# PROYECTO FINAL - CARRO MÓVIL CON ARDUINO


**PRESENTADO POR:**

**LUIS MATEO ORTEGA GOYES**

**MANUEL ALEJANDRO MACIAS SILVA**

**PRESENTADO A:**

**CARLOS HERNAN TOBAR ARTEAGA**

**UNIVERSIDAD DEL CAUCA**

**FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES**

**TECNOLOGÍA EN TELEMÁTICA**

**POPAYÁN - NOVIEMBRE - 2023**

# I. INTRODUCCIÓN

En esta práctica, se realizaron dos prototipos de carros con funcionalidades distintas, pero en ambos casos se utilizó Arduino. Para el primer prototipo, se creó un carrito a control remoto utilizando Arduino Uno, que se controló a través de Bluetooth. El dispositivo Bluetooth utilizado fue el HC-05, y el carrito se controló con un teléfono celular Android, descargando la aplicación desde la Play Store.

Para el segundo prototipo, el objetivo era crear un robot que avanzara en línea recta mientras no detectara la presencia de un obstáculo. En caso de detectar algún obstáculo en su camino, el robot giraba sobre sí mismo hasta esquivar el obstáculo y poder seguir su camino. El sensor HC-SR04 permitió percibir la presencia de un obstáculo y, con el control de los motores de corriente continua, se pudo accionar el robot para avanzar y girar ante los obstáculos.

# II. MARCO TEÓRICO

## ¿Qué es Arduino?

Arduino es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para los creadores y desarrolladores. Esta plataforma permite crear diferentes tipos de microordenadores de una sola placa a los que la comunidad de creadores puede darles diferentes tipos de uso.

Para poder entender este concepto, primero vas a tener que entender los conceptos de hardware libre y el software libre. El hardware libre son los dispositivos cuyas especificaciones y diagramas son de acceso público, de manera que cualquiera puede replicarlos. Esto quiere decir que Arduino ofrece las bases para que cualquier otra persona o empresa pueda crear sus propias placas, pudiendo ser diferentes entre ellas, pero igualmente funcionales a partir de la misma base.

El software libre son los programas informáticos cuyo código es accesible por cualquiera para que quien quiera pueda utilizarlo y modificarlo. Arduino ofrece la plataforma Arduino IDE (Entorno de Desarrollo Integrado), que es un entorno de programación con el que cualquiera puede crear aplicaciones para las placas Arduino, de manera que se les puede dar todo tipo de utilidades.

El proyecto nació en 2003, cuando varios estudiantes del Instituto de Diseño Interactivo de Ivrea, Italia, con el fin de facilitar el acceso y uso de la electrónica y programación. Lo hicieron para que los estudiantes de electrónica tuviesen una alternativa más económica a las populares BASIC Stamp, unas placas que por aquel entonces valían más de cien dólares, y que no todos lo podían comprar.

El resultado fue Arduino, una placa con todos los elementos necesarios para conectar periféricos a las entradas y salidas de un microcontrolador, y que puede ser programada tanto en Windows como macOS y GNU/Linux. Un proyecto que promueve la filosofía 'learning by doing', que viene a querer decir que la mejor manera de aprender es cacharreando.

## ¿Cómo funciona el Arduino?

Las funciones de Arduino, como ocurre con la mayoría de las placas de microcontroladores, se pueden resumir en 3 factores:

Cuenta con una interfaz de entrada. Esta puede estar directamente unida a los periféricos, o conectarse a ellos a través de puertos.

La interfaz de entrada tiene como objetivo trasladar la información al microcontrolador. El microcontrolador es la pieza que se encarga de procesar esos datos. Además, varía dependiendo de las necesidades del proyecto en el que se desee usar la placa, y existe una gran variedad de fabricantes y versiones disponibles.

También cuenta con interfaz de salida. Este se encarga de llevar la información procesada a los periféricos autorizados de hacer el uso final de esos datos. En algunos casos puede tratarse de otra placa en la que se centraliza y procesa la información de forma totalmente renovada, o sencillamente, puede ser una pantalla o un altavoz encargado de mostrar la versión final de los datos.

Lenguaje de la programación con Arduino: C++

¿Qué lenguaje utiliza este tipo de programación? La plataforma Arduino se programa con un lenguaje propio basado en el lenguaje de programación de alto nivel Processing, lo que significa que es similar a C++.

