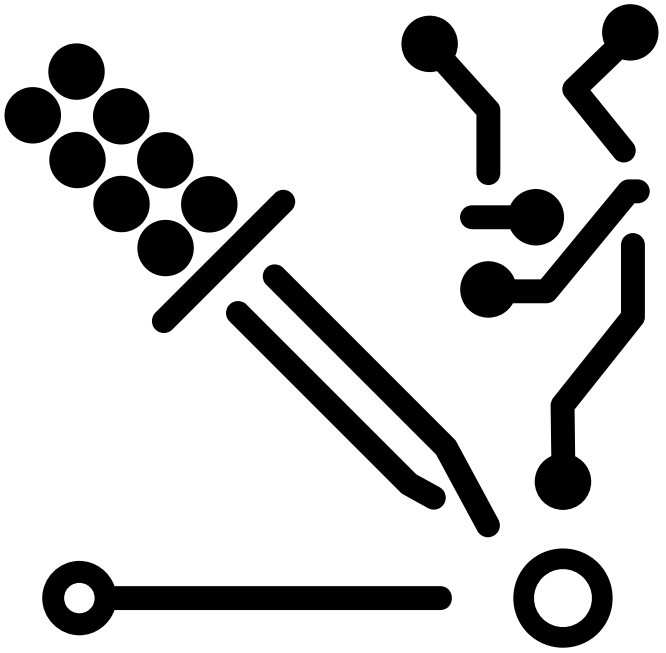


TALLER DE ROBÓTICA CON ARDUINO

FUNCIONES

Una función encapsula código que realiza una tarea especifica permitiendo:

- Diseñar e implementar código de forma independiente.
- Ordenar el código.
- Facilitar el mantenimiento del código.
- Evitar repetir las mismas instrucciones una y otra vez.

FUNCIONES ÚTILES

millis() → Devuelve el número de milisegundos transcurridos desde que la placa Arduino comenzó a ejecutar el programa actual. Este número se desbordará (volverá a cero) después de aproximadamente 50 días.

```
unsigned long tiempo = millis();
```

»»» millis()
»»» Ejemplo retardo no bloqueante

FUNCIONES ÚTILES

micros() → Devuelve el número de microsegundos desde que la placa Arduino comenzó a ejecutar el programa actual. Este número se desbordará (volverá a cero) después de aproximadamente 70 minutos. En placas Arduino de 16MHz esta función tiene una resolución de cuatro microsegundos (es decir, el valor devuelto es siempre un múltiplo de cuatro). En placas Arduino de 8MHz, esta función tiene una resolución de ocho microsegundos.

```
unsigned long tiempo = micros();
```

»»» micros()

FUNCIONES ÚTILES

random() → La función aleatoria genera números pseudoaleatorios.

```
long numero1 = random(max);
long numero2 = random(min, max);
```

Donde:

- min: límite inferior del valor aleatorio, incluido (opcional).
- max: límite superior del valor aleatorio, exclusivo.

»»»» random()

FUNCIONES ÚTILES

randomSeed() → Inicializa el generador de números pseudoaleatorios, haciendo que comience en un punto arbitrario en su secuencia aleatoria. Esta secuencia, aunque muy larga y aleatoria, es siempre la misma. Use randomSeed() junto a analogRead() en un pin no conectado para inicializar el generador de números aleatorios con una entrada bastante aleatoria. Por el contrario, en caso de utilizar secuencias pseudoaleatorias que se repiten exactamente deberá llamar a randomSeed() con un número fijo.

```
long aleatorio;

void setup() {
    Serial.begin(9600);
    randomSeed(analogRead(0));
    aleatorio = random(500);
}
```

»»» randomSeed()

FUNCIONES ÚTILES

tone() → Genera una onda cuadrada de la frecuencia especificada (en Hz) y un ciclo de trabajo del 50% en un pin. Se puede especificar una duración; de lo contrario, continúa hasta una llamada a **noTone()**. Se deberá considerar lo siguiente:

- Solo se puede generar un tono a la vez.
- El uso de la función tone() interferirá con la salida PWM en los pines 3 y 11.
- No es posible generar tonos inferiores a 31Hz.

```
tone(pin, frecuencia)
tone(pin, frecuencia, duración)
```

La duración del tono es opcional y en milisegundos.

»»» tone()
»»» noTone()

IMPLEMENTAR UNA FUNCIÓN

Al crear funciones debemos poner atención a la función en sí y en la forma en que interactúa con otras funciones, como **loop()**. Esto incluye transmitir datos en forma correcta a una función cuando es invocada y devolver valores de una función.
Una función se invoca, o utiliza, dando el nombre de la función y transmitiéndole datos, como argumentos, en el paréntesis que sigue al nombre de la función.
La función invocada debe ser capaz de aceptar los datos que le son transmitidos por la función que hace la llamada. Sólo después que la función invocada recibe con éxito los datos pueden ser manipulados éstos para producir un resultado útil.

