DISEÑO DEL PROYECTO DE AULA

SARA LUCIA DUQUE PARRA

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
POPAYÁN- CAUCA
2024

ROBOT MINISUMO

SARA LUCIA DUQUE PARRA

INGENIERO CARLOS HERNAN TOBAR ARTEAGA

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES
POPAYÁN- CAUCA
2024

RESUMEN

El proyecto de aula "Robot Minisumo" se enfoca en el diseño de sistemas digitales basados en microcontroladores para crear un robot autónomo competente en combates de sumo. El proceso de diseño abarca varios pasos: la especificación del propósito y requisitos, la descripción formal de los casos de uso, la especificación del modelo de dominio y de información, y finalmente, la integración de dispositivos y componentes. El robot Minisumo debe ser autónomo, con dimensiones máximas de 10cm x 10cm y 500g, tracción libre, seguridad con switch visible y tiempo de espera, área de combate circular, fases eliminatorias, y disponibilidad local de la interfaz de control. Este proyecto se llevó a cabo en Wokwi.com, desde la especificación inicial hasta la integración final de dispositivos y componentes para lograr un robot Minisumo competente en combates autónomos.

METODOLOGÍA DE DISEÑO DE SISTEMAS DIGITALES BASADOS EN MICROCONTROLADORES
"ROBOT MINISUMO"

Paso 1. Especificación del propósito y requisitos.

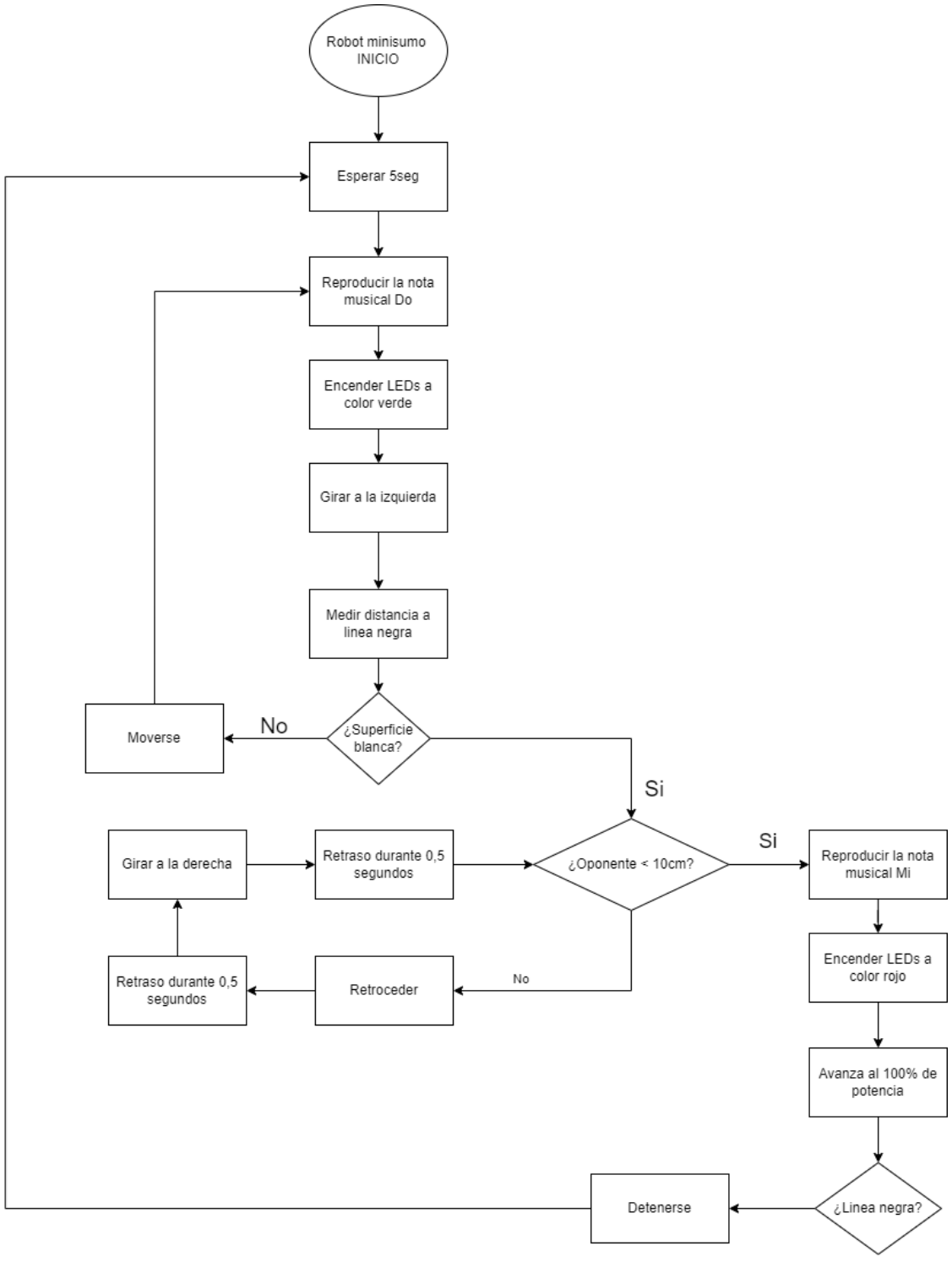
En este paso se captura el propósito del sistema, el comportamiento y requisitos, tales como requisitos de recolección de datos, requisitos de análisis de datos, requisitos de gestión del sistema, requisitos de seguridad y privacidad de datos, y requisitos de interfaz de usuario, entre otros.

