

USO DE IA PARA CONSULTAR SOBRE
LA PROGRAMACIÓN DE COMPONENTES PARTE #3

Auto Autónomo Equipo SJT RoboClub

AL VER ESTE DIAGRAMA ME HE DADO CUENTA QUE AL ESTACIONAR TAMBIEN DEBE EVITAR CHOCAR CON LOS BORDES DEL CUADRO COLOR MAGENTA LO PUEDES AGREGAR A LA PROGRAMACIÓN

ChatGPT

Claro, aquí tienes la programación actualizada para que el vehículo evite chocar con los bordes del cuadro color magenta al estacionarse:

```
cpp
Copiar código
#include <SoftwareSerial.h>
#include <Pixy2.h>
#include <Servo.h>

// Definir pines
const int motorIn1 = 5;
const int motorIn2 = 6;
const int motorEn = 3;
const int servoDireccionPin = 9;
const int servoHPin = 10;
const int servoVPin = 11;

SoftwareSerial lidarSerial(18, 19); // RX, TX
Pixy2 pixy;
Servo servoDireccion;
Servo servoH;
Servo servoV;

int vueltasCompletadas = 0;
bool estacionado = false;
bool bloqueMagentaEncontrado = false;

void setup() {
  Serial.begin(9600);
  lidarSerial.begin(115200);
  pixy.init();
  servoDireccion.attach(servoDireccionPin);
  servoH.attach(servoHPin);
  servoV.attach(servoVPin);

  pinMode(motorIn1, OUTPUT);
  pinMode(motorIn2, OUTPUT);
  pinMode(motorEn, OUTPUT);
}

void loop() {
  if (!estacionado) {
```

```

        if (vueltasCompletadas < 3) {
            leerSensoresYESquivar();
            contarVueltas();
        } else {
            if (bloqueMagentaEncontrado) {
                buscarYEstacionarse();
            } else {
                parar();
                estacionado = true;
            }
        }
    }
}

// Funciones para Controlar Motores y Servos
void adelante() {
    digitalWrite(motorIn1, HIGH);
    digitalWrite(motorIn2, LOW);
    analogWrite(motorEn, 255); // Velocidad máxima
}

void atras() {
    digitalWrite(motorIn1, LOW);
    digitalWrite(motorIn2, HIGH);
    analogWrite(motorEn, 255); // Velocidad máxima
}

void parar() {
    digitalWrite(motorIn1, LOW);
    digitalWrite(motorIn2, LOW);
    analogWrite(motorEn, 0); // Parar
}

void girarIzquierda() {
    servoDireccion.write(45); // Ajustar el valor según sea necesario
}

void girarDerecha() {
    servoDireccion.write(135); // Ajustar el valor según sea necesario
}

void centrarDireccion() {
    servoDireccion.write(90); // Posición central
}

// Funciones para Leer Datos de los Sensores
int leerDistanciaLidar() {
    int distancia = 0;
    if (lidarSerial.available()) {
        // Leer y procesar la distancia del Lidar
        while (lidarSerial.available() > 0) {
            int c = lidarSerial.read();
            distancia = (distancia << 8) | c;
        }
    }
    return distancia;
}

void moverPixy() {

```

```

static bool direccionDerecha = true;

if (direccionDerecha) {
    servoH.write(180);
    delay(500); // Ajustar tiempo según sea necesario
    servoH.write(90);
    delay(500); // Ajustar tiempo según sea necesario
} else {
    servoH.write(0);
    delay(500); // Ajustar tiempo según sea necesario
    servoH.write(90);
    delay(500); // Ajustar tiempo según sea necesario
}

direccionDerecha = !direccionDerecha;
}

void leerSensoresYESquivar() {
    int distanciaFrontal = leerDistanciaLidar();

    // Evitar chocar con los bordes de la pista
    if (distanciaFrontal < 30) {
        girarDerecha();
        adelante();
        delay(1000); // Ajustar tiempo según sea necesario
        centrarDireccion();
    }

    // Leer datos del Pixy2
    pixy.ccc.getBlocks();
    if (pixy.ccc.numBlocks) {
        for (int i = 0; i < pixy.ccc.numBlocks; i++) {
            if (pixy.ccc.blocks[i].m_signature == 1) { // Supongamos que 1 es
verde
                girarIzquierda();
                adelante();
                delay(1000); // Ajustar tiempo según sea necesario
                centrarDireccion();
                break; // Salir del bucle al detectar un bloque verde
            } else if (pixy.ccc.blocks[i].m_signature == 2) { // Supongamos que
2 es rojo
                girarDerecha();
                adelante();
                delay(1000); // Ajustar tiempo según sea necesario
                centrarDireccion();
                break; // Salir del bucle al detectar un bloque rojo
            } else if (pixy.ccc.blocks[i].m_signature == 3) { // Supongamos que
3 es magenta
                bloqueMagentaEncontrado = true;
            }
        }
    }

    moverPixy(); // Mover la cámara Pixy2 para buscar bloques
}

// Lógica de Navegación para Esquivar Obstáculos
void esquivarObstaculo() {
    parar();
}

