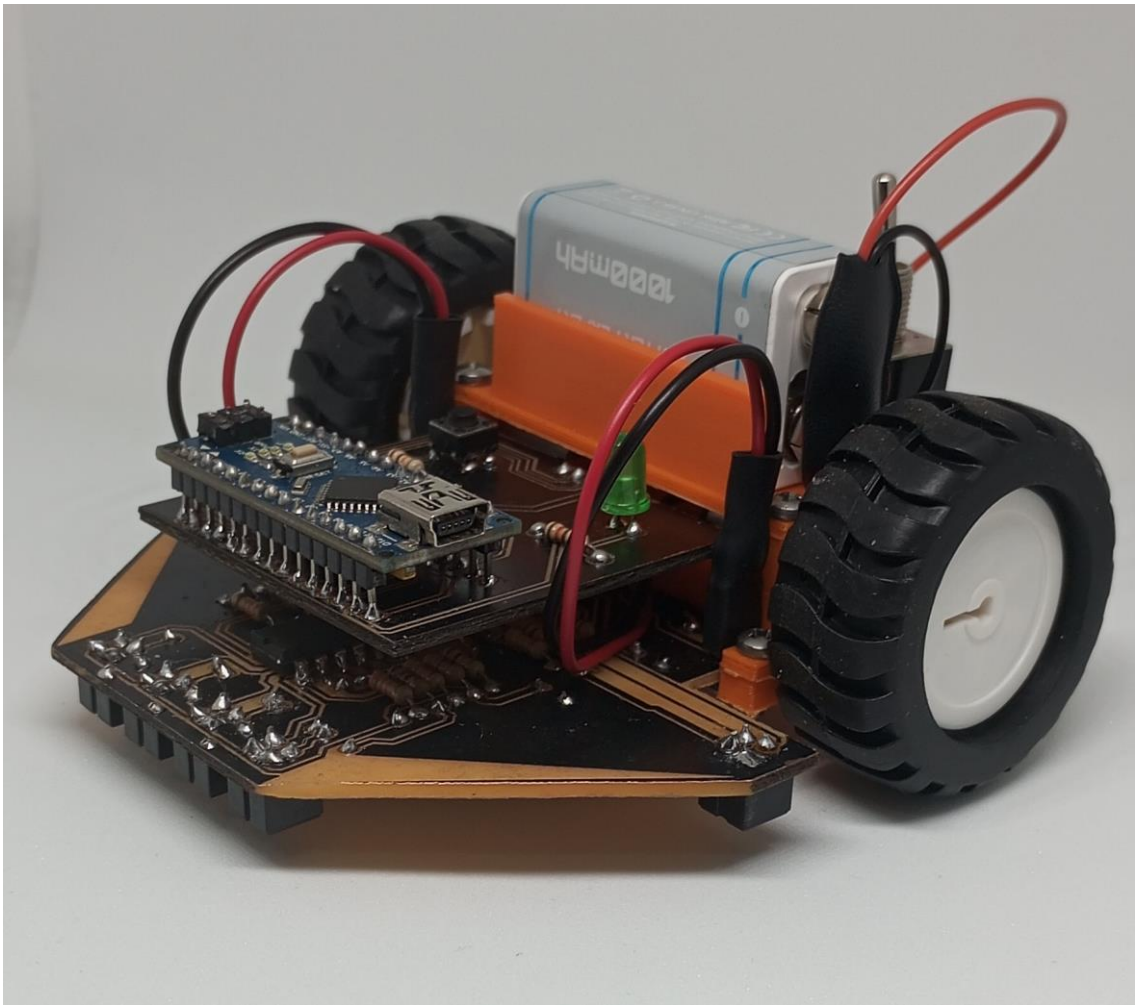
SEGUIDOR DE LÍNEA

Proyecto con Arduino Nano



Autor: Fernández Hernán Gustavo
Fecha: 11/04/2025

Índice

1. Introducción

Este proyecto consiste en el diseño de un robot seguidor de línea basado en Arduino Nano. Un robot seguidor de línea es un sistema autónomo capaz de detectar y seguir una trayectoria marcada, generalmente una línea negra sobre un fondo blanco, utilizando sensores ópticos para detectar contrastes de color.

El presente robot se caracteriza por incorporar un sistema de calibración en tres pasos que permite adaptarse a diferentes condiciones de iluminación y superficies, así como la detección inteligente tanto de la línea negra como de la ausencia de superficie, lo que añade una capa adicional de seguridad al funcionamiento.

El control de movimiento se realiza mediante un algoritmo PD (Proporcional-Derivativo) que permite un seguimiento suave y preciso de la línea, incluso en curvas cerradas.

2. Objetivos

- Desarrollar un robot autónomo capaz de seguir líneas negras sobre superficie clara.
- Implementar un sistema de calibración que permita adaptarse a diferentes condiciones de luz ambiental.
- Incorporar un algoritmo de control PD para optimizar el seguimiento.
- Añadir funciones de seguridad como detección de ausencia de superficie.
- Diseñar un sistema indicador mediante LED para conocer el estado del robot en todo momento.
- Implementar un sistema de depuración mediante comunicación serial.

3. Fundamentos teóricos

3.1 Principio de funcionamiento

Los robots seguidores de línea operan bajo el principio de detección de contraste. Utilizan sensores infrarrojos reflexivos que emiten luz y miden la cantidad reflejada. Las superficies claras reflejan más luz que las oscuras, permitiendo distinguir entre la línea y el fondo.

El robot utiliza esta información para determinar su posición relativa respecto a la línea y ajustar la velocidad de sus motores para mantener la trayectoria deseada. La disposición de múltiples sensores en línea permite no solo detectar la presencia de la línea sino también calcular un "error" de posición que indica cuánto se ha desviado el robot del centro de la línea

3.2 Control PID

Para el seguimiento preciso de la línea, se implementa un controlador PD (Proporcional-Derivativo), una variante simplificada del control PID (Proporcional-Integral-Derivativo) donde:

- **Control Proporcional (P)**: Produce una corrección proporcional al error actual. Si el robot se desvía a la derecha, el control proporcional generará una corrección hacia la izquierda proporcional a esa desviación.
- **Control Derivativo (D)**: Produce una corrección basada en la tasa de cambio del error. Esto anticipa hacia dónde se está moviendo el robot y añade estabilidad al sistema, evitando oscilaciones.

La fórmula implementada es:

Corrección = (Kp * P) + (Kd * D)

Donde:

- **Kp** es la constante proporcional (35.0 en este proyecto)
- **Kd** es la constante derivativa (15.0 en este proyecto)
- **P** es el error actual
- **D** es la diferencia entre el error actual y el anterior

4. Diseño del Hardware

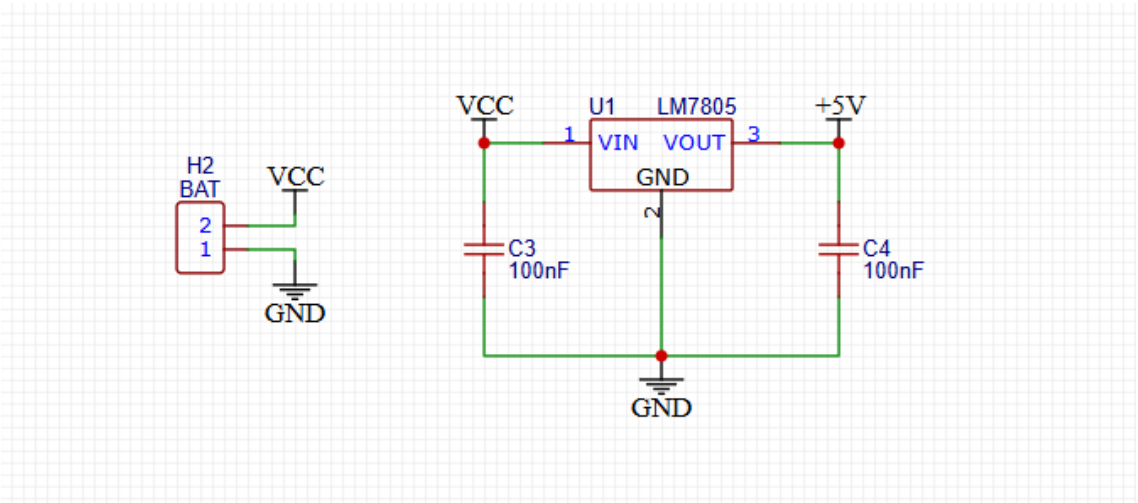
4.1 Arquitectura general

El hardware del robot seguidor de línea está organizado en dos placas interconectadas:

1. **Placa de potencia**: Contiene los circuitos para:
   - Regulación de voltaje
   - Interfaz de sensores
   - Control de motores
2. **Placa de control**: Contiene:
   - Microcontrolador Arduino Nano
   - Conectores para sensores y motores
   - Interfaz de usuario (LED indicador y pulsador)

Esta separación modular facilita el mantenimiento y permite una clara división entre la etapa de potencia y la etapa de control lógico.

