

GUÍA DE ACTIVIDADES



Mulita
Robot Educativo



UNIVERSIDAD
ADVENTISTA DEL PLATA

Facultad de Ciencias Económicas y de la Administración
Carrera de Ingeniería en Sistemas
Instituto de Informática y Sistemas - INIS

Contacto



0343-4918000 interno 2782



mulita@uap.edu.ar



+54 9 343 572-3461

Autor: Octavio Javier da Silva Gillig

Diseño del manual: Mariel Mambretti

Copyright y licencias

Mulita y logo de Mulita

Marcas/*Trademarks*

Atmel® and AVR® are registered trademarks or trademarks of Atmel Corporation or its subsidiaries, in the US and/or other countries.

Arduino and Arduino UNO

Fritzing

miniBloq

Otros productos, nombres de firmas o empresas, marcas o brand names son marcas registradas o *trademarks* de sus respectivos propietarios. Cualquier omisión es no intencional.

Descargo de responsabilidad/*Disclaimer*:

La editorial y el autor hacen el mayor esfuerzo posible por garantizar la exactitud de la información presentada en este documento. Sin embargo, ni la editorial ni el autor se responsabilizan por los errores o las inexactitudes que pudieran aparecer. La información contenida en este documento está sujeta a cambios sin previo aviso. Todos los productos, marcas y nombres de firmas o empresas mencionados en este documento son propiedad exclusiva de sus respectivos propietarios. Cualquier omisión o error es absolutamente no intencional.


Este trabajo, el *software* o los elementos que eventualmente lo acompañen (sean estos de cualquier clase) son provistos por los dueños de los derechos intelectuales y por quienes contribuyeron a su desarrollo "como son", renunciando ellos a cualquier tipo de garantía explícita o implícita, incluyendo, aunque no limitándose, a las garantías implícitas de comercialización y de adaptación a propósitos particulares. Bajo ninguna circunstancia serán los dueños de los derechos intelectuales y quienes contribuyeron al desarrollo responsables por daño alguno, directo, indirecto, incidental, casual, causal (incluyendo pero no limitándose a daños a la vida y/o a la propiedad, pérdida de datos, lucro cesante, interrupción de negocios), aunque este ocurra bajo cualquier teoría de derecho, producido en cualquier forma de uso de este desarrollo o desarrollos de él derivados, aun cuando se avise o no de dicho daño o su posibilidad.



ÍNDICE

Introducción	4
Placa controladora	6
Actividad 1. Conexión de la placa con la computadora	8
Conexión y programación de un LED con un <i>protoboard</i>	15
Actividad 2. Conexión de un LED a la placa	20
Conexión y programación de un servo	25
Actividad 3. Conexión de la placa a un servo y programación	27
Conexión y programación de un sensor pasivo	37
Actividad 4. Conexión y programación de un sensor pasivo	38
Conexión y programación de un sensor activo	49
Actividad 5. Programación y cableado de un sensor activo	51
Conexión y programación de un sensor ultrasónico	60
Actividad 6. Uso de un sensor ultrasónico	62
Motores, ruedas y tracción de Mulita	73
Actividad 7. Cableado y programación de un motor paso a paso	75
Mulita viajera	86
Actividad 8. Alternativas para trabajar con Mulita en el aula	88
Trabajo final	102


INTRODUCCIÓN



Mulita es un robot educativo que surgió a partir del deseo de un alumno de la Universidad Adventista del Plata de crear un dispositivo programable para resolver problemas de inteligencia artificial. Mientras cursaba la materia Robótica de la carrera de Sistemas de Información, Leandro Bazán empezó a diseñar un pequeño robot cuyo comportamiento pudiera ser modificado según el desafío que se le presentase. A partir de sus desarrollos, creamos un grupo de investigación que piense en las características que debería tener ese robot para ser utilizado en educación.

Durante varios años, pensamos e investigamos cómo crear un dispositivo educativo que acompañe la enseñanza de chicos en edad escolar y que sirva especialmente para transmitir valores a través de la tecnología. Así fue que diseñamos Mulita para motivar a los estudiantes a ser críticos, creativos y libres.

Críticos para evaluar la tecnología que utilizan. En el contexto del movimiento Hacker, esperamos que los usuarios de Mulita

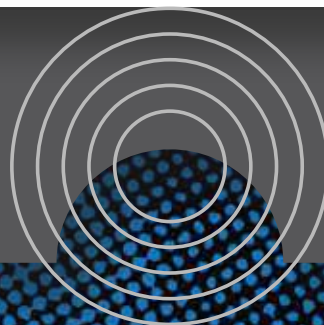


piensen mejoras para hacerle y adaptaciones para que realice funciones específicas que los estudiantes deseen.

Creativos, ya que Mulita está enmarcada en el movimiento *Maker* a través del cual cada persona “enriquece su vida creando algo nuevo y aprendiendo nuevas habilidades” (Dougherty, 2021). O, como lo describe Marya Nieto (2016): “El movimiento consiste en crear objetos de forma artesanal pero utilizando la tecnología o técnicas tecnológicas como la impresión 3D, la robótica, el corte con láser o la fresadora”. Desde este lugar, queremos que quien utilice Mulita no solo sea crítico, sino que también tenga la posibilidad de modificarla creativamente.

Libres para compartir sus producciones y para utilizar información generada por otros usuarios. En este sentido, al diseñar Mulita como un robot *open-source*, entendemos que el *software* libre “es una cuestión de libertad y no de precio... [y que estas libertades] son esenciales no solamente para el bien del usuario individual sino para la sociedad entera, porque promueven la solidaridad social: compartir y cooperar” (Stallman).

Por lo tanto, Mulita no es un robot definitivo, sino el punto de partida para que los estudiantes vuelquen sus ideas tecnológicas de manera creativa, sin miedo a modificar el dispositivo con el que están aprendiendo... mientras juegan.



PLACA CONTROLADORA

Objetivos

- Introducir los conceptos básicos de programación para empezar a trabajar con la placa controladora de Mulita.
- Comprender que nos comunicamos con el robot usando lenguajes de programación y que estos, al igual que cualquier idioma, tienen una sintaxis y una semántica.

Lenguajes de programación

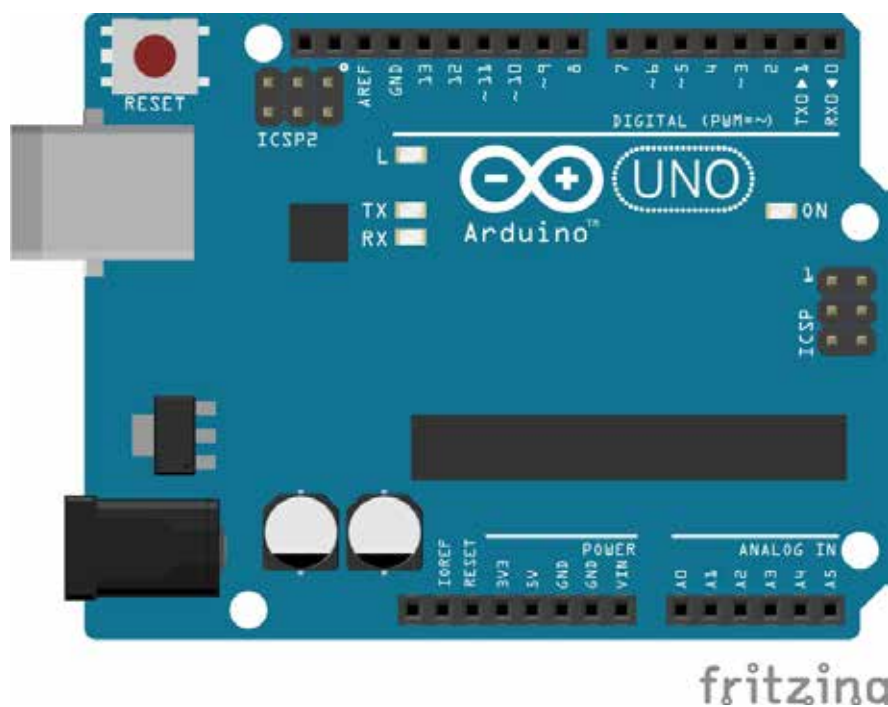
El robot educativo Mulita posee una placa programable Arduino UNO como la que se muestra en la imagen que está a continuación. Antes de empezar a trabajar con el robot, vamos a hacer algunas pruebas sencillas.

Programar un robot es decirle qué hacer. Para esto, usamos lenguajes de programación que son idiomas diseñados para comunicarnos con las computadoras de manera que nos

puedan entender. En este manual, usaremos dos tipos de lenguajes:

- miniBloq: un lenguaje de bloques pensado para enseñar a programar de manera fácil; y
- C/C++: un lenguaje basado en texto para el cual necesitaremos el software Arduino IDE.

Aclaración: el lenguaje de programación es C/C++. Arduino IDE es un “entorno de programación”, o sea, un *software* específico que nos permite comunicarnos con nuestro robot hablando en lenguaje C/C++.



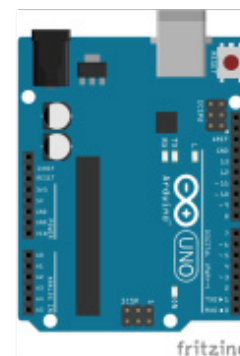
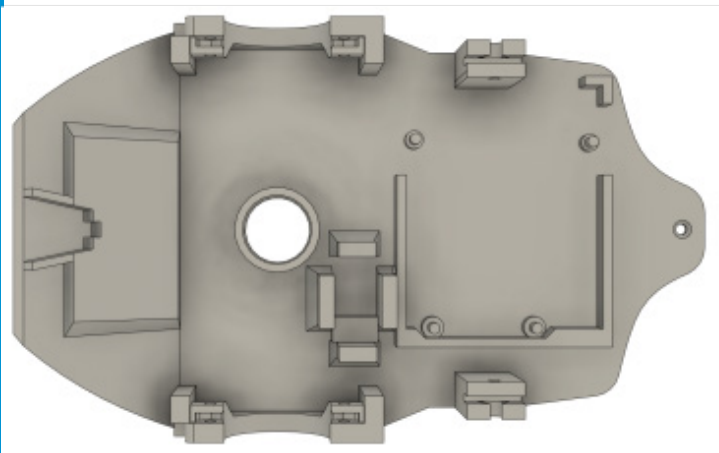
*Fuente: elaboración propia

ACTIVIDAD 1

Conexión de la placa con la computadora

Partes utilizadas:

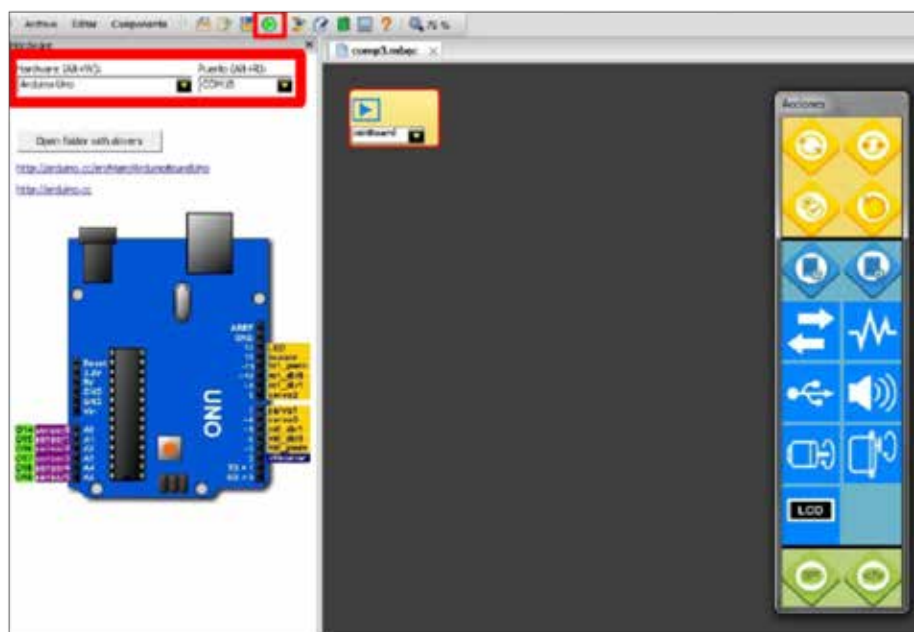
- Placa Arduino UNO
- Chasis de Mulita
- Cable USB



*Fuente: elaboración propia

Para esta primera actividad, colocamos la placa controladora sobre el chasis de Mulita. La parte trasera del chasis está diseñada para que estas partes se encastran sin necesidad de usar tornillos. Luego, conectamos la placa con la computadora utilizando el cable USB.

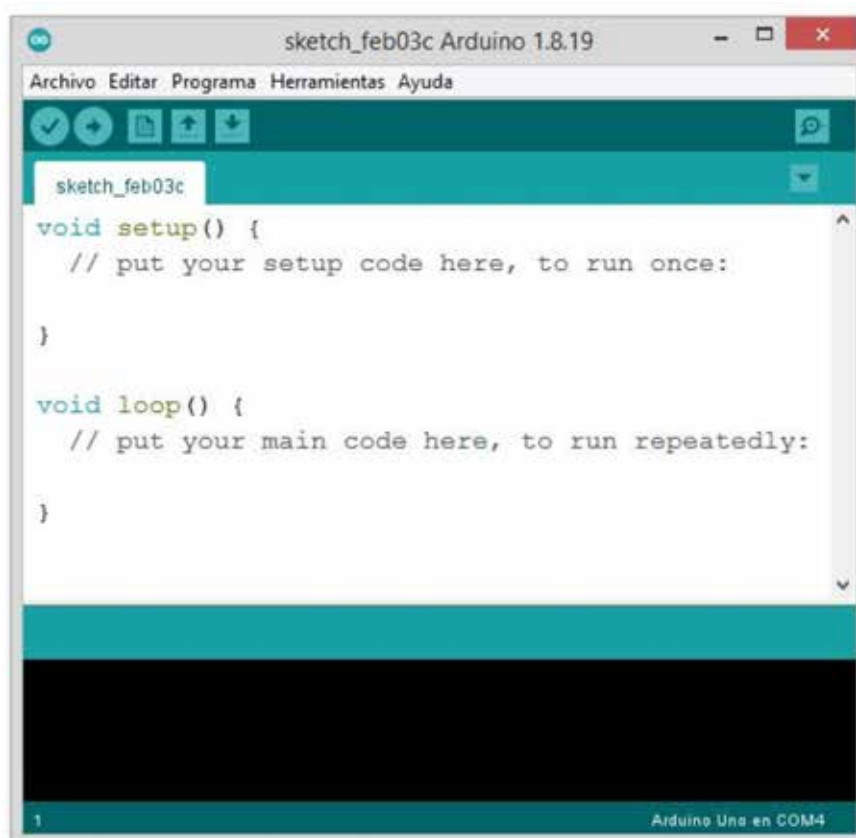
En el caso de utilizar miniBloq para programar, nos vamos a encontrar con una pantalla como la que se muestra a continuación, en la que tenemos que seleccionar en la sección "hardware" la placa que utilizaremos y, si ya tenemos conectado el cable USB y la computadora reconoció la placa, también debemos seleccionar el puerto "COM".



*Fuente: elaboración propia

Para que el programa realizado en miniBloq se descargue vía USB a la placa controladora, presionamos el botón “ejecutar” (flecha verde).

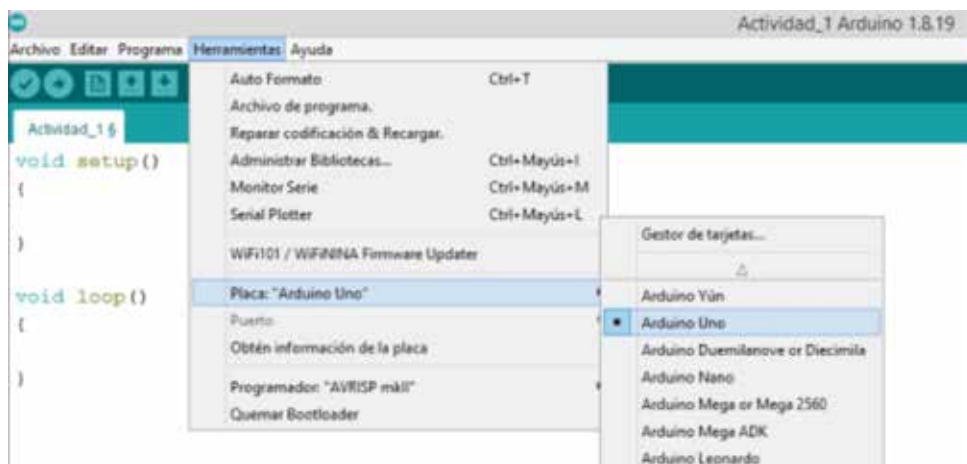
Si preferimos escribir nuestro propio programa utilizando código basado en texto, abrimos Arduino IDE, que nos mostrará la siguiente pantalla:



*Fuente: elaboración propia

Al igual que con miniBloq, desde este programa también podemos mandarle al robot las órdenes que queremos que ejecute. Pero antes tenemos que

asegurarnos de configurar dos cosas en el Arduino IDE: 1) la placa que estamos usando y 2) el puerto de comunicación. En "herramientas", seleccionamos la placa Arduino UNO.



