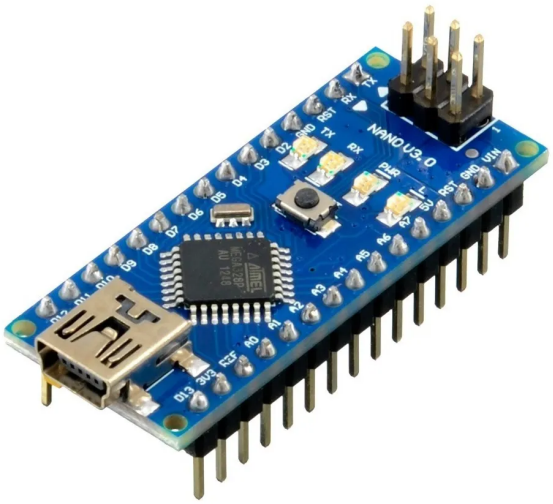
Resumen de contenidos:

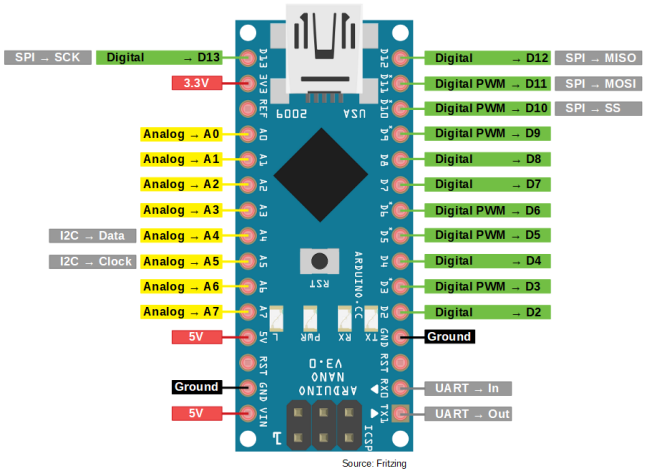
controladores, sensores y actuadores

Autor: Gaspar Fábrega R.

1. ARDUINO:

Corresponde al cerebro de nuestro robot, el controlador Arduino NANO v3 es el componente que nos permitirá recibir y almacenar información desde el ambiente, para poder generar una acción dependiendo de esta.





Source: Fritzing

1.1. Entradas y Salidas:

DIGITALES (D2-D13): Permiten recibir y emitir señales digitales (consistentes de 1s y 0s).

ANÁLOGAS (A0-A7): Permiten recibir señales análogas (valores de 0 a 5v), que serán interpretadas por el controlador como un valor entero de 0 a 1023.

También funcionan como entradas y salidas digitales.

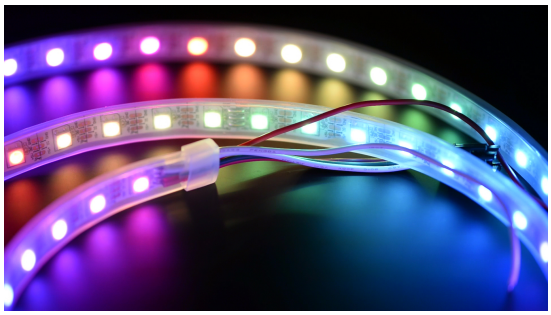
PWM: Salidas que permiten emular una señal análoga, utilizando pulsos en vez de una emisión continua, corresponden a los pines (D3, D5, D6, D9, D10 y D11).

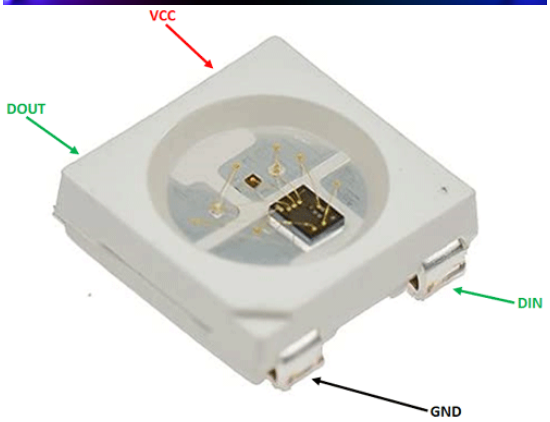
PODER Pines que permiten obtener energía o alimentar al arduino:

- **GND**: tierra o -.
- **5V**: permite alimentar componentes con 5V.
- **3v3**: permite alimentar componentes con 3.3V.
- **VIN**: permite alimentar el arduino de 3V a 10V.

