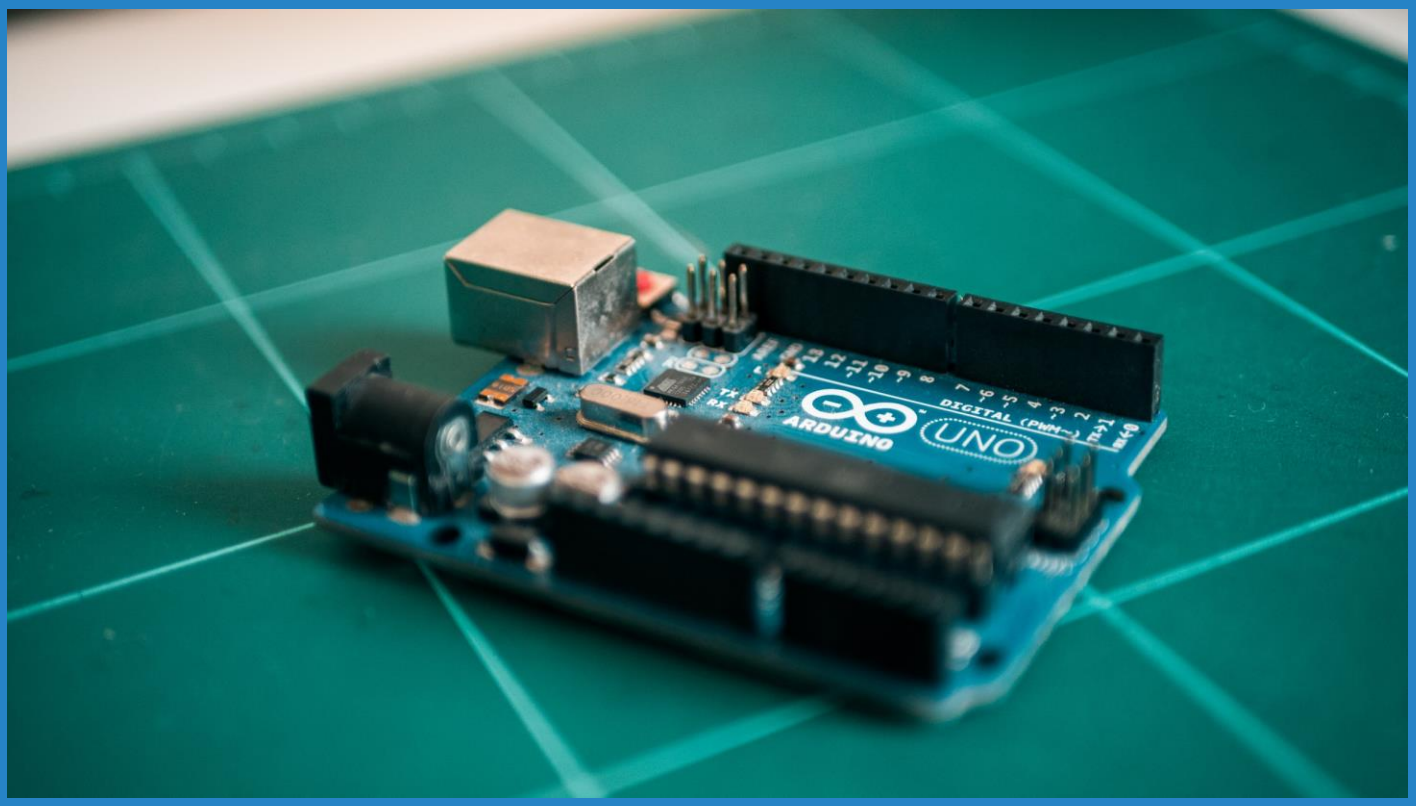


MANUAL CURSO ARDUINO

Instituto Técnico La Piedad

Alumnos:
- Franco Roche
- Geronimo Moraga
- Canclini Camilo

Proyecto Robótica 2021

Avellaneda 324, B8000 Bahía

Blanca, Provincia de Buenos Aires

CONTENIDO

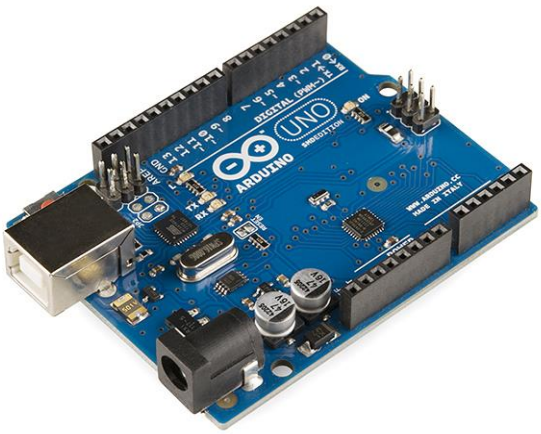
INTRODUCCIÓN

¿Qué es este manual?

En este manual se hará una introducción a Arduino, partiendo de los conceptos teóricos, la aplicación de los mismos en un proyecto final que intentará abarcar todo el contenido visto en el manual, por lo que se hará uso del kit que acompaña al manual y de los archivos adjuntos. El objetivo es explorar las posibilidades del software de Arduino, así como sus distintos componentes. Para este manual requerimos que se tengan: bases de teoría eléctrica, electrónica, programación, matemática y de computación, ya que, si bien en el manual explicaremos desde lo más mínimo, si es recomendable que se tenga una pequeña noción de los temas para un mejor entendimiento. Una vez dicho esto, comencemos.

¿Qué es Arduino?

Arduino es una placa que tiene todos los elementos necesarios para conectar periféricos a las entradas y salidas de un microcontrolador. Es decir, es una placa impresa con los componentes necesarios para que funcione el microcontrolador y su comunicación con un ordenador a través de la comunicación serial. Arduino cuenta con una amplia variedad de microcontroladores diseñados para su uso en esta plataforma, desde un servomotor (pequeño motor que puede moverse) hasta un medidor de humedad.



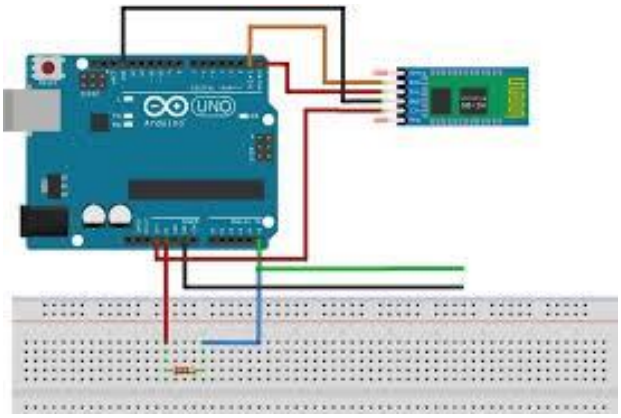
Cabe aclarar que existen varias versiones de Arduino que varían en capacidad del hardware (pines, potencia, almacenamiento, inclusión de wifi y bluetooth, etc.). En lo que respecta a software esta plaqueta cuenta con su propia plataforma de desarrollo o de programacion. Ademas, todo lo que respecta a codigo es de carácter abierto, lo que significa que cualquiera puede utilizarlo y crear nuevas herramientas que puede proveer a la comunidad.

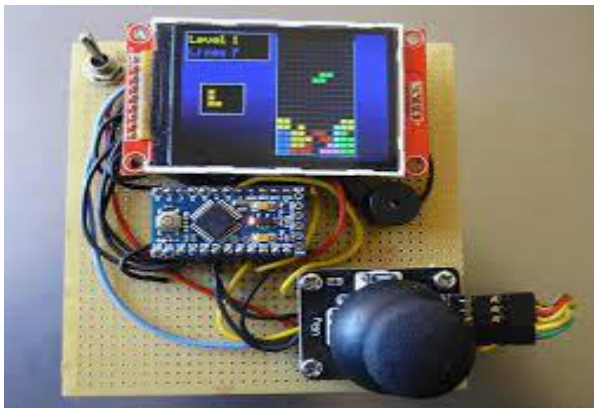
¿Qué es ESP32?

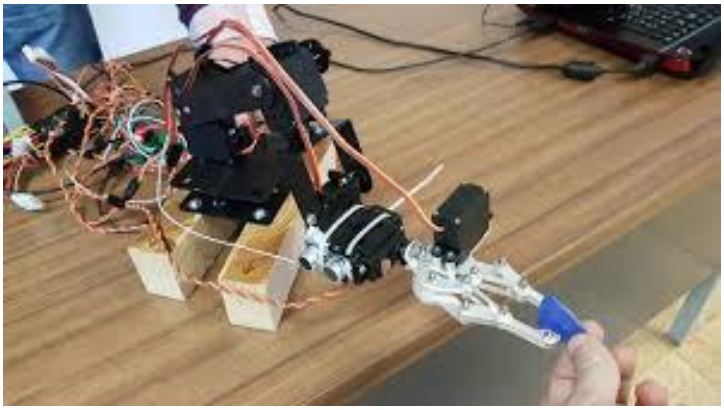
Es un dispositivo similar a Arduino que también soporta el software del mismo, permitiéndonos realizar las mismas tareas, agregando, además nuevas funciones integradas, como el uso de bluetooth y Wifi. Además, contiene un microcontrolador muchísimo más potente que el de las placas Arduino, llegando a tener un doble núcleo. Por estas características es que podemos realizar tareas mucho mas complejas y diversas.

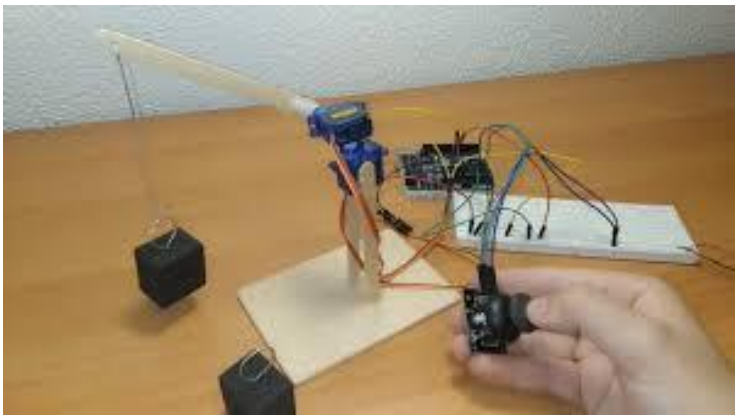
¿Para qué se usan?

- Mecanismo de elevación de portones.
- Medidor de clima.
- Auto a control remoto.
- Automatizar el encendido y apagado de luces.
- Pochoclera.
- Protesis Biónicas.
- Bombas de agua.
- y básicamente cualquier tipo de sistema que pueda automatizarse.







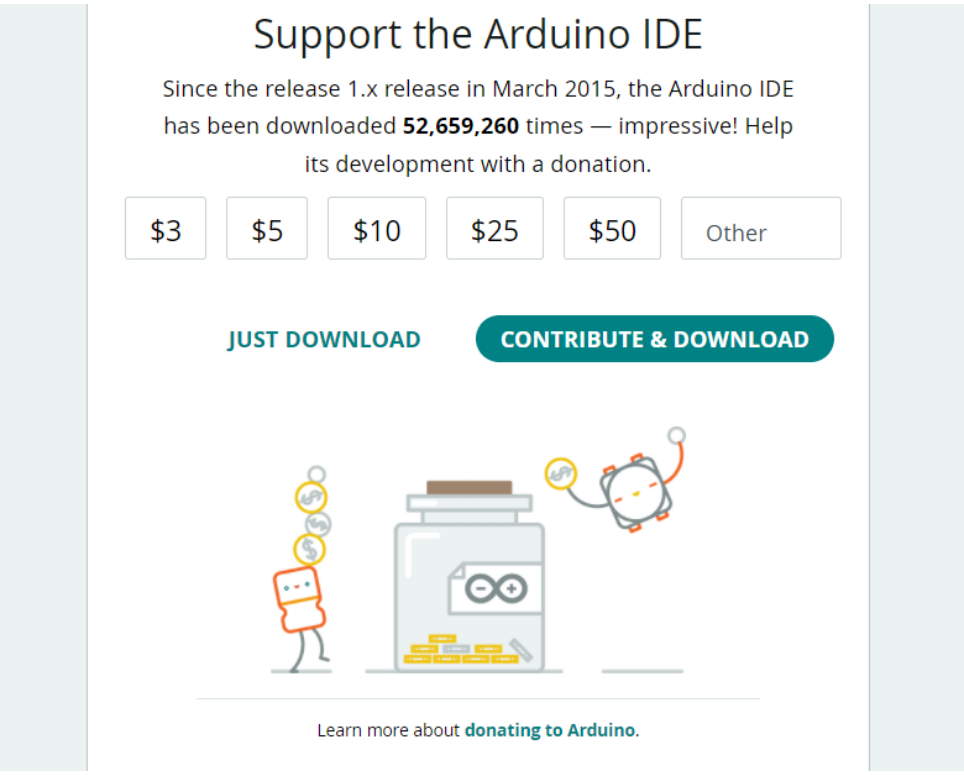


Se usan para diversos proyectos, ya que, pueden tener tantas funciones como periféricos se les agreguen con el único límite de sus pines reales y la capacidad de programación de su usuario. Actualmente se utiliza para realizar prototipos a escala menor de proyectos más grande y avanzados. Es debido a su sencillez y facilidad de programar que lo convierte en una herramienta didacta para enseñar y aprender.

¿CÓMO DESCARGAR EL IDE DE ARDUINO?

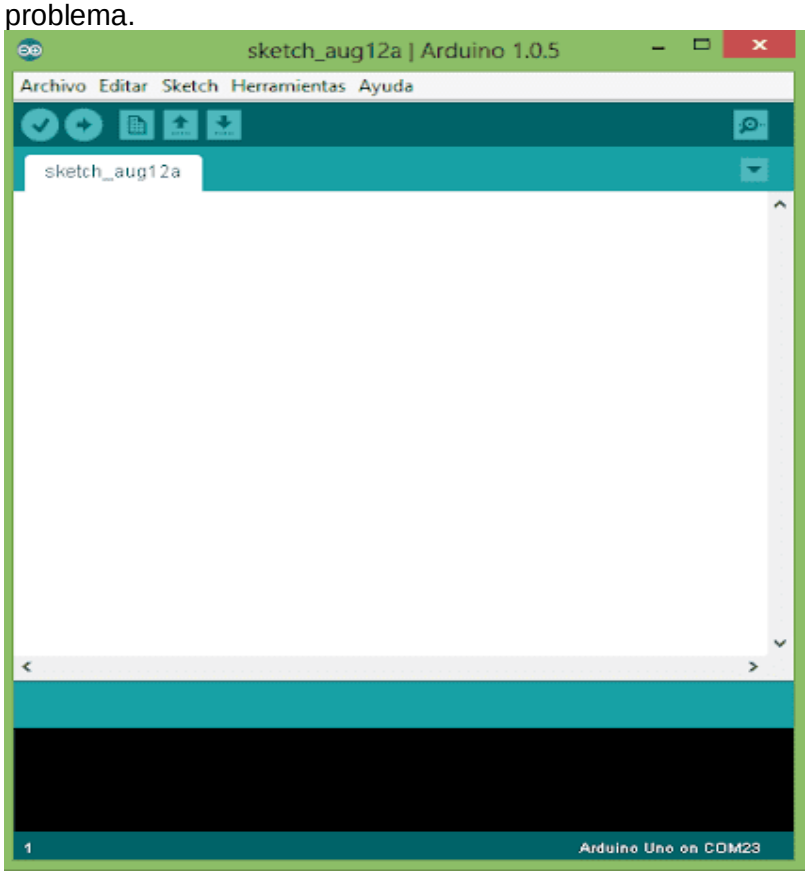
Primero entramos al link: https://www.arduino.cc/en/software
Al entrar al link que nos manda a la página oficial de Arduino para descargar dicho programa, nos aparecen varias opciones para descargar, vamos a elegir la que sea acorde a nuestro sistema operativo que estamos utilizando.

cuando ya sabemos cuál es la que tenemos que elegir le damos click, nos manda al **Support the Arduino IDE** Le damos click a la opción que dice **just DOWNLOAD** y comenzará la descarga.



¿Cómo instalarlo?

Una vez descargado le damos click derecho y le damos **ejecutar como administrador** se nos abrirá una ventana que solamente tenemos que darle a siguiente hasta que parezca la ruta de donde queremos que se nos instale el Arduino IDE cuando sepamos en que disco queremos que este le damos el botón de instalar, se nos pondrá a cargar una barra que tardará unos minutos, al terminar la descarga podremos cerrar la ventana y ya tendríamos instalado Arduino sin ningún problema.



¿Cómo conectar el Arduino en la PC?

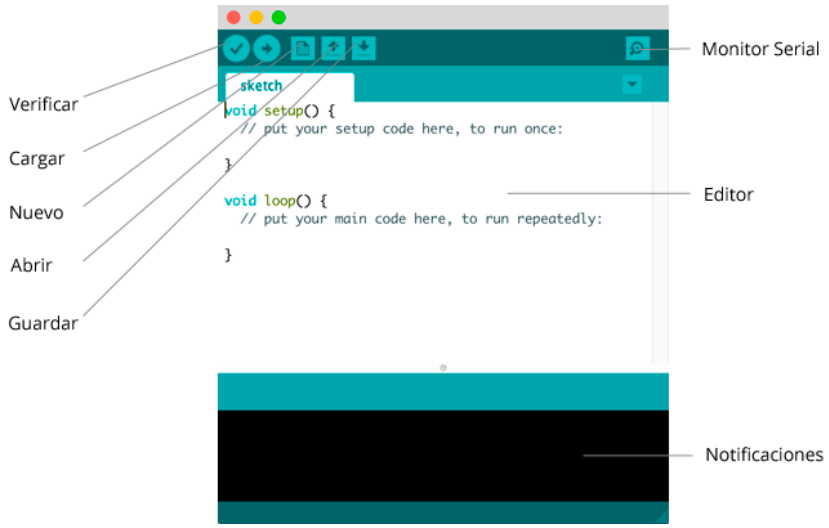
Primero abrimos el programa de Arduino que instalamos, se nos abre la interfaz de Arduino.
Cuando tengamos el Arduino con su cable USB a USB plug lo conectamos en la computadora
El Arduino nos muestra una luz led que nos indica que está conectado a la computadora. Lo que tenemos que hacer ahora es configurar la aplicación para que nuestro Arduino funcione correctamente.

En la interfaz de Arduino nos vamos a la sección de herramientas y vamos a donde dice Placa, ahí nos vamos a encontrar con varios tipos de Arduino, seleccionamos nuestro tipo de Arduino que en este caso sería “Arduino UNO” y ya podríamos programar tranquilamente con Arduino.

En caso de que el pc o el entorno de Arduino no reconozca la placa, se tendrá que borrar el driver e instalarlo nuevamente, o bien reemplazar el cable (por lo general se debe a esta última).

ARDUINO IDE (MENU Y ESPACIO DE TRABAJO)

En el menú hay varias opciones, que como veremos en la siguiente imagen son:



Verificar: Verifica si hay alguna parte mal redactada en el código. esto no asegura que el programa funcione para lo que queremos que sirva, solo que el Arduino podrá leer y ejecutar el código.