PROTOTIPOS DE FUNCIÓN

Antes que una función pueda ser llamada, debe ser declarada la función que hará la llamada. La instrucción de declaración para una función se conoce como un **prototipo de función**. El prototipo de función le indica a la función que llama el tipo de valor que será devuelto formalmente, si es que hay alguno, y el tipo de datos y orden de los valores que la función que llama deberá transmitir a la función llamada. Por ejemplo:

```
void encontrarMax(int, int);
```

declara que la función **encontrarMax()** espera que se le envíen dos valores enteros, y que esta función particular devuelve de manera formal ningún valor (void).

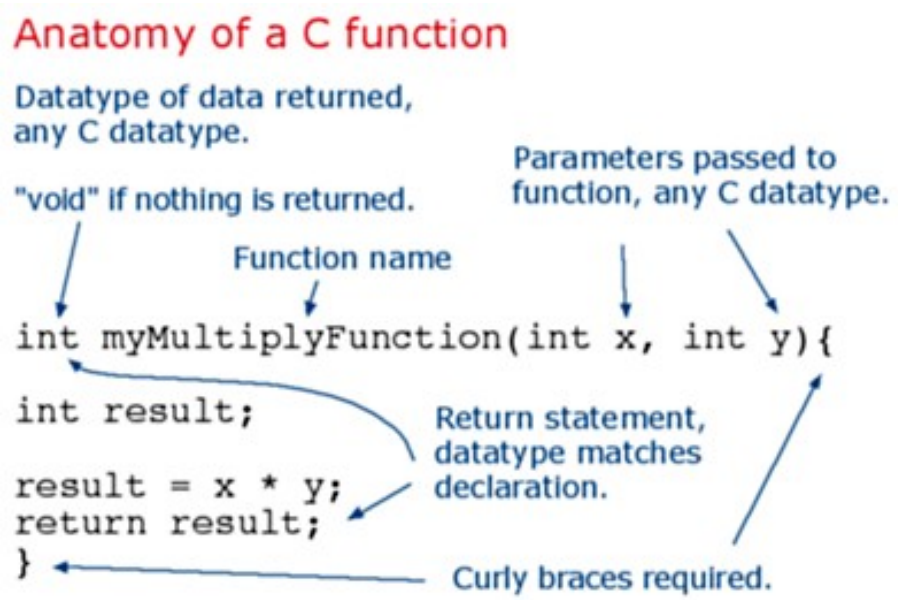
PROTOTIPOS DE FUNCIÓN

La forma general de las instrucciones de prototipo de función es:

tipo-de-datos-a-devolver nombre-de-función (lista de tipos de datos para los argumentos);

Los prototipos de función pueden colocarse con las instrucciones de declaración de variable de la función que llama, encima del nombre de la función que llama, o en un archivo de encabezado separado que se incluirá utilizando una instrucción de preprocesamiento **#include**.

ANATOMÍA DE UNA FUNCIÓN



En una función que devuelve un valor siempre debe tener la instrucción **Return**, este termina una función y devuelve un valor a quien ha llamado a la función.

»»»» return

PARÁMETROS POR VALOR

Pasar parámetros a las funciones **“por valor”** quiere decir que cuando el control pasa a la función, los valores de los parámetros en la llamada se copian a “objetos locales” de la función, estos “objetos” son de hecho los propios parámetros.
Dentro de la función esos parámetros y sus valores son modificados, sin embargo al retornar a la función que la llama, conservan sus valores originales.

PARÁMETROS POR VALOR

```
void loop() {
    int numeroA = 4;
    int numeroB = 10;
    Serial.print(numeroA);
    Serial.print("\t");
    Serial.print(numeroB);
    Serial.print("\t");
    Serial.println(miFuncion(numeroA, numeroB));
}

int miFuncion(int numeroA, int numeroB) {
  numeroA = numeroA + 1;
  numeroB = numeroB - 2;
  return numeroA + numeroB;
}
```

PARÁMETROS POR REFERENCIA

Si queremos que los cambios realizados en los parámetros dentro de la función se conserven al retornar de la llamada, deberemos pasarlos por referencia. Esto se hace declarando los parámetros de la función como referencias a objetos utilizando el operador de referencia (&).

»»» Operador &

PARÁMETROS POR VALOR

```
void loop() {
    int numeroA = 4;
    int numeroB = 10;
    Serial.print(miFuncion(numeroA, numeroB));
    Serial.print("\t");
    Serial.print(numeroA);
    Serial.print("\t");
    Serial.println(numeroB);
}

int miFuncion(int &numeroA, int &numeroB) {
  numeroA = numeroA + 1;
  numeroB = numeroB - 2;
  return numeroA + numeroB;
}
```

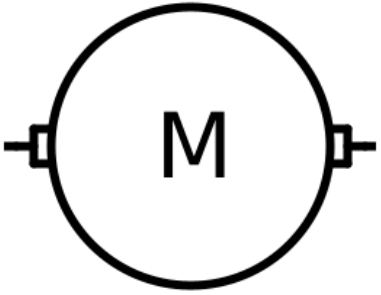
MOTOR C.C.