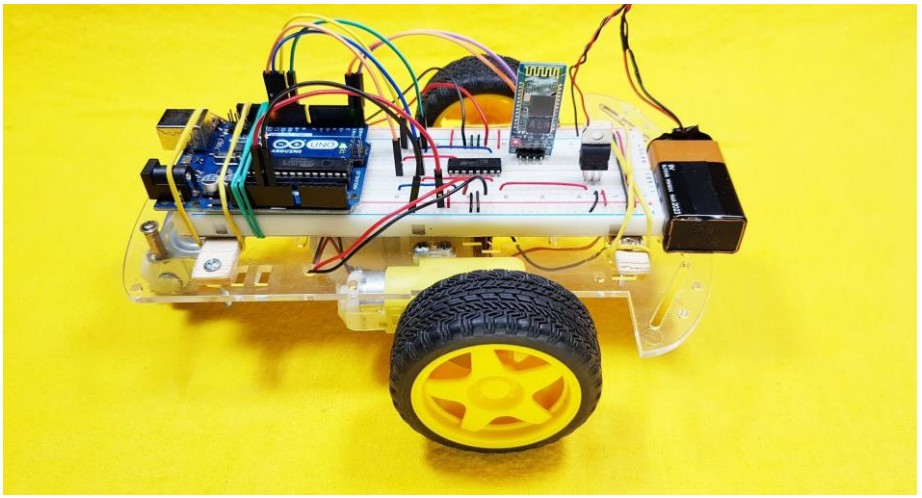
¿Qué quiere decir esto? Que se trata de un lenguaje de programación de propósito general asociado a un sistema operativo llamado UNIX.

Este lenguaje de medio nivel, trata con objetos básicos como caracteres, números, bits y direcciones de memoria, entre otros.

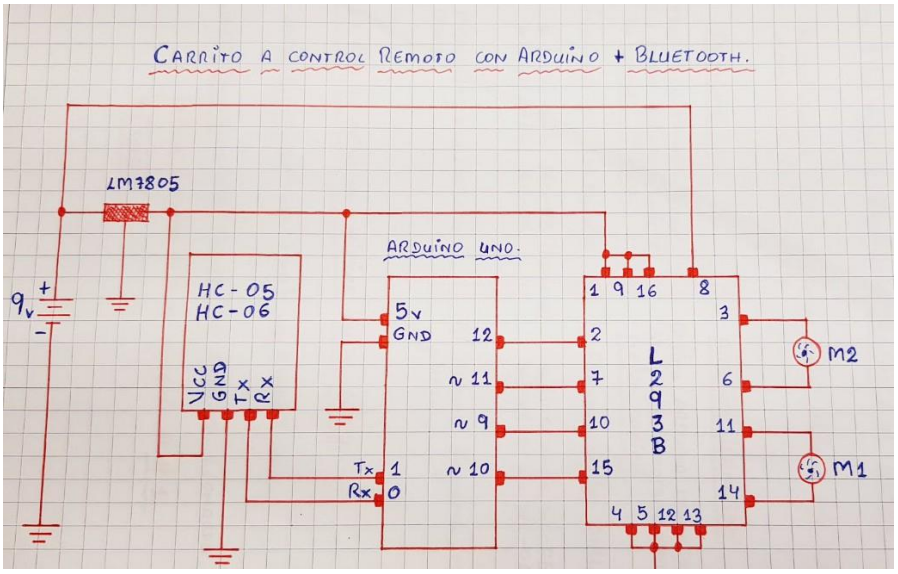
Este tipo de lenguaje posee una gran portabilidad. Gracias a ello se suele utilizar para la programación de sistemas como la construcción de intérpretes, compiladores, y editores de texto.

## III. DESARROLLO

- **PRIMER PROTOTIPO**



- **Diagrama:**




- **Materiales:**

- 1 Arduino UNO.
- 1 Módulo Bluetooth el HC-06.
- 1 Circuito integrado el L293B (driver para los motorreductores)
- 1 Regulador de voltaje el LM7805.
- 1 Regulador de voltaje el LM7809.
- 2 Motorreductores.
- 2 Llantas.
- 1 Rueda Loca.
- 1 Protoboard.
- Pilas de 9V.
- Cablecillos y/o Jumpers para las conexiones.

