Integración de sistemas mecatrónicos con ATmega328P: comunicación SPI/I^{2}C, matriz LED RGB y robot de mini fútbol

1° Lucas Elizalde
Ingeniería en Mecatrónica
UTEC
Fray Bentos, Río Negro
lucas.elizalde@estudiantes.utec.edu.uy

2° Juan Manuel Ferreira
Ingeniería en Mecatrónica
UTEC
Fray Bentos, Río Negro
juan.ferreira.m@estudiantes.utec.edu.uy

3° Felipe Morrudo
Ingeniería en Mecatrónica
UTEC
Fray Bentos, Río Negro
felipe.morrudo@estudiantes.utec.edu.uy

***Resumen*—En este informe se presenta el diseño e implementación de cuatro sistemas mecatrónicos desarrollados con el microcontrolador ATmega328P: una arquitectura maestro–esclavo con comunicación SPI para el sensado de variables ambientales y el control distribuido de actuadores; una segunda arquitectura equivalente utilizando el protocolo I^{2}C, permitiendo comparar el desempeño, cableado y complejidad de ambos buses; un sistema de visualización dinámica en una matriz LED RGB 8x8 basado en animaciones generadas por UART y el protocolo WS2812B; y un módulo interactivo que desplaza un LED sobre la matriz a partir de la inclinación captada por el sensor inercial MPU6050, incorporando además cambio de color mediante entrada digital. Finalmente, se implementó un robot de mini fútbol con tracción diferencial, sensores de línea, mecanismo frontal de golpeo y comunicación Bluetooth para control remoto.**

Las pruebas en simulación (Proteus) y en montaje físico demostraron un funcionamiento estable en todos los subsistemas, validando la capacidad del ATmega328P para integrar adquisición analógica (ADC), control PWM, comunicación serial (UART, SPI, I^{2}C), control de Matriz de Leds con giroscopio como con animaciones.

***Index Terms*—ATmega328P, SPI, I^{2}C, UART, matriz LED RGB, WS2812B, MPU6050, robot móvil, Bluetooth, sistemas embebidos**

I. Introducción

El presente informe tiene como finalidad documentar el desarrollo, implementación y análisis de los distintos sistemas mecatrónicos propuestos en el Laboratorio 4 de la unidad curricular *Tecnologías de los Microprocesadores*. Todas las prácticas se llevaron a cabo utilizando el microcontrolador ATmega328P, integrando conceptos de comunicación digital (SPI, I^{2}C y UART), control de actuadores, sensado, procesamiento de datos y visualización mediante matrices RGB. El trabajo articula los contenidos teóricos del curso con su aplicación práctica en laboratorio, garantizando una comprensión integral del diseño, programación y validación de sistemas embebidos.

En primera instancia, se aborda el diseño de arquitecturas maestro–esclavo utilizando comunicación SPI e I^{2}C. En ambos casos, un microcontrolador se encarga de la adquisición de datos mediante sensores y de la visualización mediante una pantalla LCD, mientras que el dispositivo esclavo ejecuta acciones sobre actuadores tales como motores, LEDs y buzzers. Estos sistemas permiten comparar los protocolos de comunicación, evaluando características como velocidad, sincronización, cableado y respuesta del sistema ante distintos comandos.

Posteriormente, se desarrolla un sistema orientado al control de animaciones en una matriz LED RGB, donde cada imagen se representa como un conjunto de valores RGB almacenados en memoria. La selección de animaciones se realiza mediante comunicación UART, permitiendo alternar entre distintos patrones visuales sin interrumpir la actualización del display. Asimismo, la estructura por *frames* facilita el diseño de secuencias dinámicas y la visualización ordenada de cada cuadro de la animación.

Finalmente, se implementa un sistema interactivo de movimiento sobre la matriz RGB mediante un sensor MPU6050. A partir de las inclinaciones medidas en los ejes X e Y, un LED se desplaza en la matriz respetando los límites físicos del área visible. Este módulo incorpora además un botón para el cambio de color, integrando sensado inercial, control en tiempo real y retroalimentación visual. En conjunto, las prácticas permiten comprender la integración de sensores, actuadores, protocolos de comunicación y técnicas de control embebido en aplicaciones reales de laboratorio.

II. Objetivos

II-A. Objetivo General

- Desarrollar e implementar sistemas mecatrónicos basados en el microcontrolador ATmega328P que integren comunicación digital, procesamiento de señales, control de actuadores y visualización, aplicando los protocolos SPI, I^{2}C y UART en distintos contextos de laboratorio.