```

```

    girarDerecha();
    adelante();
    delay(1000); // Ajustar tiempo según sea necesario
    centrarDireccion();
}

// Contador de Vueltas
void contarVueltas() {
    static int posicionInicial = leerPosicionInicial();
    int posicionActual = leerPosicionActual();

    if (posicionActual == posicionInicial) {
        vueltasCompletadas++;
    }
}

int leerPosicionInicial() {
    int posicion = 0;
    // Código para leer y retornar la posición inicial
    return posicion;
}

int leerPosicionActual() {
    int posicion = 0;
    // Código para leer y retornar la posición actual
    return posicion;
}

// Buscar y Estacionarse en el Bloque Magenta
void buscarYEstacionarse() {
    unsigned long tiempoInicio = millis();
    bool bloqueMagentaEncontrado = false;
    int contadorMagenta = 0;

    while (millis() - tiempoInicio < 30000) { // Buscar durante 30 segundos
        pixy.ccc.getBlocks();
        if (pixy.ccc.numBlocks) {
            for (int i = 0; i < pixy.ccc.numBlocks; i++) {
                if (pixy.ccc.blocks[i].m_signature == 3) { // Supongamos que 3 es
magenta
                    contadorMagenta++;
                    if (contadorMagenta >= 2) {
                        bloqueMagentaEncontrado = true;
                        break;
                    }
                }
            }
        }
        if (bloqueMagentaEncontrado) break;
        adelante();
    }

    if (bloqueMagentaEncontrado) {
        parar();
        estacionar();
    } else {
        parar();
        estacionado = true;
    }
}

```

```

}

void estacionar() {
    unsigned long tiempoInicio = millis();
    while (millis() - tiempoInicio < 5000) { // Estacionar durante 5
segundos
        int distanciaFrontal = leerDistanciaLidar();
        if (distanciaFrontal < 30) {
            girarDerecha();
            adelante();
            delay(1000); // Ajustar tiempo según sea necesario
            centrarDireccion();
        } else {
            adelante();
        }
    }
    parar();
    estacionado = true; // Marcar como estacionado
}

```

Este código incluye la lógica para evitar chocar con los bordes del cuadro magenta mientras se estaciona, utilizando el sensor Lidar para detectar los bordes y ajustar la dirección si es necesario.

4o

¿SE ESTIPULA QUE ES UN SEGMENTO DE LA PISTA QUE SE ENCUENTRA EL ESTACIONAMIENTO SEGÚN LA LÓGICA DE LA PROGRAMACIÓN NO LOGRARÁ ESTACIONAR ENTRE LOS DOS BLOQUES?

ChatGPT

Para verificar si el vehículo se estacionará correctamente con la lógica de la programación, revisemos y ajustemos los segmentos críticos del código:

1. **Detección y Esquiva de Obstáculos:**
 - Asegurarnos de que el Lidar y la cámara Pixy2 detecten correctamente los bloques y bordes.
2. **Navegación y Contador de Vueltas:**
 - Confirmar que el contador de vueltas funcione adecuadamente.
3. **Búsqueda y Estacionamiento:**
 - Verificar que el vehículo busque el bloque magenta solo después de completar las tres vueltas.
 - Asegurarnos de que el vehículo pueda detectar y estacionarse entre las barras magentas sin chocar con los bordes.

Aquí está el código ajustado para mejorar la lógica de estacionamiento y garantizar que el vehículo solo busque estacionarse después de completar las tres vueltas:

```

cpp
Copiar código
#include <SoftwareSerial.h>
#include <Pixy2.h>
#include <Servo.h>