*Fuente: elaboración propia

Luego, seleccionamos "herramientas", "puerto", y elegimos el puerto con el número más alto, ya que, normalmente, al conectar la placa a la computadora, se genera un puerto nuevo para que se comuniquen.

Ahora ya podemos descargar nuestro programa a la placa presionando el botón "subir" (tiene dibujada una flecha que apunta hacia el costado). En la parte inferior de la pantalla, podemos ver los mensajes del compilador (en caso de haber algún error, va a aparecer un texto en rojo sobre fondo negro; en caso de que la descarga haya sido exitosa, lo informará con un texto de color blanco).





*Fuente: elaboración propia

Lo que hemos hecho hasta el momento ha sido grabar en nuestra placa controladora un programa que no hace nada y que lo utilizamos para asegurarnos de que la placa del robot y la computadora están comunicadas. Veamos qué significa cada parte del código.

```
void setup()  
{  
}
```

```
void loop()  
{  
}
```

Todos los programas que escribamos con Arduino IDE tendrán estas dos funciones: setup() y loop(). Dentro de las primeras llaves {}, las correspondientes a la función setup(), escribiremos un código que se ejecutará una única vez; mientras que todo lo que

escribamos dentro de las llaves de loop() se repetirá siempre que la placa esté encendida.

Por último, al igual que en cualquier idioma, el código se leerá de arriba hacia abajo y cada sentencia que escribamos será analizada y ejecutada por el procesador de la placa programable.

Transversalidad

Podemos clasificar los lenguajes en dos tipos: 1) naturales y 2) formales. Los primeros son los que usamos para comunicarnos normalmente, como el español, inglés, alemán e incluso idiomas con alfabetos distintos como el hebreo, griego o chino mandarín. Los segundos son los que utilizamos para programar computadoras y, entre otras características, nos permiten expresar frases ambiguas. Por ejemplo, en un lenguaje natural podemos hablar con ironía, crear metáforas e incluso acompañar lo que decimos con gestos; con los lenguajes formales, tenemos que expresar órdenes específicas para que sean ejecutadas por máquinas. Cuando expresamos una idea mediante el uso de fórmulas matemáticas, estamos usando un lenguaje formal, mientras que, cuando armamos una oración en inglés, estamos utilizando un lenguaje natural.

Para investigar

¿Qué otros entornos de programación basados en bloques se podrían utilizar para programar a Mulita?

Definición

Lenguaje formal es “cualquier lenguaje diseñado por humanos que tiene un propósito específico, como la representación de ideas matemáticas o programas de computadoras; todos los lenguajes de programación son lenguajes formales” (Downey et al., 2002).

Resumen

En esta primera actividad, comunicamos a la placa controladora con la computadora y vimos cómo enviar a esta un programa propio, aunque este no haga nada. Además, aprendimos que hay distintos lenguajes que nos permiten comunicarnos con un dispositivo tecnológico.

CONEXIÓN Y PROGRAMACIÓN DE UN LED CON UN *PROTOBOARD*

Objetivos

- Introducir el concepto de salida digital.
- Vincular la placa controladora con un dispositivo de salida para programar su comportamiento.

Entradas y salidas digitales

La placa controladora de Mulita permite manejar dispositivos de entrada y salida a partir de lo que llamamos “pines”. Un pin digital podrá entregar 5 voltios en caso de “estar encendido” o 0 voltios si lo apagamos. En el código que tenemos a continuación, vemos cómo hacer para que un LED conectado al pin digital 13 se encienda y se apague a intervalos de medio segundo. ¿Por qué elegimos el pin número 13? Porque nuestra placa tiene un LED asociado a este pin y, por lo tanto, en

una primera instancia vamos a poder testear nuestro código sin necesidad de armar un pequeño circuito.

```
void setup()
{
  pinMode(13, OUTPUT); // Seteamos el pin 13 como salida.
}
void loop()
{
  digitalWrite(13, LOW); // Apagamos el pin 13.
  delay(500);
  digitalWrite(13, HIGH); // Encendemos el pin 13.
  delay(500);
}
```

Las tres sentencias que utilizamos son las siguientes:

```
pinMode(13, OUTPUT);
```

Con esta indicación, configuramos al pin número 13 para que funcione como salida utilizando la palabra reservada **OUTPUT**.

```
digitalWrite(13, LOW);
```

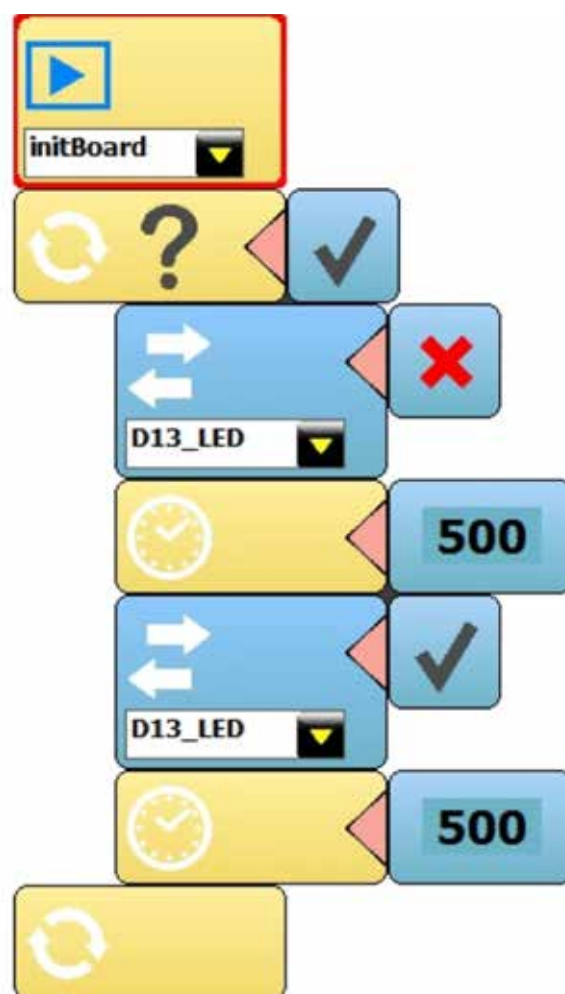
Con esta orden, le indicamos al pin si debe entregar 0 o 5 voltios. En caso de usar la palabra **LOW**, estaremos "apagando" el dispositivo conectado al pin, mientras que con **HIGH**, lo encenderemos (5V).

```
delay(500);
```

Esta sentencia nos permite pedirle al procesador que espere antes de ejecutar una nueva orden. Su único parámetro de entrada define un tiempo en milisegundos. Por lo tanto, para

que nuestro LED “parpadee” a intervalos regulares de medio segundo, utilizamos el número 500. Si quisiéramos que los intervalos fueran de un segundo, el parámetro de entrada a utilizar sería 1000.

Si quisiéramos realizar este mismo programa utilizando mini-Bloq, podríamos hacerlo con el código que se muestra en la siguiente imagen:



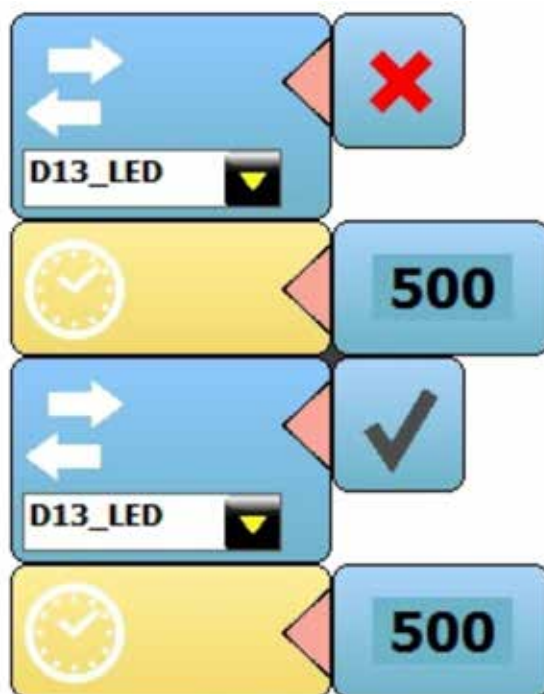
*Fuente: elaboración propia

Como vimos, Arduino IDE tiene una función `loop()` que es un ciclo infinito. En el caso de miniBloq, ese ciclo tenemos que generarlo nosotros a partir de los bloques que se muestran en la siguiente imagen. El bloque con el signo de interrogación es el equivalente a una sentencia "while" (mientras) y el tilde que está al lado significa que el parámetro de entrada para este ciclo "while" será siempre "verdadero".



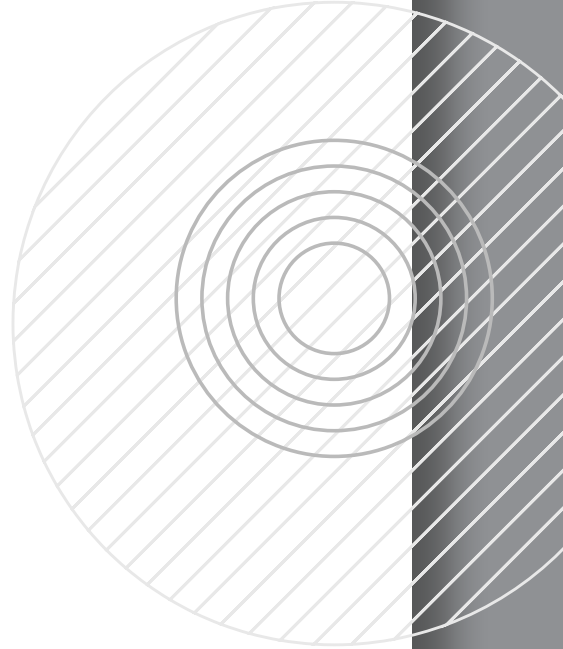
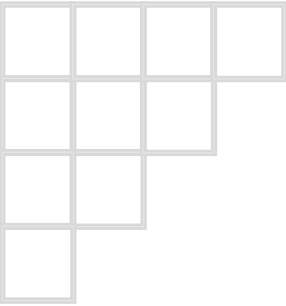
*Fuente: elaboración propia

Por otro lado, la secuencia de "apagar salida digital 13 – esperar medio segundo – encender salida digital 13 – esperar medio segundo" la programaremos de la siguiente manera:



*Fuente: elaboración propia

Aclaración: el código que enciende y apaga el pin tiene que quedar encerrado entre los dos bloques que se generaron cuando se agregó el ciclo "while".



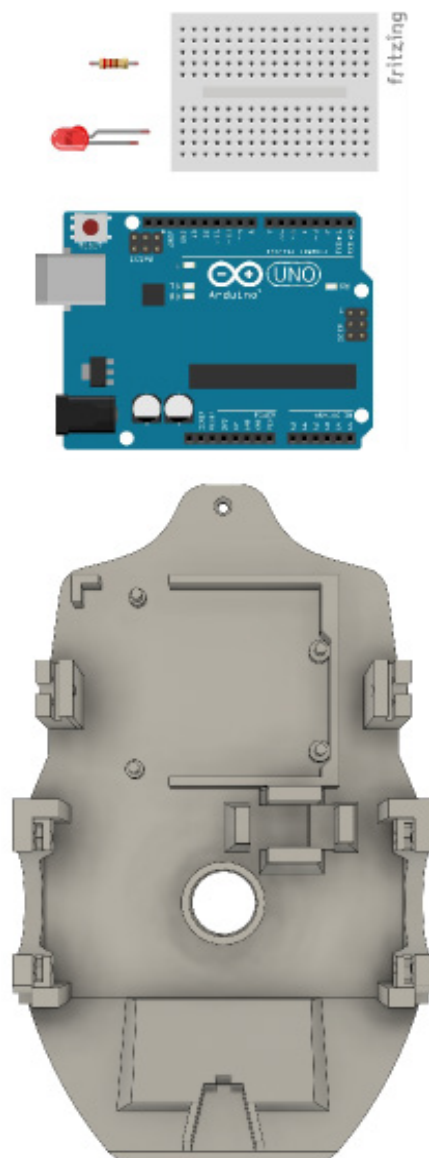
ACTIVIDAD 2

Conexión de un LED a la placa

Conectar un LED a la placa Arduino UNO. Luego programar la placa para que el LED se encienda y se apague a intervalos de tiempo regulares.

Partes utilizadas:

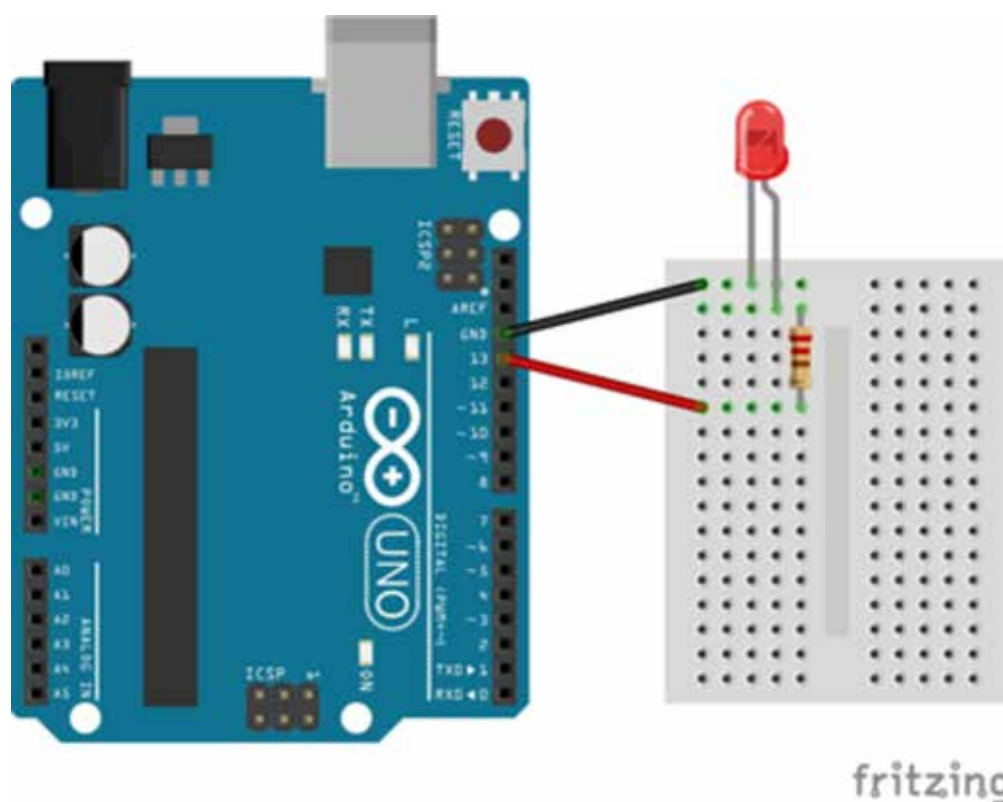
- Placa Arduino UNO
- Chasis de Mulita
- Cable USB
- *Mini protoboard*
- LED
- Resistencia
- Cables de un hilo



*Fuente: elaboración propia

Utilizando el mismo código, vamos a hacer que esta vez se encienda un LED conectado a la placa Arduino UNO a través de un *protoboard*. Para esto, armamos el circuito que se encuentra en la siguiente imagen:





*Fuente: elaboración propia

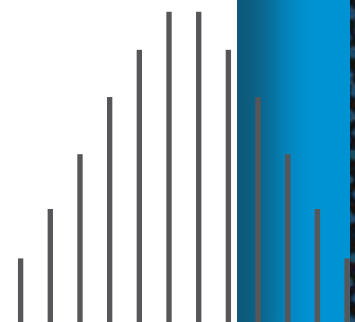
En el esquema que tenemos, conectamos el pin digital 13 a una línea del *mini protoboard*. Siguiendo el circuito, colocamos una resistencia entre este cable y la pata larga del LED (ánodo o positivo). Luego, a través de un cable de un solo hilo, conectamos la pata corta del LED (cátodo o negativo) al pin GND (tierra) de la placa controladora.

Con nuestro ejemplo en funcionamiento, vamos a realizar modificaciones tanto en el código como en el circuito para conseguir otros resultados:

- Modificar el código para que el LED titile más rápido.
- Volver a modificarlo para que el tiempo que el LED está apagado sea más largo que el tiempo que está encendido.
- Agregar un segundo LED al circuito y programar la placa para que se enciendan y se apaguen de manera alternada (cuando un LED está encendido, el otro está apagado).
- Utilizando tres LEDs de colores, armar y programar un semáforo.

Transversalidad

Desde la electrónica, podemos estudiar la Ley de Ohm para calcular qué resistencias podrían utilizarse en nuestro circuito. Incluso para entender por qué, dependiendo de la resistencia que usemos, el LED brilla más o menos. "Las resistencias impiden el flujo de corriente. Su capacidad para hacer esto se mide en ohmios... La corriente ' I ', que fluirá a través de una resistencia con resistencia ' R ', dado un voltaje aplicado, ' V ', es ' $I=V/R$ '. Esto se conoce como ley de Ohm" (Jones et al., 2018).