Propósito	Robot Minisumo autónomo capaz de competir en un área de combate para lograr que el robot oponente salga del área de combate.
Comportamiento	• El robot Minisumo debe ser autónomo, sin necesidad de mando a distancia. • Una vez iniciado el robot inicie el combate debe tener la capacidad de desplegar los comandos.
Requisitos Generales	• **Dimensiones y peso:** El robot no debe exceder las dimensiones de 10cm x 10cm y 500g de peso. • **Tipo de tracción:** La tracción del robot es de libre elección. • **Seguridad:** Debe contar con un switch visible para encendido y apagado, así como un tiempo de espera antes de iniciar el combate. • **Área de combate:** El dojo debe ser circular, con ciertas dimensiones y características específicas. • **Grupos y combates:** Se realizarán enfrentamientos entre robots en grupos, con fases eliminatorias hasta llegar a la final. • La interfaz de control del robot debe estar disponible localmente en el dispositivo.
Requisito de gestión del sistema	El robot debe ofrecer funciones de **control automático** para ajustar su comportamiento y recibir información sobre su comportamiento en el combate.
Requisito de análisis de datos	El sistema debe ser capaz de analizar los datos recopilados durante las competencias para mejorar su estrategia y rendimiento.
Requisito de seguridad	Se requiere que el robot sea seguro para el usuario, iniciando su operación una vez el usuario se retira.

Tabla 1: Especificación del propósito y requisitos.

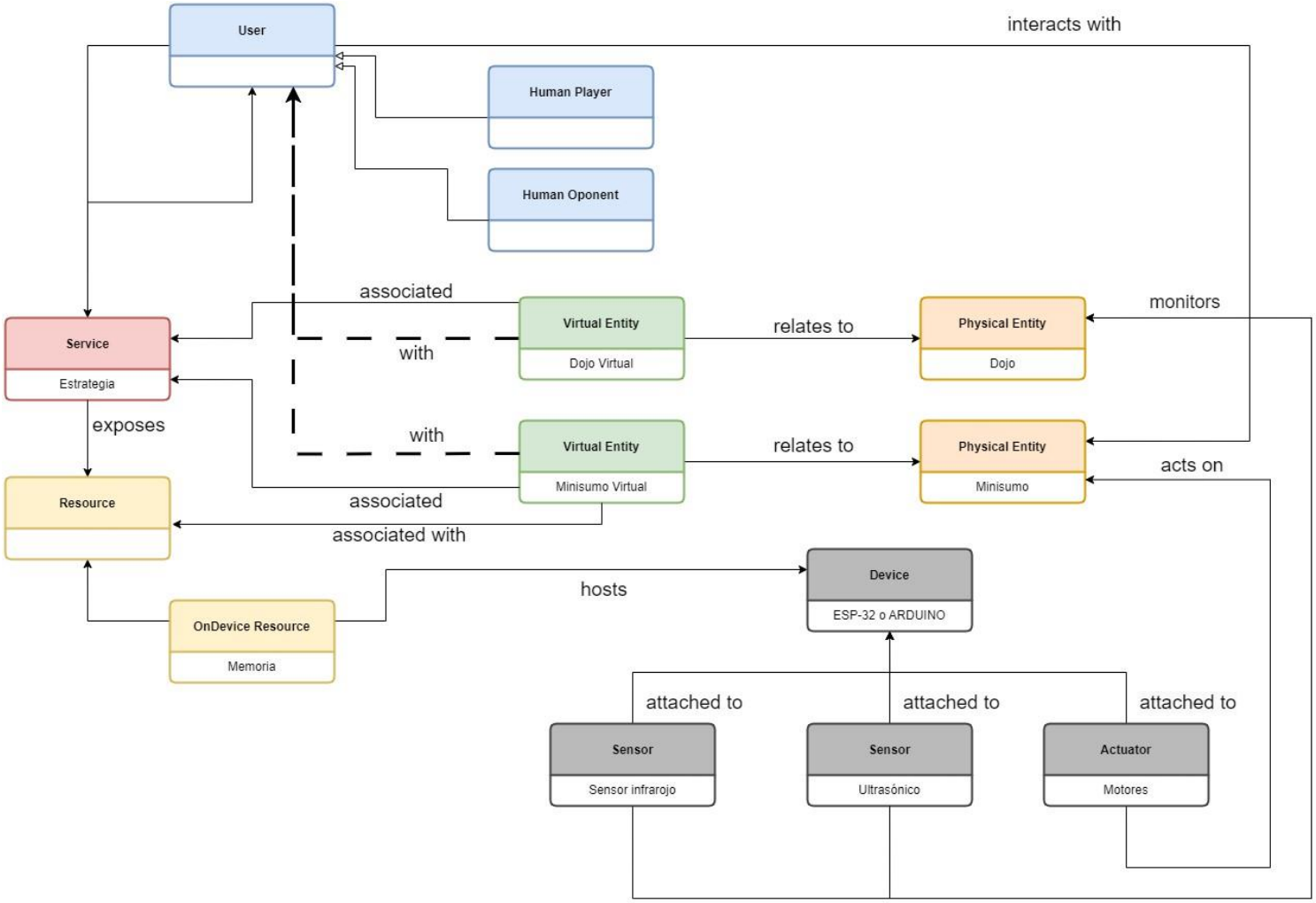
Paso 2. Especificación del proceso.

