Código para Zumo con Arduino UNO y sensores Sharp + QTR-8

Aquí tienes un código completo para controlar tu Zumo con Arduino UNO, utilizando 3 sensores Sharp (frontal y laterales), el sensor QTR-8 para línea, y 3 switches para seleccionar el modo de ataque.

cpp Copy

```cpp
#include <ZumoMotors.h>
#include <QTRSensors.h>

// Definición de pines
#define FRONT_SENSOR A0
#define LEFT_SENSOR A1
#define RIGHT_SENSOR A2
#define SWITCH_FRONT 2
#define SWITCH_LEFT 3
#define SWITCH_RIGHT 4

// Umbrales de detección (ajustar según calibración)
#define OBSTACLE_DISTANCE 30  // en cm
#define LINE_THRESHOLD 500    // valor para detectar línea

// Velocidades de los motores
#define MAX_SPEED 400
#define REVERSE_SPEED -400
#define TURN_SPEED 300

ZumoMotors motors;
QTRSensorsRC qtr((unsigned char[]) {5, 6, 7, 8, 9, 10, 11, 12}, 8);

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

  // Configurar pines de los switches como entrada con pull-up
  pinMode(SWITCH_FRONT, INPUT_PULLUP);
  pinMode(SWITCH_LEFT, INPUT_PULLUP);
  pinMode(SWITCH_RIGHT, INPUT_PULLUP);

  // Calibrar sensor de línea
  calibrateLineSensor();
}

void loop() {
  // Leer sensores de distancia
  int frontDistance = readSharpSensor(FRONT_SENSOR);
  int leftDistance = readSharpSensor(LEFT_SENSOR);
  int rightDistance = readSharpSensor(RIGHT_SENSOR);

  // Leer sensor de línea
  unsigned int lineValues[8];
  int linePosition = qtr.readLine(lineValues);
  bool onLine = (lineValues[0] > LINE_THRESHOLD || lineValues[7] > LINE_THRESHOLD);
```

```
  // Leer modo de ataque seleccionado
  bool attackFront = !digitalRead(SWITCH_FRONT);
  bool attackLeft = !digitalRead(SWITCH_LEFT);
  bool attackRight = !digitalRead(SWITCH_RIGHT);

  // Si detecta línea, retroceder
  if (onLine) {
    motors.setSpeeds(REVERSE_SPEED, REVERSE_SPEED);
    delay(500);
    return;
  }

  // Comportamiento según modo de ataque seleccionado
  if (attackFront) {
    frontAttackMode(frontDistance, leftDistance, rightDistance);
  }
  else if (attackLeft) {
    leftAttackMode(frontDistance, leftDistance, rightDistance);
  }
  else if (attackRight) {
    rightAttackMode(frontDistance, leftDistance, rightDistance);
  }
  else {
    // Si no hay ningún modo seleccionado, no hacer nada
    motors.setSpeeds(0, 0);
  }
}

void frontAttackMode(int front, int left, int right) {
  if (front < OBSTACLE_DISTANCE) {
    // Obstáculo frontal - atacar
    motors.setSpeeds(MAX_SPEED, MAX_SPEED);
  }
  else if (left < OBSTACLE_DISTANCE) {
    // Obstáculo a la izquierda - girar ligeramente
    motors.setSpeeds(TURN_SPEED, MAX_SPEED);
  }
  else if (right < OBSTACLE_DISTANCE) {
    // Obstáculo a la derecha - girar ligeramente
    motors.setSpeeds(MAX_SPEED, TURN_SPEED);
  }
  else {
    // No detecta nada - retroceder al máximo
    motors.setSpeeds(REVERSE_SPEED, REVERSE_SPEED);
  }
}

void leftAttackMode(int front, int left, int right) {
  if (left < OBSTACLE_DISTANCE) {
    // Obstáculo a la izquierda - atacar girando
    motors.setSpeeds(-TURN_SPEED, MAX_SPEED);
  }
  else if (front < OBSTACLE_DISTANCE) {
    // Obstáculo frontal - girar hacia izquierda
    motors.setSpeeds(-TURN_SPEED, TURN_SPEED);
  }
```

```
  else if (right < OBSTACLE_DISTANCE) {
    // Obstáculo a la derecha - ignorar/alejarse
    motors.setSpeeds(TURN_SPEED, -TURN_SPEED);
  }
  else {
    // No detecta nada - retroceder al máximo
    motors.setSpeeds(REVERSE_SPEED, REVERSE_SPEED);
  }
}

void rightAttackMode(int front, int left, int right) {
  if (right < OBSTACLE_DISTANCE) {
    // Obstáculo a la derecha - atacar girando
    motors.setSpeeds(MAX_SPEED, -TURN_SPEED);
  }
  else if (front < OBSTACLE_DISTANCE) {
    // Obstáculo frontal - girar hacia derecha
    motors.setSpeeds(TURN_SPEED, -TURN_SPEED);
  }
  else if (left < OBSTACLE_DISTANCE) {
    // Obstáculo a la izquierda - ignorar/alejarse
    motors.setSpeeds(-TURN_SPEED, TURN_SPEED);
  }
  else {
    // No detecta nada - retroceder al máximo
    motors.setSpeeds(REVERSE_SPEED, REVERSE_SPEED);
  }
}

int readSharpSensor(int pin) {
  // Leer sensor Sharp y convertir a distancia en cm
  // Esta función necesita ser calibrada según tu modelo específico de sensor
  int raw = analogRead(pin);
  // Fórmula aproximada - ajustar según tu sensor
  return 13000 / (raw + 1);  // Conversión aproximada a cm
}

void calibrateLineSensor() {
  // Calibración del sensor de línea
  for (int i = 0; i < 100; i++) {
    qtr.calibrate();
    delay(20);
  }
}
```

