de trabajo de los motores, así que el chip lo aguanta, pero lo de tu motor no necesariamente aguanta esta tensión máxima del puente H.

El Pinout del chip es:



Veamos que son los diferentes pines del L293D:

ENABLE	CONTROL PIN 2	CONTROL PIN 7	MOTOR STATUS
LOW	–	–	Motor parado
HIGH	HIGH	LOW	Gira adelante
HIGH	LOW	HIGH	Gira al revés
HIGH	HIGH	HIGH	Motor parado
HIGH	LOW	LOW	Motor parado

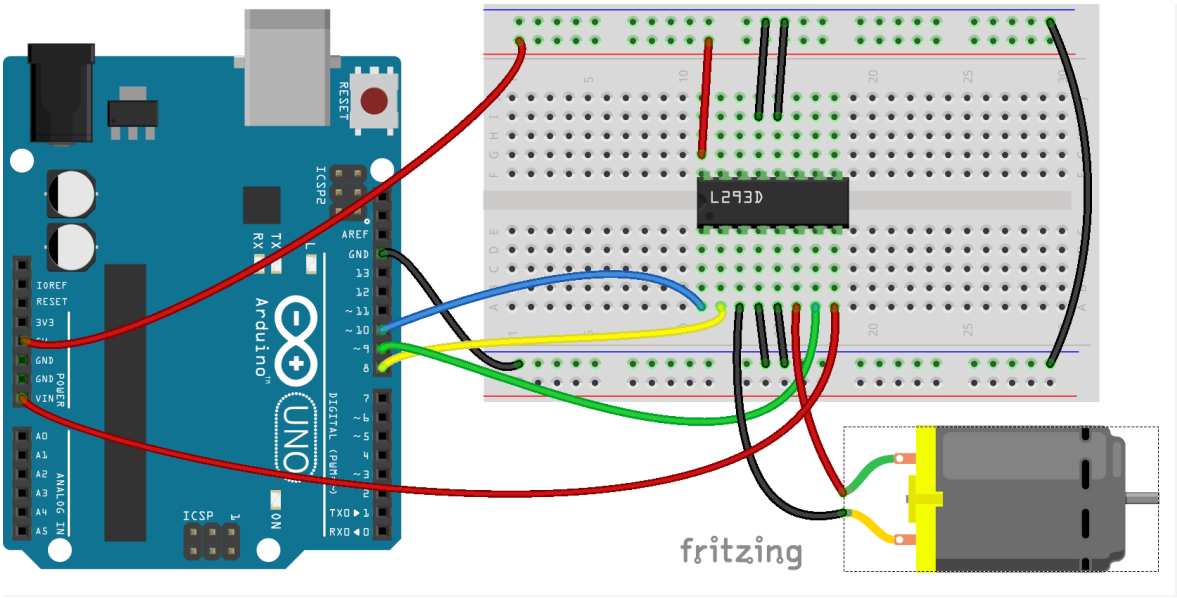
Por tanto tenemos que activar el pin enable para que el motor gire y después usamos los pines Input1 e Input2 con valor opuestos para hacer girar el motor en una dirección o en la contraria. Veamos cómo hacer el montaje con nuestro Arduino.
Hagamos un resumen de conexiones

PIN L293D	PIN ARDUINO	DESCRIPCIÓN
1	10	Enable
2	9	Input 1
3	–	Motor1 +
4,5, 12,13	GND	GND
6	–	Motor1 –
7	8	Input 2
8	Vin	Alimentación del motor
16	5V	Alimentación del L293D

Vamos con el esquema de protoboard.

ESQUEMA DE PROTOBOARD

Una vez que tenemos claras las conexiones, el diagrama de protoboard del chip L293D a nuestro Arduino será algo así:



Veamos el programa que vamos a usar para controlar este motor. Vamos a usar los 3 pines de la tabla anterior. Arduino Pin 10 es el Enable del Motor 1 y usamos los pines 8 y 9 para controlar el sentido de giro. Por tanto

```
#define E1 10  // Enable Pin para motor 1
#define I1 8     // Control pin 1 para motor 1
#define I2 9     // Control pin 2 para motor 1

void setup()
  {
      for (int i = 8 ; i<11 ; i++)                 // Inicializamos los pines
      pinMode( i, OUTPUT);
  }
void loop()
  {   digitalWrite(E1, HIGH);                    // Activamos Motor1
      digitalWrite(I1, HIGH);                   // Arrancamos
      digitalWrite(I2, LOW);
      delay(3000);

      digitalWrite(E1, LOW);                   // Paramos Motor 1
      delay(1000);
      digitalWrite(E1, HIGH);                  // Activamos Motor1

      digitalWrite(I1, LOW);      // Arrancamos con cambio de dirección
      digitalWrite(I2, HIGH);
```

```
    delay(3000);

    digitalWrite(E1, LOW);      // Paramos Motor 1
    delay(1000);
}
```

El programa no puede ser más sencillo. Activamos el Enable1 para arrancar Motor1, y luego usamos I1 e I2 con valores invertidos. El motor arranca y lo paramos a los 3 segundos. A la de 1 segundo levantamos de nuevo el Enable1y al intercambiar los valores de I1 e I2 el giro del motor se inicia y en la dirección contraria.

Conectando el pin Enable1 al pin 10 del Arduino que es PWM (Así, como sin querer) y los **L293D** vienen diseñados para variar la velocidad de giro de los motores correspondientes con la tensión aplicada a este pin, por lo que resulta trivial variar la velocidad del motor, solo dandole valores analógicos.

Si por ejemplo añadimos un potenciómetro de nuevo conectado al A1, podemos escribir este valor directamente (dividido por 4, claro) al pin 10 como analógico, sustituyendo la línea

```
digitalWrite(E1, HIGH);  // Activamos Motor1
```

Por esta otra que escribe un valor analógico:

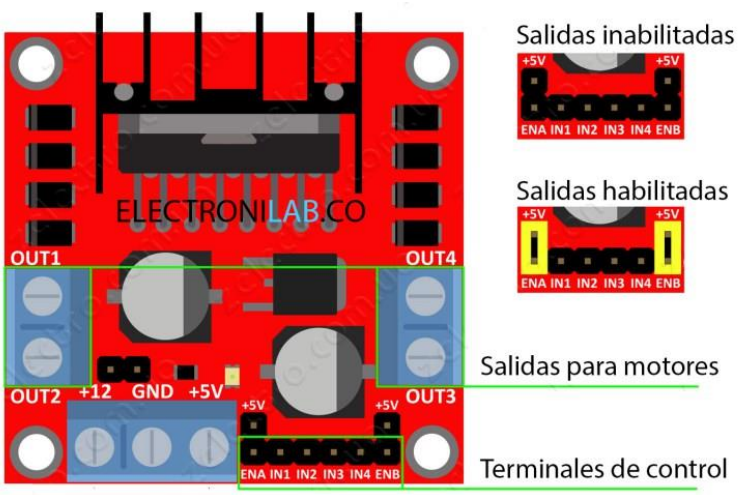
```
analogWrite(E1, analogRead(A1) /4 ) ;
```

Fijaremos la velocidad de giro del motor en función del valor del potenciómetro. El programa corregido, quedaría poco más o menos así:

```
#define E1 10  // Enable Pin for motor 1
#define I1 8     // Control pin 1 for motor 1
#define I2 9     // Control pin 2 for motor 1

void setup()

 {
    for (int i = 8 ; i<11 ; i++)                  // Inicializamos los pines
    pinMode( i, OUTPUT);
 }

void loop()

 {  analogWrite(E1, analogRead(A1) / 4);           // Activamos Motor1
    digitalWrite(I1, HIGH);      // Arrancamos
    digitalWrite(I2, LOW);
    delay(3000);

    digitalWrite(E1, LOW);                        // Paramos Motor 1
    delay(1000);
    analogWrite(E1, analogRead(A1) / 4);          // Activamos Motor1

    digitalWrite(I1, LOW);       // Arrancamos con cambio de direccion
    digitalWrite(I2, HIGH);
    delay(3000);

    digitalWrite(E1, LOW);                        // Paramos Motor 1
    delay(1000);
 }
```

Driver L298N para motores DC y pasó a paso con Arduino

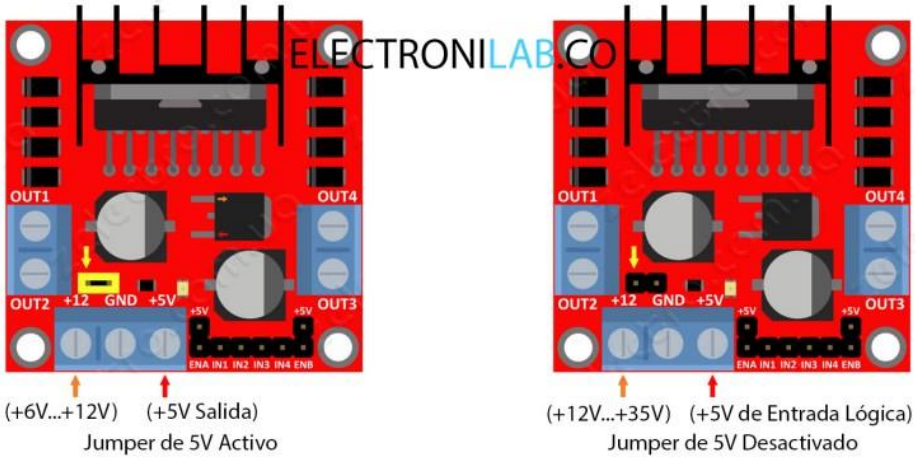
El **Driver dual para motores (Full-Bridge) – L298N**. Este módulo basado en el chip L298N te permite controlar dos motores de corriente continua o un motor paso a paso bipolar de hasta 2 amperios.
El módulo cuenta con todos los componentes necesarios para funcionar sin necesidad de elementos adicionales, entre ellos diodos de protección y un regulador **LM7805** que suministra 5V a la parte lógica del integrado L298N. Cuenta con jumper de selección para habilitar cada una de las salidas del módulo (A y B). La **salida A** esta conformada por **OUT1** y **OUT2** y la **salida B** por **OUT3** y **OUT4**. Los pines de habilitación son **ENA** y **ENB** respectivamente.



En la parte inferior se encuentran los pines de control del módulo, marcados como **IN1**, **IN2**, **IN3** e **IN4**.

Conexión de alimentación

Este módulo se puede alimentar de 2 maneras gracias al regulador integrado LM7805.



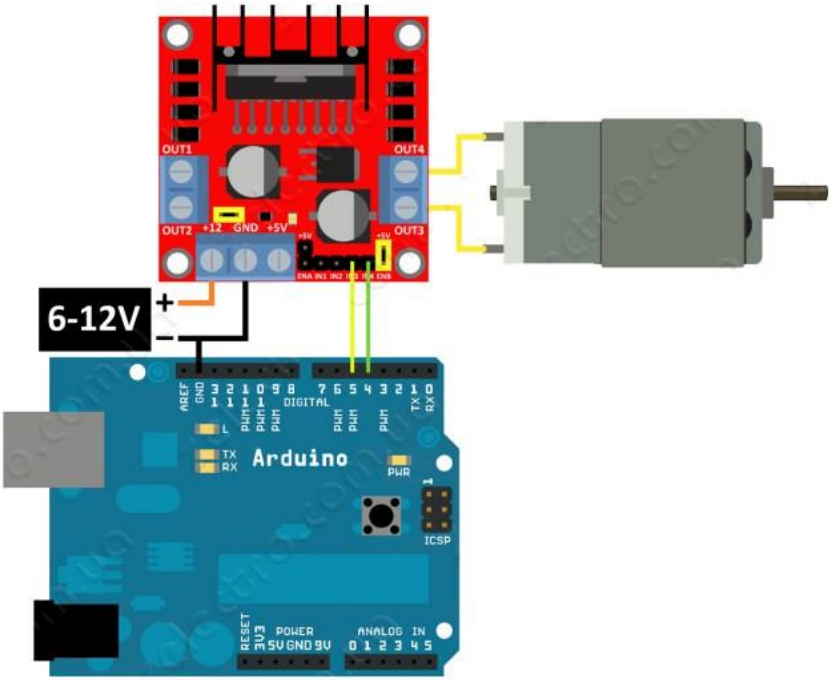
Cuando el jumper de selección de 5V se encuentra **activo**, el módulo permite una alimentación de entre **6V a 12V DC**. Como el regulador se encuentra activo, el pin marcado como +5V tendrá un voltaje de 5V DC. Este voltaje se puede usar para alimentar la parte de control del módulo ya sea un microcontrolador o un Arduino, pero recomendamos que el consumo no sea mayor a 500 mA.
Cuando el jumper de selección de 5V se encuentra **inactivo**, el módulo permite una alimentación de entre **12V a 35V DC**. Como el regulador no está funcionando, tendremos que conectar el pin de +5V a una tensión de 5V para alimentar la parte lógica del L298N. Usualmente esta tensión es la misma de la parte de control, ya sea un microcontrolador o Arduino.

Recomendamos nunca conectar una tensión de entrada al pin de +5V, cuando el jumper de selección de 5V se encuentre activado. Esto provocaría un corto y podría dañar permanentemente el módulo.

Control de un motor DC

Como demostración, vamos a controlar un motor DC a través de la salida B del módulo. El pin **ENB** se conectará con el jumper a +5V.
El ejemplo está desarrollado en **Arduino UNO**, pero el código es compatible con cualquier **Arduino** o pingüino que es un proyecto parecido a arduino pero con el microcontrolador PIC 18F2550 ó el 18F4550.