Para investigar

¿Qué es un LED? ¿Y una resistencia? ¿Qué relación hay entre el flujo de corriente en un circuito y estos componentes electrónicos?

Definición

"El tema del paso de información *hacia* la computadora o desde ella se llama entrada/salida o normalmente E/S" (Tanenbaum, 1986). También se pueden utilizar los términos en inglés: *input/output* o I/O.

Resumen

En esta actividad trabajamos utilizando los pines de salida digitales de la placa Arduino y armamos un circuito sencillo. Es importante empezar a pensar a nuestra placa controladora como un dispositivo que trabaja a partir de un modelo de entradas y salidas (*in/out*): llamamos "entradas" a los pines o dispositivos que mandan información hacia el procesador y "salidas" a los que la envían hacia afuera de la placa.

CONEXIÓN Y PROGRAMACIÓN DE UN SERVO

Objetivos

- Aprender a utilizar un servomotor como salida.
- Introducir el concepto de variable en programación.

Mulita dibujante

Un servo está formado por un motor eléctrico, una reducción de engranajes y un potenciómetro; todo esto encapsulado en una carcasa de plástico. Con él podemos mover partes de un dispositivo con mucha precisión. En el caso de Mulita, lo utilizaremos para desplazar hacia arriba y hacia abajo un fibrón que nos permitirá dibujar la trayectoria que siga el robot. En esta actividad, vamos a realizar un programa sencillo que simule el movimiento necesario para desplazar el fibrón.

Siempre que se utilice un componente electrónico es conveniente descargar la hoja de datos del fabricante. Para esta actividad, utilizaremos un servo modelo sg90 del cual podremos encontrar mucha información en internet. En particular, su hoja de datos (*datasheet* en inglés) nos servirá para conocer datos tan importantes como la función de cada cable de conexión, dimensiones, torque y el voltaje con el que lo tenemos que alimentar.

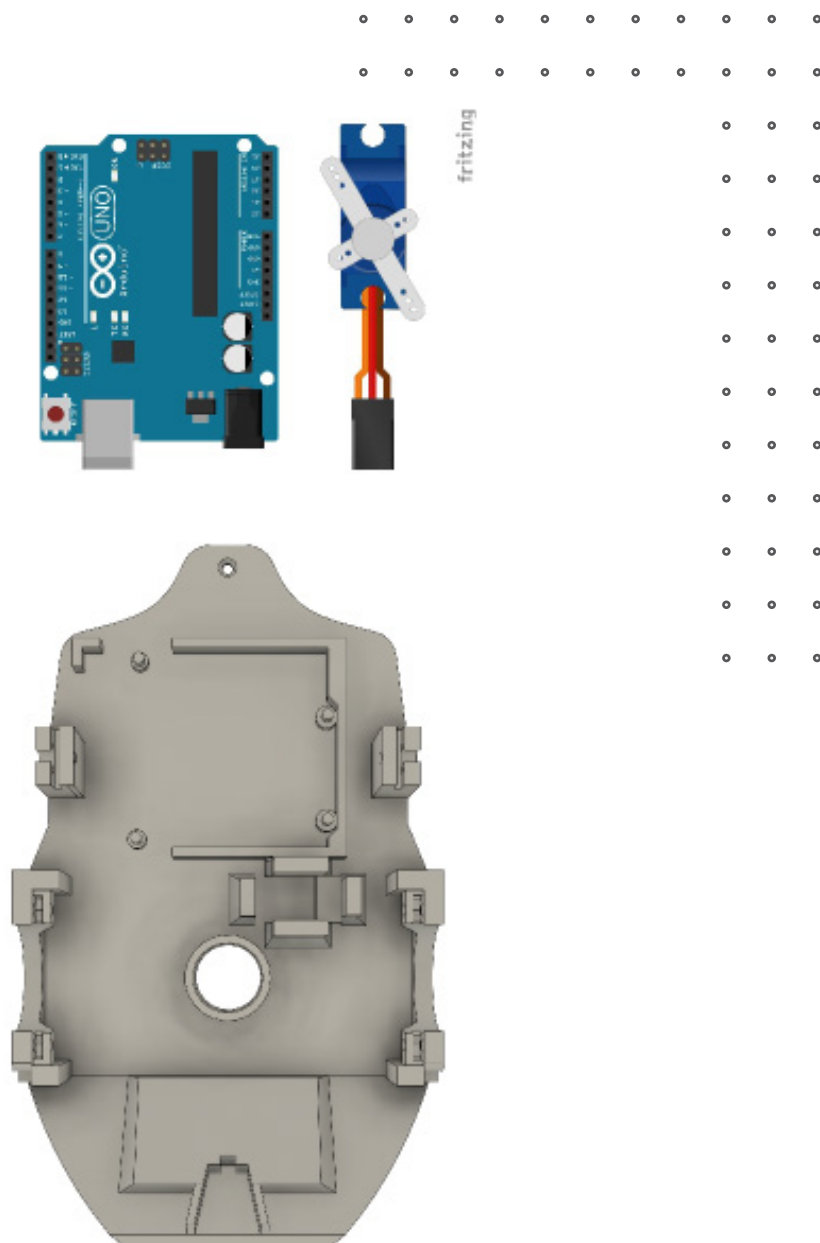
ACTIVIDAD 3

Conexión de la placa a un servo y programación

Conectar la placa controladora con un servo sg90 y programarla para que este se mueva en dos direcciones: "ida y vuelta".

Partes utilizadas:

- Placa Arduino UNO
- Chasis de Mulita
- Servo sg90
- Cables de un hilo macho-macho
- Cable USB

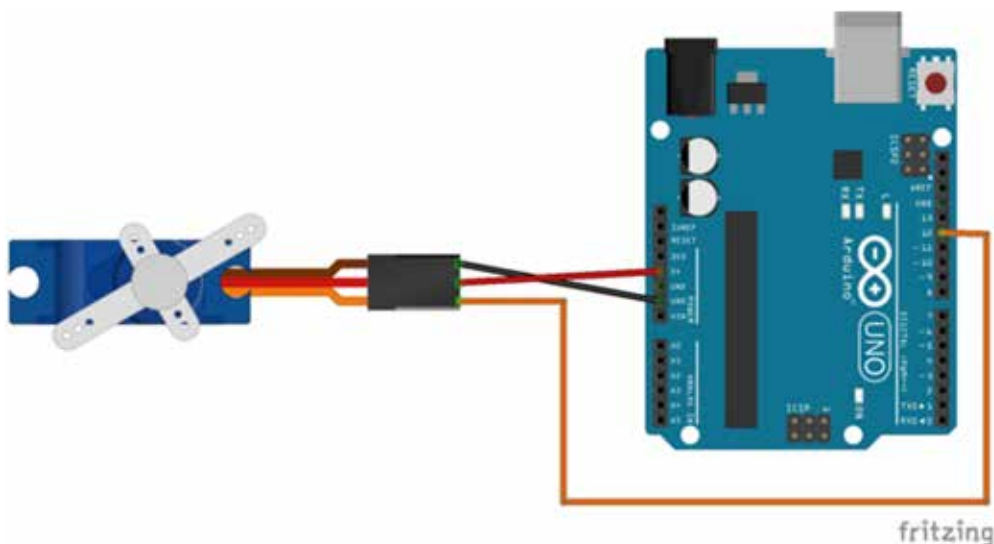


*Fuente: elaboración propia

Al igual que en la actividad anterior, trabajamos sobre el chasis de Mulita y montamos el servo y la placa en los soportes que están diseñados especialmente para mantenerlos firmes. Por otro lado, si descargamos la hoja de datos, podemos ver que el servo posee tres cables de conexión:

- PWM (cable naranja)
- Vcc (cable rojo)
- *Ground* (tierra, cable marrón)

Por lo tanto, conectamos estos como se muestra en el esquema que aparece a continuación. Conviene utilizar cables de un hilo macho-macho entre el servo y los pines de la placa Arduino.



*Fuente: elaboración propia

Aclaración: a veces, cuando trabajamos con motores o servomotores, como en este caso, puede ocurrir que la placa se “resetea” al intentar ejecutar las órdenes que le pedimos. La mayoría de las veces, esto se puede resolver alimentando al servo sin utilizar la placa; o sea, conectando los cables marrón y rojo a una fuente externa de alimentación.



A continuación, vamos a escribir un programa para que el servo mueva su eje desde una posición inicial hacia una segunda posición y se mantenga en cada una de ellas durante un segundo.

```
#include <Servo.h>
Servo miServo;
int pos1 = 0;
int pos2 = 180;

void setup()
{
  miServo.attach(12);
}

void loop()
{
  miServo.write(pos1);
  delay(1000);
  miServo.write(pos2);
  delay(1000);
}
```

Al comienzo de nuestro código, tenemos dos líneas de código que son indispensables para trabajar con servo:

```
#include <Servo.h>
Servo miServo;
```


La primera de ellas incluye la librería de servos en nuestro programa. Esta línea de código nos permite utilizar las funciones que usaremos a continuación. La segunda la necesitamos para "crear" nuestro servo en el programa. Sin entrar en muchos detalles, en este tipo de ejemplos tenemos que inicializar cada servo que utilicemos desde el código.

```
int pos1 = 0;  
int pos2 = 180;
```

Luego tenemos dos líneas de código que usaremos para crear variables que nos ayudarán a mantener más legible y claro nuestro código. En este caso, cada variable es un número entero (por eso usamos la palabra `int`), una de ellas se llamará `pos1` y la otra `pos2`, y en ambas guardaremos un número que representará un ángulo. En este punto solo necesitamos entender que, de ahora en más, cuando usemos el término "`pos1`", el procesador utilizará el número 0 en su lugar y, cuando utilicemos el término "`pos2`", el procesador lo reemplazará internamente por el número entero 180.

```
miServo.attach(12);
```

De la misma manera que hicimos con los pines digitales, tenemos que decirle a nuestro programa qué pin se encargará de mandar datos al servo. En nuestro ejemplo, conectamos el cable naranja al pin 12 y,



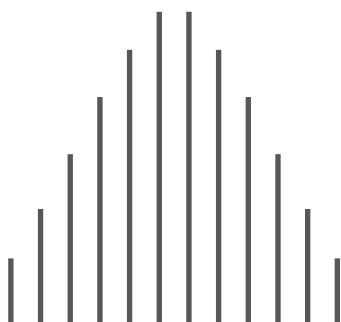
por lo tanto, le indicamos a la placa que "miServo" (nombre con el que creamos a nuestro servo) recibirá órdenes a través del pin 12 (`attach[12]`).

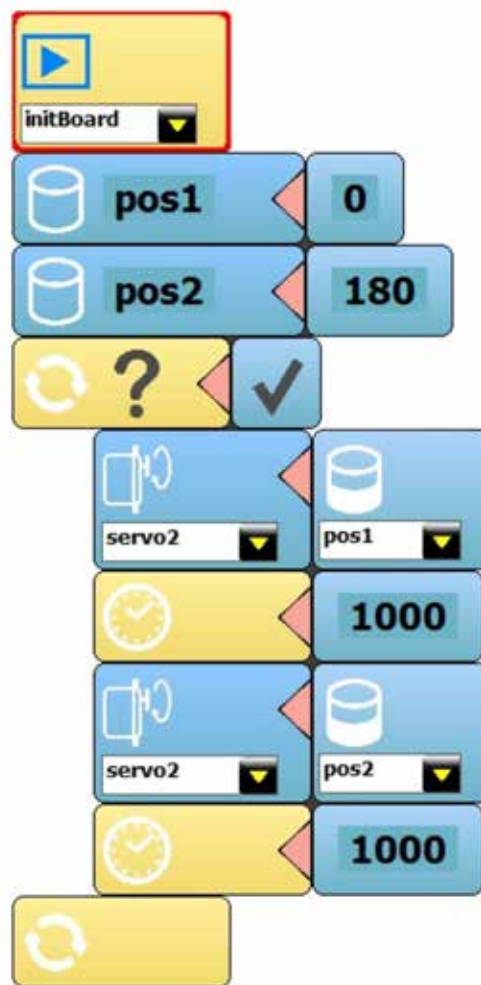
Aclaración: es importante notar que entre el nombre del servo y la orden que indica el número de pin no hay espacios, sino que hay un punto.

```
miServo.write(pos1);
```

La última sentencia que tenemos que entender para nuestro programa es precisamente la que le indica al servo en qué ángulo posicionar su eje. Para eso, volvemos a utilizar la notación anterior con el punto entre el nombre de nuestro servo y la sentencia que queremos que se ejecute y, en este caso, vamos a escribir el nombre de la variable que contenía el valor 0.

Para replicar este ejemplo utilizando miniBloq, vamos a utilizar el pin 8 en lugar del 12. Nuestro código quedará como muestra la siguiente imagen:





*Fuente: elaboración propia

A diferencia del código que escribimos en Arduino IDE, en miniBloq no tenemos que agregar la librería de servos ni crear la instancia de este. Por otro lado, al igual que nos pasó en la actividad 2, simulamos la función `loop()` utilizando un ciclo "while" infinito.

Para crear nuestras variables, usamos los siguientes bloques y, como hicimos con Arduino, elegimos los

nombres nosotros. Luego asignamos los valores que queremos que guarden inicialmente.



*Fuente: elaboración propia

El otro bloque que nos interesa es el que asigna al servo elegido el ángulo al que girar su eje. En esta asignación tenemos que tener en cuenta dos cosas:

- Tenemos que elegir el servo que queremos entre tres opciones que nos ofrece miniBloq y chequear en el esquema que se encuentra en la sección "hardware" a qué pin se conecta. En nuestro ejemplo vamos a usar el servo2 cuyo pin es el 8.
- El valor que le asignemos será una variable de las que generamos previamente.



*Fuente: elaboración propia

Actividades

A partir de lo realizado, desarrollar los siguientes programas:

- Hacer que el eje del servo se desplace de 0 a 90 grados una única vez.
- Hacer que el eje del servo se desplace de 0 a 180 grados de manera gradual.
- Agregarle al punto 2 un código para que el eje del servo vuelva desde los 180 grados a 0.

Transversalidad

Desde la asignatura Física se puede estudiar el concepto de torque. En la hoja de datos del servo encontramos que el modelo que estamos usando posee un torque de 2,5 kg/cm. A partir de este dato, podemos experimentar con el servo y explicar de manera práctica qué significa.

Para investigar

Servos: ¿cuáles son las partes de un servo? ¿Cómo interactúan entre ellas y qué función cumple cada una? ¿Se podrían reemplazar los motores a los que se conectan las ruedas por servos? ¿Por qué?

Variables: ¿qué ventaja tiene utilizar una variable en vez de un valor fijo? ¿Cómo se llama a la práctica de pasar valores numéricos sin utilizar variables?

Definición

La variable es un espacio de memoria al que referenciamos con un nombre y en el que guardamos un dato que podemos modificar y consultar.

Resumen

En esta actividad programamos a nuestra placa para que pueda mover un dispositivo conectado a un servo. Los servos poseen un motor eléctrico en su interior, pero, en algunos casos, solo pueden mover su eje a posiciones entre 0 y 180 grados como en nuestro ejemplo. Además, utilizamos variables para reemplazar valores numéricos que pasamos como parámetros de entrada en algunas funciones.

CONEXIÓN Y PROGRAMACIÓN DE UN SENSOR PASIVO

Objetivos

- Introducir el concepto de entrada analógica.
- Aprender a testear el funcionamiento de un sensor.

Comunicación serial

Cuando conectamos nuestra placa Arduino UNO con la computadora a través del cable USB, establecemos una conexión entre un emisor y un receptor. Decimos que la información viaja a través del cable USB de manera serial. Esto es, "de a un bit a la vez", a diferencia de la comunicación en paralelo en la que se pueden mandar varios bits al mismo tiempo. En la actividad que vamos a realizar a continuación, utilizaremos el cable USB para tomar información de la placa y mostrarla en la computadora.

ACTIVIDAD 4

Conexión y programación de un sensor pasivo

Joseph Jones y Anita Flynn nos cuentan que “los sensores son meros transductores que convierten algunos fenómenos físicos en señales eléctricas que el microprocesador puede leer” (Jones et al., 2018). En el caso de las fotorresistencias, decimos que son sensores pasivos porque no emiten señales ni modifican el medio en el que se encuentran. Simplemente cambian su capacidad resistiva dependiendo de la luminosidad del lugar en el que se encuentran. En esta actividad conectaremos un sensor pasivo a la placa controladora y visualizaremos los valores que detecta mediante la comunicación serial del USB.