II-B. Objetivos Específicos

- Diseñar arquitecturas maestro–esclavo que permitan el intercambio de datos mediante SPI e I^{2}C, integrando sensores, pantalla LCD y actuadores controlados en tiempo real.
- Implementar un sistema de animaciones RGB mediante el uso de matrices LED, gestionando cuadros de imagen

(*frames*) y permitiendo la selección dinámica a través de comandos UART.
- Integrar un sensor MPU6050 para controlar el desplazamiento de un LED dentro de la matriz RGB en función de la inclinación, incorporando cambio de color mediante entrada digital.
- Comparar protocolos de comunicación digital en términos de estructura, cableado, velocidad y comportamiento en aplicaciones prácticas.

III. Materiales

- Microcontrolador ATmega328P.
- Matriz de LEDs RGB 8x8.
- Giroscopio y acelerómetro MPU 6050

IV. Marco Teórico

IV-A. Comunicación SPI

El protocolo SPI (Serial Peripheral Interface) es una interfaz síncrona full–duplex desarrollada originalmente por Motorola. Utiliza cuatro líneas principales (MOSI, MISO, SCK y SS) y permite una comunicación rápida y simple entre un maestro y uno o varios esclavos. Su ventaja radica en la baja latencia, ausencia de direccionamiento y capacidad de transmisión simultánea. (Motorola, 1999).

En el microcontrolador ATmega328P, el módulo SPI está implementado mediante los registros `SPCR`, `SPSR` y `SPDR`, que permiten configurar modo maestro/esclavo, velocidad y fase/polaridad del reloj. El hardware del AVR incorpora un registro de desplazamiento de 8 bits que garantiza la transmisión y recepción simultánea de datos en cada flanco del reloj. (Microchip, 2016).

IV-B. Comunicación I²C / TWI

El protocolo I²C fue desarrollado por Philips (hoy NXP) y permite la comunicación entre múltiples dispositivos utilizando solo dos líneas: SDA y SCL. Opera mediante un esquema maestro–esclavo, con direccionamiento de 7 o 10 bits, comprobación de ACK/NACK y generación explícita de condiciones START y STOP. (NXP, 2014).

El ATmega328P implementa este protocolo mediante el módulo TWI (Two-Wire Interface), controlado con los registros `TWBR`, `TWCR`, `TWDR` y `TWSR`. Este módulo soporta velocidades estándar como 100 kHz y 400 kHz, incluyendo la gestión automática del bit de ACK. (Microchip, 2016).

IV-C. Sensor DHT11

El DHT11 es un sensor digital de temperatura y humedad que utiliza un protocolo propietario de un solo hilo (single–wire) basado en pulsos de duración variable. La trama enviada por el sensor consta de 40 bits divididos en humedad entera, humedad decimal, temperatura entera, temperatura decimal y checksum. Las respuestas del sensor dependen estrictamente del tiempo, por lo cual se requiere un control preciso del microcontrolador. (Aosong Electronics, 2013).

IV-D. LDR y Potenciómetro

La LDR (Light Dependent Resistor) varía su resistencia según la cantidad de luz incidente, permitiendo obtener una medida proporcional mediante un divisor resistivo conectado al ADC del microcontrolador. A mayor iluminación, menor resistencia presenta la LDR, lo que incrementa la tensión leída. (Electronic Components Handbook, 2018).

El potenciómetro lineal se comporta como un divisor de tensión variable; la posición mecánica de su cursor define la fracción de voltaje aplicada a la entrada del ADC. Es uno de los métodos más simples y estables para obtener valores analógicos de referencia. (Horowitz & Hill, 2015).

IV-E. Actuadores: LED PWM, Servomotor y Buzzer

El control de brillo en un LED se realiza mediante PWM (Pulse Width Modulation), técnica en la cual se mantiene constante la frecuencia pero se varía el ciclo de trabajo para ajustar la potencia promedio entregada al diodo. En el ATmega328P esto se implementa mediante temporizadores internos como el Timer0 en modo Fast PWM. (Microchip, 2016).

Los servomotores estándar utilizan un pulso PWM con periodo fijo cercano a 20 ms y un ancho de pulso que varía típicamente entre 0.5 ms y 2.4 ms para representar ángulos entre 0° y 180°. En el ATmega328P, esto se genera utilizando el Timer1 en modo Fast PWM con tope definido por `ICR1`. (TowerPro, 2012).