Esquema de conexión



Código en Arduino

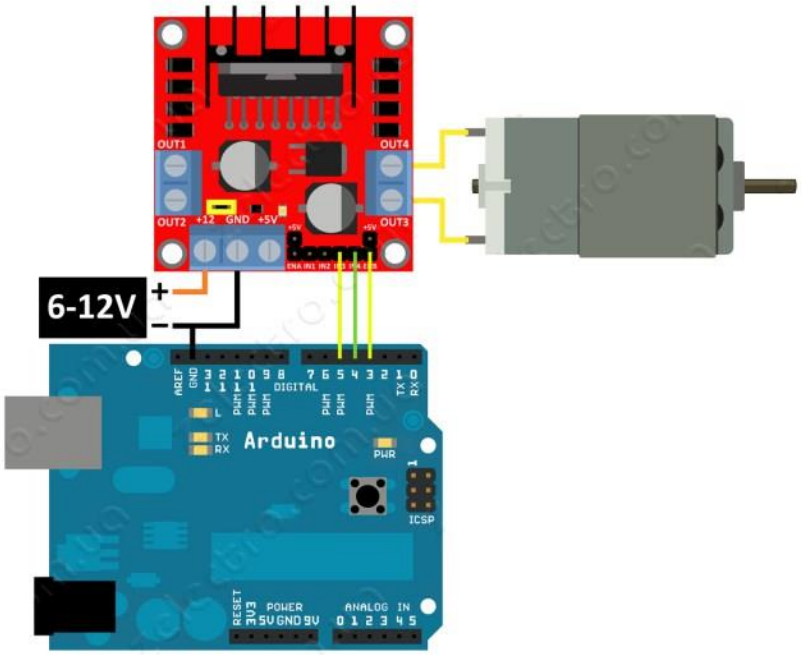
El programa básicamente activa el motor en un sentido por 4 segundos, luego detiene el motor por 0.5 segundos, después activa el motor en sentido inverso por 4 segundos y por último detiene el motor por 5 segundos. Luego repite la acción indefinidamente.

```
int IN3 = 5;
int IN4 = 4;

void setup()
{
  pinMode (IN4, OUTPUT);    // Input4 conectada al pin 4
  pinMode (IN3, OUTPUT);    // Input3 conectada al pin 5
}
void loop()
{

  digitalWrite (IN4, HIGH);    // Motor gira en un sentido
  digitalWrite (IN3, LOW);
  delay(4000);

  digitalWrite (IN4, LOW);     // Motor no gira
  delay(500);

  digitalWrite (IN3, HIGH);    // Motor gira en sentido inverso
  delay(4000);

  digitalWrite (IN3, LOW);     // Motor no gira
  delay(5000);
}
```

Control motor DC variando su velocidad

Si queremos controlar la velocidad del motor, tenemos que hacer uso de PWM. Este PWM será aplicado a los pines de activación de cada salida o pines ENA y ENB respectivamente, por tanto los jumper de selección no serán usados.

Esquema de conexión



Código en Arduino

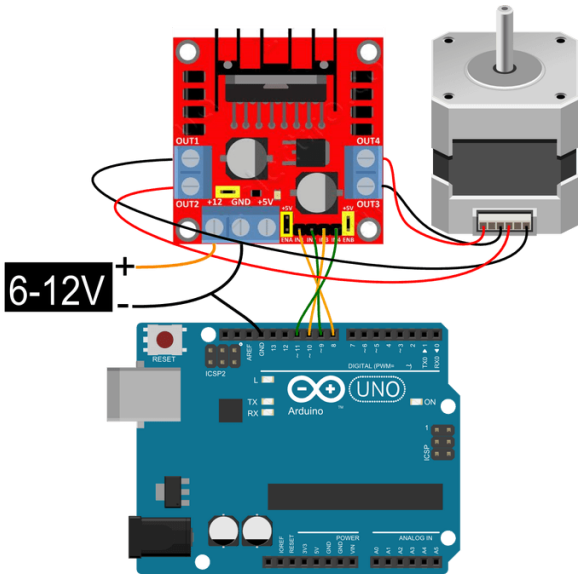
El programa controla la velocidad de un motor DC aplicando PWM al pin ENB del módulo L298N.

```
int IN3 = 5;    // Input3 conectada al pin 5
int IN4 = 4;    // Input4 conectada al pin 4
int ENB = 3;    // ENB conectada al pin 3 de Arduino
void setup()
{
 pinMode (ENB, OUTPUT);
 pinMode (IN3, OUTPUT);
 pinMode (IN4, OUTPUT);
}
void loop()
{

  digitalWrite (IN3, HIGH);    //Preparamos la salida para que el motor gire en un sentido
  digitalWrite (IN4, LOW);

  analogWrite(ENB,55);       // Aplicamos PWM al pin ENB, haciendo girar el motor, cada 2 seg aumenta la velocidad
  delay(2000);
  analogWrite(ENB,105);
  delay(2000);
  analogWrite(ENB,255);
  delay(2000);

  analogWrite(ENB,0);        // Apagamos el motor y esperamos 5 seg
  delay(5000);
}
```

Control de un motor pasó a paso bipolar

Los motores paso a paso pueden ser bipolares o unipolares. En este caso será un Bipolar. El máximo consumo permitido es de 2 amperios.

Esquema de conexión



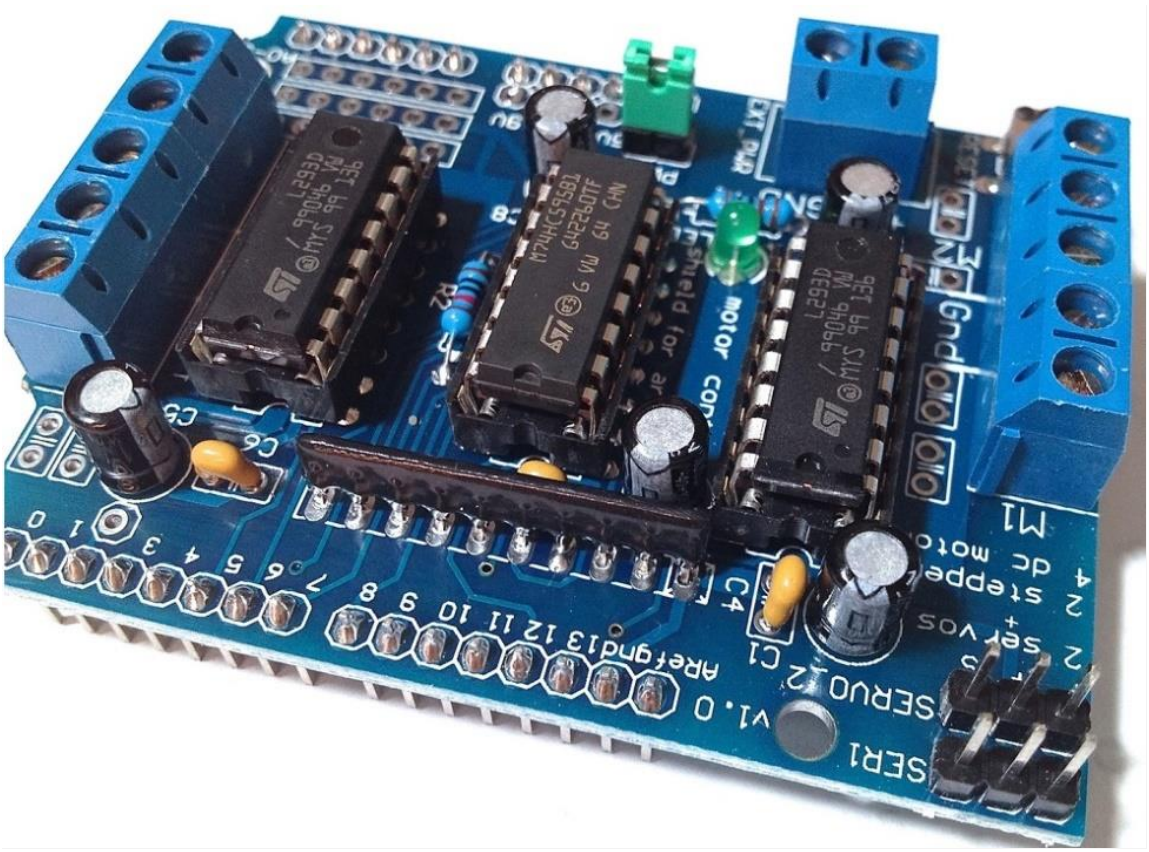
El esquema de conexión muestra la conexión utilizada entre el módulo L298N y el motor paso a paso. Cada una de las bobinas del motor está conectada a una salida del módulo. Para identificar las bobinas de un motor paso a paso utilizamos un multímetro en modo de continuidad. Los cables que dan continuidad son los extremos de cada bobina.
En este caso, como el motor paso a paso es de 12 VDC, utilizamos el jumper de selección de +5V, para activar el regulador interno del módulo y solo hacer uso de una fuente de 12 VDC para alimentar el motor. Los jumper de activación ENA y ENB los hemos activado de igual manera.

Código de Arduino

El código de Arduino hace girar el motor paso a paso una vuelta en un sentido y luego ejecuta otra vuelta en sentido opuesto. Este código hace uso de la librería '*Stepper*', que se instala por defecto en las últimas versiones del IDE de Arduino.
El valor de la variable **stepsPerRevolution** depende del número de pasos del motor paso a paso. Este valor se encuentra en las especificaciones de la hoja de datos del motor. En nuestro caso el motor paso a paso utilizado es de 48 pasos/vuelta.

```
#include <Stepper.h>
const int stepsPerRevolution = 48;                  // cambie este valor por el numero de pasos de su motor
                                                    // inicializa la librería 'stepper' en los pines 8 a 11
Stepper myStepper(stepsPerRevolution, 8,9,10,11);

void setup() {
                                                    // establece la velocidad en 60rpm
  myStepper.setSpeed(60);
                                                    // inicializa el puerto serial
  Serial.begin(9600);
}

void loop() {
                                                   // gira una vuelta en una dirección
   Serial.println("clockwise");
  myStepper.step(stepsPerRevolution);
  delay(500);
                                                   // gira otra vuelta en la otra dirección
  Serial.println("counterclockwise");
  myStepper.step(-stepsPerRevolution);
  delay(500);
}
```

ADAFRUIT MOTOR SHIELD V1.0

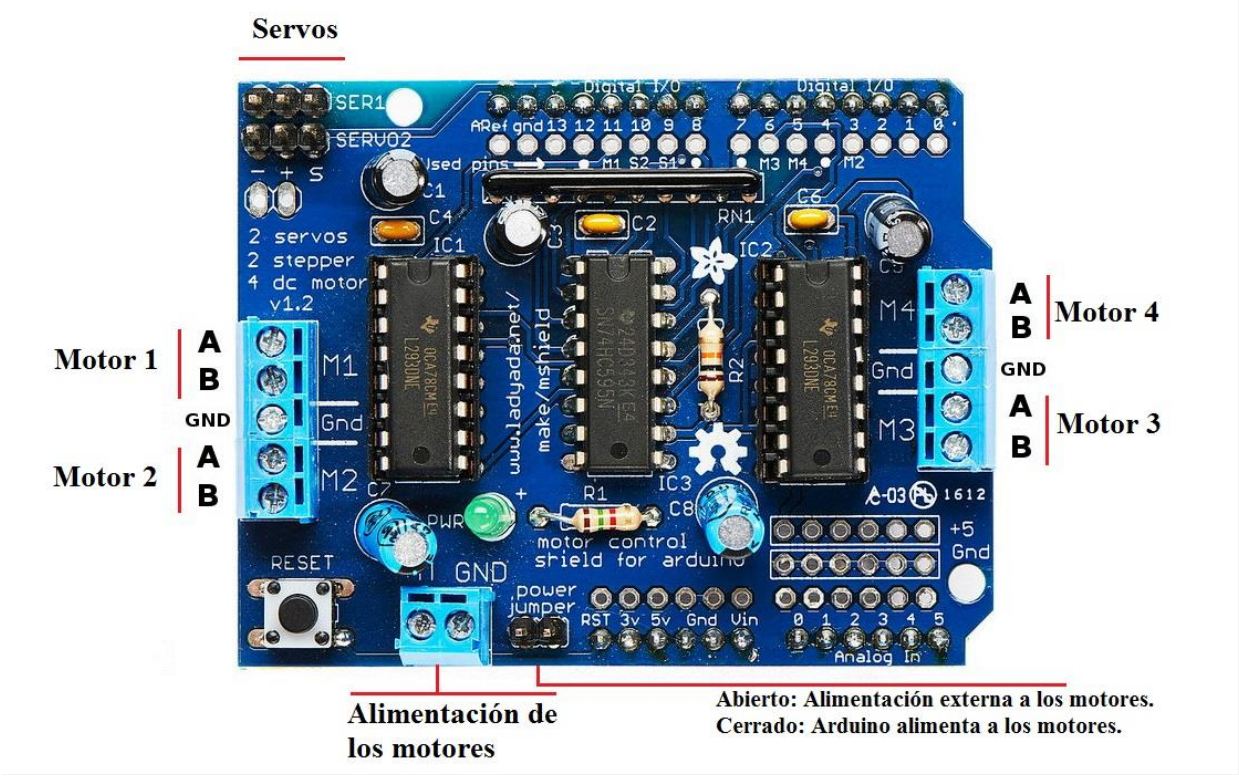


Las características principales son:

4 H Bridges *incluidos con dos* ***L293D*** *chips.*
Hasta ***4 motores CC*** *con control bidireccional y selección de velocidad de 8 bits.*
Máxima corriente de 0,6 Amperios (Aunque acepta picos de hasta 1,2) con una protección de sobre temperatura.
Acepta motores cuya alimentación va desde 4,5 a 25V y una corriente máxima de trabajo de 0,6A.
Podemos añadir otros dos motores Servos o paso a paso.
Dispone de alimentación de motores, separada de la del Shield para evitar ruido e interferencias.
Compatible con Arduino ***UNO*** *y* ***Mega****.*
Dispone de una librería cómoda para manejar los motores.

