

Tutorial para principiantes:

¿Cómo aprender a programar un Dreamster v2.0?



Por Gisela Farace

Mayo 2016

Revisión 2016-05-30



Índice

1. ¿Qué es un Arduino?

2. El robot Dreamster

2.1. Estructura

2.2. Arduino

2.2.1. Señales analógicas y digitales

2.3. Conexionado

3. ¿Qué necesitamos instalar?

4. Antes de empezar

4.1. Estructura del código

4.2 Referencias importantes

5. Aprendiendo a programar

5.1. LED

5.1.1. ¿Qué es un LED?

5.1.2. ¿Cómo prender un LED?

5.1.3. Ciclo de trabajo (Duty Cycle)

5.2. El Puerto Serie

5.3. Sensores de ultrasonido

5.3.1. ¿Dónde se encuentran los sensores de ultrasonido en el robot?

5.3.2. ¿Cómo funcionan los sensores de ultrasonido?

5.3.3. ¿Cómo medir distancias con sensores de ultrasonido?

5.4. Sensores infrarrojos

5.4.1. ¿Cómo funcionan los sensores infrarrojos?

5.4.2. ¿Cómo distinguir blanco y negro con sensores infrarrojos?

5.5. Motores

5.5.1. Servomotores de rotación continua

5.5.2. Señal de control

5.5.3. Sentido y velocidad

5.5.4. Sentido de movimiento del robot

5.5.5. ¿Cómo se controlan los motores?

5.6. ¿Cómo crear funciones?

6. Desafíos

6.1. Seguidor de líneas

6.2. Resolvedor de laberintos

6.3. Empujando cajas

7. Anexo

- 7.1. pinMode()
- 7.2. digitalWrite()
- 7.3. analogWrite()
- 7.4. analogRead()
- 7.5. delay()
- 7.6. delayMicroseconds()
- 7.7. Serial.begin()
- 7.8. Serial.print(), Serial.println()
- 7.9. pulseIn()
- 7.10. if...else
- 7.11. for
- 7.12. while
- 7.13. <Servo.h>
 - 7.13.1. attach()
 - 7.13.2. write()

8. Páginas de referencia

1. ¿Qué es un Arduino?

Arduino es una plataforma de hardware libre basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares. Con Arduino podemos controlar luces, motores, interruptores, así como también transmitir datos a otros dispositivos. Además podemos obtener información del entorno a través de sus entradas analógicas y digitales utilizando diferentes sensores (temperatura, humedad, distancia, luz, etc.).



El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación C++ (utilizando comúnmente la biblioteca Arduino que simplifica la programación) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden modificarse y compilarse sin necesidad de que la placa esté conectada a una computadora.

El programa se desarrolla utilizando el entorno de programación propio de Arduino y se transfiere al microcontrolador por medio de un cable USB. Mientras la placa se encuentre conectada a la computadora no es necesario utilizar una fuente de alimentación externa ya que la propia computadora proporciona la alimentación a través del cable USB.

Para conocer más sobre Arduino: www.arduino.cc

2. El robot Dreamster

Dreamster es una plataforma robótica de código abierto que incluye desde la parte estructural, como la mecánica, hasta programas orientados a implementar robótica avanzada (ROS)¹. El primer robot Dreamster fue pensado para ser extremadamente flexible en su diseño, gracias a que sus partes estructurales están diseñadas y fabricadas con impresoras 3D, lo que permite modificarlas y recrearlas de forma muy sencilla por cualquier persona.

Actualmente su hardware se compone de un *shield*² personalizado, compatible para Arduino Uno y Arduino Leonardo, que permite controlar sensores de ultrasonido (US) y de luz infrarroja (IR), así como dos motores de tensión continua (DC).

En la figura 1 podemos observar la versión actual del robot Dreamster. Toda la información sobre cómo está construido Dreamster, y los archivos fuentes para poder reproducirlo se pueden encontrar en: dreamsterbot.org

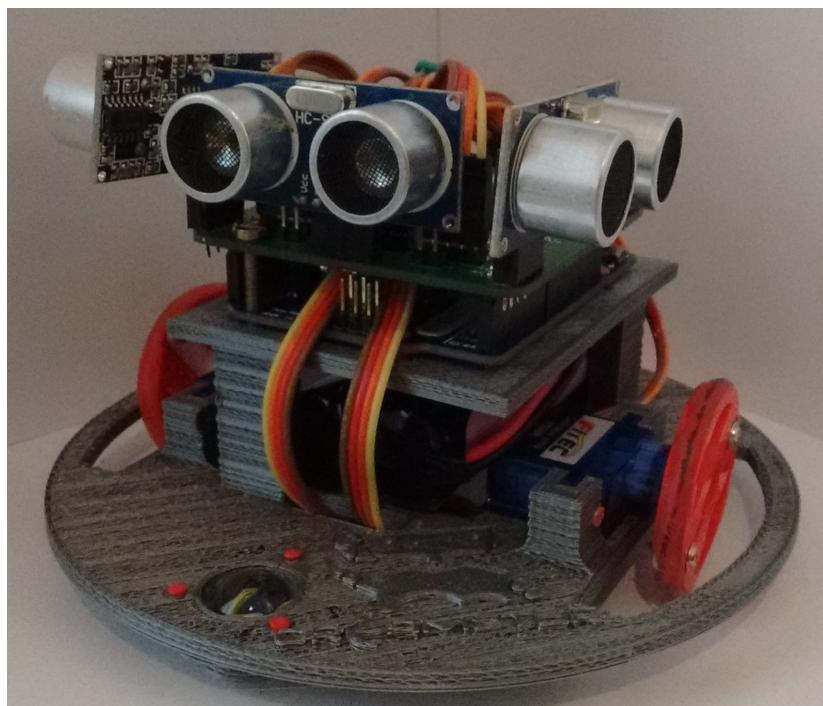


Figura 1. El robot Dreamster

2.1. Estructura

Para desarrollar la estructura del robot empleamos OpenSCAD que es un programa de diseño asistido por computadora utilizado para dibujo 2D y modelado 3D. Luego, las piezas se fabricaron utilizando impresoras 3D.

¹ el módulo de ROS (Robot Operating System) se encuentra en etapa de desarrollo.

² un shield es una placa que se conecta arriba de la placa de Arduino y permite extender su funcionalidad, por ejemplo agregando controladores de motores, leds, etc.

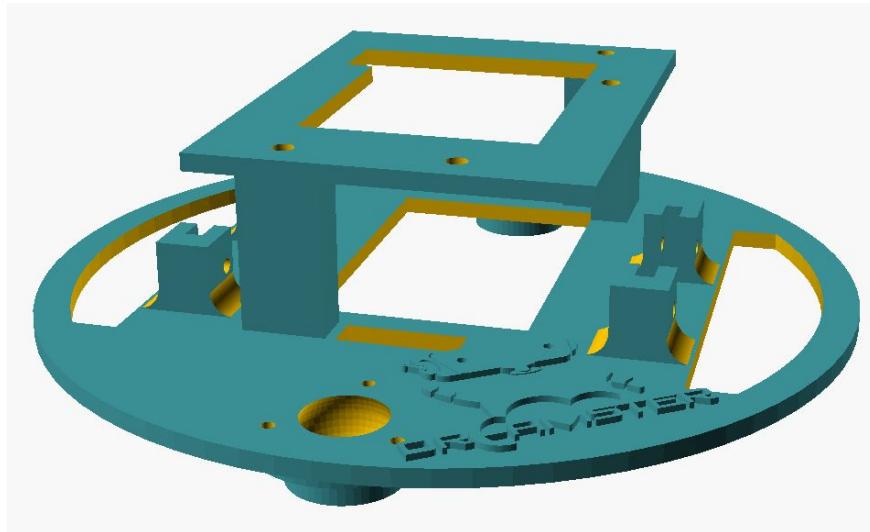


Figura 2. Modelo 3D en OpenSCAD de la estructura del robot Dreamster

2.2. Arduino

Existen muchos modelos de placas Arduino. En particular nuestro Dreamster utiliza un Arduino Leonardo, como el que se observa en la figura 3.

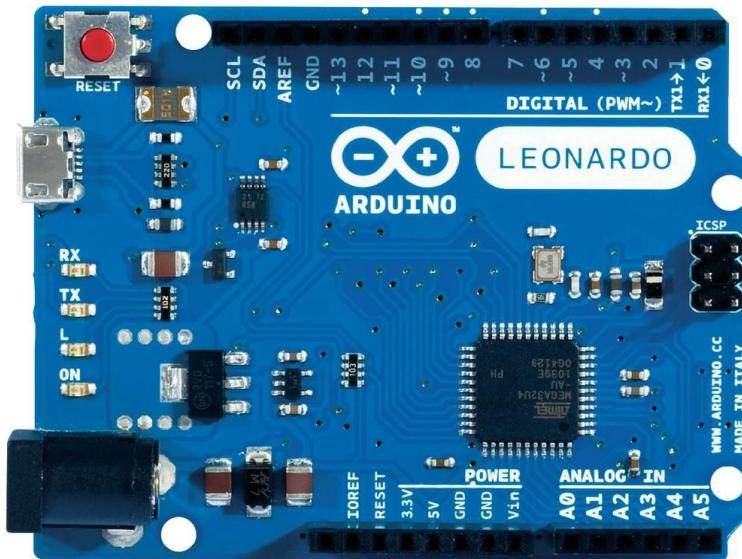


Figura 3. Vista superior del Arduino Leonardo

Llamamos pin a cada uno de los terminales hembra negros que se encuentran en el borde de la placa, que son los que nos permiten acceder a los terminales del microcontrolador. Cada uno de los pines se encuentran nomenclados y agrupados según su función. En la parte superior de la figura 3 pueden identificarse los pines “digitales” y en la parte inferior se encuentran las entradas “analógicas”.