- **Código Arduino y explicación:**

```
int m1a = 10;            //Motor 1, pin 10 del arduino va al pin 15 del L293B.
int m1b = 9;             //Motor 1, pin 9 del arduino va al pin 10 del L293B.
int m2a = 12;            //Motor 2, pin 12 DEL arduino va al pin 2 del L293B.
int m2b = 11;            //Motor 2, pin 11 del arduino va al pin 7 del L293B.
char val;

void setup()
{
pinMode(m1a, OUTPUT);  // Digital pin 10 set as output Pin
pinMode(m1b, OUTPUT);  // Digital pin 9 set as output Pin
pinMode(m2a, OUTPUT);  // Digital pin 12 set as output Pin
pinMode(m2b, OUTPUT);  // Digital pin 11 set as output Pin
Serial.begin(9600);
}

void loop()
```

```
{
  while (Serial.available() > 0)
  {
  val = Serial.read();
  Serial.println(val);
  }

  if( val == 'F')                    // Hacia adelante
    {
      digitalWrite(m1a, HIGH);
      digitalWrite(m1b, LOW);
      digitalWrite(m2a, HIGH);
      digitalWrite(m2b, LOW);
    }
  else if(val == 'B')               // Hacia atrás
    {
      digitalWrite(m1a, LOW);
      digitalWrite(m1b, HIGH);
      digitalWrite(m2a, LOW);
      digitalWrite(m2b, HIGH);
    }

    else if(val == 'L')              // Izquierda
    {
    digitalWrite(m1a, LOW);
    digitalWrite(m1b, LOW);
    digitalWrite(m2a, HIGH);
    digitalWrite(m2b, LOW);
    }
    else if(val == 'R')               //Derecha
    {
    digitalWrite(m1a, HIGH);
    digitalWrite(m1b, LOW);
    digitalWrite(m2a, LOW);
    digitalWrite(m2b, LOW);
    }
      else if(val == 'S')                //Stop - Pare, Carrito detenido
    {
    digitalWrite(m1a,  LOW);
    digitalWrite(m1b,  LOW);
    digitalWrite(m2a,  LOW);
    digitalWrite(m2b, LOW);
    }
}
```

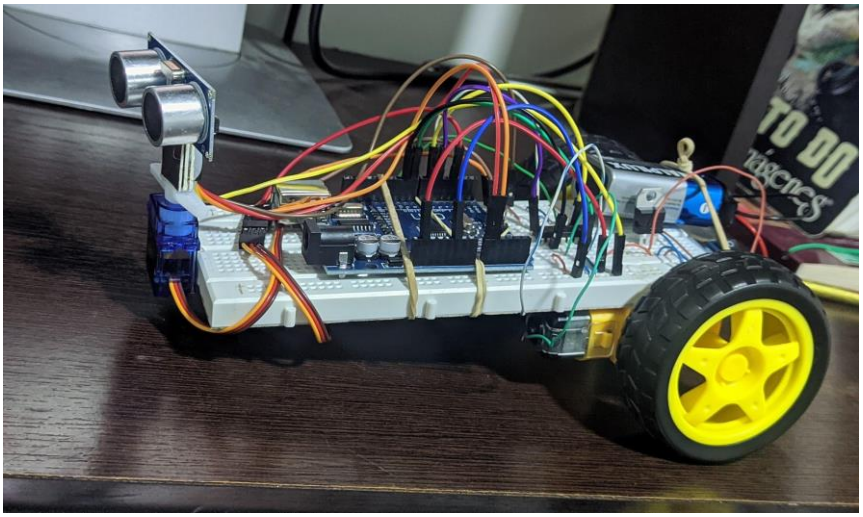
**Variables de pin del motor:** Las primeras cuatro líneas definen las variables para los pines del motor que están conectados al controlador de motor L293B. Los pines 10 y 9 del Arduino están conectados a los pines 15 y 10 del L293B para el Motor 1, respectivamente. Los pines 12 y 11 del Arduino están conectados a los pines 2 y 7 del L293B para el Motor 2, respectivamente.

**Función setup():** Esta función se ejecuta una vez cuando el Arduino se enciende o se reinicia. Aquí, se configuran los pines del motor como salidas y se inicia la comunicación serial a una velocidad de 9600 baudios.

