DEMO SEGUIDOR DE LINEA

```
#include <QTRSensors.h>

// ====================================================
//          DEFINICIONES DE PINES Y CONSTANTES
// ====================================================
#define NUM_SENSORS   6
#define TIMEOUT       2500
#define EMITTER_PIN   14       // Emisor de los sensores (LEDON)
#define THRESHOLD     850      // Ajusta según contraste
#define BOTON         12       // Botón para iniciar la calibración
#define TRIG_PIN      13       // Pin TRIG del sensor ultrasónico (HC-SR04)
#define ECHO_PIN      2        // Pin ECHO del sensor ultrasónico
#define LED_VERDE     27       // LED indicador de funcionamiento

// --- Pines para el L298N ---
#define MOTOR_1_PWM   22       // PWM Motor 1 (ENA)
#define MOTOR_2_PWM   15       // PWM Motor 2 (ENB)
#define MOTOR_1_DIR1  19       // Dirección Motor 1 (IN1)
#define MOTOR_1_DIR2  21       // Dirección Motor 1 (IN2)
#define MOTOR_2_DIR1  4        // Dirección Motor 2 (IN3)
#define MOTOR_2_DIR2  5        // Dirección Motor 2 (IN4)
// MOTOR 1: derecha, MOTOR 2: izquierda

// ====================================================
//                VARIABLES GLOBALES
// ====================================================
const uint8_t sensorPins[NUM_SENSORS] = {33, 32, 35, 34, 36, 39};
QTRSensors qtr;
uint16_t sensorValues[NUM_SENSORS];

enum RobotState { NORMAL, SEARCH };
RobotState state = NORMAL;

// Variables para el modo SEARCH (búsqueda de línea)
unsigned long searchStartTime = 0;
unsigned long previousBlinkTime = 0;
bool ledState = false;

// ====================================================
//          FUNCIONES AUXILIARES DE MOTORES
// ====================================================

// Mueve ambos motores hacia adelante a velocidades diferenciadas
void moveForward(int speedRight, int speedLeft) {
  // Motor 1 (derecho)
  digitalWrite(MOTOR_1_DIR1, HIGH);
  digitalWrite(MOTOR_1_DIR2, LOW);
  analogWrite(MOTOR_1_PWM, speedRight);

  // Motor 2 (izquierdo)
  digitalWrite(MOTOR_2_DIR1, HIGH);
  digitalWrite(MOTOR_2_DIR2, LOW);
  analogWrite(MOTOR_2_PWM, speedLeft);
}

// Gira hacia la izquierda: activa el motor derecho y detiene el motor
izquierdo.
```

```
void turnLeft(int speed) {
  // Motor derecho activo para avanzar
  digitalWrite(MOTOR_1_DIR1, HIGH);
  digitalWrite(MOTOR_1_DIR2, LOW);
  analogWrite(MOTOR_1_PWM, speed);

  // Motor izquierdo detenido
  digitalWrite(MOTOR_2_DIR1, LOW);
  digitalWrite(MOTOR_2_DIR2, LOW);
  analogWrite(MOTOR_2_PWM, 0);
}

// Gira hacia la derecha: activa el motor izquierdo y detiene el motor
derecho.
void turnRight(int speed) {
  // Motor izquierdo activo para avanzar
  digitalWrite(MOTOR_2_DIR1, HIGH);
  digitalWrite(MOTOR_2_DIR2, LOW);
  analogWrite(MOTOR_2_PWM, speed);

  // Motor derecho detenido
  digitalWrite(MOTOR_1_DIR1, LOW);
  digitalWrite(MOTOR_1_DIR2, LOW);
  analogWrite(MOTOR_1_PWM, 0);
}

// Detiene ambos motores.
void stopMotors() {
  analogWrite(MOTOR_1_PWM, 0);
  analogWrite(MOTOR_2_PWM, 0);
}

// ======================================================
//      FUNCIONES DE CALIBRACIÓN Y ESPERA DEL BOTÓN
// ======================================================

// Espera activa con debounce para el botón.