2.2.1. Señales analógicas y digitales

Una señal eléctrica (o simplemente “señal”) es una representación de la variación de la tensión en función del tiempo. Existen dos tipos de señales: analógicas o digitales. Las últimas son de especial interés ya que son las únicas que entienden el microcontrolador o “cerebro” del robot.

Una señal analógica podría representar una variable física, como ser la temperatura o la intensidad de luz. Las magnitudes físicas, en general, son continuas, por lo tanto entre dos posibles valores de la señal pasamos necesariamente por todos los valores intermedios. Existen sensores que tienen salida analógica y se necesita de un proceso llamado *conversión analógico-digital* para que el microcontrolador las entienda. En Arduino veremos más adelante que este proceso está simplificado y se realiza a través de una función llamada `analogRead()`.

Por otro lado, una señal digital sólo puede tomar un número finito de valores. En particular, nos interesan las señales digitales que sólo puede tomar dos valores o estados: 0 y 1. Estas señales digitales binarias podrían representar botones presionados o no presionados, la salida de un fin de carrera, el estado de una barrera infrarroja. Como característica de una señal digital, notamos que la curva es discontinua.

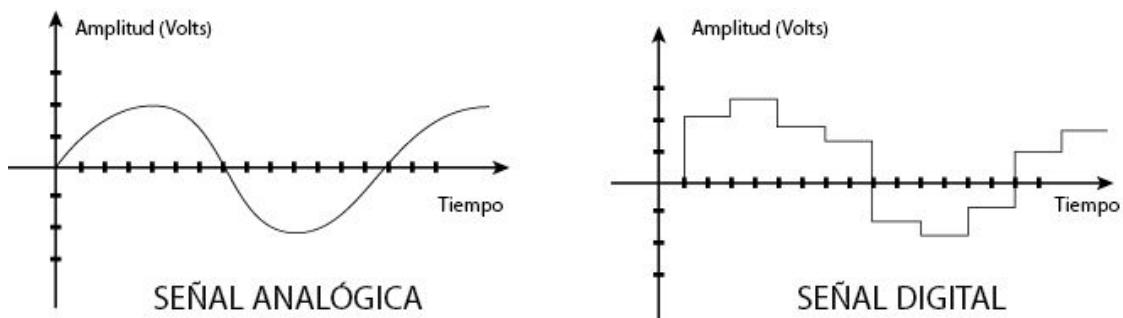


Figura 4a. Una señal analógica es una función continua respecto del tiempo y por lo tanto puede tomar cualquier valor, mientras que una señal digital es una función discontinua que solo puede tomar un número finito de valores

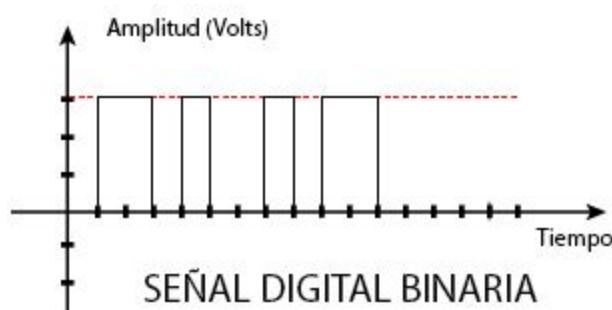


Figura 4b. Señal digital binaria, sólo puede tomar dos valores

2.3. Conexionado

En el Arduino los pines pueden ser de salida o entrada. Un pin es entrada cuando lo utilizamos para obtener información del entorno (es decir, la información entra al microcontrolador). Por otro lado, un pin es de salida cuando lo utilizamos para transmitir información o generar una acción (por ejemplo encender un LED o enviar datos a una computadora, donde la información sale del microcontrolador).

La mayoría de los pines pueden funcionar tanto como entrada o como salida digital (o ambas) a excepción de los que tienen una “A” antes del número. Esta letra indica que dichos pines son entradas analógicas. Además, los pines que contienen la marca “~”, son pines que pueden utilizarse para generar una señal de salida digital de modulación de ancho de pulso o PWM (*Pulse Width Modulation*). Esta clase de señal se usa generalmente para regular la velocidad de los motores o ajustar el brillo de un led y la estudiaremos con más detalle en la sección 5.5.2.

A continuación detallamos las conexiones de los LEDs, sensores y motores del robot en los pines del Arduino.

Función	Pin Name	Función	Pin Name
Motor Izquierdo	6 (~)	Sensor US A Trigger	A3
Motor Derecho	9 (~)	Sensor US A Echo	8
Led Verde	11 (~)	Sensor US B Trigger	A4
Led Rojo	12	Sensor US B Echo	2
Led Azul	13 (~)	Sensor US C Trigger	A5
Infrarrojo Derecho	A0	Sensor US C Echo	7
Infrarrojo Izquierdo	A1		

Tabla 1. Conexiones del Dreamster en la placa Arduino Leonardo

3. ¿Qué necesitamos instalar?

Primero debemos instalar el Software de Arduino (IDE) el cual podemos descargar gratuitamente en: <http://www.arduino.cc/en/Main/Software>

Una vez instalado, abrimos el programa. Se verá de una forma parecida a la siguiente.

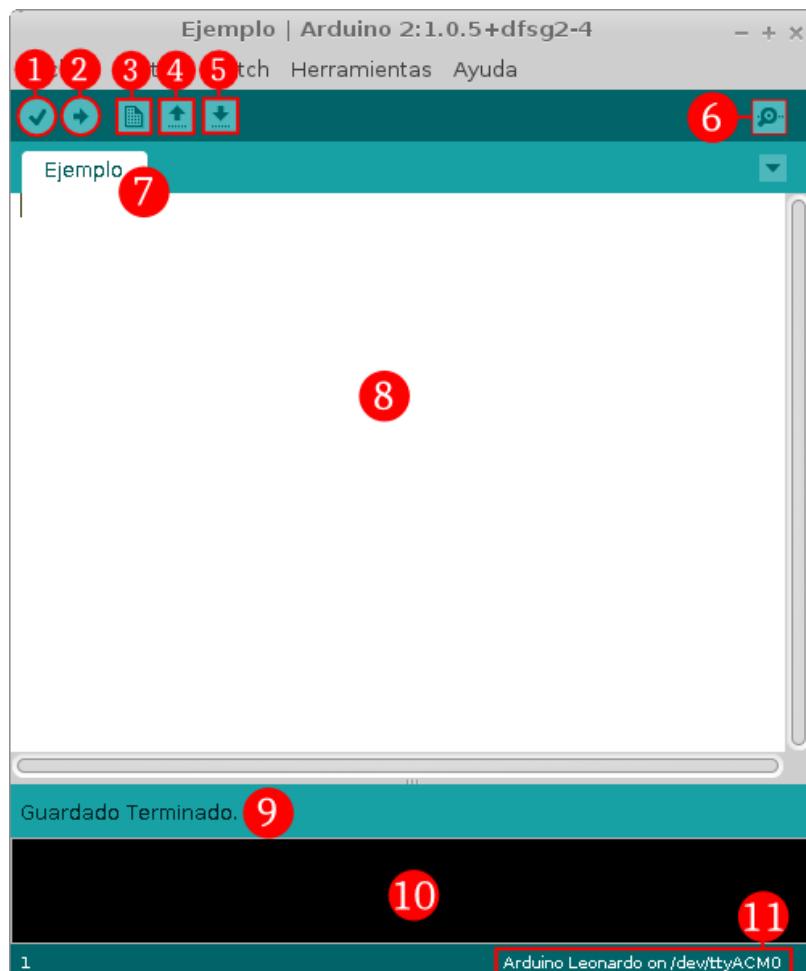


Figura 5. Estructura de la IDE

1. **Verificar:** Compila y prueba el código. Notifica si encuentra errores.
2. **Cargar:** Envía el código a la placa de Arduino. Cuando seleccionamos esta opción deberíamos observar que las luces de nuestra placa parpadean.
3. **Nuevo:** Abre una nueva ventana de código.
4. **Abrir:** Abre los diferentes códigos existentes.
5. **Guardar:** Guarda los cambios de nuestro código.
6. **Monitor Serie:** Abre una ventana que muestra la información que envía la placa a través del puerto serie. Este monitor es muy útil para depurar o encontrar errores en el código.
7. **Nombre del Sketch:** Muestra el nombre del archivo en el que estamos trabajando.
8. **Área de código:** En esta sección se escribe el código.

9. **Área de mensajes:** En esta área la IDE muestra si se encuentra algún error en nuestro código.
10. **Consola de texto:** Muestra el mensaje de error completo. Para depuración, la consola de texto es muy útil.
11. **Tarjeta y Puerto Serial:** Muestra qué modelo de Arduino está configurado y en qué puerto está conectado.