El **motor de corriente continua** (CC), denominado también motor de corriente directa (DC), es una máquina que convierte energía eléctrica en mecánica, provocando un movimiento rotatorio, gracias a la acción de un campo magnético. Un motor de corriente continua se compone, principalmente, de dos partes:

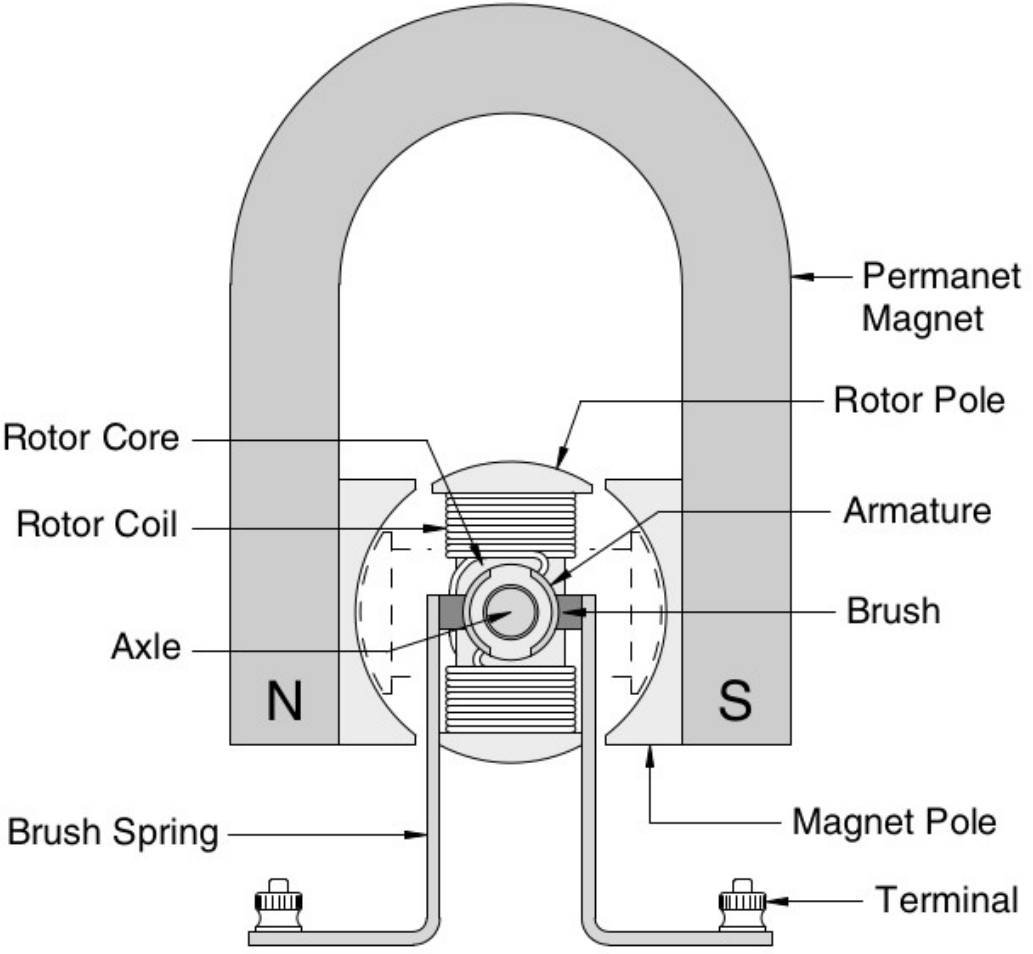
- **Estátor:** parte que da soporte mecánico al aparato y contiene los polos de la máquina, que pueden ser devanados de hilo de cobre sobre un núcleo de hierro o imanes permanentes.
- **Rotor:** es un componente generalmente de forma cilíndrica, también devanado y con núcleo, alimentado con corriente directa a través del colector formado por delgas. Las delgas se fabrican generalmente de cobre y están en contacto alternante con las escobillas fijas.

»»» Motor de corriente continua

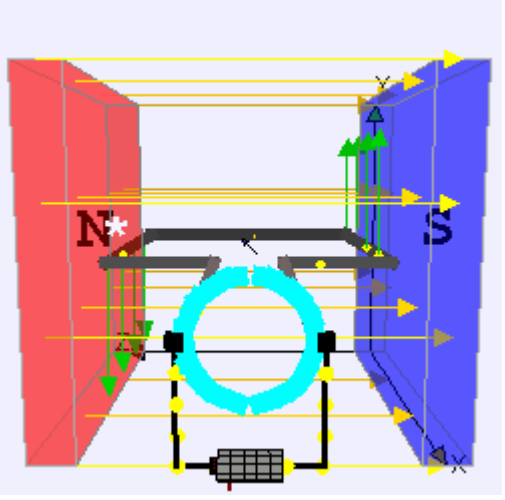
MOTOR C.C.