Instrucciones de uso:

1. **Conexiones**:
   - Conecta los sensores Sharp a los pines A0 (frontal), A1 (izquierdo) y A2 (derecho)
   - Conecta los switches a los pines 2 (frontal), 3 (izquierdo) y 4 (derecho)
   - Conecta el sensor QTR-8 a los pines 5-12
2. **Modos de ataque**:

- Switch frontal: Ataca obstáculos frontales
- Switch izquierdo: Ataca obstáculos a la izquierda
- Switch derecho: Ataca obstáculos a la derecha

3. **Comportamiento**:
   - Si no detecta ningún obstáculo, retrocede a máxima velocidad
   - Si detecta línea (con el QTR-8), retrocede inmediatamente
   - El comportamiento de ataque varía según el modo seleccionado

Ajustes necesarios:

1. Calibra los valores de `OBSTACLE_DISTANCE` y `LINE_THRESHOLD` según tus sensores.
2. Ajusta la función `readSharpSensor()` según el modelo específico de tus sensores Sharp.
3. Puedes modificar las velocidades (MAX_SPEED, REVERSE_SPEED, TURN_SPEED) según necesites.

Este código proporciona una base sólida que puedes modificar según tus necesidades específicas.

incorpora si es posible,sensor sharp trasero

Código Actualizado para Zumo con Sensor Sharp Trasero

Aquí tienes el código actualizado que incluye un sensor Sharp trasero para mejorar la detección de obstáculos:

