Gestion de Obstáculos

El robot autónomo inicia en un estado de prueba donde se verifican los actuadores: el servo recorre sus posiciones extremas y el motor realiza movimientos hacia adelante y hacia atrás. Una vez que el test finaliza, el vehículo comienza a avanzar automáticamente con la dirección centrada.

Durante el funcionamiento, los sensores ultrasónicos frontal, izquierdo y derecho realizan mediciones continuas. El sensor frontal es el encargado de determinar cuándo existe un objeto demasiado cerca del vehículo. Si la distancia detectada es menor o igual al límite de seguridad programado, el robot detiene su avance e inicia una secuencia de evasión.

Primero, el motor se apaga por unos milisegundos y posteriormente el vehículo retrocede ligeramente para generar espacio. Después de retroceder, se analizan las distancias medidas por los sensores laterales: el sistema compara cuál de los dos lados está más libre y selecciona esa dirección para girar el servomotor.

Una vez decidido el lado, el servo gira hacia esa dirección y el robot avanza nuevamente durante un breve periodo para ejecutar la maniobra. Al finalizar el giro, el servomotor regresa a la posición central y el robot continúa su avance normal con el sensor frontal vigilando continuamente la distancia hacia los objetos.

Todo el proceso se repite de manera constante gracias al ciclo principal del programa, permitiendo que el robot avance mientras evita colisiones.

```
#include <Arduino.h>

#include <ESP32Servo.h>



// ========================================

//         PINES Y CONFIGURACIÓN DEL ESP32

// ========================================


// --- Pines sensores ultrasónicos ---

#define TRIG_FRONT 14

#define ECHO_FRONT 26

#define TRIG_LEFT 25

#define ECHO_LEFT 33

#define TRIG_RIGHT 13
```

Gestion de Obstáculos

```
#define ECHO_RIGHT 12


// --- Motor de Propulsión Trasero (Puente H) ---

#define MOTOR_IN1 32 // Control de avance/sentido (Pin definido por el usuario)

#define MOTOR_IN2 35 // Control de retroceso/sentido (Pin definido por el usuario - POSIBLE FUENTE DE ERROR)

#define MOTOR_ENA 23 // Habilitador del Puente H (Velocidad - Control Digital ON/OFF)


// --- Servomotor de Dirección (Ackermann) ---

#define SERVO_PIN 27


// --- LED de Estado (Pin por defecto) ---

#define LED_STATUS 2


// ==========================================

//          PARÁMETROS DE FUNCIONAMIENTO

// ==========================================

// Distancia de parada aumentada para mayor seguridad.

#define DISTANCIA_OBSTACULO 57 // Distancia mínima segura para detenerse (cm)


// --- Parámetros del Servo (INVERTIDOS PARA CORREGIR LA DIRECCIÓN FÍSICA) ---

#define SERVO_CENTRO 90  // Dirección recta

#define SERVO_IZQ 50     // Ángulo para girar físicamente a la IZQUIERDA

#define SERVO_DER 130    // Ángulo para girar físicamente a la DERECHA


// AJUSTES PARA DISMINUIR LA VELOCIDAD PERCIBIDA:
```

Gestion de Obstáculos

#define TIEMPO_GIRO 2500 // Reducido a 500ms para giros más cortos a máxima velocidad.

#define TIEMPO_ESPERA 50 // Aumentado a 300ms para pausas más largas entre movimientos.

#define LOOP_DELAY_MS 50 // NUEVA PAUSA: Ralentiza la frecuencia de lectura de sensores y control.

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

//          VARIABLES GLOBALES

// ==========================================

Servo direccion;


// ==========================================

//          FUNCIONES DE SENSORES

// ==========================================


/**

 * @brief Mide la distancia en cm usando el sensor ultrasónico.

 * @param trig Pin TRIG del sensor.

 * @param echo Pin ECHO del sensor.

 * @return Distancia en centímetros (long).

 */

long medirDistancia(int trig, int echo) {

  // Asegurarse de que el Trigger esté apagado al inicio

  digitalWrite(trig, LOW);

  delayMicroseconds(2);


  // Enviar pulso de 10us para medir
```

Gestion de Obstáculos

```
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);

  // Leer la duración del pulso de retorno. 20000us es el timeout máximo.
  long duracion = pulseIn(echo, HIGH, 20000);

  // Convertir duración a distancia (fórmula: tiempo * 0.034 / 2)
  long distancia = duracion * 0.034 / 2; // cm

  // Limitar la distancia máxima para evitar lecturas raras
  if (distancia == 0 || distancia > 100) {
    return 100;
  }
  return distancia;
}

// ==========================================
//         FUNCIONES DE ACTUADORES
// ==========================================

/**
 * @brief Mueve el motor de propulsión hacia adelante (a MÁXIMA VELOCIDAD).
 */
void motorAdelante() {
  // Avance: IN1=HIGH, IN2=LOW
```

Gestion de Obstáculos

```
  digitalWrite(MOTOR_IN1, HIGH);
  digitalWrite(MOTOR_IN2, LOW);
  // Aplicar velocidad MÁXIMA (Digital HIGH)
  digitalWrite(MOTOR_ENA, HIGH);
}

/**
 * @brief Mueve el motor de propulsión hacia atrás (a MÁXIMA VELOCIDAD).
 */
void motorAtras() {
  // Retroceso: IN1=LOW, IN2=HIGH
  digitalWrite(MOTOR_IN1, LOW);
  digitalWrite(MOTOR_IN2, HIGH);
  // Aplicar velocidad MÁXIMA (Digital HIGH)
  digitalWrite(MOTOR_ENA, HIGH);
}