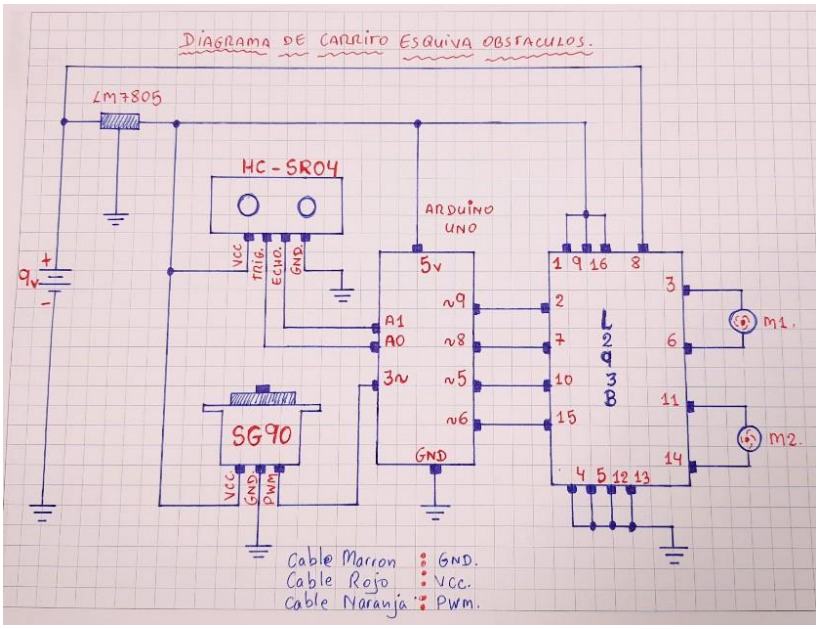
**Función loop():** Esta función se ejecuta repetidamente en un bucle después de que se ejecuta la función setup(). Aquí, se lee cualquier dato disponible en el puerto serial y se almacena en la variable 'val'.

**Control del motor:** Dependiendo del valor de 'val', se controla la dirección de los motores. Si 'val' es 'F', el carro se mueve hacia adelante. Si es 'B', el carro se mueve hacia atrás. Si es 'L', el carro gira a la izquierda. Si es 'R', el carro gira a la derecha. Si es 'S', el carro se detiene.

## ❖ SEGUNDO PROTOTIPO



- **Diagrama:**



- **Materiales:**

- 1 sensor ultrasónico el HC-SR04.
- 1 servomotor, el SG90.
- 1 regulador de voltaje, el LM7805.
- 1 Arduino UNO.
- 1 circuito integrado, el L293B para controlar los motores.
- Pilas de 9V.
- 1 interruptor pequeño.
- 2 motores DC.
- 2 llantas.
- 1 rueda loca.
- 1 protoboard.
- Cables o jumpers para realizar todas las conexiones entre el Arduino y la protoboard.

**- SERVOMOTOR EL SG90:** El servomotor SG90 es de tamaño pequeño ideal para proyectos de bajo torque y donde se requiera poco peso. Muy usado en aeromodelismo, pequeños brazos robóticos y mini artrópodos. Un servo ideal para aprender a programar en Arduino.

Puede rotar aproximadamente 180 grados (90° en cada dirección). Tiene la facilidad de poder trabajar con diversidad de plataformas de desarrollo como Arduino, PICs, Raspberry Pi, o en general a cualquier microcontrolador.

Los cables en el conector están distribuidos de la siguiente forma:

- Café o marrón = Tierra (Gnd).
- Rojo = VCC (5V).
- Naranja = Señal de control (PWM).

Características:

- Voltaje de Operación: 3.0 - 7.2V.
- Velocidad: 0.1seg / 60 grados
- Torque reposo: 1.3Kg x cm (4.8V), 1.6Kg (6.0V)
- Ancho de pulso: 4 seg (Dead band)
- Dimensiones: 22*11.5*27 mm
- Longitud del conductor: 150mm

**- SENSOR ULTRASONICO HC-SR04:** El sensor HC-SR04 es un sensor de distancia de bajo costo que utiliza ultrasonido para determinar la distancia de un objeto en un rango de 2 a 450 cm. Destaca por su pequeño tamaño, bajo consumo energético, buena precisión y excelente precio. El sensor HC-SR04 es el más utilizado dentro de los sensores de tipo ultrasonido, principalmente por la cantidad de información y proyectos disponibles en la web.

De igual forma es el más empleado en proyectos de robótica como robots laberinto o sumo, y en proyectos de automatización como sistemas de medición de nivel o distancia.