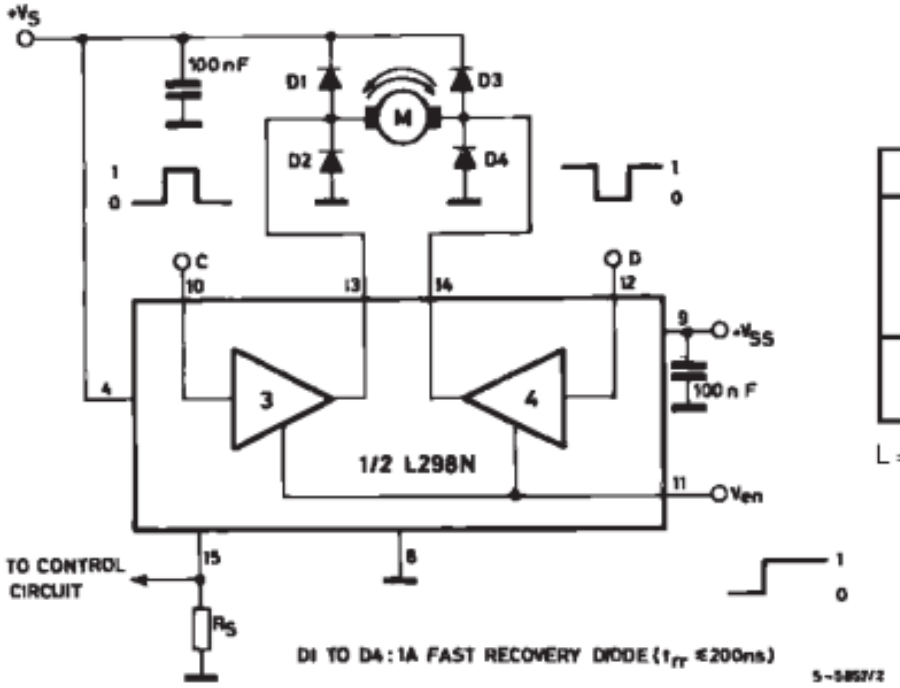
Símbolo



MOTOR C.C.