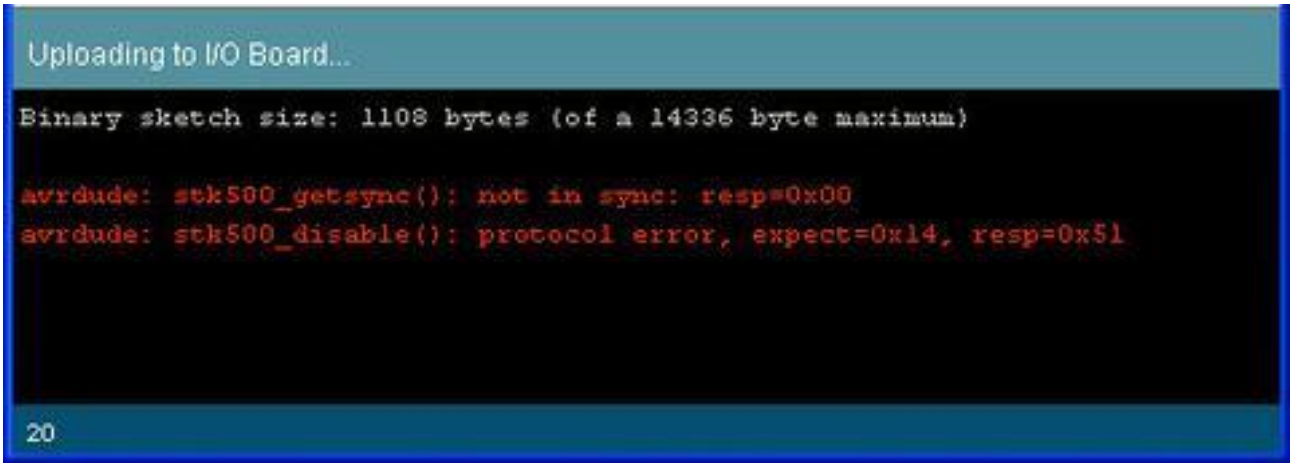
Cargar: se sube el código al Arduino, además si se selecciona Arduino IDE automáticamente hará una verificación.

Nuevo: abre una nueva pestaña de Arduino IDE

Abrir: nos deja seleccionar un archivo guardado en el pc que se mostrará y se podrá modificar en el editor.

Guardar: nos dejará guardar el programa actual en el pc.
Monitor serial: no muestra los textos que se ejecutan con ***Serial.print*** en el código y sirve para saber si alguna ruta del código se ejecuta

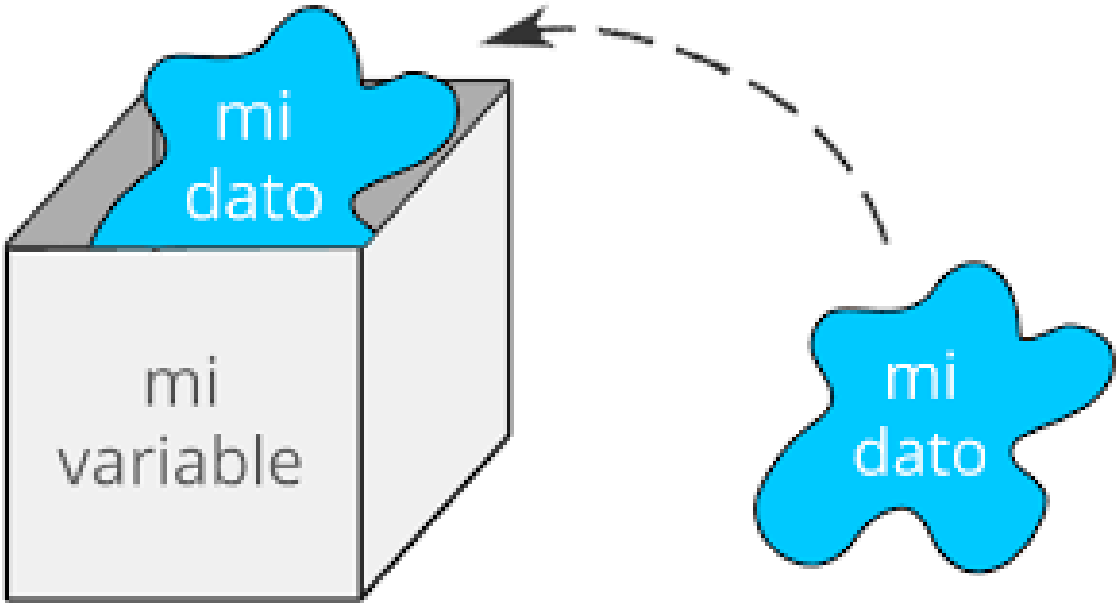
Además, el programa tiene el PANEL DE EDITOR (fondo blanco) en el que se muestra el código de programación que tiene el programa y el PANEL DE NOTIFICACIONES (el de fondo negro, en la parte inferior) en el que se señalará si hay algún error, este último puede ser de gran ayuda, ya que, si se tiene un mínimo conocimiento de inglés y de programación básica, podemos encontrar la raíz de un problema en el código.



BASE DE PROGRAMACIÓN

Variables

Técnicamente hablando es un espacio asignado en la memoria, al cual se puede recurrir en cualquier momento de la ejecución. Básicamente podemos verlo como si fuera una caja en la cual nosotros guardamos determinado valor, ya sea un número, un estado, o una cadena de texto(palabra).



A su vez estas tienen un formato o tipo de variable que tenemos que determinar, los cuales son:

int(entero): permite números enteros entre -32,768 y 32,767.

float: permite números con coma.

double: permite también números con coma, pero con mayor amplitud que float.

string: se utiliza para cadenas de caracteres, ósea textos y números.

char: almacena solamente un carácter, es como un string reducido a un solo carácter

boolean(estado): Permite definir dos estados, true(verdadero) y false(falso), son los 2 únicos valores que puede almacenar este tipo de variables

Hay más formatos, pero estos son los que generalmente se utilizan a la hora de programar en Arduino.

PINES DE ARDUINO

Arduino tiene 2 grandes grupos de pines de salida analógicos y digitales, las señales digitales representan valores discretos, en lugar de valores dentro de un cierto rango y las analógicas en cambio tienen valores continuos tanto en tiempo como en magnitud (una señal analógica para cambiar de 10 a 11 tiene que pasar por todos los decimales que hay entre estos y una señal digital hace el cambio de inmediato sin pasar por sus valores intermedios) siendo las señales analógicas producidas en la naturaleza y las digitales por ordenadores. Las salidas de pines digitales son los que están numerados del 2 al 13 y los analógicos van del A0 al A5.

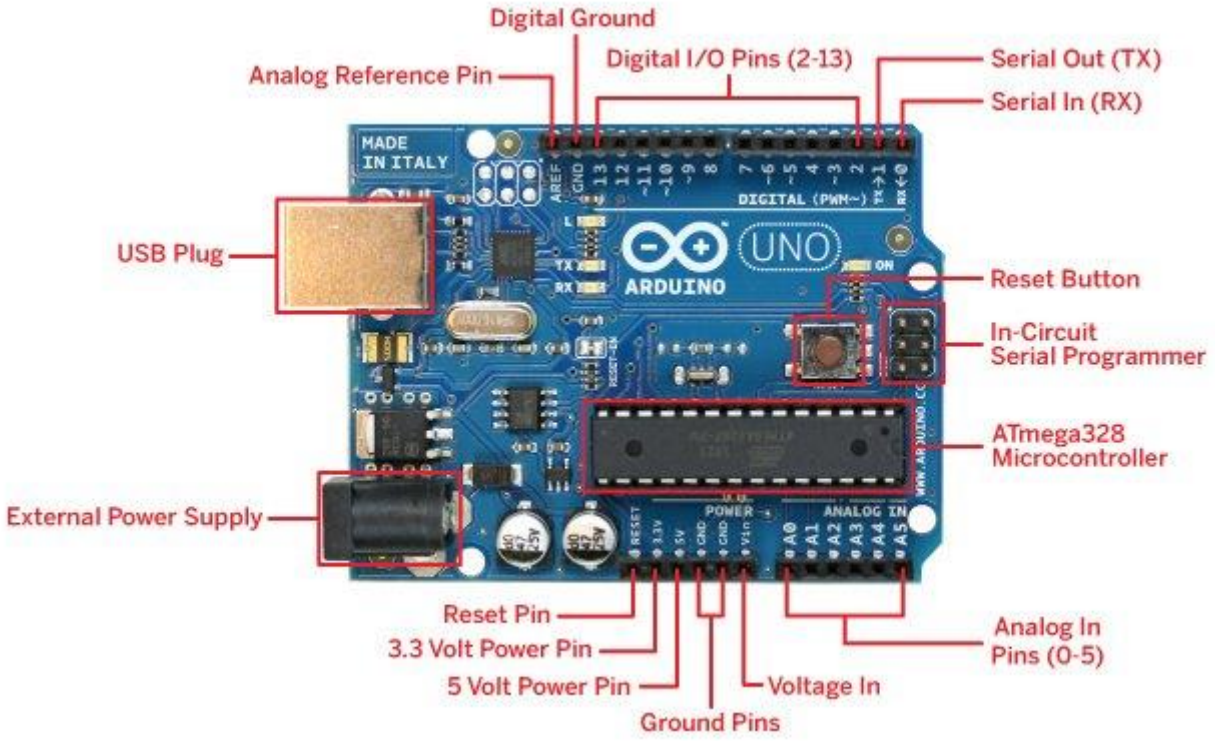
Se conecta a la computadora mediante un USB tipo A (no tipo A mini).

El pin VIN permite aplicar una fuente de alimentación externa en el rango de 12 a 6 volts DIRECTAMENTE a la entrada del regulador de la tarjeta Arduino.

Los pines de GND es la terminal de tierra a 0 volts y nos permite cerrar un circuito eléctrico.

- El pin 5v este pin nos da una tensión de salida de 5 voltios DC.
- El pin 3.3v este pin nos da una tensión de salida de 3.3 voltios DC.
- El pin reset no solo reinician el Arduino, sino que también lo pone en modo recuperación, por lo que si has flasheado algún programa que provoca que el PC no te lo reconozca, y por tanto no puedes volver a flashearlo, pulsando el botón reset, o haciendo un puente del pin reset a GND.
- El pin AREF permite conectar una tensión externa de referencia, contra la que se comparará la señal que leamos en los pines A0 a A5.
- Los Pines 0 (RX) y 1 (TX) se utilizan para recibir (RX) y transmitir (TX) datos en serie TTL.
- Cuando presionamos el pulsador cerramos el circuito y el botón reset se conecta a la tierra (0v) y nuestra placa de Arduino se resetea.

El microcontrolador ATMega328 es el que define qué función tomará cada pin.



PROGRAMACION EN ARDUINO IDE

Estructura inicial de un código Arduino

Las 2 funciones que nombraremos a continuación son las que, por defecto, aparecen al iniciar un nuevo proyecto en el IDE de Arduino.

DECLARAN VARIABLES:

Se pueden declarar variables numéricas de varios tipos, en cualquiera de las secciones de declaración global PUBLIC o PRIVATE. Para declarar una variable debe indicarse el tipo (int, unsigned short o unsigned char) seguido del nombre para la variable. Vea el ejemplo a continuación:

```
int contador=0;
```

Se declara una variable tipo INT, llamada CONTADOR y se le asigna un valor de 0.

SETUP("Void Setup( ){ }"):

Todo lo que pongamos dentro de las llaves "{" "}" de esta función se realizará UNA VEZ, con "setup" se refiere a que aquí se configuran las variables y tareas iniciales, como por ejemplo la conexión a una red (ya que solo se conecta una vez, no se reconecta a no ser que haya un error)

LOOP("Void Loop( ){ }")

Todo lo que pongamos dentro de las llaves "{" "}" de esta función se realizará **INFINITAMENTE,** de ahí su nombre "loop". Aquí es donde se pondrán todos los procesos repetitivos de nuestro proyecto, por ejemplo, la acción de medir de un determinado sensor (por lo general, mide constantemente)

delay():

Crea un retardo en el código del tiempo indicado en milisegundos, en otras palabras, crea un tiempo de espera para realizar la siguiente línea de código.

pinMode() y digitalWrite()

Para prender un led, un motor y realizar un pitido, básicamente enviar corriente a un punto específico, en Arduino se utilizan las funciones: “pinMode()” y “digitalWrite()”.

pinMode

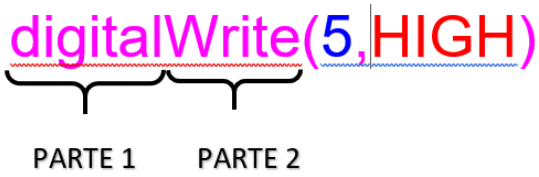
Esta función nos permite determinar que pines de la plaqueta serán de salida de (transmisores de datos), y entrada de corriente (receptores de datos).

```
pinMode(5,OUTPUT)
```

pinMode(): Aquí se llama a la función y dentro de los paréntesis se ponen los argumentos con los que trabajara, separados por comas.
5: Este “5” es el primer argumento, aquí se determina el número de pin que se seteara.
OUTPUT: Este es el segundo argumento, este indica el estado del pin, solo tiene 2 valores: “OUTPUT”, que significa que el pin se configurará para salida de corriente y ¨INPUT¨, que significa que se configurará para entrada de corriente(receptor).

digitalWrite(), analogWrite(), digitalRead y analogRead()

Las funciones tipo ¨Write¨ nos permiten habilitar y deshabilitar el paso de corriente en los pines, básicamente el enviar datos a los dispositivos que estén conectados a la plaqueta. Y las tipo “Read“, nos permiten saber si el pin está recibiendo corriente de un dispositivo. Hay que saber cómo están establecidos los pines que utilizamos, porque puede ocurrir que estemos enviando corriente a un pin que está establecido como de entrada, ósea como INPUT.
Estas funciones tienen dos partes:

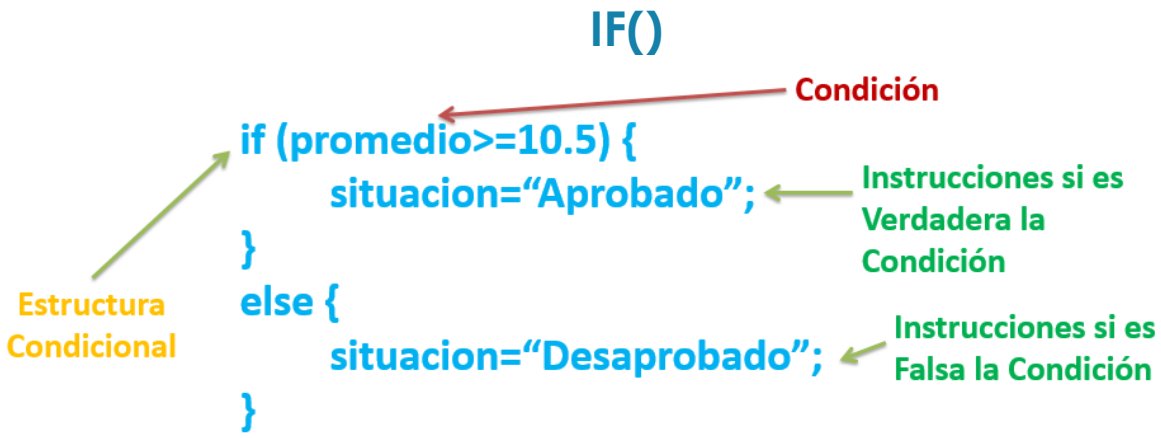


En la primera se define el tipo de pin, si es análogo o digital, y en la segunda parte la acción que se va a realizar, si es “Write” se enviará corriente y si es “Read” se verificará si el pin tiene corriente, de ser este último caso no se especificara segundo argumento, vea la sintaxis a continuación.

```
digitalWrite(5,HIGH)
analogWrite(5,HIGH)
digitalRead(HIGH)
analogRead(HIGH)
```

pinMode(): Aquí se llama a la función y dentro de los paréntesis se ponen los argumentos con los que trabajara, separados por comas.
5: Este “5” es el primer argumento, aquí se determina el número de pin que se utilizara para la acción de la función(Read o Write).
HIGH: Este es el segundo argumento, este se utiliza si la función es tipo Write (como mencionamos), puede tener 2 valores: HIGH, que significa que se envía corriente y LOW que significa que NO se envía más corriente.

Sentencias condicionales



Es una sentencia condicional, es decir dependiendo de una condición procederá a hacer una serie de instrucciones, en caso de que no se cumpla le podemos indicar que haga otra serie de acciones mediante un else, que evalué otra condición para cumplir otras instrucciones mediante un else if o que directamente no haga nada

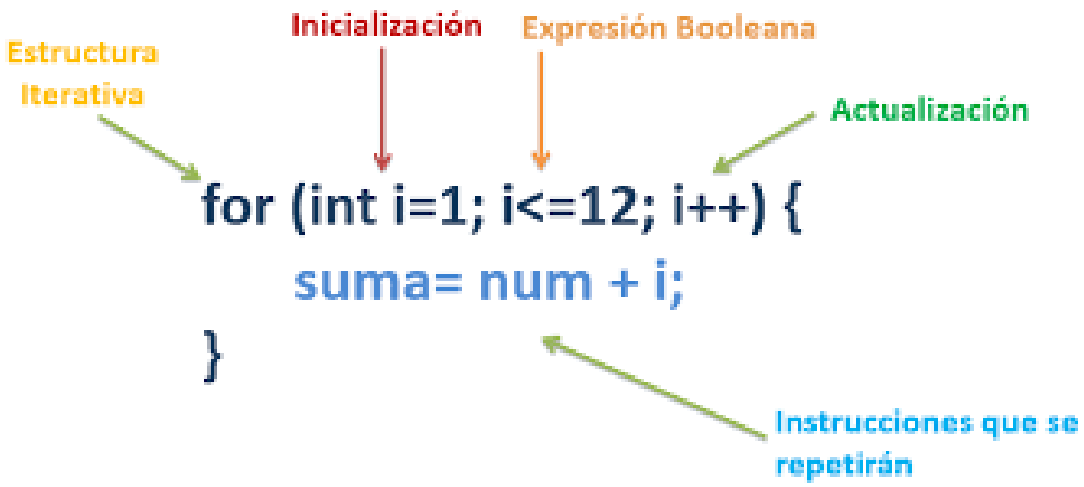
```
int b;
b=6;
if (b<5){
        Serial.print("b es menor a 5");
} else if (b=5){
        Serial.print("b es igual 5");
} else {
        Serial.print("b es mayor 5");
}
```

en este caso dependiendo de si b es mayor, igual o menor a 5 mostrara diferentes textos. En el primer caso Hace la comparación mediante un if, en el segundo caso mediante un else if que nos servirá ya que en el último caso usa directamente un else, es decir que si no se cumple ninguna de las condiciones anteriores ejecutara esas instrucciones. Si en este código no se hubiera usado el else if y en lugar de eso se usará un if aparte, en el tercer caso se tendría también que haber usado un if especificando que b sea mayor a 5.

Sentencias iterativas

FOR ():

Es una estructura iterativa(repetitiva) para ejecutar un mismo segmento(parte) de código una cantidad de veces deseada, conociendo previamente un valor de inicio, un tamaño de paso y un valor final para el ciclo.



INICIALIZACIÓN:
Declaramos una variable que es de tipo “INT” y que inicializarse en uno. nuestro bucle empieza a iniciarse en uno.

EXPRESION BOOLEANA:
Es una expresión que va a determinar si el bucle se va a seguir ejecutando o no. Por ejemplo: nuestro bucle se va ejecutar siempre cuando nuestro contador sea menor o igual que 12.

ACTUALIZACION:
Es la manera que se va a ir incrementando o decrementando el contador de nuestro bucle, en este caso i++ quiere decir que va ir incrementando en uno en uno.