El sensor HC-SR04 posee dos transductores: un emisor y un receptor piezoeléctricos, además de la electrónica necesaria para su operación. El funcionamiento del sensor es el siguiente: el emisor piezoeléctrico emite 8 pulsos de ultrasonido(40KHz) luego de recibir la orden en el pin TRIG, las ondas de sonido viajan en el aire y rebota al encontrar un objeto, el sonido de rebote es detectado por el receptor piezoeléctrico, luego el pin ECHO cambia a Alto (5V) por un tiempo igual al que demoró la onda desde que fue emitida hasta que fue detectada, el tiempo del pulso ECO es medido por el microcontrolador y así se puede calcular la distancia al objeto. El funcionamiento del sensor no se ve afectado por la luz solar o material de color negro (aunque los materiales blandos acústicamente como tela o lana pueden llegar a ser difíciles de detectar).

Características del hc-sr04:

- Voltaje de Operación: 5V DC
- Corriente de reposo: < 2mA
- Corriente de trabajo: 15mA
- Rango de medición: 2cm a 450cm
- Precisión: +- 3mm
- Ángulo de apertura: 15°
- Frecuencia de ultrasonido: 40KHz
- Duración mínima del pulso de disparo TRIG (nivel TTL): 10 μS
- Duración del pulso ECO de salida (nivel TTL): 100-25000 μS
- Dimensiones: 45mm x 20mm x 15mm
- Tiempo mínimo de espera entre una medida y el inicio de otra 20ms (recomendable 50ms)

**- ARDUINO UNO:** Arduino es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para los creadores y desarrolladores. Esta plataforma permite crear diferentes tipos de microordenadores de una sola placa a los que la comunidad de creadores puede darles diferentes tipos de uso.

Para poder entender este concepto, primero vas a tener que entender los conceptos de hardware libre y el software libre. El hardware libre son los dispositivos cuyas especificaciones y diagramas son de acceso público, de manera que cualquiera puede replicarlos. Esto quiere decir que Arduino ofrece las bases para que cualquier otra persona o empresa pueda crear sus propias placas, pudiendo ser diferentes entre ellas, pero igualmente funcionales a partir de la misma base.

La placa Arduino UNO es la mejor placa para iniciar con la programación y la electrónica. Si es tu primera experiencia con la plataforma Arduino, el Arduino UNO es la opción más robusta, más usada y con mayor cantidad de documentación de toda la familia Arduino.

Arduino UNO es una placa basada en el microcontrolador ATmega328P. Tiene 14 pines de entrada/salida digital (de los cuales 6 pueden ser usados con PWM), 6 entradas analógicas, un cristal de 16Mhz, conexión USB, conector jack de alimentación, terminales para conexión ICSP y un botón de reseteo. Tiene toda la electrónica necesaria para que el microcontrolador opere, simplemente hay que conectarlo a la energía por el puerto USB ó con una pila de 9 o 12v por el conector Jack.

**- CIRCUITO INTEGRADO L293B:** El Driver puente H para motores L293B es un circuito integrado de cuatro canales diseñado para manejar cargas inductivas tales como relevos, solenoides, motores DC y motores paso a paso, muy similar al L293D pero con una salida máxima de corriente de

1A por canal, en la configuración conocida como puente H. Es posible configurarlo para manejar hasta 2 puentes H. El control de las entradas digitales del L293B puede hacerse desde cualquier microcontrolador, Arduino, Raspberry pi, TI Launchpad, BeagleBone, o cualquier otro sistema embebido.

Características:

- Salida de corriente por canal de forma continua: 1A.
- Salida de corriente por canal pico (No repetitivo): 2A.
- Pin o patilla para la función habilitador o Enable: 2.
- Máximo voltaje soportado por los pines Enable: 7Vdc.
- Protección de sobrecalentamiento.
- Rango de voltaje de alimentación: 4.5Vdc a 36Vdc.
- Rango de temperatura de operación: -40°C a +150°C.
- Encapsulado: DIP-16.
- Fabricante: STMicroelectronics.

- **Código Arduino y explicación:**

Para programar el código de este prototipo, se utilizó el paradigma de la programación orientada a objetos para Arduino. A continuación, se presentan y se explican las clases que componen el proyecto para controlar el carro evasor de obstáculos.