```

```

// Definir pines
const int motorIn1 = 5;
const int motorIn2 = 6;
const int motorEn = 3;
const int servoDireccionPin = 9;
const int servoHPin = 10;
const int servoVPin = 11;

SoftwareSerial lidarSerial(18, 19); // RX, TX
Pixy2 pixy;
Servo servoDireccion;
Servo servoH;
Servo servoV;

int vueltasCompletadas = 0;
bool estacionado = false;
bool bloqueMagentaEncontrado = false;

void setup() {
  Serial.begin(9600);
  lidarSerial.begin(115200);
  pixy.init();
  servoDireccion.attach(servoDireccionPin);
  servoH.attach(servoHPin);
  servoV.attach(servoVPin);

  pinMode(motorIn1, OUTPUT);
  pinMode(motorIn2, OUTPUT);
  pinMode(motorEn, OUTPUT);
}

void loop() {
  if (!estacionado) {
    if (vueltasCompletadas < 3) {
      leerSensoresYESquivar();
      contarVueltas();
    } else {
      if (bloqueMagentaEncontrado) {
        buscarYEstacionarse();
      } else {
        parar();
        estacionado = true;
      }
    }
  }
}

// Funciones para Controlar Motores y Servos
void adelante() {
  digitalWrite(motorIn1, HIGH);
  digitalWrite(motorIn2, LOW);
  analogWrite(motorEn, 255); // Velocidad máxima
}

void atras() {
  digitalWrite(motorIn1, LOW);
  digitalWrite(motorIn2, HIGH);
  analogWrite(motorEn, 255); // Velocidad máxima
}

```

```

void parar() {
    digitalWrite(motorIn1, LOW);
    digitalWrite(motorIn2, LOW);
    analogWrite(motorEn, 0); // Parar
}

void girarIzquierda() {
    servoDireccion.write(45); // Ajustar el valor según sea necesario
}

void girarDerecha() {
    servoDireccion.write(135); // Ajustar el valor según sea necesario
}

void centrarDireccion() {
    servoDireccion.write(90); // Posición central
}

// Funciones para Leer Datos de los Sensores
int leerDistanciaLidar() {
    int distancia = 0;
    if (lidarSerial.available()) {
        // Leer y procesar la distancia del Lidar
        while (lidarSerial.available() > 0) {
            int c = lidarSerial.read();
            distancia = (distancia << 8) | c;
        }
    }
    return distancia;
}

void moverPixy() {
    static bool direccionDerecha = true;

    if (direccionDerecha) {
        servoH.write(180);
        delay(500); // Ajustar tiempo según sea necesario
        servoH.write(90);
        delay(500); // Ajustar tiempo según sea necesario
    } else {
        servoH.write(0);
        delay(500); // Ajustar tiempo según sea necesario
        servoH.write(90);
        delay(500); // Ajustar tiempo según sea necesario
    }

    direccionDerecha = !direccionDerecha;
}

void leerSensoresYESquivar() {
    int distanciaFrontal = leerDistanciaLidar();

    // Evitar chocar con los bordes de la pista
    if (distanciaFrontal < 30) {
        girarDerecha();
        adelante();
        delay(1000); // Ajustar tiempo según sea necesario
        centrarDireccion();
    }
}

```

```

    }

    // Leer datos del Pixy2
    pixy.ccc.getBlocks();
    if (pixy.ccc.numBlocks) {
        for (int i = 0; i < pixy.ccc.numBlocks; i++) {
            if (pixy.ccc.blocks[i].m_signature == 1) { // Supongamos que 1 es
verde
                girarIzquierda();
                adelante();
                delay(1000); // Ajustar tiempo según sea necesario
                centrarDireccion();
                break; // Salir del bucle al detectar un bloque verde
            } else if (pixy.ccc.blocks[i].m_signature == 2) { // Supongamos que
2 es rojo
                girarDerecha();
                adelante();
                delay(1000); // Ajustar tiempo según sea necesario
                centrarDireccion();
                break; // Salir del bucle al detectar un bloque rojo
            } else if (pixy.ccc.blocks[i].m_signature == 3) { // Supongamos que
3 es magenta
                bloqueMagentaEncontrado = true;
            }
        }
    }

    moverPixy(); // Mover la cámara Pixy2 para buscar bloques
}

// Lógica de Navegación para Esquivar Obstáculos
void esquivarObstaculo() {
    parar();
    girarDerecha();
    adelante();
    delay(1000); // Ajustar tiempo según sea necesario
    centrarDireccion();
}

// Contador de Vueltas
void contarVueltas() {
    static int posicionInicial = leerPosicionInicial();
    int posicionActual = leerPosicionActual();

    if (posicionActual == posicionInicial) {
        vueltasCompletadas++;
    }
}

int leerPosicionInicial() {
    int posicion = 0;
    // Código para leer y retornar la posición inicial
    return posicion;
}

int leerPosicionActual() {
    int posicion = 0;
    // Código para leer y retornar la posición actual
    return posicion;
}