En este paso se describen formalmente los casos de uso del sistema a partir de la especificación del propósito y requisitos. Se enfoca en detallar cómo el sistema será utilizado y las interacciones que tendrá el robot (MiniSumo) con el oponente, proporcionando una visión clara de las funcionalidades que debe cumplir.



Paso 3. Especificación del modelo de dominio.

El modelo de dominio describe los conceptos principales, entidades y objetos en el dominio del sistema que se está diseñando. Se describen los atributos de los objetos y las relaciones entre ellos. Provee una representación abstracta de los conceptos, objetos y entidades en el dominio del sistema, independientemente de cualquier tecnología o plataforma específica.



Paso 4. Especificación del modelo de información.

El modelo de información define la estructura de toda la información en el sistema digital, por ejemplo, atributos de entidades, relaciones, etc., proporciona una visión clara de cómo se organiza la información en el sistema sin entrar en detalles de implementación. Es fundamental para el diseño y desarrollo de la arquitectura del sistema digital, facilitando la comprensión de la información que será manejada por el robot autónomo durante los combates de sumo.

El diagrama representa un modelo de entidad-atributo que describe las características y comportamientos de dos "Entidades Virtuales": **"Dojo Virtual"** y **"MiniSumo"**. A continuación, se detallan los componentes del diagrama:

1. **Dojo Virtual**: Dojo Virtual es un entorno donde se lleva a cabo la competencia de Minisumos

Tipo de Entidad: Dojo **ID**: Dojo1	**Atributos**: • **superficieColor**: Representa el color de la superficie del dojo (por ejemplo, blanco o negro). • **Espacio**: Indica si el MiniSumo está dentro o fuera del dojo.

2. **MiniSumo**: Robot autónomo que compite en el Dojo Virtual

Tipo de Entidad: MiniSumo **ID**: MiniSumo1-1	**Atributos**: • **tipoDeVelocidad**: Representa la velocidad del MiniSumo (puede ser rápido, lento, girar o desplazarse). • **posición**: Indica la posición actual del MiniSumo (identificado, no identificado, movimiento, empuje o espera). • **oponente**: Es un atributo booleano que indica si el MiniSumo ha detectado un oponente. • **estrategia**: Le indicará al robot que deberá hacer en la zona de competencia

En resumen, el diagrama muestra cómo se relacionan las entidades virtuales y sus atributos en el contexto de una competencia de Minisumos. El **Dojo Virtual** proporciona el entorno, mientras que el **MiniSumo** tiene atributos como velocidad, posición, detección de oponente y estrategia.