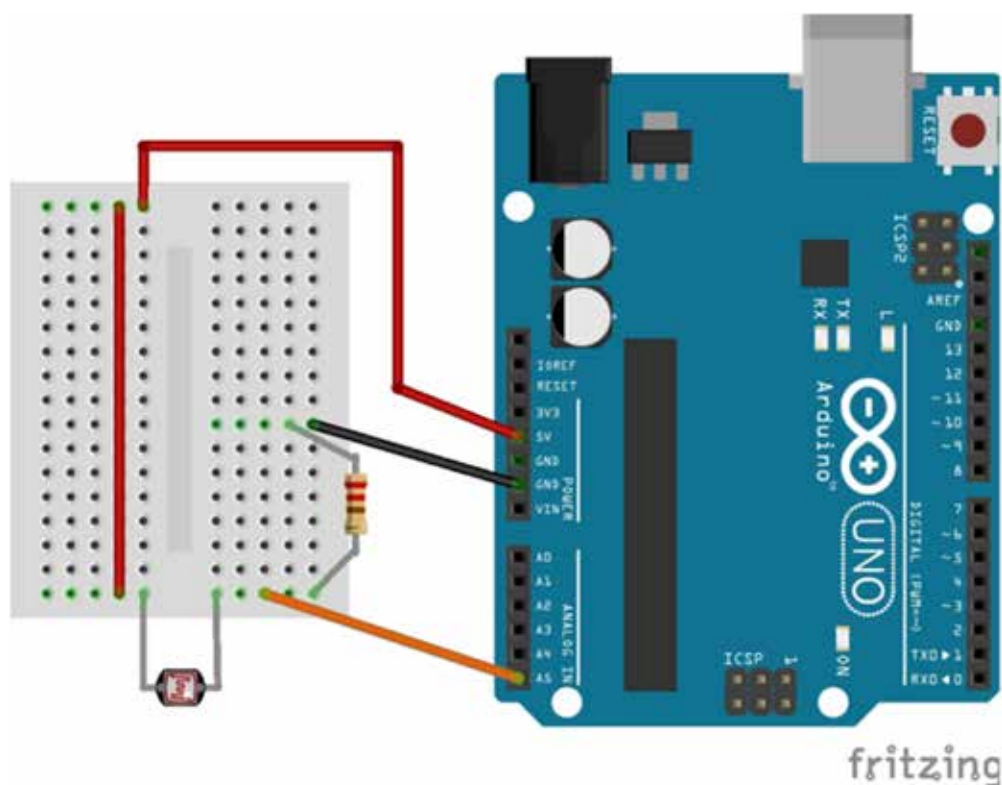
Partes utilizadas:

- Placa Arduino UNO
- Chasis Mulita
- Resistencia

- Célula fotosensible (LDR)
- *Mini protobard*
- Cables de un hilo



*Fuente: elaboración propia




*Fuente: elaboración propia

Con el circuito armado, programamos la placa para ver en la pantalla de la computadora los valores con los que trabaja el sensor. Es importante conocer estos valores cada vez que trabajamos con un sensor, ya que conocer su comportamiento nos servirá para poder programarlo.

```
int LDR = 0;

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

void loop()
```

```
{  
  LDR = analogRead(5);  
  Serial.println(LDR);  
  delay(500);  
}
```

En la primera línea de código, por fuera de las funciones `setup()` y `loop()`, creamos una variable a la que llamaremos LDR.

```
int LDR = 0;
```

En un principio, guardamos el valor 0 en la variable LDR, pero más adelante usaremos este espacio de memoria para guardar lo que detecte nuestro sensor.

```
Serial.begin(9600);
```

Dentro de la función `setup()`, configuramos la velocidad de transferencia de datos que usaremos entre la placa y la computadora. Como la comunicación es de tipo serial, utilizamos esta sentencia que indica que dicha velocidad será de 9600 baudios (bits por segundo).

```
LDR = analogRead(5);
```

Dentro del ciclo `loop()`, le asignamos a nuestra variable LDR lo que detecte la entrada analógica identificada como A5 en la placa Arduino. El parámetro de entrada, en este caso 5, identifica a dicho pin.

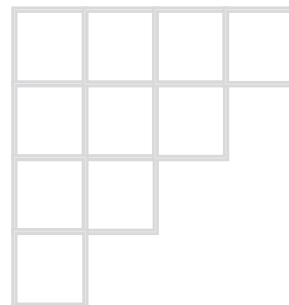

```
Serial.println(LDR);
```

Luego mostramos por pantalla el valor guardado en la variable LDR. Para eso, usamos otra función relacionada con la comunicación serial y le pasamos como parámetro de entrada el nombre de la variable.

```
delay(500);
```

Por último, agregamos una espera de medio segundo para poder visualizar mejor los resultados obtenidos por el sensor.

Una vez que hayamos descargado el programa a la placa —el cable USB debe permanecer conectado—, abrimos el “serial monitor” con el botón que se encuentra a la derecha arriba de la pantalla de Arduino IDE, tal como se muestra en la siguiente figura:





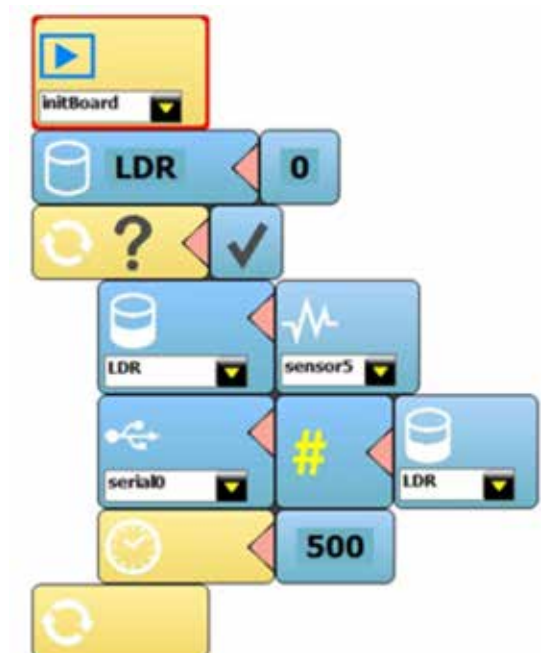
*Fuente: elaboración propia

A continuación, se abrirá una nueva pantalla que nos mostrará los datos que llegan desde la placa. En la parte superior izquierda, el "serial monitor" indicará el puerto que se está utilizando. En nuestro ejemplo es el COM20.



*Fuente: elaboración propia

El código en miniBloq se verá de la siguiente manera:



*Fuente: elaboración propia

En cuanto el código comience, creamos la variable LDR y le asignamos el valor 0. Estos bloques son

equivalentes a la línea de código que usamos en Arduino IDE: `int LDR = 0;`



*Fuente: elaboración propia

Creamos un ciclo infinito como en la actividad anterior y, dentro de este, le asignamos a nuestra variable LDR el valor detectado por la entrada analógica 5 (sensor5 en miniBloq).

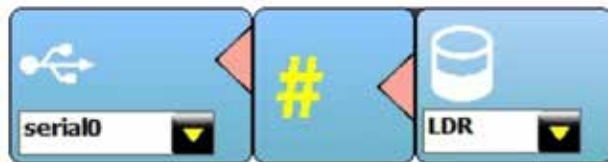


*Fuente: elaboración propia

A continuación, mostramos por pantalla lo que se almacenó en la variable. Para esto, utilizamos tres bloques juntos:

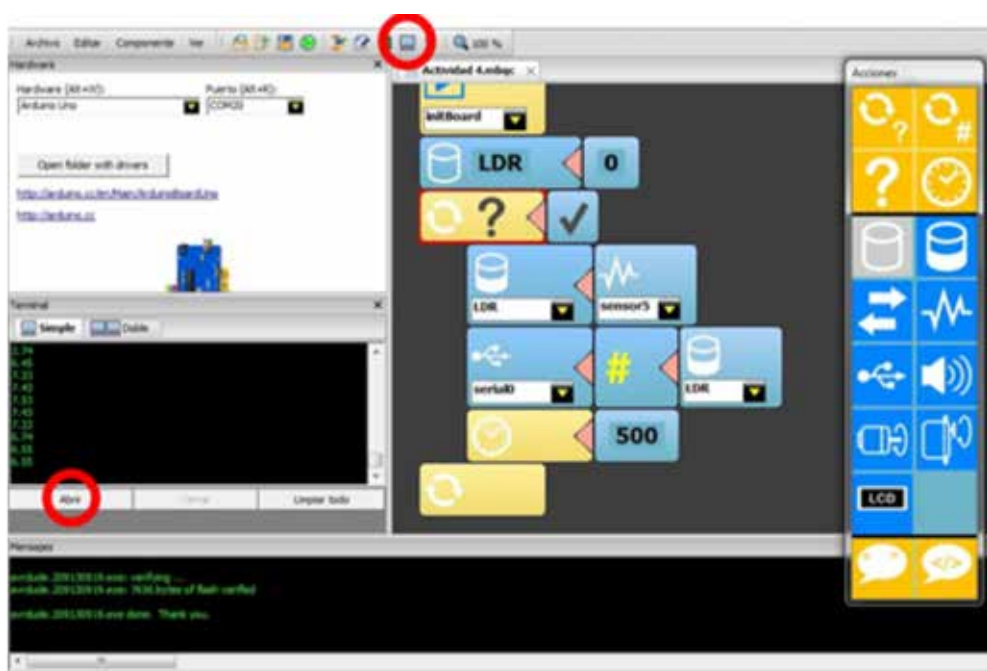
- El bloque que permite enviar datos a través del cable USB.
- El bloque numeral que indicará que vamos a enviar datos numéricos.
- Un bloque que asigne la variable LDR como parámetro de entrada para ser enviado a través del USB.

Por último, al igual que hicimos con Arduino IDE, agregamos un pequeño tiempo de espera al final.



*Fuente: elaboración propia

Una vez terminado el código, lo enviamos a la placa y, cuando ya esté grabado en ella, presionamos el botón "terminal" de miniBloq (tiene como ícono una pantalla). Se abrirá una pequeña pantalla en la que, luego de presionar "abrir", podremos ver los datos enviados por la placa.



*Fuente: elaboración propia

Actividades

- Si se dispone de otros sensores pasivos, armar un circuito para testarlos utilizando el mismo código de esta actividad.
- Desarrollar un código con el que se pueda testear más de un sensor al mismo tiempo.
- Armar un circuito unificando el cable que va a Gnd para utilizar más de un sensor.

Transversalidad

Desde espacios curriculares como Química o Física, se pueden estudiar distintos componentes que son modificados por situaciones externas a ellos y que podrían utilizarse como sensores pasivos a partir de cambios de temperatura, presión, humedad, etc.

Para investigar

¿Qué otros sensores pasivos podemos encontrar en una casa de electrónica? ¿Qué mediciones podríamos realizar con esos sensores? ¿Qué hace que un sensor pasivo sirva para detectar algo?

Definición

"USB es un estándar de serie (que usa un cable delgado) que es capaz de transferir alrededor de 1,5 MBps. Es bien conocido en la industria de las PC y permite



que los periféricos se conecten y desconecten rápidamente de la computadora, incluso con la alimentación encendida. Para los robots, un USB puede ser una forma fácil de conectarse a otra computadora para comunicarse o descargar” (Bergren, 2003).

Resumen

En esta actividad, utilizamos un componente electrónico —una resistencia fotosensible— como si fuera un sensor de nuestro robot. A partir de la comunicación que establecemos con la computadora a través de un cable USB, mostramos el comportamiento del sensor en diversas situaciones en la pantalla de la computadora. De esta manera pudimos sacar conclusiones sobre el uso de este dispositivo de entrada.

CONEXIÓN Y PROGRAMACIÓN DE UN SENSOR ACTIVO

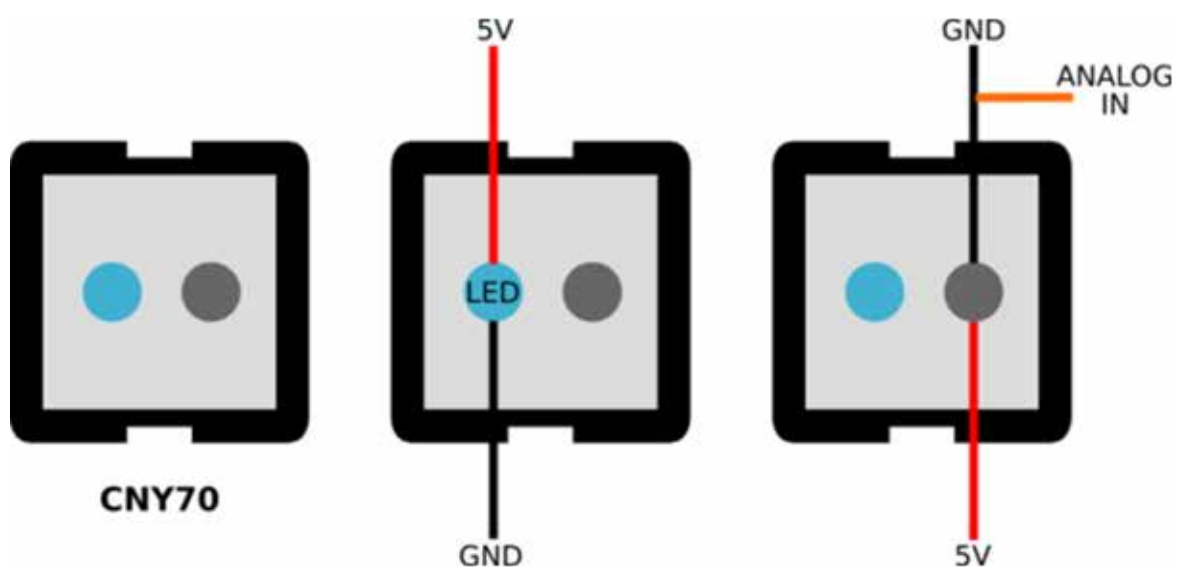
Objetivos

- Entender la diferencia entre sensores activos y pasivos desde un ejemplo práctico.
- Aumentar la complejidad de los programas que se desarrollan mediante la combinación de entradas y salidas.

Robots seguidores de línea

En las competencias de robótica para estudiantes, suele haber una prueba en la que los robots tienen que realizar un recorrido dibujado en una pista. Por lo general, este recorrido es una línea negra de 2 cm de ancho dibujada sobre un suelo blanco. El contraste entre el blanco y el negro hace que sea más fácil para los sensores del robot saber si está sobre la línea negra o sobre el piso blanco. Para este tipo de pruebas, es común usar el sensor infrarrojo CNY70, un sensor activo

que emite un haz de luz infrarrojo y luego mide con qué intensidad este vuelve al sensor. Internamente, este sensor tiene dos componentes: un diodo emisor y un fotorresistor que funciona como receptor. Por lo tanto, cuando armemos el circuito para testearlo, vamos a hacerlo en dos etapas: primero, el LED infrarrojo y, luego, el fotorresistor.



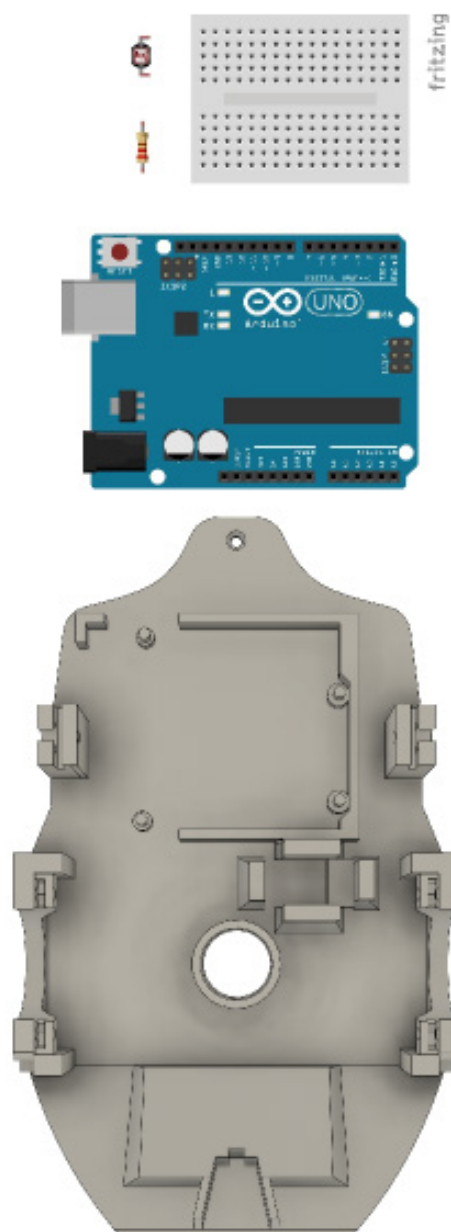
*Fuente: elaboración propia

ACTIVIDAD 5

Programación y cableado de un sensor activo

Partes utilizadas:

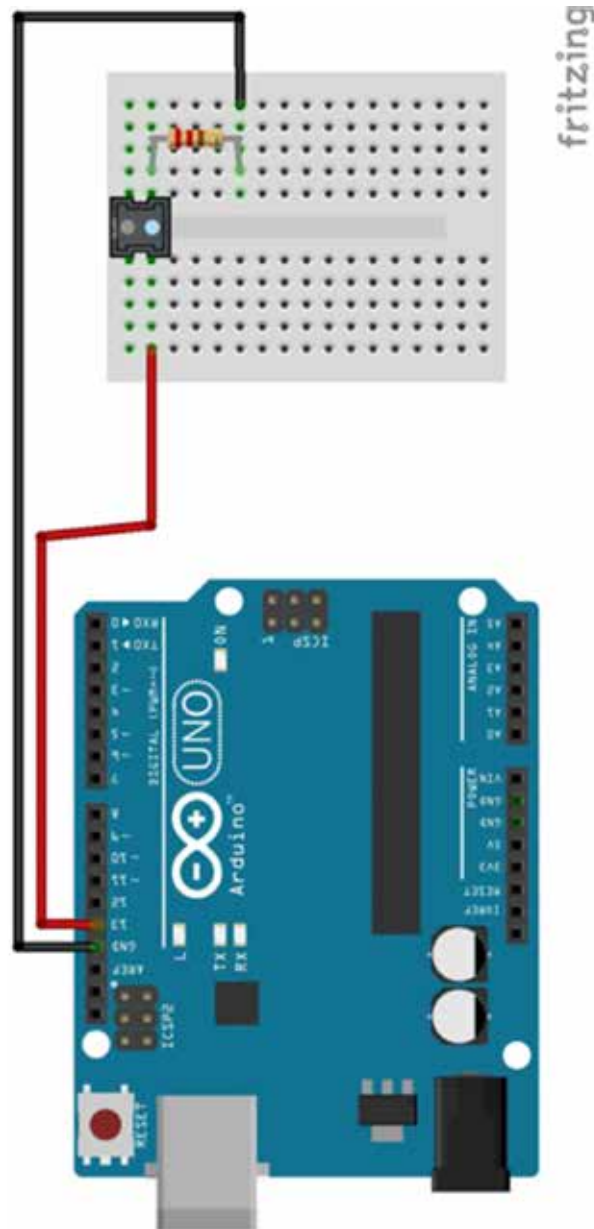
- Placa Arduino UNO
- Chasis Mulita
- Sensor CNY70
- Dos resistencias
- Cables de un hilo
- *Mini protoboard*



*Fuente: elaboración propia

Lo primero que haremos será conectar la parte del sensor CNY70 que corresponde al LED. Luego, lo vamos a programar para que se encienda, aunque no podremos verlo a simple vista. Podemos corroborar

el funcionamiento del emisor a través de la cámara de un celular. Esto se debe a que la luz que emite el LED es infrarroja.