```



```

}

// Buscar y Estacionarse en el Bloque Magenta
void buscarYEstacionarse() {
    unsigned long tiempoInicio = millis();
    bool bloqueMagentaEncontrado = false;
    int contadorMagenta = 0;

    while (millis() - tiempoInicio < 30000) { // Buscar durante 30 segundos
        pixy.ccc.getBlocks();
        if (pixy.ccc.numBlocks) {
            for (int i = 0; i < pixy.ccc.numBlocks; i++) {
                if (pixy.ccc.blocks[i].m_signature == 3) { // Supongamos que 3 es
magenta
                    contadorMagenta++;
                    if (contadorMagenta >= 2) {
                        bloqueMagentaEncontrado = true;
                        break;
                    }
                }
            }
        }
        if (bloqueMagentaEncontrado) break;
        adelante();
    }

    if (bloqueMagentaEncontrado) {
        parar();
        estacionar();
    } else {
        parar();
        estacionado = true;
    }
}

void estacionar() {
    unsigned long tiempoInicio = millis();
    while (millis() - tiempoInicio < 5000) { // Estacionar durante 5
segundos
        int distanciaFrontal = leerDistanciaLidar();
        if (distanciaFrontal < 30) {
            girarDerecha();
            adelante();
            delay(1000); // Ajustar tiempo según sea necesario
            centrarDireccion();
        } else {
            adelante();
        }
    }
    parar();
    estacionado = true; // Marcar como estacionado
}

```

Este código ajusta la lógica para buscar y estacionarse después de completar tres vueltas. Durante el estacionamiento, el vehículo verifica constantemente la distancia usando el Lidar para evitar chocar con los bordes del cuadro magenta.

DEBE QUEDAR ENTRE LOS DOS BLOQUES MAGENTA EN LA SECCION DEL BORDE DE LA PISTA DE TRES METROS CUADRADO

ChatGPT

Entiendo, el auto debe estacionarse entre dos bloques magenta que están en una sección específica de la pista de tres metros cuadrados. Vamos a ajustar el código para que el auto busque y se estacione entre los dos bloques magenta y evite chocar con los bordes.

Código ajustado:

```
cpp
Copiar código
#include <SoftwareSerial.h>
#include <Pixy2.h>
#include <Servo.h>

// Definir pines
const int motorIn1 = 5;
const int motorIn2 = 6;
const int motorEn = 3;
const int servoDireccionPin = 9;
const int servoHPin = 10;
const int servoVPin = 11;

SoftwareSerial lidarSerial(18, 19); // RX, TX
Pixy2 pixy;
Servo servoDireccion;
Servo servoH;
Servo servoV;

int vueltasCompletadas = 0;
bool estacionado = false;
bool bloqueMagentaEncontrado = false;

void setup() {
  Serial.begin(9600);
  lidarSerial.begin(115200);
  pixy.init();
  servoDireccion.attach(servoDireccionPin);
  servoH.attach(servoHPin);
  servoV.attach(servoVPin);

  pinMode(motorIn1, OUTPUT);
  pinMode(motorIn2, OUTPUT);
  pinMode(motorEn, OUTPUT);
}

void loop() {
  if (!estacionado) {
    if (vueltasCompletadas < 3) {
      leerSensoresYESquivar();
      contarVueltas();
    } else {
      if (bloqueMagentaEncontrado) {
```

```

        buscarYEstacionarse();
    } else {
        parar();
        estacionado = true;
    }
}
}

// Funciones para Controlar Motores y Servos
void adelante() {
    digitalWrite(motorIn1, HIGH);
    digitalWrite(motorIn2, LOW);
    analogWrite(motorEn, 255); // Velocidad máxima
}

void atras() {
    digitalWrite(motorIn1, LOW);
    digitalWrite(motorIn2, HIGH);
    analogWrite(motorEn, 255); // Velocidad máxima
}

void parar() {
    digitalWrite(motorIn1, LOW);
    digitalWrite(motorIn2, LOW);
    analogWrite(motorEn, 0); // Parar
}

void girarIzquierda() {
    servoDireccion.write(45); // Ajustar el valor según sea necesario
}

void girarDerecha() {
    servoDireccion.write(135); // Ajustar el valor según sea necesario
}

void centrarDireccion() {
    servoDireccion.write(90); // Posición central
}

// Funciones para Leer Datos de los Sensores
int leerDistanciaLidar() {
    int distancia = 0;
    if (lidarSerial.available()) {
        // Leer y procesar la distancia del Lidar
        while (lidarSerial.available() > 0) {
            int c = lidarSerial.read();
            distancia = (distancia << 8) | c;
        }
    }
    return distancia;
}

void moverPixy() {
    static bool direccionDerecha = true;

    if (direccionDerecha) {
        servoH.write(180);
        delay(500); // Ajustar tiempo según sea necesario
    }
}