4.2 Circuito de alimentación



El sistema de alimentación está basado en un regulador de voltaje LM7805 (U1), que convierte el voltaje de entrada (VCC, normalmente proporcionado por una batería de 9V o 7.4V) a un voltaje estable de +5V para alimentar los componentes lógicos del circuito.

Componentes principales:

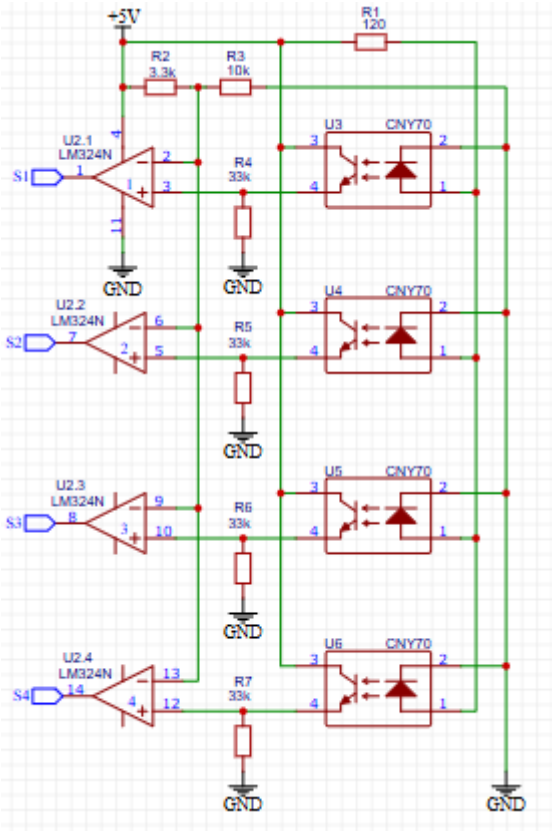
- **U1 (LM7805)**: Regulador de tensión con salida fija de 5V.
- **C3, C4**: Condensadores de 100nF para filtrado y estabilización.

Funcionamiento:

1. El voltaje de entrada (VCC) llega al pin de entrada del LM7805.
2. El regulador estabiliza este voltaje a 5V en su salida.
3. Los condensadores C3 y C4 filtran ruidos y estabilizan la salida ante cambios rápidos de carga.

Este circuito asegura que todos los componentes del robot, especialmente el Arduino Nano y los sensores, reciban un voltaje constante y estable, independientemente de las fluctuaciones en la batería.

4.3 Circuito de sensores S1, S2, S3 & S4

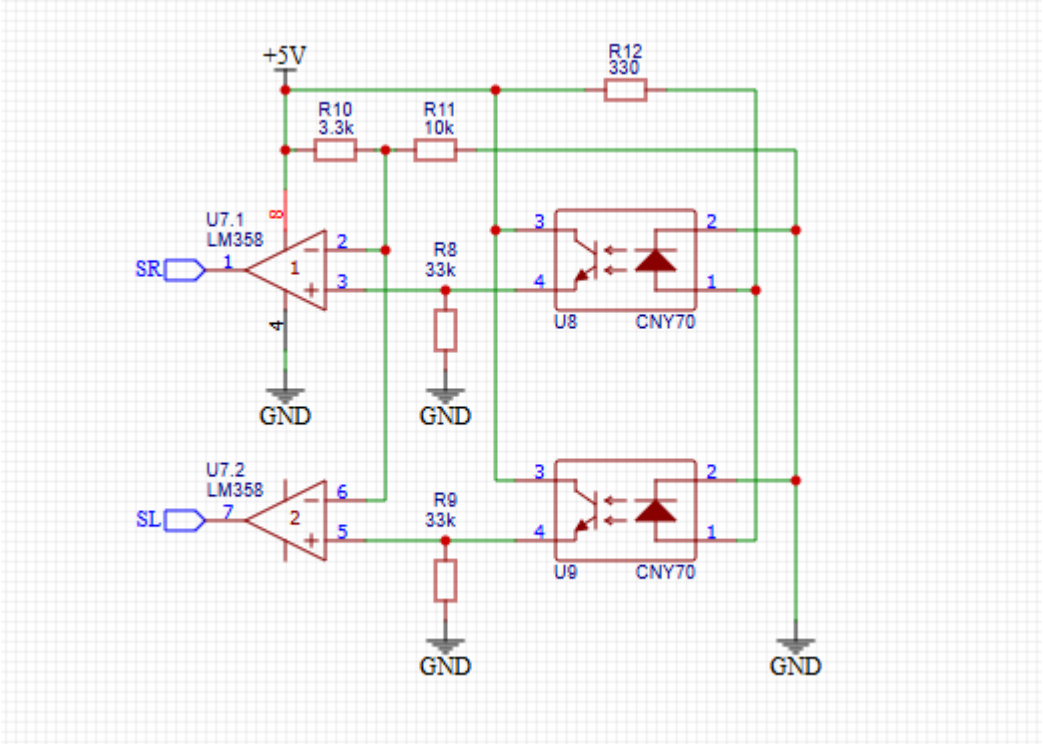


La interfaz de sensores S1, S2, S3 y S4 utiliza un amplificador operacional LM324N configurado como comparador para procesar las señales analógicas de los sensores infrarrojos.

Componentes principales:

- **U2.1-U2.4, UT.1-UT.2**: Amplificador operacional LM324N (cada chip contiene 4 amplificadores).
- **R1**: Resistencia de 120Ω para protección del diodo emisor del sensor.
- **R4, R5, R6, R7**: Resistencias de 33kΩ para limitar corriente.
- **R2**: Resistencia de 3.3kΩ como pull-up.
- **R3**: Resistencia de 10kΩ para crear el voltaje de referencia.
- **U3, U4, U5, U6**: Optoacopladores CNY70 para aislar la etapa lógica de la de potencia.

4.4 Circuito de sensores SL & SR



La interfaz de sensores SL y SR utiliza un amplificador operacional LM358 configurado como comparador para procesar las señales analógicas de los sensores infrarrojos.

Componentes principales:

- **U7.1-U7.2**: Amplificador operacional LM358 (cada chip contiene 2 amplificadores).
- **R12**: Resistencia de 330Ω o 120Ω para protección del diodo emisor del sensor.
- **R8, R9**: Resistencias de 33kΩ para limitar corriente.
- **R10**: Resistencia de 3.3kΩ como pull-up.
- **R11**: Resistencia de 10kΩ para crear el voltaje de referencia.
- **U8, U9**: Optoacopladores CNY70 para aislar la etapa lógica de la de potencia.