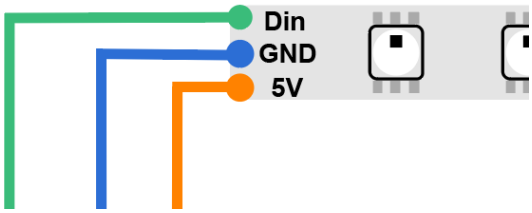
2. LEDS:

Utilizamos LEDs *WS2812*, que tienen la capacidad de ser controlados de manera digital, utilizando alguna de las múltiples librerías disponibles para arduino (FastLED, NeoPixel, etc...).





2.1. Conexiones:



Las cintas LEDs que utilizamos tienen tres conexiones:

- **DIN:** entrada digital que permite controlar la intensidad y el color de cada LED. Se conecta a un pin digital del arduino.
- **VCC:** Alimentación positiva de energia, se conecta a 5V en arduino.
- **GND:** Tierra, se conecta a GND en arduino.

2.2. *setup()*

Los pasos necesarios para poder controlar la tira LED utilizando la librería ***FastLED.h*** son:

1) Cargar a librería:

```
#include "FastLED.h"
```

2) Definir variables necesarias:

- *NUM_LEDS:* el numero de LEDs que se van a controlar.
- *leds:* una lista que contendrá elementos de tipo CRGB que tendrá el largo *NUM_LEDS* (la cantidad de LEDs en la tira), para poder controlar de manera especifica el color de cada LED.
- *PIN:* el pin mediante el cual se controlará la tira led. Debe ser una salida digital.

```
#define NUM_LEDS 30

CRGB leds[NUM_LEDS];

#define PIN 3
```

3) inicializar cinta de leds:

Utilizar la función *FASTLED.addLeds* junto a las variables definidas para poder iniciar la cinta con el largo adecuado.

```
void setup()
{
  FastLED.addLeds<WS2811, PIN, GRB>(leds,
      NUM_LEDS).setCorrection( TypicalLEDStrip );
}
```

2.3. *loop():*

Para utilizar nuestra tira LED de manera eficiente, tenemos la siguientes funciones base que nos permiten controlar los LEDs de manera individual y global.

1) mostrarLeds():

Esta función permite actualizar el estado actual de la tira LED, mostrando los LEDs tal y como se encuentran configurados. Debe ser llamada cada vez que se quiera ver una nueva configuracion en la tira LED.

Lo que hace en realidad es llamar la función *FastLED.show()*, desde la librería *FastLED.h*.

```
void mostrarLeds() {
   FastLED.show();
}
```

2) setPixel(Pixel, red, green, blue):

Esta función permite cambiar el color de los LEDs de manera individual, identificado la posición con un entero (***Pixel***) y un color en formato RGB, donde cada valor debe ser entregado en forma de entero de 8bits (0-255) para cada color (***red*** = rojo, ***green*** = verde, ***blue*** = azul).

```
1 void setPixel(int Pixel, byte red, byte green, byte
      blue) {
2    leds[Pixel].r = red;
3    leds[Pixel].g = green;
4    leds[Pixel].b = blue;
5 }
```

2) setAll(red, green, blue):

Esta función permite cambiar el color de toods los LEDs al mismo tiempo, entregandole un color en formato RGB, donde cada valor debe ser entregado en forma de entero de 8bits (0-255)(***red*** = rojo, ***green*** = verde, ***blue*** = azul).