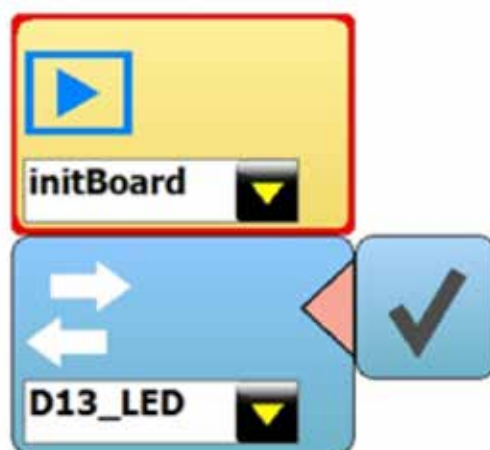


*Fuente: elaboración propia

Para hacer funcionar esta mitad del circuito, simplemente configuramos el pin digital 13 como salida y lo encendemos. Esto lo podemos hacer dentro de la función `setup()`.

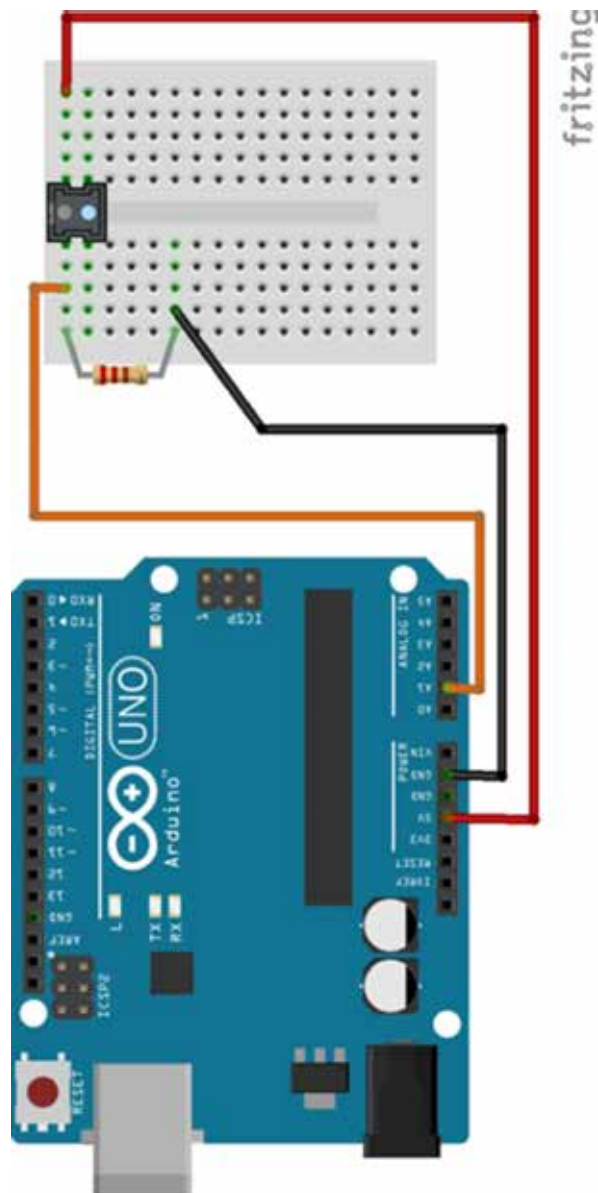
```
void setup()
{
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
}
void loop()
{ }
```

Este mismo código se puede escribir en miniBloq de la siguiente manera:



*Fuente: elaboración propia

Ahora vamos a cablear la otra mitad del circuito, la que corresponde al fototransistor que recibe el rebote de lo emitido por el LED infrarrojo. Para esto, armamos el siguiente circuito:



*Fuente: elaboración propia

Como conectamos un cable entre la resistencia y el sensor a la entrada analógica A1, la información se leerá en dicho pin. Para eso, completamos el código que ya teníamos de la siguiente manera:

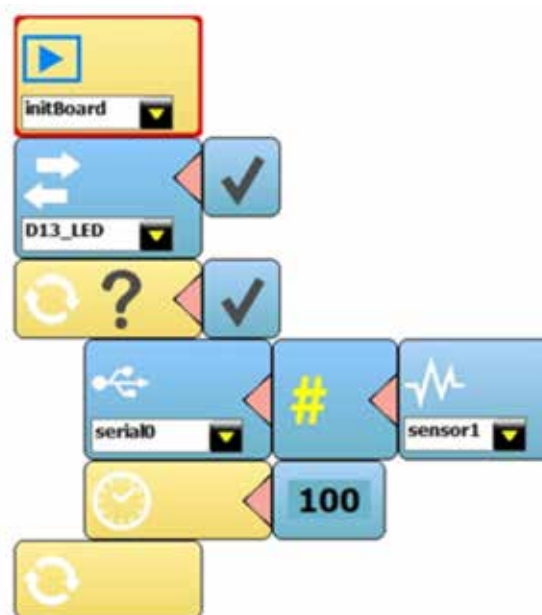

```

void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
}

void loop()
{
  Serial.println(analogRead(1));
  delay(100);
}

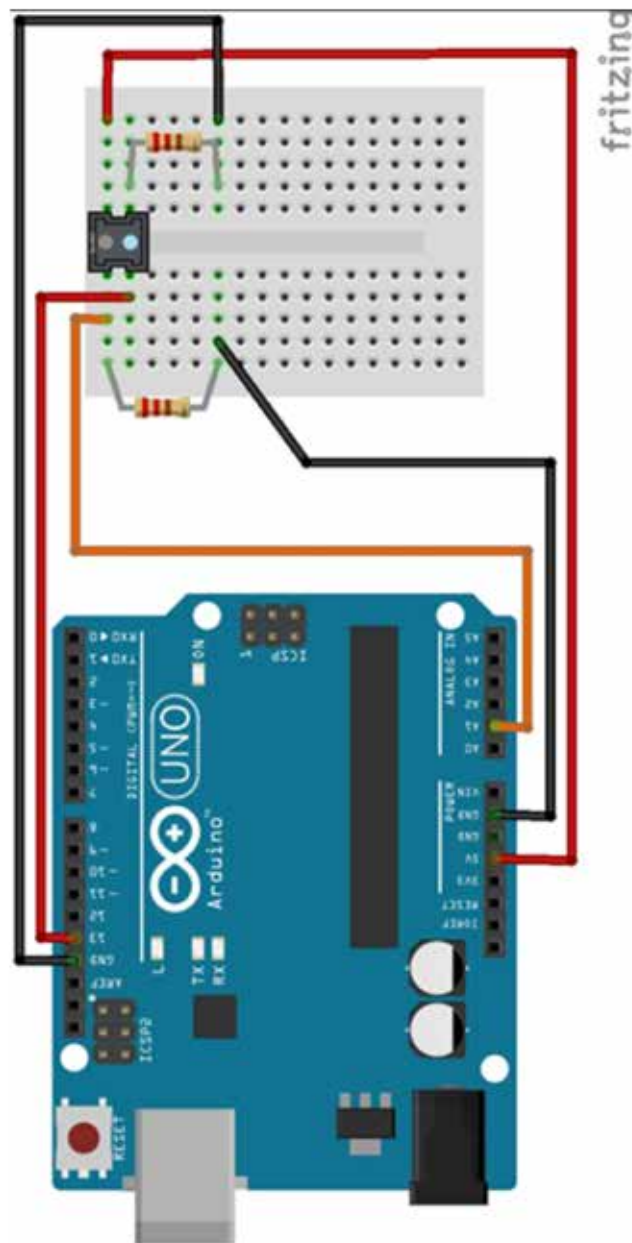
```

De esta manera, mostramos por pantalla lo detectado por el sensor infrarrojo. Podemos generar un código en miniBloq que haga lo mismo de la siguiente forma:



*Fuente: elaboración propia

Por último, el circuito completo para testear un sensor CNY70 quedará como se muestra en la siguiente imagen:



*Fuente: elaboración propia



Actividad

Armar un circuito con su programa correspondiente que permita testear dos sensores CNY70 al mismo tiempo.

Transversalidad

Desde la materia Física, se puede hablar del espectro electromagnético y hacer especial hincapié en el espectro visible.

Para investigar

¿Por qué no podemos ver la luz que emite el sensor CNY70 a simple vista? ¿Y por qué sí podemos verla a través de la cámara de un celular? ¿Cómo funciona un *encoder*? ¿Por qué es común utilizar sensores CNY70 para su implementación?

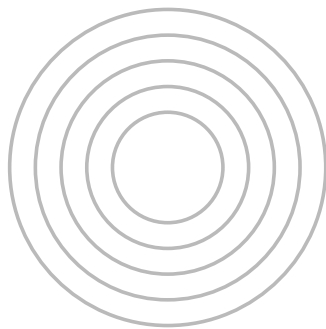
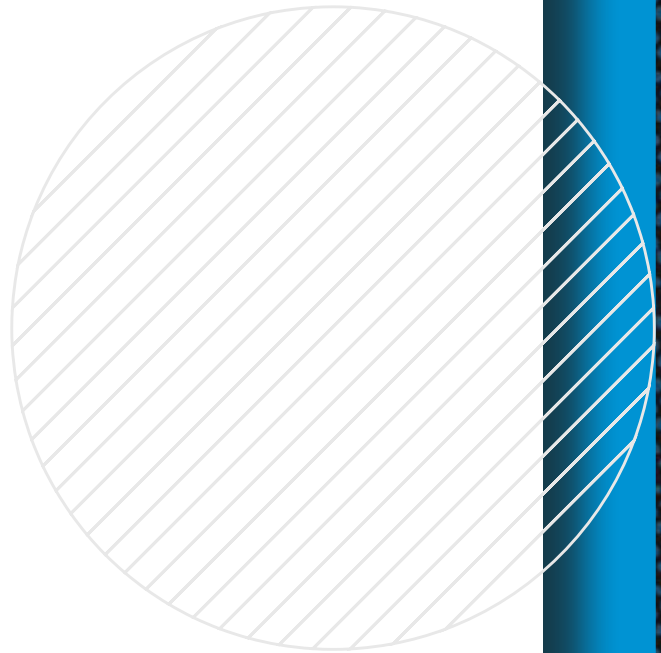
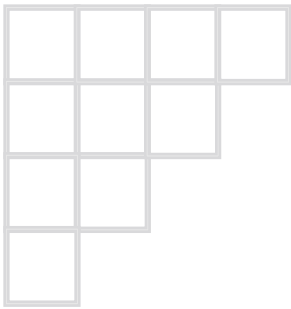
Definición

“Los sensores activos emiten energía al medio ambiente y luego miden la reacción ambiental” (Siegwart, 2004).

Resumen

En esta actividad introducimos el uso de un sensor activo. En este caso, fue un sensor que emite un haz de luz infrarrojo y luego mide la intensidad del rebote. Al igual que hicimos en la actividad anterior, las

mediciones las mostramos por pantalla y utilizamos la comunicación serial entre computadora y placa controladora a través de cables USB.



CONEXIÓN Y PROGRAMACIÓN DE UN SENSOR ULTRASÓNICO

Objetivos

- Comprender el funcionamiento de un sensor activo cuyo uso demanda un programa más complejo.
- Utilizar funciones alternativas en el uso de sensores activos.

Competencias de robots para estudiantes

Desde hace varios años, se volvieron comunes las competencias de robótica para estudiantes. En estas suele haber varios formatos en los que se puede participar: presentar un desarrollo propio en un stand —como pasa en las ferias de ciencias—, competir contra el robot de otros estudiantes —como pasa en las competencias de seguidores de línea— o, incluso, se pueden encontrar situaciones de colaboración entre estudiantes que no se conocen antes de la competencia para

competir contra otro equipo de desconocidos. A este último tipo de competencia se le suele llamar prueba colaborativa.

Para todos estos casos, resulta útil saber cómo agregar un sensor que pueda medir distancia, que sea fácil de conseguir y que no sea difícil de cablear. El kit Mulita posee un sensor HC-SR04 que nos va a permitir saber qué tan lejos se encuentra un objeto con buena precisión. A este sensor, que a veces lo llaman “sonar”, emite un sonido —que, por su frecuencia, no podemos escuchar— y luego recibe “el rebote” de la emisión que realizó. A partir de nuestro programa, podemos pedirle que calcule el tiempo que transcurrió entre la emisión y la recepción y luego convertir la medición en un valor numérico que representa una distancia en centímetros. Al igual que el sensor CNY70, este también es activo, pero puede detectar objetos a mayor distancia, lo que lo hace especialmente útil para competencias de robótica.



ACTIVIDAD 6

Uso de un sensor ultrasónico

Vamos a utilizar el sensor HC-SR04 para medir distancias en centímetros. En esta actividad, vamos a conectar el sonar directamente a la placa Arduino UNO, pero la programación será un poco más compleja que en los ejemplos anteriores.

Partes utilizadas:

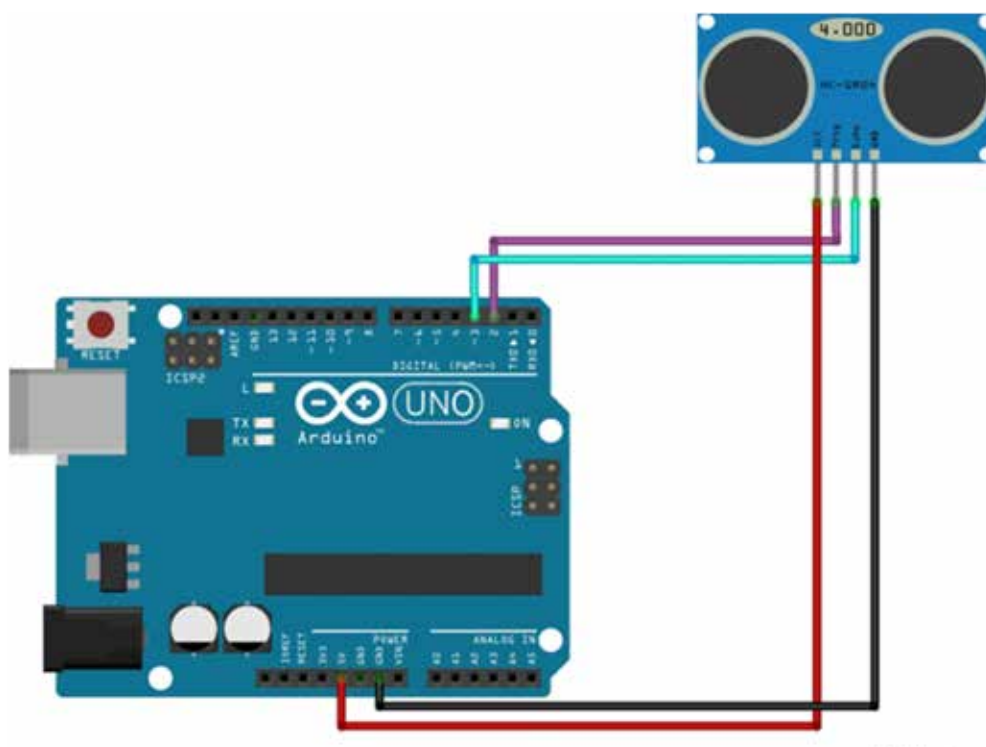
- Placa Arduino UNO
- Sensor HCSR-04
- Cables de un hilo macho-hembra



*Fuente: elaboración propia

Para la conexión entre el sensor y la placa, tenemos que usar los cuatro pines que posee el HC-SR04. Mulita utiliza cables macho-hembra para este cableado,

pero se podrían usar cables macho-macho y un *mini protoboard* en caso de querer utilizar el sensor sin necesidad de usar el robot. De los cuatro pines, dos son para positivo y negativo, mientras que los dos centrales, llamados Echo y Trig, son los encargados de emitir y recibir respectivamente. Mulita usa el pin digital 2 para Trig y el 3 para Echo. Si bien se pueden utilizar otros pines, en nuestro ejemplo programaremos utilizando estos.