Funcionamiento del comparador (LM324N & LM358):

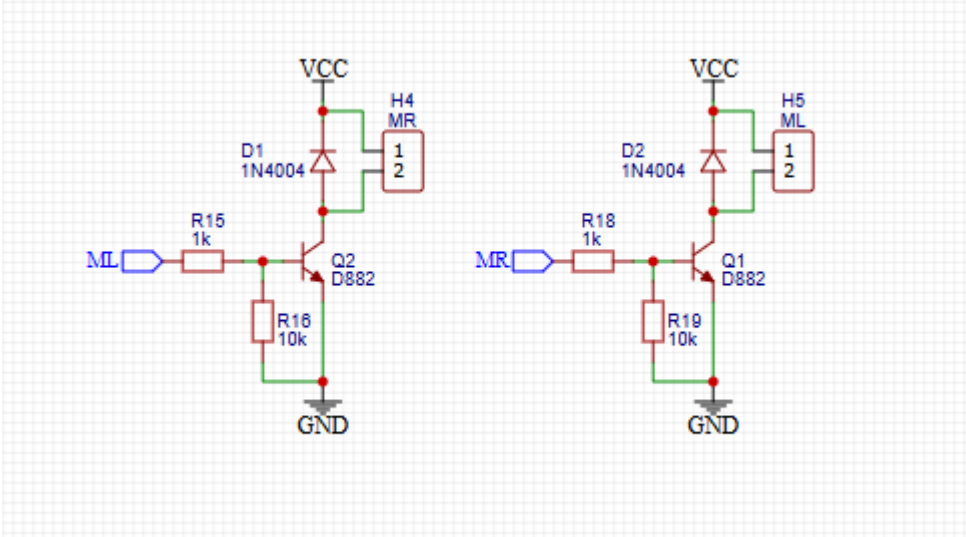
1. El sensor IR reflectivo envía una señal analógica que varía según la reflectividad de la superficie.
2. El amplificador operacional compara esta señal con un voltaje de referencia.
3. El voltaje de referencia se establece mediante un divisor de tensión (R2/R10 y R3/R11).
4. Cuando la señal del sensor supera el umbral, la salida del comparador cambia de estado.
5. La resistencia de 33kΩ limita la corriente hacia el optoacoplador.
6. El optoacoplador proporciona aislamiento galvánico entre las etapas de control y potencia.

Ventajas de esta configuración:

- Permite ajustar la sensibilidad de los sensores mediante el voltaje de referencia.
- El aislamiento óptico protege el circuito de control de posibles ruidos eléctricos de los motores.
- La comparación directa proporciona una respuesta rápida a los cambios de superficie.

Los seis sensores (S1, S2, S3, S4, SL, SR) están dispuestos en línea en la parte frontal del robot, con los sensores laterales (SL y SR) ligeramente separados para detectar curvas cerradas.

4.5 Etapa de potencia para motores



El control de los motores DC se realiza mediante transistores de potencia D882 que actúan como interruptores controlados por corriente.

Componentes principales:

- **Q1, Q2**: Transistores NPN D882 de potencia (uno por salida de motor).
- **D1, D2**: Diodos 1N4004 para protección contra corrientes inversas generadas por los motores.
- **R15, R18**: Resistencias de 1kΩ para limitar la corriente de base de los transistores.
- **R16, R19**: Resistencias de 10kΩ como pull-down para asegurar el apagado.

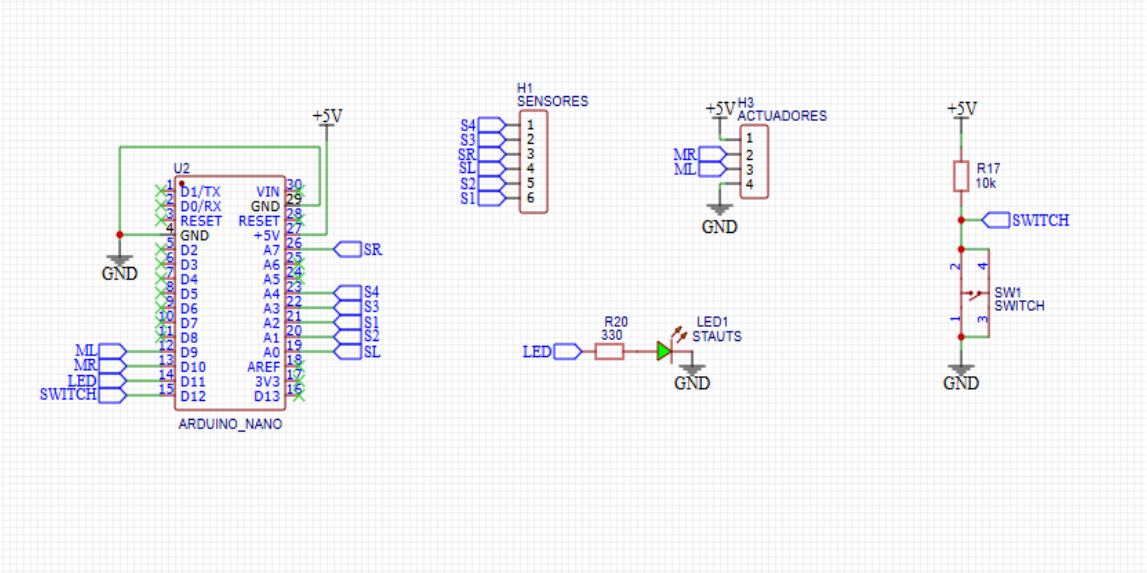
Funcionamiento:

1. Las señales PWM generadas por el Arduino (pines D9 y D10) se aplican a través de las resistencias de base a los transistores.
2. Cada transistor D882 controla una salida específica del motor, funcionando en modo corte/saturación.
3. Los diodos de protección 1N4004 absorben los picos de voltaje inverso generados por la naturaleza inductiva de los motores.

Principio de operación:

- Cuando el D882 recibe señal en su base, entra en saturación y conecta la salida del motor a la alimentación.
- La modulación PWM permite controlar la velocidad al variar el ciclo de trabajo de la señal.

4.6 Circuito de control con Arduino Nano



El cerebro del robot es un Arduino Nano, que procesa las señales de los sensores y controla los motores a través de la placa de potencia.

Componentes principales:

- **Arduino Nano**: Microcontrolador basado en ATmega328P.
- **LED1**: LED indicador con resistencia limitadora R20 (330Ω).
- **SWITCH**: Pulsador con resistencia pull-up R17 (10kΩ).
- **Conectores**: Para sensores (H7) y motores (H8).

Conexiones principales:

- **A0-A5**: Entradas analógicas conectadas a los sensores infrarrojos.
- **D9, D10**: Salidas PWM para control de velocidad de motores.
- **D11**: Salida digital para el LED indicador.
- **D12**: Entrada digital para el pulsador de control.

Funcionamiento:

1. El Arduino lee los valores analógicos de los sensores (A0-A5).
2. Procesa esta información para determinar la posición de la línea.
3. Calcula las correcciones necesarias mediante el algoritmo PD.
4. Genera señales PWM (D9, D10) para controlar la velocidad de los motores.
5. Utiliza el LED (D11) para indicar el estado actual del robot.
6. Detecta pulsaciones del botón (D12) para cambiar entre estados.
7. Envía información de depuración a través de la comunicación serial.

El circuito del LED indicador permite visualizar el estado operativo del robot mediante diferentes patrones de parpadeo:

- Parpadeo lento: Estado ESPERA
- Parpadeo medio: Estado CALIBRADO_BLANCO
- Parpadeo rápido: Estado CALIBRADO_NEGRO
- Encendido continuo: Estado ACTIVO

4.7 Cálculos de componentes

Cálculos para la polarización de los transistores D882:

Los transistores **D882** se utilizan en conmutación (corte/saturación) para controlar los motores. Para un correcto funcionamiento debemos calcular:

1. **Resistencia de base para asegurar saturación**:

   Para garantizar que el transistor entre en saturación, se debe cumplir:

   IB > IC / β

   Donde:

   - IC: Corriente de colector (corriente del motor)
   - β (hFE): Ganancia del transistor

   Para un motor que consume 500mA y un D882 con hFE mínimo de 60:

   IB_min = 500mA / 60 = 8.3mA

   Para tener un margen de seguridad, se suele multiplicar por un factor de 1.5-2:

   IB_design = 8.3mA × 1.5 = 12.5mA

   La resistencia de base se calcula entonces como:

   RB = (Vin - VBE) / IB_design
   RB = (5V - 0.7V) / 12.5mA = 344Ω

   Se utiliza el valor estándar de 1kΩ, que proporciona aproximadamente:

   IB_actual = (5V - 0.7V) / 1kΩ = 4.3mA

   Esto sigue siendo suficiente porque:

   IC_max con saturación = 4.3mA × 60 = 258mA

   Para corrientes mayores, el transistor operará con una caída de tensión colector-emisor mayor, pero seguirá conduciendo correctamente.

2. **Potencia disipada por el transistor**:

   En saturación, la caída de tensión colector-emisor (VCE_sat) es aproximadamente 0.2V para el D882:

   P_diss = VCE_sat × IC = 0.2V × 500mA = 0.1W

   El D882 puede disipar hasta 30W con un disipador adecuado, por lo que opera muy por debajo de su límite.