Lo que esta función realiza es un ciclo for, donde repite ***setPixel()*** para cada LED de 1 a **NUM_LEDS**, con el mismo color.

```
1 void setAll(byte red, byte green, byte blue) {
2   for(int i = 0; i < NUM_LEDS; i++ ) {
3     setPixel(i, red, green, blue);
4   }
5   showStrip();
6 }
```

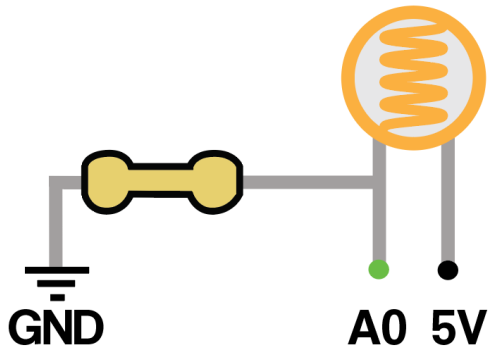
3. LDR:

Una fotoresistencia o *"Light Dependent Resistor"*, es un componente electrónico que cambia su resistencia (al paso de los electrones), dependiendo de la cantidad de luz que brilla sobre este.

3.1. Conexiones:

Para poder leer de manera correcta el valor de la corriente que pasa a través de la fotoresistencia, y poder relacionarla con alguna intensidad, debemos conectarla a alguna de las entradas análogas como un "divisor de voltaje".

Para esto, debemos conectar uno de los terminales a una entrada análoga, y a tierra mediante una resistencia y el otro terminal debe ser conectado a 5v.



3.2. *setup():*

Para poder recibir valores correspondientes al nivel de luz que recibe el LDR sólo es necesario edfinir la variable que se utilizará para almacenar el valro leido por el pin de entrada análoga al que está conectado, ya que no se utilizan librerías, y no es necesario inicializar el pin para leer valores análogos:

1) definir variables necesarias:

Se define una variable que recibirá el valor leído por el pin análogo al que está conectado el LDR. Esta deberá ser de tipo entero, y puede o no estar inicializada con algún valor.

```
1 int ldr = 0;
```

3.3. *loop():*

1) analogRead(APin):

Para poder leer los valores del LDR, utilizamos la función ***analogRead()***, la que nos entregará un valor entero entre 0 y 1023, dependiendo de cuanto voltaje recibe el pin análogo ***APin***.

```
1 ldr = analogRead(A0);
```

En este caso, estaríamos leyendo el valor del pin A0, y guardandolo en la variable "ldr".

2) map(value, fromLow, fromHigh, toLow, toHigh):

Para interpretar estos valores de mejor manera, dependiendo del uso que le vamos a dar, podemos utilizar la función ***map()*** a la que le entregamos:

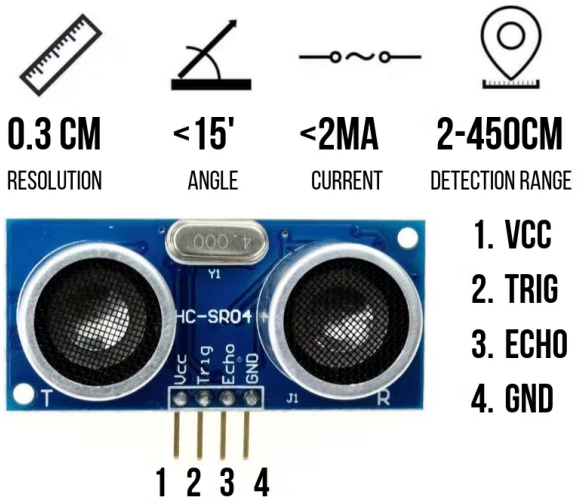
- *value:* el valor que queremos asociar a un mapa.
- *fromLow:* el valor mínimo que toma el valor original (0).
- *fromHigh:* el valor máximo que toma el valor original (1023).
- *toLow:* el valor mínimo que queremos.
- *toHigh:* el valor máximo que queremos.

```
1 ldr = map(ldr, 0, 1023, 0, 100);
```

En este caso, estaríamos actualizando el valor de la variable "ldr", pasando de un valor en el rango [0,1023] a [0,100].

4. ULTRASONIDO:

El sensor de ultrasonido nos permite calcular la distancia a un objeto inmediatamente al frente del sensor, con una resolución de 3mm y hasta una distancia de 4 metros y medio.