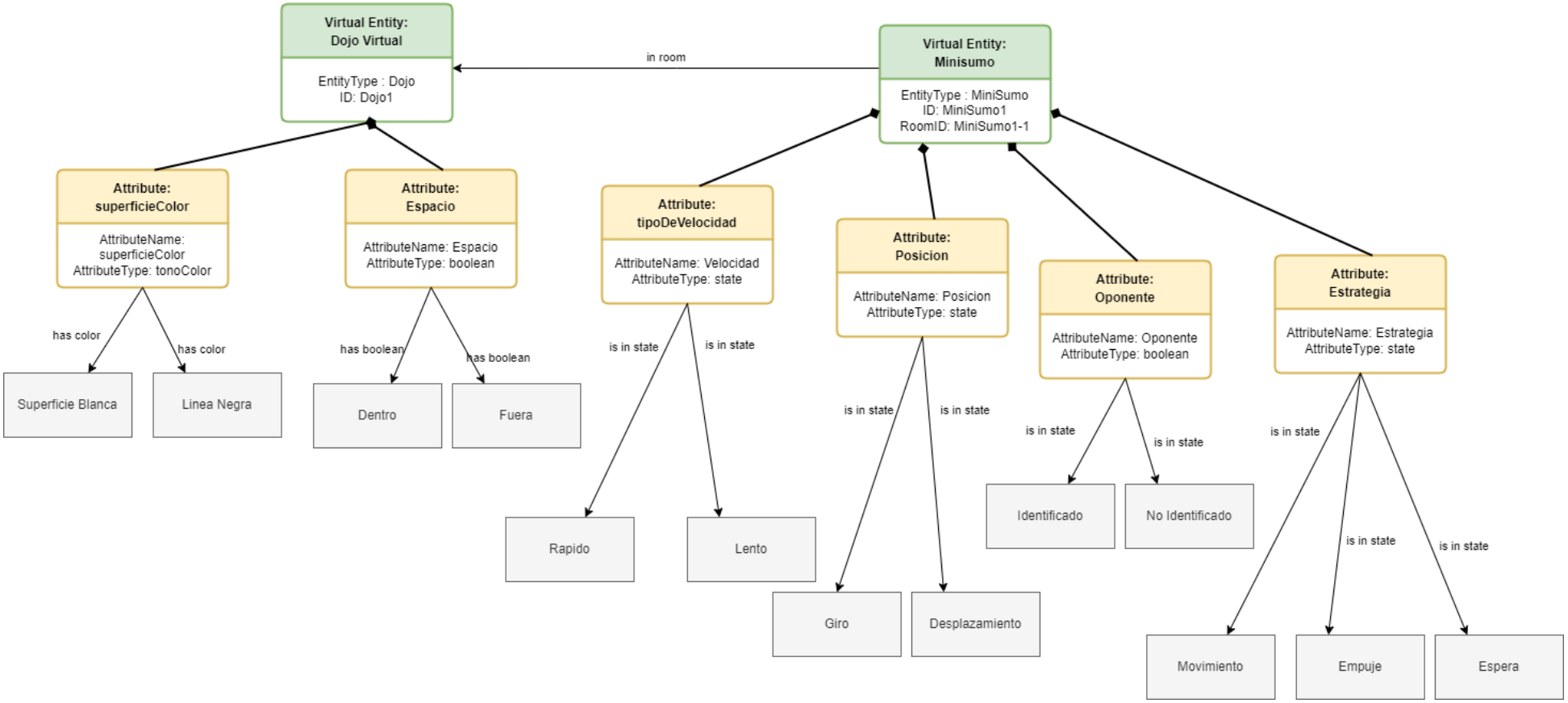
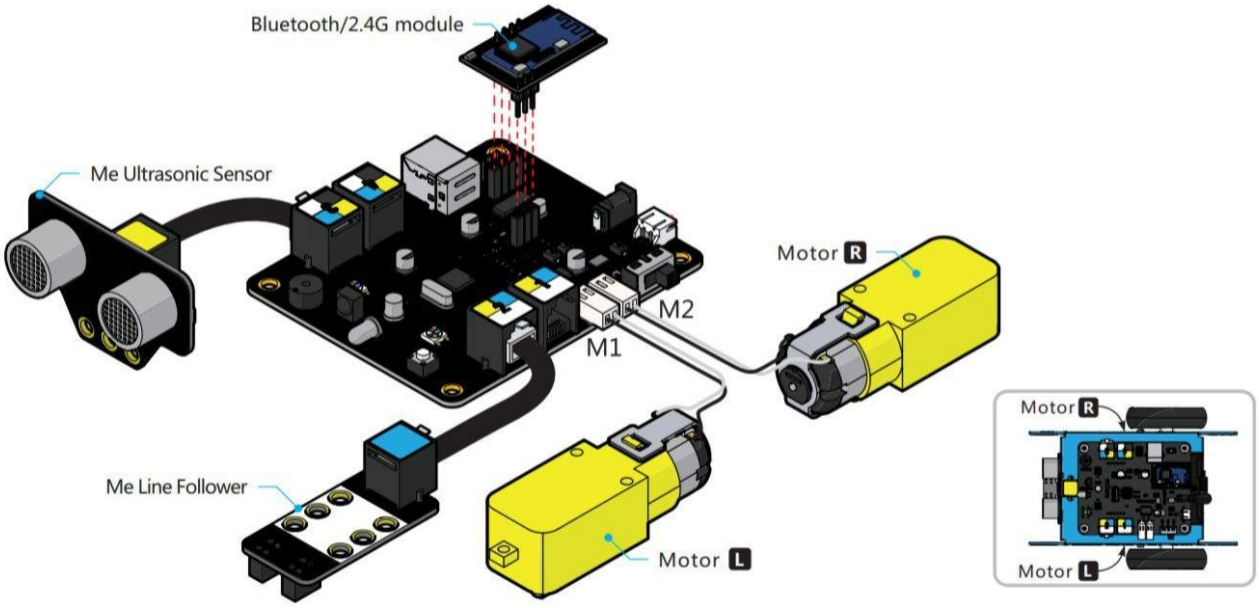


Ilustración 3: Especificación del modelo de información.