Luego debemos configurar el modelo de Arduino que se va a utilizar

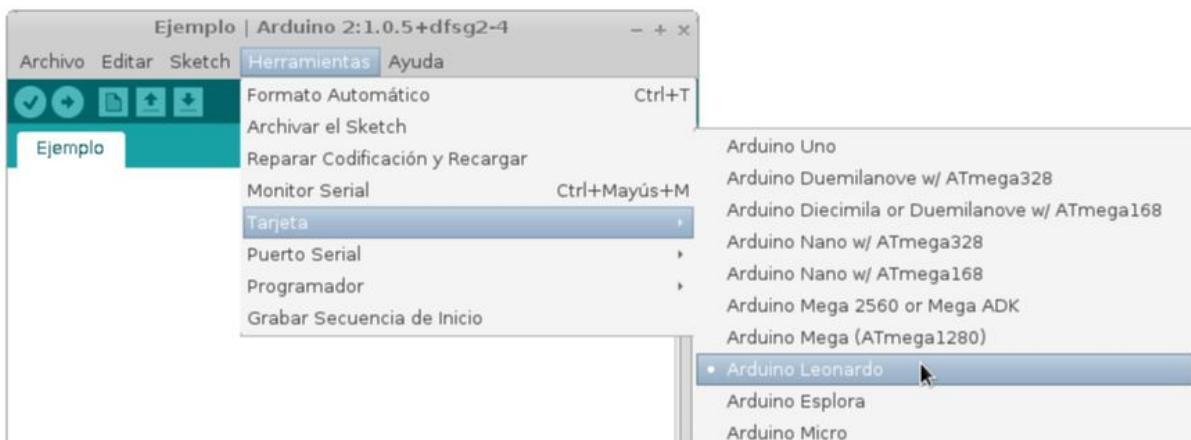


Figura 6. Configuración del Arduino Leonardo en la IDE

Por último debemos configurar en qué puerto se encuentra conectado el Arduino. Muchas veces la computadora reconoce el Arduino y el puerto en el que está conectado, indicando entre paréntesis el modelo de la placa.



Figura 7. Configuración del puerto serial en la IDE

4. Antes de empezar

4.1. Estructura del código

El código se estructura de la siguiente forma:

```
/* Parpadeo del LED:  
Se prende por un segundo y luego se apaga por un segundo.  
Se repite indefinidamente  
*/  
  
int led = 13; Declaración de variables  
  
void setup() {  
    pinMode(led, OUTPUT);  
} Configuración  
  
void loop() {  
    digitalWrite(led, HIGH);  
    delay(1000);  
    digitalWrite(led, LOW);  
    delay(1000);  
} Programa principal.  
Se ejecuta infinitamente
```

Pequeña descripción

Figura 8. Estructura del código

Las estructuras principales del programa son:

void setup()

Es la primera función que se ejecuta en el programa. Se usa para inicializar las variables, el modo de los pines, etc. Se ejecuta una sola vez al empezar el programa o luego de presionar el botón de reset de la placa.

void loop()

Esta función es precisamente un ciclo infinito. Dentro se encuentra nuestro programa principal. Debemos declararla luego de haber configurado la función **setup()**.

Declaración de variables

Antes de la función de `setup()` debemos declarar las variables globales que vamos a utilizar. Esto nos permite asociar una etiqueta a un número de pin o configurar otro tipo de variables auxiliares. También se pueden crear funciones personalizadas.

4.2 Referencias importantes

(Ver Anexo para más información)

A continuación se explican las funciones y variables que vamos a utilizar durante este tutorial. En el Anexo se explica la síntesis de las funciones.

Variables (tipos de variables)

`char`: variable de tipo entero de 8 bits, preparada para almacenar caracteres de la tabla ASCII³. (Rango -127 a 127)

`int`: variable de tipo entero de 16 bits⁴. (Rango -32767 a +32767)

`long`: variable de tipo entero de 32 bits (Rango -2147483647 a +2147483647)

Modificadores de variables

`const`: deja constante el valor de la variable. Se escribe antes del tipo de variable.

`unsigned`: indica al compilador que se busca trabajar con enteros positivos. Por ejemplo una variable `unsigned char` es de tipo entero de 8 bits pero en un rango de 0 a 255.

Estados de algunas variables

`HIGH`, `LOW`: constantes que definen el estado del pin en alto (`HIGH`) o en bajo (`LOW`).

`INPUT`, `OUTPUT`: configura el pin como entrada (`INPUT`) o como salida (`OUTPUT`)

Funciones

`pinMode()`: configura un pin específico para que se comporte como entrada o salida.

`digitalWrite()`: escribe el valor `HIGH` o `LOW` a un pin digital.

`analogWrite()`: escribe un valor analógico al pin. Puede ser usado para variar el brillo de un LED o cambiar la velocidad de un motor.

`analogRead()`: lee el valor de una entrada analógica, y convierte dicho valor a un número entero (generalmente entre 0 y 255).

`delay()`: pausa el programa por una cantidad de tiempo medida en milisegundos.

³ <https://es.wikipedia.org/wiki/ASCII>

⁴ Mientras que el tamaño de una variable `char` siempre es 8 bits, el tamaño de las variable `int` y `long` depende del microcontrolador utilizado. En algunos microcontroladores, una variable `int` es de 32 bits y una variable `long` es de 64 bits.

delayMicroseconds(): pausa el programa por una cantidad de tiempo medida en microsegundos.

Serial.begin(): configura la tasa de transferencia de bits por segundo para la transmisión por puerto serie.

Serial.print(), **Serial.println()**: imprime la información que se envía por el puerto serie. En el caso del **println** luego de imprimir comienza en una nueva línea.

pulseIn(): lee el estado del pin, ya sea **HIGH** o **LOW**.

Estructuras

if ... else if ... else: esquema de decisión a partir de una condición o varias condiciones:

```
if(condición_1) {sentencias_1}
else if(condición_2) {sentencias_2}
else {sentencias_else}.
```

switch: esquema de decisión a partir del valor de una variable.

```
switch (variable) {
    case 0: sentencias
        break;
    case 1: sentencias
        break;
    ...
    default: sentencias
        break;
}
```

El valor **default** se utiliza para juntar todos los valores no contemplados en la lista.

for: se utiliza para repetir un bloque de sentencias encerradas entre llaves.

```
for (iniciación; condición; incremento) {sentencias}
```

while: mientras la condición es verdadera se comporta como un ciclo continuo.

Cuando la expresión dentro de la condición es falsa, sale del ciclo while.

```
while (condición) {sentencias}
```

Comentarios

```
// Comenta una linea
```

```
/* Comenta todo lo contenido entre las barras,
pueden ser párrafos con varias líneas */
```

Es importante comentar el código explicando lo que se hizo en algunas líneas y una descripción del programa en general. Esto sirve a futuro cuando luego de varios días de no leer el código, no recordamos qué intentamos hacer. También es importante para hacer más legible el programa a otra persona, sobretodo cuando se trabaja en equipo.

Biblioteca <Servo.h>

Servo myservo: declara a la variable ‘myservo’ como un objeto de Servo

myservo.attach(): configura el pin donde se encuentra conectado el servomotor

myservo.write(): escribe el valor del ángulo al servo (sentido y velocidad de giro)

5. Aprendiendo a programar

5.1. LED

5.1.1. ¿Qué es un LED?

Un LED (*light-emitting diode*) es un diodo emisor de luz. Los LEDs se usan como indicadores en muchos dispositivos y también en iluminación. Se los puede encontrar con diferente tipo de luz, abarcando el espectro infrarrojo, visible y ultravioleta.



El LED posee polaridad: ánodo (positiva) y cátodo (negativa). Se puede distinguir fácilmente la polaridad observando los terminales del dispositivo. El más largo corresponde al ánodo. También en el lado del cátodo la base del led tiene un borde plano sobre el plástico que lo recubre.

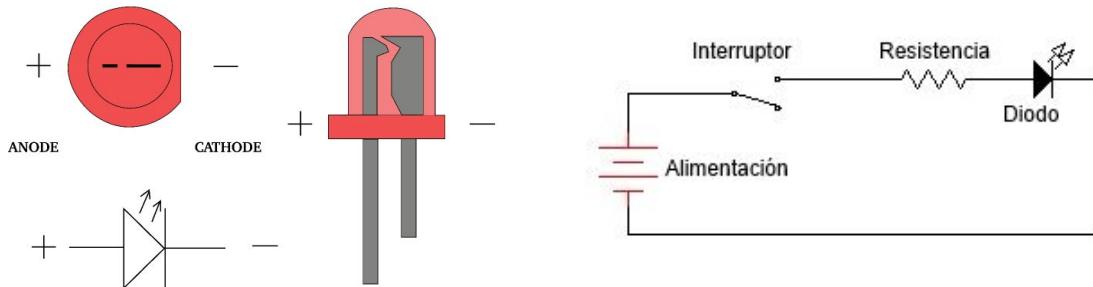


Figura 9. (Izquierda) Símbolo y esquema de un LED. (Derecha) Circuito simple de conexión de un LED, el resistor se utiliza para establecer el valor de corriente eléctrica (y por lo tanto intensidad de luz).