Utilizamos la ecuación de velocidad para poder encontrar la distancia recorrida por un pulso emitido, en el tiempo que se demora en ir y volver al sensor.

$$\textbf{Velocidad} = \frac{\text{Distancia}}{\text{Tiempo}} \implies$$

$$Distancia = Velocidad \cdot Tiempo$$
$$D = V \cdot t$$

Conocemos la velocidad del sonido en el aire, y sabemos que el tiempo que nos entrega el sensor es el tiempo que el sonido demora de ida y de vuelta (al obstaculo), por lo que se obtiene:

$$D\ [m] = 343\ [m\ s^{-1}] \cdot \frac{t}{2}\ [s]$$

Como el arduino trabaja en milisegundos y queremos una distancia en centimetros, se tiene que:

$$D\ [cm] = 0.0034\ [cm\ ms^{-1}] \cdot \frac{t}{2}\ [ms]$$

4.1. Conexiones:

Los sensores ultrasónicos que utilizamos tienen cuatro conexiones:

- **VCC:** Alimentación positiva de energia, se conecta a 5V en arduino.
- **TRIG:** Pin de emisión del pulso de sonido, se debe conectar a un pin digital del arduino.
- **ECHO:** Pin de recepción del pulso de sonido, debe conectarse a un pin digital del arduino.
- **GND:** Tierra, se conecta a GND en arduino.

Los pasos necesarios para poder medir la distancia a un obstáculo utilizando el sensor de ultrasonido son:

4.2. *setup():*

1) Definir los pines a utilizar:

```
1 int trigPin = 5;
2 int echoPin = 6;
```

2) Definir las variables a utilizar: donde guardaremos el tiempo que el sensor demora en detectar el pulso desde que fue emitido, y la distancia equivalente.

Utilizamos variables de punto flotante (numeros reales, con decimales), para poder calcular con la mayor precisión posible la distancia.

```
1 float duration_us, distance_cm;
```

3) Inicializar pines: configurando el pin correspondiente a la emisión de sonido (*trigger*) como **INPUT** y el pin correspondiente a la recepción del sonido (*echo*) como **INPUT**.

```
1 void setup()
2 {
3   pinMode(trigPin, OUTPUT);
4   pinMode(echoPin, INPUT);
5 }
```

4.3. *loop():*

Dentro del loop, los pasos necesarios para poder calcular la distancia son:

1) emitir un pulso de sonido por 10 microegundos

```
1 digitalWrite(trigPin, HIGH);
2 delayMicroseconds(10);
3 digitalWrite(trigPin, LOW);
```

2) medir el tiempo que demora en devolverse el pulso:

```
1 duration_us = pulseIn(echoPin, HIGH);
```

3) calcular la distancia:

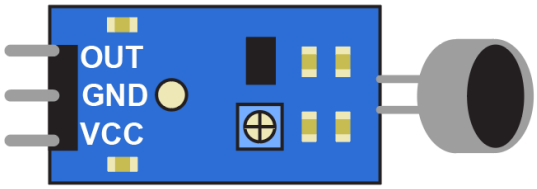
```
1 distance_cm = 0.017 * duration_us;
```

5. ELECTRET:

5.1. Conexiones:

El sensor de sonido digital que utilizamos posee tres conexiones importantes:

- **OUT:** Salida digital, envía una señal en HIGH (5V) cuando el volumen del sonido recibido por el micrófono supera el umbral definido por el potenciómetro que se encuentra en el circuito y LOW (0V), cuando el volumen es menor.
- **GND:** Tierra o -.
- **VCC:** Entrada de alimentación con 5V.