cpp Copy

```cpp
#include <ZumoMotors.h>
#include <QTRSensors.h>

// Definición de pines
#define FRONT_SENSOR A0
#define LEFT_SENSOR A1
#define RIGHT_SENSOR A2
#define REAR_SENSOR A3  // Nuevo sensor trasero
#define SWITCH_FRONT 2
#define SWITCH_LEFT 3
#define SWITCH_RIGHT 4

// Umbrales de detección (ajustar según calibración)
#define OBSTACLE_DISTANCE 30  // en cm
#define REAR_OBSTACLE_DISTANCE 20  // Umbral más bajo para el sensor trasero
#define LINE_THRESHOLD 500    // valor para detectar línea

// Velocidades de los motores
#define MAX_SPEED 400
#define REVERSE_SPEED -400
#define TURN_SPEED 300
```

```
#define REVERSE_TURN_SPEED -300

ZumoMotors motors;
QTRSensorsRC qtr((unsigned char[]) {5, 6, 7, 8, 9, 10, 11, 12}, 8);

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

  // Configurar pines de los switches como entrada con pull-up
  pinMode(SWITCH_FRONT, INPUT_PULLUP);
  pinMode(SWITCH_LEFT, INPUT_PULLUP);
  pinMode(SWITCH_RIGHT, INPUT_PULLUP);

  // Calibrar sensor de línea
  calibrateLineSensor();
}

void loop() {
  // Leer sensores de distancia
  int frontDistance = readSharpSensor(FRONT_SENSOR);
  int leftDistance = readSharpSensor(LEFT_SENSOR);
  int rightDistance = readSharpSensor(RIGHT_SENSOR);
  int rearDistance = readSharpSensor(REAR_SENSOR);  // Leer sensor trasero

  // Leer sensor de línea
  unsigned int lineValues[8];
  int linePosition = qtr.readLine(lineValues);
  bool onLine = (lineValues[0] > LINE_THRESHOLD || lineValues[7] > LINE_THRESHOLD);

  // Leer modo de ataque seleccionado
  bool attackFront = !digitalRead(SWITCH_FRONT);
  bool attackLeft = !digitalRead(SWITCH_LEFT);
  bool attackRight = !digitalRead(SWITCH_RIGHT);

  // Si detecta línea, retroceder
  if (onLine) {
    motors.setSpeeds(REVERSE_SPEED, REVERSE_SPEED);
    delay(500);
    return;
  }

  // Comportamiento según modo de ataque seleccionado
  if (attackFront) {
    frontAttackMode(frontDistance, leftDistance, rightDistance, rearDistance);
  }
  else if (attackLeft) {
    leftAttackMode(frontDistance, leftDistance, rightDistance, rearDistance);
  }
  else if (attackRight) {
    rightAttackMode(frontDistance, leftDistance, rightDistance, rearDistance);
  }
  else {
    // Si no hay ningún modo seleccionado, no hacer nada
    motors.setSpeeds(0, 0);
  }
}
```

```
void frontAttackMode(int front, int left, int right, int rear) {
  if (front < OBSTACLE_DISTANCE) {
    // Obstáculo frontal - atacar
    motors.setSpeeds(MAX_SPEED, MAX_SPEED);
  }
  else if (left < OBSTACLE_DISTANCE) {
    // Obstáculo a la izquierda - girar ligeramente
    motors.setSpeeds(TURN_SPEED, MAX_SPEED);
  }
  else if (right < OBSTACLE_DISTANCE) {
    // Obstáculo a la derecha - girar ligeramente
    motors.setSpeeds(MAX_SPEED, TURN_SPEED);
  }
  else if (rear < REAR_OBSTACLE_DISTANCE) {
    // Obstáculo trasero - avanzar para alejarse
    motors.setSpeeds(MAX_SPEED, MAX_SPEED);
  }
  else {
    // No detecta nada - retroceder al máximo
    motors.setSpeeds(REVERSE_SPEED, REVERSE_SPEED);
  }
}

void leftAttackMode(int front, int left, int right, int rear) {
  if (left < OBSTACLE_DISTANCE) {
    // Obstáculo a la izquierda - atacar girando
    motors.setSpeeds(-TURN_SPEED, MAX_SPEED);
  }
  else if (front < OBSTACLE_DISTANCE) {
    // Obstáculo frontal - girar hacia izquierda
    motors.setSpeeds(-TURN_SPEED, TURN_SPEED);
  }
  else if (right < OBSTACLE_DISTANCE) {
    // Obstáculo a la derecha - ignorar/alejarse
    motors.setSpeeds(TURN_SPEED, -TURN_SPEED);
  }
  else if (rear < REAR_OBSTACLE_DISTANCE) {
    // Obstáculo trasero - avanzar hacia izquierda
    motors.setSpeeds(MAX_SPEED, TURN_SPEED);
  }
  else {
    // No detecta nada - retroceder al máximo
    motors.setSpeeds(REVERSE_SPEED, REVERSE_SPEED);
  }
}

void rightAttackMode(int front, int left, int right, int rear) {
  if (right < OBSTACLE_DISTANCE) {
    // Obstáculo a la derecha - atacar girando
    motors.setSpeeds(MAX_SPEED, -TURN_SPEED);
  }
  else if (front < OBSTACLE_DISTANCE) {
    // Obstáculo frontal - girar hacia derecha
    motors.setSpeeds(TURN_SPEED, -TURN_SPEED);
  }
```

```
  else if (left < OBSTACLE_DISTANCE) {
    // Obstáculo a la izquierda - ignorar/alejarse
    motors.setSpeeds(-TURN_SPEED, TURN_SPEED);
  }
  else if (rear < REAR_OBSTACLE_DISTANCE) {
    // Obstáculo trasero - avanzar hacia derecha
    motors.setSpeeds(TURN_SPEED, MAX_SPEED);
  }
  else {
    // No detecta nada - retroceder al máximo
    motors.setSpeeds(REVERSE_SPEED, REVERSE_SPEED);
  }
}

int readSharpSensor(int pin) {
  // Leer sensor Sharp y convertir a distancia en cm
  // Esta función necesita ser calibrada según tu modelo específico de sensor
  int raw = analogRead(pin);
  // Fórmula aproximada - ajustar según tu sensor
  return 13000 / (raw + 1);  // Conversión aproximada a cm
}

void calibrateLineSensor() {
  // Calibración del sensor de línea
  for (int i = 0; i < 100; i++) {
    qtr.calibrate();
    delay(20);
  }
}
```