CONECTANDO EL MOTOR SHIELD

El shield conecta directamente los **H Bridges L293D** a los pines PWM de **Arduino** y además incorpora el **74HC595**, un **Shift Register**, para ahorrar pines en la conexión.
El Shield se reserva los pines 3,4,5,6,7,8,9,10,11,12, los pines 9 y 10 se usan para los servos en caso de que los incluyamos y quedan libres los pines 2 y 13, así como el 0 y 1 que se usan para la comunicación USB con nuestro PC.
Los pines A0 a A5 están disponibles para nuestro uso también para uso digital a pesar de ser analogicos.



Hay conexiones para los bornes de 4 motores, marcados como M1, M2, M3 y M4 y conviene que al conectar sigas el mismo criterio en los bornes y en el Shield porque de lo contrario conseguirás que alguno de los motores gire al revés de lo que pretendes. Lo que no es muy grave porque se arregla con facilidad.

Una cosa interesante de este shield es que nos permite separar la alimentación de los motores de la alimentación de Arduino y es de agradecer porque los motores generan mucha interferencia electromagnética que puede hacer que nuestro Arduino se comporte de forma errática.

Por eso siempre que sea posible conviene que separemos la alimentación de uno y otro (*Aunque en un robot autónomo va a ser difícil)*. Para ello basta con que quites el “power jumper” y la alimentación estará separada. Esto será imprescindible si tus motores son de digamos 9 0 12V porque tu Arduino solo funciona con 5V.

Conectamos los bornes del motor a los tornillos A y B del **Motor Shield**, dejando libre el pin central marcado como **GND**.

Como en este primer ejemplo vamos a alimentar el motor desde Arduino y el USB, debemos conectar el power jumper, para que alimentemos los motores directamente desde Arduino.

Con la **librería de Adafruit** para manejar el Shield directamente, que nos permitirá abstraernos del detalle de pines.
La librería que necesitamos es esta adafruit-Adafruit-Motor-Shield-library.
Para usarla, lo primero que tenemos que hacer es este par de instrucciones:

```
#include <AFMotor.h>
AF_DCMotor Motor1(1);
```

La primera línea, incluye la librería de AdaFruit en nuestro programa, y la segunda crea una instancia de un motor conectado a la puerta M1, y le iniciamos con el parámetro que le pasamos que puede ir del 1, M1 al 4, M4.
Para establecer la velocidad del motor hacemos:

```
Motor1.setSpeed(200);    // Definimos la velocidad de  Motor1
Motor1.run(RELEASE);
```

La primera línea pone el motor a 200 / 255 del tope de revoluciones, y la segunda línea indica que queremos dejar el motor en *punto muerto.*
Si queremos que el motor avance usamos:

```
Motor1.run(FORDWARD);
```

Y para que retroceda hacemos

```
Motor1.run(BACKWARD);
```

Y eso es todo lo necesario para controlar un motor. Si queremos hacer un primer programa sencillo que haga avanzar el motor unos segundos y después retroceda, podemos escribir un programa similar a este programa.

```
#include <AFMotor1.h>
AF_DCMotor1 Motor1(1);

void setup()
  {       Motor1.run(RELEASE);     }

void loop()
  {   Motor1.run(FORWARD);
      delay (2000);

      Motor1.setSpeed(180);
      Motor1.run(BACKWARD);
      delay (2000);
  }
```

MOVIENDO VARIOS MOTORES SIMULTÁNEAMENTE

Vamos a conectar ahora, 4 motores a nuestro **Motor Shield**.
Vamos con el programa de prueba. Lo primero es incluir la librería y definir las instancias de los motores que esta ocasión son 4 en vez de 1:

```
#include <AFMotor.h>
AF_DCMotor Motor1(1);
AF_DCMotor Motor2(2);
AF_DCMotor Motor3(3);
AF_DCMotor Motor4(4);
```

Definimos la velocidad de los 4 motores en el setup:

```
void setup()
  {    Serial.begin(9600);          // set up Serial library at 9600 bps
       Motor1.setSpeed(255);
       Motor2.setSpeed(255);
       Motor3.setSpeed(255);
       Motor4.setSpeed(255);
  }
```

Y por último vamos a mandar moverse adelante y atrás y para los 4 motores simultáneamente:

```
Motor1.run(RELEASE);
Motor2.run(RELEASE);
Motor3.run(RELEASE);
Motor4.run(RELEASE);
delay (1000) ;

Motor1.run(FORWARD) ;
Motor2.run(FORWARD);
Motor3.run(FORWARD);
Motor4.run(FORWARD);
delay (2000);

Motor1.run(BACKWARD);
Motor2.run(BACKWARD);
Motor3.run(BACKWARD);
Motor4.run(BACKWARD);
delay (2000);
}
```

Es muy similar a mover un motor solo en la programación lo que se debe tener en cuenta cuando hablamos de más de un motor es el sentido de giro este nos dará la dirección de nuestro autómata.

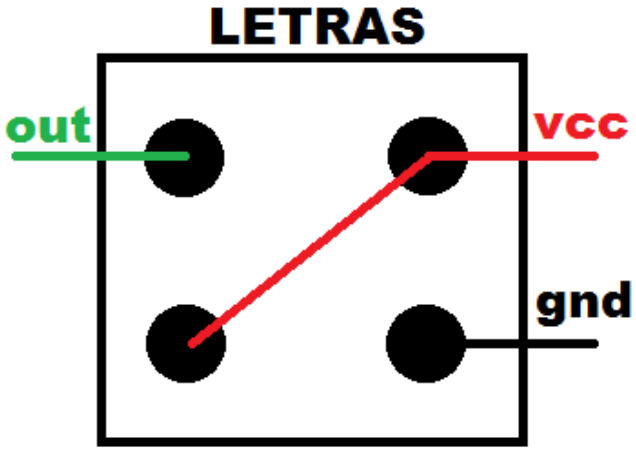
Sensores

Uso del sensor infrarrojo CNY70 con Arduino

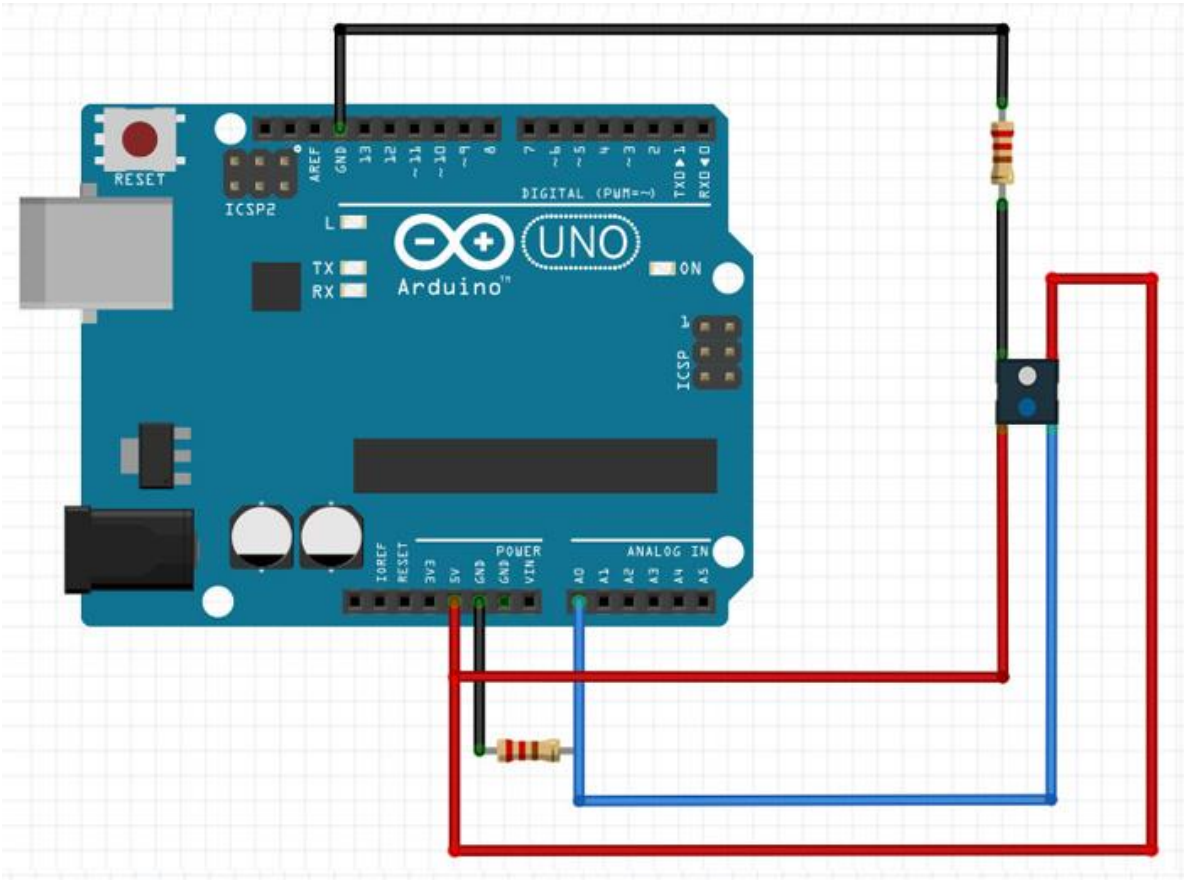
El **sensor infrarrojo CNY70** es un sensor de corto alcance (5cm aprox.) basado en un emisor de luz y un receptor, ambos apuntando en la misma dirección. El sensor CNY70 emite un haz de luz infrarroja que rebota contra los objetos y vuelve para ser captado de nuevo por el fototransistor.
Según el montaje, el sensor devuelve 1 o 0 (si usamos una entrada digital) o un valor proporcional a la cantidad de rayo reflectado por el objeto (si usamos una entrada analógica), permitiendo la detección dinámica entre blanco y negro en el caso de los robots sigue líneas. Este sensor también permite la detección de distintos colores, determinando los valores de margen que separan unos colores de otros. Sus aplicaciones son muy comunes, por ejemplo, se puede usar como fin de carrera, como sensor de proximidad, como sensor de presencia, en la detección de múltiples colores, en robots sigue líneas, entre otras.

Esquema y conexiones

El **sensor CNY70** tiene cuatro pines de conexión, dos de ellos son el ánodo y cátodo del emisor, y las otras dos son el colector y el emisor del receptor. Podemos usar las letras de la carcasa de plástico como referencia para identificar los pines.



El conexionado es muy simple, los pines VCC del sensor van a la salida de 5v del Arduino.
El pin GND del sensor va a GND del arduino con una resistencia de 100 Ohm intercalada.
El pin OUT del sensor va al pin analógico 0 del Arduino (A0).
Ponemos una resistencia de 1K entre el pin OUT del sensor y el pin GND del Arduino.



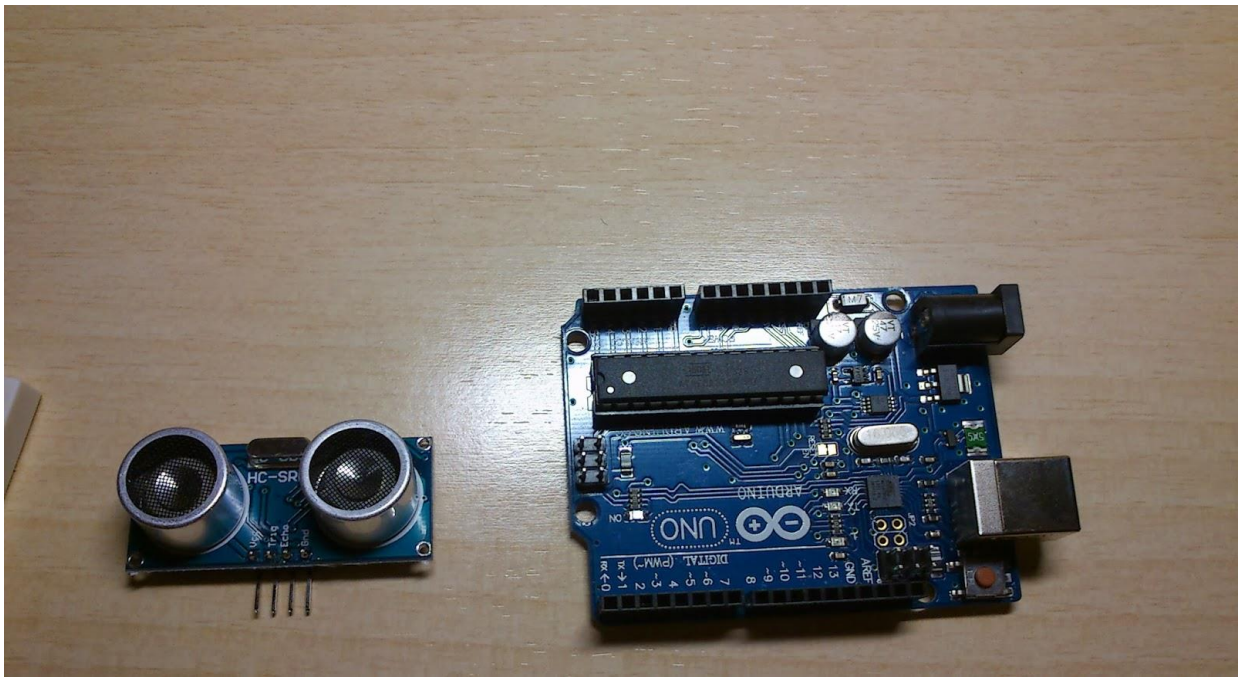
Programa

```
int pinReceptor = A0;              //Establecemos el pin a leer
int sensorVal;                     //Declaramos una variable para almacenar el valor de la lectura

void setup(){
Serial.begin(9600);                // Abrimos comunicación Serial
}

void loop(){
sensorVal = analogRead(pinReceptor);          //Guardamos la lectura del pin Analógico
Serial.println(sensorVal);                    //Sacamos la lectura por serial
delay(500);                                    //Pequeña pausa de medio segundo
}
```

Sensor ultrasonidos HC-SR04

El sensor de ultrasonidos se enmarca dentro de los sensores para medir distancias o superar obstáculos, entre otras posibles funciones.
En este caso vamos a utilizarlo para la medición de distancias. Esto lo consigue enviando un ultrasonido (inaudible para el oído humano por su alta frecuencia) a través de uno de la pareja de cilindros que compone el sensor (un transductor) y espera a que dicho sonido rebote sobre un objeto y vuelva, retorno captado por el otro cilindro.