Instrucciones que se repetirán:
Para el ejemplo de la foto se va a repetir 12 veces, va a inicializarse en uno y se va repetir mientras este mismo contador que va aumentado en uno en uno sea menor o igual a 12.

WHILE

```
while(condición){
//sentencias a ejecutar
}
```

Crea un bucle que ejecuta una sentencia especificada mientras cierta condición se evalúe como verdadera. Dicha condición es evaluada antes de ejecutar la sentencia y para que no se ejecute infinitamente hay que modificar dentro del código la variable condicional. El while se usa cuando no sabemos cuántas veces se quiere repetir el proceso.

DO WHILE

```
do{
//sentencias
} while (condicion)
```

El do while sirve para lo mismo que el while, pero cambia la sintaxis, ya que luego de ejecutarse al menos una vez evaluará la condición lo que asegura que se ejecuten por lo menos una vez las sentencias

LECCIONES

LECCIÓN A: PRENDER LED

El ejercicio primario para empezar con la programación en Arduino es prender un led, en esta lección le mostraremos cómo prenderlo y al final haremos que un conjunto de leds se prende en secuencia. Este código de ejemplo hace que se apague y se prenda un led en intervalos de 1 seg.

CÓDIGO

```
void setup()
{
  pinMode(13, OUTPUT);//SE SETEA EL PIN
}
void loop()
{ //INICIA EL LOOP
  digitalWrite(13, HIGH);//SE ENVÍA CORRIENTE(SE PRENDE)
  delay(1000); //ESPERA 1 SEGUNDO
  digitalWrite(13, LOW);//SE DEJA DE ENVIAR CORRIENTE(SE APAGA)
  delay(1000); //ESPERA 1 SEGUNDO
} //VUELVE AL INICIO DEL LOOP
```

EJERCICIO

En este ejercicio final haremos que se prendan en secuencia los leds y además explicaremos la función “for”.

CÓDIGO

```
//DECLARACION INT

int contador=0;

//SETUP

void setup(){
for(contador=5;contador<13;contador++)
pinMode(contador, OUTPUT); // pines 5 a 13 como salida
}

//PRIMERA PASADA DE LUZ Y LOOP

void loop(){
for(contador=5;contador<13;contador++) // cuenta de 5 a 13
{
digitalWrite(contador, HIGH);
delay(100);
digitalWrite(contador, LOW);
delay(100);
}

//SEGUNDA PASADA DE LUZ

for(contador=13;contador>5;contador--) // cuenta atrás de 13 a 5
{
digitalWrite(contador, HIGH);
delay(100);
```

```
digitalWrite(contador, LOW);
delay(100);
}
}
```

Utilidad de la lección

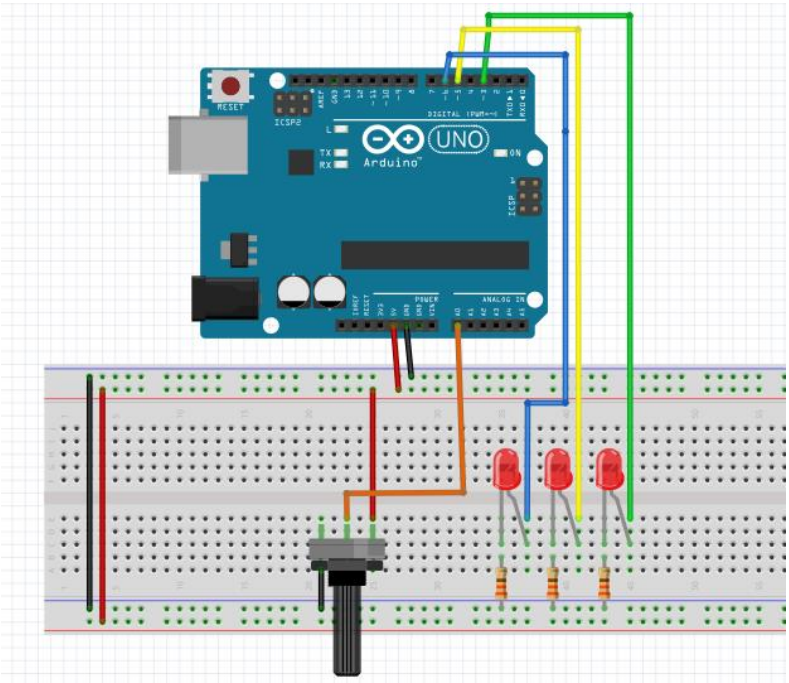
Una vez comprenda el manejo de leds en Arduino podrá modificar el comportamiento de los mismos en el robot que elija a la hora de finalizar el manual. Básicamente le permitirá añadir, más allá de una cuestión estética, indicadores que le permitan saber cosas como, el estado de la batería del robot, si algún sensor está detectando algo o bien indicar que algún motor está funcionando, las utilidades son infinitas.

LECCIÓN B: USO DE POTENCIOMETRO

En esta lección utilizaremos un potenciómetro para controlar los leds que vimos en la lección anterior.

El potenciómetro es un dispositivo electrónico que posee una perilla que al girar, dependiendo la posición en la que se encuentre, esta variará proporcionalmente el valor de una resistencia que se encuentra dentro del mismo.

Los valores que envía el potenciómetro son analógicos, teniendo esto en cuenta, tenemos que conectar esta salida de datos analógica con la entrada analógica del Arduino (véase pines de Arduino).



CÓDIGO

```
//Variable donde almacenaremos el valor del potenciómetro
long valor;

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

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

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

  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);
  pinMode(LED_3, OUTPUT);
  pinMode(LED_4, OUTPUT);
}

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

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

//Segun el valor de potenciometro prendera uno de los 4 leds
  if(valor >= 0 && valor <=255)
  {
      digitalWrite(LED_1, HIGH);
      digitalWrite(LED_2, LOW);
      digitalWrite(LED_3, LOW);
      digitalWrite(LED_4, LOW);
  } else if (valor >= 256 && valor <=511  {
      digitalWrite(LED_1, LOW);
      digitalWrite(LED_2, HIGH);
      digitalWrite(LED_3, LOW);
      digitalWrite(LED_4, LOW);
  } else if (valor >= 512 && valor <=767) {
      digitalWrite(LED_1, LOW);
      digitalWrite(LED_2, LOW);
      digitalWrite(LED_3, HIGH);
      digitalWrite(LED_4, LOW);
    } else {
      digitalWrite(LED_1, LOW);
      digitalWrite(LED_2, LOW);
      digitalWrite(LED_3, LOW);
      digitalWrite(LED_4, HIGH);
   }
}
```

Utilidad de la lección

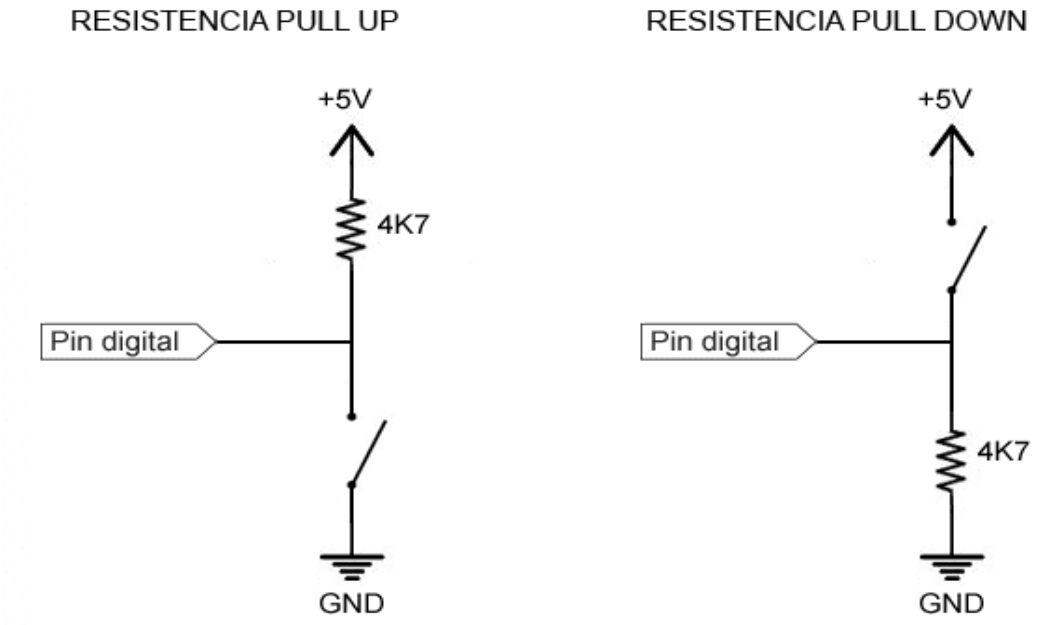
El uso de un potenciómetro, en el robot, nos puede permitir armar un control remoto físico que nos permita manejar o dirigir ciertas funciones del robot, como puede ser mover extremidades, levantar algo, encender determinados leds, controlar aceleración (en el caso de un auto).

LECCIÓN C: USO DE BOTON O PULSADOR

Es un dispositivo que cuando se encuentra “cerrado” deja pasar la corriente y cuando está “abierto” impide el paso de la misma, un pulsador no es más que un tipo de interruptor, el cual se mantendrá en posición de “cerrado” tanto tiempo como pulsado lo mantengamos.

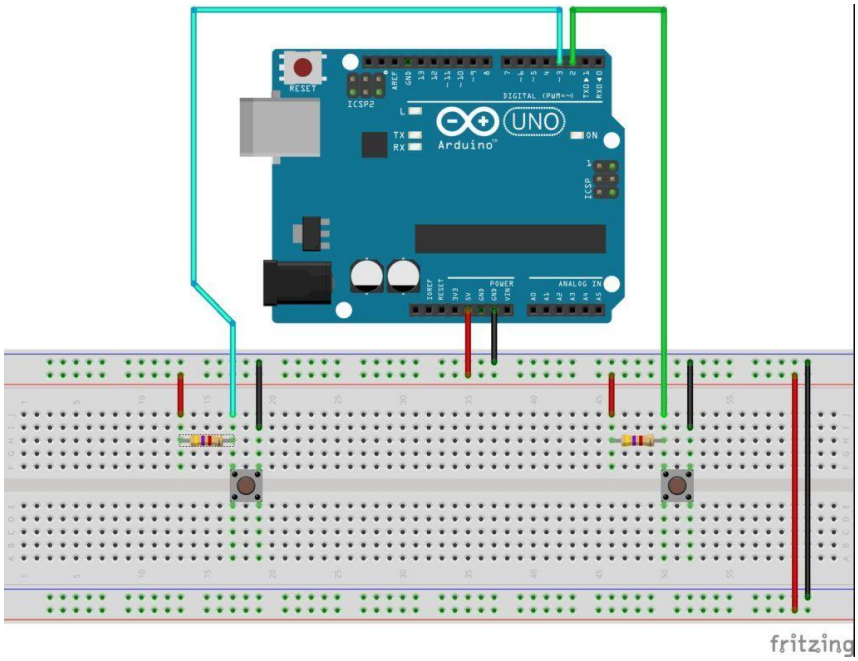
La función primordial de las resistencias en el circuito del botón es mantener un estado lógico conocido cuando el botón no está siendo accionado.
Esto es necesario ya que las entradas del microcontrolador, no poseen un estado determinado y pueden ir a cualquiera de los dos estados posibles de forma aleatoria. Según la necesidad, se pueden colocar las resistencias en pines de entrada, obteniendo los siguientes efectos:

- **La resistencia pull-up:** mantiene un estado alto en el pin mientras el pulsador no es accionado, el estado activo del botón es BAJO (cuando se presiona).
- **La resistencia pull-down**: mantiene un estado bajo mientras el pulsador no es accionado, el estado activo del botón es ALTO (cuando se presiona).



El microcontrolador AVR incluido en el Arduino UNO, posee en cada pin de entrada una resistencia pull-up que se puede activar bajo control de nuestro programa, evitándose tener que colocar de forma externa este componente.

El siguiente ejemplo muestra cómo encender un led con un pulsador y apagar con otro botón. Es un ejemplo muy sencillo pero que ilustra muy bien el uso de las funciones pinMode() y digitalRead().



CÓDIGO

```
// ASIGNACIÓN DE PINES
const int pinon = 2;
const int pinoff = 3;
const int pinled = 13;

// VARIABLES DE ESTADO DE BOTONES
int estaon = HIGH;
int estaoff = HIGH;

void setup() {
  // CONFIGURAR PINES COMO ENTRADAS
  pinMode(pinon, INPUT);
  pinMode(pinoff, INPUT);
  // CONFIGURAR PIN DE LED COMO SALIDA
  pinMode(pinled, OUTPUT);
}

void loop() {
  // LEER EL ESTADO DE PINES DE BOTON A VARIABLES
  estaon = digitalRead(pinon);
  estaoff = digitalRead(pinoff);

  // SE OPRIMIO EL BOTON DE ENCENDIDO?
  if (estaon == LOW) {
    // ENTONCES ENCENDEMOS EL LED
    digitalWrite(pinled, HIGH);
  }

  // SE OPRIMIO EL BOTON DE APAGADO?
  if (estaoff == LOW) {
    // ENTONCES APAGAMOS EL LED
    digitalWrite(pinled, LOW);
  }
}
```

Utilidad de la lección

La utilidad o finalidad de esta lección es muy similar a la del potenciómetro, permitiéndonos enviar señales directas al Arduino, pero a través de un botón. Podríamos programarlo para que apagara el dispositivo o reiniciarlo.

LECCIÓN D: SERVOMOTORES

Un servomotor es un actuador rotativo o motor que permite un control preciso en términos de posición angular, aceleración y velocidad, capacidades que un motor normal no tiene. Utiliza un motor normal y lo combina con un sensor para la retroalimentación de posición.

Tienen un propósito diferente al de aquellos que son capaces de permanecer girando por tiempo indefinido. La diferencia con otros tipos de motores es que los servomotores no ofrecen un giro continuo, en un servo indicamos el ángulo de giro deseado del eje del motor y el servo se encarga de posicionarse en este ángulo. Tienen internamente unos topes y su giro está generalmente limitado de 0 a 180º (otros modelos más atípicos pueden realizar giros diferentes tales como los de 360 grados).

Particularidades

Construcción: los servomotores hechos de metal soportan generalmente mayor presión que los de plásticos y tienen más torque debido a los materiales de construcción.

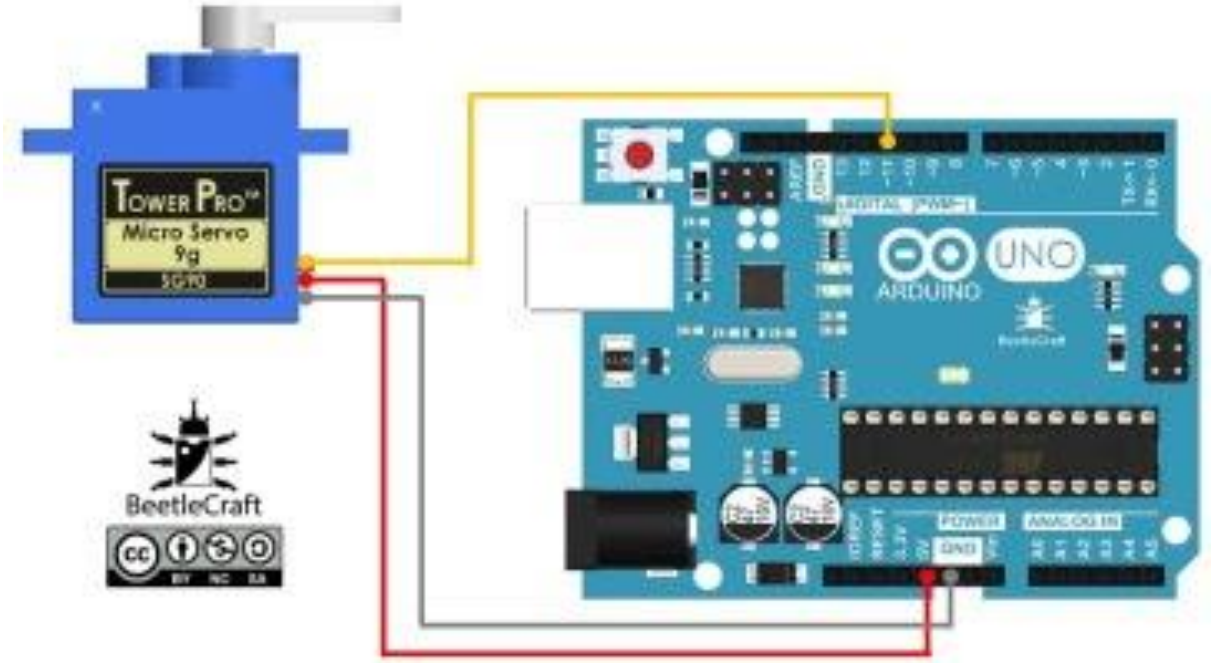
Torque/voltaje: cada servomotor tiene un torque distinto que es el peso que puede levantar y este puede variar dependiendo del voltaje que se le aplique al servomotor. Si se le aplica más de el voltaje máximo se puede quemar el servomotor.

Amperaje: Cada servo requiere una intensidad o corriente respectiva para funcionar correctamente, no debe superar los límites máximos y mínimos que soporta el servo.

Giro total: son los grados máximos de giro que permite el servomotor.

Velocidad: Se mide el tiempo que tarda el servo en llegar a una determinada posición, generalmente en tiempo/grados, siendo tiempo expresado en segundos.

Conexión



Amarillo: Es el cable de señal y debe conectarse a pines analógicos. (es similar al de la lección del potenciómetro)

Rojo: Es el cable de corriente y debe conectarse a algún pin entre 4v y 6v

Negro: Es el cable a masa y debe conectarse a GND

Librerías de Arduino para servomotores

Estas librerías vienen incluidas en Arduino IDE, recordemos que, para cargar la librería, y que, por ende, podamos hacer usos de las funciones, tenemos que llamarla con **#include <Servo.h>** (en este caso).

Las funciones de la librería son:

servo.attach(pin): Asocia al servo un pin y lo prepara para actuar, el pin se indica dentro del paréntesis.

servo.write(angulo): Cambia el valor de la señal PWM para mover el servo a una determinada posición 0 y 180, esta se indica dentro del paréntesis

writeMicroseconds(seg): Cambia el valor de la señal PWM para mover el servo a una determinada posición en microsegundos que se indican dentro del paréntesis. Es más fácil usar el comando **write().**

read(): Devuelve la posición del servo. (la posición no es la posición real si aún está en tránsito, sino la posición final).

attached(): Informa de si el servo está asociado al pin.