```

```

        servoH.write(90);
        delay(500); // Ajustar tiempo según sea necesario
    } else {
        servoH.write(0);
        delay(500); // Ajustar tiempo según sea necesario
        servoH.write(90);
        delay(500); // Ajustar tiempo según sea necesario
    }

    direccionDerecha = !direccionDerecha;
}

void leerSensoresYESquivar() {
    int distanciaFrontal = leerDistanciaLidar();

    // Evitar chocar con los bordes de la pista
    if (distanciaFrontal < 30) {
        girarDerecha();
        adelante();
        delay(1000); // Ajustar tiempo según sea necesario
        centrarDireccion();
    }

    // Leer datos del Pixy2
    pixy.ccc.getBlocks();
    if (pixy.ccc.numBlocks) {
        for (int i = 0; i < pixy.ccc.numBlocks; i++) {
            if (pixy.ccc.blocks[i].m_signature == 1) { // Supongamos que 1 es
verde
                girarIzquierda();
                adelante();
                delay(1000); // Ajustar tiempo según sea necesario
                centrarDireccion();
                break; // Salir del bucle al detectar un bloque verde
            } else if (pixy.ccc.blocks[i].m_signature == 2) { // Supongamos que
2 es rojo
                girarDerecha();
                adelante();
                delay(1000); // Ajustar tiempo según sea necesario
                centrarDireccion();
                break; // Salir del bucle al detectar un bloque rojo
            } else if (pixy.ccc.blocks[i].m_signature == 3) { // Supongamos que
3 es magenta
                bloqueMagentaEncontrado = true;
            }
        }
    }

    moverPixy(); // Mover la cámara Pixy2 para buscar bloques
}

// Lógica de Navegación para Esquivar Obstáculos
void esquivarObstaculo() {
    parar();
    girarDerecha();
    adelante();
    delay(1000); // Ajustar tiempo según sea necesario
    centrarDireccion();
}

```

```

// Contador de Vueltas
void contarVueltas() {
    static int posicionInicial = leerPosicionInicial();
    int posicionActual = leerPosicionActual();

    if (posicionActual == posicionInicial) {
        vueltasCompletadas++;
    }
}

int leerPosicionInicial() {
    int posicion = 0;
    // Código para leer y retornar la posición inicial
    return posicion;
}

int leerPosicionActual() {
    int posicion = 0;
    // Código para leer y retornar la posición actual
    return posicion;
}

// Buscar y Estacionarse en el Bloque Magenta
void buscarYEstacionarse() {
    unsigned long tiempoInicio = millis();
    bool bloqueMagentaEncontrado = false;
    int contadorMagenta = 0;

    while (millis() - tiempoInicio < 30000) { // Buscar durante 30 segundos
        pixy.ccc.getBlocks();
        if (pixy.ccc.numBlocks) {
            for (int i = 0; i < pixy.ccc.numBlocks; i++) {
                if (pixy.ccc.blocks[i].m_signature == 3) { // Supongamos que 3 es
magenta
                    contadorMagenta++;
                    if (contadorMagenta >= 2) {
                        bloqueMagentaEncontrado = true;
                        break;
                    }
                }
            }
        }
        if (bloqueMagentaEncontrado) break;
        adelante();
    }

    if (bloqueMagentaEncontrado) {
        parar();
        estacionarEntreBloquesMagenta();
    } else {
        parar();
        estacionado = true;
    }
}

void estacionarEntreBloquesMagenta() {
    unsigned long tiempoInicio = millis();

```

```

    while (millis() - tiempoInicio < 5000) { // Estacionar durante 5
segundos
        int distanciaFrontal = leerDistanciaLidar();
        if (distanciaFrontal < 30) {
            girarDerecha();
            adelante();
            delay(1000); // Ajustar tiempo según sea necesario
            centrarDireccion();
        } else {
            adelante();
        }
    }
    parar();
    estacionado = true; // Marcar como estacionado
}

```

Explicación de cambios:

- moverPixy: La cámara Pixy2 se mueve horizontalmente de izquierda a derecha para buscar bloques.
- leerSensoresYESquivar: Se añadió la función `moverPixy()` para que la cámara se mueva mientras busca bloques.
- buscarYEstacionarse: El auto busca el bloque magenta durante 30 segundos después de completar tres vueltas. Si encuentra el bloque magenta, se detiene y llama a la función `estacionarEntreBloquesMagenta`.
- estacionarEntreBloquesMagenta: El auto se mueve hacia adelante y usa el Lidar para evitar chocar con los bordes del cuadro mientras se estaciona.