Paso 5. Integración de dispositivos y componentes.

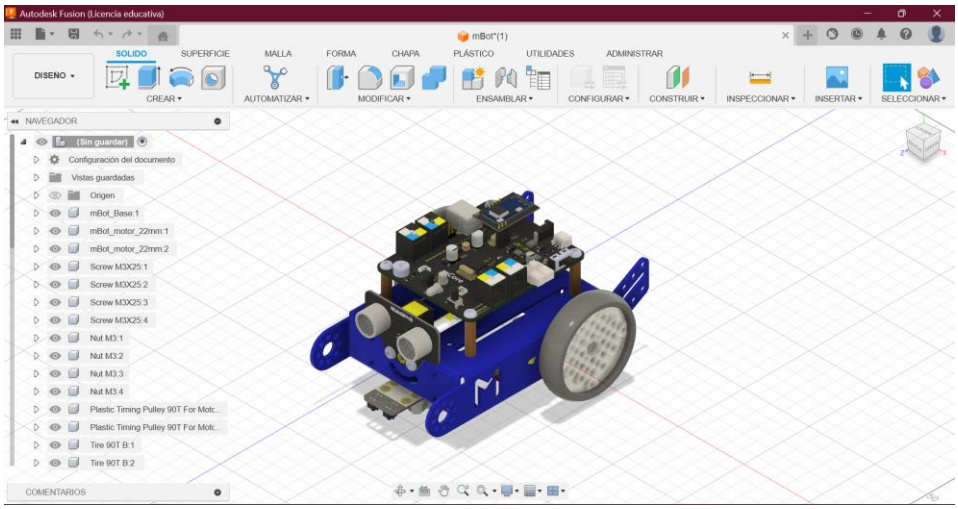
Se enfoca en la integración de dispositivos y componentes, donde se unen y ensamblan todas las partes diseñadas previamente para crear un sistema funcional y operativo. Con esta etapa, se lleva a cabo la conexión y coordinación de los diferentes elementos del robot, asegurando que trabajen de manera conjunta y eficiente para lograr el objetivo final de competir en combates de sumo de forma autónoma. La integración de dispositivos y componentes es crucial para garantizar el correcto funcionamiento y desempeño del robot durante las competencias.

El componente principal de un robot Minisumo es su placa controladora (ESP32), que contiene el cerebro del robot y se encarga de procesar las señales de los sensores y controlar los motores. Además, los Minisumos suelen tener sensores infrarrojos para detectar al oponente y motores para moverse y atacar.



Paso 6. Desarrollo:

a) **Construcción física (Hardware):**



La construcción del robot mBot comienza con el **chasis**, que es la base del robot. El chasis es un componente fundamental que proporciona la estructura para montar los demás componentes del robot. En el caso del mBot, el chasis es una caja metálica que contiene el **mCore**, el microcontrolador principal del robot, y otros componentes como sensores y actuadores.

Los **motores** son un elemento crucial en el mBot. Estos motores son conectados al chasis mediante tornillos M3 y sus correspondientes tuercas. Una vez montados los motores, se añaden las **ruedas** a cada motor utilizando tornillos de auto-performación. Esto permite que el robot pueda moverse y realizar acciones físicas.

Además del chasis y los motores, el mBot incluye varios **sensores** que permiten al robot interactuar con su entorno. Estos sensores incluyen un **sensor sigue-líneas**, que ayuda al robot a seguir una línea o ruta específica, y un **sensor ultrasónico**, que permite al robot detectar obstáculos y evitarlos.

El **sensor sigue-líneas** se monta en la parte frontal del chasis, lo que mejora su funcionalidad y facilita su uso. El **sensor ultrasónico** también se monta en el chasis, aunque puede requerir ajustes para asegurarse de que se mantenga en su lugar y funcione correctamente.

El mBot también incluye **actuadores**, como **LEDs RGB** y un **buzzer**, que permiten al robot realizar acciones visuales y auditivas. Estos componentes se montan en el chasis y se conectan a la placa principal del robot.

En resumen, la construcción del mBot implica montar el chasis, los motores, los sensores y los actuadores en un orden específico para crear un robot que pueda interactuar con su entorno y realizar tareas específicas.

b) Codificación del programa:

```
#include <MeMCore.h> // Incluir la biblioteca para el robot mBot de
Makeblock
#include <Arduino.h> // Incluir la biblioteca principal de Arduino
#include <Wire.h> // Incluir la biblioteca Wire para comunicación
I2C
#include <SoftwareSerial.h> // Incluir la biblioteca SoftwareSerial
para comunicación serial por software
```

Bibliotecas Incluidas:

- **MeMCore.h** es la biblioteca específica para controlar el robot mBot de Makeblock.
- **Arduino.h** es la biblioteca principal de Arduino, necesaria para todas las funciones básicas.
- **Wire.h** es utilizada para la comunicación I2C.
- **SoftwareSerial.h** permite la comunicación serial en cualquier pin digital.

```
MeLineFollower linefollower_2(3);
MeUltrasonicSensor ultrasonic_3(1);
MeBuzzer buzzer;
MeRGBLed rgbled_7(7, 2);
MeDCMotor motor_9(9);
MeDCMotor motor_10(10);
```

Inicialización de Componentes:

- **linefollower_2(3)** inicializa un sensor seguidor de línea en el puerto 3.
- **ultrasonic_3(1)** inicializa un sensor ultrasónico en el puerto 1.
- buzzer inicializa un zumbador.
- **rgbled_7(7, 2)** inicializa un LED RGB en los puertos 7 y 2.
- **motor_9(9) y motor_10(10)** inicializan motores DC en los puertos 9 y 10.

```
void move(int direction, int speed) {
  int leftSpeed = 0;
  int rightSpeed = 0;
  if(direction == 1) {
    leftSpeed = speed;
    rightSpeed = speed;
  } else if(direction == 2) {
    leftSpeed = -speed;
    rightSpeed = -speed;
  } else if(direction == 3) {
    leftSpeed = -speed;
    rightSpeed = speed;
  } else if(direction == 4) {
    leftSpeed = speed;
    rightSpeed = -speed;
  }
  motor_9.run((9) == M1 ? -(leftSpeed) : (leftSpeed));
  motor_10.run((10) == M1 ? -(rightSpeed) : (rightSpeed));
}
```

Función de Movimiento:

- **move(int direction, int speed)** controla la dirección y velocidad del mBot.
- Los parámetros **direction (1 adelante, 2 atrás, 3 izquierda, 4 derecha)** y speed (velocidad) determinan cómo se moverán los motores izquierdo y derecho.

```
void _delay(float seconds) {
  long endTime = millis() + seconds * 1000;
  while(millis() < endTime) _loop();
}
```

Función de Retardo:

- **_delay(float seconds)** crea un retardo en segundos, permitiendo que otras tareas se ejecuten durante este tiempo.

```
void avanzarHastaDetectarLinea() {
  while (true) {
    int sensorValue = linefollower_2.readSensors();
    if (sensorValue == 0) {
      break;
    }
    move(1, 80 / 100.0 * 255);
    _loop();
  }
  motor_9.run(0);
  motor_10.run(0);
}
```

Avanzar hasta detectar línea:

- avanzarHastaDetectarLinea() hace que el mBot avance hasta que el sensor de línea detecte una línea negra, momento en el cual se detiene.

```
void setup() {
  Serial.begin(9600);
  pinMode(A7, INPUT);
  rgbled_7.fillPixelsBak(0, 2, 1);
  while(!(analogRead(A7) > 10)) {
    _loop();
  }
  while(analogRead(A7) > 10) {
    _loop();
  }

  long startTime = millis();
  while(millis() - startTime < 5000) {
    buzzer.tone(175, 0.25 * 1000);
    _delay(0.75);
    rgbled_7.setColor(0, 255, 0, 157);
    rgbled_7.show();
    _loop();
  }

  rgbled_7.setColor(0, 0, 21, 255);
  rgbled_7.show();

  avanzarHastaDetectarLinea();

  while(1) {
    int distancia_ring_linea = linefollower_2.readSensors();
    Serial.print("Distancia de la Línea: ");
    Serial.println(distancia_ring_linea);

    while(distancia_ring_linea != 0) {
      _loop();
```

```
      double distancia = ultrasonic_3.distanceCm();
      Serial.print("Distancia del oponente: ");
      Serial.println(distancia);

      if(distancia < 10) {
        buzzer.tone(165, 0.25 * 1000);
        _delay(0.02);
        rgbled_7.setColor(0, 255, 0, 0);
        rgbled_7.show();
        move(1, 100 / 100.0 * 255);
      } else {
        motor_9.run(0);
        motor_10.run(0);
      }
      distancia_ring_linea = linefollower_2.readSensors();
    }

    buzzer.tone(65, 1 * 1000);
    _delay(0.02);
    rgbled_7.setColor(0, 140, 255, 0);
    rgbled_7.show();

    move(3, 60 / 100.0 * 255);
    _delay(0.5);
    move(3, 80 / 100.0 * 255);
    _delay(0.2);
  }
}
```

Configuración Inicial:

- **setup()** configura la comunicación serial y los pines necesarios.
- Inicializa el LED RGB y espera a que el valor del pin A7 sea mayor o menor a 10.
- Un bucle espera 5 segundos mientras suena el buzzer y cambia el color del LED RGB.
- Cambia el color del LED a azul para indicar que está buscando la línea y llama a avanzarHastaDetectarLinea().
- Dentro de un bucle infinito, mide la distancia de la línea y del oponente.
- Si detecta un oponente a menos de 10 cm, reproduce un tono y enciende el LED en rojo, moviéndose hacia adelante.
- Si no detecta al oponente, detiene los motores.
- Si la línea no es detectada, reproduce un tono y enciende el LED en verde, girando para buscar la línea.

```
void _loop() {
}
void loop() {
  _loop();
}
```

Bucle Principal:

_loop() es una función vacía que se usa dentro de los retardos y espera.

loop() llama a _loop() en un bucle continuo, asegurando que _loop() siempre esté disponible para su uso.

