Universidad Bernardo O'Higgins

Facultad de Ingeniería, Ciencia y Tecnología

Programación IoT | Sección: TEO 1

Profesor Agustín Díaz

Auto controlado por PID

27 de octubre de 2025

Aldo Hernández
aldo.hernandezt@uanl.edu.mx
Universidad Autónoma de Nuevo León
San Nicolás de los Garza, Nuevo León, MX

Diego Zavalla
dzavalla@pregrado.ubo.cl
Universidad Bernardo O'Higgins
Universidad Bernardo O'Higgins, Santiago, CL

Isaías Márquez
imarquez@pregrado.ubo.cl
Universidad Bernardo O'Higgins
Universidad Bernardo O'Higgins, Santiago, CL

Patricio Gálvez
patriciogalvez@pregrado.ubo.cl
Universidad Bernardo O'Higgins
Universidad Bernardo O'Higgins, Santiago, CL

1. Introducción

Este proyecto se desarrolla para la competencia interna de resolución de un laberinto, mediante distintas etapas se busca construir un auto que pueda moverse a través de los diferentes caminos para lograr su resolución.

En este informe se incluye la construcción del auto junto con las funcionalidades básicas para el movimiento, además de algunas observaciones y posibles mejoras.

2. Hardware

2.1. Materiales

Para el desarrollo de este proyecto se utilizaron los siguientes materiales:

- 1 Arduino UNO R3
- 1 Sensor Shield V5.0
- 1 Módulo L298N
- 2 motores DC
- 3 sensores ultrasónicos HC-SR04
- 1 batería de 9V
- 1 interruptor
- Cables
- Jumpers

Además, se utilizó una plataforma de plástico para el chasis del auto junto con trozos de cartón para sujetar los sensores.

2.2. Armado

El auto fue ensamblado como se observa en la imagen 1. El módulo L298N se colocó en la parte superior (hacia el frente) del chasis mientras que el Arduino UNO R3 (que tiene el Sensor Shield V5.0 puesto), el interruptor y la batería se incluyeron en la parte inferior; los sensores fueron instalados al frente y uno por cada dirección.

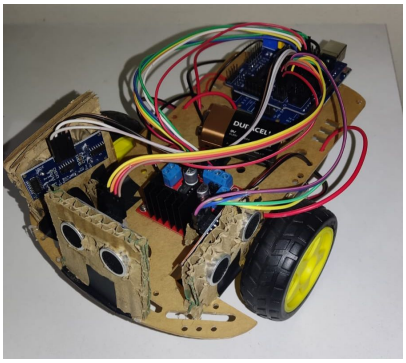

Figura 1: Auto ensamblado.

En versiones anteriores del auto, se usaban 4 baterías AA colocadas por debajo del chasis cuya salida daba a un interruptor que estaba conectado al módulo L298N. Esta funcionalidad del interruptor se busca implementar en la versión final del proyecto, de manera que la batería de 9V esté conectada al *switch* y éste al módulo L298N.

Una vez que se terminó el ensamblado, los cables fueron asegurados con cinta aislante y soldados a sus respectivos componentes para evitar falsos contactos y ruido en las señales.

3. Evidencia

El video del auto funcionando se puede revisar en el siguiente enlace de Google Drive. Se simuló una especie de pasillo juntando cajas de cartón para que el auto pasase entre ellas; la desviación final ocurrió debido a que los sensores del auto detectaron las paredes muy lejos (salió del espacio controlado con las cajas de cartón).

4. Código

En la figura 2 se pueden observar pequeños fragmentos del código utilizado. En la subfigura 2a se observan los pines conectados desde el Arduino UNO R3 a distintos componentes del auto para la transmisión de corriente y señales, mientras que en la subfigura 2b se muestran las constantes configuradas para el PID encontrados mediante **prueba y error**.

Finalmente en la subfigura 2c se enseñan los últimos cálculos realizados durante el ciclo de ejecución del código, estos se encargan de modificar los valores de los demás parámetros del PID para ajustar la velocidad y dirección de cada motor.

batbot.cpp

```
/// ---------- PINES MOTORES (puente H) ----------
const uint8_t ENA = 10;  // PWM motor izquierdo
const uint8_t IN1 = 13;
const uint8_t IN2 = 8;
const uint8_t ENB = 11;  // PWM motor derecho
const uint8_t IN3 = 9;
const uint8_t IN4 = 12;
/// ---------- PINES SENSORES ULTRASÓNICOS ----------
const uint8_t TRIG_F = 2, ECHO_F = 3;  // Frente
const uint8_t TRIG_L = 4, ECHO_L = 5;  // Izquierda
const uint8_t TRIG_R = 7, ECHO_R = 6;  // Derecha
```

(a) Pines conectados al Arduino UNO R3.

batbot.cpp

```
// Velocidades
int baseSpeed = 160;  // 0..255
int maxSpeed = 200;   // 0..255
/// ---------- PID ----------
float kp = 7.0;
float ki = 0.01;
float kd = 3.0;
float error = 0, error_ant = 0, integral = 0, derivada = 0;
unsigned long tk = 0, tk_1 = 0;
float h = 0.0;
```

(b) Parámetros actuales para el uso de PID.

batbot.cpp

```
// 4) Avance con seguimiento de pared (derecha)
// Si no hay pared a la derecha (muy libre), acércate suavemente hasta encontrarla
float referencia = objetivoDerecha_cm;
float medida = dR;
if (medida > CM_INVALIDO - 1) {
  // Lectura inválida: finge “muy lejos” para no saturar el PID
  medida = umbralLibre_cm + 10.0;
}
// Error: lo que quiero - lo que tengo (distancia a la pared derecha)
error = referencia - medida;
// PID discreto (ms → convertir a segundos para ki/kd coherentes si los calibras en s)
float h_s = h / 1000.0f;
integral += error * h_s;
// Anti-windup básico
integral = clampf(integral, -50.0f, 50.0f);
derivada = (error - error_ant) / h_s;
float pid = kp * error + ki * integral + kd * derivada;
// 5) Velocidades diferenciales (sumo/resto corrección)
int vIzq = baseSpeed + (int)pid;
int vDer = baseSpeed - (int)pid;
// Limitar
vIzq = clampi(vIzq, 0, maxSpeed);
vDer = clampi(vDer, 0, maxSpeed);
// 6) Avance
motoresAdelante(vIzq, vDer);
```

(c) Cálculos realizados con relación al PID.

Figura 2: Fragmentos del código utilizado.

Todo el código usado en el auto se puede consultar en la siguiente liga. Cuando tengamos una versión final, se trasladará a un proyecto más estructurado que se podrá consultar en el siguiente repositorio que actualmente cuenta con una versión antigua del código utilizado; este proyecto está estructurado en diferentes archivos que definen distintos tipos de funciones según su uso, fomentando la modularización.

5. Observaciones y mejoras

En cuanto a los sensores laterales se observa un error promedio aproximado de 1 cm del valor deseado mientras que la lectura del sensor central es buena y tiene un error bastante bajo (ya que se detiene el auto justo cuando debería). Para el PID se implementó un *anti-windup* y se consideraron los siguientes valores que fueron ajustados mediante **prueba y error**:

- $K_p = 7{,}0$
- $K_i = 0{,}01$
- $K_p = 3{,}0$

Actualmente se busca mejorar la velocidad y particularmente el giro ya que es un tanto descontrolado mediante el ajuste de velocidades y constantes, además se planea integrar un algoritmo de resolución de laberintos (aunque sea el más simple). Junto con todo esto, queremos realizar una refactorización del código fuente para facilitar la legibilidad.