5.1.2. ¿Cómo prender un LED?

Mirando la tabla 1 podemos saber en qué pines se encuentran conectados los LEDs. En este caso:

- Led Verde: pin 11 (~)
- Led Rojo: pin 12
- Led Azul: pin 13 (~)

Ejemplo 1

```
// En el siguiente ejemplo el led azul parpadea con una
// frecuencia de un segundo

int led_azul = 13; // Usamos la etiqueta de led_azul para
// referirnos al pin 13

void setup() {
```

```

pinMode(led_azul, OUTPUT); // Configuramos el led como pin
de salida
}
void loop() {
    digitalWrite(led_azul, HIGH); // Se prende el led
    delay(1000); // Espera 1 segundo
    digitalWrite(led_azul, LOW); // Se apaga el led
    delay(1000); // Espera 1 segundo
}

```

Ejercicios:

1. Escribir un programa con la siguiente secuencia:
 - a. prender led azul, esperar un segundo, apagar led azul
 - b. prender led rojo, esperar un segundo, apagar led rojo
 - c. prender led verde, esperar un segundo, apagar led verde.
 - d. Repetir.
2. Cambiar el programa para prender dos leds en simultáneo, ¿qué colores se observan?
3. Para el led azul: hacer que tiempo que está prendido sea el doble que el tiempo apagado.

5.1.3. Ciclo de trabajo (Duty Cycle)

En electrónica, el ciclo de trabajo, ciclo útil o régimen de trabajo es la relación que existe entre el tiempo en que la señal se encuentra en estado alto y el periodo de la misma. El *duty cycle* se expresa como un porcentaje.

En los ejercicios anteriores, se utilizó un delay para mantener el led en un determinado estado (prendido o apagado). Al prender y apagar el led se obtiene una señal cuadrada. Cuando se prende el led por un segundo y luego se apaga por otro segundo, se obtiene un período de 2 segundos. Como el tiempo de prendido es la mitad del período, el *duty cycle* será del 50%.

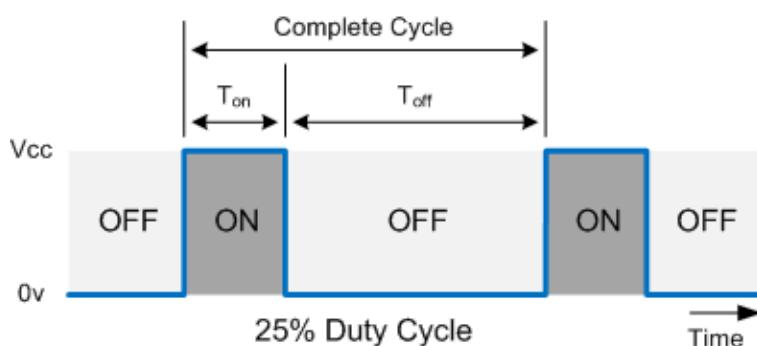


Figura 10. Ejemplo de una señal con duty cycle del 25%.

Se encuentra un 25% del período prendido y un 75% apagado.

El período, T, se define como la suma del tiempo prendido, t_{ON} , y el tiempo apagado, t_{OFF} .

$$T = t_{ON} + t_{OFF}$$
$$\text{Duty Cycle} = \frac{t_{ON}}{T}$$

Ejercicios:

4. Escribir un programa que parpadee un led con un período de 5s y duty cycle del 75%. Definir las variables periodo, duty cycle, t_{ON} y t_{OFF} .

```
long period = 5000;  
long duty_cycle = 75;  
long tON = ____;  
long tOFF = ____;
```

¿Qué ecuaciones necesitamos usar para tON y tOFF? Ayuda: Definir t_{ON} y t_{OFF} en función del periodo y duty cycle.

Operadores: Producto: * División entera: /

¿Por qué usamos `long` en lugar de `int`? Una variable de tipo `int` puede almacenar números hasta 32767. Al realizar el producto `period*duty_cycle`, estamos haciendo la cuenta $5000*75 = 375000$.

Si usáramos una variable de tipo `int` el resultado no puede almacenarse correctamente, y se produce un *overflow*, donde el número a almacenar excede el límite para ese tipo de variable, desbordando esa capacidad y perdiendo información (como cuando un vaso se desborda por exceso de líquido).

5. Cambiar el duty cycle a 25%. ¿Observamos alguna diferencia?
6. Cambiar el período a 10ms y repetir el ejercicio con duty cycle de 25% y 75%. ¿Por qué no lo vemos apagarse? ¿Por qué el led cambia su intensidad de brillo al cambiar el *duty cycle*?

Este tipo de señal, que consiste en modificar el ciclo de trabajo de una señal cuadrada de período constante, se conoce como modulación del ancho de pulso o PWM (*Pulse Width Modulation*). El valor medio de una señal de PWM es directamente proporcional al duty cycle, con lo cual una señal PWM controla la tensión media que recibe el led y por lo tanto la intensidad de brillo.

En conclusión, para poder cambiar la intensidad del brillo del led tenemos que calcular los tiempos de prendido y apagado para poder obtener el brillo que necesitamos. Por suerte, existe la función `analogWrite` que nos simplifica esta tarea.

Ejemplo 2

```
/* Usamos analogWrite para generar un PWM con un duty cycle
definido como entero (integer) entre 0 (0%, siempre apagado) y
255 (100%, siempre prendido) */

const int blue_led = 13;

int duty_cycle; // Variable auxiliar para cambiar el valor del
duty cycle

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

void loop() {
    duty_cycle = 0; // Inicializamos en 0%
    analogWrite(blue_led, duty_cycle); // Escribe el valor del
    duty_cycle en el led, en este caso es de 0%
    delay(1000); // espera un segundo
    duty_cycle = 127; // Es aprox. un duty cycle del 50%
    analogWrite(blue_led, duty_cycle); // Escribe el valor del
    duty_cycle en el led, en este caso es de 47%
    delay(1000); // espera un segundo
}
```

La función `analogWrite` genera la señal PWM, con el duty cycle elegido, y lo hará en el pin que le sea indicado. Cuando se usa `analogWrite` el periodo del PWM está fijo y no puede modificarse. En este caso es 490 Hz

Ejercicio:

7. Escribir un programa que incremente el ciclo de trabajo cada 500 ms, de 0 a 100% cada 5%, utilizando la función `analogWrite()`

Ayuda: utilizar la sentencia “for”

```
for (inicialización; condición; incremento) {
    // Sentencias si la condición es verdadera
}
```

5.2. El Puerto Serie

Como se explicó antes, el puerto serie sirve para enviar información desde el Arduino a la computadora. En este caso lo utilizaremos principalmente para depurar el código y verificar que algunos componentes estén cumpliendo su función correctamente, como por ejemplo los sensores.

Ejemplo 3

```
/* En este ejemplo, se habilita la opción de comunicación serie. Se generan números aleatorios entre 0 y 299. Luego se imprime en el monitor serie los números mayores a 100. Para inicializar el puerto serie se usa un baud rate de 9600 (default), que corresponde al número de bits por segundo (bps) máximo que puede transmitir el canal. */

int aleatorio = 0; // Declaro una variable para guardar números aleatorios

void setup() {
    Serial.begin(9600); // Configuro el baud rate en 9600
}

void loop() {
    aleatorio = random(300); // Números aleatorios entre 0 y 299
    // Si el número es mayor a 100
    if (aleatorio > 100) {
        Serial.print("Número aleatorio: ");
        Serial.println(aleatorio); // Imprime el número aleatorio
    }
    delay(100);
}
```

El ejemplo anterior en el monitor serie:

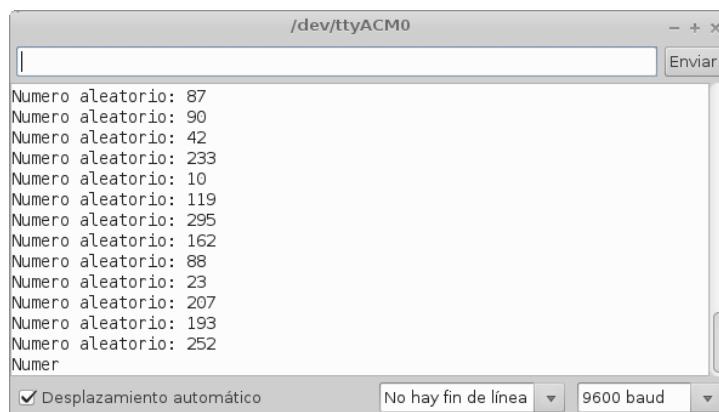


Figura 11. Muestra de números aleatorios enviados a través del puerto serie.

5.3. Sensores de ultrasonido

5.3.1. ¿Dónde se encuentran los sensores de ultrasonido en el robot?

Los sensores de ultrasonido se encuentran ubicados como se muestra en la figura 12. En el caso de los sensores B y C, hay dos posibles ubicaciones en la placa y se puede intercambiar la posición del sensor. Estas cuatro ubicaciones en particular están conectadas de forma de cruz. Por esta razón no es posible ubicar dos sensores en forma cruzada, por ejemplo ambos en las ubicaciones B. La máxima cantidad de sensores que se pueden usar al mismo tiempo es de tres.