3. **Resistencia pull-down**:

   Las resistencias R16 y R19 de 10kΩ garantizan que los transistores estén en corte cuando no hay señal:

   I_leak_max = VCC / R_pulldown = 5V / 10kΩ = 0.5mA

   Esta corriente es mucho menor que la necesaria para activar el motor.

Cálculos para el LM358/LM324N configurado como comparador:

El amplificador operacional LM324N y LM358 se configuran como comparador para detectar la presencia de la línea negra. Los cálculos relevantes son:

1. **Voltaje de referencia**:

   El voltaje de referencia se establece mediante un divisor de tensión formado por R2/R10 (3.3kΩ) y R3/R11 (10kΩ):

   Vref = VCC × R3 / (R2 + R3)
   Vref = 5V × 10kΩ / (3.3kΩ + 10kΩ) = 3.75V

   Este valor es el umbral con el que se comparan las señales de los sensores.

2. **Resistencia limitadora de salida**:

   Las resistencias R4-R9 (33kΩ) limitan la corriente de salida del comparador:

   I_out_max = (VCC - V_LED) / R_out
   I_out_max = (5V - 1.2V) / 33kΩ = 0.12mA

   Donde V_LED es la caída de tensión típica en el LED del optoacoplador.

3. **Corriente de polarización del LM324N**:

   La corriente de polarización de entrada del LM324N es aproximadamente 100nA, lo que produce un error de offset de:

   V_offset = I_bias × R = 100nA × 33kΩ = 3.3mV

   Este error es insignificante comparado con las variaciones de la señal.

Resistencia limitadora para el LED indicador:

La resistencia R20 (330Ω) para el LED se calcula mediante:

R = (Vcc - Vled) / I = (5V - 2V) / 10mA = 300Ω

Se usa el valor estándar de 330Ω, limitando la corriente a aproximadamente 9.1mA, lo que proporciona buena luminosidad sin sobrecargar el pin de salida del Arduino.

Disipación de potencia del regulador:

Para el regulador LM7805, la disipación de potencia con una entrada de 9V y carga de 200mA:

PD = (Vin - Vout) × I_load = (9V - 5V) × 200mA = 0.8W

5. Implementación del Software

5.1 Estructura del código

El código para el Arduino Nano se organiza en varias secciones principales:

1. **Definición de pines**: Asignación de pines para sensores, motores y elementos de control.

```
// Sensores de línea
#define S1            A2  // Sensor izquierda extrema
#define S2            A1  // Sensor izquierda media
#define S3            A3  // Sensor derecha media
#define S4            A4  // Sensor derecha extrema
#define SL            A0  // Sensor lateral izquierdo
#define SR            A5  // Sensor lateral derecho

// Control
#define SWITCH        12  // Pulsador de inicio
#define LED           11  // LED indicador de estado

// Motores
#define MOTOR_RIGHT   10  // Motor derecho (PWM)
#define MOTOR_LEFT    9   // Motor izquierdo (PWM)
```

17. **Configuración del controlador PD**: Constantes y variables para el algoritmo de control.

```
float Kp = 35.0;          // Constante proporcional
float Kd = 15.0;          // Constante derivativa
float error = 0;          // Error actual
float lastError = 0;      // Error anterior
int BASE_SPEED = 100;     // Velocidad base
int MAX_SPEED = 255;      // Velocidad máxima
```

24. **Valores de calibración**: Almacenamiento de umbrales y lecturas de referencia.

```
25.  // Umbrales calculados
26.  int umbralS1 = 500;       // Umbral para sensor S1
27.  int umbralS2 = 500;       // Umbral para sensor S2
28.  // ...
29.
30.  // Lecturas calibradas para superficie blanca
31.  int blanco1, blanco2, blanco3, blanco4, blancoL, blancoR;
32.
33.  // Lecturas calibradas para línea negra
34.  int negro1, negro2, negro3, negro4, negroL, negroR;
```

35. **Estados del robot**: Definición de los estados operativos.

```
36.  #define ESPERA 0  // Esperando primera pulsación para calibrar
   fondo blanco
37.  #define CALIBRADO_BLANCO 1    // Calibrado fondo blanco,
   esperando pulsación para calibrar línea negra
38.  #define CALIBRADO_NEGRO 2     // Calibrado completo, esperando
   pulsación para iniciar
39.  #define ACTIVO 3              // Robot en funcionamiento
40.
41.  byte estadoRobot = ESPERA;    // Estado inicial
```

42. **Funciones de configuración**: Setup inicial y configuración de pines.

```
43.  void setup() {
44.    // Configurar pines
45.    configurarPines();
46.
47.    // Iniciar comunicación serial
48.    Serial.begin(9600);
49.
50.    // Mostrar mensaje de bienvenida
51.    mostrarInstrucciones();
52.
53.    // Parpadeo inicial para indicar que el sistema está listo
54.    parpadearLED(3, 100);
55.  }
```

56. **Funciones de control**: Gestión del pulsador y estados.
57. **Funciones de calibración**: Calibrado de valores de blanco y negro.
58. **Funciones de seguimiento**: Algoritmo principal para seguir la línea.

5.2 Proceso de calibración

El robot implementa un proceso de calibración en tres pasos, diseñado para adaptarse dinámicamente a diferentes condiciones de luz y superficies:

1. **Calibración de fondo blanco**:
    - Se activa con la primera pulsación del botón
    - El robot toma 10 muestras de cada sensor sobre la superficie blanca
    - Calcula el promedio para cada sensor y lo almacena como referencia
2. **Calibración de línea negra**:
    - Se activa con la segunda pulsación del botón
    - El robot toma 10 muestras de cada sensor sobre la línea negra
    - Calcula el promedio para cada sensor y lo almacena como referencia
3. **Cálculo de umbrales**:
    - Determina automáticamente el umbral óptimo para cada sensor como el punto medio entre las lecturas de blanco y negro
    - Detecta si el negro da valores más altos o más bajos que el blanco para cada sensor (adaptación a diferentes tipos de sensores)

Este proceso de calibración permite que el robot funcione correctamente en diferentes condiciones de iluminación y con diferentes características de superficie, sin necesidad de ajustes manuales en el código.

5.3 Algoritmo de seguimiento

El algoritmo de seguimiento de línea implementa las siguientes características:

1. **Detección de línea**:
    - Compara los valores de los sensores con los umbrales calculados
    - Determina si cada sensor está detectando línea negra o fondo blanco
    - La comparación es dinámica según los valores calibrados (adaptable a sensores que pueden comportarse de manera diferente)
2. **Control PD**:
    - Calcula el error de posición basado en las lecturas de los sensores
    - Los sensores tienen diferentes pesos según su posición (-3, -1, 1, 3)
    - Los sensores laterales tienen pesos mayores (-5, 5) para curvas cerradas
    - Aplica corrección proporcional y derivativa para ajustar velocidades
3. **Control de motores**:
    - Ajusta la velocidad de cada motor según la corrección calculada
    - Implementa lógica especial para curvas cerradas detectadas con sensores laterales
    - Mantiene velocidades dentro de los límites establecidos (0-255)
4. **Detección de ausencia de superficie**:
    - Verifica si las lecturas de los sensores están fuera del rango esperado
    - Detiene el robot si detecta ausencia de superficie (seguridad)
    - Proporciona indicación visual mediante parpadeo del LED
5. **Depuración**:
    - Envía información del estado de los sensores por comunicación serial
    - Permite monitorear el funcionamiento en tiempo real

5.4 Control PD

El control PD (Proporcional-Derivativo) se implementa en la función **procesarSeguimientoLinea()**:

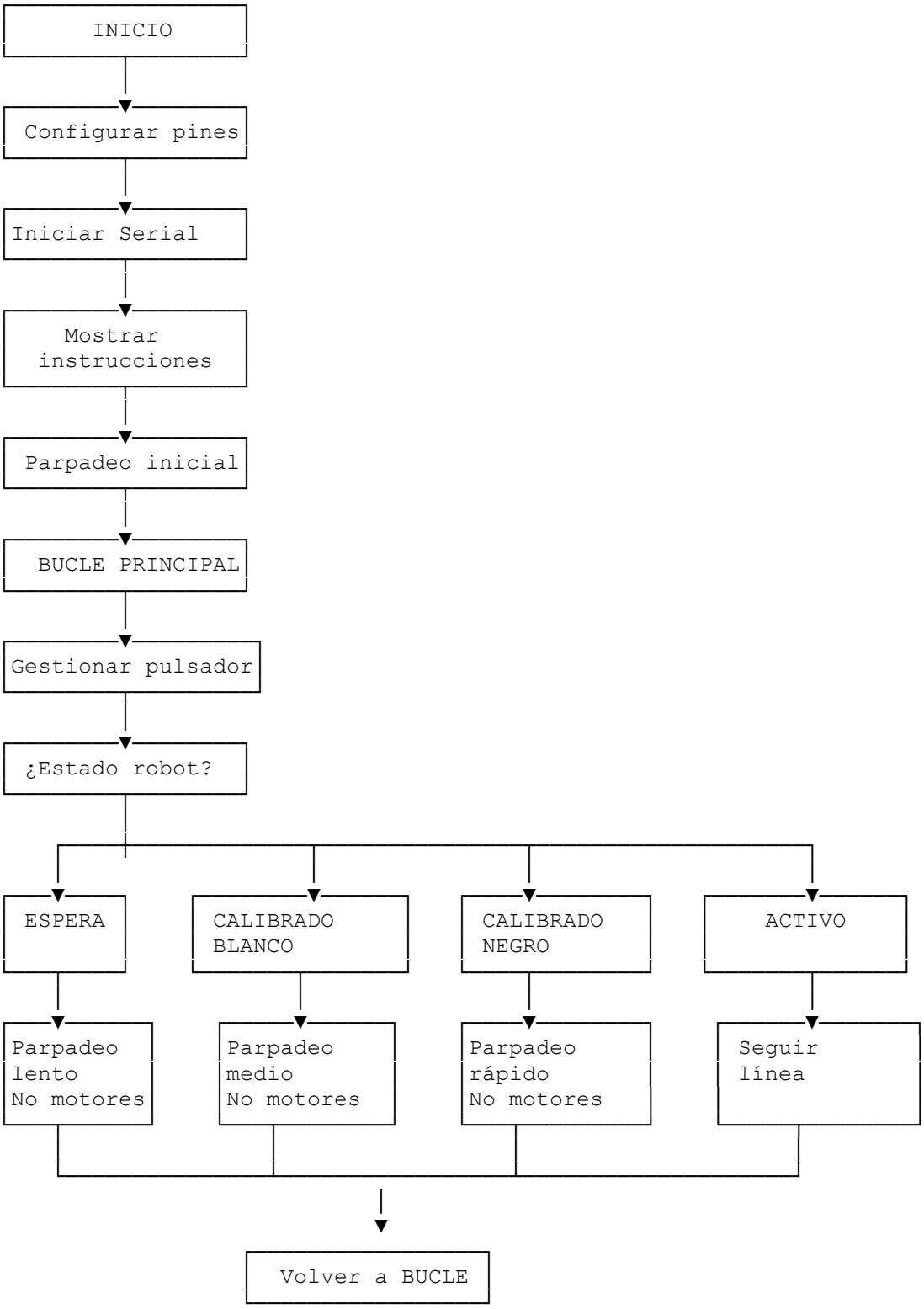
```
void procesarSeguimientoLinea(byte s1, byte s2, byte s3, byte s4, byte
sl, byte sr) {
  // Cálculo del error para línea negra
  error = (s1 * -3) + (s2 * -1) + (s3 * 1) + (s4 * 3);

  // Ajuste con sensores laterales
  if (sl) error -= 5;  // SL produce error negativo (giro a la
izquierda)
  if (sr) error += 5;  // SR produce error positivo (giro a la
derecha)

  // Control PD
  float P = error;
  float D = error - lastError;
  lastError = error;

  float correction = (Kp * P) + (Kd * D);

  // Ajustar velocidad de motores según la corrección calculada
  int motorSpeedLeft = constrain(BASE_SPEED - correction, 0,
MAX_SPEED);
  int motorSpeedRight = constrain(BASE_SPEED + correction, 0,
MAX_SPEED);

  // Control para curvas cerradas
  if (sl && !s1 && !s2) {
    // Curva cerrada izquierda
    motorSpeedLeft = MAX_SPEED;
    motorSpeedRight = 0;
  }

  if (sr && !s3 && !s4) {
    // Curva cerrada derecha
    motorSpeedLeft = 0;
    motorSpeedRight = MAX_SPEED;
  }

  // Aplicar velocidades a los motores
  analogWrite(MOTOR_LEFT, motorSpeedLeft);
  analogWrite(MOTOR_RIGHT, motorSpeedRight);
}
```

Este algoritmo:

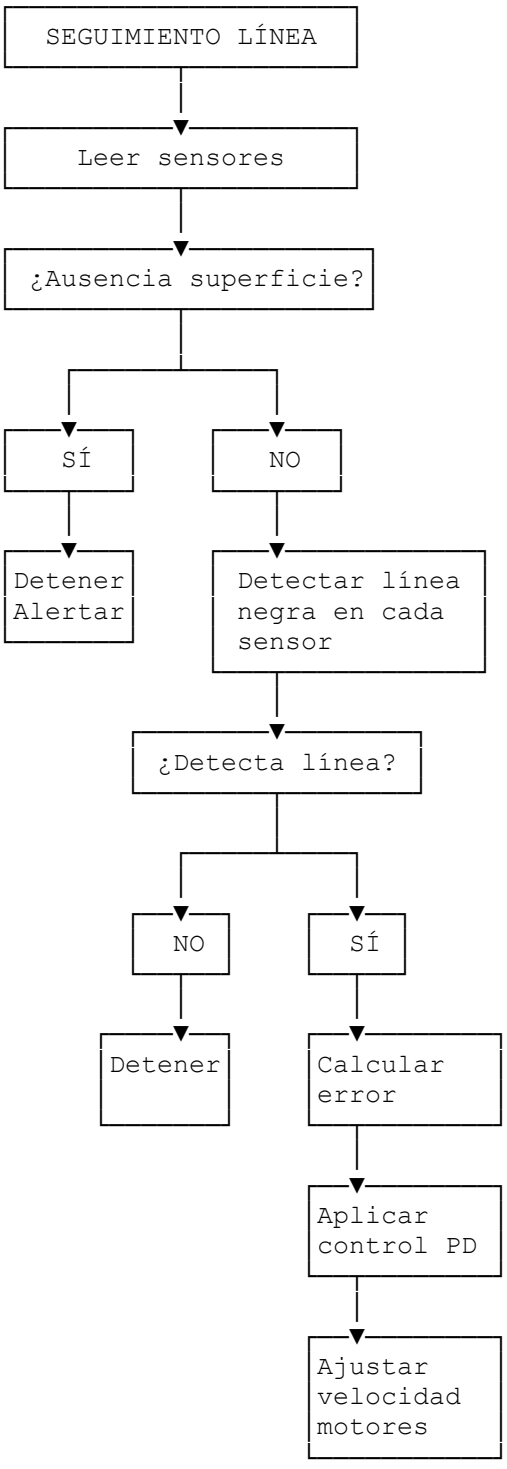
1. Calcula un error ponderado basado en qué sensores detectan la línea
2. Aplica una corrección proporcional (P) e incorpora el factor derivativo (D)
3. Ajusta las velocidades de los motores para corregir la trayectoria
4. Incluye manejo especial para curvas cerradas