5.2. *setup():*

1) Definir los pines a utilizar:

```
1 int pinMic = 3;
```

2) Definir las variables a utilizar:

```
1 int mic = 0;
```

3) Inicializar pines:

```
1 void setup()
2 {
3     pinMode(pinMic, INPUT)
4 }
```

5.3. *loop():*

Para leer los valores, utilizamos la función *digitalRead(PIN)*, dándole el pin desde el cual vamos a leer la señal emitida por el micrófono.0El valor recibido será **0**, si no detecta sonido, y **1** si el sonido detectado supera el umbral definido por la posición del potenciómetro.

```
1 mic = digitalRead(pinMic)
```

6. SERVOS:

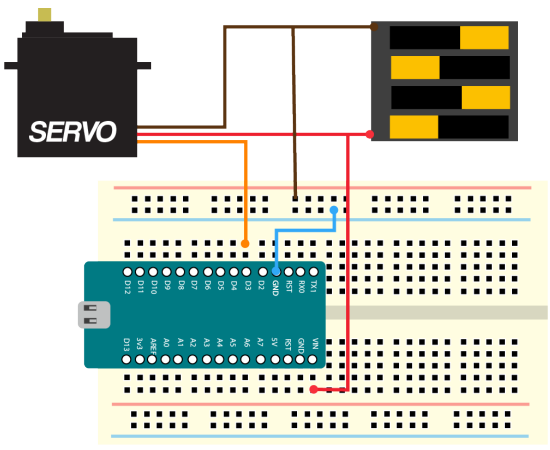
6.1. Conexiones:

El servo que utilizamos tiene tres conexiones básicas:

- **GND:** Entrada de conexión a tierra.
- **5V:** Entrada de alimentación con 5V.
- **Din:** Entrada digital, donde podremos controlar la posición angular del servo utilizando uno de los pines con capacidad de utilizar PWM.

Dependiendo de cómo vamos a alimentar el servo, podemos utilizar las siguientes configuraciones:

6.1.1. Servo y Arduino en paralelo:



6.2. *setup()*

Los pasos necesarios para poder controlar los servos utilizando la librería ***Servo.h*** son:

1) Cargar la librería:

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

2) Inicializar servo:

```
1 Servo miServo;
```

3) definir pines a utilizar:

```
1 int pinServo = 3;
```

4) inicializar pines a utilizar:

```
1 void setup()
2 {
3   myservo.attach(pinServo);
4 }
```

6.3. *loop()*

Para poder controlar el angulo, utilizamos la función *write()* implementada en la librería Servo.h, primero llamando al servo previamente definido (en este caso, *miServo*). A esta función le entregamos un ángulo entre 0 y 180 grados.

Debemos agregar una pausa para que el servo tenga tiempo para llegar a la posición deseada.

```
1 miServo.write(15);
2 delay(20);
```

7. MOTORES - PUENTE H:

7.1. Módulo L298N

Un puente H es un dispositivo compuesto por 4 transistores, el cual permite controlar el sentido de la corriente, permitiéndonos así, controlar la velocidad y sentido de giro de los motores DC. En nuestro caso contamos con un modulo L298N el cuál cuenta con 2 puente H, con los cuales podemos controlar 2 motores a la vez.

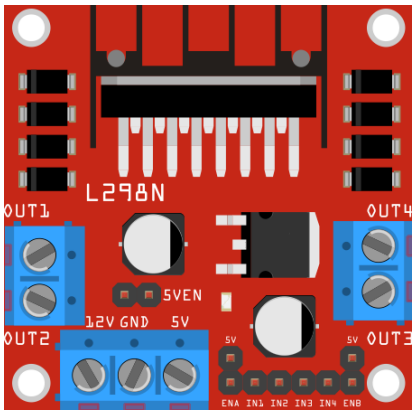


Figura 1: Módulo L298N

En la figura 1 se muestra el módulo L298N, cuyos terminales de conexión energéticos corresponden a:

- **Vin**: Terminal de alimentación del módulo, admite entre 3V y 12V
- **GND**: Terminal de conexión a tierra
- **5V**: Terminal V lógico, entrega un voltaje de 5V
- **Salida Motor A**: Terminales OUT1 y OUT2
- **Salida Motor B**: Terminales OUT3 y OUT4

Nota: El módulo L298N cuenta con un regulador de voltaje que entrega 5V en el terminal de voltaje lógico siempre y cuando se alimente el terminal **Vin** con una tensión de hasta 12V y mientras el jumper se encuentre conectado en **5VEN**.