Mejoras incorporadas:

1. **Sensor trasero añadido**:
   - Conectado al pin analógico A3
   - Umbral de detección específico (`REAR_OBSTACLE_DISTANCE`) más bajo para mayor sensibilidad
2. **Nuevos comportamientos**:
   - Si detecta un obstáculo trasero, en lugar de retroceder avanzará hacia adelante
   - En modo izquierda/derecha, si detecta obstáculo trasero avanzará con tendencia a girar
3. **Velocidad de giro en reversa**:
   - Añadida constante `REVERSE_TURN_SPEED` para movimientos más controlados

Configuración adicional:

1. **Conexión del sensor trasero**:
   - Conecta el sensor Sharp trasero al pin A3
   - Asegúrate de que esté orientado correctamente hacia atrás
2. **Calibración**:

- Ajusta `REAR_OBSTACLE_DISTANCE` según la distancia deseada para detección trasera
- Calibra la función `readSharpSensor()` para tu modelo específico de sensor

3. **Comportamiento**:
   - El robot ahora prioriza alejarse de obstáculos traseros en lugar de retroceder
   - Si no hay obstáculos en ninguna dirección, mantiene el comportamiento de retroceder

Este código proporciona una detección más completa de 360 grados alrededor del robot.

ahora repite el codigo pero usando solo 4 sensores de linea en vez de los 8 y que siempre que detecte al oponente se coloque rapidamente enfrente y empuje al maximo

Código para Zumo con 4 Sensores de Línea y Ataque Frontal

cpp Copy

```cpp
#include <ZumoMotors.h>
#include <QTRSensors.h>

// Definición de pines
#define FRONT_SENSOR A0
#define LEFT_SENSOR A1
#define RIGHT_SENSOR A2
#define REAR_SENSOR A3
#define SWITCH_FRONT 2
#define SWITCH_LEFT 3
#define SWITCH_RIGHT 4

// Configuración sensores de línea (solo 4)
#define NUM_LINE_SENSORS 4
unsigned char lineSensorPins[NUM_LINE_SENSORS] = {5, 6, 7, 8}; // Usando solo 4 pines

// Umbrales de detección
#define OBSTACLE_DISTANCE 30    // en cm (frontal/laterales)
#define REAR_OBSTACLE_DISTANCE 20 // en cm (trasero)
#define LINE_THRESHOLD 500       // valor para detectar línea

// Velocidades
#define MAX_SPEED 400
#define REVERSE_SPEED -400
#define TURN_SPEED 350
#define ALIGN_SPEED 200          // Velocidad para alinearse con oponente

ZumoMotors motors;
QTRSensorsRC qtr(lineSensorPins, NUM_LINE_SENSORS); // Solo 4 sensores de línea

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