*Fuente: elaboración propia

En cuanto a la programación, a modo de testeo, simplemente mostramos por pantalla qué tan lejos está un objeto de nuestro sensor, al cual ubicaremos en la parte frontal de Mulita. A diferencia de lo que hicimos

con otros sensores, en este caso tenemos que pedirle al sensor que “emita un pulso” y luego medir el tiempo que tarda ese pulso en “rebotar y volver” al sensor.

```
int Trig = 2;
int Echo = 3;
int distancia = 0;
int duracion = 0;

void setup()
{
  pinMode(Trig, OUTPUT);
  pinMode(Echo, INPUT);
  Serial.begin(9600);
}

void loop()
{
  digitalWrite(Trig, LOW);
  delay(2);
  digitalWrite(Trig, HIGH);
  delay(10);
  digitalWrite(Trig, LOW);

  duracion = pulseIn(Echo, HIGH);
  distancia = duracion / 58.2;

  Serial.println(distancia);
  delay(50);
}
```



Para darle más claridad al código, lo primero que haremos será declarar algunas variables por fuera de las funciones `setup()` y `loop()`.

```
int Trig = 2;  
int Echo = 3;  
int distancia = 0;  
int duracion = 0;
```

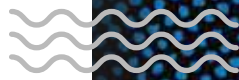
Usamos las variables Trig y Echo para determinar los pines a los que conectamos el sensor, mientras que las variables "distancia" y "duracion" nos sirven para guardar valores con los que trabajamos en la función `loop()`.

```
pinMode(Trig, OUTPUT);  
pinMode(Echo, INPUT);  
Serial.begin(9600);
```

Dentro de la función `setup()`, configuramos tres cosas:

- El pin Trig será una salida.
- El pin Echo será una entrada.
- La velocidad de transferencia será de 9600 baudios.

```
digitalWrite(Trig, LOW);  
delay(2);  
digitalWrite(Trig, HIGH);  
delay(10);  
digitalWrite(Trig, LOW);
```



Dentro de la función `loop()`, tenemos la parte más importante del código. En primer lugar, “apagamos” el pin al que tenemos conectado el Trig del sensor. Recordemos que este pin será el encargado de emitir el pulso ultrasónico. Luego “encendemos” este mismo pin por 10 milisegundos y lo volvemos a apagar. De esta manera, emitimos un pulso con la duración que establecimos en la función `delay()`. Hasta acá, solamente hemos generado una señal que no podemos escuchar debido a la frecuencia que utiliza el sensor HC-SR04.

```
duracion = pulseIn(Echo, HIGH);
```

En esta línea de código, estamos haciendo dos cosas:

- medir el tiempo que tardó en volver el pulso (HIGH) al pin Echo mediante la función `pulseIn()`, y
- guardar el valor detectado por `pulseIn()` en la variable “duracion”.

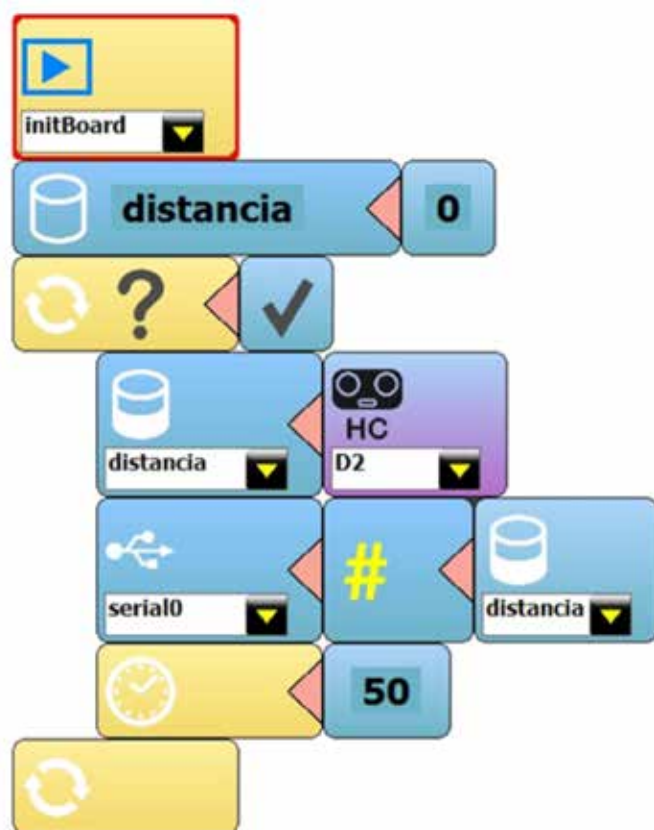
```
distancia = duracion / 58.2;
```

A continuación, dividimos por 58.2 el valor almacenado en la variable “duracion” y le asignamos el resultado a la variable “distancia”. Esta cuenta es necesaria para convertir un valor que representa tiempo en uno que representa distancia. El número 58.2 lo obtenemos a partir de la velocidad del sonido.


```
Serial.println(distancia);  
delay(50);
```

Por último, mostramos por la pantalla del monitor serial el valor almacenado en la variable "distancia" y agregamos una pequeña espera.

El código en miniBloq para testear el sensor HCSR-04 es muy distinto, ya que no tenemos que encargarnos de emitir pulsos ni de hacer conversiones.



*Fuente: elaboración propia

La idea de usar programas como miniBloq es facilitar la tarea del programador que, en estos casos, suele

ser un estudiante. Por eso se simplifica el uso de un sensor como este. Al igual que como hicimos con el sensor LDR, empezamos creando en el código una variable que llamaremos "distancia".



*Fuente: elaboración propia

Seguidamente, creamos un ciclo infinito para simular la función loop() de Arduino. Es importante tener en cuenta que, cuando usamos un bloque de una estructura de control, siempre se generan, al menos, dos bloques: uno para indicar dónde empieza el ciclo y otro para indicar dónde termina.



*Fuente: elaboración propia

Entre estos dos bloques, vamos a escribir las órdenes para que el terminal de miniBloq muestre los valores detectados por el sensor. Primero, guardamos en la variable "distancia" el valor detectado por el

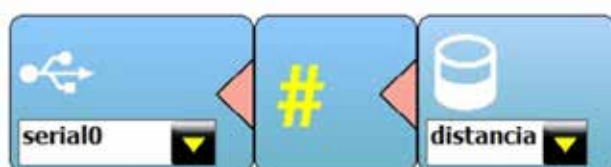


sensor HCSR-04, cuyo pin está conectado a la salida D2 de la placa.



*Fuente: elaboración propia

Luego, mostramos por pantalla el valor guardado en la variable utilizando el terminal.



*Fuente: elaboración propia

Por último, agregamos una pequeña espera para que la información se muestre claramente por pantalla.



*Fuente: elaboración propia

Este código se puede mejorar para que, antes de mostrar el valor detectado, se pueda leer un mensaje de texto como "distancia". Además, se puede agregar la abreviación de "centímetros" a continuación del valor detectado.



Actividades

- Utilizando el sensor HCSR-04, desarrollar un programa que, al detectar una distancia menor a 10 cm, muestre un mensaje de alerta por el terminal.
- Escribir otro programa que relacione lo detectado por el sensor con un actuador. Por ejemplo, se podría utilizar un servo para que gire su eje a una posición relacionada con la distancia a la que se encuentra el sensor de un determinado objeto.

Transversalidad

Desde Matemática, se puede trabajar la conversión de medidas aprovechando que el sensor ultrasónico mide tiempo, pero por pantalla se muestra la distancia en centímetros.

Para investigar

¿Qué otros sensores de distancia se pueden encontrar en el mercado? ¿Funcionan de la misma forma? ¿Qué diferencias en cuanto a precio y prestaciones hay entre esos sensores y el que usa Mulita?

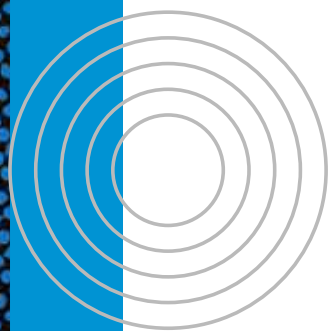
Definición

"Mientras que un sensor infrarrojo solo proporciona información de proximidad, un sensor ultrasónico

puede proporcionar información de distancia porque es posible medir el tiempo de vuelo entre el inicio de un pulso y el retorno de su eco. Al medir el tiempo de vuelo y conocer la velocidad del sonido en el aire, es posible calcular la distancia recorrida por el viaje de ida y vuelta del ping” (Jones et al., 2018).

Resumen

En esta actividad trabajamos con un sensor ultrasónico. Este tipo de entradas son muy utilizadas en robots educativos por la gran cantidad de actividades que se puede hacer con ellos. En el caso del modelo que utiliza Mulita, el HCSR-04, es importante entender su funcionamiento para lograr mayor precisión en las mediciones.



MOTORES, RUEDAS Y TRACCIÓN DE MULITA

Objetivos

- Aprender a utilizar motores paso a paso para lograr movimientos de desplazamiento precisos con nuestro robot.
- Entender cómo trabajar con actuadores y placas controladoras en espacios de Robótica Educativa.

Tracción de robots

El robot educativo Mulita posee dos motores paso a paso 28BYJ-48. Este modelo es fácil de conseguir y, al igual que el resto de la electrónica de Mulita, es de bajo costo. La elección de este motor en el desarrollo de nuestro robot permite mayor precisión en los movimientos que podrá realizar el robot por encima de la velocidad a la que se desplace. Estos motores, que se fijan al chasis, poseen un eje con dos

lados planos en su parte más extrema. Esto permite que las ruedas de Mulita se encastran sin necesidad de usar tornillos y puedan girar para desplazar al robot.



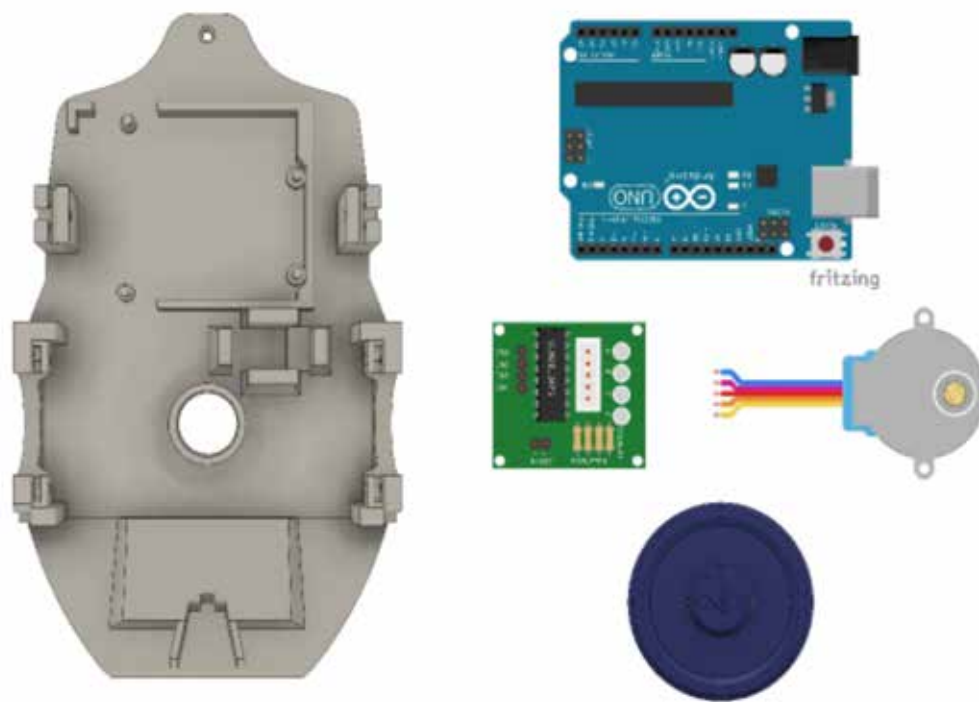
ACTIVIDAD 7

Cableado y programación de un motor paso a paso

En esta actividad vamos a conectar y programar el motor izquierdo del robot. El cableado del motor derecho será similar, pero utilizará otros pines digitales.

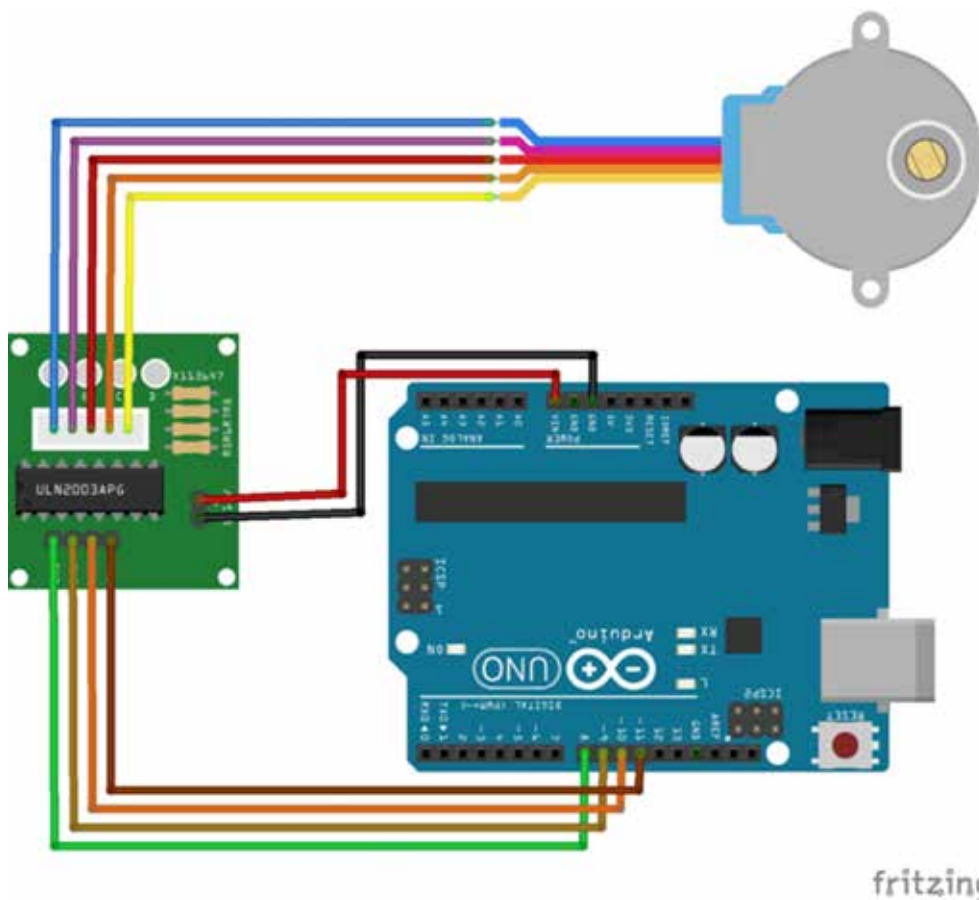
Partes utilizadas:

- Placa Arduino UNO
- Motor 28BYJ-48
- Controlador de motores con ULN2003
- Cables de un hilo macho-macho



*Fuente: elaboración propia

El cableado entre el motor y la placa es más complejo que en las actividades anteriores, ya que es necesario utilizar el controlador de motores. Esta pequeña placa posee un micro ULN2003 que requiere ser conectado a pines de alimentación. Por lo tanto, utilizaremos los pines digitales 8, 9, 10 y 11 para el controlador y, además, dos cables para la alimentación conectados a VIN y GND de la placa Arduino UNO.



*Fuente: elaboración propia

Para que el motor empiece a girar, tenemos que tener en cuenta algunas cosas:

- Cada pin digital alimentará una de las bobinas del motor. Por lo tanto, tendremos que encenderlas y apagarlas de a una o dos a la vez.
- Desde la placa Arduino UNO, podemos alimentar la placa que controla el motor. Si preferimos utilizar una fuente de alimentación externa para los motores, tendremos que unificar el cable que conectamos a GND. O sea, un cable de un

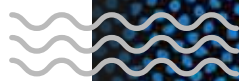
hilo unirá la conexión a tierra de la placa con un pin GND de Arduino.

- El código que escribamos para esta actividad estará dentro de la función `loop()`, ya que eso nos permitirá ver mejor qué tan rápido gira la rueda (o el eje).