/**
 * @brief Detiene el motor de propulsión.
 */
void motorDetener() {
  // Detener motor deshabilitando el pin ENA
  digitalWrite(MOTOR_ENA, LOW);
  digitalWrite(MOTOR_IN1, LOW);
  digitalWrite(MOTOR_IN2, LOW);
}
```

Gestion de Obstáculos

```
// --- Control del servo ---
void girarCentro() { direccion.write(SERVO_CENTRO); }
void girarIzquierda() { direccion.write(SERVO_IZQ); }
void girarDerecha() { direccion.write(SERVO_DER); }

// ==========================================
//          TESTEO DE ACTUADORES
// ==========================================

/**
 * @brief Ejecuta una secuencia de prueba de movimientos para verificar motor y servo.
 */
void testActuadores() {
  Serial.println("--- INICIANDO TEST DE ACTUADORES (32, 35) ---");

  // 1. Prueba de Servo
  Serial.println("TEST: Girando a 0 y 180 grados...");
  direccion.write(0);
  delay(1000);
  direccion.write(180);
  delay(1000);
  Serial.println("TEST: Centrando...");
  girarCentro();
  delay(1000);
```

Gestion de Obstáculos

```
  // 2. Prueba de Motor
  Serial.println("TEST: Motor Adelante (MÁXIMA VELOCIDAD)...");
  motorAdelante();
  delay(1500);

  Serial.println("TEST: Motor Detener...");
  motorDetener();
  delay(500);

  Serial.println("TEST: Motor Atrás (MÁXIMA VELOCIDAD)...");
  motorAtras();
  delay(1500);

  Serial.println("TEST: Motor Detener...");
  motorDetener();
  delay(500);

  Serial.println("--- TEST DE ACTUADORES FINALIZADO ---");
  Serial.println("INICIANDO MARCHA AUTÓNOMA (Detección a 40cm)...");
}



// ==========================================
//          SETUP
// ==========================================
```

Gestion de Obstáculos

```
void setup() {
  Serial.begin(115200);

  // Configuración de pines de motor (Todos como simple OUTPUT)
  pinMode(MOTOR_IN1, OUTPUT);
  pinMode(MOTOR_IN2, OUTPUT);
  pinMode(MOTOR_ENA, OUTPUT); // ENA configurado como simple pin digital

  // Configuración de pines de sensores
  pinMode(TRIG_FRONT, OUTPUT);
  pinMode(ECHO_FRONT, INPUT);
  pinMode(TRIG_LEFT, OUTPUT);
  pinMode(ECHO_LEFT, INPUT);
  pinMode(TRIG_RIGHT, OUTPUT);
  pinMode(ECHO_RIGHT, INPUT);

  // Configuración de pines de control
  pinMode(LED_STATUS, OUTPUT);

  // Inicializar servomotor
  direccion.attach(SERVO_PIN);

  // Estado inicial de actuadores
  motorDetener();
  girarCentro();
```

Gestion de Obstáculos

```
  // Ejecutar prueba de actuadores
  testActuadores();

  // INICIO INMEDIATO:
  motorAdelante();        // El robot arranca hacia adelante a máxima velocidad
  digitalWrite(LED_STATUS, HIGH); // LED fijo: en marcha
}

// ==========================================
//          LOOP PRINCIPAL (LÓGICA AUTÓNOMA)
// ==========================================

void loop() {
  // 1. Medir las distancias
  long distFront = medirDistancia(TRIG_FRONT, ECHO_FRONT);
  long distLeft = medirDistancia(TRIG_LEFT, ECHO_LEFT);
  long distRight = medirDistancia(TRIG_RIGHT, ECHO_RIGHT);

  Serial.printf("D: %.0f | L: %.0f | R: %.0f cm\n", (float)distFront, (float)distLeft,
(float)distRight);


  // 2. Lógica de Evasión de Obstáculos
  if (distFront <= DISTANCIA_OBSTACULO) {
    // OBSTÁCULO EN FRENTE! Iniciar secuencia de evasión.
    Serial.println("!!! OBSTÁCULO DETECTADO (<40cm). EVADIENDO...");
    digitalWrite(LED_STATUS, LOW); // LED apagado: maniobra
```

Gestion de Obstáculos

```
// a. Detener la marcha
motorDetener();
delay(TIEMPO_ESPERA); // Pausa más larga

// b. Retroceder un poco para ganar espacio
motorAtras();
girarCentro();
delay(400);
motorDetener();
delay(TIEMPO_ESPERA); // Pausa más larga

// c. Decidir la dirección de giro (la más despejada)
if (distLeft > distRight) {
  // Izquierda está más libre
  Serial.println("Decisión: Girar Izquierda (Ruta libre por Izquierda)");
  girarIzquierda();
} else {
  // Derecha está más libre o ambas son iguales
  Serial.println("Decisión: Girar Derecha (Ruta libre por Derecha)");
  girarDerecha();
}

// d. Avanzar manteniendo el ángulo de giro para esquivar
motorAdelante();
delay(TIEMPO_GIRO); // Giro más corto
```

Gestion de Obstáculos

```
    // e. Volver a centrar la dirección (rectificar)
    motorDetener();
    girarCentro();
    delay(TIEMPO_ESPERA);

    // f. Reanudar el avance normal
    digitalWrite(LED_STATUS, HIGH); // LED fijo: avance normal

  } else {
    // No hay obstáculo en frente, continuar avanzando
    motorAdelante();
    girarCentro();
  }

  // PAUSA AÑADIDA: Ralentiza la frecuencia de comprobación de sensores y control.
  delay(LOOP_DELAY_MS);
}
```

Gestion de Obstáculos