6. Diagramas de flujo

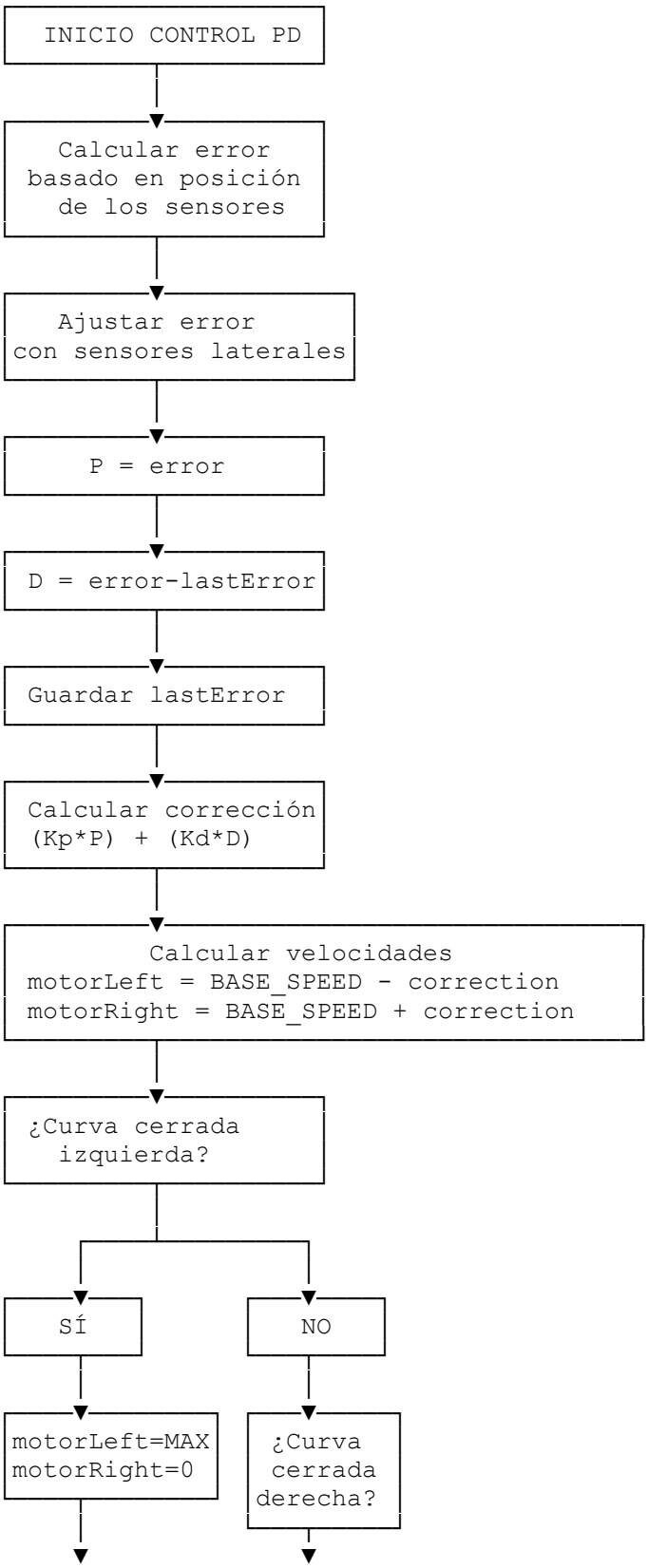
6.1 Diagrama de flujo principal

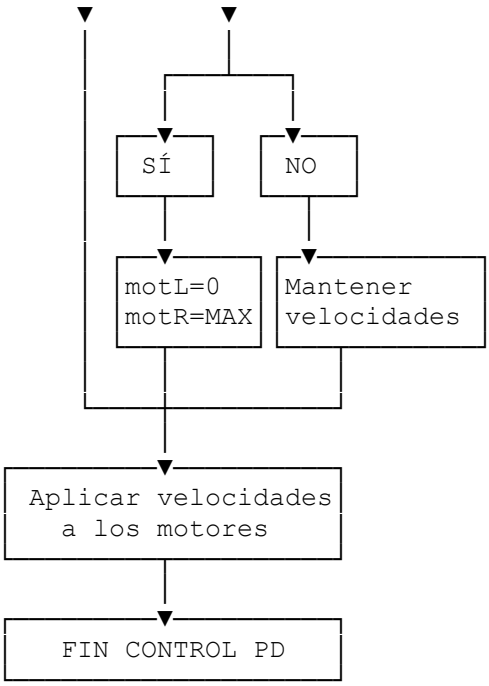


6.2 Diagrama de seguimiento de línea



6.3 Diagrama de control PD





7. Pruebas y resultados

7.1 Mediciones de rendimiento

Durante las pruebas se realizaron mediciones de varios parámetros clave:

Parámetro	Valor medido	Valor esperado	Comentarios
Consumo en reposo	45mA	<50mA	Cumple especificaciones
Consumo en movimiento	280mA	<300mA	Picos de hasta 350mA en giros
Velocidad máxima	25cm/s	30cm/s	Limitada para mayor precisión
Duración batería	1.5h	>1h	Con batería de 1000mAh

7.2 Resultados de calibración

Los resultados de calibración muestran la adaptabilidad del sistema en diferentes condiciones:

Condición	Lecturas blanco	Lecturas negro	Umbral calculado
Luz alta	120-150	800-850	~485
Luz media	200-250	750-800	~500
Luz baja	300-350	650-700	~525