void waitForButtonPress() {
  Serial.println("Esperando a que se presione el botón...");
  while (digitalRead(BOTON) == HIGH) {
    delay(50);
  }
  delay(200);
  while (digitalRead(BOTON) == LOW) {
    delay(50);
  }
  delay(200);
  Serial.println("¡Botón pulsado!");
}

// Calibración automática con movimientos cortos.
void autoCalibrateSensors() {
  waitForButtonPress();
  Serial.println("Iniciando calibración automática con movimientos
cortos...");

  // Parpadea el LED para indicar inicio de calibración.
  for (int blink = 0; blink < 3; blink++) {
    digitalWrite(LED_VERDE, HIGH);
```

```
    delay(100);
    digitalWrite(LED_VERDE, LOW);
    delay(100);
  }

  // Realiza 8 ciclos (0.5 s cada uno; total 4 s) alternando direcciones.
  for (int cycle = 0; cycle < 8; cycle++) {
    if (cycle % 2 == 0) {
      // Giro: Motor 1 adelante, Motor 2 atrás.
      digitalWrite(MOTOR_1_DIR1, HIGH);
      digitalWrite(MOTOR_1_DIR2, LOW);
      digitalWrite(MOTOR_2_DIR1, LOW);
      digitalWrite(MOTOR_2_DIR2, HIGH);
    } else {
      // Giro inverso: Motor 1 atrás, Motor 2 adelante.
      digitalWrite(MOTOR_1_DIR1, LOW);
      digitalWrite(MOTOR_1_DIR2, HIGH);
      digitalWrite(MOTOR_2_DIR1, HIGH);
      digitalWrite(MOTOR_2_DIR2, LOW);
    }
    analogWrite(MOTOR_1_PWM, 80);
    analogWrite(MOTOR_2_PWM, 80);

    // Durante cada ciclo se calibran los sensores.
    for (int i = 0; i < 30; i++) {
      qtr.calibrate();
      delay(20);
    }
  }

  // Detiene los motores y regresa a la posición inicial.
  stopMotors();
  Serial.println("Calibración automática completada. Robot en posición
inicial.");

  // Indica finalización de calibración con LED encendido 1 s.
  digitalWrite(LED_VERDE, HIGH);
  delay(1000);
  digitalWrite(LED_VERDE, LOW);
  delay(2000);
}

// ======================================================
//         FUNCIONES PARA EL SENSOR ULTRASÓNICO
// ======================================================

// Mide la distancia en cm usando el HC-SR04.
long measureDistance() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  long duration = pulseIn(ECHO_PIN, HIGH);
  long distance = (duration / 2) / 29.1; // Conversión a centímetros
  return distance;
}

bool isObstacleDetected() {
```

```
  long distance = measureDistance();
  Serial.print("Distancia: ");
  Serial.println(distance);
  return (distance < 15);
}

// ====================================================
//       FUNCIONES DE CONTROL DISCRETO DEL SEGUIDOR
// ====================================================

// Controla los motores según combinaciones discretas de sensores.
// Asignación de sensores:
// S2 --> sensorValues[0] (derecha)
// S3 --> sensorValues[1] (adyacente a la derecha)
// S4 --> sensorValues[2] (centro)
// S5 --> sensorValues[3] (centro)
// S6 --> sensorValues[4] (adyacente a la izquierda)
// S7 --> sensorValues[5] (izquierda)
void controlMotors() {
  // Caso ideal "Centro": los sensores centrales (S[2] y S[3]) activados y
los extremos (S[0] y S[5]) sin señal.