- **Archivo .h (MiRobot.h):**

```
#ifndef MiRobot_h
#define MiRobot_h

#include <NewPing.h>
#include <Servo.h>

class MiRobot {
public:
  MiRobot();
  void iniciar();
  void ejecutar();

private:
  int distance;
  int motorPin1, motorPin2, motorPin3, motorPin4;
  NewPing sonar;
  Servo myservo;

  void moveStop();
  void moveForward();
  void moveBackward();
  void turnRight();
  void turnLeft();
  int lookRight();
  int lookLeft();
  int readPing();
```

```
};

#endif
```

**Inclusión de bibliotecas:** Las bibliotecas NewPing.h y Servo.h se incluyen al principio del archivo. NewPing.h es una biblioteca para el sensor de ultrasonidos y Servo.h es una biblioteca para controlar servomotores.

**Clase MiRobot**: Se define una clase MiRobot con métodos públicos y privados.

**Métodos públicos:** Los métodos públicos son iniciar() y ejecutar(). Estos métodos se pueden llamar desde fuera de la clase. El método iniciar() se utiliza para inicializar el robot y el método ejecutar() se utiliza para ejecutar las acciones del robot.

**Métodos privados:** Los métodos privados son moveStop(), moveForward(), moveBackward(), turnRight(), turnLeft(), lookRight(), lookLeft() y readPing(). Estos métodos se utilizan para controlar los movimientos del robot y leer los datos del sensor de ultrasonidos.

**Variables privadas:** Las variables privadas son distance, motorPin1, motorPin2, motorPin3, motorPin4, sonar y myservo. distance almacena la distancia medida por el sensor de ultrasonidos. motorPin1, motorPin2, motorPin3 y motorPin4 son los pines de control del motor. sonar es una instancia de la clase NewPing para el sensor de ultrasonidos y myservo es una instancia de la clase Servo para el servomotor.

- **Archivo .cpp (MiRobot.cpp):**

```
#include "MiRobot.h"

MiRobot::MiRobot() : sonar(A0, A1, 200), motorPin1(8), motorPin2(9), motorPin3(5),
motorPin4(6) {
  // Constructor, inicializar variables si es necesario
}

void MiRobot::iniciar() {
  myservo.attach(3);
  myservo.write(115);
  delay(2000);

  for (int i = 0; i < 4; ++i) {
    distance = readPing();
    delay(100);
  }
}

void MiRobot::ejecutar() {
  distance = readPing();

  if (distance <= 15) {
    moveStop();
    delay(200);
    moveBackward();
```

```
    delay(800);
    moveStop();
    delay(200);

    int distanceR = lookRight();
    delay(200);
    int distanceL = lookLeft();
    delay(200);

    if (distanceR >= distanceL) {
      turnRight();
      moveStop();
    } else {
      turnLeft();
      moveStop();
    }

  } else {
    moveForward();
  }
  delay(50);  // Ajuste para estabilidad
}

int MiRobot::lookRight() {
  myservo.write(50);
  delay(500);
  int distance = readPing();
  delay(100);
  myservo.write(115);
  return distance;
}

int MiRobot::lookLeft() {
  myservo.write(170);
  delay(500);
  int distance = readPing();
  delay(100);
  myservo.write(115);
  return distance;
}

int MiRobot::readPing() {
  delay(70);
  int cm = sonar.ping_cm();
  if (cm == 0) {
    cm = 250;
  }
  return cm;
}

void MiRobot::moveStop() {
  analogWrite(motorPin1, 0);
  analogWrite(motorPin2, 0);
```

```
  analogWrite(motorPin3, 0);
  analogWrite(motorPin4, 0);
}

void MiRobot::moveForward() {
  analogWrite(motorPin1, 180);
  analogWrite(motorPin2, 0);
  analogWrite(motorPin3, 180);
  analogWrite(motorPin4, 0);
}

void MiRobot::moveBackward() {
  analogWrite(motorPin1, 0);
  analogWrite(motorPin2, 180);
  analogWrite(motorPin3, 0);
  analogWrite(motorPin4, 180);
}

void MiRobot::turnRight() {
  analogWrite(motorPin1, 180);
  analogWrite(motorPin2, 0);
  analogWrite(motorPin3, 0);
  analogWrite(motorPin4, 180);
  delay(300);
  moveForward();
}

void MiRobot::turnLeft() {
  analogWrite(motorPin1, 0);
  analogWrite(motorPin2, 180);
  analogWrite(motorPin3, 180);
  analogWrite(motorPin4, 0);
  delay(300);
  moveForward();
}
```

**Constructor (MiRobot::MiRobot()):** El constructor se encarga de inicializar los objetos miembros de la clase. En este caso, se inicializa el sensor ultrasónico (sonar) con los pines A0 y A1, y un rango máximo de 200 cm. También se establecen los pines para controlar los motores (motorPin1, motorPin2, motorPin3, motorPin4). La inicialización se realiza mediante una lista de inicialización.