8. Código fuente

```
/*
 * ROBOT SEGUIDOR DE LÍNEAS NEGRAS
 * ------------------------------
 * Este programa controla un robot seguidor de líneas con un proceso de calibración
 * en tres pasos y detección   inteligente tanto de línea negra como de  ausencia  de
* superficie.
 *
 * Autor: Fernández Hernán Gustavo
 * Última actualización: Abril 2025
 */

//--------------------------------------------------------------------------------
// DEFINICIÓN DE PINES
//--------------------------------------------------------------------------------

// Sensores de línea
#define S1            A2  // Sensor izquierda extrema
#define S2            A1  // Sensor izquierda media
#define S3            A3  // Sensor derecha media
#define S4            A4  // Sensor derecha extrema
#define SL            A0  // Sensor lateral izquierdo
#define SR            A5  // Sensor lateral derecho

// Control
#define SWITCH        12  // Pulsador de inicio
#define LED           11  // LED indicador de estado

// Motores
#define MOTOR_RIGHT   10  // Motor derecho (PWM)
#define MOTOR_LEFT    9   // Motor izquierdo (PWM)


//--------------------------------------------------------------------------------
// CONFIGURACIÓN DEL CONTROLADOR PID
//--------------------------------------------------------------------------------

float Kp = 35.0;          // Constante proporcional
float Kd = 15.0;          // Constante derivativa
float error = 0;          // Error actual
float lastError = 0;      // Error anterior
int BASE_SPEED = 100;     // Velocidad base
int MAX_SPEED = 255;      // Velocidad máxima
```

```
//--------------------------------------------------------------------------------
// VALORES DE CALIBRACIÓN
//--------------------------------------------------------------------------------

// Umbrales calculados
int umbralS1 = 500;       // Umbral para sensor S1
int umbralS2 = 500;       // Umbral para sensor S2
int umbralS3 = 500;       // Umbral para sensor S3
int umbralS4 = 500;       // Umbral para sensor S4
int umbralSL = 500;       // Umbral para sensor SL
int umbralSR = 500;       // Umbral para sensor SR

// Lecturas calibradas para superficie blanca
int blanco1, blanco2, blanco3, blanco4, blancoL, blancoR;

// Lecturas calibradas para línea negra
int negro1, negro2, negro3, negro4, negroL, negroR;



//--------------------------------------------------------------------------------
// ESTADOS DEL ROBOT
//--------------------------------------------------------------------------------

#define ESPERA 0              // Esperando primera pulsación para calibrar fondo
blanco
#define CALIBRADO_BLANCO 1    // Calibrado fondo blanco, esperando pulsación para
calibrar línea negra
#define CALIBRADO_NEGRO 2     // Calibrado completo, esperando pulsación para iniciar
#define ACTIVO 3              // Robot en funcionamiento

byte estadoRobot = ESPERA;    // Estado inicial
bool buttonPressed = false;   // Estado del botón



//--------------------------------------------------------------------------------
// CONFIGURACIÓN INICIAL
//--------------------------------------------------------------------------------

void setup() {
  // Configurar pines
  configurarPines();

  // Iniciar comunicación serial
  Serial.begin(9600);

  // Mostrar mensaje de bienvenida
  mostrarInstrucciones();

  // Parpadeo inicial para indicar que el sistema está listo
  parpadearLED(3, 100);
}
```

```
//--------------------------------------------------------------------------------
// BUCLE PRINCIPAL
//--------------------------------------------------------------------------------

void loop() {
  // Gestionar pulsador
  gestionarBoton();

  // Actuar según el estado actual del robot
  switch (estadoRobot) {
    case ESPERA:
      // Esperando calibrar fondo blanco - parpadeo lento
      apagarMotores();
      digitalWrite(LED, (millis() / 500) % 2);
      break;

    case CALIBRADO_BLANCO:
      // Esperando calibrar línea negra - parpadeo medio
      apagarMotores();
      digitalWrite(LED, (millis() / 250) % 2);
      break;

    case CALIBRADO_NEGRO:
      // Calibrado completo, esperando inicio - parpadeo rápido
      apagarMotores();
      digitalWrite(LED, (millis() / 100) % 2);
      break;

    case ACTIVO:
      // Robot en funcionamiento
      seguirLinea();
      break;
  }


//--------------------------------------------------------------------------------
// FUNCIONES DE CONFIGURACIÓN
//--------------------------------------------------------------------------------

/**
 * Configura todos los pines necesarios para el funcionamiento del robot
 */

void configurarPines() {
  // Configurar pines de sensores como entradas
  pinMode(S1, INPUT);
  pinMode(S2, INPUT);
  pinMode(S3, INPUT);
  pinMode(S4, INPUT);
  pinMode(SL, INPUT);
  pinMode(SR, INPUT);

  // Configurar pulsador con resistencia pull-up interna
  pinMode(SWITCH, INPUT_PULLUP);

  // Configurar LED y motores como salidas
  pinMode(LED, OUTPUT);
  pinMode(MOTOR_RIGHT, OUTPUT);
  pinMode(MOTOR_LEFT, OUTPUT);

  // Asegurar que los motores estén apagados al inicio
  apagarMotores();
}
```

```
/**
 * Muestra las instrucciones iniciales en el monitor serie
 */

void mostrarInstrucciones() {
  Serial.println(F("==== ROBOT SEGUIDOR DE LINEAS ===="));
  Serial.println(F("Calibracion en 3 pasos:"));
  Serial.println(F("1. Primera pulsacion: Calibrar FONDO BLANCO"));
  Serial.println(F("2. Segunda pulsacion: Calibrar LINEA NEGRA"));
  Serial.println(F("3. Tercera pulsacion: INICIAR seguimiento"));
  Serial.println(F("================================="));
}


//--------------------------------------------------------------------------------
// FUNCIONES DE CONTROL
//--------------------------------------------------------------------------------

/**
 * Gestiona el pulsador y cambia el estado del robot según la secuencia
 */
void gestionarBoton() {
  bool buttonState = digitalRead(SWITCH);
  // Detectar pulsación (flanco descendente)
  if (buttonState == LOW && !buttonPressed) {
    buttonPressed = true;
  }
  // Detectar liberación (flanco ascendente)
  else if (buttonState == HIGH && buttonPressed) {
    buttonPressed = false;
    // Cambiar estado según secuencia
    switch (estadoRobot) {
      case ESPERA:
        // Primera pulsación - Calibrar fondo blanco
        Serial.println(F("Calibrando FONDO BLANCO..."));
        calibrarFondoBlanco();
        estadoRobot = CALIBRADO_BLANCO;
        Serial.println(F("FONDO BLANCO calibrado. Presione boton para calibrar LINEA
NEGRA"));
        break;

      case CALIBRADO_BLANCO:
        // Segunda pulsación - Calibrar línea negra
        Serial.println(F("Calibrando LINEA NEGRA..."));
        calibrarLineaNegra();
        calcularUmbrales();
        estadoRobot = CALIBRADO_NEGRO;
        Serial.println(F("LINEA NEGRA calibrada. Presione boton para INICIAR"));
        break;

      case CALIBRADO_NEGRO:
        // Tercera pulsación - Activar
        estadoRobot = ACTIVO;
        Serial.println(F("Robot ACTIVADO"));
        break;

      case ACTIVO:
        // Cuarta pulsación - Detener y volver a espera calibrado
        estadoRobot = CALIBRADO_NEGRO;
        apagarMotores();
        Serial.println(F("Robot DETENIDO. Presione boton para reactivar"));
        break;
    }
    delay(50); // anti-rebote
  }
}
```

```
/**
 * Apaga ambos motores
 */

void apagarMotores() {
  analogWrite(MOTOR_LEFT, 0);
  analogWrite(MOTOR_RIGHT, 0);
}




/**
 * Hace parpadear el LED un número específico de veces
 * @param veces Número de veces a parpadear
 * @param duracion Duración de cada parpadeo en ms
 */

void parpadearLED(byte veces, int duracion) {
  for (byte i = 0; i < veces; i++) {
    digitalWrite(LED, HIGH);
    delay(duracion);
    digitalWrite(LED, LOW);
    delay(duracion);
  }
}




/**
 * Espera un tiempo determinado mientras parpadea el LED
 * @param ms Tiempo a esperar en ms
 */

void esperar(int ms) {
  unsigned long inicio = millis();
  while (millis() - inicio < ms) {
    digitalWrite(LED, (millis() / 100) % 2);
    delay(10);
  }
}
```

```
//--------------------------------------------------------------------------------
// FUNCIONES DE CALIBRACIÓN
//--------------------------------------------------------------------------------


/**
 * Calibra el robot para el fondo blanco
 */


void calibrarFondoBlanco() {
  // Indicación visual de inicio de calibración
  parpadearLED(3, 50);

  // Tomar múltiples muestras y promediar
  byte muestras = 10;
  int suma1 = 0, suma2 = 0, suma3 = 0, suma4 = 0, sumaL = 0, sumaR = 0;

  for (byte i = 0; i < muestras; i++) {
    suma1 += analogRead(S1);
    suma2 += analogRead(S2);
    suma3 += analogRead(S3);
    suma4 += analogRead(S4);
    sumaL += analogRead(SL);
    sumaR += analogRead(SR);
    digitalWrite(LED, !digitalRead(LED)); // Parpadeo durante la medición
    delay(10);
  }


// Calcular promedios
  blanco1 = suma1 / muestras;
  blanco2 = suma2 / muestras;
  blanco3 = suma3 / muestras;