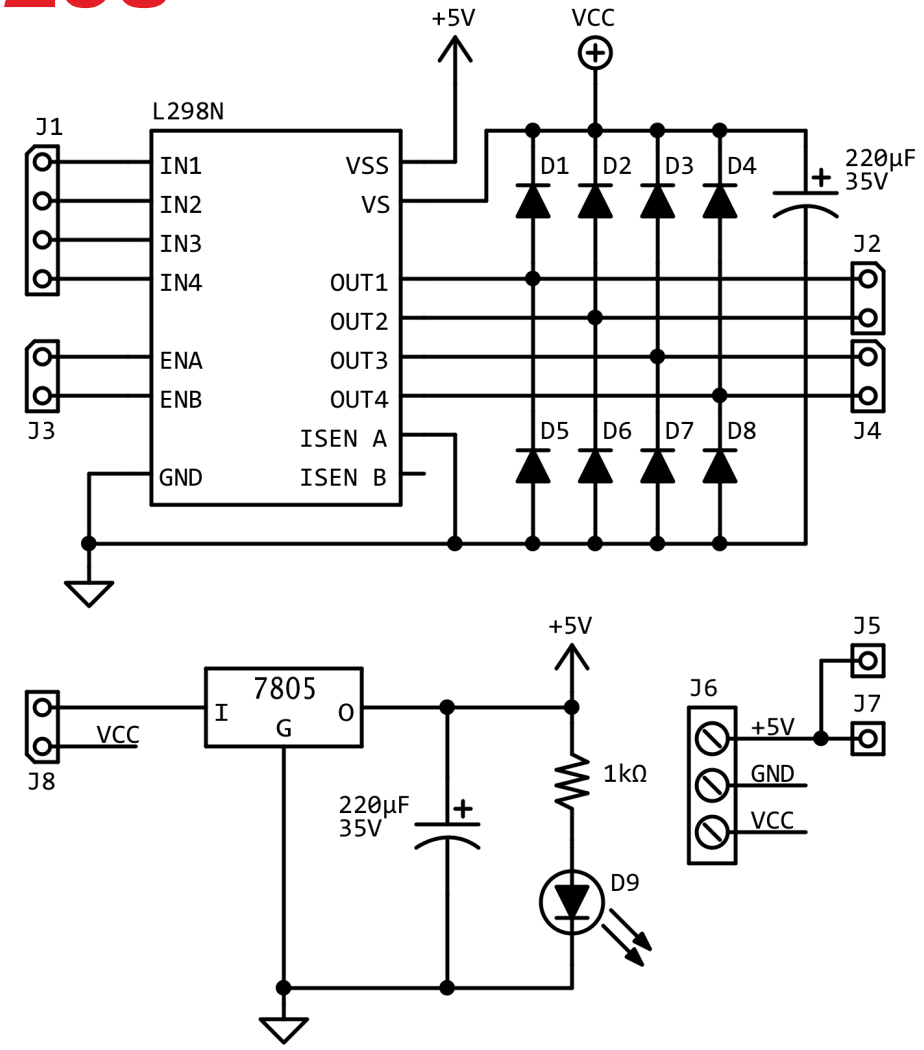
CONTROL DE MOTORES



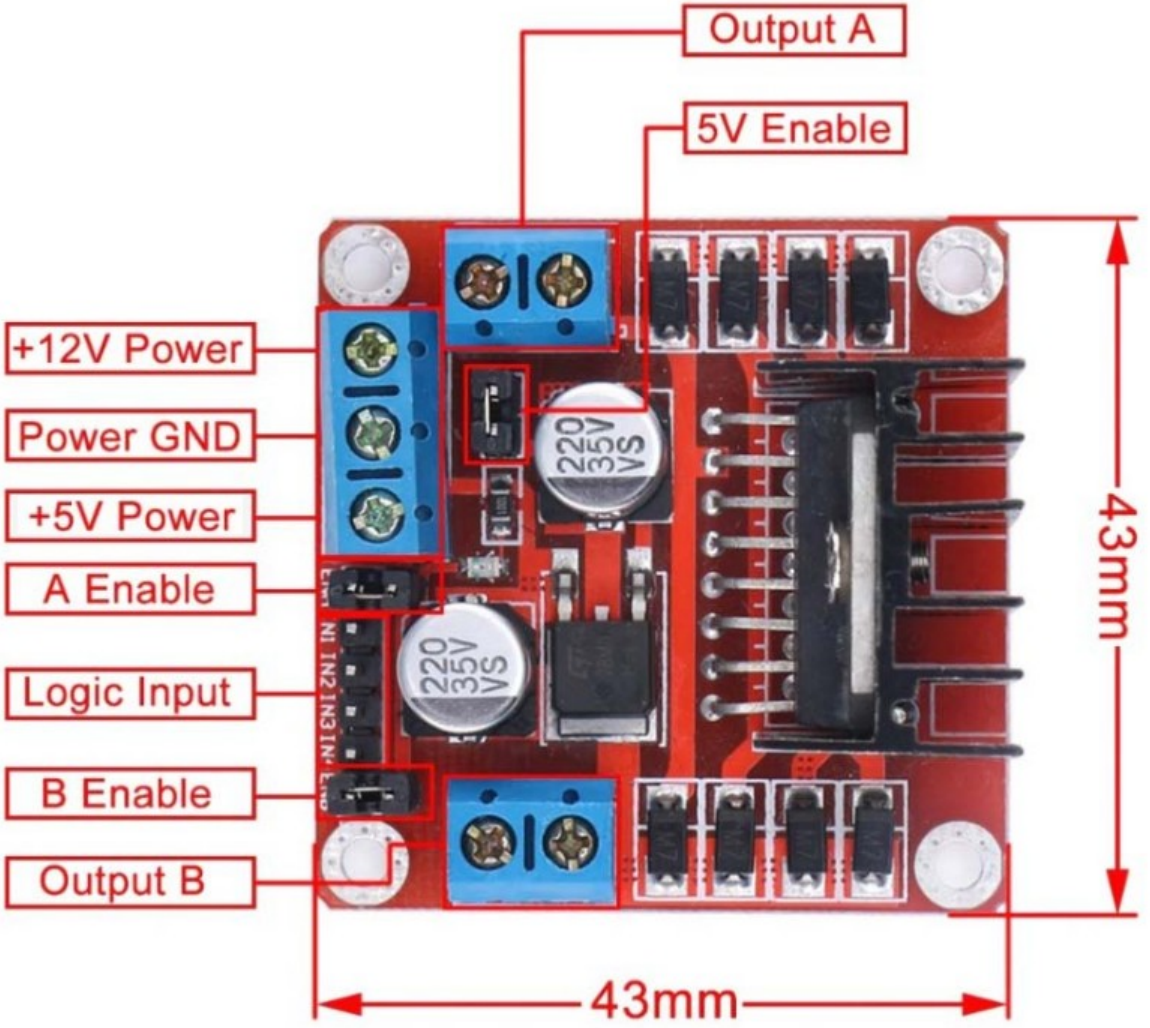
Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