**Método iniciar():** Este método se encarga de la configuración inicial del robot. Primero, adjunta el servomotor al pin 3 (myservo.attach(3)) y establece una posición inicial (myservo.write(115)). Luego, realiza cuatro lecturas iniciales del sensor ultrasónico para calibrar la distancia.

**Método ejecutar():** Este método representa el bucle principal de funcionamiento del robot. Lee la distancia actual del sensor ultrasónico (distance = readPing()) y toma decisiones basadas en esa distancia. Si la distancia es menor o igual a 15 cm, la lógica dentro del condicional se ejecuta para evitar obstáculos; de lo contrario, el robot avanza hacia adelante utilizando el método moveForward().

**Otros Métodos:**

int MiRobot::lookRight() { /* ... */ }
int MiRobot::lookLeft() { /* ... */ }
int MiRobot::readPing() { /* ... */ }
void MiRobot::moveStop() { /* ... */ } void
MiRobot::moveForward() { /* ... */ } void
MiRobot::moveBackward() { /* ... */ } void
MiRobot::turnRight() { /* ... */ }
void MiRobot::turnLeft() { /* ... */ }

Estos son los métodos adicionales que implementan funciones específicas del robot, como girar a la derecha, girar a la izquierda, detenerse, avanzar, retroceder y explorar con el servomotor en diferentes direcciones.

En resumen, el archivo MiRobot.cpp implementa los métodos de la clase MiRobot que definen el comportamiento del robot. Cada método realiza tareas específicas para controlar los motores, el sensor ultrasónico y el servomotor, proporcionando una organización modular y fácil de entender para el funcionamiento del robot.

- **Archivo .ino (main.ino):**

```
#include "MiRobot.h"

MiRobot miRobot;

void setup() {
  miRobot.iniciar();
}

void loop() {
  miRobot.ejecutar();
}
```

**Función setup():** La función setup() es una función de configuración que se ejecuta una vez al inicio del programa. En este caso, llama al método iniciar() de la instancia de MiRobot, que se encarga de la configuración inicial del robot.

**Función loop():** La función loop() es el bucle principal del programa que se ejecuta continuamente después de la función setup(). En este caso, llama al método ejecutar() de la instancia de MiRobot. El método ejecutar() contiene la lógica principal del robot, que incluye la lectura del sensor ultrasónico y la toma de decisiones basada en la distancia.

En resumen, el archivo main.ino se encarga de crear una instancia de la clase MiRobot, llamar al método de configuración inicial en setup(), y luego ejecutar el bucle principal llamando al método principal ejecutar() en loop(). Esto proporciona una estructura clara y organizada para el programa principal del robot.

## IV. REFERENCIAS

**[1]** V. Perfil, «Carrito a control remoto con arduino / Bluetooth / Android, circuito en protoboard y en baquelita.», 26 de junio de 2021. http://www.electronicaivanespinoza.com/2021/06/carrito-control-remoto-con-arduino.html

**[2]** Ivan Espinoza., «Carrito a control remoto con arduino / Bluetooth / Android, en Protoboard», YouTube. 24 de junio de 2021. [En línea]. Disponible en: https://www.youtube.com/watch?v=hqNz4wcLMy8

**[3]** V. Perfil, «Carrito evasor de obstáculos con arduino, con sensor ultrasónico y servomotor, en protoboard y baquelita.», 8 de noviembre de 2021. http://www.electronicaivanespinoza.com/2021/11/carrito-evasor-de-obstaculos-con.html

**[4]** Ivan Espinoza., «Carrito evasor de obstáculos con arduino, sensor ultrasónico y servomotor, en protoboard.», YouTube. 2 de noviembre de 2021. [En línea]. Disponible en: https://www.youtube.com/watch?v=j0RuO9pQvCI

**[5]** Manuel Alejandro Macias Silva «Link de accceso al repositorio GITHUB.» https://github.com/MatitasGamexD/Digitales-2