En el código que escribiremos, vamos a activar dos pines digitales y apagaremos los otros dos. El orden en el que activamos los pines y los pares que elegimos definirán la dirección de la rueda. En nuestro caso, como estamos usando la rueda derecha, activamos primero los pines de los extremos (11 y 8) y luego los más altos (11 y 10). A continuación, los pines del medio (10 y 9) y, por último, los más bajos (9 y 8).

```
void setup()
{
  pinMode(11, OUTPUT); //IN4
  pinMode(10, OUTPUT); //IN3
  pinMode(9, OUTPUT); //IN2
  pinMode(8, OUTPUT); //IN1
}
```

```
void loop()
{
  digitalWrite(11, HIGH);
  digitalWrite(10, LOW);
  digitalWrite(9, LOW);
  digitalWrite(8, HIGH);
  delay(20);
}
```




```
digitalWrite(11, HIGH);  
digitalWrite(10, HIGH);  
digitalWrite(9, LOW);  
digitalWrite(8, LOW);  
delay(20);
```

```
digitalWrite(11, LOW);  
digitalWrite(10, HIGH);  
digitalWrite(9, HIGH);  
digitalWrite(8, LOW);  
delay(20);
```

```
digitalWrite(11, LOW);  
digitalWrite(10, LOW);  
digitalWrite(9, HIGH);  
digitalWrite(8, HIGH);  
delay(20);
```

```
}
```

En primer lugar, dentro de la función `setup()`, configuramos como salidas a los cuatro pines digitales que vamos a utilizar.

```
void setup()  
{  
  pinMode(11, OUTPUT); //IN4  
  pinMode(10, OUTPUT); //IN3  
  pinMode(9, OUTPUT); //IN2  
  pinMode(8, OUTPUT); //IN1  
}
```



Para empezar, dentro de la función `loop()`, encendemos dos pines y apagamos otros dos. El orden en que hacemos esta secuencia es muy importante, ya que determina cómo girará el eje del motor. Por cada bloque de cuatro líneas de código (una por cada pin), agregamos una pequeña espera.

```
digitalWrite(11, HIGH);  
digitalWrite(10, LOW);  
digitalWrite(9, LOW);  
digitalWrite(8, HIGH);  
delay(20);
```

Los siguientes bloques de código encenderán de a dos los pines conectados al controlador de motores.

```
digitalWrite(11, HIGH);  
digitalWrite(10, HIGH);  
digitalWrite(9, LOW);  
digitalWrite(8, LOW);  
delay(20);
```

```
digitalWrite(11, LOW);  
digitalWrite(10, HIGH);  
digitalWrite(9, HIGH);  
digitalWrite(8, LOW);  
delay(20);
```

```
digitalWrite(11, LOW);  
digitalWrite(10, LOW);  
digitalWrite(9, HIGH);  
digitalWrite(8, HIGH);  
delay(20);
```


En el caso de querer programar un motor paso a paso utilizando miniBloq, podemos usar la misma secuencia de pasos que escribimos en Arduino, pero esta vez no hará falta inicializar los pines digitales como salida.



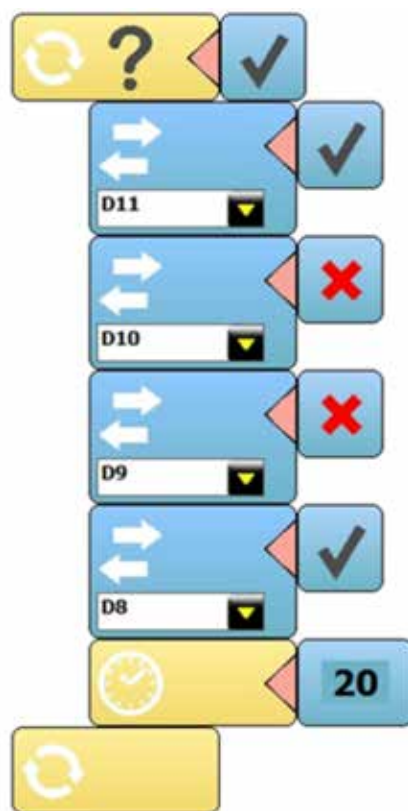
*Fuente: elaboración propia

En primer lugar, generamos un ciclo infinito que simula la función `loop()`. Los bloques que encienden y apagan los pines digitales estarán encerrados dentro de este ciclo.



*Fuente: elaboración propia

Luego, agregamos cuatro bloques de pines digitales y uno de tiempo de espera. Los configuramos con los valores D11, D10, D9 y D8 y la secuencia "encendido - apagado - apagado - encendido".



*Fuente: elaboración propia

Luego, repetimos el patrón de cuatro bloques de salidas digitales con un tiempo de espera de manera tal que encendamos y apaguemos en orden dos pines a la vez.

Actividades

- Modificar el código de esta actividad para que el motor avance a mayor velocidad.
- Modificar el código para que el motor gire en sentido contrario al que avanza en esta actividad.
- Utilizando los pines digitales 4, 5, 6 y 7, hacer andar al otro motor paso a paso.
- Con los dos motores funcionando, hacer que Mulita avance en línea recta.

Transversalidad

Desde el espacio de Física, se puede trabajar con motores eléctricos: explicar su funcionamiento, identificar sus partes, investigar las diferencias que hay entre modelos y sus posibles aplicaciones.

A partir de estudiar cómo se interrelacionan las partes de un motor de corriente continua, se puede aumentar la complejidad al mostrar el lugar que estos

ocupan dentro de un servo. Además, también se puede hablar de motores eléctricos brushless, como los utilizados en drones, para luego aumentar la complejidad de las explicaciones hasta llegar a los motores paso a paso.

Para investigar

¿Qué diferencia hay entre motores de corriente continua, servomotores y motores paso a paso? ¿Por qué los drones suelen utilizar motores “sin escobillas”?

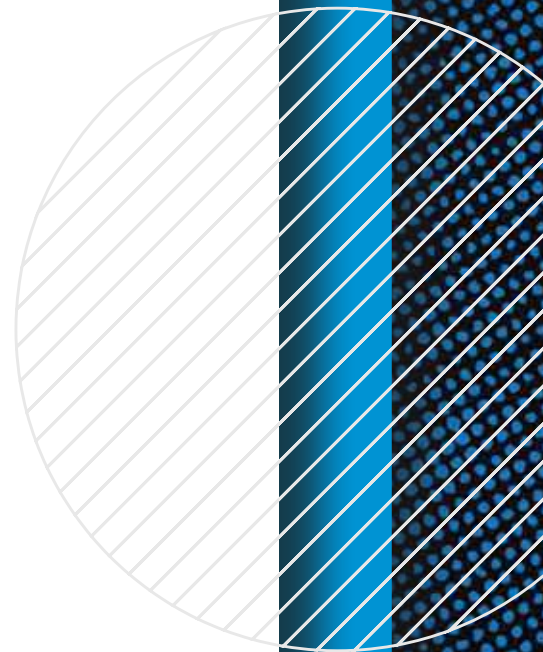
En cuanto a los motores paso a paso como los que utiliza Mulita, ¿qué ventajas tienen que justifican su uso en estos robots? ¿En qué casos convendría reemplazarlos por motores comunes? ¿Cómo se podría mejorar el funcionamiento de los motores desde el código?

Definición

“Tanto desde el punto de vista de la programación como de la construcción, la tracción diferencial puede ser uno de los sistemas de locomoción menos complicados. [...] El esquema diferencial consiste en dos ruedas sobre un eje común, cada rueda accionada independientemente. Tal disposición le da al robot la capacidad de conducir en línea recta, girar en su lugar y moverse en un arco” (Jones et al., 2018).

Resumen

En esta actividad hicimos andar un motor paso a paso como los que usa Mulita. A partir de este punto, podemos agregar otro motor y programarlo por separado. La elección de este tipo de motores en un robot educativo ayuda a que podamos realizar movimientos con mucha precisión y priorizar esta característica sobre la velocidad a la que se desplaza el robot.



MULITA VIAJERA

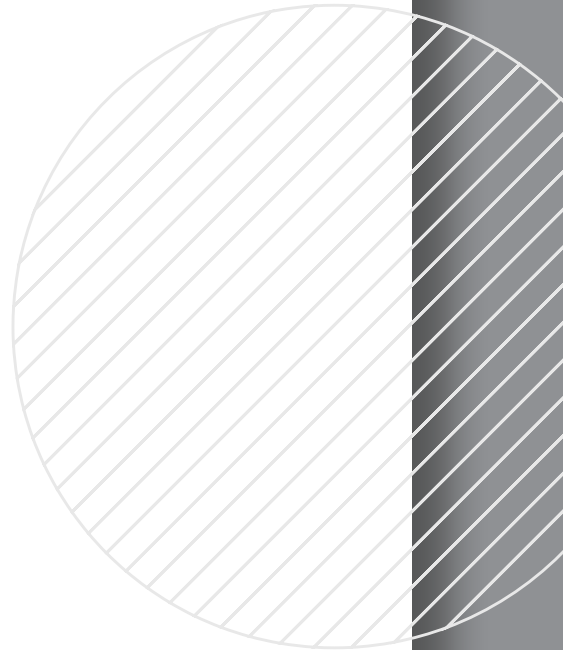
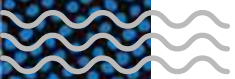
Objetivos

- Conocer alternativas sencillas y transversales para trabajar con Mulita en el aula.
- Realizar actividades que puedan utilizarse con grupos de estudiantes con menor conocimiento sobre tecnología.
- Incorporar el uso de pseudocódigo para documentar los programas que queremos realizar.

Programas para aprender a programar

En los últimos años, se han desarrollado muchos programas educativos cuya finalidad es hacer más fácil el proceso de aprender a programar. La principal característica de una gran cantidad de estos es que el código que realizan los estudiantes se forma apilando bloques, como en el caso de miniBloq. Si bien con Mulita se puede utilizar cualquier entorno

de programación que sea compatible con las placas Arduino, en miniBloq encontramos bloques específicos para trabajar con nuestro robot.



ACTIVIDAD 8

Alternativas para trabajar con Mulita en el aula

Para esta actividad, vamos a utilizar a Mulita completamente cableada. Es importante que estén funcionando tanto los motores y el servo como el sensor frontal. Si alguno de estos componentes falla, se lo puede testear realizando alguna de las actividades anteriores en las que se trabajó con cada parte por separado.

Partes utilizadas:

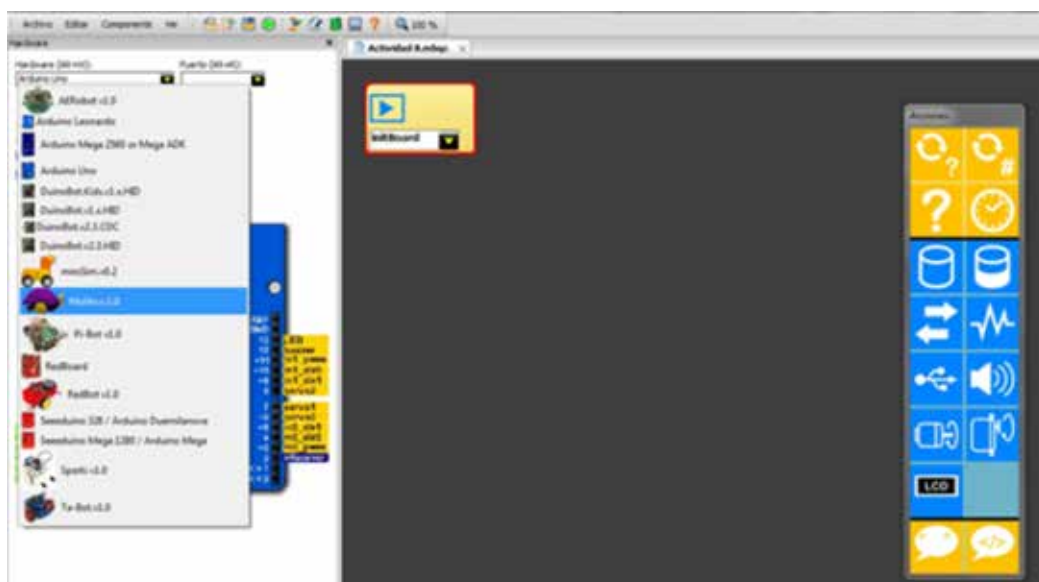
- Mulita completa.



*Fuente: elaboración propia

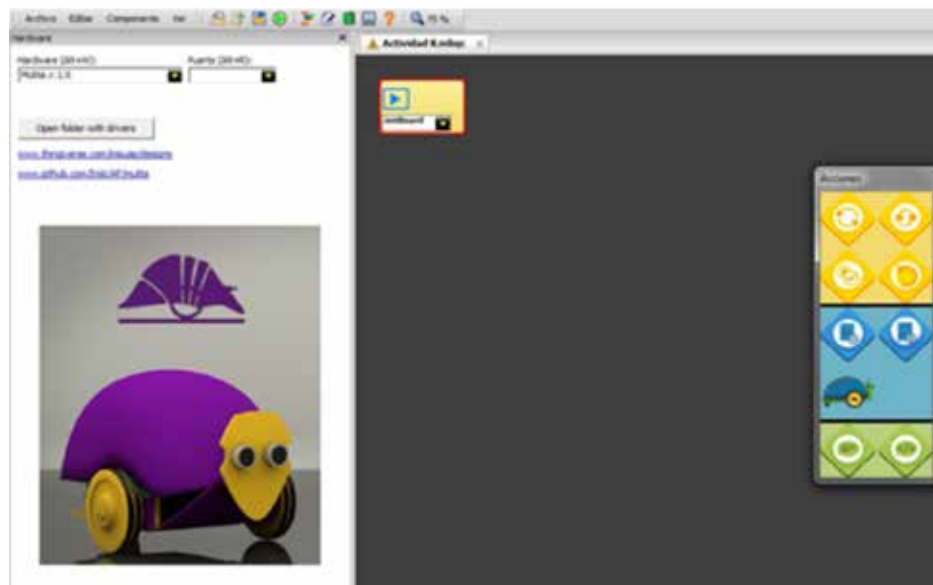
Actividades

Lo primero que debemos hacer es seleccionar a Mulita como el *hardware* a utilizar por miniBloq. Eso se hace desde el menú desplegable "hardware", tal como se muestra en la siguiente imagen:



*Fuente: elaboración propia

Vamos a notar que cambiaron dos cosas: la primera es que apareció Mulita en la pantalla en el sector del "hardware". La segunda es que la lista de bloques que podemos utilizar cambió.



*Fuente: elaboración propia

Los bloques disponibles pueden ser clasificados en tres grupos:

- **Bloques naranjas:** proveen estructuras de control y la posibilidad de pedirle al programa que deje de ejecutarse por un determinado tiempo.
- **Bloques celestes:** sirven para manipular variables y, además, tienen los bloques específicos de Mulita.
- **Bloques verdes:** permiten agregar comentarios o código externo.



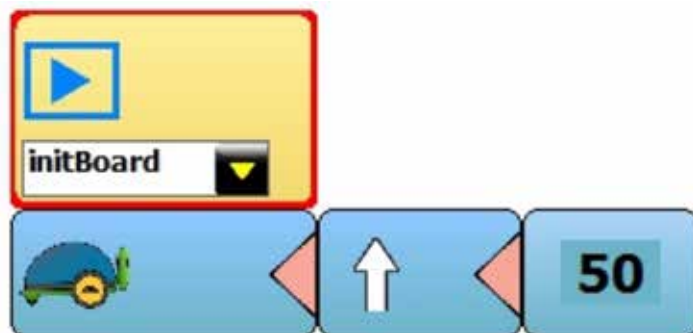
*Fuente: elaboración propia

Otra particularidad de los bloques para programar a Mulita es que, si seleccionamos la opción Mulita, tendremos un menú desplegable con más opciones. Y, al elegir cualquier opción de las disponibles, se podrán presentar nuevas dependiendo del bloque elegido.