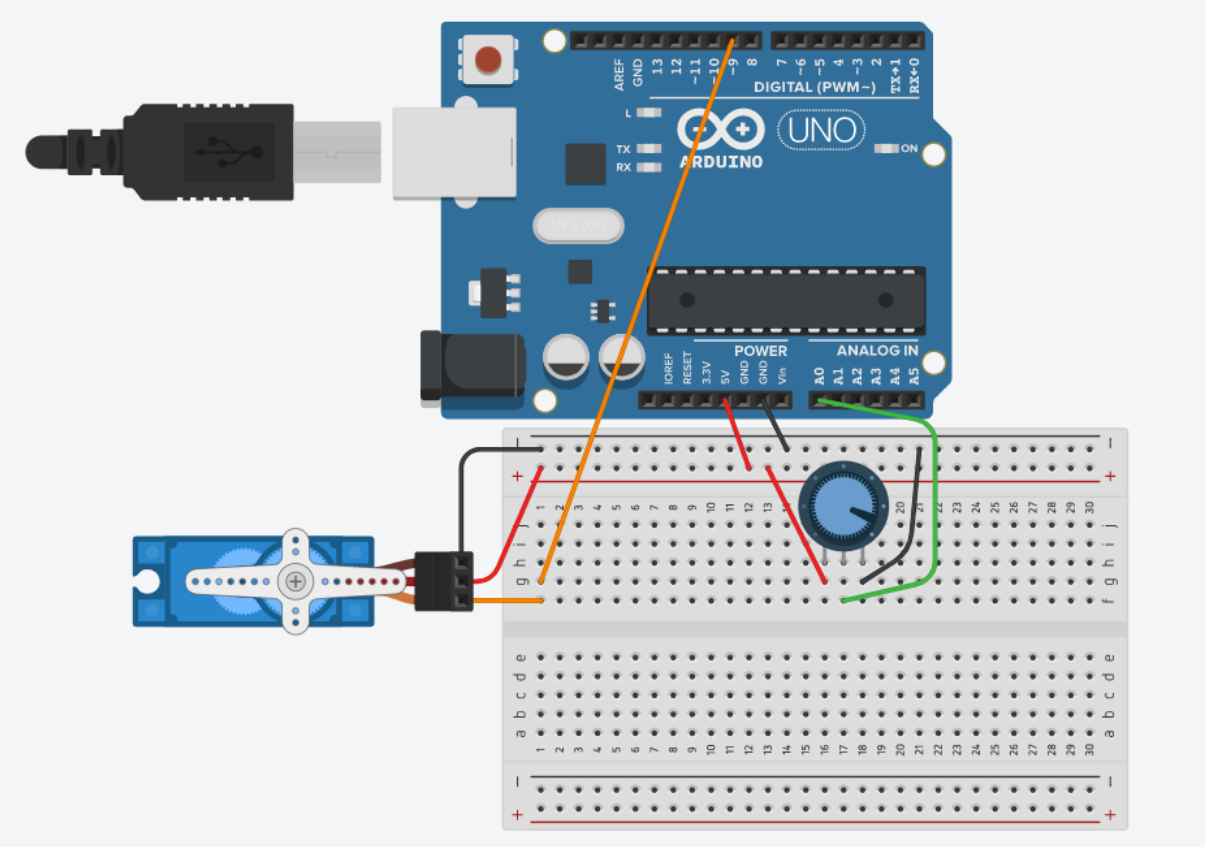
detach(): Desconecta y libera el pin del servo.

Ejemplos dentro de Arduino IDE

En el caso de esta lección el usuario dispondrá de ejemplos propios del IDE de Arduino en la sección de ejemplos ya tiene un par de ellos muy sencillos para ilustrar el manejo de la librería.

- Ejemplos->Servos->Knob. Control de la posición del servomotor con un potenciómetro.
- Ejemplos->Servos->Sweep. Movimientos de ida y vuelta entre ambos topes de giro desde 0º a 180º.

A continuación le mostraremos como conectar el circuito para hacer uso de los ejemplos provistos por el IDE de Arduino.



SERVOS

Si desea ver algunos ejemplos de servomotores, le dejamos a continuación una tabla comparativa.

Modelo Servo	Torque/Voltaje	Velocidad de giro	Ángulo Máximo	Tamaño	Peso
 SG90	1.6Kgxcm (4.8v) 1.8Kgxcm (6v)	0.12s/60º (4.8v) 0.10s/60º (6.0v)	180º	23mm x 12.2mm x 29mm	9g
	SG90: Es el servo que más se vende y el más barato. Es un microservo muy ligero y muy popular. Tiene engranajes de plástico				
 MG90	1.8kgxcm (4.8v) 2.2kgxcm (6v)	0.10s/60º (4.8v) 0.08s/60º (6v)	180º	22.8mm x 12.2mm x 28.5mm	13.4g
	MG90: Muy similar al primero pero más robusto y más rápido. Tiene engranajes metálicos.				
 Hitec HS422 (*)	3.3Kgxcm (4.8v) 4.1Kgxcm (6.0v)	0.21/60º (4.8v) 0.16/60º (6.0v)	180º	40.39mm x 19.56mm x 36.58mm	45.5g
	Hitec HS422: Es un servo muy popular en la gama de los 45.5 gramos y el fabricante ofrece información bastante completa. *(*) Sobre los consumos de corriente del servo Hitec HS422 véase más adelante.*				

Utilidad de la lección

A partir de aquí las lecciones que brindamos permitirán al usuario dotar de acciones concretas al robot que deseen construir, un ejemplo para el uso de servos puede ser la construcción de un brazo mecánico

LECCIÓN D: MOTORES PASO A PASO

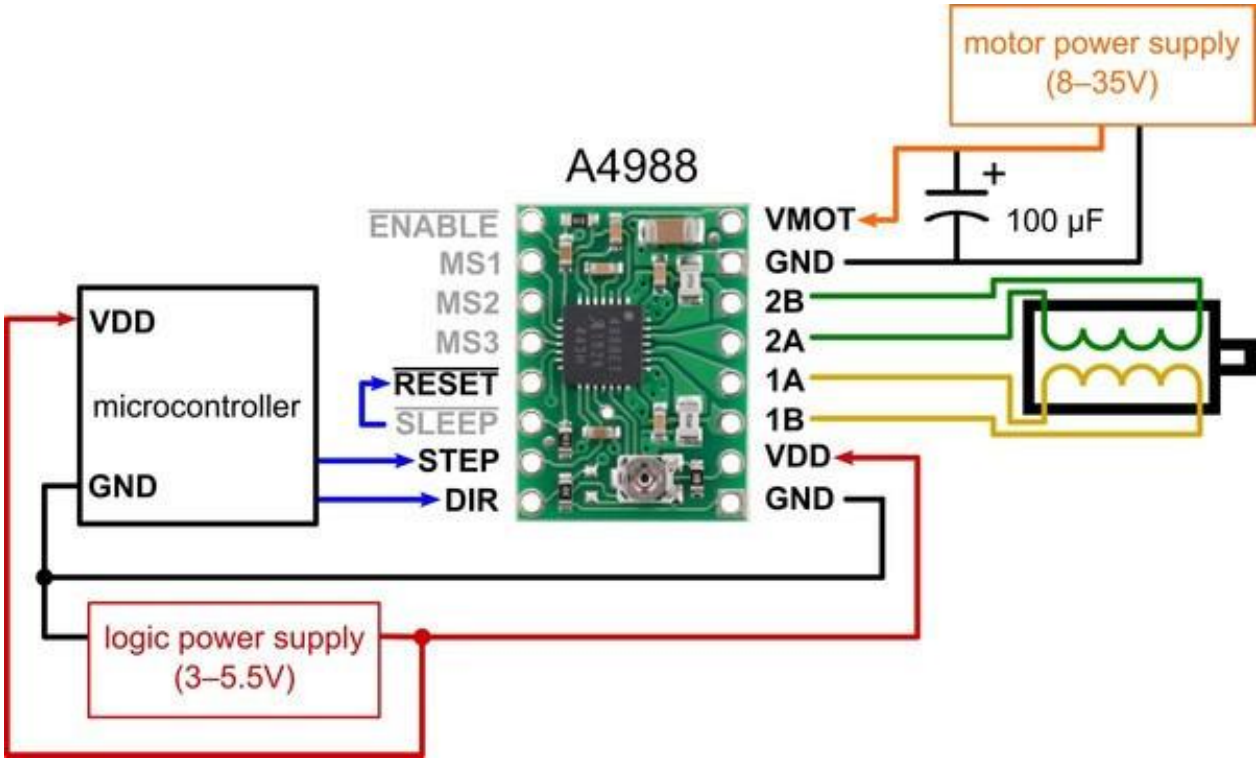
El motor paso a paso (Stepper) conocido también como motor de pasos es un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares discretos, lo que significa que es capaz de girar una cantidad de grados (paso o medio paso) dependiendo de sus entradas de control.

DRIVER

A la hora de trabajar con cualquier motor podemos, y en algunos casos debemos, incluir un driver controlador, Un driver para motores es un circuito que permite utilizar y controlar los motores de una forma muy simple, este actúa de interfaz entre la plaqueta controladora (Arduino), la fuente de alimentación externa, y el motor.

Estos controladores permiten manejar los voltajes e intensidades a los que se está suministrando al motor para así controlar la velocidad de giro, la posición e incluso la dirección de giro. Sirviendo de mediador entre la batería, el Arduino y el motor ya que, si se conecta el Arduino al motor, el primero no tendrá la potencia necesaria para moverlo, si se conecta la batería directamente al Arduino este se quemará y si se conecta la batería al motor no hay una parte lógica que le indique cuando y cuanto debe moverse.

Cómo se conectan



A simple vista el conexionado del driver puede verse engorroso, pero vamos paso a paso. Primero tenemos que distinguir los componentes que interacción con el driver directamente, estos son: el motor (parte negra derecha), la fuente del motor (parte naranja) y el Arduino o el esp32, que en este caso es, la fusión entre la fuente lógica (parte roja) y el microcontrolador (parte negra izquierda). En el esquema aparecen estos 2 últimos componentes separados porque, no todos los Arduino son capaces de entregar entre 3 y 5.5V, es por eso que aparece separados, pero si tenemos un pin del Arduino que entrega la tensión requerida, no necesitaremos de la fuente lógica.
Por último, notaremos que haremos uso de una fuente externa superior a 8V, esto se debe a que el Arduino por sí solo no es capaz de mover el motor, por lo que se requiere de una fuente externa que suministre la energia necesaria al motor en cuestion.

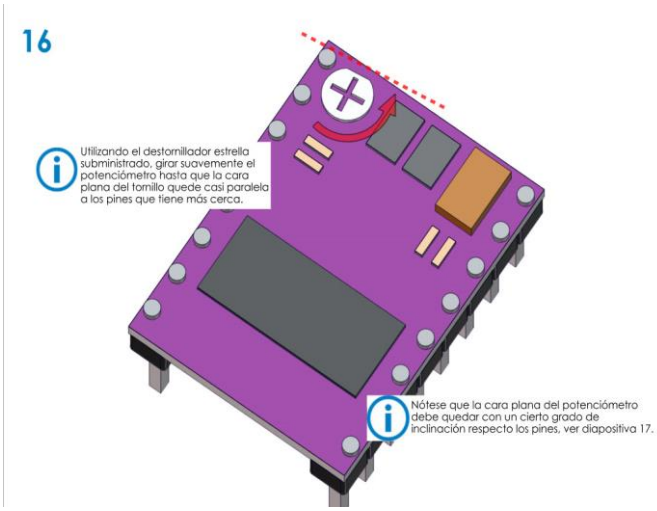
A continuación, explicaremos para que sirve cada pin del driver:

VDD: este pin se conecta al positivo de nuestro Arduino
GND: este pin se conecta a tierra (es la misma para los 2 GND)
VMOT: Este pin se conecta al positivo de la fuente externa
RESET Y SLEEP: Se conectan juntos, sirven para entrar en 2 estados especiales, de esta manera se anulan.
1A, 1B, 2A Y 2B: estos se conectan a los 4 pines que tiene el motor.
MS1, MS2 Y MS3: estos pines son de los micro pasos, si no se utilizaran micro pasos no es necesario conectarlos (véase la tabla de micro pasos)
ENABLE: este pin no se conecta a nada, si se conecta a una salida positiva o se le da corriente, el dispositivo se deshabilitará (no se rompe).

REGULACION

El driver cuenta con un tornillo que al ajustarlo se modifica el valor de la resistencia y por lo tanto el amperaje que pasa por él. Si se ve como una canilla, la resistencia seria la canilla y el amperaje el agua que sale. Los drivers debemos ajustarlos o regularlos, para que suministre la corriente justa al motor, no tiene pasarse del máximo que admite, ni del mínimo necesario, ya que, en el caso que necesitemos que este sea preciso, aparecerá un problema conocido como “perdida de pasos”. Sin mencionar que si no recibe la corriente necesaria no tendrá fuerza de torque(empuje) y si nos pasamos podría quemarse.

Para regularlo tenemos que conectar el tester puesto en modo voltímetro y medir entre el positivo que viene de la fuente (VMOT) y la cabeza del tornillo. Para saber que amperaje requiere nuestro motor tenemos que ver la información que nos provee el fabricante.



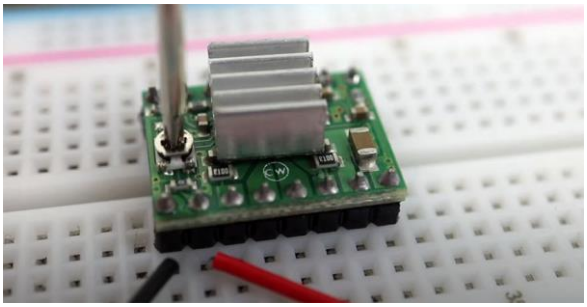


Tabla para saber cantidad de corriente max

Modelo	Rs	Fórmula reducida
A4988	50	I_max = 0,625 * V_ref
A4988	100	I_max = 1,25 * V_ref
A4988	200	I_max = 2 ,2* V_ref
DRV8825	100	I_max = 2 * V_ref

Una vez que sabemos que corriente máxima soporta el motor lo único que tenemos que hacer es despejar la formula, para saber que tenemos que medir con el tester.

Modelos
En particular, el DRV8825 permite trabajar con tensiones superiores al A4988 (45V frente a 35V), e intensidades superiores (2.5A frente a 2A). Además, añade un nuevo modo de micropasos (1/32) que no está presente en el A4988.

MICROPASOS

La técnica de micro pasos, consiste en que un motor paso a paso alcance posiciones intermedias entre un paso y un medio paso. De esta manera, en un motor de 1,8° por paso, por ejemplo, realizando 8 micro pasos por paso se podría obtener, movimientos cada 0,225°. En la foto de conexiones mostrada a continuación se ven los pines de micro pasos (MS1, MS2, MS3).

Si se utilizan los micro pasos se le necesitaran indicar más cantidad de pasos para realizar la misma vuelta, ya que el motor avanzara menos.

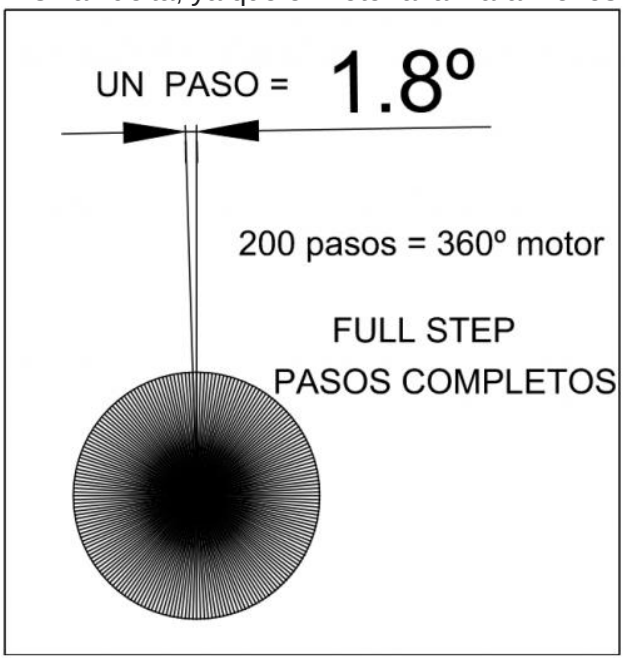


Tabla de micro pasos

Resolucion		Pines M0, M1 y M2		
A4988	**DRV8825**	**MODE0**	**MODE1**	**MODE2**
Full step	Full step	Low	Low	Low
1/2 step	1/2 step	High	Low	Low
1/4 step	1/4 step	Low	High	Low
1/8 step	1/8 step	High	High	Low
-	1/16 step	Low	Low	High
-	1/32 step	High	Low	High
-	1/32 step	Low	High	High
1/16 step	1/32 step	High	High	High

Dejar todos los pines desconectados dará lugar a usar **modo Full Step**

CÓDIGO

```
const int dirPin = 8;
const int stepPin = 9;
const int steps = 200;
int stepDelay;
void setup() {
  // Marcar los pines como salida
  pinMode(dirPin, OUTPUT);
  pinMode(stepPin, OUTPUT);
}
void loop() {
  //Activar una direccion y fijar la velocidad con stepDelay
  digitalWrite(dirPin, HIGH);
  stepDelay = 250;
  // Giramos 200 pulsos para hacer una vuelta completa
  for (int x = 0; x < steps * 2; x++) {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(stepDelay);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(stepDelay);
  }
  delay(1000);
  //Cambiamos la direccion y aumentamos la velocidad
  digitalWrite(dirPin, LOW);
  stepDelay = 150;
  // Giramos 400 pulsos para hacer dos vueltas completas
  for (int x = 0; x < 400; x++) {
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(stepDelay);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(stepDelay);
  }
  delay(1000);
}
```

Utilidad de la lección

Con esta lección podremos crear las bases mecánicas de complejos mecanismos que van desde autos a control remotos hasta impresoras 3D y CNCs.

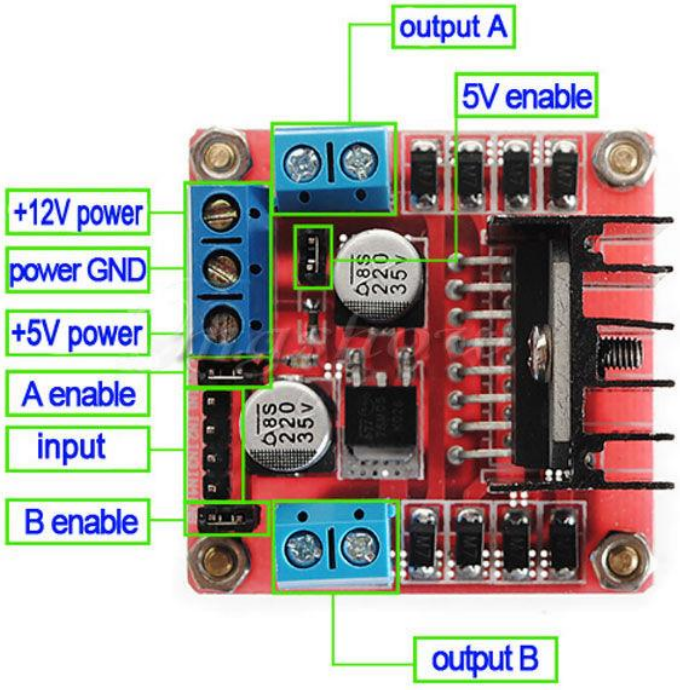
LECCIÓN E: MOTORES DC

Este tipo de motores son los más fáciles de entender, ya que, si bien tienen las mismas características que el paso a paso y el funcionamiento es más simplificado, como contraparte, podemos encontrar que son menos precisos y ofrecen menos funcionalidades.

Como primera diferencia notable podemos encontrar que estos tienen solo 2 terminales a diferencia de los paso a paso que tienen 4. Además, como podemos ver en las imágenes anteriores, los motores DC pueden incluir reguladores (véase la imagen derecha). El regulador es una caja adicional que se le agrega al motor con el fin de aumentar su potencia de torque (véase el apartado de regulación de motores paso a paso), aunque esto reduce significativamente su velocidad. Otra diferencia, no visible a simple viste, es que no podemos controlar la dirección, la velocidad de giro, ni la posición, a no ser que tengamos un driver. En el caso del driver, utilizaremos el L298N.