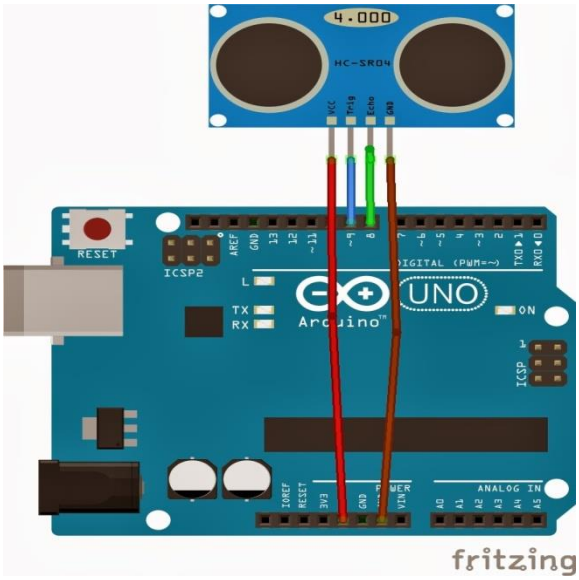
Este sensor en concreto tiene un rango de distancias sensible entre 3cm y 3m con una precisión de 3mm. El tiempo que transcurre entre el envío y la recepción del ultrasonido.



Cómo traducir dicho tiempo en distancia

Aprovechando que la velocidad de dicho ultrasonido en el aire es de valor **340 m/s**, o **0,034 cm/us** (ya que trabajaremos con centímetros y microsegundos). Para calcular la distancia, recordaremos que **v=d/t** (definición de velocidad: distancia recorrida en un determinado tiempo).
De la fórmula anterior despejamos **d**, obteniendo **d=v·t**, siendo v la constante anteriormente citada y t el valor devuelto por el sensor a la placa Arduino.
También habrá que dividir el resultado entre **2** dado que el tiempo recibido es el tiempo de ida y vuelta.

Conexiones



El sensor consta de 4 pines: "VCC" conectado a la salida de 5V de la placa, "Trig" conectado al pin digital de la placa encargado de enviar el pulso ultrasónico, "Echo" al pin de entrada digital que recibirá el eco de dicho pulso y "GND" a tierra.

Programa

```
long distancia;
long tiempo;

void setup(){

  Serial.begin(9600);
  pinMode(9, OUTPUT);       //activación del pin 9 como salida: para el pulso
  pinMode(8, INPUT);        //activación del pin 8 como entrada: tiempo del rebote
}
void loop(){

  digitalWrite(9,LOW);          //Por cuestión de estabilización del sensor
  delayMicroseconds(5);
  digitalWrite(9, HIGH);         //envío del pulso ultrasónico
  delayMicroseconds(10);
  tiempo=pulseIn(8, HIGH);       //(*Función para medir la longitud del pulso entrante).
  Serial.println("Distancia ");      //Monitorización en centímetros por el monitor serial

  Serial.println(distancia);
  Serial.println(" cm");
  delay(1000);
}
```

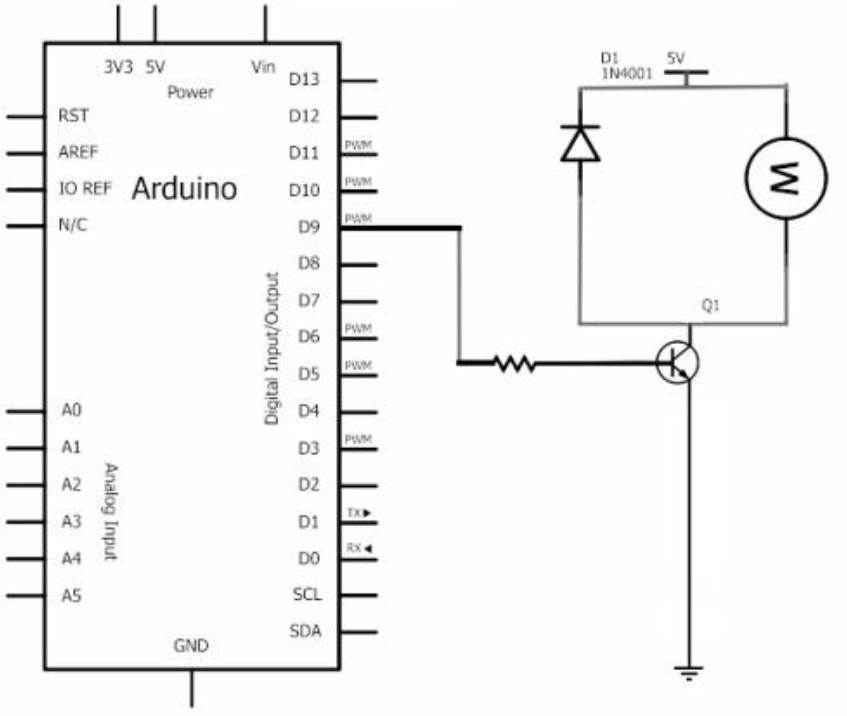
*(Función para medir longitud entrante). Mide el tiempo que transcurrido entre el envío
Del pulso ultrasónico y cuando el sensor recibe el rebote, es decir: desde que el pin 12 empieza a recibir el rebote, HIGH, hasta que
Deja de hacerlo, LOW, la longitud del pulso entrante
Distancia= int(0.017*tiempo); /*fórmula para calcular la distancia obteniendo un valor entero

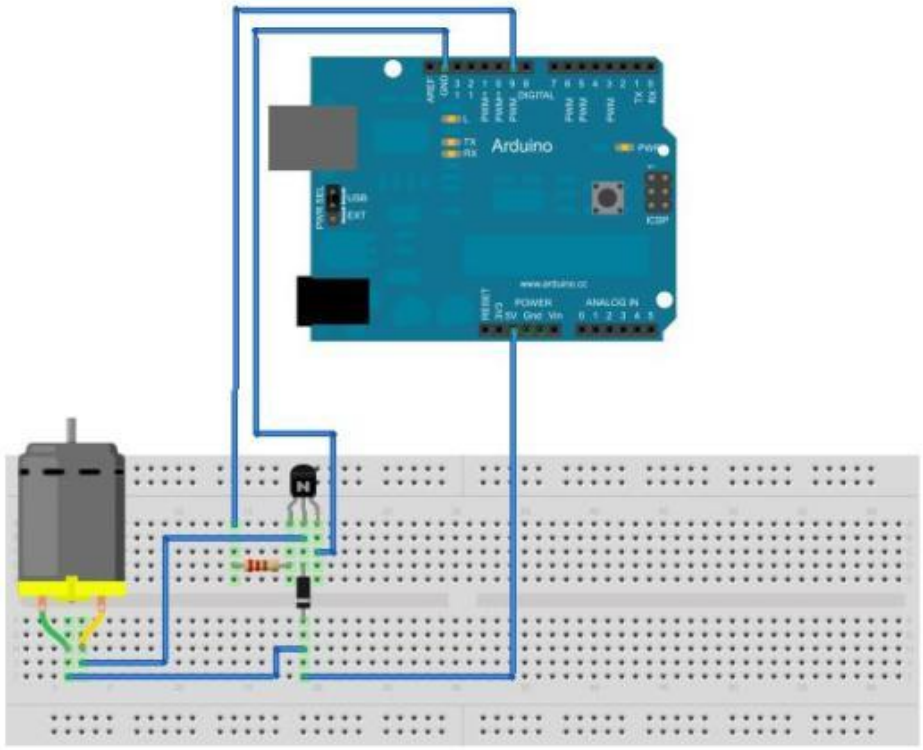
ACTUADORES
MOTOR CONTROLADO POR ARDUINO

A continuación podéis observar el programa, en lenguaje C de programación, que ejecutaremos a la hora de poner en funcionamiento el circuito de control de un motor DC.

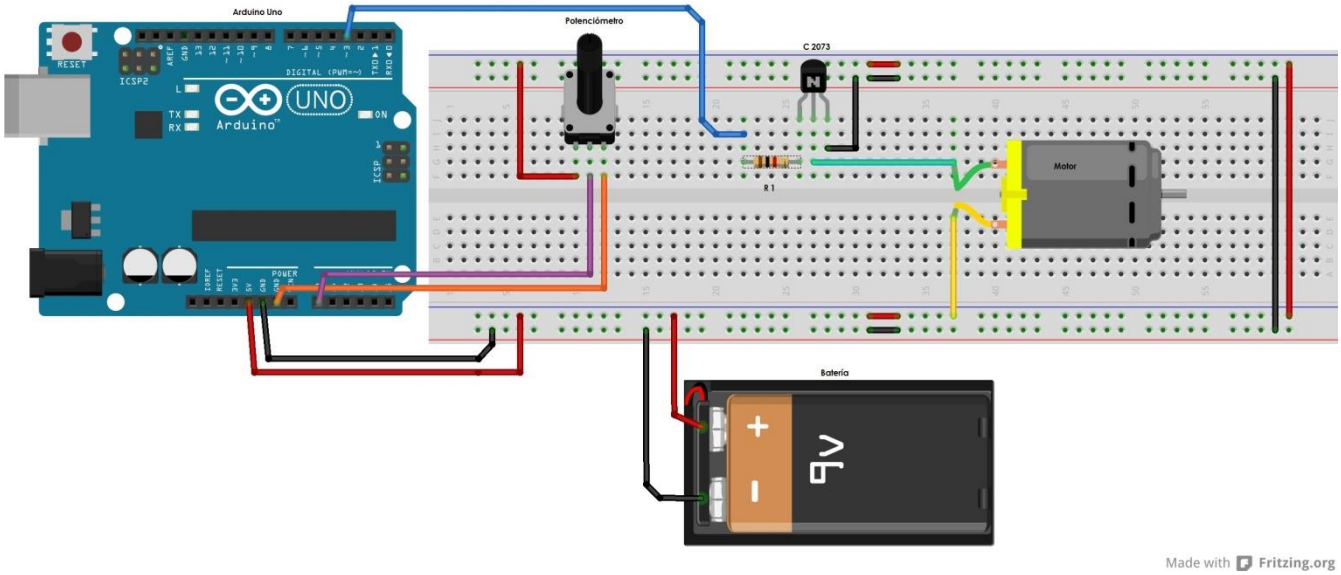
```
    const int transistorPin = 9;          // conectado a la base del transistor
void setup() {

    pinMode(transistorPin, OUTPUT);
}
void loop() {                             // loop= repetir esto constantemente

    digitalWrite(transistorPin, HIGH);    // enciende el motor
    delay(2000);                          // durante 2 segundos
    digitalWrite(transistorPin, LOW);     // apaga el motor
    delay(2000);                          // durante 2 segundos
}
```

Una vez obtenido el programa, elaboramos el montaje del circuito mediante el programa **"Fritizing"**.
Podemos ver dos imágenes, la primera correspondiente al esquema eléctrico y la segunda a la protoboard.





Control de Velocidad de Motor DC por Potenciómetro

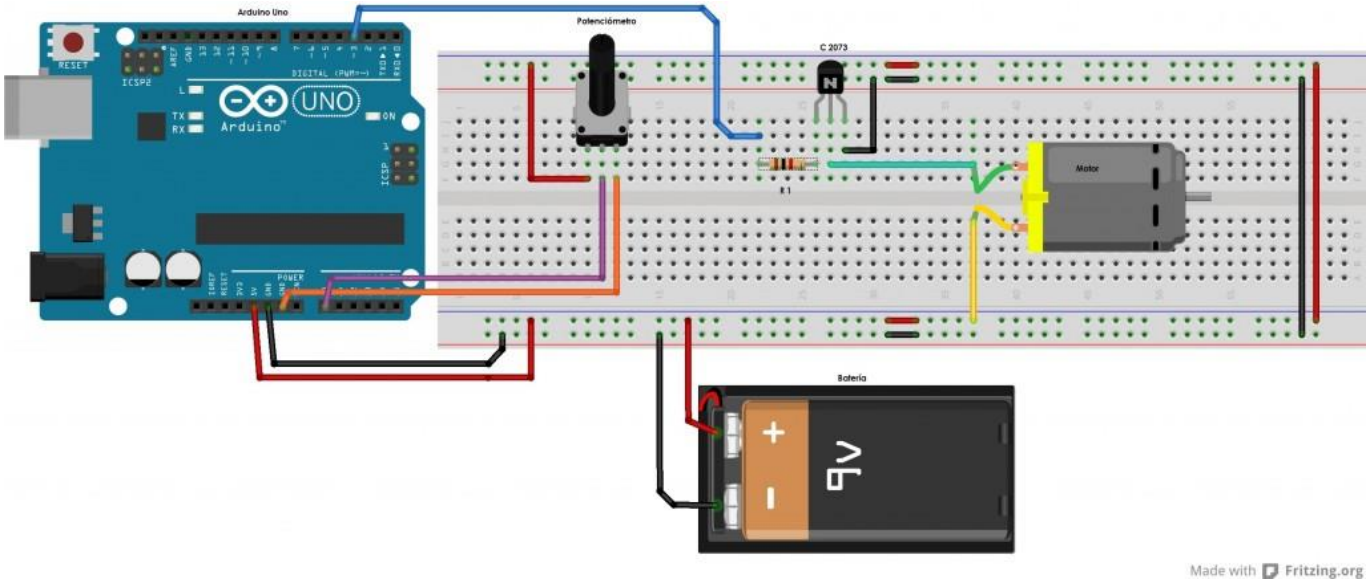


La importancia del Control de velocidad de Motor DC por potenciómetro radica en que el usuario podrá regular directamente la velocidad del motor DC, ajustándolo así a sus necesidades.