*Fuente: elaboración propia

Si quisiéramos que Mulita avance en línea recta medio metro (50 centímetros), podríamos crear el siguiente código:



*Fuente: elaboración propia

¿Y si quisiéramos que la trayectoria de Mulita fuera una figura geométrica? Podríamos, por ejemplo, dibujar un cuadrado. Pero, a diferencia de cómo abordamos las actividades anteriores, esta vez vamos a escribir primero el programa en pseudocódigo. O sea, vamos a redactar con nuestras palabras qué nos gustaría que hiciera el robot y luego vamos a utilizar miniBloq para decirle que lo haga. Por lo tanto, nuestro pseudocódigo para que Mulita haga una trayectoria con forma de cuadrado podría ser el siguiente:

Avanzar
 Girar 90 grados
 Avanzar
 Girar 90 grados
 Avanzar
 Girar 90 grados
 Avanzar



Si quisiéramos traducir esto a miniBloq, podríamos armar el siguiente programa:



*Fuente: elaboración propia

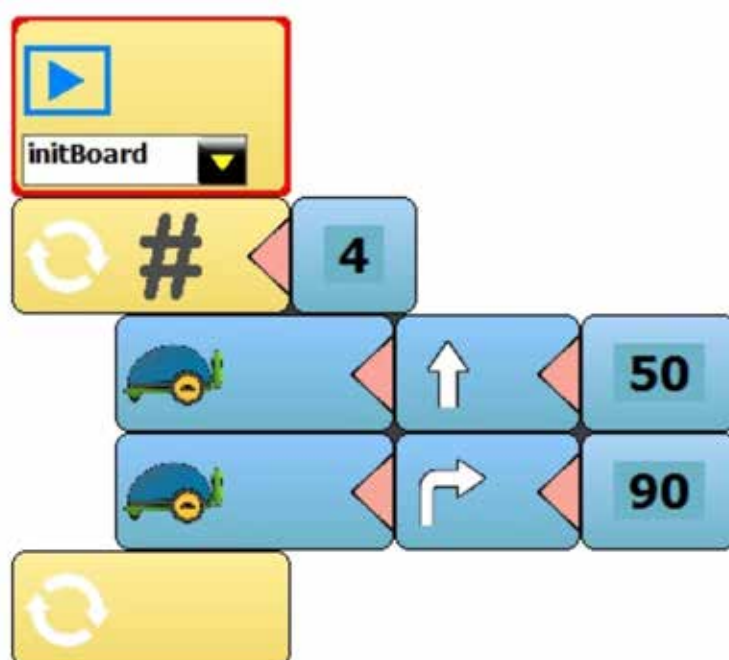
En este cuadrado utilizamos dos órdenes: avanzar 50 centímetros y girar 90 grados. Este código se podría simplificar mucho al utilizar una estructura de control que repitiera órdenes las veces que queramos. Si utilizamos el bloque "repetir", podemos pedirle eso. El pseudocódigo para este programa podría escribirse de la siguiente manera:

Repetir 4 veces:

Avanzar 50 centímetros

Girar 90 grados

Luego, el código en miniBloq correspondiente al pseudocódigo quedaría de la siguiente manera:



*Fuente: elaboración propia

Si bien este código hace que Mulita realice una trayectoria en forma de cuadrado, podríamos hacer que esta quede dibujada. Utilizando un marcador conectado al sistema del servo de Mulita, vamos a hacer que dibuje la trayectoria mientras la realiza. Para eso, podemos armar el siguiente código:

Bajar fibrón

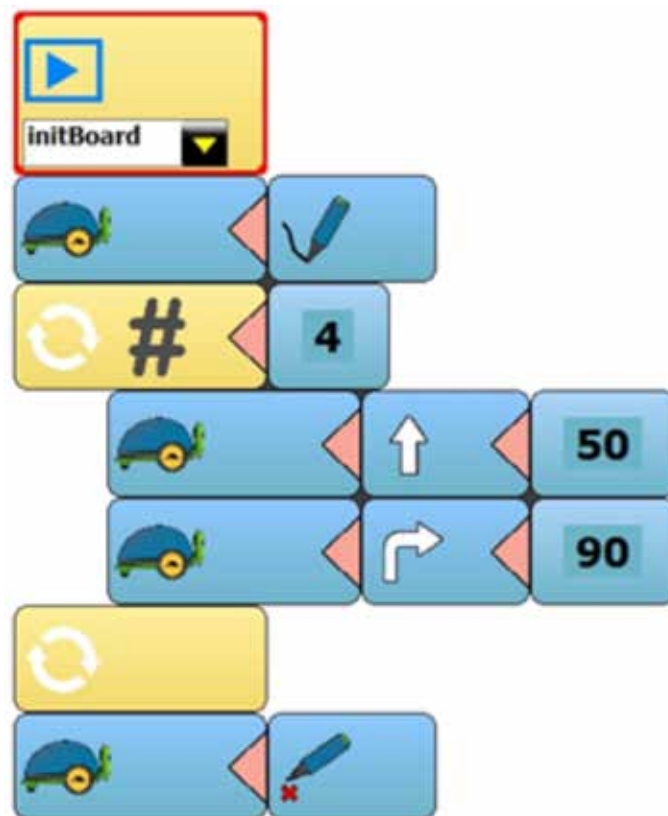
Repetir 4 veces:

Avanzar 50 centímetros

Girar 90 grados

Subir fibrón

A diferencia del código anterior, ahora agregamos bloques para que Mulita pinte (que el servo desplace el fibrón hacia abajo) y para que deje de pintar cuando termine el recorrido (que el servo desplace el fibrón hacia arriba). Es importante notar que los bloques que se utilizan para los servos están fuera del ciclo "repetir", ya que queremos que se ejecuten una sola vez.



*Fuente: elaboración propia

Por último, vamos a hacer que Mulita avance en línea recta hasta detectar un obstáculo y que, cuando lo encuentre, gire para esquivarlo y luego siga avanzando. Para este código, de vuelta, lo primero que hacemos es escribirlo en pseudocódigo. Debemos tener en cuenta que este comportamiento por parte del robot demandará que armemos un ciclo que no termine y que, además, tendremos que elegir una estructura de control para que el robot decida qué trayectoria realizar en función de lo que detecta el sensor.

Por siempre

Si el sensor detecta algo a menos de 10 cm

Girar 45 grados

Si no

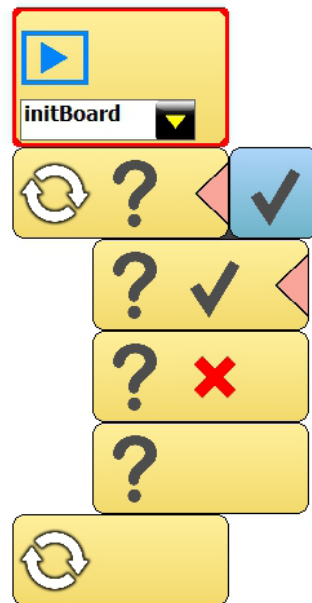
Avanzar en línea recta

Para el ciclo infinito ("por siempre" en el pseudocódigo), utilizamos la siguiente combinación de bloques:



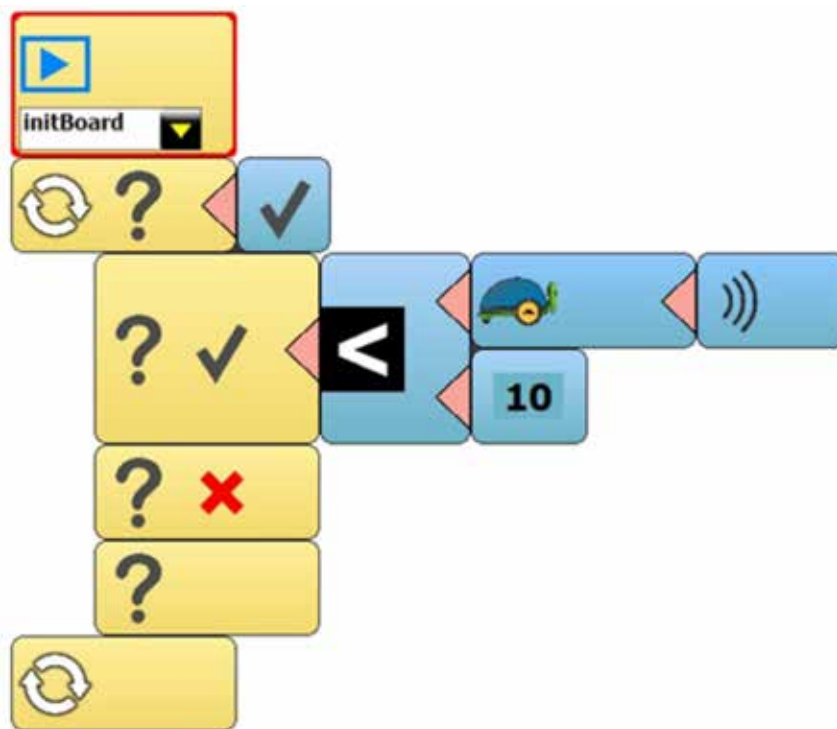
*Fuente: elaboración propia

Todo lo que realice el robot tendrá que estar encerrado entre el comienzo y el final de este ciclo “while”. Luego, podemos agregar una nueva estructura de control que permita decidir entre dos opciones. Estas dependerán de lo detectado por el sensor frontal.



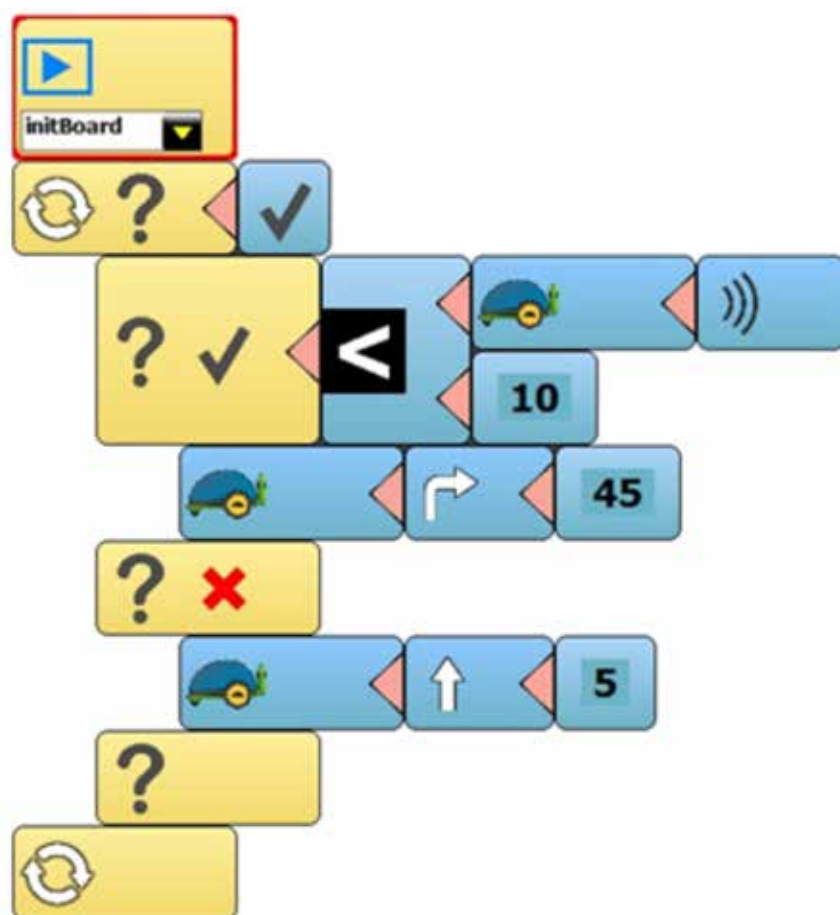
*Fuente: elaboración propia

En nuestro caso, la condición que determinará si el robot avanza o gira será la detección del sensor. Por lo tanto, tenemos que comparar lo detectado por el sensor con un valor de referencia que elijamos, por ejemplo, 10 (en este caso representa 10 centímetros).



*Fuente: elaboración propia

Ahora, con la lógica de decisión lista, pasamos a escribir qué tiene que hacer el robot en cada caso: girar si la distancia a un objeto es menor a 10 cm o avanzar si la distancia es mayor o igual.



*Fuente: elaboración propia

Con este código, el robot debería censar lo que pasa frente a él y avanzar lentamente mientras no encuentre obstáculos.

Transversalidad

En la primera parte de la actividad se trabajó con figuras geométricas. Este tema se puede abordar desde Matemática y reflexionar sobre la relación entre los lados de una figura y los ángulos de esta. Sin embargo, hay que tener en cuenta la diferencia entre

ángulo interno y ángulo externo, ya que Mulita usará el segundo para que la figura quede bien dibujada. Para esto se puede sugerir dibujar más formas como se propone a continuación.

Actividades

- Modificar el código del cuadrado para que dibuje las siguientes figuras geométricas:
 - Triángulo
 - Pentágono
 - Octógono
- Ajustar el código del robot que avanza hasta detectar un obstáculo para que el robot avance más rápido.
- Agregarle al código anterior la posibilidad de marcar en el piso la trayectoria que realiza el robot.

Para investigar

Investigar la sentencia "if" utilizada en lenguajes de programación como C/C++ y Java. Este es el equivalente al bloque que utilizamos en miniBloq para comparar el valor detectado por el sensor frontal con la distancia a la que pretendemos detectar un objeto

antes de girar. Esta sentencia se puede utilizar sola o asociada a la palabra reservada "else".

Definición

El pseudocódigo es la secuencia de órdenes escritas en un lenguaje natural expresadas de manera similar a un algoritmo computacional.

Resumen

En esta actividad programamos a Mulita con mini-Bloq. A diferencia de las actividades anteriores en las que programamos por separado cada componente, esta vez utilizamos el robot completo; comenzamos con un código sencillo y luego usamos uno más complejo.



TRABAJO FINAL

Para cerrar este manual, proponemos crear una actividad con Mulita que involucre aspectos técnicos y pedagógicos. Al pensar esta actividad, debemos tener en cuenta cuatro etapas del desarrollo de un proyecto que consideramos importantes:

- Identificación del problema a resolver
- Diseñar una solución
- Implementar el diseño
- Testear el proyecto

Cada una de estas cuatro etapas nos permitirá generar documentación que podremos utilizar al final del proyecto para presentarlo. Es recomendable almacenar cada documento generado (imágenes, pseudocódigo, esquemas, bocetos, etc.) en una carpeta compartida en la red.

Identificación del problema a resolver

Lo primero que debemos hacer antes de empezar a armar el proyecto es identificar claramente cuál es el problema que queremos resolver. En este caso, vamos a armar una actividad completa que se pueda realizar con Mulita y cuyo código demande programar, al menos, un actuador y un sensor. Es importante que tengamos en cuenta cuáles son los componentes de los que disponemos y qué queremos enseñar con ellos. Las preguntas del siguiente cuestionario deberían guiarnos en la primera etapa del proyecto.

Electrónica:

- ¿Qué componentes tengo a disposición?
- ¿Voy a usar un solo actuador o más de uno?
- ¿Qué sensor o sensores voy a utilizar?

Programación:

- ¿Qué tan complejo quiero que sea el programa que voy a realizar?
- ¿Qué estructuras de control me convendrá usar para dicho programa?

Objetivos:

- ¿Qué contenido busco enseñar? ¿Es específico de una materia tecnológica o es transversal?

- ¿A quiénes está dirigida la actividad?
- ¿Qué conocimientos poseen los estudiantes que lo van a realizar?

Con estos puntos identificados, podemos empezar a bocetar nuestro robot, realizar un esquema sencillo de los componentes e, incluso, comenzar a armar el programa en pseudocódigo. Debemos digitalizar los esquemas que generemos a partir de estas preguntas y guardarlos para presentarlos al final del trabajo como parte de la documentación del proyecto.

Diseñar una solución

Con el problema identificado y teniendo clara la dirección que queremos seguir, podemos empezar con el diseño formal. En este punto, redactamos en un archivo de texto los aspectos pedagógicos del trabajo (objetivos, a quién va dirigido, conceptos a enseñar). Además, documentamos la electrónica que vamos a utilizar y sus conexiones a partir de esquemas gráficos. También relacionamos dichos esquemas con el pseudocódigo asociado. Este será la base del programa que implementaremos a continuación.



Implementar el diseño

En la etapa de implementación es cuando ponemos manos a la obra. En el orden que nos resulte más cómodo, armamos el circuito con los componentes que elegimos, basado en el esquema que realizamos previamente, y escribimos el código del programa.

Podemos guardar el código que generamos en la carpeta de documentación. También podemos fotografiar al robot para tener un registro del trabajo realizado con él.

Testear el proyecto

En la etapa de testeo ponemos a prueba cada parte del proyecto: primero, el funcionamiento del robot y, segundo, el cumplimiento de los objetivos pedagógicos. Lo más importante de esta etapa es que trabajemos a partir de la observación y que, si el robot no se comporta como pretendíamos o no se están cumpliendo los objetivos propuestos, revisemos la documentación de las etapas anteriores para realizar las modificaciones necesarias. A partir de esta etapa, comienza un ciclo que hará que revisemos el diseño o la implementación, que modifiquemos lo que nos parezca necesario y que volvamos a testear todo el

conjunto. La etapa de testeo finalizará cuando el robot se comporte como pretendíamos que lo hiciera.

Una vez terminado el proyecto, podemos compartirlo mediante una presentación en la que mostremos el proceso creativo por el que pasamos y la documentación que se ha generado.

BIBLIOGRAFÍA

Bergren, C. (2003). *Anatomy of a robot*. McGraw Hill.

Dougherty, D. (2021). The maker movement [Internet]. mitpressjournals.org/doi/pdf/10.1162/INOV_a_00135

Downey, A., Elkner, J., y Meyers, C. (2002). *Aprenda a pensar como un programador con Python*. Green Tea Press.

Jones, J., Flynn, A., y Sieger, B. (2018). *Mobile robots inspiration to implementation*. CRC Press.

Nieto, M. (2016). La nueva tendencia tecnológica se llama “movimiento maker” [Internet]. *El País*. https://elpais.com/tecnologia/2016/09/22/actualidad/1474560262_851948.html

Stallman, R. (2022). *Por qué el “código abierto” pierde de vista lo esencial del software libre* [Internet]. El sistema operativo GNU. <https://www.gnu.org/philosophy/open-source-misses-the-point.es.html>

Siegwart, R. (2004). *Introduction to autonomous mobile robots*. Massachusetts Institute of Technology.

Tanenbaum, A. (1986). *Organización de computadoras: un enfoque estructurado*. Prentice Hall.