  if (sensorValues[2] > THRESHOLD && sensorValues[3] > THRESHOLD &&
      sensorValues[0] < THRESHOLD && sensorValues[5] < THRESHOLD) {
    Serial.println("Centro: línea detectada en sensores 4 y 5. Avanzando en
línea recta.");
    moveForward(150, 150);
  }
  // Ajuste leve a la derecha:
  // Si los sensores centrales están activos pero además S[1] (lado derecho,
adyacente a S[2]) muestra un valor moderado,
  // se interpreta que la línea se ha desplazado levemente hacia la derecha.
  else if (sensorValues[2] > THRESHOLD && sensorValues[3] > THRESHOLD &&
           sensorValues[1] > THRESHOLD*0.8) {
    Serial.println("Ajuste leve a la derecha.");
    // Para corregir hacia la derecha, se puede aumentar la velocidad del
MOTOR 1 (derecho)
    // y reducir la del MOTOR 2 (izquierdo).
    moveForward(170, 120);
  }
  // Ajuste leve a la izquierda:
  // Si los sensores centrales están activos pero S[4] (lado izquierdo,
adyacente a S[3]) muestra un valor moderado,
  // se interpreta que la línea se ha desplazado levemente hacia la
izquierda.
  else if (sensorValues[2] > THRESHOLD && sensorValues[3] > THRESHOLD &&
           sensorValues[4] > THRESHOLD*0.8) {
    Serial.println("Ajuste leve a la izquierda.");
    // Para corregir hacia la izquierda, se aumenta la velocidad del MOTOR 2
(izquierdo)
    // y se reduce la del MOTOR 1 (derecho).
    moveForward(120, 170);
  }
  // Lado Izquierdo (extremo): Si S[0] está activado (o S[0] y S[1] en
exceso) indica que la línea está muy desplazada a la derecha.
  else if (sensorValues[0] > THRESHOLD) {
    Serial.println("Extremo Izquierdo: giro brusco a la derecha (activando
MOTOR 2).");
    turnRight(190);
  }
```

```
  // Lado Derecho (extremo): Si S[5] está activado, indica que la línea está
muy desplazada a la izquierda.
  else if (sensorValues[5] > THRESHOLD) {
    Serial.println("Extremo Derecho: giro brusco a la izquierda (activando
MOTOR 1).");
    turnLeft(190);
  }
  else {
    Serial.println("Condición no definida. Deteniendo motores.");
    stopMotors();
    digitalWrite(LED_VERDE, LOW);
  }
}
// =====================================================
//           FUNCIONES PARA EL MODO SEARCH
// =====================================================

// En modo SEARCH se busca la línea perdida mediante giros predefinidos y
parpadeo del LED.
void searchForLine() {
  unsigned long currentTime = millis();
  unsigned long elapsed = currentTime - searchStartTime;

  // Parpadeo no bloqueante del LED cada 50 ms.
  if (currentTime - previousBlinkTime >= 50) {
    previousBlinkTime = currentTime;
    ledState = !ledState;
    digitalWrite(LED_VERDE, ledState ? HIGH : LOW);
  }

  // Durante los primeros 1.5 s, gira en una dirección (por ejemplo, Motor 1
adelante, Motor 2 atrás).
  if (elapsed < 1500) {
    digitalWrite(MOTOR_1_DIR1, HIGH);
    digitalWrite(MOTOR_1_DIR2, LOW);
    digitalWrite(MOTOR_2_DIR1, LOW);
    digitalWrite(MOTOR_2_DIR2, HIGH);
    analogWrite(MOTOR_1_PWM, 100);
    analogWrite(MOTOR_2_PWM, 100);
  }
  // Durante los siguientes 1.5 s, gira en dirección opuesta (Motor 1 atrás,
Motor 2 adelante).
  else if (elapsed < 3000) {
    digitalWrite(MOTOR_1_DIR1, LOW);
    digitalWrite(MOTOR_1_DIR2, HIGH);
    digitalWrite(MOTOR_2_DIR1, HIGH);
    digitalWrite(MOTOR_2_DIR2, LOW);
    analogWrite(MOTOR_1_PWM, 100);
    analogWrite(MOTOR_2_PWM, 100);
  }
  // Si han transcurrido 3 s sin recuperar la línea, detener el robot.