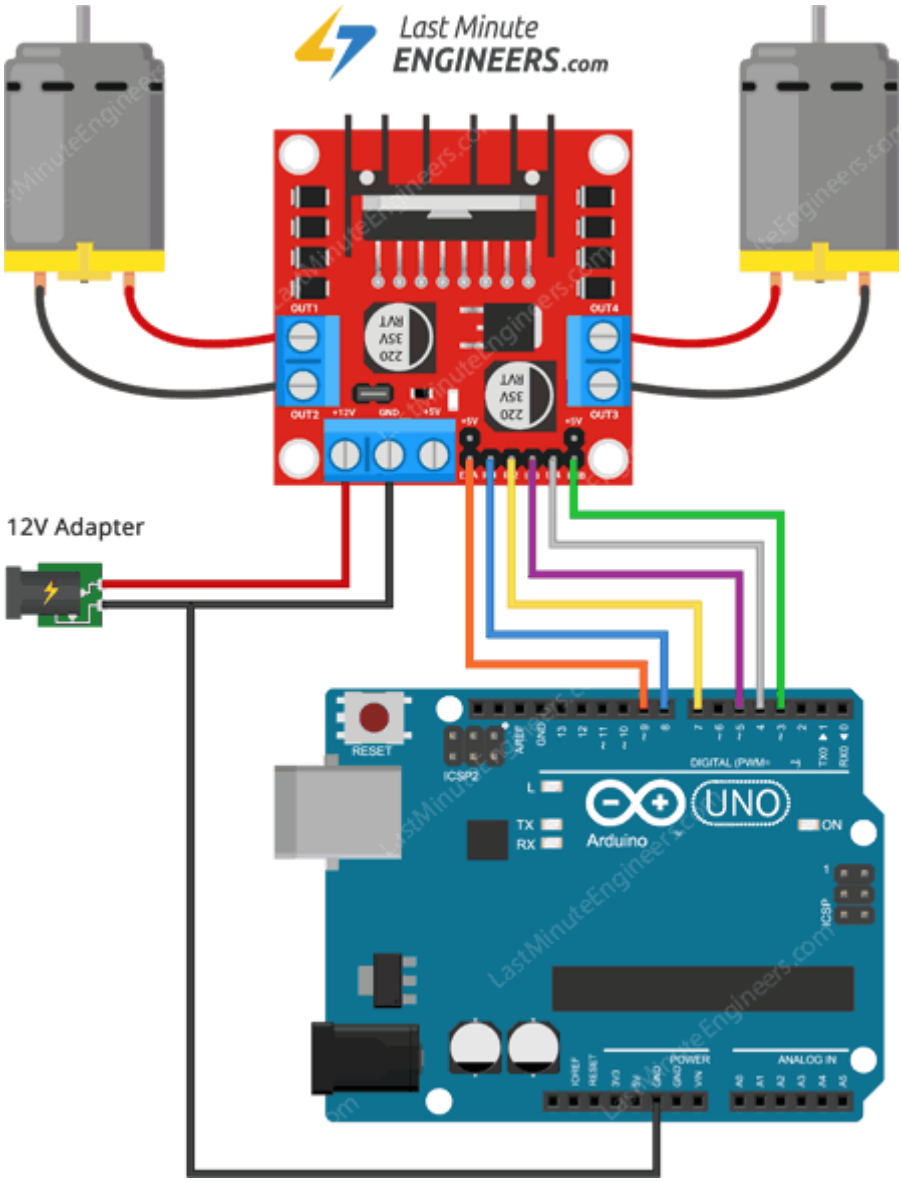
DRIVER

Como mencionamos anteriormente, usaremos para este tipo de motores el L298N, al igual que con el pololu (A4988), este driver añade nuevas funcionalidades al motor y nos permite añadirle una fuente externa para suministrar correctamente los motores.



Algo que necesitamos aclarar es que si bien, podemos agregarle el driver, no es necesario, ya que, a diferencia de los paso a paso, este tipo de motores pueden usarse solo con el pin de 3.3v y 5v del Arduino(UNO). Pero si es recomendable que se incluya, ya que, le brinda al motor potencia extra y como se dijo anteriormente añade funciones adicionales, como: El control de velocidad, dirección de giro y fuerza de torque.

Cómo se conectan

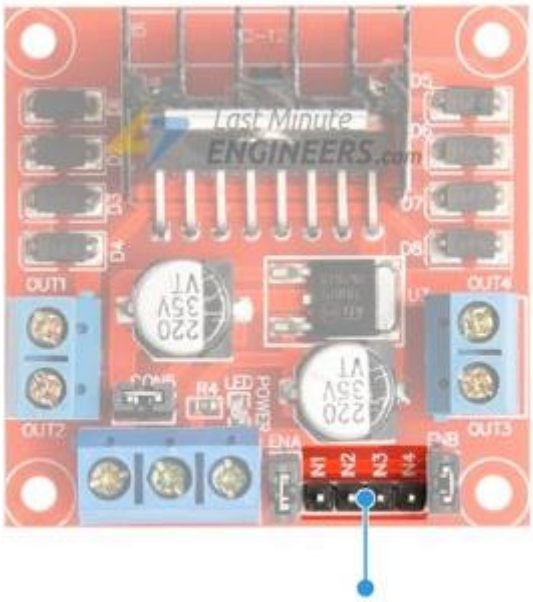


Los cables que salen de “INPUT” (VEASE EL DIAGRAMA DEL DRIVER), se conectan a los pines digitales de Arduino y cada uno cumple una función específica:
Los 4 cuatro del centro (azul, amarillo, violeta y gris) corresponden a los pines de control, Usando los pines de control de dirección, podemos controlar si el motor gira hacia adelante o hacia atrás. Estos pines controlan realmente los interruptores del circuito H-Bridge dentro del IC L298N.

El módulo tiene dos pines de control de dirección para cada canal. Los pines IN1 e IN2 controlan la dirección de giro del motor A mientras que IN3 e IN4 controlan el motor B.

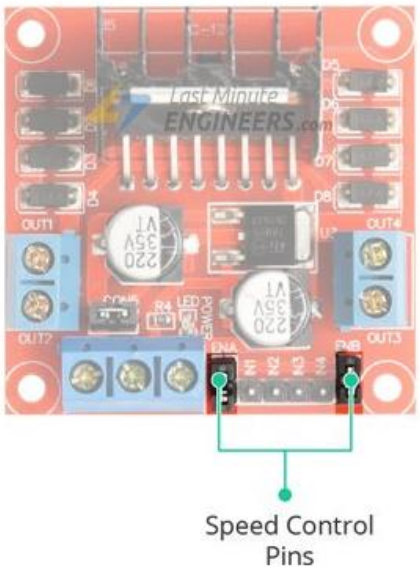
La dirección de giro de un motor se puede controlar aplicando una lógica ALTA (5 voltios) o una lógica BAJA (tierra) a estas entradas. La siguiente tabla ilustra cómo se hace esto.

Input1	Input2	Spinning Direction
Low(0)	Low(0)	Motor OFF
High(1)	Low(0)	Forward
Low(0)	High(1)	Backward
High(1)	High(1)	Motor OFF



Direction Control Pins

Los pines de control de velocidad a saber. ENA y ENB se utilizan para encender y apagar los motores y controlar su velocidad.
Poner estos JUMPERS en HIGH hará que los motores giren, y ponerlos en LOW hará que se detengan. Pero, con Pulse Width Modulation (PWM), podemos controlar la velocidad de los motores.
El módulo generalmente viene con un puente en estos pines. Cuando este puente está en su lugar, el motor se habilita y gira a la velocidad máxima. Si desea controlar la velocidad de los motores mediante programación, debe quitar los JUMPERS y conectarlos a los pines habilitados para PWM en Arduino.



Speed Control Pins

CÓDIGO

```
// Motor A connections
int enA = 9;
int in1 = 8;
int in2 = 7;
// Motor B connections
int enB = 3;
int in3 = 5;
int in4 = 4;

void setup() {
	// Set all the motor control pins to outputs
	pinMode(enA, OUTPUT);
	pinMode(enB, OUTPUT);
	pinMode(in1, OUTPUT);
	pinMode(in2, OUTPUT);
	pinMode(in3, OUTPUT);
	pinMode(in4, OUTPUT);

	// Turn off motors - Initial state
	digitalWrite(in1, LOW);
	digitalWrite(in2, LOW);
	digitalWrite(in3, LOW);
	digitalWrite(in4, LOW);
}

void loop() {
	directionControl();
	delay(1000);
	speedControl();
	delay(1000);
}

// This function lets you control spinning direction of motors
void directionControl() {
	// Set motors to maximum speed
	// For PWM maximum possible values are 0 to 255
	analogWrite(enA, 255);
	analogWrite(enB, 255);

	// Turn on motor A & B
	digitalWrite(in1, HIGH);
	digitalWrite(in2, LOW);
	digitalWrite(in3, HIGH);
	digitalWrite(in4, LOW);
	delay(2000);

	// Now change motor directions
	digitalWrite(in1, LOW);
	digitalWrite(in2, HIGH);
	digitalWrite(in3, LOW);
	digitalWrite(in4, HIGH);
	delay(2000);

	// Turn off motors
	digitalWrite(in1, LOW);
	digitalWrite(in2, LOW);
	digitalWrite(in3, LOW);
	digitalWrite(in4, LOW);
```

```
}

// This function lets you control speed of the motors
void speedControl() {
	// Turn on motors
	digitalWrite(in1, LOW);
	digitalWrite(in2, HIGH);
	digitalWrite(in3, LOW);
	digitalWrite(in4, HIGH);

	// Accelerate from zero to maximum speed
	for (int i = 0; i < 256; i++) {
		analogWrite(enA, i);
		analogWrite(enB, i);
		delay(20);
	}

	// Decelerate from maximum speed to zero
	for (int i = 255; i >= 0; --i) {
		analogWrite(enA, i);
		analogWrite(enB, i);
		delay(20);
	}

	// Now turn off motors
	digitalWrite(in1, LOW);
	digitalWrite(in2, LOW);
	digitalWrite(in3, LOW);
	digitalWrite(in4, LOW);
}
```

Utilidad de la lección

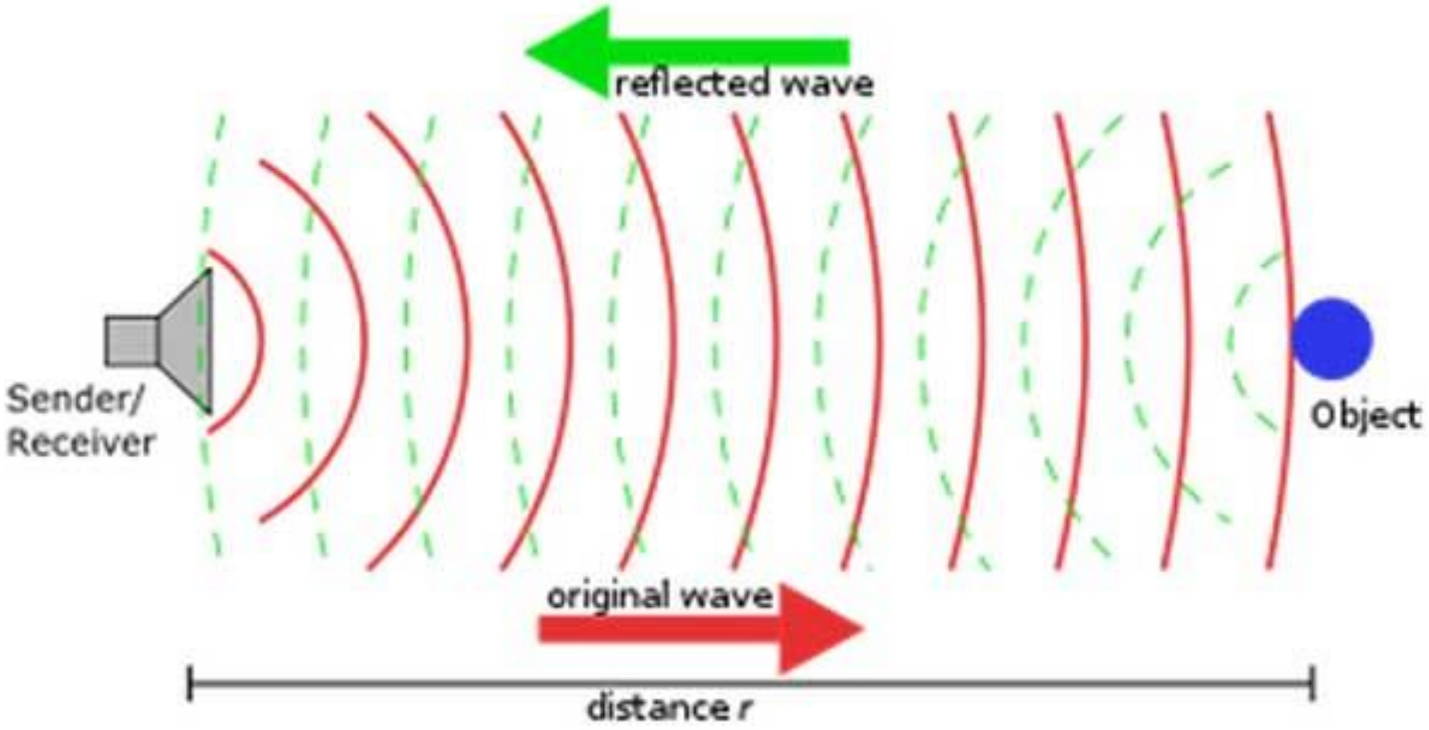
Como se mencionó anteriormente, estos motores tienen menos funcionalidades que un paso a paso, pero, igualmente pueden usarse para crear autos a control remoto o autónomos, básicamente se puede usar para cualquier mecanismo que no requiera controlar la posición exacta del motor, solo que requiera que gire.

LECCIÓN F: SENSOR ULTRASONICO HC-SR04

El HC-SR04 es un sensor del tipo ultrasónico, el cual es capaz de detectar la distancia a un objeto mediante la emisión sonido y recepción de sonido, como si fuera un murciélago.

Su funcionalidad en su mayoría se debe a los dos transductores que posee: un emisor y un receptor piezoeléctricos, además de la electrónica necesaria para su operación. El funcionamiento del sensor es el siguiente:

El emisor piezoeléctrico emite 8 pulsos de ultrasonido(40KHz) luego de recibir la orden en el pin TRIG, las ondas de sonido viajan en el aire y rebota al encontrar un objeto, el sonido de rebote es detectado por el receptor piezoeléctrico, luego el pin ECHO cambia a Alto (5V) por un tiempo igual al que demoró la onda desde que fue emitida hasta que fue detectada, el tiempo del pulso ECO es medido por el microcontrolador y así se puede calcular la distancia al objeto.

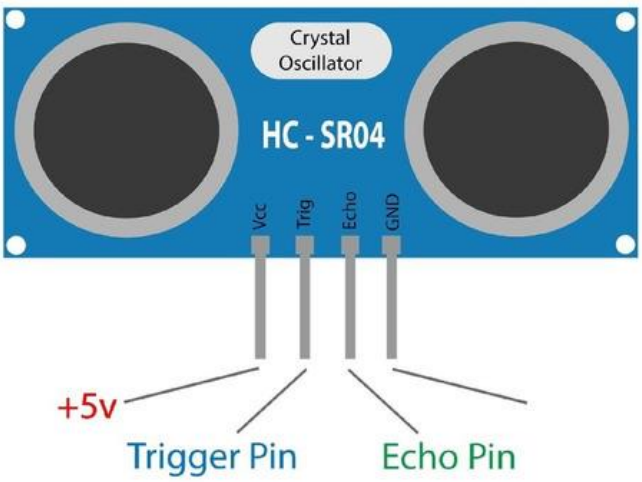


El funcionamiento del sensor no se ve afectado por la luz solar o material de color negro (aunque los materiales blandos acústicamente como tela o lana pueden llegar a ser difíciles de detectar).

La distancia se puede calcular utilizando la siguiente fórmula:

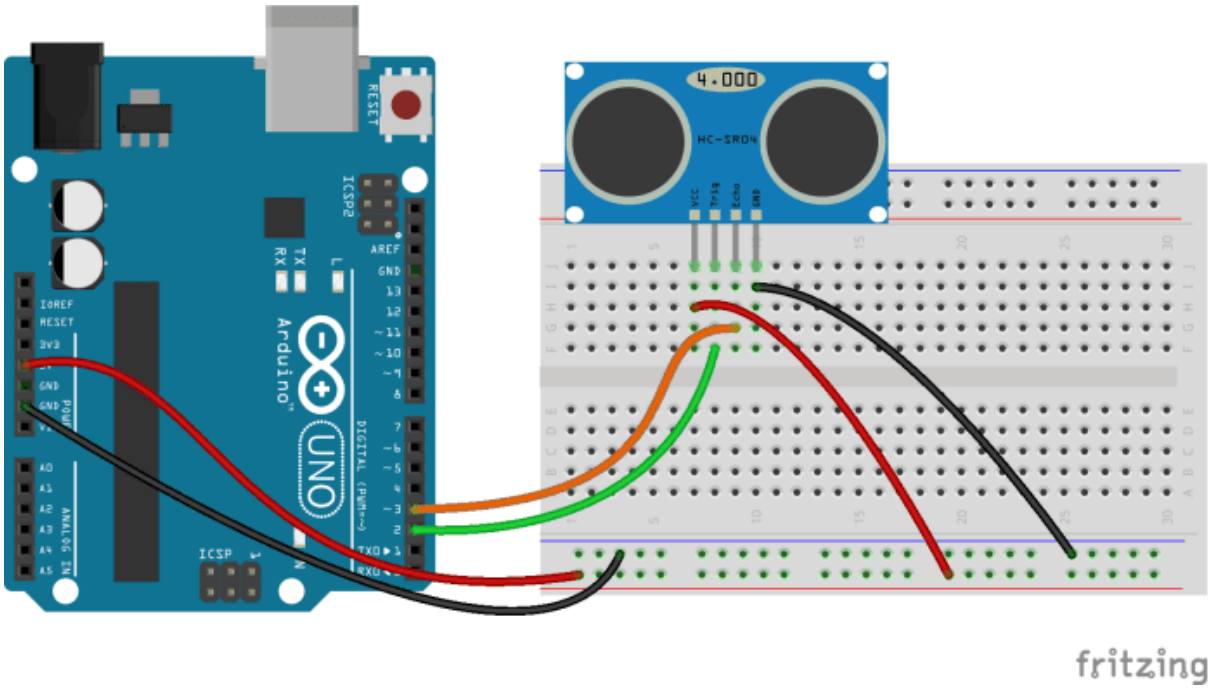
Distancia(m) = {(Tiempo del pulso ECO) * (Velocidad del sonido=340m/s)}/2

El dispositivo consta de 4 terminales, véase la siguiente imagen.



5V y GND: Al igual que los otros dispositivos que vimos corresponden al positivo y negativo.
TRIGGER: Recibe un pulso de habilitación de parte del microcontrolador, mediante el cual se le indica al módulo que comience a realizar la medición de distancia.
ECHO: El sensor devuelve al microcontrolador un pulso cuyo ancho es proporcional al tiempo que tarda el sonido en viajar del transductor al obstáculo y luego de vuelta al módulo.