  blanco4 = suma4 / muestras;
  blancoL = sumaL / muestras;
  blancoR = sumaR / muestras;

  // Mostrar valores leídos
  Serial.println(F("Valores FONDO BLANCO:"));
  Serial.print(F("S1: ")); Serial.print(blanco1);
  Serial.print(F(" | S2: ")); Serial.print(blanco2);
  Serial.print(F(" | S3: ")); Serial.print(blanco3);
  Serial.print(F(" | S4: ")); Serial.println(blanco4);
  Serial.print(F("SL: ")); Serial.print(blancoL);
  Serial.print(F(" | SR: ")); Serial.println(blancoR);
}
```

```
/**
 * Calibra el robot para la línea negra
 */


void calibrarLineaNegra() {
  // Indicación visual de inicio de calibración
  parpadearLED(3, 50);

  // Tomar múltiples muestras y promediar
  byte muestras = 10;
  int suma1 = 0, suma2 = 0, suma3 = 0, suma4 = 0, sumaL = 0, sumaR = 0;

  for (byte i = 0; i < muestras; i++) {
    suma1 += analogRead(S1);
    suma2 += analogRead(S2);
    suma3 += analogRead(S3);
    suma4 += analogRead(S4);
    sumaL += analogRead(SL);
    sumaR += analogRead(SR);
    digitalWrite(LED, !digitalRead(LED)); // Parpadeo durante la medición
    delay(10);
  }

  // Calcular promedios
  negro1 = suma1 / muestras;
  negro2 = suma2 / muestras;
  negro3 = suma3 / muestras;
  negro4 = suma4 / muestras;
  negroL = sumaL / muestras;
  negroR = sumaR / muestras;

  // Mostrar valores leídos
  Serial.println(F("Valores LINEA NEGRA:"));
  Serial.print(F("S1: ")); Serial.print(negro1);
  Serial.print(F(" | S2: ")); Serial.print(negro2);
  Serial.print(F(" | S3: ")); Serial.print(negro3);
  Serial.print(F(" | S4: ")); Serial.println(negro4);
  Serial.print(F("SL: ")); Serial.print(negroL);
  Serial.print(F(" | SR: ")); Serial.println(negroR);
}
```

```
/**
 * Calcula los umbrales óptimos basados en las lecturas de calibración
 */

void calcularUmbrales() {
  // Calcular umbrales como punto medio entre blanco y negro
  umbralS1 = (blanco1 + negro1) / 2;
  umbralS2 = (blanco2 + negro2) / 2;
  umbralS3 = (blanco3 + negro3) / 2;
  umbralS4 = (blanco4 + negro4) / 2;
  umbralSL = (blancoL + negroL) / 2;
  umbralSR = (blancoR + negroR) / 2;

  // Determinar para cada sensor si el negro da valores más altos o más bajos que el
blanco
  bool negroMasAltoS1 = (negro1 > blanco1);
  bool negroMasAltoS2 = (negro2 > blanco2);
  bool negroMasAltoS3 = (negro3 > blanco3);
  bool negroMasAltoS4 = (negro4 > blanco4);
  bool negroMasAltoSL = (negroL > blancoL);
  bool negroMasAltoSR = (negroR > blancoR);

  // Mostrar umbrales y comportamiento de sensores
  Serial.println(F("\nUmbrales calculados:"));
  Serial.print(F("S1: "));
  Serial.print(umbralS1);
  Serial.print(negroMasAltoS1 ? F(" (Negro>Blanco)") : F(" (Blanco>Negro)"));

  Serial.print(F(" | S2: "));
  Serial.print(umbralS2);
  Serial.print(negroMasAltoS2 ? F(" (Negro>Blanco)") : F(" (Blanco>Negro)"));

  Serial.print(F(" | S3: "));
  Serial.print(umbralS3);
  Serial.print(negroMasAltoS3 ? F(" (Negro>Blanco)") : F(" (Blanco>Negro)"));

  Serial.print(F(" | S4: "));
  Serial.print(umbralS4);
  Serial.println(negroMasAltoS4 ? F(" (Negro>Blanco)") : F(" (Blanco>Negro)"));

  Serial.print(F("SL: "));
  Serial.print(umbralSL);
  Serial.print(negroMasAltoSL ? F(" (Negro>Blanco)") : F(" (Blanco>Negro)"));

  Serial.print(F(" | SR: "));
  Serial.print(umbralSR);
  Serial.println(negroMasAltoSR ? F(" (Negro>Blanco)") : F(" (Blanco>Negro)"));
}
```

```
//--------------------------------------------------------------------------------
// FUNCIONES DE SEGUIMIENTO DE LÍNEA
//--------------------------------------------------------------------------------

/**
 * Función principal de seguimiento de línea
 */

void seguirLinea() {
  digitalWrite(LED, HIGH);  // LED encendido durante operación normal

  // Leer valores analógicos de los sensores
  int valorS1 = analogRead(S1);
  int valorS2 = analogRead(S2);
  int valorS3 = analogRead(S3);
  int valorS4 = analogRead(S4);
  int valorSL = analogRead(SL);
  int valorSR = analogRead(SR);

  // Verificar si se detecta superficie
  if (verificarAusenciaSuperficie(valorS1, valorS2, valorS3, valorS4, valorSL,
valorSR)) {
    return;  // Salir si no se detecta superficie
  }

  // Determinar si cada sensor está detectando línea negra
  // La comparación es dinámica según los valores calibrados
  byte s1 = (valorS1 > umbralS1) == (negro1 > blanco1) ? 1 : 0;
  byte s2 = (valorS2 > umbralS2) == (negro2 > blanco2) ? 1 : 0;
  byte s3 = (valorS3 > umbralS3) == (negro3 > blanco3) ? 1 : 0;
  byte s4 = (valorS4 > umbralS4) == (negro4 > blanco4) ? 1 : 0;
  byte sl = (valorSL > umbralSL) == (negroL > blancoL) ? 1 : 0;
  byte sr = (valorSR > umbralSR) == (negroR > blancoR) ? 1 : 0;

  // Verificar si algún sensor detecta línea negra
  bool detectaLineaNegra = (s1 || s2 || s3 || s4 || sl || sr);

  // Mostrar estado de los sensores cada 100 iteraciones
  static int contador = 0;
  if (++contador >= 100) {
    contador = 0;
    Serial.print(F("Sensores: "));
    Serial.print(s1); Serial.print(s2); Serial.print(s3); Serial.print(s4);
    Serial.print(F(" SL:")); Serial.print(sl);
    Serial.print(F(" SR:")); Serial.println(sr);
  }

  // Si no detecta línea negra, detener
  if (!detectaLineaNegra) {
    apagarMotores();
    digitalWrite(LED, LOW);
    Serial.println(F("LINEA NEGRA PERDIDA - Robot detenido"));
    // Parpadear LED rápidamente para indicar detección
    parpadearLED(5, 100);
    return;
  }

  // Procesa los datos para seguir la línea
  procesarSeguimientoLinea(s1, s2, s3, s4, sl, sr);
}
```

```
/**
 * Verifica si hay ausencia de superficie debajo del robot
 * @return true si no se detecta superficie, false en caso contrario
 */

bool verificarAusenciaSuperficie(int valorS1, int valorS2, int valorS3, int valorS4,
int valorSL, int valorSR) {
  // Definir umbrales para "no superficie" con un margen del 20%
  int margen = 20; // en porcentaje
  bool noHaySuperficie = false;

  // Verificar cada sensor para valores erroneos
  if (valorS1 < min(blanco1, negro1) - (abs(blanco1 - negro1) * margen / 100) ||
      valorS1 > max(blanco1, negro1) + (abs(blanco1 - negro1) * margen / 100)) {
    noHaySuperficie = true;
  }

  if (valorS2 < min(blanco2, negro2) - (abs(blanco2 - negro2) * margen / 100) ||
      valorS2 > max(blanco2, negro2) + (abs(blanco2 - negro2) * margen / 100)) {
    noHaySuperficie = true;
  }

  if (valorS3 < min(blanco3, negro3) - (abs(blanco3 - negro3) * margen / 100) ||
      valorS3 > max(blanco3, negro3) + (abs(blanco3 - negro3) * margen / 100)) {
    noHaySuperficie = true;
  }

  if (valorS4 < min(blanco4, negro4) - (abs(blanco4 - negro4) * margen / 100) ||
      valorS4 > max(blanco4, negro4) + (abs(blanco4 - negro4) * margen / 100)) {
    noHaySuperficie = true;
  }

  if (valorSL < min(blancoL, negroL) - (abs(blancoL - negroL) * margen / 100) ||
      valorSL > max(blancoL, negroL) + (abs(blancoL - negroL) * margen / 100)) {
    noHaySuperficie = true;
  }

  if (valorSR < min(blancoR, negroR) - (abs(blancoR - negroR) * margen / 100) ||
      valorSR > max(blancoR, negroR) + (abs(blancoR - negroR) * margen / 100)) {
    noHaySuperficie = true;
  }

  // Si no hay superficie, alertar y detener
  if (noHaySuperficie) {
    apagarMotores();

    // Parpadeo - tres parpadeos muy rápidos
    parpadearLED(3, 50);

    Serial.println(F("ALERTA! No se detecta superficie - Robot detenido"));
    return true;
  }

  return false;
}
```

```
/**
 * Procesa los datos de los sensores y controla los motores para seguir la línea
 */

void procesarSeguimientoLinea(byte s1, byte s2, byte s3, byte s4, byte sl, byte sr) {
  // Cálculo del error para línea negra
  error = (s1 * -3) + (s2 * -1) + (s3 * 1) + (s4 * 3);

  // Ajuste con sensores laterales
  if (sl) error -= 5;  // SL produce error negativo (giro a la izquierda)
  if (sr) error += 5;  // SR produce error positivo (giro a la derecha)

  // Control PD
  float P = error;
  float D = error - lastError;
  lastError = error;

  float correction = (Kp * P) + (Kd * D);

  // Ajustar velocidad de motores según la corrección calculada
  int motorSpeedLeft = constrain(BASE_SPEED - correction, 0, MAX_SPEED);
  int motorSpeedRight = constrain(BASE_SPEED + correction, 0, MAX_SPEED);

  // Control para curvas cerradas detectadas con sensores laterales
  if (sl && !s1 && !s2) {
    // Curva cerrada izquierda
    motorSpeedLeft = MAX_SPEED;
    motorSpeedRight = 0;
  }

  if (sr && !s3 && !s4) {
    // Curva cerrada derecha
    motorSpeedLeft = 0;
    motorSpeedRight = MAX_SPEED;
  }

  // Aplicar velocidades a los motores
  analogWrite(MOTOR_LEFT, motorSpeedLeft);
  analogWrite(MOTOR_RIGHT, motorSpeedRight);

  delay(10); // pausa para estabilidad
}
```

9. Repositorio del proyecto

Todo el código fuente, esquemas electrónicos y documentación adicional de este proyecto están disponibles en el siguiente repositorio de GitHub:

Referencia

1. Fernández, Hernán Gustavo (2025). *Robot seguidor de línea con Arduino Nano*. GitHub: https://github.com/gustavofernandez/Robot-seguidor-de-linea