```
  // Configurar switches
  pinMode(SWITCH_FRONT, INPUT_PULLUP);
  pinMode(SWITCH_LEFT, INPUT_PULLUP);
  pinMode(SWITCH_RIGHT, INPUT_PULLUP);

  // Calibrar sensores de línea
  calibrateLineSensor();
}

void loop() {
  // Leer sensores de distancia
  int frontDistance = readSharpSensor(FRONT_SENSOR);
  int leftDistance = readSharpSensor(LEFT_SENSOR);
  int rightDistance = readSharpSensor(RIGHT_SENSOR);
  int rearDistance = readSharpSensor(REAR_SENSOR);

  // Leer sensores de línea
  unsigned int lineValues[NUM_LINE_SENSORS];
  qtr.readLine(lineValues);
  bool onLine = checkLineDetection(lineValues);

  // Leer modo de ataque
  bool attackFront = !digitalRead(SWITCH_FRONT);
  bool attackLeft = !digitalRead(SWITCH_LEFT);
  bool attackRight = !digitalRead(SWITCH_RIGHT);

  // Si detecta línea, retroceder
  if (onLine) {
    motors.setSpeeds(REVERSE_SPEED, REVERSE_SPEED);
    delay(300);
    return;
  }

  // Comportamiento principal
  if (frontDistance < OBSTACLE_DISTANCE) {
    // Oponente frontal - ataque máximo
    alignAndAttack(frontDistance, leftDistance, rightDistance);
  }
  else if (leftDistance < OBSTACLE_DISTANCE) {
    // Oponente izquierda - girar y alinear
    approachOpponent(LEFT_SENSOR);
  }
  else if (rightDistance < OBSTACLE_DISTANCE) {
    // Oponente derecha - girar y alinear
    approachOpponent(RIGHT_SENSOR);
  }
  else if (rearDistance < REAR_OBSTACLE_DISTANCE) {
    // Oponente trasero - girar 180°
    motors.setSpeeds(TURN_SPEED, -TURN_SPEED);
    delay(400);
  }
  else {
    // No detecta oponente - buscar
    searchPattern();
  }
}
```

```
void alignAndAttack(int front, int left, int right) {
  // Pequeños ajustes para alinearse perfectamente
  if (left < right && (right - left) > 5) {
    // Girar ligeramente a la derecha para alinear
    motors.setSpeeds(MAX_SPEED, ALIGN_SPEED);
    delay(50);
  }
  else if (right < left && (left - right) > 5) {
    // Girar ligeramente a la izquierda para alinear
    motors.setSpeeds(ALIGN_SPEED, MAX_SPEED);
    delay(50);
  }

  // Ataque frontal máximo
  motors.setSpeeds(MAX_SPEED, MAX_SPEED);
}

void approachOpponent(int side) {
  // Acercarse al oponente detectado en un lateral
  if (side == LEFT_SENSOR) {
    // Girar izquierda y avanzar
    motors.setSpeeds(-TURN_SPEED, MAX_SPEED);
    delay(200);
  }
  else {
    // Girar derecha y avanzar
    motors.setSpeeds(MAX_SPEED, -TURN_SPEED);
    delay(200);
  }

  // Avanzar un poco después de girar
  motors.setSpeeds(MAX_SPEED, MAX_SPEED);
  delay(100);
}

void searchPattern() {
  // Patrón de búsqueda cuando no detecta oponente
  static unsigned long lastSearchTime = 0;
  static int searchState = 0;

  if (millis() - lastSearchTime > 1000) {
    searchState = (searchState + 1) % 4;
    lastSearchTime = millis();
  }

  switch (searchState) {
    case 0: motors.setSpeeds(TURN_SPEED, -TURN_SPEED); break; // Girar derecha
    case 1: motors.setSpeeds(MAX_SPEED/2, MAX_SPEED/2); break; // Avanzar
    case 2: motors.setSpeeds(-TURN_SPEED, TURN_SPEED); break; // Girar izquierda
    case 3: motors.setSpeeds(MAX_SPEED/2, MAX_SPEED/2); break; // Avanzar
  }
}

bool checkLineDetection(unsigned int* lineValues) {
  // Verificar si algún sensor de línea detecta línea
```