Los componentes utilizados en esta experiencia fueron:

- 1 Batería de 9 Voltios
- 1 Transistor NPN (C2073) o similar
- Jumpers
- 1 Resistencia de 1 kΩ
- 1 Motor DC
- 1 Arduino Uno
- 1 Protoboard
- 1 Potenciómetro 1KΩ a 10KΩ

Para esta experiencia se empleó la siguiente configuración:



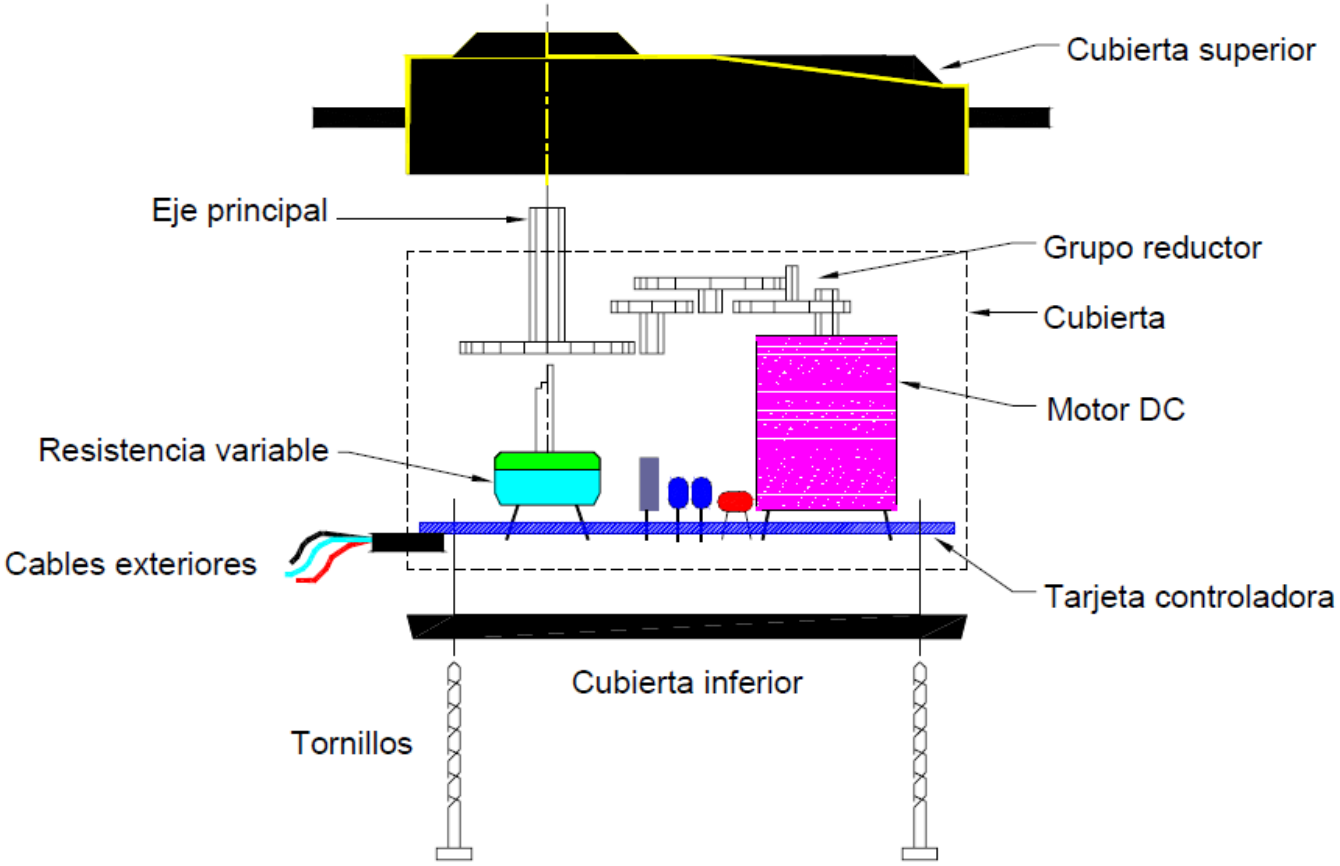
En esta experiencia se utilizó el potenciómetro, debido a que es un circuito con baja corriente, ya que este disipa poca potencia.
Los extremos del potenciómetro se conectan al +5V y a GND. El punto central, se conecta a una entrada analógica, en este caso (A0).
Como recordamos los pines analógicos en Arduino son manejados por un convertidor analógico/digital de 10 bits, por lo que entregan a su salida, valores entre 0 y 1023. De esta manera la tensión que entrega el potenciómetro a la entrada analógica, en la función ***analogRead***, variará entre 0 (cuando esté a 0V) y 1023 (cuando esté a 5V).

Programa

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

float niv=analogRead(A0)/4;
analogWrite(3, niv);
Serial.println (niv);
 delay(1000);

}
```

Servomotores con arduino

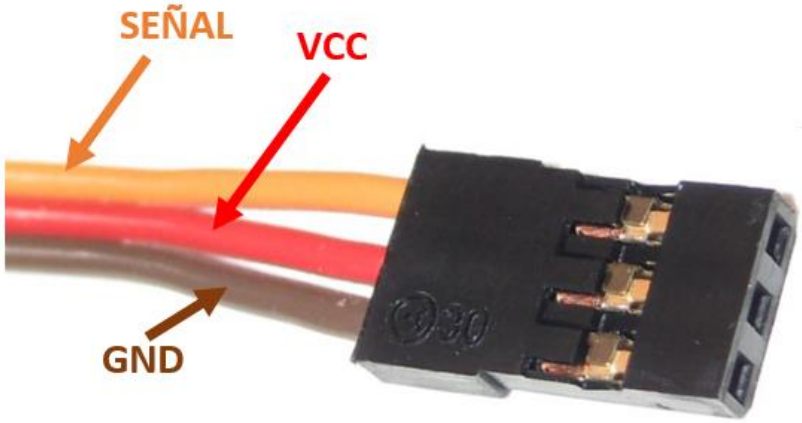
Un servomotor o comúnmente llamado servo, es un motor DC con la capacidad de ubicar su eje en una posición o ángulo determinado, internamente tiene una caja reductora la cual le aumenta el torque y reduce la velocidad, un potenciómetro encargado de censar la posición del eje y una pequeña placa electrónica que junto al potenciómetro forman un control de lazo cerrado.



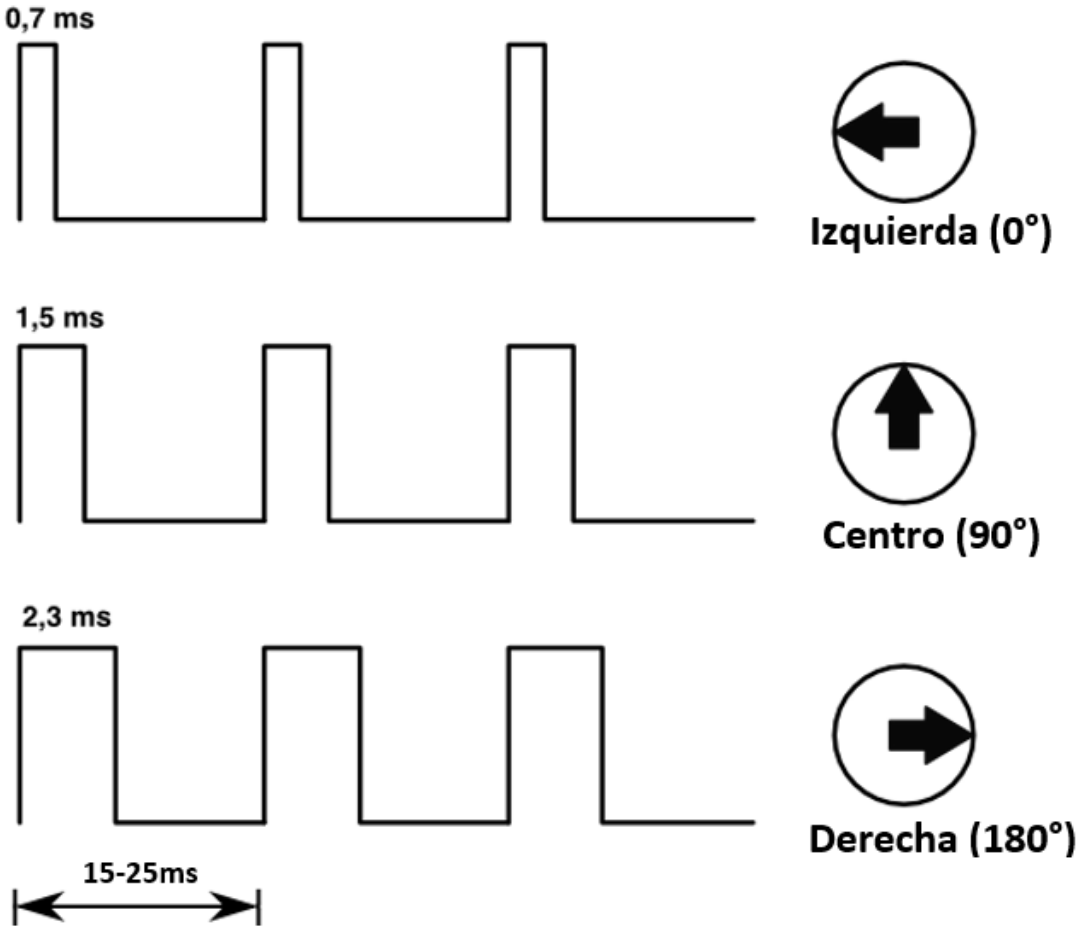
En el mercado existen diferentes modelos de servomotores, la principal diferencia entre ellos es el torque. Este punto es bueno tenerlo claro para elegir el servomotor adecuado. Es mejor elegir un servo con torque superior al que requerimos, pues el consumo de corriente es proporcional a la carga. En cambio sí sometemos un servomotor a cargas superiores a su torque, corremos el riesgo de malograr tanto la parte mecánica (engranes) y eléctrica del servo, incluso podemos generar ruido en la fuente.

Servomotor	Modelo	Voltaje	Torque
	SG90	3V-7.2V	1-1.6 Kg/cm
	SG-5010	4.8V-6V	3.5K-6.5 Kg/cm
	MG995	4.8V – 7.2V	8.5-10 kg/cm
	MG946	4.8V – 7.2V	10.5-14 Kg/cm
	RDS3115	5V – 7.2V	13.5-15Kg/cm

Todos los servos usados para robótica, tiene un conector de 3 cables. VCC (rojo), GND (Marrón) y Señal (Naranja) por lo general esto puede variar según la marca y modelo en algunas ocasiones.



La señal o dato que hay que enviarle al servo es una señal de PWM donde el tiempo en alto es equivalente al ángulo o posición del servo. Estos valores pueden variar y van desde 0.5 a 1 milisegundo para la posición 0° y 2 a 2.4 milisegundos para la posición de 180°, el periodo de la señal debe ser cercano a 20 milisegundos.

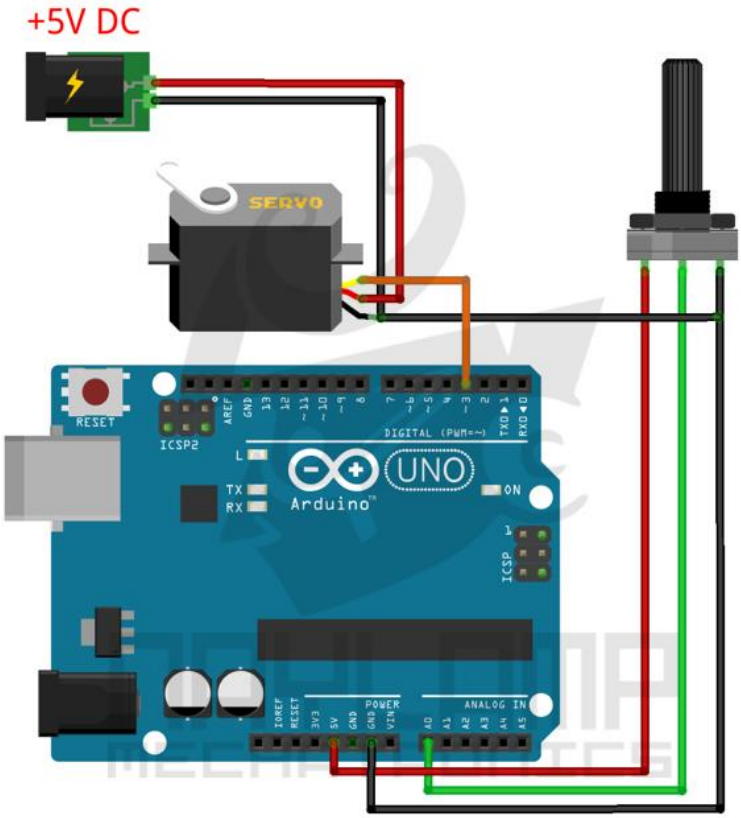


Librería servo de Arduino

El IDE Arduino trae una librería completa para el control de servomotores, la documentación completa lo encontramos en su página oficial.
Expliquemos brevemente las funciones de la librería.
La biblioteca Servo admite hasta 12 motores en la mayoría de las placas Arduino y 48 en el Arduino Mega. En las placas que no sean los de Mega, el uso de la biblioteca desactiva la funcionalidad PWM en las patillas 9 y 10. En el Arduino Mega, hasta 12 servos pueden ser utilizados sin interferir con la funcionalidad PWM, pero si se usan de 12 a 23 motores desactivará el PWM en los pines 11 y 12.
A continuación se muestran las funciones de la librería Servo:

attach(Pin): Establece el pin indicado en la variable servo. Ej: *servo.attach(2);*
attach(Pin,min,max): Establece el pin indicado en la variable servo, considerando min el ancho de pulso para la posición 0° y max el ancho de pulso para 180°.Ej: *servo.attach(2,900,2100);*
write(angulo): Envía la señal correspondiente al servo para que se ubique en el ángulo indicado, ángulo es un valor entre 0 y 180°. Ej: *servo.write(45);*
writeMicroseconds(tiempo): Envía al servo el ancho de pulso=tiempo en microsegundos.
Ej: *servo.writeMicroseconds(1500);*
read (): Lee la posición actual del servo, devuelve un valor entre 0 y 180. Ej: *angulo=read () ;*
attached(Pin): Verifica si la variable servo está unido al pin indicado, devuelve true o false.
Ej: *if(attached(2))*
detach(pin): Separa la variable Servo del pin indicado. Ej: *servo. detach(2)*;

Control de un servomotor a través de un potenciómetro:

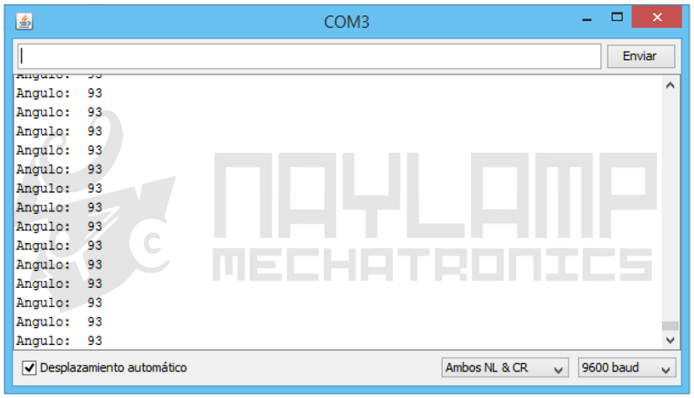


La alimentación del motor puede ser la misma que el Arduino siempre y cuando la fuente soporte la potencia del servo y sea de 5V. Los 5V del USB solo soportan un servo SG90, más servos o de otro tipo se necesita usar una fuente externa.

Una vez hecho las conexiones necesitamos cargar el siguiente programa.