Cómo se conectan



CÓDIGO

```
const int pinTrigger = 2;  //Variable que contiene el número del pin al cual conectamos la señal "trigger"
const int pinEcho = 3;    //Variable que contiene el número del pin al cual conectamos la señal "echo"

void setup() {
  Serial.begin(9600);//Configuramos la comunicación serial
  pinMode(pinTrigger, OUTPUT); //Configuramoms el pin de "trigger" como salida
  pinMode(pinEcho, INPUT);  //Configuramoms el pin de "echo" como entrada
  digitalWrite(pinTrigger, LOW);//Ponemos en voltaje bajo(0V) el pin de "trigger"
}

void loop()
{

  unsigned long t; //Variable de tipo unsigned long que contendrá el tiempo que le toma a la señal ir y regresar
  float d; //Variable de tipo float que contendrá la distancia en cm

  digitalWrite(pinTrigger, HIGH);//Ponemos en voltaje alto(5V) el pin de "trigger"
  delayMicroseconds(10);//Esperamos en esta línea para conseguir un pulso de 10us
  digitalWrite(pinTrigger, LOW);//Ponemos en voltaje bajo(0V) el pin de "trigger"

  t = pulseIn(pinEcho, HIGH);         //Utilizamos la función  pulseIn() para medir el tiempo del pulso/echo
  d = t * 0.000001 * 34300.0 / 2.0; //Obtenemos la distancia considerando que la señal recorre dos veces la distancia a medir y que la velocidad del sonido es 343m/s

  Serial.print("Distancia: ");
  Serial.print(d);
  Serial.print("cm");
  Serial.println();
  delay(100);          //Nos mantenemos en esta línea durante 100ms antes de terminar el loop
}
```

ACLARACIONES

unsigned long: tipo de dato para enteros sin signo (32 bits), números sin punto decimal. Los enteros largos puede ser tan grandes como 4294967295 y tan pequeños como 0. Son almacenados como 32 bits (4 bytes) de información.

pulsein(): Lee un pulso (HIGH o LOW) en un pin. Por ejemplo, si el valor es HIGH , pulseIn() espera a que el pin se ponga HIGH , comienza a cronometrar, luego espera a que el pin pase a LOW y detiene el cronometraje. Devuelve la longitud del pulso en microsegundos. Finaliza y devuelve 0, si no se inicia ningún pulso dentro de un tiempo de espera especificado.

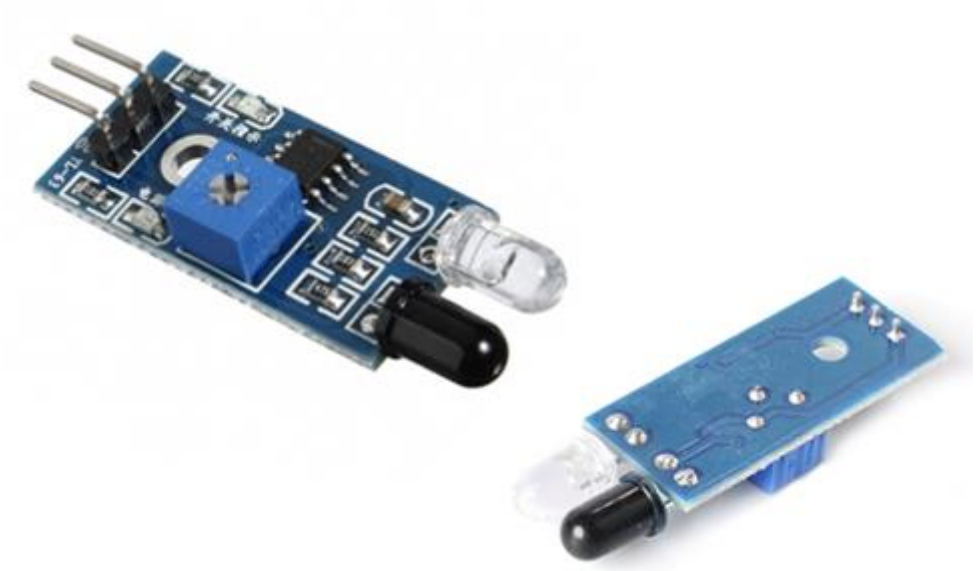
La sincronización de esta función se ha determinado empíricamente y probablemente mostrará errores en pulsos más largos. Funciona en pulsos de 10 microsegundos a 3 minutos de duración.

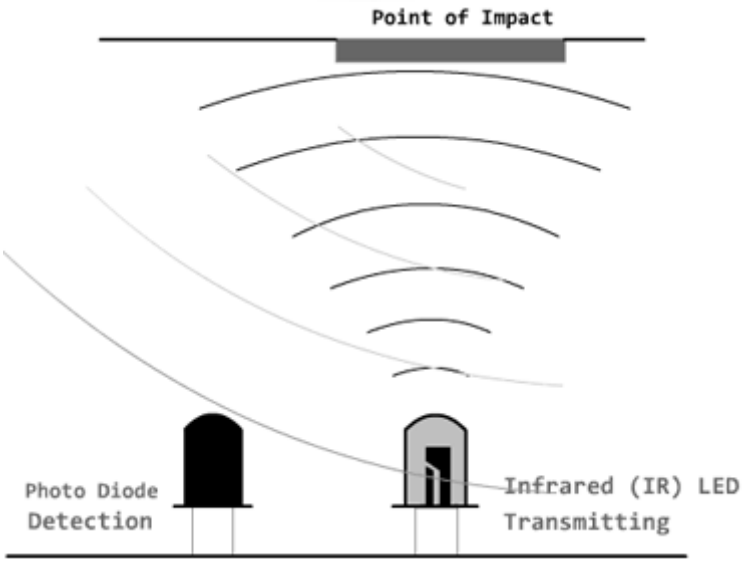
Utilidad de la lección

Este dispositivo nos permite dotar a nuestro sistema, o en este caso, robot, de la función de detectar objetos y medir la distancia que existe entre el y ellos. Llevado a la práctica, el ejemplo más sencillo que podemos encontrar esta en los robots de limpieza, los cuales detectan los obstáculos y los esquivan.

LECCIÓN G: SENSOR INFRARROJO

Un detector de obstáculos infrarrojo es un dispositivo que detecta la presencia de un objeto mediante la reflexión que produce en la luz. El uso de luz infrarroja (IR) es simplemente para que esta no sea visible para los humanos. Constitutivamente son sensores sencillos. Se dispone de un LED emisor de luz infrarroja y de un fotodiodo (tipo BPV10NF o similar) que recibe la luz reflejada por un posible obstáculo.

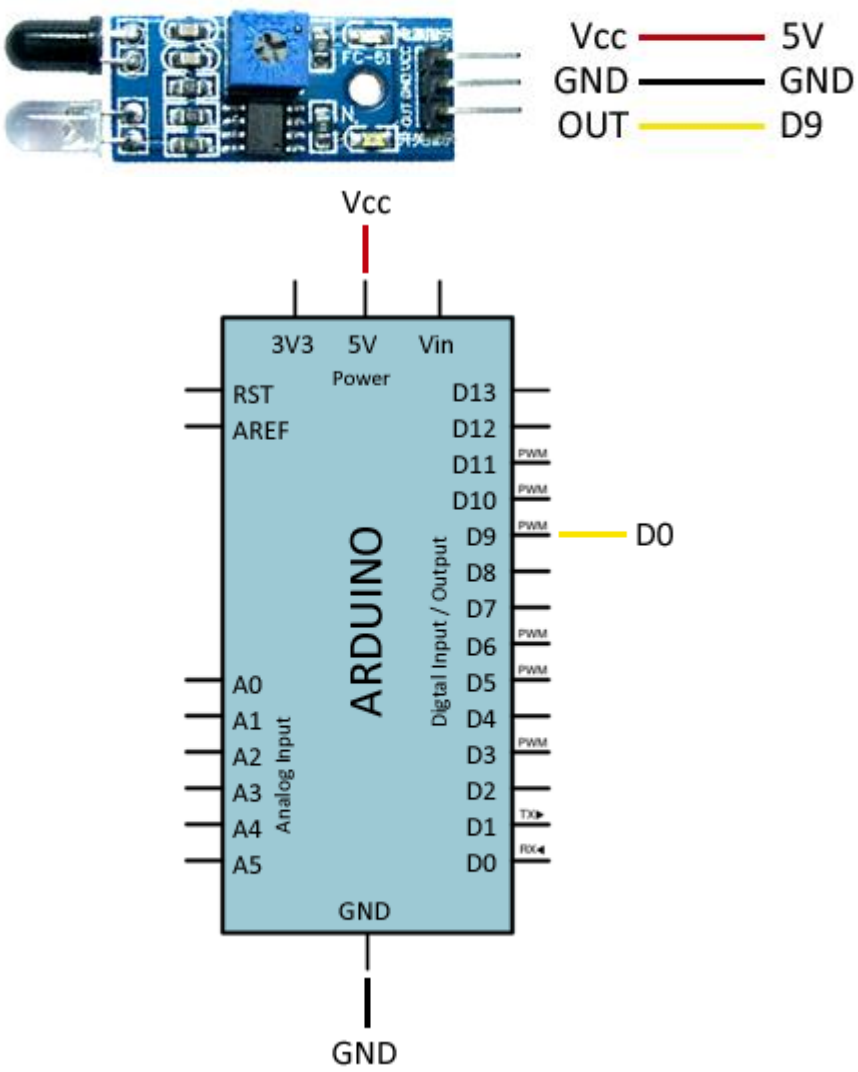




Los detectores de obstáculo suelen proporcionarse con una placa de medición estándar con el comparador LM393, que permite obtener la lectura como un valor digital cuando se supera un cierto umbral, que se regula a través de un potenciómetro ubicado en la placa. Con potenciómetro nos referimos al tornillo que se encuentra en la parte superior (véase la imagen anterior), es similar a los drivers de los motores paso a paso.

Este tipo de sensores actúan a distancias cortas, típicamente de 5 a 20mm. Además, la cantidad de luz infrarroja recibida depende del color, material, forma y posición del obstáculo, por lo que no disponen de una precisión suficiente para proporcionar una estimación de la distancia al obstáculo. Para calibrar este umbral de disparo tenemos que acercar un objeto al detector de obstáculos y regulando la salida digital con el potenciómetro. Podemos saltarnos este paso dejando el potenciómetro en un valor medio.

Cómo se conectan



CÓDIGO

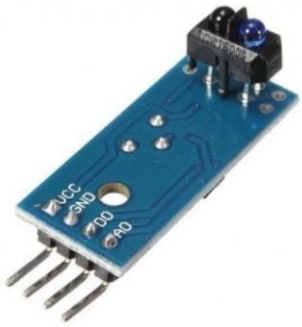
```
const int sensorPin = 9;

void setup() {
  Serial.begin(9600);   //iniciar puerto serie
  pinMode(sensorPin , INPUT);  //definir pin como entrada
}

void loop(){
  int value = 0;
  value = digitalRead(sensorPin );  //lectura digital de pin

  if (value == HIGH) {
     Serial.println("Detectado obstaculo");
  }
  delay(1000);
}
```

Utilidad de la lección

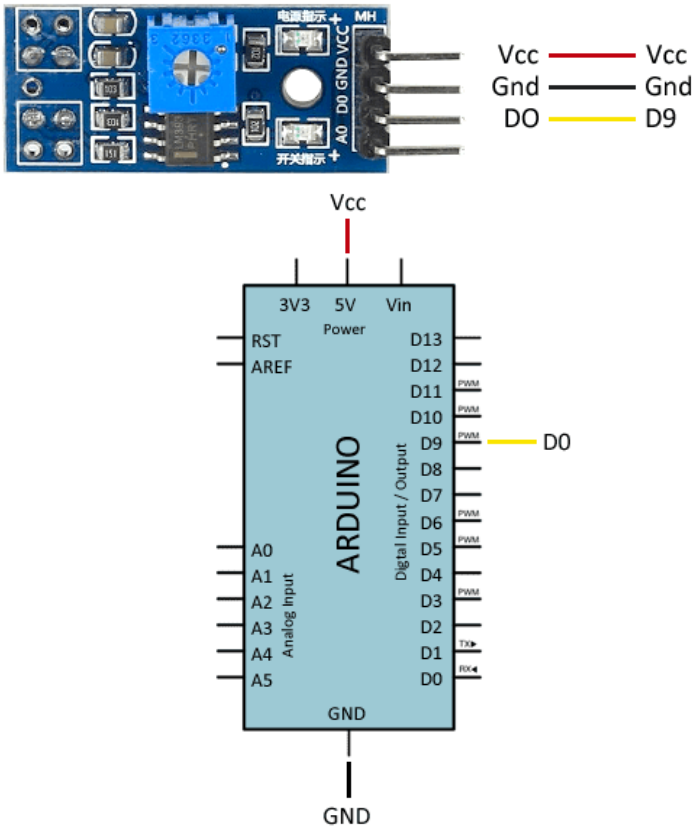
Este tipo de dispositivos son utilizados para la detección de obstáculos en pequeños vehículos o robots. Su bajo coste hace que sea frecuente ubicarlos en el perímetro, de forma que detectemos obstáculos en varias direcciones.
También son útiles en otro tipo de aplicaciones como, por ejemplo, detectar la presencia de un objeto en una determinada zona, determinar una puerta está abierta o cerrada, o si una máquina ha alcanzado un cierto punto en su desplazamiento.

LECCIÓN H: SENSOR TRACKER O SEGUIDOR DE LINEA

En este caso veremos el TCRT5000L, el cual es un modelo de sensor infrarrojo que posee la función de detectar la el cambio de color en un objeto, así como también poder medir la distancia entre el objeto que detecta, por lo que, se podría tomar como una versión económica del HC-SR04. Pero tiene un rango de detección muy bajo, de 0,5 mm a 15mm.



Cómo se conectan



Como se mencionó anteriormente, este infrarrojo es capaz de medir la distancia a la que se encuentra el objeto, para habilitar esta función debemos conectar el pin A0 del dispositivo con algún pin analógico del Arduino, por ejemplo, el A1.

Utilidad de la lección

Con estos dispositivos podemos crear dispositivos que distingan ciertos patrones y que actúen en base a esta detección, Hoy en día podemos encontrar robots que siguen líneas o circuitos, llevado a gran escala podemos incluso compararlo con los autos inteligentes.

CÓDIGO

```
const int sensorPin = 9;

void setup() {
  Serial.begin(9600);   //iniciar puerto serie
```

```
  pinMode(sensorPin, INPUT);  //definir pin como entrada
}

void loop(){
  int value = 0;
  value = digitalRead(sensorPin );  //lectura digital de pin

  if (value == LOW) {
     Serial.println("TCRT5000L activado");  //zona oscura
  }
  delay(1000);
}
}
```

LECCIÓN I: PILAS Y BATERIAS

Para nuestros proyectos autónomos con Arduino necesitaremos de fuente de energía autónomas para suministrar a todo el sistema, es aquí donde entran las pilas y baterías. Lo que debemos tener en cuenta a la hora de utilizar una de estas es, el amperaje que entregan y la tensión en la que trabajan, ya que, tenemos que calcular las tensiones y amperajes totales del sistema para saber con qué tipo de pilas o baterías deberíamos utilizar.

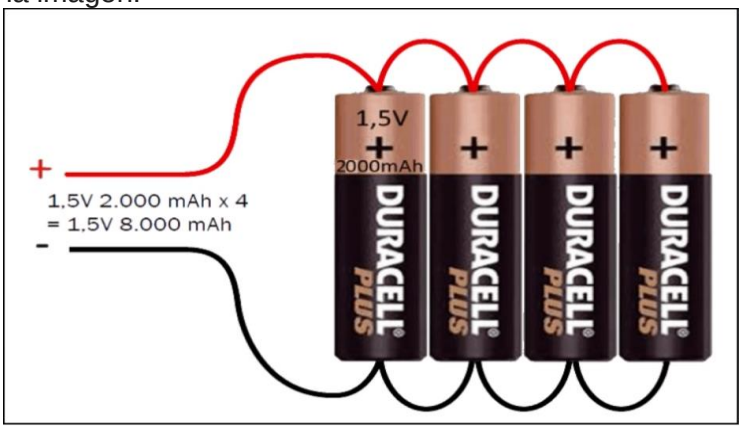
Sobre el amperaje tenemos que tener en cuenta, el mAh significa el amperaje dará durante una hora y luego de esto comenzará a disminuir gradualmente hasta que se acabe su vida útil. En caso de que sea, una pila recargable o batería, se necesitara recargar.

Otra posible necesidad es alimentar el Arduino con baterías. En este apartado hay que considerar el voltaje de operación de la placa Arduino y el tiempo de operación que requerimos, para elegir el

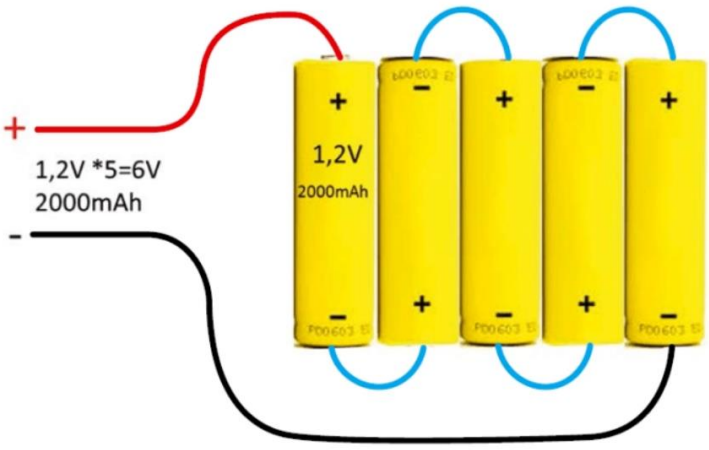
tipo de batería adecuado según la aplicación. Los siguientes son algunos tipos de baterías comunes:

En algunos casos puede ser necesario aumentar el voltaje de operación o la capacidad total de batería, por esto es importante conocer el funcionamiento de los arreglos en serie y paralelo de baterías.

En caso de requerir aumentar la capacidad de las baterías, se puede utilizar un arreglo de baterías conectadas en paralelo, es decir, conectando los positivos de las baterías todos juntos, como se muestra en la imagen:



En caso de que se requiera obtener un voltaje mayor, se pueden utilizar baterías conectadas en serie. Por ejemplo, para obtener voltaje para alimentar un arduino UNO se pueden utilizar 5 baterías de 1.2 volts recargables.



Baterías de ejemplo

- Podemos utilizar una batería de plomo – ácido a 12 volts y alimentar el Arduino a través del jack de alimentación externa. Para alimentar nuestra tarjeta de esta manera, es aconsejable realizar un cable especial con el plug invertido soldado en un extremo y terminales adecuadas según la batería.
- Utilizando un arreglo de 5 o 6 baterías recargables AA NiMH o un arreglo de 4 baterías alcalinas podemos alimentar el Arduino a través del pin VIN. Podemos utilizar una pila cuadrada de 9 volts con un plug invertido para alimentar el Arduino a través del jack de alimentación externa. Aunque este método no es eficiente y debemos esperar una vida no muy larga de la pila.
- Podemos usar un arreglo de baterías de Polímero de Litio (lipo) de 7.4 volts (2 celdas en serie) para alimentar nuestro Arduino. Normalmente estos arreglos ya se venden hechos como una sola unidad de alta capacidad.

¿Cuál es el voltaje de alimentación del Arduino?

Arduino es una placa de prototipado y uno de los elementos que incluye para ayudar a esa condición es un regulador de voltaje, por ello normalmente se recomienda alimentar Arduino entre 7v y 12v para placas con tensión nominal de 5v.

¿Qué sucede si se alimenta la placa de Arduino con exceso de corriente eléctrica?

Si conectamos demasiada carga, la placa Arduino suele tener un comportamiento anómalo pudiendo se resetear el micro.

¿Dónde se conecta?

En el caso de querer suministrar la plaqueta de Arduino Uno, tendremos que conectar la fuente de al conector que se muestra en la siguiente imagen.