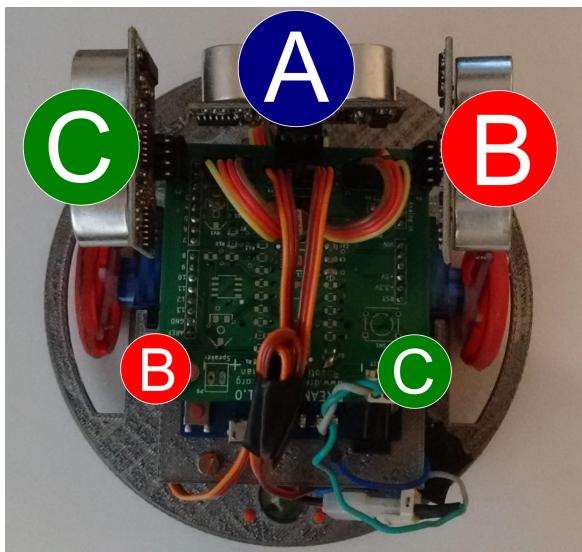


Figura 12. Ubicación de los sensores de ultrasonido en el robot

5.3.2. ¿Cómo funcionan los sensores de ultrasonido?

Estos sensores nos permiten medir la distancia a un objeto mediante el uso de pulsos ultrasónicos. A través del *Trigger* se envía un pulso que le avisa al sensor que debe empezar a medir. Luego, el sensor envía una señal ultrasónica que rebota contra el objeto y vuelve al sensor. La señal que vuelve es recibida por un "micrófono ultrasónico" y luego el sensor pone el pin *Echo* en "estado alto" por un tiempo proporcional a la distancia del objeto detectado.



Los sensores utilizados son los HC-SR04 que se muestra en la imagen.

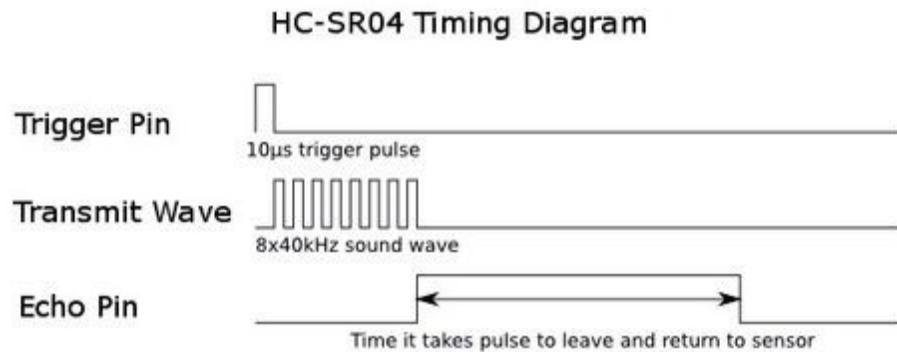


Figura 13. Diagrama de tiempos del sensor de ultrasonido.

5.3.3. ¿Cómo medir distancias con sensores de ultrasonido?

De la tabla 1, los pines que corresponden a cada sensor son:

	Sensor A	Sensor B	Sensor C
Trigger	pin A3	pin A4	pin A5
Echo	pin 8	pin 2	pin 7

Tabla 2. Ubicación de los pines para los sensores de ultrasonido.

Ejemplo 4

```

/* Ejemplo de lectura del sensor A a través del puerto serie.
Indica la distancia a la que se encuentra el objeto. El pin de
Trigger se define como salida mientras que el pin de Echo como
entrada. */

const int trigger_a = A3;
const int echo_a = 8;
long duracion = 0; // Variable auxiliar que mide la duración
// del pulso Echo
long distancia = 0; // Variable auxiliar para medir la
// distancia a partir de la duración

void setup() {
    pinMode(trigger_a, OUTPUT); // Se configura como salida
    pinMode(echo_a, INPUT); // Se configura como entrada
    Serial.begin(9600); // Configuro el baud rate en 9600
}

void loop() {
    digitalWrite(trigger_a, LOW); // Nos aseguramos que el
    trigger esté en cero
    delayMicroseconds(2);
}

```

```

digitalWrite(trigger_a, HIGH); // Activamos el trigger
delayMicroseconds(10);
digitalWrite(trigger_a, LOW); // Desactivamos el trigger
duracion = pulseIn(echo_a, HIGH); // Medimos la duración
del pulso Echo
distancia = duracion/2; // La duración medida es la de ida
y vuelta de la señal, recorre dos veces la distancia

Serial.print("Distancia = ");
Serial.println(distancia); // Imprimimos por pantalla la
distancia
delay(1000);
}

```

¿Qué distancia observamos en pantalla? ¿Está en centímetros, metros, milímetros? ¡¿Kilómetros?! Realmente no sabemos lo que nos está mostrando.
¿Cómo podemos convertir esta información en algo útil para nosotros?

Para poder observar la distancia en alguna unidad de medida conocida, vamos a realizar un simple cálculo matemático.

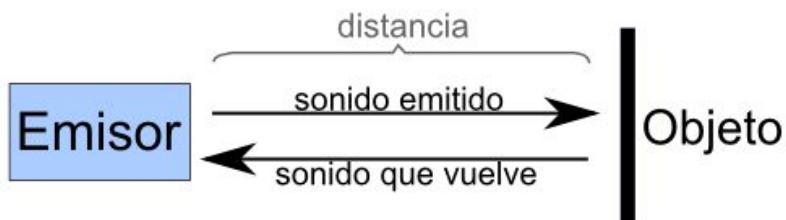


Figura 14. Recorrido del sonido emitido por el sensor.

El sensor emite un sonido que va hacia el objeto, rebota y vuelve. A partir del código anterior, se obtuvo el tiempo desde que el sonido sale del sensor hasta que vuelve al mismo.

De la figura 13 podemos realizar la siguiente ecuación:

$$2 \cdot d = v_s \cdot t$$

Donde d es la distancia entre el emisor y el objeto, v_s es la velocidad del sonido y t el tiempo medido en segundos desde que la señal sale del emisor hasta que vuelve. Observar que, como el tiempo medido es el de ida y vuelta de la señal, se recorre dos veces la distancia, por eso se la multiplica por dos.

La velocidad del sonido es un dato conocido, aproximadamente $v_s = 340 \text{ m/s}$. A partir de esta ecuación queremos calcular la distancia a la que se encuentra el objeto.

$$d = \frac{v_s \cdot t}{2} = \frac{340 \frac{\text{m}}{\text{s}} \cdot t}{2}$$

La línea “`duracion = pulseIn(echo_a, HIGH);`” nos devuelve un duración de tiempo en microsegundos. Como podemos ver, la velocidad del sonido se encuentra en metros/segundo, por eso debemos hacer una conversión de unidades para poder operar. Además, queremos que la distancia quede expresada en milímetros.

$$d = \frac{340 \cdot 10^3 \frac{\text{m}}{\text{s}} \cdot duracion \cdot 10^{-6} \text{s}}{2} = 0,17 \cdot duracion \text{ [mm]}$$

Finalmente, debemos realizar esta cuenta antes de imprimir nuestro resultado por pantalla. Agregar las siguientes líneas donde correspondan en tu código:

```
/* Dentro de la declaración de variables */
long distancia_mm = 0; // Variable auxiliar para cambiar las
unidades a mm

/* Dentro de la función void(loop) */
duracion = pulseIn(echo_a, HIGH); // Medimos la duración del
pulso Echo
distancia_mm = duracion*0.17; // Convertimos la duración en
distancia en milímetros

Serial.print("Distancia = ");
Serial.print(distancia_mm); // Imprimimos por pantalla la
distancia en milímetros
Serial.println(" mm");
delay(1000);
```

Ejercicios:

1. Armar un código donde se muestre en el monitor serie la distancia de los 3 sensores de ultrasonido en milímetros de la forma:
“Distancia A: valor A mm // Distancia B: valor B mm // Distancia C: valor C mm”
2. Usar los leds para indicar las siguientes distancias:
 - a. Azul para distancias menores a 65 mm
 - b. Verde para distancias entre 65 y 150 mm
 - c. Rojo para distancias mayores a 150 mm

Ayuda: Utilizar la sentencia “if...else”

```
if (condicional) {
    // Sentencia si la condición es verdadera
}
else{
    // Sentencia si la condición es falsa
}
```

Operadores comparadores:

igual: == diferente: != menor: < menor o igual: <=

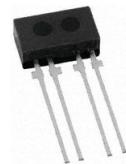
y: && o: || no: ! mayor: > mayor o igual: >=

5.4. Sensores infrarrojos

5.4.1. ¿Cómo funcionan los sensores infrarrojos?

Los sensores infrarrojos se utilizan para diferenciar “colores” en el piso. Poseen como emisor un LED infrarrojo y como receptor un fototransistor.

Este sensor es capaz de medir la cantidad de luz infrarroja reflejada en la superficie. Dependiendo del material y del color, la luz infrarroja emitida por el LED puede reflejarse totalmente, en el caso del blanco, o absorberse totalmente en el caso del negro. En particular el blanco y el negro no son considerados colores sino presencia o ausencia de luz.



Estos sensores son muy útiles para robots seguidores de línea. Los sensores utilizados en nuestro robot son los TCRT1000.