Las conexiones que se utilizan para controlar el puente H corresponden a

- **IN1 - IN2**: interruptores para controlar dirección de giro del motor A.
- **IN3 - IN4**: interruptores para controlar dirección de giro del motor B.
- **ENA**: pin para regular potencia del motor A (puede estar conectado directamente a 5V mediante un "jumper", si no se requiere control de velocidad)
- **ENB**: pin para regular potencia del motor B (puede estar conectado directamente a 5V mediante un "jumper", si no se requiere control de velocidad)

7.2. Conexión motores

Aprovechamos el regulador de voltaje del L298N para alimentar tanto la placa arduino, como los motores, con el fin de evitar la sobrecarga de la placa. Para este fin debemos contar con una batería que se adecue al voltaje máximo explicado anteriormente correspondiente a 12V. En el siguiente ejemplo, se utiliza una batería de 9V, y las conexiones se muestran en la siguiente imagen

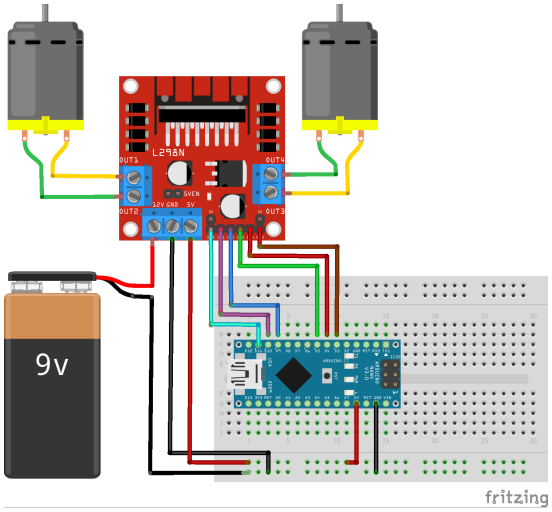


Figura 2: Conexión de Motores y Puente H

Se observa además, los pines de control conectados a pines de salida digital del arduino (D1-D12), con ENA y ENB, conectados a pines con capacidad de generar PWM para poder controlar al velocidad.

7.3. *setup():*

1) Definir pines a utilizar: Los pines correspondientes a las entradas ENX (control de velocidad), deben ir conectados a pines con capacidad PWM. Los pines de control de interruptores INX deben estar conectados a salidas digitales.

```
1  int enA = 5;
2  int in1 = A0;
3  int in2 = A1;
4
5  int enB = 6;
6  int in3 = A2;
7  int in4 = A3;
```

2) Inicializar pines: todos los pines deben ser salidas.

```
1  void setup() {
2    pinMode(in1, OUTPUT);
3    pinMode(in2, OUTPUT);
4    pinMode(in3, OUTPUT);
5    pinMode(in4, OUTPUT);
6
7    pinMode(enA, OUTPUT);
8    pinMode(enB, OUTPUT);
9  }
```

7.4. *loop():*

1) controlar la dirección de giro:

Para controlar la direcicon de giro, debemos configurar el estado de los pines INX correspondientes, a continuación, se presenta una tabla con las posibles ocnfiguraciones para IN1 e IN2, destacando que es equivalente para IN3 e IN4.

IN1	IN2	MODO
HIGH	LOW	giro hacia adelante
LOW	HIGH	giro hacia atrás
HIGH	HIGH	freno (cortocircuito)
LOW	LOW	freno (neutro)

Tabla 1: Modos de operación para los motores

Esto se define en el código de la siguiente manera (ambas ruedas girando en la misma dirección):

```
digitalWrite(in1, LOW);
digitalWrite(in2, HIGH);

digitalWrite(in3, LOW);
digitalWrite(in4, HIGH);
```

2) controlar la velocidad:

La velocidad se controla utilizando los pines PWM, con la capacidad de emular una señal análoga de 8 bits (0-255) de 0V a 5V.

Una velocidad de 0, equivalente a los motores completamente detenidos se escribe:

```
analogWrite(enA, 0);
analogWrite(enB, 0);
```

Una velocidad de 255, equivalente al máximo que puede entregar el puente H se escribe::

```
analogWrite(enA, 255);
analogWrite(enB, 255);
```