```
  for (int i = 0; i < NUM_LINE_SENSORS; i++) {
    if (lineValues[i] > LINE_THRESHOLD) {
      return true;
    }
  }
  return false;
}

int readSharpSensor(int pin) {
  // Leer sensor Sharp y convertir a distancia
  int raw = analogRead(pin);
  return 13000 / (raw + 1); // Conversión aproximada a cm
}

void calibrateLineSensor() {
  // Calibración de los 4 sensores de línea
  for (int i = 0; i < 100; i++) {
    qtr.calibrate();
    delay(20);
  }
}
```

Características principales:

1. **Sensores de línea reducidos**:
   - Usa solo 4 sensores (pines 5-8) en lugar de 8
   - Detección eficiente de bordes del doyo
2. **Comportamiento agresivo frontal**:
   - Cuando detecta oponente frontal, se alinea perfectamente y ataca a máxima velocidad
   - Realiza pequeños ajustes para mantener el frente hacia el oponente
3. **Detección y aproximación**:
   - Si detecta oponente lateral, gira rápidamente para colocarse frente a él
   - Para oponente trasero, realiza giro de 180° para enfrentarlo
4. **Patrón de búsqueda**:
   - Cuando no detecta oponentes, realiza un patrón alternado de giros y avances
5. **Protección del doyo**:
   - Si cualquier sensor de línea detecta el borde, retrocede inmediatamente

Instrucciones de uso:

1. Conecta los 4 sensores de línea a los pines digitales 5-8
2. Los sensores Sharp siguen en A0 (frontal), A1 (izquierdo), A2 (derecho) y A3 (trasero)
3. Los switches de selección de modo en pines 2-4 (aunque el comportamiento ahora es más autónomo)

Este código está optimizado para un combate Sumo agresivo, priorizando siempre colocarse frente al oponente y empujar a máxima potencia.

cambia el switch sensor izquierdo para seleccionar deteccion sensor trasero

Código Modificado con Selección de Sensor Trasero

Aquí tienes el código actualizado donde el switch izquierdo activa/desactiva la detección del sensor trasero:

cpp Copy

```cpp
#include <ZumoMotors.h>
#include <QTRSensors.h>

// Definición de pines
#define FRONT_SENSOR A0
#define LEFT_SENSOR A1
#define RIGHT_SENSOR A2
#define REAR_SENSOR A3
#define SWITCH_FRONT 2
#define SWITCH_REAR 3  // Ahora controla sensor trasero
#define SWITCH_RIGHT 4

// Configuración sensores de línea (4)
#define NUM_LINE_SENSORS 4
unsigned char lineSensorPins[NUM_LINE_SENSORS] = {5, 6, 7, 8};

// Umbrales de detección
#define OBSTACLE_DISTANCE 30    // en cm
#define REAR_OBSTACLE_DISTANCE 20
#define LINE_THRESHOLD 500

// Velocidades
#define MAX_SPEED 400
#define REVERSE_SPEED -400
#define TURN_SPEED 350
#define ALIGN_SPEED 200

ZumoMotors motors;
QTRSensorsRC qtr(lineSensorPins, NUM_LINE_SENSORS);

// Variables de estado
bool rearDetectionEnabled = false;  // Control de sensor trasero

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

  // Configurar switches
  pinMode(S
```

muestrame todo el codigo porfavor

Código Completo para Zumo Sumo con Control de Sensor Trasero

cpp Copy