  else {
    Serial.println("Búsqueda agotada, deteniendo motores.");
    stopMotors();
    digitalWrite(LED_VERDE, LOW);
  }
}

// =====================================================
```

```
//                          SETUP
// ====================================================
void setup() {
  Serial.begin(115200);
  delay(500);

  // Configuración de sensores QTR.
  qtr.setTypeAnalog();
  qtr.setSensorPins(sensorPins, NUM_SENSORS);
  qtr.setTimeout(TIMEOUT);
  qtr.setEmitterPin(EMITTER_PIN);

  // Configuración de pines de E/S.
  pinMode(BOTON, INPUT_PULLUP);
  pinMode(EMITTER_PIN, OUTPUT);
  pinMode(LED_VERDE, OUTPUT);

  // Configuración de pines de los motores.
  pinMode(MOTOR_1_PWM, OUTPUT);
  pinMode(MOTOR_2_PWM, OUTPUT);
  pinMode(MOTOR_1_DIR1, OUTPUT);
  pinMode(MOTOR_1_DIR2, OUTPUT);
  pinMode(MOTOR_2_DIR1, OUTPUT);
  pinMode(MOTOR_2_DIR2, OUTPUT);

  // Configuración de pines del sensor ultrasónico.
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  Serial.println("Presiona el botón para iniciar el programa.");
  waitForButtonPress();
  autoCalibrateSensors();

  Serial.println("Inicio del programa. Modo NORMAL.");
}

// ====================================================
//                          LOOP
// ====================================================
void loop() {
  // Actualizar lecturas de sensores calibrados.
  qtr.readCalibrated(sensorValues);

  // Depuración: imprimir valores de los sensores.
  for (int i = 0; i < NUM_SENSORS; i++) {
    Serial.print(sensorValues[i]);
    Serial.print("\t");
  }
  Serial.println();

  // Verificar obstáculo: prioridad máxima.
  if (isObstacleDetected()) {
    Serial.println("Obstáculo detectado, deteniendo motores.");
    stopMotors();
    digitalWrite(LED_VERDE, LOW);
    delay(100);
    return;
  }
```

```
  // Contar cuántos sensores detectan "negro".
  int blackCount = 0;
  for (int i = 0; i < NUM_SENSORS; i++) {
    if (sensorValues[i] > THRESHOLD) {
      blackCount++;
    }
  }

  // ----------------- CONDICIONES DE SEGURIDAD EN MODO NORMAL --------------

  if (state == NORMAL) {
    if (blackCount > 5) {
      Serial.println("Todos los sensores detectan negro, deteniendo
motores.");
      stopMotors();
      digitalWrite(LED_VERDE, LOW);
      delay(100);
      return;
    }
    if (blackCount == 0) {
      Serial.println("Todos los sensores detectan blanco, deteniendo
motores.");
      stopMotors();
      digitalWrite(LED_VERDE, LOW);
      delay(100);
      return;
    }
  }

  // ----------------------- GESTIÓN DEL ESTADO -----------------------
  // Si en modo NORMAL se detecta que menos de 2 sensores muestran negro, se
asume pérdida de línea.
  if (state == NORMAL && blackCount < 2) {
    state = SEARCH;
    searchStartTime = millis();
    Serial.println("Cambio a modo SEARCH.");
  }

  // Si en modo SEARCH se recupera la línea (blackCount >= 2), volver a
NORMAL.
  if (state == SEARCH && blackCount >= 2) {
    state = NORMAL;
    Serial.println("Línea recuperada, cambiando a modo NORMAL.");
  }

  // ----------------------- EJECUCIÓN SEGÚN ESTADO -----------------------
  if (state == NORMAL) {
    digitalWrite(LED_VERDE, HIGH);  // LED fijo encendido en NORMAL.
    controlMotors();
  } else if (state == SEARCH) {
    searchForLine();
  }

  delay(10);
}
```