```
#include <Servo.h>
 Servo myservo;                          //creamos un objeto servo

void setup()
{
  myservo.attach(9);                      // asignamos el pin 9 al servo.
   Serial.begin(9600);
}

void loop()
{
  int adc = analogRead(A0);               // realizamos la lectura del potenciómetro
  int angulo = map(adc, 0, 1023, 0, 180); // escalamos la lectura a un valor entre 0 y 180
  myservo.write(angulo);                  // enviamos el valor escalado al servo.
  Serial.print("Angulo:  ");
  Serial.println(angulo);
  delay(10);
}
```

Como se observa en el código, básicamente se hace la lectura del potenciómetro y se lo escala entre valores de 0° y 180° para después enviarlo al servomotor.
Una vez cargado el programa el servo debe moverse cada vez que movemos le potenciómetro, y si abrimos el monitor serial debería mostrar el ángulo del servomotor.



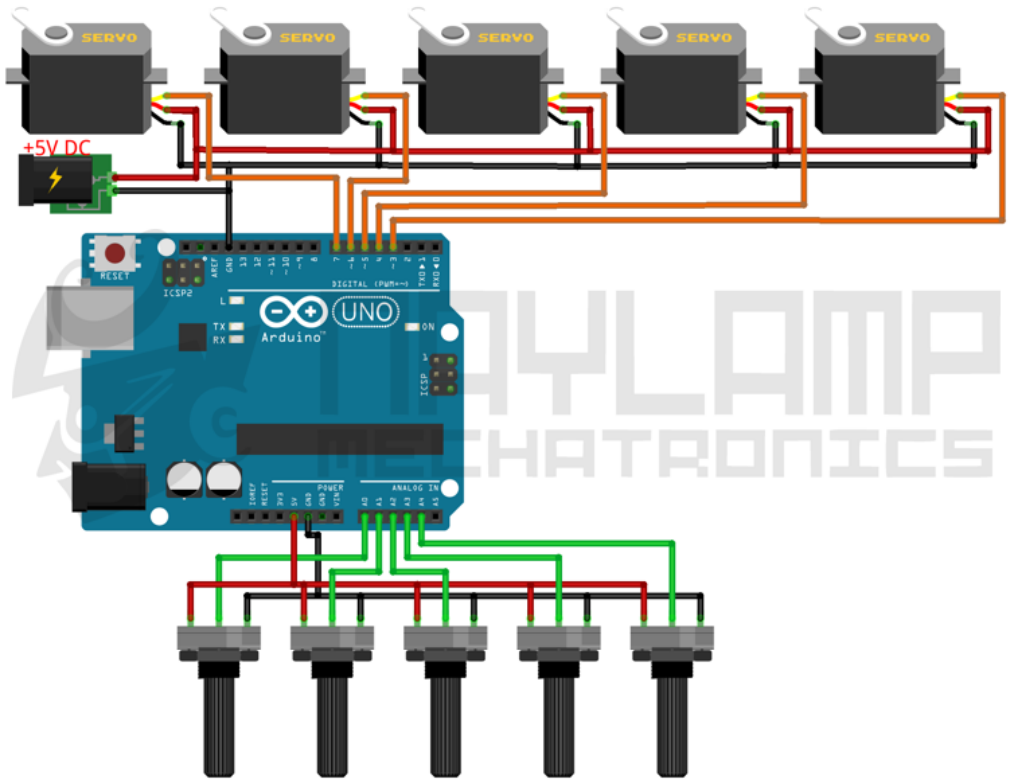
Si la poción del servomotor no coincide con el ángulo correspondiente, es porque el ancho del pulso que maneja la librería no coincide con los del servo, para esto es necesario indicarle el ancho de pulso min(para 0°) y el máximo (para 180°), la única modificación seria en la siguiente línea:

```
myservo.attach(9,900,2100);  // asignamos el pin 9 al servo, 900->0° y 2100->180°
```

Deben cambiar los valores 900 y 2100 hasta lograr las posiciones correctas.

Controlando varios Servomotores con potenciómetros

Este ejemplo es una extensión del ejemplo anterior, las conexiones serían las siguientes.

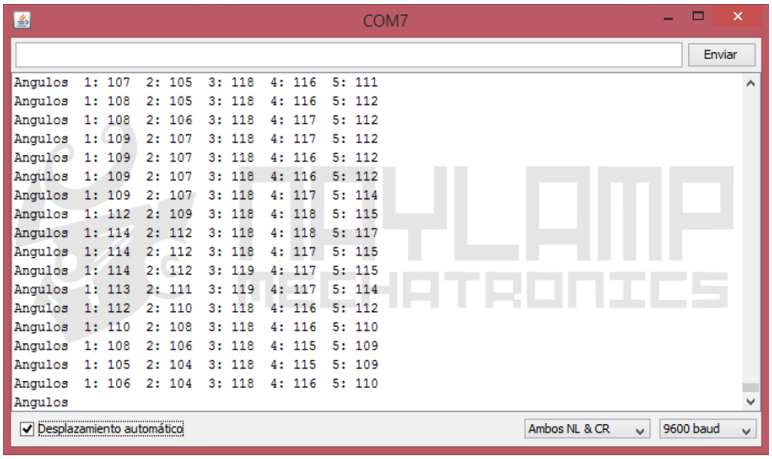


El código se muestra a continuación:

```
#include <Servo.h>

Servo servo1;                 //creamos un objeto servo
Servo servo2;                 //creamos un objeto servo
Servo servo3;                 //creamos un objeto servo
Servo servo4;                 //creamos un objeto servo
Servo servo5;                 //creamos un objeto servo
void setup()
{
  servo1.attach(3);          // asignamos el pin al servo.
  servo2.attach(4);          // asignamos el pin al servo.
  servo3.attach(5);          // asignamos el pin al servo.
  servo4.attach(6);          // asignamos el pin al servo.
  servo5.attach(7);          // asignamos el pin al servo.
   Serial.begin(9600);
}
```

```
void loop()
{
  int angulo1 = map(analogRead(A0), 0, 1023, 0, 180);        // escalamos la lectura a un valor entre 0 y 180
  int angulo2 = map(analogRead(A1), 0, 1023, 0, 180);        // escalamos la lectura a un valor entre 0 y 180
  int angulo3 = map(analogRead(A2), 0, 1023, 0, 180);        // escalamos la lectura a un valor entre 0 y 180
  int angulo4 = map(analogRead(A3), 0, 1023, 0, 180);        // escalamos la lectura a un valor entre 0 y 180
  int angulo5 = map(analogRead(A4), 0, 1023, 0, 180);        // escalamos la lectura a un valor entre 0 y 180

  servo1.write(angulo1);                                     // enviamos el valor escalado al servo.
  servo2.write(angulo2);                                     // enviamos el valor escalado al servo.
  servo3.write(angulo3);                                     // enviamos el valor escalado al servo.
  servo4.write(angulo4);                                     // enviamos el valor escalado al servo.
  servo5.write(angulo5);                                     // enviamos el valor escalado al servo.