```cpp
#include <ZumoMotors.h>
#include <QTRSensors.h>

// ========== CONFIGURACIÓN DE PINES ==========
#define FRONT_SENSOR A0    // Sensor Sharp frontal
#define LEFT_SENSOR A1     // Sensor Sharp izquierdo
#define RIGHT_SENSOR A2    // Sensor Sharp derecho
#define REAR_SENSOR A3     // Sensor Sharp trasero
#define SWITCH_FRONT 2     // Switch frontal (no usado actualmente)
#define SWITCH_REAR 3      // Switch izquierdo - control sensor trasero
#define SWITCH_RIGHT 4     // Switch derecho (no usado actualmente)

// Configuración de 4 sensores de línea (minimizados)
#define NUM_LINE_SENSORS 4
unsigned char lineSensorPins[NUM_LINE_SENSORS] = {5, 6, 7, 8}; // Pines digitales

// ========== PARÁMETROS AJUSTABLES ==========
#define OBSTACLE_DISTANCE 30      // Distancia para detección frontal/lateral (cm)
#define REAR_OBSTACLE_DISTANCE 20 // Distancia para detección trasera (cm)
#define LINE_THRESHOLD 500        // Valor umbral para detección de línea

// Velocidades de los motores
#define MAX_SPEED 400             // Velocidad máxima de avance
#define REVERSE_SPEED -400        // Velocidad máxima de retroceso
#define TURN_SPEED 350            // Velocidad de giro
#define ALIGN_SPEED 200           // Velocidad para alineación fina
#define SEARCH_SPEED 200          // Velocidad de búsqueda

// ========== OBJETOS Y VARIABLES GLOBALES ==========
ZumoMotors motors;                // Controlador de motores
QTRSensorsRC qtr(lineSensorPins, NUM_LINE_SENSORS); // Sensores de línea

bool rearDetectionEnabled = false; // Control de sensor trasero
unsigned long lastSearchChange = 0;
int searchMode = 0;

// ========== FUNCIÓN SETUP ==========
void setup() {
  Serial.begin(9600);

  // Configuración de pines de interruptores
  pinMode(SWITCH_FRONT, INPUT_PULLUP);
  pinMode(SWITCH_REAR, INPUT_PULLUP);
  pinMode(SWITCH_RIGHT, INPUT_PULLUP);

  // Calibración de sensores de línea
  calibrateLineSensor();
```

```
}

// ========== FUNCIÓN PRINCIPAL LOOP ==========
void loop() {
  // 1. Leer estado de los interruptores
  rearDetectionEnabled = !digitalRead(SWITCH_REAR); // Control sensor trasero

  // 2. Leer todos los sensores
  int frontDist = readSharpSensor(FRONT_SENSOR);
  int leftDist = readSharpSensor(LEFT_SENSOR);
  int rightDist = readSharpSensor(RIGHT_SENSOR);
  int rearDist = rearDetectionEnabled ? readSharpSensor(REAR_SENSOR) : 999;

  unsigned int lineValues[NUM_LINE_SENSORS];
  qtr.readLine(lineValues);
  bool lineDetected = checkLineDetection(lineValues);

  // 3. Comprobar si está en el borde del dohyo
  if (lineDetected) {
    evadeEdge();
    return;
  }

  // 4. Lógica principal de ataque
  if (frontDist < OBSTACLE_DISTANCE) {
    attackFront(frontDist, leftDist, rightDist);
  }
  else if (leftDist < OBSTACLE_DISTANCE) {
    attackLeft();
  }
  else if (rightDist < OBSTACLE_DISTANCE) {
    attackRight();
  }
  else if (rearDetectionEnabled && rearDist < REAR_OBSTACLE_DISTANCE) {
    turnAround();
  }
  else {
    searchOpponent();
  }
}

// ========== FUNCIONES DE COMPORTAMIENTO ==========

void evadeEdge() {
  // Retrocede y gira aleatoriamente al detectar línea
  motors.setSpeeds(REVERSE_SPEED, REVERSE_SPEED);
  delay(300);
  motors.setSpeeds(TURN_SPEED, -TURN_SPEED);
  delay(random(200, 400));
}

void attackFront(int front, int left, int right) {
  // Ajuste fino de alineación
  if (abs(left - right) > 5) {
    if (left < right) {
      motors.setSpeeds(MAX_SPEED, ALIGN_SPEED);
```