Para verificar que los sensores están prendidos podemos utilizar la cámara de un celular. Si la enfocamos hacia los sensores podemos ver que uno de los puntos del sensor se encuentra con una luz violeta. Esto quiere decir que el LED emisor infrarrojo del sensor está encendido.



Figura 15. Visión de luz infrarroja a través de una cámara de celular.
Arriba: LEDs apagados. Abajo: LEDs encendidos.

5.4.2. ¿Cómo distinguir blanco y negro con sensores infrarrojos?

Según la tabla 1, los sensores infrarrojos se encuentran conectados en:

Sensor Infrarrojo Derecha: pin A0

Sensor Infrarrojo Izquierdo: pin A1

Estos pines corresponden a la salida del fototransistor que indica la luz recibida por el sensor. Por ser una señal analógica fueron conectados a dos pines de entrada analógicos y, consecuentemente, estos pines deben ser configurados como entrada.

En el robot Dreamster los LEDS infrarrojos del sensor están conectados directamente a la tensión de alimentación y consecuentemente siempre están encendidos. Por eso no es necesario configurar el pin del LED del sensor como salida.

Ejemplo 5

```
/* En el siguiente ejemplo se utiliza el led azul como
indicador de lo que ve el sensor infrarrojo. El led va
cambiando la intensidad de la luz dependiendo de la superficie
en la que se encuentra el sensor. En una superficie "blanca" el
led está completamente prendido mientras que en una superficie
"negra" el led se apaga. Además envía por puerto serie el valor
del sensor. */

const int sensor = A0;
const int led = 13;
int medicion;

void setup() {
    pinMode(sensor, INPUT); // Configuro el sensor como entrada
    pinMode(led, OUTPUT); // Configuro el led como salida
    Serial.begin(9600); // Configuro el baud rate en 9600
}

void loop() {
    medicion = analogRead(sensor); // Lee el valor del sensor
    analogWrite(led, medicion); // Escribe el valor en el led
    Serial.println(medicion, DEC); // Imprime el valor en el
    monitor serie en forma decimal
}
```

Ejercicios:

1. Escribir un programa usando los dos sensores infrarrojos del robot, donde cambie la intensidad del led según lo que mida el sensor y se indique con led azul para uno de los sensores y con led verde para el otro sensor al mismo tiempo.
2. Escribir un programa donde se muestre por el monitor serie la distancia de los dos sensores infrarrojos de la siguiente forma:
“Derecha: valor_derecha // Izquierda: valor_izquierda”
3. Usar los leds para indicar lo siguiente para los valores de los sensores IR:
 - a. Azul si los valores derecho e izquierdo son menores a 100
 - b. Verde si los valores derecho e izquierdo son mayores a 100
 - c. Rojo si el valor del derecho es menor a 100 y del izquierdo mayor a 100, o viceversa.

5.5. Motores

5.5.1. Servomotores de rotación continua

Existen dos tipos de servomotores: de posición angular y de rotación continua. Los servomotores de posición angular están limitados en movimiento a un rango entre 0° y 180° y la señal de control se utiliza para indicar el ángulo deseado. En cambio, los servomotores de rotación continua no se encuentran limitados (giran 360°), y esto los hace más parecidos a un motor convencional. En este caso la señal de control se utiliza para indicar la dirección y velocidad de giro.

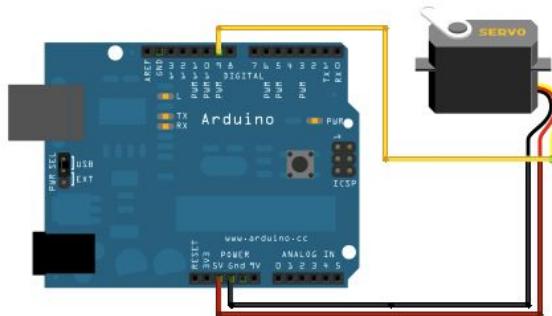


Figura 16. Ejemplo de conexionado de un servomotor en el Arduino

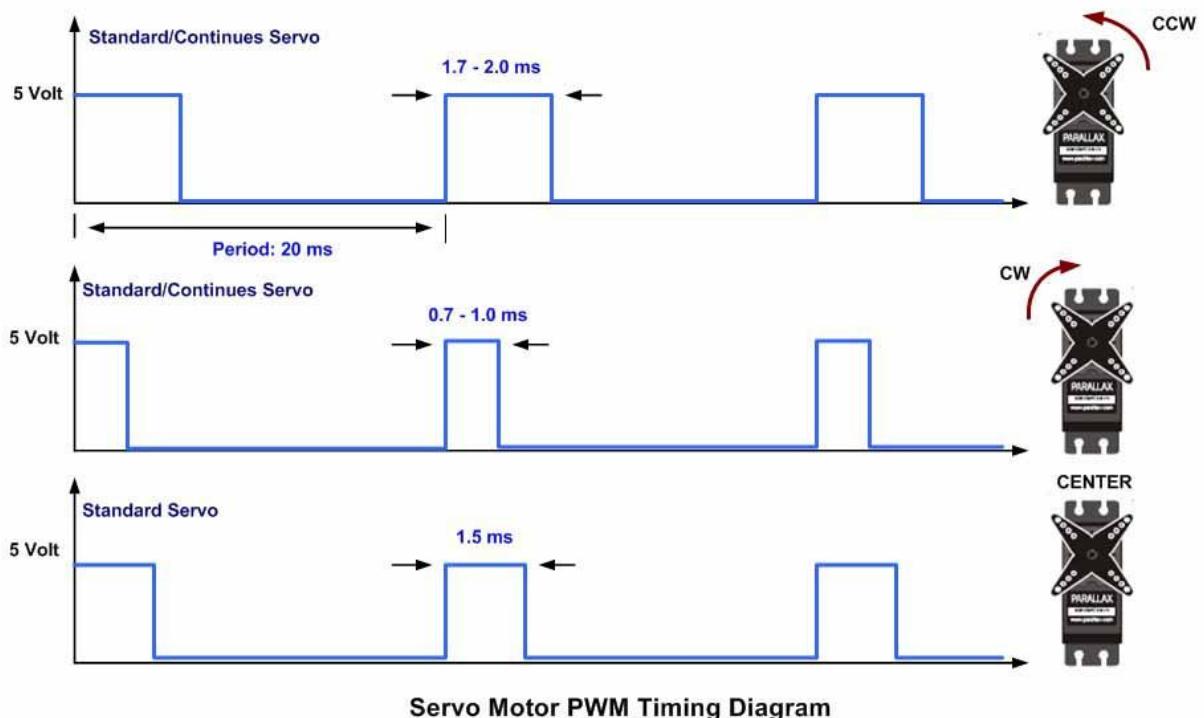
Estos motores tienen tres cables: dos de alimentación (rojo (+) y marrón (-)), y uno de señal de control (naranja) que se conecta a un pin del Arduino, como podemos observar en la figura 16.

En esta versión de Dreamster usaremos los servomotores de rotación continua modelo *FEETECH FS90R*.

5.5.2. Señal de control

Para poder regular el sentido y la velocidad con la que giran los motores necesitamos una señal de control, y en este caso se utiliza un PWM. Para comprender cómo funciona el PWM en un servo de rotación continua analizaremos el ejemplo de la figura 17⁵. La rotación y la velocidad del motor están determinadas por la duración de un pulso en alto entre un rango de 0,7 a 2 ms.

⁵ Tener en cuenta que los valores utilizados pueden variar según el tipo de motor y su fabricante. En este caso utilizaremos como ejemplo un servo de marca Parallax.



Servo Motor PWM Timing Diagram

Figura 17. (1) Rango de pulsos para que el servo gire en sentido contrario a las agujas del reloj. (2) Rango de pulsos para que el servo gire en el sentido de las agujas del reloj. (3) Rango para que el servo se mantenga quieto.

Cuando el ancho del pulso se encuentra en 1,5 ms, el motor se mantendrá quieto. A medida que el pulso aumenta desde 1,5 ms, el servo comenzará a rotar en sentido contrario a las agujas del reloj (visto de frente). Cuanto mayor sea la duración del pulso respecto de 1,5 ms, mayor será la velocidad. De forma análoga, si la duración del pulso disminuye respecto de 1,5 ms, el servo rotará en el sentido opuesto, es decir, en el sentido de las agujas del reloj (visto de frente). A medida que disminuye el ancho del pulso, la velocidad irá aumentando.

5.5.3. Sentido y velocidad

Afortunadamente existe una biblioteca de Arduino que simplifica la generación de esta señal de PWM y el duty cycle correspondiente. Esta biblioteca fue desarrollada para servomotores de posición de ángulo, por lo tanto trabajaremos con ángulos entre 0° y 180°, donde la posición de reposo será el ángulo 90°.

Recordando el ejemplo de la figura 17, sabemos que para un ancho de pulso de 1,5 ms el motor se mantiene quieto (posición de reposo), y si el ancho de pulso era menor a 1,5 ms el motor gira en un sentido, y si era mayor a 1,5 ms giraba en el otro sentido. Cuanto mayor es la diferencia entre el ancho del pulso aplicado con respecto a 1,5 ms, la velocidad aumenta.