ARMADO DEL ROBOT

Hemos llegado al tramo final, aquí llevaremos a la práctica todo lo visto anteriormente, y concretaremos un proyecto con Arduino, basándonos en la teoría provista por el manual. Si bien este manual abarca los dispositivos más conocidos de Arduino, para este armado no se utilizarán todas las piezas vistas anteriormente. A continuación, ensamblaremos un robot autónomo detector de obstáculos, se requerirá:

A. ARDUINO UNO
B. L298N DRIVER
C. 2 MOTORES DC CON REGULADOR
D. HC-SR04
E. 2 INFRARROJOS
F. SERVOMOTOR SG90
G. RUEDA LOCA/PIBOTE
H. PILA 9V
I. CONECTOR DE 12V CON BORNERA
J. LLAVE DE PASO

Con la construcción de este robot realizaremos un cierre practico a lo visto en el manual hasta ahora, como mencionamos anteriormente, al no utilizar todo lo visto en el manual no podremos poner en práctica todos los dispositivos que se mostraron, pero de igual manera este robot permite practicar y aprovechar una gran cantidad de dispositivos.
La función que cumplirá este robot es la de moverse de manera autónoma por el entorno hasta el punto en el que detecte un objeto con el sensor ultrasónico o infrarrojos, es en este momento donde se realizara un cambio en la trayectoria del mismo para evitar la colisión. Es en este proceso donde entra en juego gran parte de la programación.
La estructura del robot consta de piezas diseñadas e impresas en con impresoras 3D, todos los códigos y archivos de piezas podrán ser encontrados en las bibliografías correspondientes.

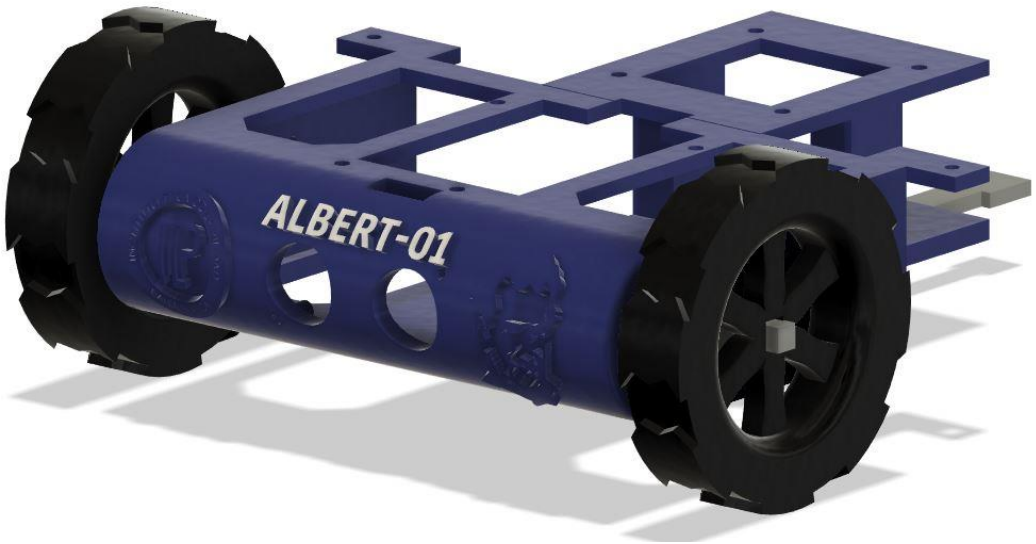
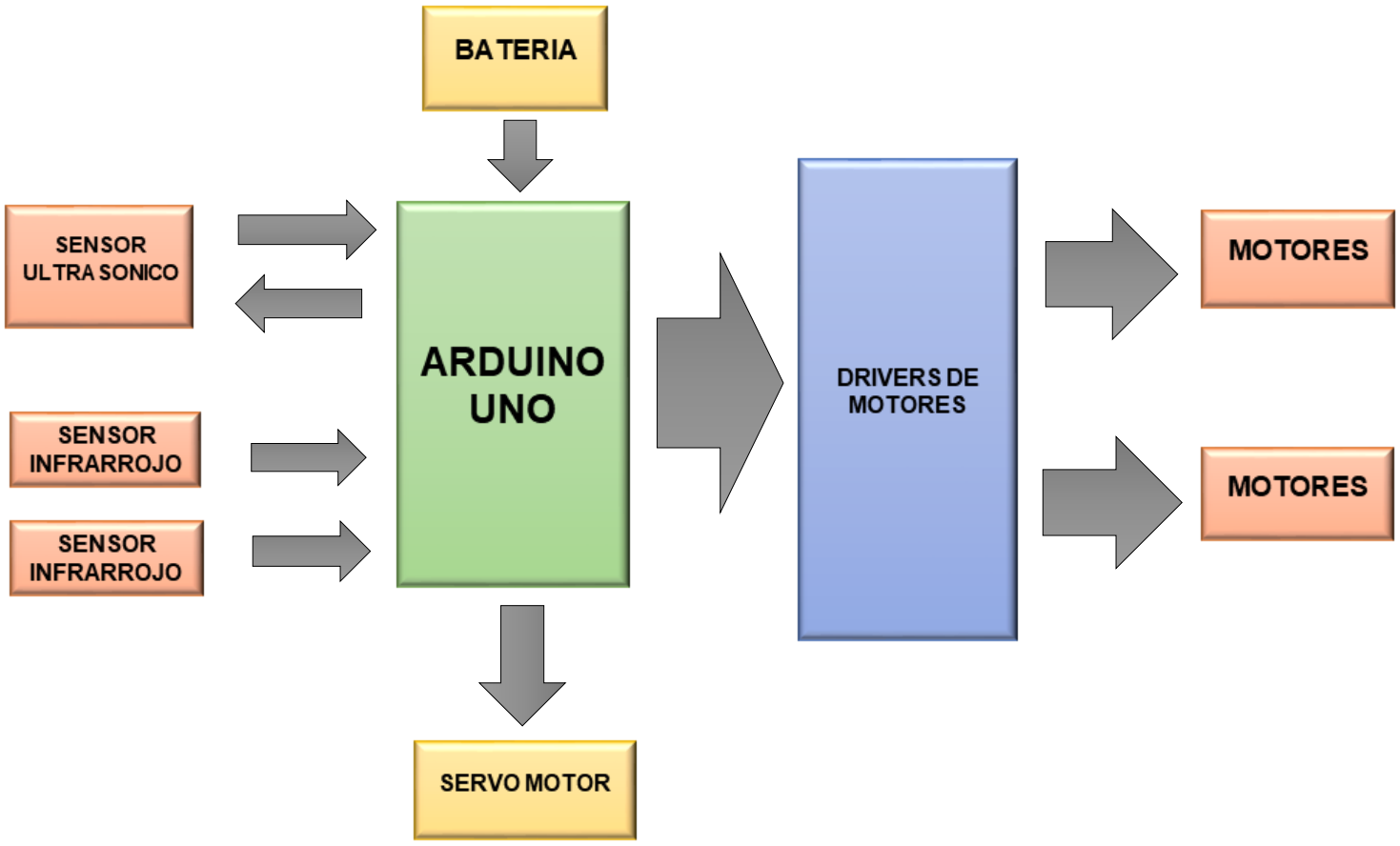


DIAGRAMA DE FUNCIONAMIENTO

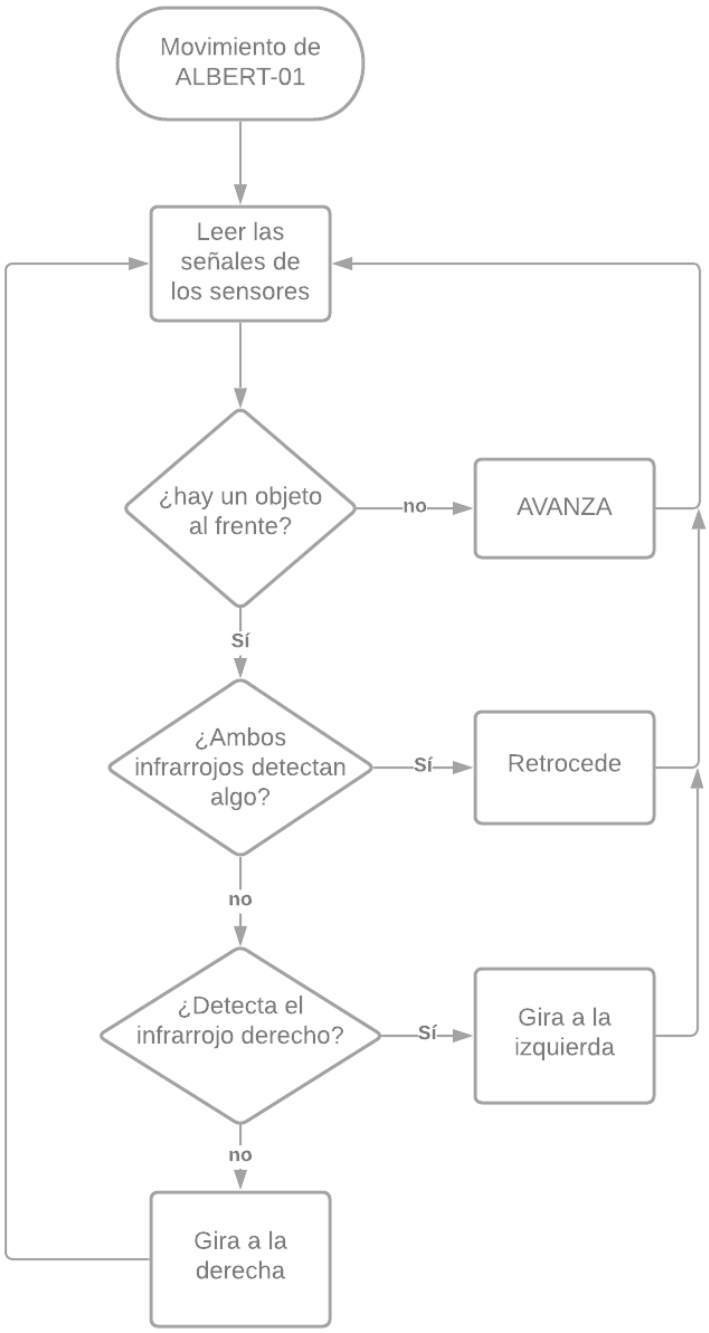
A continuación, presentamos un diagrama que representa el funcionamiento lógico interno del robot de la manera mas simplificada posible, esto con el fin de dar a entender como viajan los datos en el sistema.



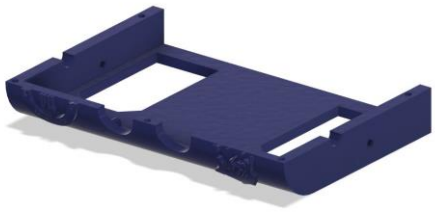
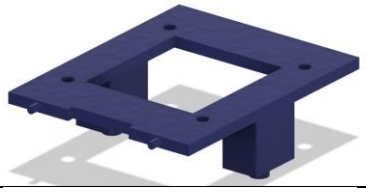
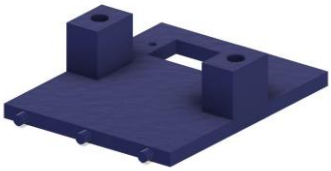
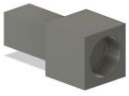
Como se puede ver el centro de control lógico es el Arduino ya que es el cerebro de todas las operaciones, todos los datos pasan a través de él, y a la vez es el que dirige a los demás componentes para que actúen. Posteriormente mostraremos el diagrama de las conexiones reales.

DIAGRAMA DE FLUJO

A continuacion le mostraremos el diagrama de flujo del comportamiento que seguira el robot, este sirve para entender mucho mejor las acciones que tomara este.



PIEZAS

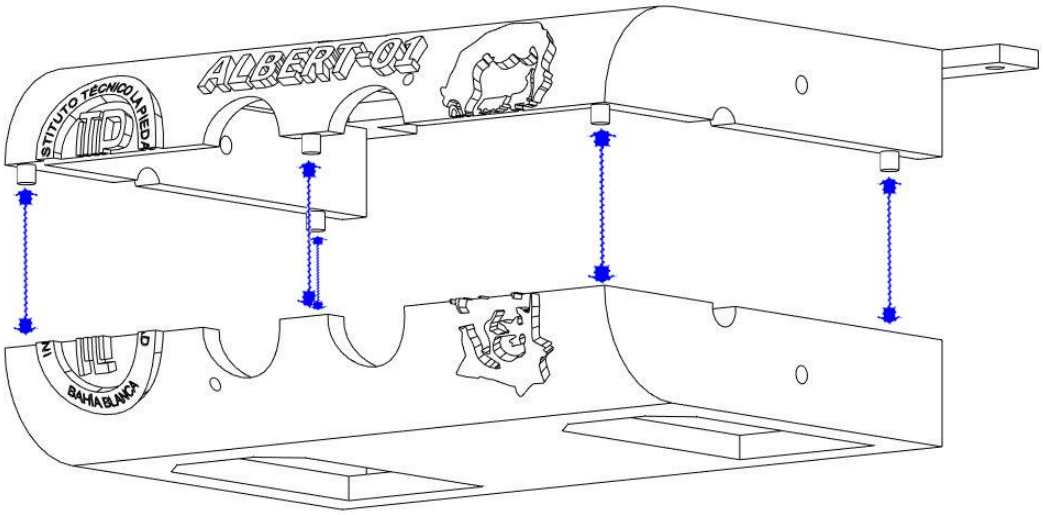
NOMBRE	IMAGEN
BASE SUPERIOR	
BASE INFERIOR	
COLA SUPERIOR	
COLA INFERIOR	
RUEDAS	
UNIONES RUEDAS	
BRAZO SERVO	

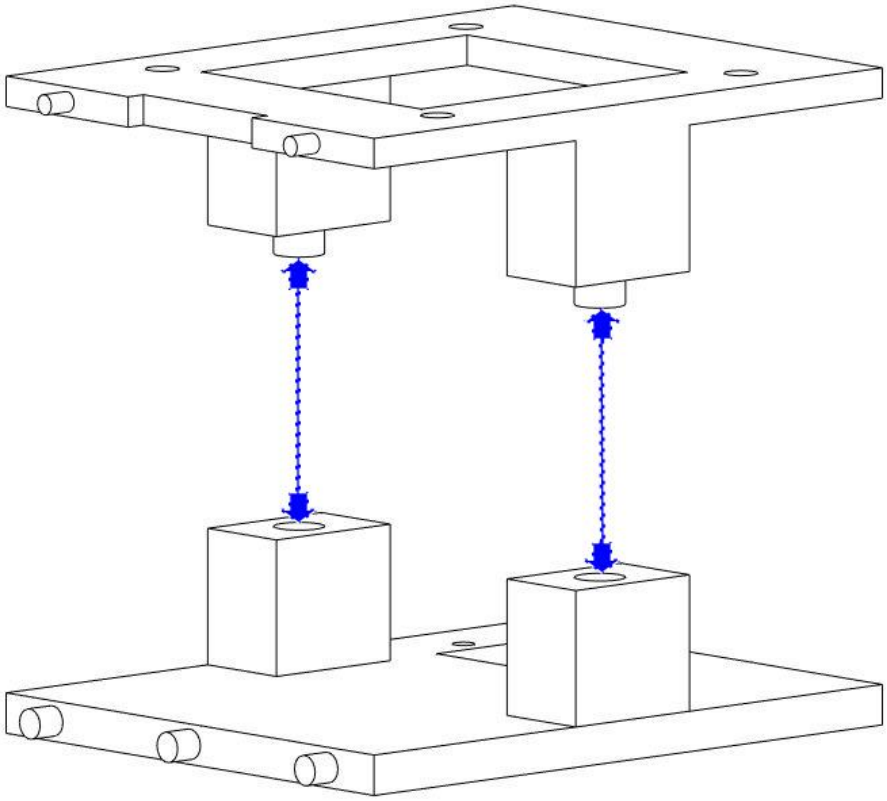
DISPOSITIVOS

NOMBRE	IMAGEN	NOMBRE	IMAGEN
ARDUINO UNO		SENSORES INFRARROJOS	
L298N		RUEDA PIBOTE	
HC-SR04		PILA DE 9V	
SG90		BORNERA DE 12V	
MOTORES DC		LLAVE DE PASO	
PROTOBOARD MINI			