Paso 7. Pruebas:

1. **PRUEBA DE ESPERA DE 5 SEG ANTES DE EMPEZAR EL RING:**

```
// Añadir el nuevo while que espera 5 segundos y realiza las
acciones específicas
long startTime = millis();
while(millis() - startTime < 5000) {     // Esperar 5 segundos
      // Tocar el pitido
      buzzer.tone(175, 0.25 * 1000);
      _delay(0.75);                       // Esperar 0,75 segundos
      para que el pitido se repita cada segundo

      // Encender LED con color específico
      rgbled_7.setColor(0, 255, 0, 157);
      rgbled_7.show();
      _loop();
  }
```

En este fragmento del código, se ha añadido un bucle `while` que espera 5 segundos y realiza acciones específicas durante este tiempo. Primero, se almacena el tiempo de inicio utilizando `millis()`, que devuelve el número de milisegundos transcurridos desde que el programa comenzó a ejecutarse. Luego, el bucle `while` se ejecuta mientras la diferencia entre el tiempo actual y `startTime` sea menor que 5000 milisegundos (5 segundos). Dentro del bucle, el buzzer reproduce un pitido con una duración de 0.25 segundos, seguido de un retardo de 0.75 segundos, haciendo que el pitido se repita cada segundo. Simultáneamente, el LED RGB se enciende con un color específico, rosado, (valores RGB: 255, 0, 157) y se muestra el color. La llamada a `_loop()` permite que se ejecuten otras tareas mientras se espera, manteniendo el sistema receptivo a otras posibles operaciones. Este bucle esencialmente crea un efecto de sonido y luz durante un periodo de 5 segundos al inicio del programa.

2. PRUEBA DE DETECCIÓN DE LINEA:

```
// Encender LED en azul para indicar que está buscando la distancia
de la línea
  rgbled_7.setColor(0, 0, 21, 255); // Establecer el color del LED
RGB en azul
  rgbled_7.show(); // Mostrar el color

avanzarHastaDetectarLinea();

------------------------------------------------------------
void avanzarHastaDetectarLinea() {
        while (true) {
          int sensorValue = linefollower_2.readSensors();
          if (sensorValue == 0) { // Si los sensores detectan la
      línea (posiblemente, dependiendo de cómo estén configurados)
            break; // Salir del bucle si se detecta la línea
          }
          move(1, 80 / 100.0 * 255); // Mueve hacia adelante a
      velocidad máxima
          _loop();
        }
        // Detener el movimiento una vez que se detecta la línea
        motor_9.run(0);
        motor_10.run(0);
}
```

En este fragmento del código, primero se enciende el LED RGB en color azul para indicar que el mBot está en modo de búsqueda de la línea. La función `rgbled_7.setColor(0, 0, 21, 255)` establece el color del LED RGB con valores específicos (R: 0, G: 0, B: 21, brillo: 255), y `rgbled_7.show()` actualiza el LED para mostrar el color seleccionado.

A continuación, se llama a la función `avanzarHastaDetectarLinea()`. Dentro de esta función, se ejecuta un bucle `while (true)` que lee continuamente los valores del sensor seguidor de línea utilizando `linefollower_2.readSensors()`. Si los sensores detectan la línea (cuando `sensorValue` es 0), el bucle se rompe y el robot se detiene. Si no se detecta la línea, el robot sigue avanzando a velocidad máxima (80% de 255) usando la función `move(1, 80 / 100.0 * 255)`. La llamada a `_loop()` permite que se ejecuten otras tareas mientras el robot se mueve. Una vez que se detecta la línea y se sale del bucle, se detienen ambos motores con `motor_9.run(0)` y `motor_10.run(0)`, finalizando la búsqueda de la línea.

3. PRUEBA DE DETECTAR OPONENTE Y EMPUJARLO:

```
while(distancia_ring_linea != 0) { // Mientras se detecte la línea
            _loop(); // Permitir que se ejecuten otras tareas
      durante el bucle

            double distancia = ultrasonic_3.distanceCm(); // Medir
      la distancia utilizando el sensor ultrasónico
            Serial.print("Distancia del oponente: ");
            Serial.println(distancia);

            // OPONENTE
            if(distancia < 10) { // Si el oponente está a menos de
      10 cm
              // TOCAR NOTA MUSICAL: MI
              buzzer.tone(165, 0.25 * 1000); // Reproducir una nota
      musical (E) durante 0,25 segundos
              _delay(0.02); // Retraso durante 0,02 segundos
              // ALUMBRAR ROJO
              rgbled_7.setColor(0, 255, 0, 0); // Establecer el
      color del LED RGB en rojo
              rgbled_7.show(); // Mostrar el color

              // Mover hacia adelante a toda velocidad
              move(1, 100 / 100.0 * 255);

            } else {
              // Si el oponente está a más de 10 cm
              motor_9.run(0);
              motor_10.run(0);
            }

            distancia_ring_linea = linefollower_2.readSensors();
      // Leer nuevamente los sensores del seguidor de línea para
      detectar si se sacó al oponente
    }
```