Como dijimos, cuando usemos esta biblioteca para servomotores de posición de ángulo, trabajaremos con ángulos entre 0° y 180°, donde la posición en que nuestro motor está quieto (ancho de pulso de 1,5 ms) corresponderá al ángulo 90°. Si el ángulo es mayor a 90°,

el motor girará en sentido contrario a las agujas del reloj (visto de frente), mientras que si el es menor a 90° , girará en el otro sentido. La velocidad del motor será mayor, cuanto mayor sea el módulo de la diferencia entre el ángulo aplicado y 90° .

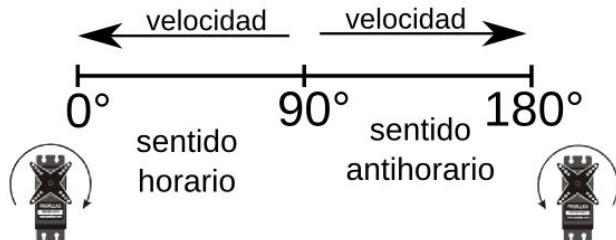


Figura 18. Ejemplo de sentido y velocidad de los motores.

5.5.4. Sentido de movimiento del robot

Este robot utiliza una tracción diferencial, donde la diferencia de velocidades entre las ruedas hace que el robot gire hacia la derecha o la izquierda.

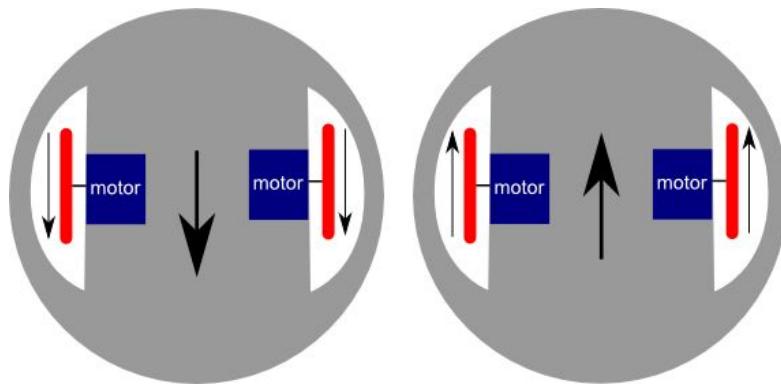


Figura 19. Si las dos ruedas se encuentran girando en el mismo sentido, el robot puede avanzar o retroceder.

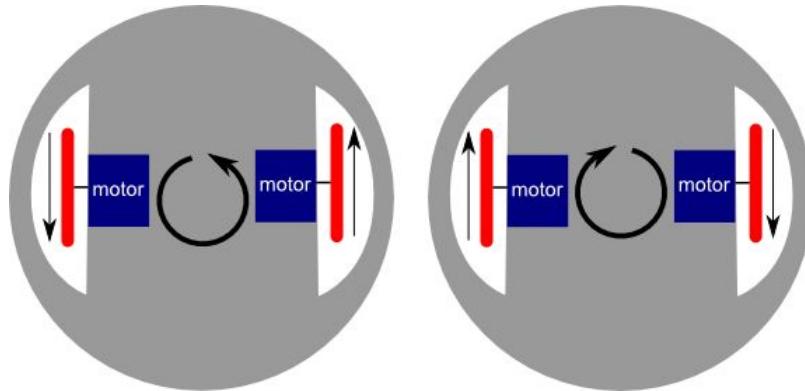


Figura 20. Si las ruedas se encuentran girando en sentidos opuestos, el robot girará en el lugar.

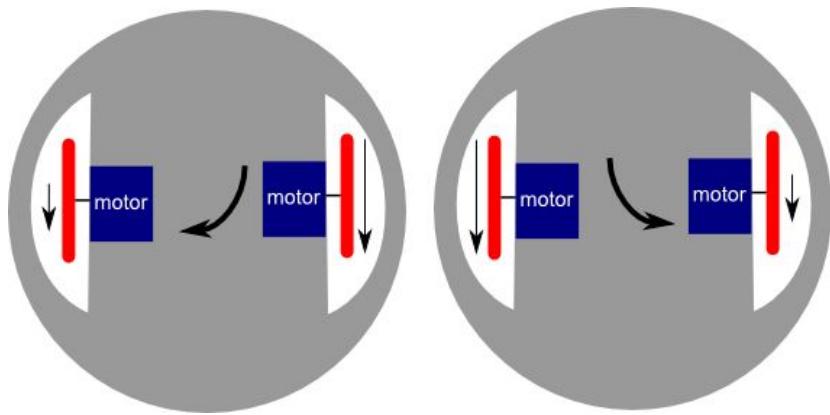


Figura 21. Si una de las ruedas se encuentra a mayor velocidad que la otra, el robot avanzará girando.

5.5.5. ¿Cómo se controlan los motores?

De la tabla 1, los pines de los motores se encuentran ubicados en los siguientes pines.

Motor Izquierda: pin 6
Motor Derecha: pin 9

Importante: según como se encuentre posicionado el motor en el robot, el sentido de rotación puede ser al revés al de la figura 18, es decir, para un rango entre 0° y 90° el motor puede girar hacia la izquierda, y para un rango entre 90° y 180°, hacia la derecha.

Con el siguiente ejemplo vamos a poder verificar cómo están ubicados los motores.

Ejemplo 6

```
/* Ejemplo del movimiento de los motores hacia adelante */

#include <Servo.h> // Incluimos la biblioteca

Servo motor_der;
Servo motor_izq; // Crea los objetos para controlar los servos

int angulo = 70; // Ángulo del servo

void setup() {
    // Vinculamos los pines de los servos a los objetos
    // declarados
    motor_der.attach(9);
    motor_izq.attach(6);
}

void loop() {
    // Le dice al servo que vaya hacia la posición de la variable
    'angulo'
```

```

    motor_der.write(angulo);
    motor_izq.write(angulo);
    delay(500);
}

```

En caso de que con la prueba anterior alguno de los dos motores va hacia atrás, solamente hay que hacer la siguiente cuenta:

```
motor_x.write(180-angulo);
```

Ejemplo 7

```

/* Ejemplo del movimiento incremental de los motores hacia adelante y luego hacia atrás. */

#include <Servo.h> // Incluimos la biblioteca

Servo motor_der;
Servo motor_izq; // Crea los objetos para controlar los servos

int pos = 0; // Variable que guarda la posición del servo

void setup() {
    // Vinculamos los pines de los servos a los objetos declarados
    motor_der.attach(9);
    motor_izq.attach(6);
}

void loop() {
    // Incrementamos la velocidad progresivamente hacia adelante
    for (pos = 0; pos <= 180; pos += 1) { // Va desde 0° a 180° con un paso de 1°
        motor_der.write(pos);
        motor_izq.write(pos); // Le dice al servo que vaya hacia la posición de la variable 'pos'
        delay(15); // Espera 15 ms para que el servo llegue a la posición
    }

    // Incrementamos la velocidad progresivamente hacia atrás
    for (pos = 180; pos >= 0; pos -= 1) { // Va desde 180° a 0° con un paso de -1°
        motor_der.write(pos);
        motor_izq.write(pos); // Le dice al servo que vaya hacia la posición de la variable 'pos'
        delay(15); // Espera 15 ms para que el servo llegue a la posición
    }
}

```

5.6. ¿Cómo crear funciones?

Muchas veces, para mayor comodidad y prolijidad a la hora de programar, se crean funciones. Estas sirven para no repetir muchas veces las mismas líneas dentro del código. Un ejemplo de esto, es la lectura de los tres sensores de ultrasonido, donde se repiten tres veces las mismas líneas pero con variables diferentes. Esto hace que el código sea más largo y con mayor probabilidad de error al copiar y pegar las líneas.

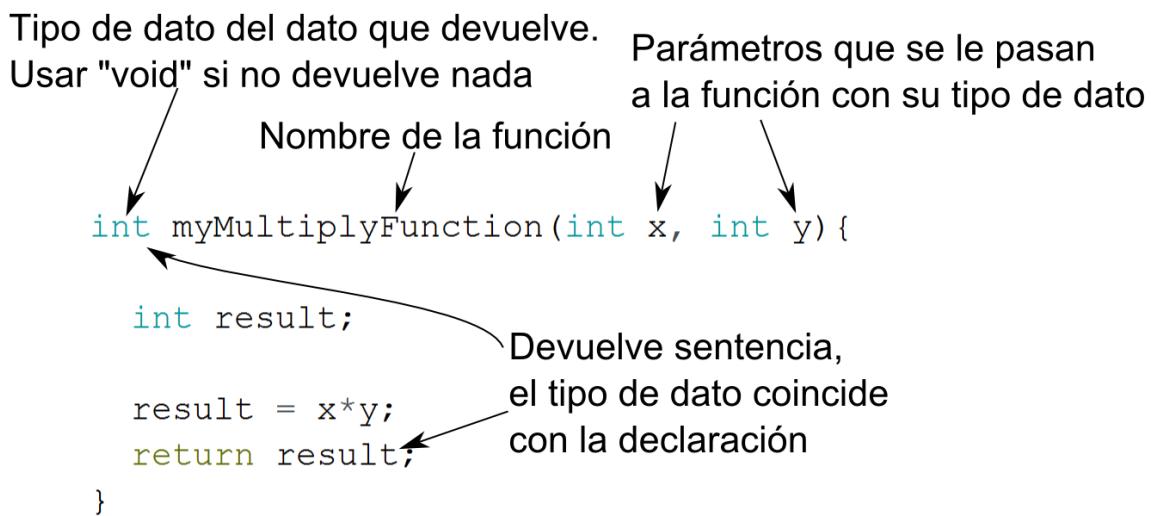


Figura 22. Estructura para crear una función

Veamos un ejemplo: quiero realizar una función que calcule la potencia 'p' de un número 'x' (x^p). Para esto, necesitamos que la función reciba el número 'x' y la potencia 'p' a la cual elevar el número.