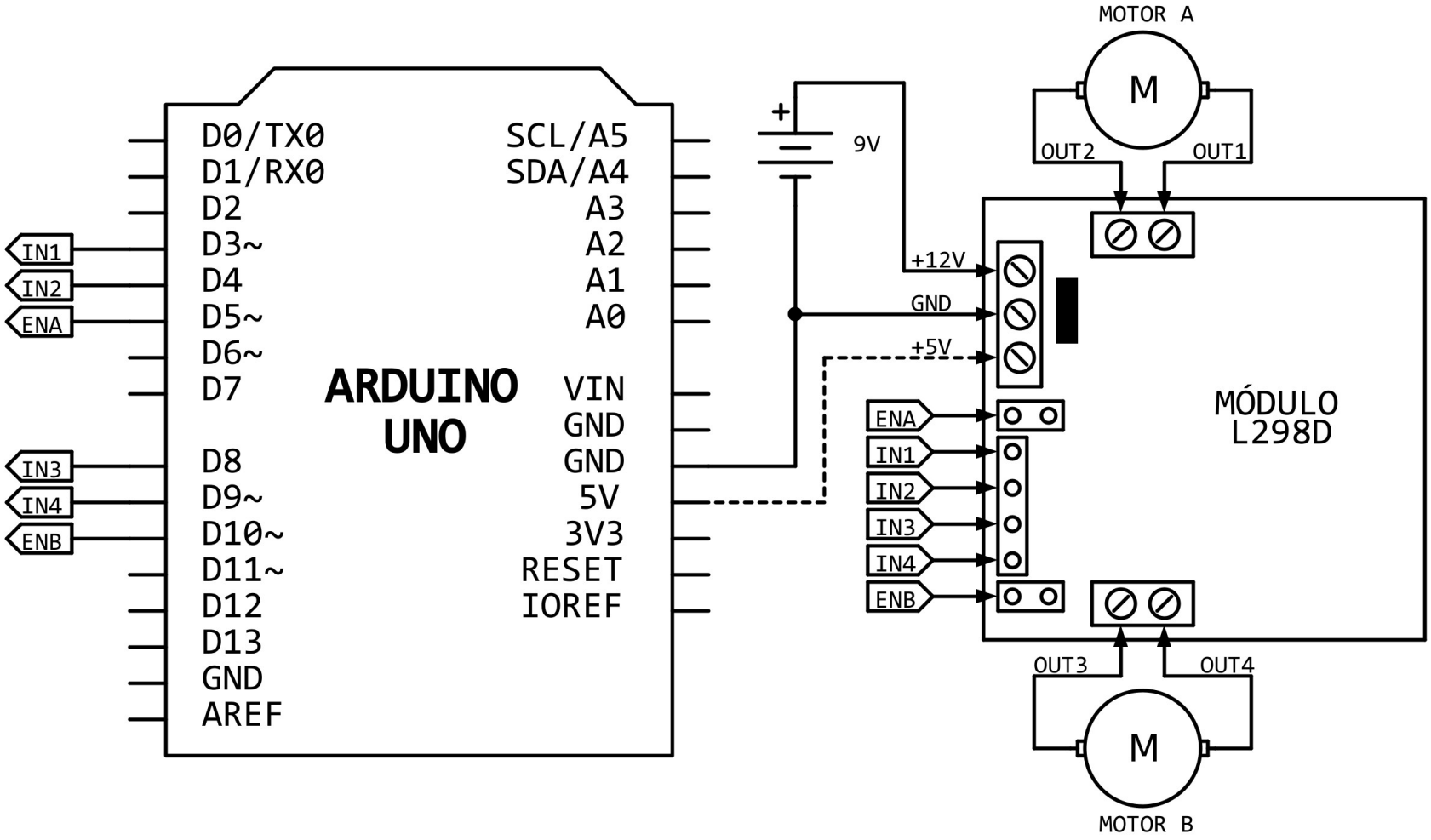
MÓDULO L298


+5V
VCC
L298N
J1
IN1
IN2
IN3
IN4
ENA
ENB
J3
GND
VSS
VS
OUT1
OUT2
OUT3
OUT4
ISEN A
ISEN B
D1
D2
D3
D4
D5
D6
D7
D8
220µF
35V
J2
J4
+5V
7805
I
G
O
VCC
J8
220µF
35V
1kΩ
D9
J5
J6
J7
+5V
GND
VCC