  Serial.print("Angulos  1: ");
  Serial.print(angulo1);
  Serial.print("  2: ");
  Serial.print(angulo2);
  Serial.print("  3: ");
  Serial.print(angulo3);
  Serial.print("  4: ");
  Serial.print(angulo4);
  Serial.print("  5: ");
  Serial.println(angulo5);
  delay(10);
}
```

Como se observa es similar al primer ejemplo, se necesita declarar una variable (objeto) servo para cada servomotor.
Si abrimos el monitor serial deben de visualizarse los ángulos correspondientes para cada servo



Se puede eliminar las últimas líneas de código si no necesitan visualizar los ángulos a través del monitor serial.

MOVER MOTORES PASO A PASO CON ARDUINO

Motor pasó a paso

Un motor paso a paso (también llamado stepper) es un dispositivo electromagnético que convierte impulsos eléctricos en movimientos mecánicos de rotación. La principal característica de estos motores es que se mueven un paso por cada impulso que reciben. Normalmente los pasos pueden ser de 1,8º a 90º por paso, dependiendo del motor. Por ejemplo: un motor paso a paso que se mueve 2º cada paso, quiere decir que para completar una vuelta (360º) tendrá que dar (360º/2ºpor paso) 180 pasos.
Son motores con mucha precisión, que permiten quedar fijos en una posición (como un servomotor) y también son capaces de girar libremente en un sentido u otro (como un motor DC).

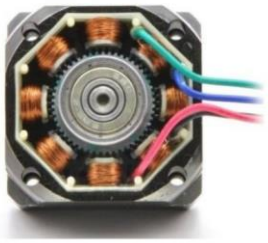
Hay tres tipos de motores paso a paso:

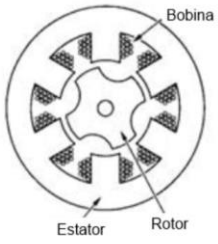
-Motor pasó a paso de imán permanente (los que usaremos).
-Motor pasó a paso de reluctancia variable.
-Motor paso a paso hibrido.

Los Motores paso a paso están formados por dos partes:

-El **estator** es la parte fija del motor donde sus cavidades van depositadas las bobinas.
-El **rotor** es la parte móvil del motor construido por un imán permanente.

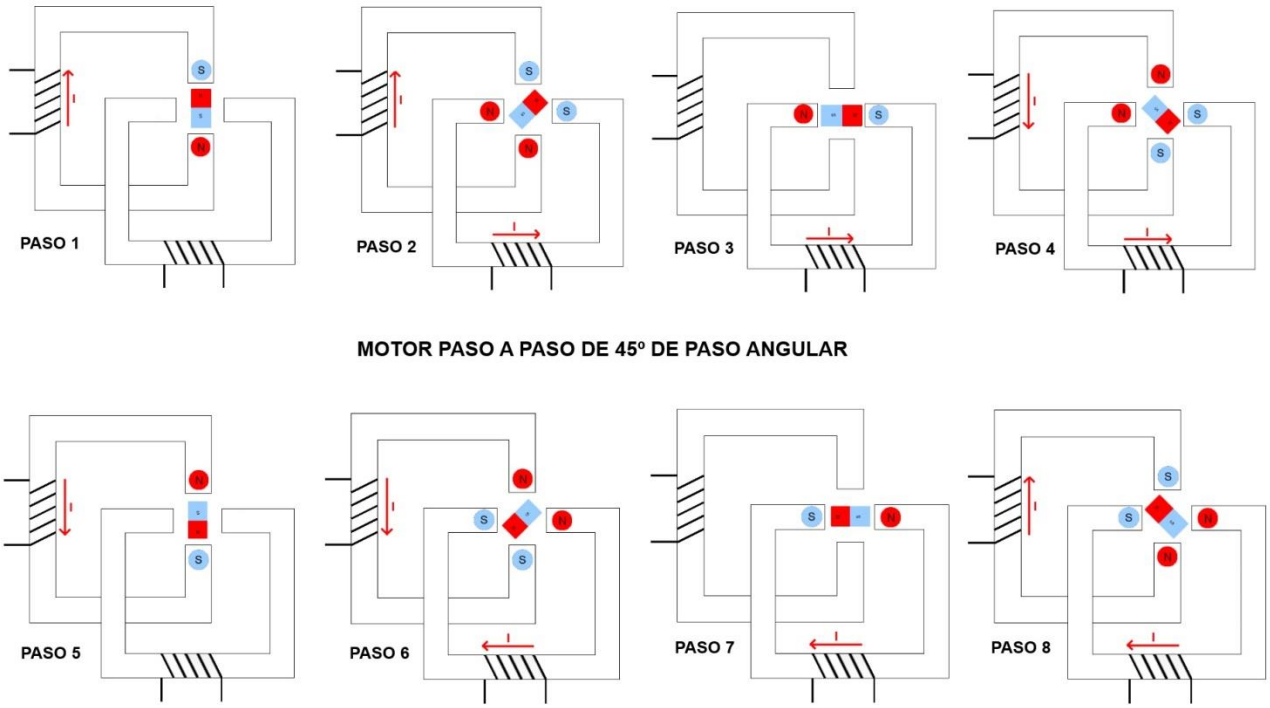
Estas dos partes van montadas sobre un eje.





Funcionamiento

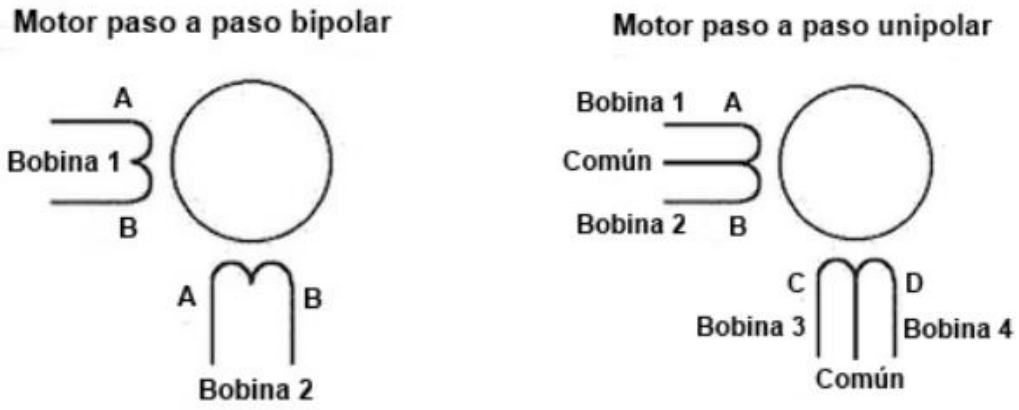
Cuando circula corriente por una o más bobinas del estator se crea un campo magnético creando los polos Norte-Sur. Luego el rotor se equilibrará magnéticamente orientando sus polos Norte-Sur hacia los polos Sur-Norte del estator. Cuando el estator vuelva a cambiar la orientación de sus polos a través de un nuevo impulso recibido hacia sus bobinas, el rotor volverá a moverse para equilibrarse magnéticamente. Si se mantiene esta situación, obtendremos un movimiento giratorio permanente del eje. El ángulo de paso depende de la relación entre el nombre de polos magnéticos del estator y el nombre de polos magnéticos del rotor.



MOTOR PASO A PASO DE 45º DE PASO ANGULAR

Dentro de la anterior clasificación hay dos tipos de motores paso a paso: los unipolares y los bipolares.

Los bipolares se componen de 2 bobinas y los unipolares de 4 bobinas. Para diferenciarlos físicamente basta con observar el número de terminales de cada motor. Los bipolares siempre tienen 4 terminales, dos para cada bobina, y los unipolares normalmente tienen 6 terminales, dos para cada bobina y los otro dos son los comunes de estas. Hay motores unipolares con 5 terminales en que los dos comunes están unidos internamente.
La diferencia entre los dos es que un motor paso a paso unipolar se activa una bobina a la vez mientras que un motor bipolar se activa más de una bobina a la vez. Esto hace que un motor bipolar tenga más torque que un motor unipolar. Por contra, un motor bipolar es más complejo de controlar que un unipolar.



Secuencia para controlar motores paso a paso Bipolares

Un motor paso a paso bipolar necesita invertir la corriente que circula por sus bobinas en una secuencia determinada para provocar el movimiento del eje.

Paso	Bobina 1A	Bobina 1B	Bobina 2A	Bobina 2B
Paso 1	1	0	1	0
Paso 2	1	0	0	1
Paso 3	0	1	0	1
Paso 4	0	1	1	0

Secuencia para controlar motores paso a paso Unipolares

Hay tres secuencias para controlar los motores paso a paso unipolares
Simple o wave drive: Es una secuencia donde se activa una bobina a la vez. Esto hace que el motor tenga un paso más suave pero por el contrario tenga menos torque y menos retención.

Paso	Bobina A	Bobina B	Bobina C	Bobina D
Paso 1	1	0	0	0
Paso 2	0	1	0	0
Paso 3	0	0	1	0
Paso 4	0	0	0	1

Normal: Es la secuencia más usada y la que recomiendan los fabricantes. Con esta secuencia el motor avanza un paso por vez y siempre hay dos bobinas activadas. Con esto se obtiene un mayor torque y retención.

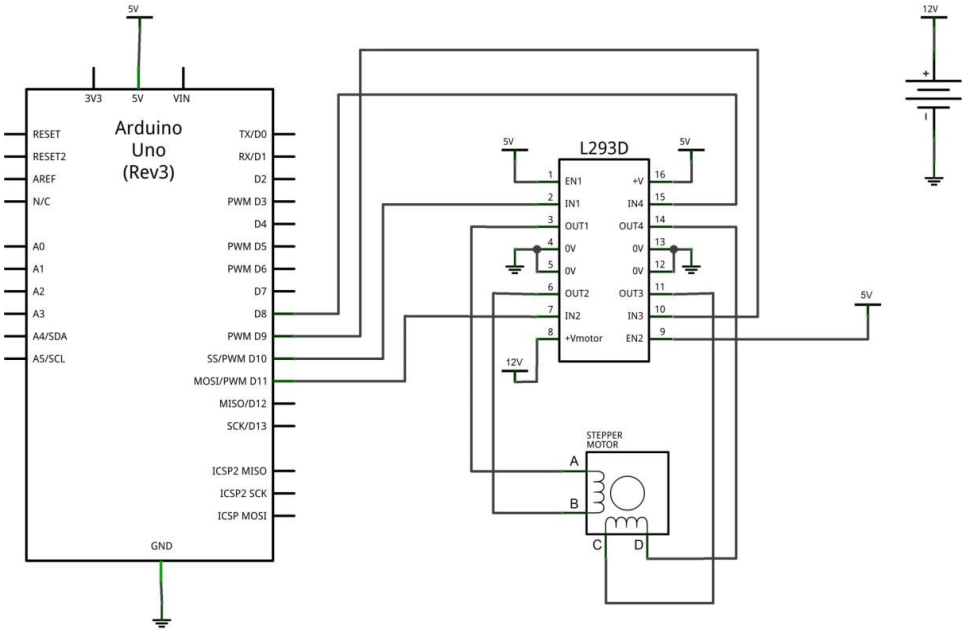
Paso	Bobina A	Bobina B	Bobina C	Bobina D
Paso 1	1	1	0	0
Paso 2	0	1	1	0
Paso 3	0	0	1	1
Paso 4	1	0	0	1

Medio paso: Se activan primero dos bobinas y después solo una y así sucesivamente. Esto provoca que el motor avance la mitad del paso real. Esto se traduce en un giro más suave y preciso.

Paso	Bobina A	Bobina B	Bobina C	Bobina D
Paso 1	1	0	0	0
Paso 2	1	1	0	0
Paso 3	0	1	0	0
Paso 4	0	1	1	0
Paso 5	0	0	1	0
Paso 6	0	0	1	1
Paso 7	0	0	0	1
Paso 8	1	0	0	1

Controlar motor paso a paso Bipolar con Arduino

Como hemos explicado antes, los motores bipolares son más complejos de controlar ya que el flujo de corriente tiene que cambiar de dirección a través de las bobinas con una secuencia determinada. Para esto debemos conectar cada una de las dos bobinas en un puente en H (H-Bridge). Para esto, utilizaremos el integrado L293 que contiene dos H-Bridge.



Para controlar motores paso a paso con Arduino, utilizaremos la librería ***<Stepper.h>*** que viene con el software de este.

```
#include <Stepper.h>           //Importamos la librería para controlar motores paso a paso

#define STEPS 200       //Ponemos el número de pasos que necesita para dar una vuelta. 200 en este caso
                                                // Ponemos nombre al motor, el número de pasos y los pines de control
Stepper stepper(STEPS, 8, 9, 10, 11);       //Stepper nombre motor (número de pasos por vuelta, pines de control

void setup()
{
  stepper.setSpeed(100);                    // Velocidad del motor en RPM
}

void loop()
{
                                             //Girar una vuelta entera en un sentido
  stepper.step(200);
  delay(500);                                //pausa de medio segundo

                                            //Girar una vuelta entera en sentido contrario
  stepper.step(-200);
  delay(500);                               //pausa de medio segundo
}
```

Utilizando esta conexión tenemos un pequeño problema, y es que usamos cuatro pines del Arduino para controlar el motor, y eso son muchos. Si queremos controlar más de un motor paso a paso y usar otros pines para otras conexiones, nos quedaremos cortos. Esto se puede solucionar con una conexión diferente que solo utiliza conexión a dos pines de control del Arduino.

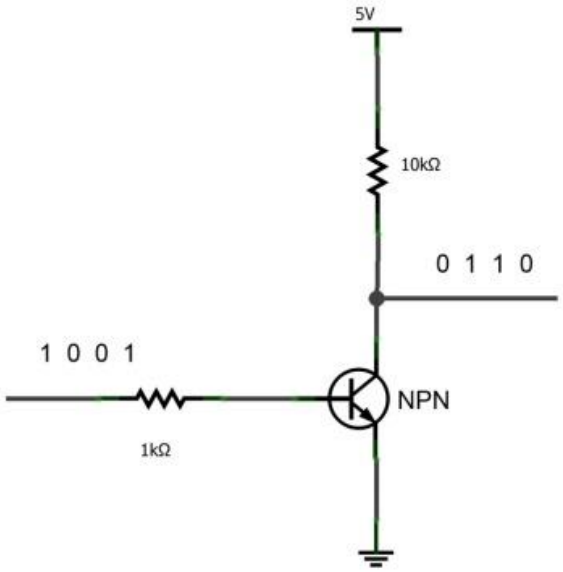
Conexión a dos pines

Para entender esta conexión, primero hay que fijarse en la secuencia del motor paso a paso bipolar que hemos visto antes. Si nos fijamos bien, veremos que los extremos de cada bobina están invertidos entre sí.

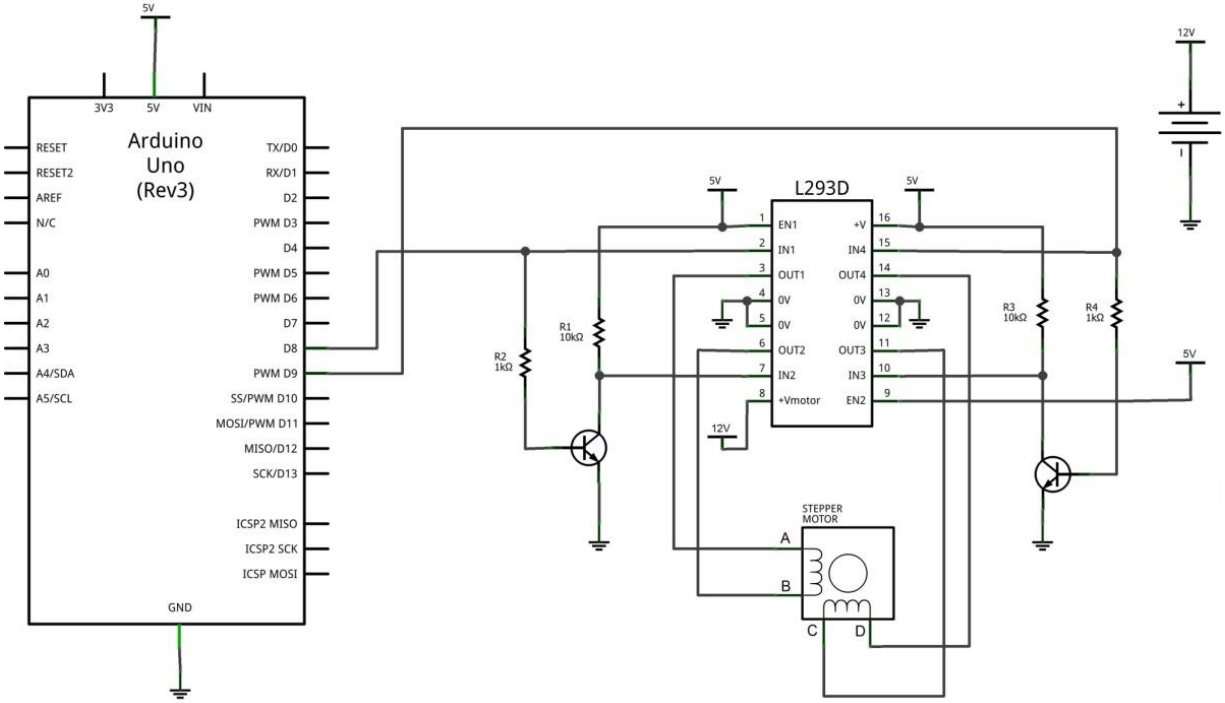
Paso	Bobina 1A	Bobina 1B	Bobina 2A	Bobina 2B
Paso 1	1	0	1	0
Paso 2	1	0	0	1
Paso 3	0	1	0	1
Paso 4	0	1	1	0

Invertidos Invertidos

De esta manera solo tenemos que usar dos pines de control e invertirlos. Para invertirlos usaremos dos transistores NPN (BC547, BD139, o alguno similar polarizados en emisor común y que trabajen en modo saturación-corte. Así el transistor trabaja como un inversor de señal.



De este modo, usaremos solo dos pines del Arduino y con los dos NPN en modo de inversor obtendremos los cuatro pines de control necesarios para activar el L293.

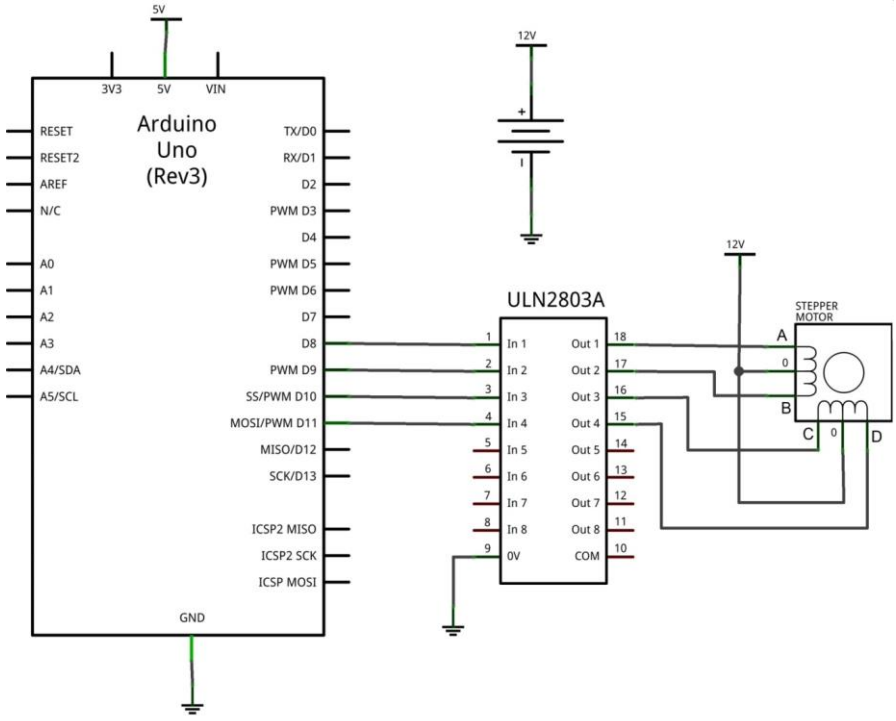


El programa es el mismo que antes, con la diferencia que cambiaremos los pines de control:
En vez de usar:
Stepper stepper(STEPS, 8, 9,10,11);
Cambiamos por:
Stepper stepper(STEPS, 8, 9);