Ejemplo 8

```
/* Ejemplo de función que eleva un número 'x' a una potencia
   'p'. En este caso, calcula las 21 primeras potencias de número
   2 (2^0, 2^1, 2^2,..., 2^20)*/

// Declaración de una función, que recibe dos parámetros
// enteros (int) y devuelve un resultado también entero.
double calculo_potencia (int x, int p){
    double resultado;
    resultado = pow(x,p); // Cálculo de la potencia
    return resultado;
}

int numero = 2; // Número al que quiero elevar
int potencia = 0; // Variable de la potencia
```

```

double result; // Variable donde se guarda el resultado

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

void loop() {
    for (potencia = 0; potencia <= 20; potencia += 1) { // La
        potencia va desde 0 a 20
        result = calculo_potencia(numero, potencia); // Llamo a la
        función que deseo usar pasándole los parámetros necesarios
        Serial.print(numero);
        Serial.print("^");
        Serial.print(potencia);
        Serial.print(" = ");
        Serial.println(result,0); // Imprime sin decimales
        delay(1000);
    }
}

```

¿Hacemos una función para los sensores de ultrasonido? Te dejo una ayuda. Ahora falta que la llames desde tu función principal void loop() enviando los parámetros correctamente.

```

/* Función que recibe parámetros de un echo y un trigger y
devuelve una distancia en milímetros */

long sensor_US (int echo, int trigger){
    long distancia;
    long duracion;
    digitalWrite(trigger, LOW); // Trigger en cero
    delayMicroseconds(2);
    digitalWrite(trigger, HIGH); // Activamos el trigger
    delayMicroseconds(10);
    digitalWrite(trigger, LOW); // Desactivamos el trigger
    duracion = pulseIn(echo, HIGH); // Duración del pulso Echo
    distancia = duracion*0.17; // Distancia en mm
    return distancia;
}

```

Desde la función principal loop(), la función se la llama de la siguiente forma para los tres sensores:

```

distancia_A = sensor_US(echo_a, trigger_a);
distancia_B = sensor_US(echo_b, trigger_b);
distancia_C = sensor_US(echo_c, trigger_c);

```

¡No te olvides de declarar las variables!

Ejercicios:

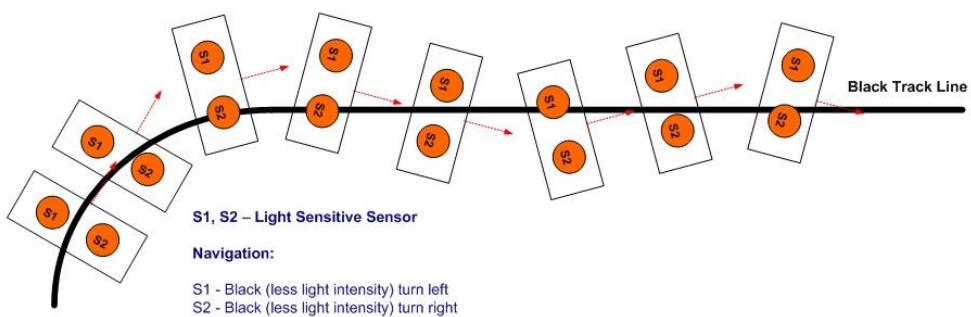
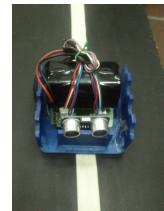
1. Crear funciones para que el robot:
 - a. Gire en el lugar 90° hacia la derecha.
 - b. Gire en el lugar 90° hacia la izquierda.
 - c. Gire en el lugar 180° .
2. Hacer que el robot describa un cuadrado, es decir, que avance derecho durante 1 segundo, luego que gire a la derecha y avance por otro segundo, gire de nuevo a la derecha hasta completar la vuelta.
3. Repetir ejercicio 2 pero que el robot gire hacia la izquierda.

6. Desafíos

¡Ahora es tiempo de integrar todo lo que aprendimos en el tutorial! Para eso te damos algunas ideas para empezar a jugar con el robot. También podés inventar tu propia modalidad. ¡A jugar!

6.1. Seguidor de líneas

¿Querés hacer que tu Dreamster sea un seguidor de líneas? Simplemente tenés que lograr programar los sensores infrarrojos y los motores para que el robot pueda seguir la línea. Te mostramos un ejemplo:



6.2. Resolvedor de laberintos

Si querés salir de un laberinto con tu Dreamster tenés que programar los sensores de ultrasonido para que detecten las paredes a su alrededor.

¿Sabías que siguiendo siempre la pared derecha o la izquierda podés salir del laberinto? Para eso, cambia de lugar los sensores de ultrasonido al lado que más te guste, programalos para que el robot siempre siga esa pared y además asegurarte que avance derecho.



6.3. Empujando cajas

Esta modalidad consiste en un recinto delimitado por una línea donde se encuentran cajas. El objetivo del robot es empujar las cajas fuera del recinto sin salirse del mismo. Para eso necesitás usar los sensores infrarrojos y los sensores de ultrasonido. ¿Te animas?



7. Anexo

Todas las funciones existentes:

<http://www.arduino.cc/en/Reference/HomePage>

7.1. **pinMode()**

Sintaxis: **pinMode(pin, modo)**

Parámetros:

pin: número de pin al que se quiere configurar

modo: INPUT o OUTPUT

Devuelve: Nada

7.2. **digitalWrite()**

Sintaxis: **digitalWrite(pin, valor)**

Parámetros:

pin: número de pin

valor: HIGH o LOW

Devuelve: Nada

7.3. **analogWrite()**

Sintaxis: **analogWrite(pin, valor)**

Parámetros:

pin: el pin al que se le quiere escribir.

valor: ciclo de trabajo, entre 0 (siempre apagado) a 255 (siempre prendido).

Devuelve: Nada

7.4. **analogRead()**

Sintaxis: **analogRead(pin)**

Parámetros:

pin: el número del pin analógico de entrada a leer.

Devuelve: int (0 a 1023)

7.5. `delay()`

Sintaxis: `delay(ms)`

Parámetros:

`ms`: el número de milisegundos a pausar. 1000 milisegundos es 1 segundo.

Devuelve: Nada

7.6. `delayMicroseconds()`

Sintaxis: `delayMicroseconds(us)`

Parámetros:

`us`: el número de microsegundos a pausar.

Devuelve: Nada

7.7. `Serial.begin()`

Sintaxis: `Serial.begin(velocidad)`

Parámetros:

`velocidad`: en bits por segundo

Devuelve: Nada

7.8. `Serial.print()`, `Serial.println()`

Sintaxis:

`Serial.println(val)`

`Serial.println(val, formato)`

Parámetros:

`val`: el valor a imprimir - cualquier tipo de dato.

`formato`: especifica la base del número, i.e. DEC para decimal.

Devuelve: `size_t (long)`: `println()` devuelve la cantidad de bytes escritos.

7.9. `pulseIn()`

Sintaxis: `pulseIn(pin, valor)`

Parámetros:

`pin`: número de pin del cual se quiere leer el pulso.

`valor`: tipo de pulso leído, HIGH o LOW.

Devuelve: la longitud del pulso en microsegundos

7.10. if...else

Estructura:

```
if (condición_1) {  
    // Acción A  
}  
else if (condición_2) {  
    // Acción B  
}  
else {  
    // Acción C  
}
```

7.11. for

Estructura:

```
for (inicialización; condición; incremento) {  
    // Sentencias  
}
```

7.12. while

Estructura:

```
while (expresión) {  
    // Sentencias;  
}
```

7.13. <Servo.h>

```
#include <Servo.h> // Incluye la biblioteca Servo  
  
Servo myservo; // Declara a 'myservo' como un objeto de Servo
```

7.13.1. attach()

Sintaxis: myservo.attach(*pin*)

Parámetros:

myservo: una variable de tipo Servo

pin: número del pin donde se encuentra el servomotor

7.13.2. write()

Sintaxis: `myservo.write(angulo)`

Parámetros:

`myservo`: una variable de tipo Servo

`angulo`: valor del ángulo para escribir en el servo, entre 0 y 180 grados.

8. Páginas de referencia

Dreamster

www.dreamsterbot.org/es

Arduino

www.arduino.cc

OpenSCAD

<http://www.openscad.org/>

Sensores de ultrasonido: HC-SR04

www.micropik.com/PDF/HCSR04.pdf

Sensores infrarrojos: TCRT1000

www.vishay.com/docs/83752/tcrt1000.pdf

Servomotores de rotación continua: FEETECH FS90R

<http://www.feetechrc.com>