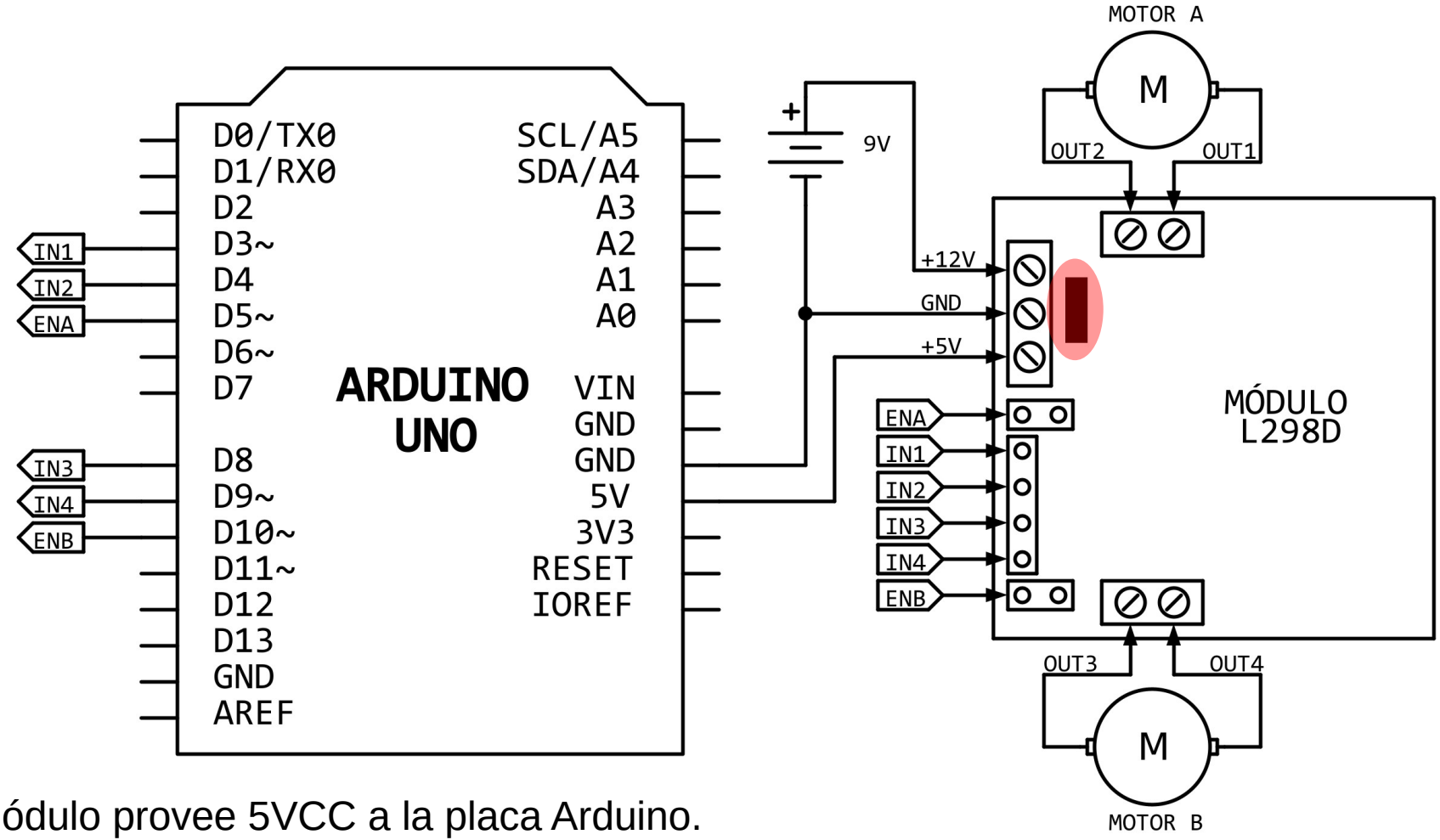
MÓDULO L298


Output A
5V Enable
+12V Power
Power GND
+5V Power
A Enable
Logic Input
B Enable
Output B
220 35V VS
220 35V VS
43mm
43mm