En este fragmento del código, se inicia un bucle `while` que se ejecuta mientras el sensor seguidor de línea detecte la línea (cuando `distancia_ring_linea` es diferente de 0). Dentro de este bucle, la función `_loop()` se llama para permitir que otras tareas se ejecuten simultáneamente.

A continuación, se mide la distancia al oponente utilizando el sensor ultrasónico con `ultrasonic_3.distanceCm()`. El valor de la distancia se imprime en el monitor serial para fines de depuración con `Serial.print` y `Serial.println`.

Si el oponente se encuentra a menos de 10 cm, el buzzer reproduce una nota musical y luego se espera 0.02 segundos. Simultáneamente, el LED RGB se enciende en rojo

(`rgbled_7.setColor(0, 255, 0, 0)`) y se muestra el. El robot se mueve hacia adelante a máxima velocidad.

Si el oponente está a más de 10 cm, los motores se detienen (`motor_9.run(0)` y `motor_10.run(0)`).

Finalmente, se actualiza el valor de `distancia_ring_linea` leyendo nuevamente los sensores del seguidor de línea (`linefollower_2.readSensors()`). Este ciclo se repite mientras el seguidor de línea continúe detectando la línea, asegurando que el robot siga intentando empujar al oponente mientras permanece dentro del área delimitada.

4. **DETECCION DE LINEA DESPUES DE HABER SACADO AL OPONENTE**

   La detección de la línea después de haber sacado al oponente no está funcionando según lo esperado en el código actual. A pesar de que se mencionó anteriormente que `distancia_ring_linea` debería leer nuevamente los sensores del seguidor de línea para detectar si se ha sacado al oponente, esto no está ocurriendo en la implementación actual del código. Para abordar este problema, se están realizando cambios al código para asegurar que se vuelvan a leer los parámetros del seguidor de línea después de haber sacado al oponente.

   Sin embargo, en el video que se adjuntará se puede observar que el robot eventualmente detecta la línea después de un tiempo, pero no se puede determinar con certeza si esto se debe a la superficie sobre la que se está moviendo o si es porque el robot empujó al oponente a toda velocidad, lo que no le dio tiempo suficiente para leer los parámetros del seguidor de línea. Este comportamiento inconsistente resalta la necesidad de ajustar y mejorar la lógica de detección de línea en el código para garantizar un rendimiento más fiable del robot en situaciones de competición.

Se adjuntará un video de las pruebas realizadas al código en el repositorio de GitHub asociado al proyecto (https://github.com/luciaduque248/MBot-MiniSumo.git). Este video proporcionará una representación visual de las pruebas ejecutadas, lo que permitirá una mejor comprensión del comportamiento del robot en diversas situaciones, incluida la detección de la línea después de haber sacado al oponente. El análisis del video ayudará a identificar posibles áreas de mejora en el algoritmo y la implementación del código, lo que contribuirá a optimizar el rendimiento general del robot en la competición.

CONCLUSIÓN

Este proyecto de minisumo ha sido una experiencia invaluable para explorar y aplicar conceptos de robótica, programación y diseño de algoritmos en un contexto práctico y desafiante. A lo largo del desarrollo, se han enfrentado varios desafíos, desde la configuración y calibración de los sensores hasta la implementación y optimización de algoritmos de control de movimiento y detección de línea.

Se ha evidenciado la importancia de la iteración y la experimentación en el proceso de desarrollo, ya que se han realizado múltiples ajustes y mejoras en el código y la configuración del robot para abordar problemas y optimizar su rendimiento. Además, la documentación detallada y el análisis crítico de los resultados han sido fundamentales para comprender el funcionamiento del robot y orientar las futuras iteraciones del proyecto.

A pesar de los desafíos encontrados, se ha logrado avanzar significativamente en la construcción de un robot capaz de competir en la categoría de minisumo, con capacidades de detección de línea, evasión de obstáculos y estrategias de empuje contra oponentes. Sin embargo, queda claro que aún hay margen para mejorar y refinar el diseño y el rendimiento del robot, especialmente en lo que respecta a la detección de la línea después de haber sacado al oponente.

En resumen, este proyecto de minisumo ha sido una oportunidad emocionante para aplicar habilidades técnicas, resolver problemas prácticos y aprender de manera práctica sobre robótica y programación. A través del análisis de los resultados y la iteración continua, se espera continuar mejorando y refinando el robot para alcanzar un rendimiento óptimo en futuras competiciones y desafíos.