```
#include <Stepper.h>                    //Importamos la librería para controlar motores paso a paso
#define STEPS 200                       //Ponemos el número de pasos que necesita para dar una vuelta. 200 en nuestro caso

                                        // Ponemos nombre al motor, el número de pasos y los pines de control
Stepper stepper(STEPS, 8, 9);
                                        //Stepper nombre motor (número de pasos por vuelta, pines de control)
void setup()
{
                                         // Velocidad del motor en RPM
  stepper.setSpeed(100);
}
void loop()
{
                                        //Girar una vuelta entera en un sentido
  stepper.step(200);
  delay(500);                           //pausa de medio segundo
                                        //Girar una vuelta entera en sentido contrario
  stepper.step(-200);
  delay(500);                           //pausa de medio segundo
}
```

Controlar motor paso a paso Unipolar con Arduino

Un motor paso a paso unipolar es más sencillo que controlar. Utilizaremos el integrado ULN2803 que es un array de 8 transistores tipo Darlington capaz de soportar cargas de hasta 500 mA. Conectaremos los cuatro pines del Arduino a las entradas del ULN2803 y las salidas de este a las bobinas. Los comunes a 12V.

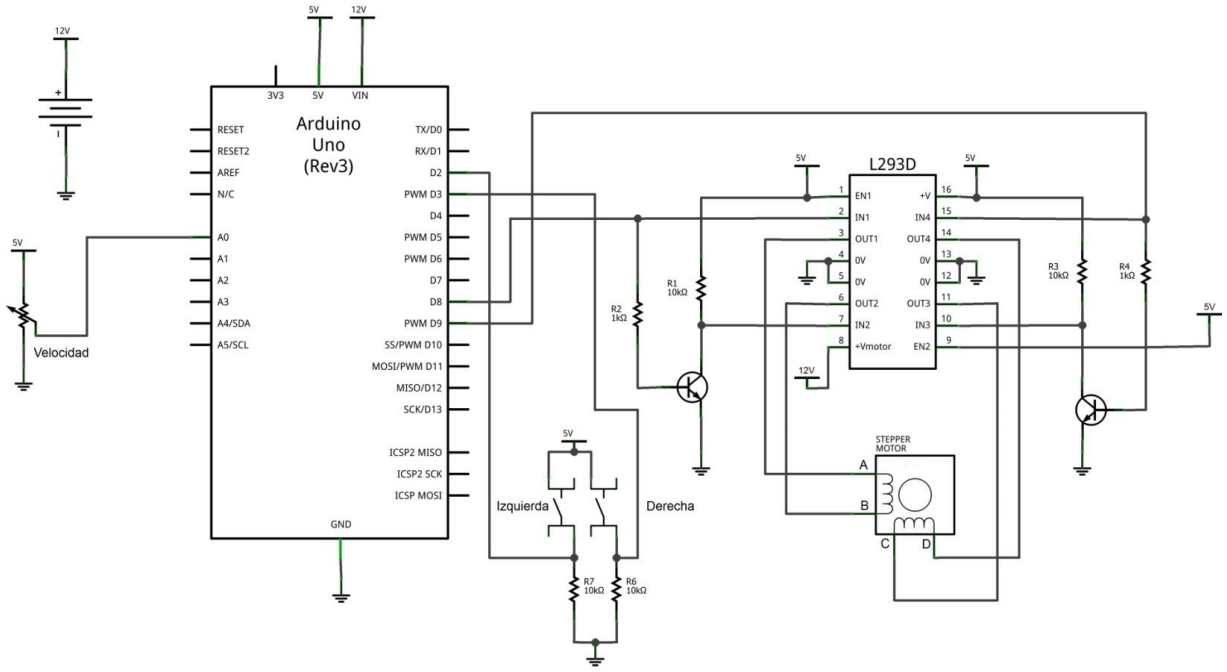


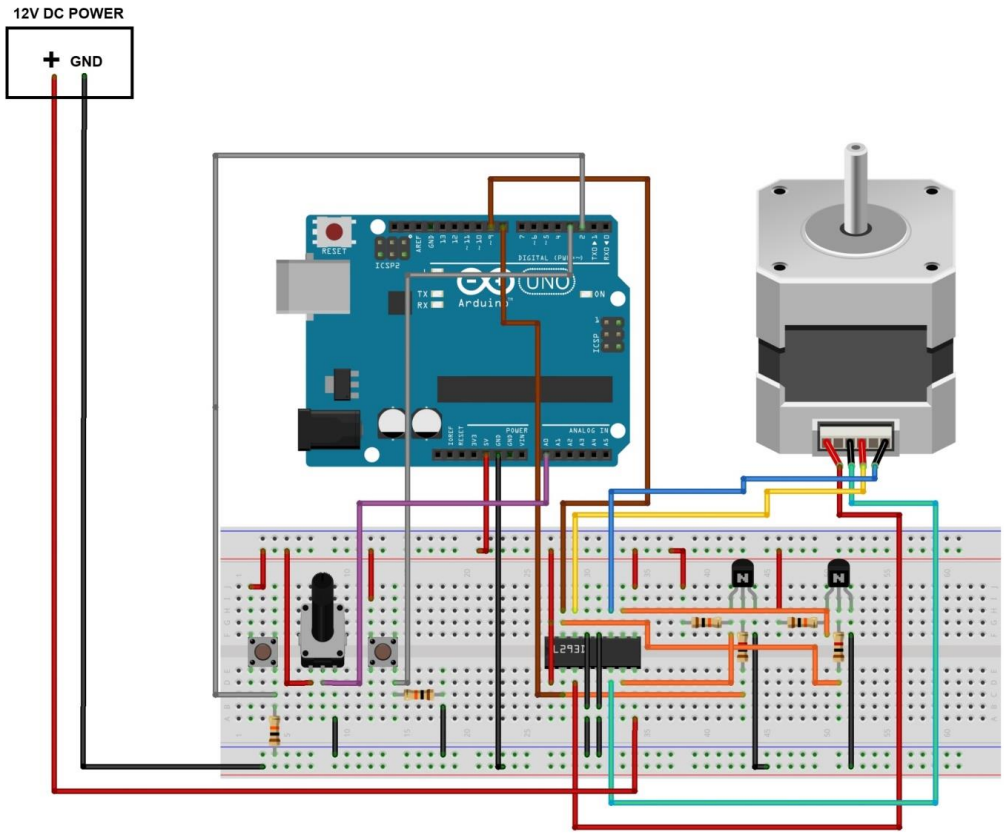
El programa es el mismo que hemos usado antes con el motor bipolar de 4 pines:

```
#include <Stepper.h>                    //Importamos la librería para controlar motores paso a paso
#define STEPS 200       //Ponemos el número de pasos que necesita para dar una vuelta. 200 en nuestro caso
                                        // Ponemos nombre al motor, el número de pasos y los pines de control
Stepper stepper(STEPS, 8, 9, 10, 11);   //Stepper nombre motor (número de pasos por vuelta, pines de control)

void setup()
{
                                        // Velocidad del motor en RPM
  stepper.setSpeed(100);
}
void loop()
{
                                        //Girar una vuelta entera en un sentido
  stepper.step(200);
  delay(500);                           //pausa de medio segundo
                                        //Girar una vuelta entera en sentido contrario
  stepper.step(-200);
  delay(500);                           //pausa de medio segundo
}
```

Controlar velocidad y sentido de un motor paso a paso Bipolar con Arduino

Ahora que ya sabemos las diferentes maneras y conexiones de controlar los motores paso a paso, vamos a hacer un pequeño proyecto con un motor bipolar. Con un potenciómetro regularemos la velocidad de este, y con dos pulsaremos seleccionaremos el sentido de giro.





-Ajuntar las masas del Arduino y de la fuente de alimentación.
-Los dos pulsadores conectados en modo Pull-Down.
-Para detectar las bobinas, basta con conectar el tester en continuidad en dos terminales, cuando marque continuidad o una baja carga óhmica quiere decir que entre esos dos terminales hay una bobina.
-Un **consejo:** mejor no alimentar el Arduino solo con el cable USB, sino con la fuente de alimentación a través del pin **Vin** o del conector **jack**. Esto es para que no haya errores, sobre todo a la hora de controlar motores. El cable USB solo conectarlo cuando se use la comunicación serie con el PC. Si no es así, desconectarlo.

Programa

```
#include <Stepper.h>                    //Importamos la librería para controlar motores paso a paso
#define STEPS 200                       //Ponemos el número de pasos que necesita para dar una vuelta 200 en nuestro caso

                                        // Ponemos nombre al motor, el número de pasos y los pins de control
Stepper stepper(STEPS, 8, 9);           //Stepper nombre motor  (número de pasos por vuelta, pins de control)

int pot;                                //Variable lectura potenciómetro
int derecha=3;                          //Pulsador derecha
int izquierda=2;                        //Pulsador izquierda
int direccion;                          //Variable para indicar la dirección
```

```
void setup()
{
  pinMode(derecha,INPUT);
  pinMode(izquierda,INPUT);
}

void loop()
{
  pot=analogRead(A0);                          //Lectura potenciómetro
  pot = map(pot, 0, 1023, 30, 150);
                                               //Establecemos la velocidad entre 30 y 150 rpm

  stepper.setSpeed(pot);                       //Indicamos la velocidad al motor
  stepper.step(direccion);                     //Indicamos la dirección al motor

  if(digitalRead(izquierda)==HIGH)
  {
    direccion=200;                              //Si pulsamos el pulsador izquierdo, el motor gira a la izquierda
  }

  if(digitalRead(derecha)==HIGH)
  {
    direccion=-200;                            //Si pulsamos el pulsador derecho, el motor gira a la derecha
  }
}
```

Con esta introducción a lo que es los dispositivos arduino que serán utilizados en el curso de robótica educativa los actuadores más usuales; los que son de uso común con ejemplos de programación y principios de funcionamientos. El fin de esta introducción es la de poder tener los conocimientos y la práctica previa para comprender el funcionamiento y la lógica de los autómatas a realizar.