```
    } else {
      motors.setSpeeds(ALIGN_SPEED, MAX_SPEED);
    }
    delay(30);
  }
  // Ataque frontal máximo
  motors.setSpeeds(MAX_SPEED, MAX_SPEED);
}

void attackLeft() {
  // Giro de 90° izquierda y ataque
  motors.setSpeeds(-TURN_SPEED, TURN_SPEED);
  delay(250);
  motors.setSpeeds(MAX_SPEED, MAX_SPEED);
  delay(100);
}

void attackRight() {
  // Giro de 90° derecha y ataque
  motors.setSpeeds(TURN_SPEED, -TURN_SPEED);
  delay(250);
  motors.setSpeeds(MAX_SPEED, MAX_SPEED);
  delay(100);
}

void turnAround() {
  // Giro de 180° cuando detecta oponente trasero
  motors.setSpeeds(TURN_SPEED, -TURN_SPEED);
  delay(500);
}

void searchOpponent() {
  // Patrón de búsqueda con cambio periódico
  if (millis() - lastSearchChange > 1500) {
    searchMode = random(0, 4);
    lastSearchChange = millis();
  }

  switch (searchMode) {
    case 0: // Avance lento
      motors.setSpeeds(SEARCH_SPEED, SEARCH_SPEED);
      break;
    case 1: // Giro derecha
      motors.setSpeeds(TURN_SPEED, -TURN_SPEED);
      break;
    case 2: // Giro izquierda
      motors.setSpeeds(-TURN_SPEED, TURN_SPEED);
      break;
    case 3: // Avance en curva
      motors.setSpeeds(SEARCH_SPEED, SEARCH_SPEED/2);
      break;
  }
}

// ========== FUNCIONES AUXILIARES ==========
```

```
bool checkLineDetection(unsigned int* values) {
  for (int i = 0; i < NUM_LINE_SENSORS; i++) {
    if (values[i] > LINE_THRESHOLD) return true;
  }
  return false;
}

int readSharpSensor(int pin) {
  int raw = analogRead(pin);
  // Fórmula aproximada para GP2Y0A21YK0F (10-80cm)
  return 2076 / (raw - 11); // Ajustar según sensor específico
}

void calibrateLineSensor() {
  // Calibración automática durante 2 segundos
  for (int i = 0; i < 100; i++) {
    qtr.calibrate();
    delay(20);
  }
}
```

Características Clave:

1. **Control del sensor trasero**:
   - Activación/desactivación mediante el switch izquierdo (pin 3)
   - Cuando está desactivado, ignora completamente los obstáculos traseros
2. **Estrategia de ataque mejorada**:
   - Movimientos precisos de 90° para enfrentar oponentes laterales
   - Giro de 180° rápido para oponentes traseros (cuando está activado)
   - Ajuste fino de alineación para ataque frontal perfecto
3. **Detección de bordes**:
   - Usa solo 4 sensores de línea para mayor simplicidad
   - Retroceso y giro aleatorio al detectar el borde del dohyo
4. **Patrón de búsqueda**:
   - Comportamiento aleatorio cuando no detecta oponentes
   - Cambia cada 1.5 segundos entre diferentes patrones
5. **Optimizaciones**:
   - Código estructurado en funciones claras
   - Fácil ajuste de parámetros en la sección de configuración
   - Detección precisa de distancias con fórmulas calibradas

Instrucciones de Conexión:

1. **Sensores Sharp**:

- Frontal: A0
- Izquierdo: A1
- Derecho: A2
- Trasero: A3

2. **Sensores de Línea**:
   - Conectar a pines digitales 5-8
3. **Interruptores**:
   - Frontal: Pin 2 (no usado actualmente)
   - Izquierdo (control trasero): Pin 3
   - Derecho: Pin 4 (no usado actualmente)

Este código proporciona un comportamiento agresivo pero controlado, ideal para combates de sumo robótico, con la flexibilidad de activar o desactivar la detección trasera según las necesidades tácticas.