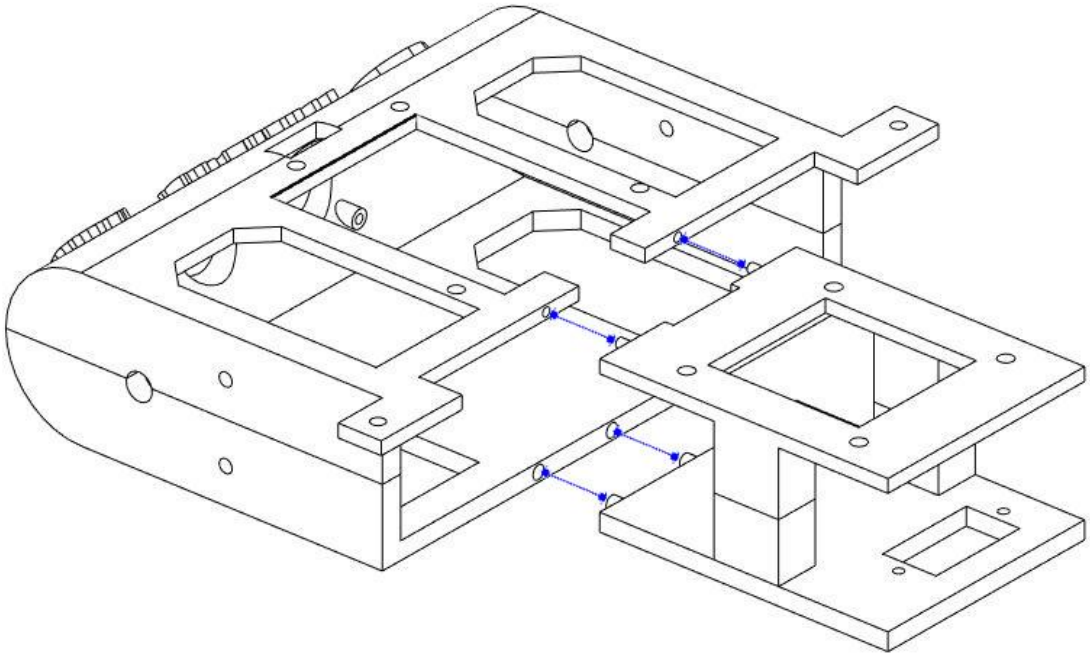
ARMADO

PASO 1-INFRAESTRUCTURA

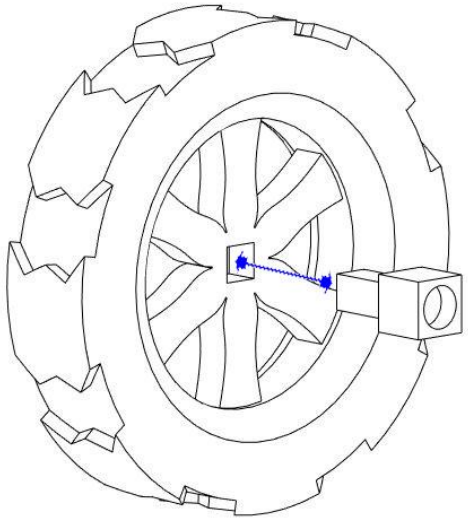




PASO 2-INFRAESTRUCTURA



PASO 3-INFRAESTRUCTURA



PASO 4-INFRAESTRUCTURA

FOTO ARDUINO, DRIVER E INFRARROJOS

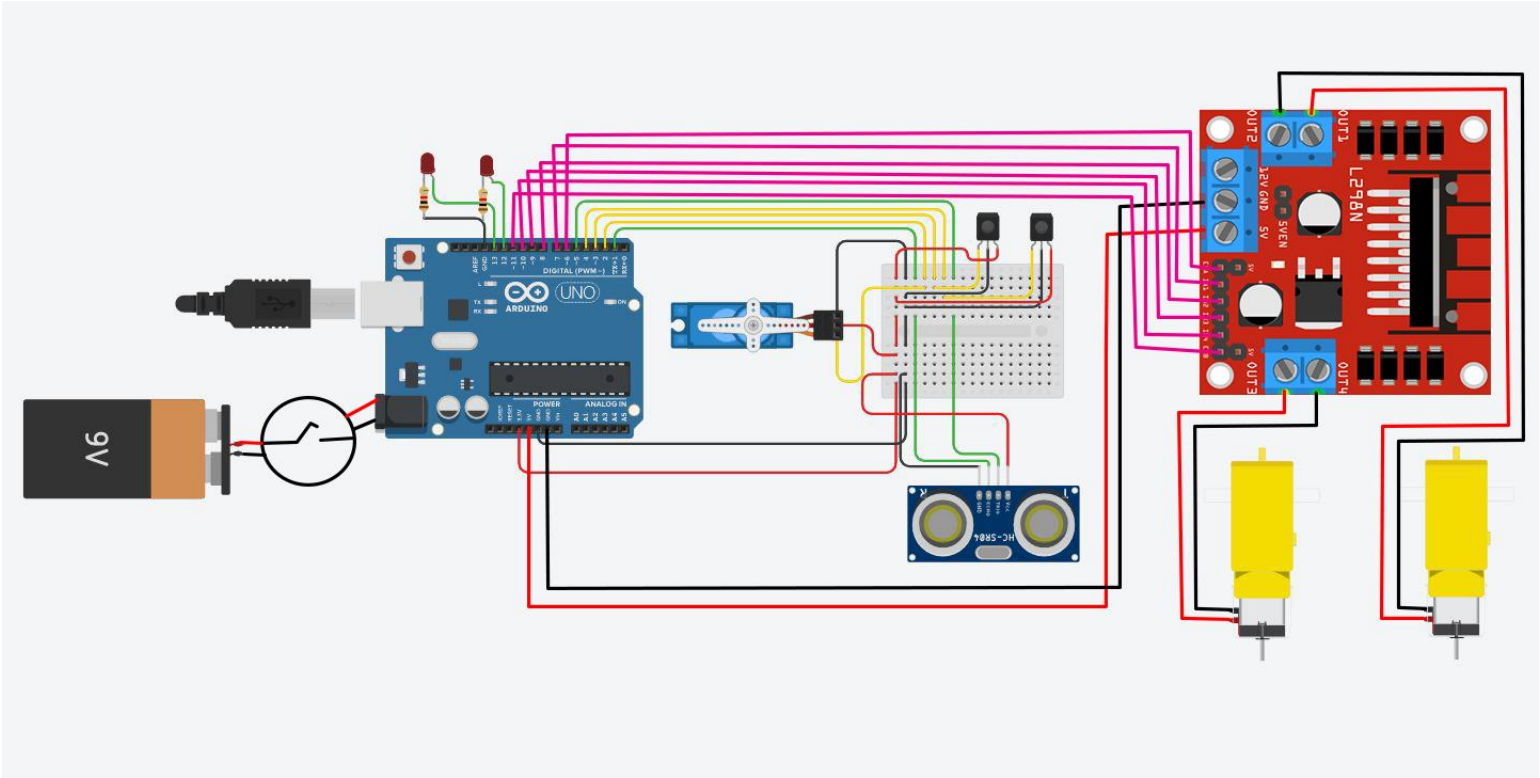
PASO 5-INFRAESTRUCTURA

FOTO MOTORES Y HC(ULTRASONICO)

PASO 6-INFRAESTRUCTURA

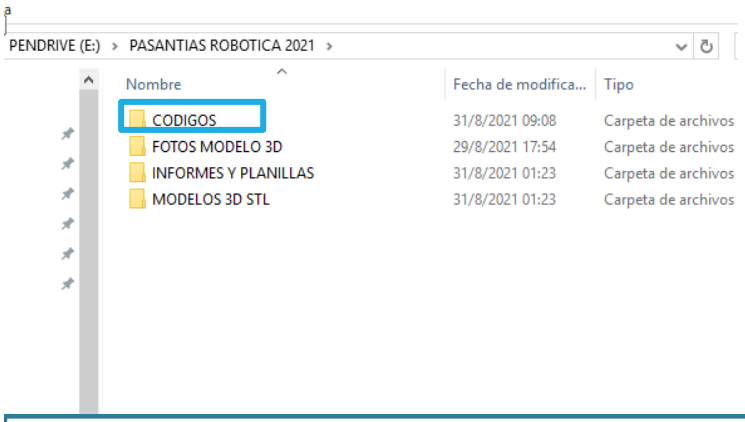
FOTO SERVOMOTOR

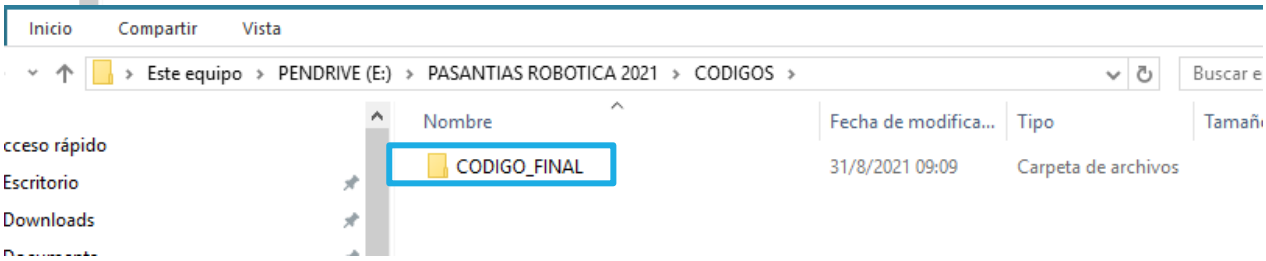
PASO 1-CONEXIONES

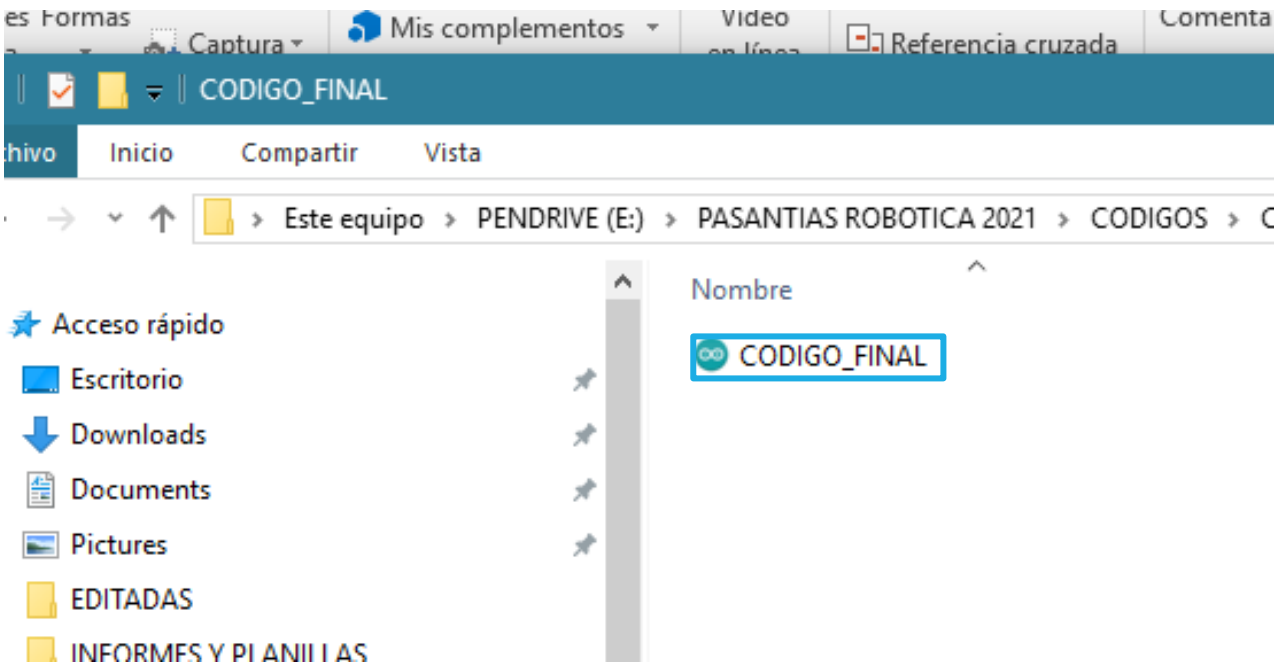


PASO FINAL-CODIGO

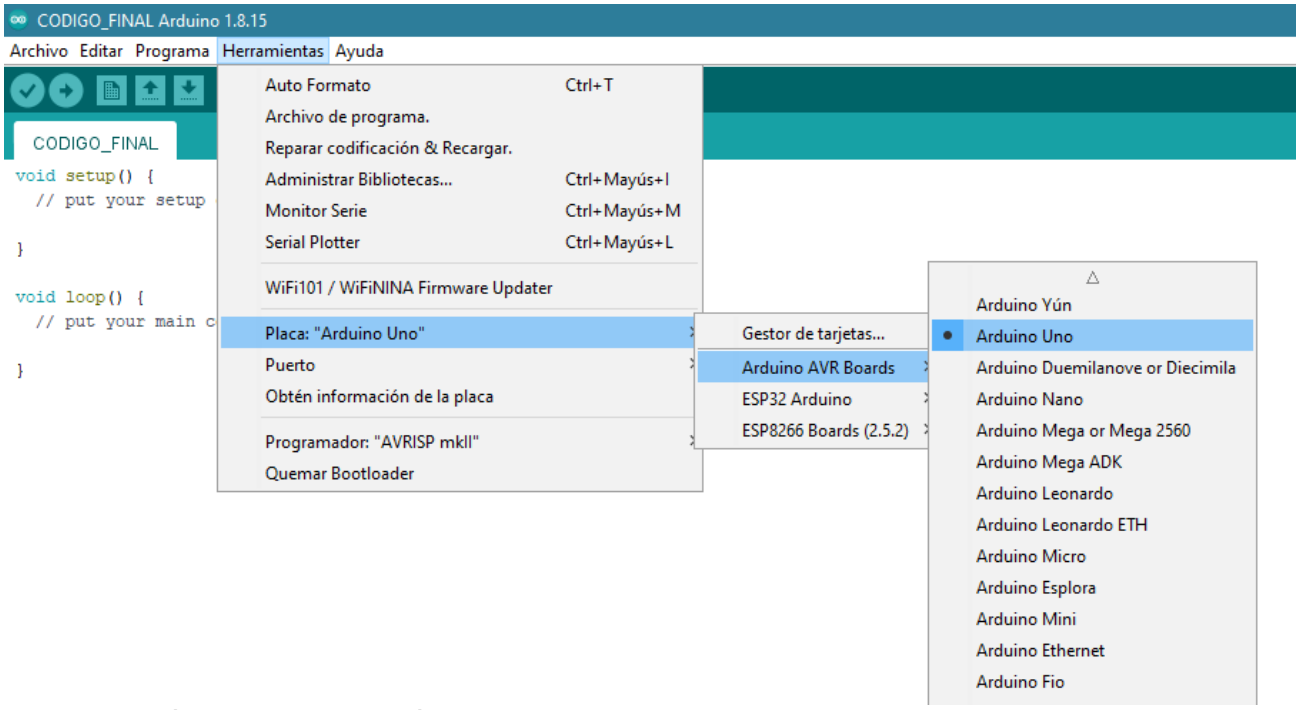
Para finalizar tenemos que cargar el siguiente código al Arduino, lo conectamos por USB al pc, entramos a la carpeta de “CODIGOS” y abrimos el archivo adjuntado al manual llamado “CODIGO FINAL.ino”.







Una vez abierto, seleccionamos el apartado de “herramientas”, abrimos “placas”, “Arduino AVR Boards” y seleccionamos “Arduino Uno”



A continuación, le dejamos el código:

```
//LIBRERIAS-------------------------------------------------------------------------------------
#include <Servo.h>
Servo myservo;
const int InfrarrojoIzquierdo = 3; //lector del infrarrojo izquierdo
const int InfrarrojoDerecho = 4;   //Lector del infrarrojo
const int pinENA = 6;              //
const int pinIN1 = 7;              //
const int pinIN2 = 8;              //Pines del motor de la izquierda
const int pinIN3 = 9;              //
const int pinIN4 = 10;             //
const int pinENB = 11;             //Pines del motor de la derecha
const int waitTime = 2000;         //espera entre fases
const int speed = 200;             //velocidad de giro
const int MotorDerecha[3] = { pinENA, pinIN1, pinIN2 };    //Se crea un arreglo que contiene todas las variables de el motor derecho para que sea mas facil llamar al motor en las funciones
const int MotorIzquierdo[3] = { pinENB, pinIN3, pinIN4 };  //Lo mismo pero con el motor izquierdo
const int EchoPin = 1;             //
const int TriggerPin = 5;          //Pines de el sensor ultrasonico
//SETUP-----------------------------------------------------------------------------------------
void setup()
{
  pinMode(pinIN1, OUTPUT);
  pinMode(pinIN2, OUTPUT);
  pinMode(pinENA, OUTPUT);
  pinMode(pinIN3, OUTPUT);
  pinMode(pinIN4, OUTPUT);
  pinMode(pinENB, OUTPUT);
  myservo.attach(2);
  pinMode(InfrarrojoDerecho , INPUT);
  pinMode(InfrarrojoIzquierdo , INPUT);
  pinMode(TriggerPin, OUTPUT);
  pinMode(EchoPin, INPUT);
}
//LOOP------------------------------------------------------------------------------------------
void loop()
{
 int cm = ping(TriggerPin, EchoPin); //se crea una variable en la que se le asigna el valor que lee el sensor calculado por la funcion ping
 int value = 0;                    //Se crea la vaiable value
 value = digitalRead(InfrarrojoDerecho );//se le asigna a value el valor del infrarrojo derecho
 int value1 = 0;                   //Se crea la vaiable value1
 value1 = digitalRead(InfrarrojoIzquierdo );//se le asigna a value1 el valor del infrarrojo izquierda
```

```
  if(cm>20){ //se crea una vaiable condicional en la que segun los sensores
      MoverParaAdelante(MotorDerecha, MotorIzquierdo, 180);
      myservo.write(90);
  }else if(value1==LOW || value ==LOW){
       MoverParaAtras(MotorDerecha, MotorIzquierdo, 180);
       myservo.write(90);
  }else if(value1==HIGH){
    girarDerecha(MotorDerecha, MotorIzquierdo, 180);
  }else{
    girarIzquierda(MotorDerecha, MotorIzquierdo, 180);
  }
}
//FUNCIONES-----------------------------------------------------------------------------------------
void MoverParaAdelante(const int pinMotor[3], const int pinmotor[3], int speed)
{
  myservo.write(0);
   digitalWrite(pinMotor[1], HIGH);
   digitalWrite(pinMotor[2], LOW);
   analogWrite(pinMotor[0], speed);
   digitalWrite(pinmotor[1], HIGH);
   digitalWrite(pinmotor[2], LOW);
   analogWrite(pinmotor[0], speed);
}
void MoverParaAtras(const int pinMotor[3],const int pinmotor[3], int speed)
{
  myservo.write(0);
   digitalWrite(pinMotor[1], LOW);
   digitalWrite(pinMotor[2], HIGH);
   analogWrite(pinMotor[0], speed);
   digitalWrite(pinmotor[1], LOW);
   digitalWrite(pinmotor[2], HIGH);
   analogWrite(pinmotor[0], speed);
}
void girarDerecha(const int pinMotor[3],const int pinmotor[3], int speed){
   digitalWrite(pinMotor[1], LOW);
   digitalWrite(pinMotor[2], HIGH);
   analogWrite(pinMotor[0], speed);
   digitalWrite(pinmotor[1], LOW);
   digitalWrite(pinmotor[2], HIGH);
   analogWrite(pinmotor[0], speed);
   myservo.write(135);
}
void girarIzquierda(const int pinMotor[3],const int pinmotor[3], int speed){
   digitalWrite(pinMotor[1], HIGH);
   digitalWrite(pinMotor[2], LOW);
   analogWrite(pinMotor[0], speed);
   digitalWrite(pinmotor[1], HIGH);
   digitalWrite(pinmotor[2], LOW);
   analogWrite(pinmotor[0], speed);
   myservo.write(45);
}
int ping(int TriggerPin, int EchoPin) {
   long duration, distanceCm;

   digitalWrite(TriggerPin, LOW);  //para generar un pulso limpio ponemos a LOW 4us
   delayMicroseconds(4);
   digitalWrite(TriggerPin, HIGH);  //generamos Trigger (disparo) de 10us
   delayMicroseconds(10);
   digitalWrite(TriggerPin, LOW);

   duration = pulseIn(EchoPin, HIGH);  //medimos el tiempo entre pulsos, en microsegundos

   distanceCm = duration * 10 / 292/ 2;   //convertimos a distancia, en cm
   return distanceCm;
}
```

REFERENCIAS-DATASHEETS COMPONENTES

DATASHEETS:

Las siguientes fichas técnicas son para obtener datos más precisos y técnicos sobre los componentes presentados en el manual.

Arduino UNO-DATASHEET
https://html.alldatasheet.es/html-pdf/791637/ETC2/ARDUINO101/99/1/ARDUINO101.html
LED-DATASHEET
https://html.alldatasheet.es/html-pdf/344183/CITIZEN/LED/290/1/LED.html
SERVO MOTOR-DATASHEET
http://www.datasheet.es/PDF/791970/SG90-pdf.html
A4988-DATASHEET
https://html.alldatasheet.es/html-pdf/338780/ALLEGRO/A4988/292/1/A4988.html
DRV8825-DATASHEET
https://html.alldatasheet.es/html-pdf/432263/TI/DRV8825/24/1/DRV8825.html
L298N-DATASHEET
https://html.alldatasheet.es/html-pdf/22440/STMICROELECTRONICS/L298N/1619/1/L298N.html
HC-SR04-DATASHEET
https://html.alldatasheet.es/html-pdf/1132203/ETC2/HC-SR04/111/1/HC-SR04.html
MOTOR NEMA17-DATASHEET
https://datasheetspdf.com/pdf/1260602/Schneider/NEMA17/1

REFERENCIAS:

Arduino: https://www.arduino.cc
Bases de programacion:
Potenciómetro: https://techmake.com
Botón o pulsador: https://www.geekfactory.mx
Servomotor: https://www.luisllamas.es
Motores paso a paso: https://www.luisllamas.es
Drivers: https://www.luisllamas.es
Motores DC: https://www.luisllamas.es
HC-SR04: https://www.luisllamas.es
Sensor infrarrojo: https://www.luisllamas.es
Tracker: https://www.luisllamas.es
Pilas o baterías: https://www.youtube.com/channel/UCIcEjQhXL8Z2TuLF_q0BG6Q
https://www.geekfactory.mx