CONEXIONES MÓDULO L298

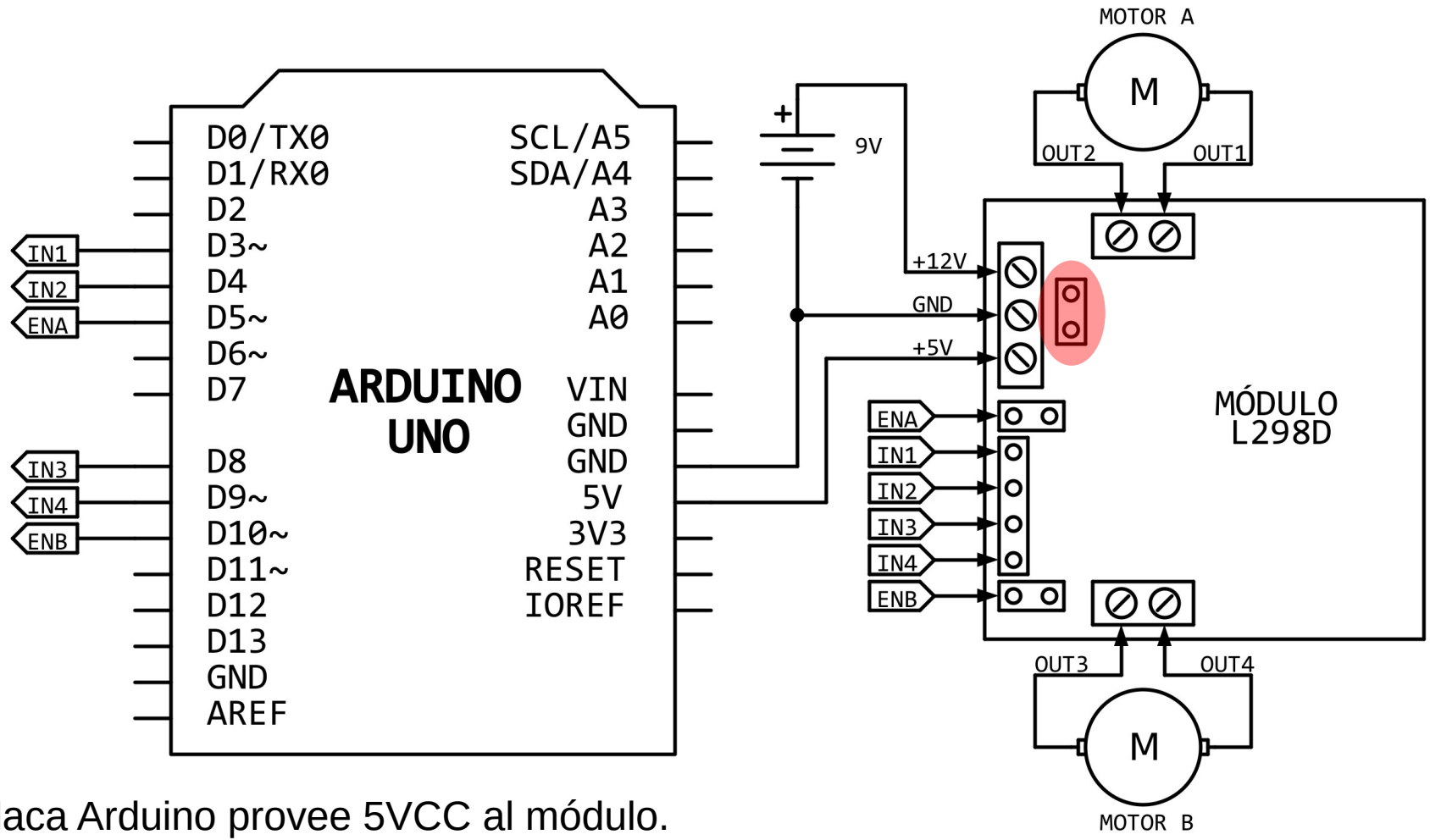


CONEXIONES MÓDULO L298



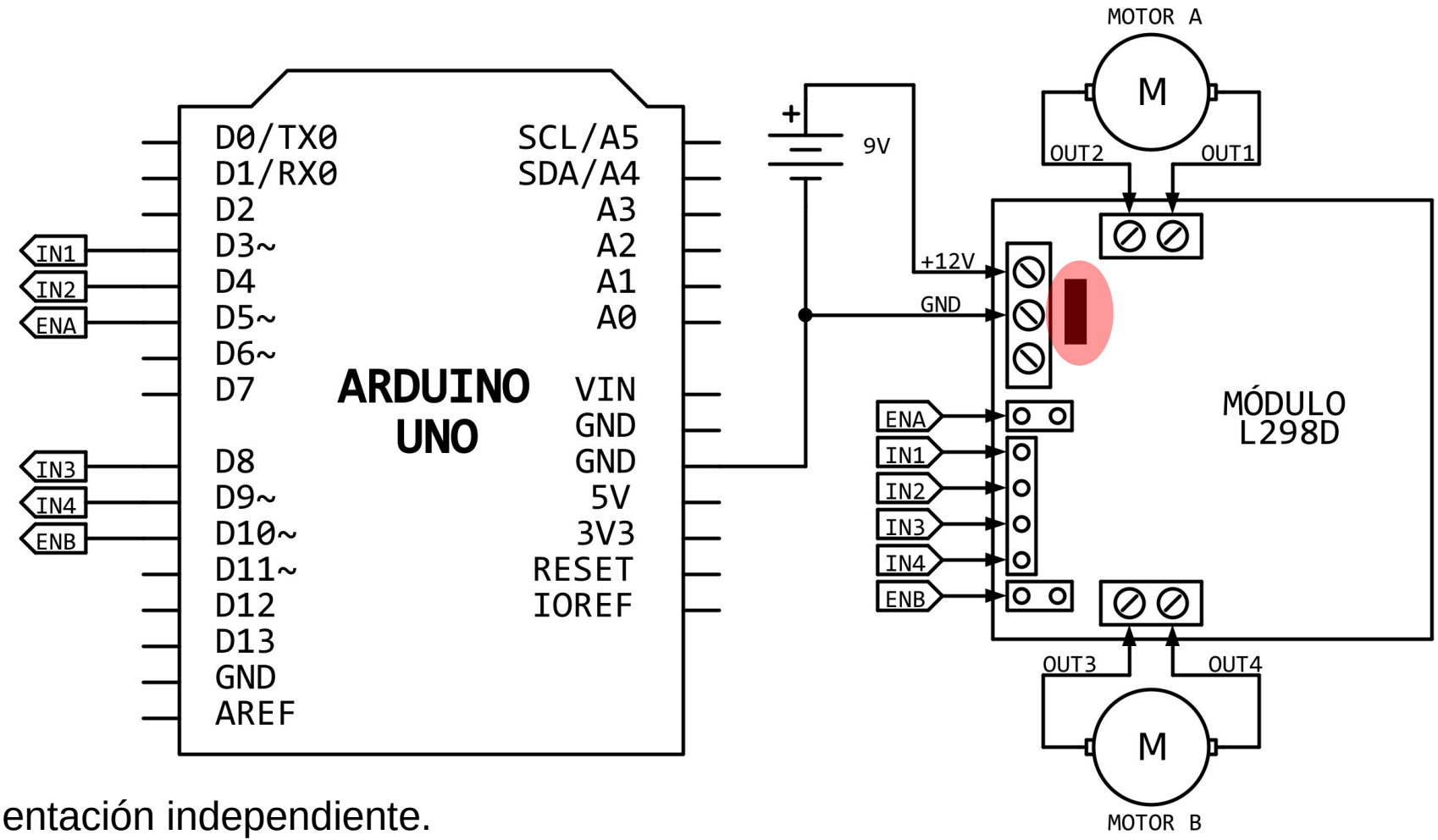
El módulo provee 5VCC a la placa Arduino.

CONEXIONES MÓDULO L298



La placa Arduino provee 5VCC al módulo.

CONEXIONES MÓDULO L298



Alimentación independiente.

EJEMPLO MÓDULO L298

```
const int MOTOR_1A = 2;
const int MOTOR_2A = 4;
const int MOTOR_EN = 3;

void setup() {
    pinMode(MOTOR_1A, OUTPUT);
    pinMode(MOTOR_2A, OUTPUT);
}

void loop() {
    analogWrite(MOTOR_EN, 100);
    digitalWrite(MOTOR_1A, HIGH);
    digitalWrite(MOTOR_2A, LOW);
}
```

CRÉDITOS

Lucas Martín Treser
lmtreser@gmail.com – www.automatismos-mdq.com.ar