El buzzer pasivo requiere una señal cuadrada cuya frecuencia determina el tono emitido. En el ATmega328P se genera mediante el Timer2 configurado en modo CTC, alternando el pin asociado en cada desbordamiento del comparador. (Microchip, 2016).

IV-F. Pantalla LCD por I²C mediante PCF8574

Las pantallas basadas en el controlador HD44780 operan en modo de 8 o 4 bits y requieren señales paralelas para su funcionamiento. Para simplificar las conexiones, se utiliza el expansor PCF8574, un dispositivo de NXP que convierte comandos recibidos por I²C en señales paralelas hacia el LCD. El protocolo I²C permite controlar la pantalla utilizando solo dos líneas, mientras que el PCF8574 gestiona las señales RS, RW, EN y D4–D7. (NXP, 2015). El controlador HD44780 define el set de instrucciones, tiempos de ejecución y estructura interna del display. (Hitachi, 1998).

IV-G. Concepto de Frame en Animaciones Digitales

En el ámbito de las imágenes y animaciones digitales, un *frame* se define como una imagen estática que forma parte de una secuencia. Cuando varios frames se muestran de manera consecutiva y a una velocidad adecuada, se genera la percepción de movimiento continuo. Este principio es fundamental en cine, videojuegos y sistemas de visualización embebidos (Gonzalez & Woods, 2018).

En matrices LED RGB, cada frame consiste en un conjunto de valores codificados en formato RGB, donde cada tripleta `(R,G,B)` define el color que exhibirá un LED determinado.

La reproducción secuencial de estos frames permite crear animaciones como desplazamientos, parpadeos o efectos visuales más elaborados. Debido a que los microcontroladores poseen recursos limitados, los frames suelen almacenarse como arreglos numéricos optimizados en memoria de programa (`PROGMEM`) para reducir el consumo de RAM (Yiu, 2019).

IV-H. LED Matrix Studio

LED Matrix Studio es un software diseñado para la creación visual de patrones y animaciones en matrices LED de distintos tamaños. Su interfaz permite definir el color de cada píxel mediante una paleta RGB y construir animaciones cuadro por cuadro (*frame-by-frame*). Una de sus principales ventajas es la capacidad de exportar automáticamente cada frame como un arreglo de bytes listo para ser insertado en programas escritos para microcontroladores, simplificando el flujo de desarrollo para sistemas embebidos (LED Matrix Studio, 2020).

El uso de este software permite reducir el tiempo de diseño y minimizar errores en el mapeo de LEDs, ya que cada cuadro es generado de forma visual en lugar de construirse manualmente en código. Esto resulta especialmente útil en matrices RGB basadas en protocolos estrictos como WS2812B, donde una única discrepancia en los valores RGB puede reflejarse en errores visibles en la animación final.

IV-I. Giroscopio y acelerómetro MPU 6050

EL MPU6050 es una unidad de medición inercial o IMU (Inertial Measurment Units) de 6 grados de libertad (DoF) pues combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes. Este sensor es muy utilizado en navegación, goniometría, estabilización, etc.(Tutorial MPU6050, Acelerómetro y Giroscopio, s. f.).

IV-J. Matriz de LEDs RGB 8x8

Este módulo es un panel cuadrado con una interfaz XH2.54 de 3 pines. Este módulo de matriz LED a todo color RGB de 8x8 se basa en LED de control inteligentes WS2812. Cada LED se puede direccionar de forma independiente con píxeles RGB que pueden alcanzar 256 niveles de brillo DFRobot (s. f.).

IV-K. Dead Zone

Zona muerta, también llamado banda muerta, se refiere a la rango o banda de valores de entrada dentro de la cual no hay cambios en la salida.(Lemau, 2025)

IV-L. Módulo Bluetooth HC-05

El HC-05 es un módulo de comunicación inalámbrica basado en el estándar Bluetooth 2.0, diseñado para establecer enlaces serie entre microcontroladores y dispositivos externos. Opera bajo el perfil *Serial Port Profile* (SPP), permitiendo la transmisión de datos mediante una interfaz UART convencional. Su simplicidad de uso y su bajo costo lo han convertido en un componente ampliamente utilizado en sistemas embebidos orientados a control remoto, telemetría y robot móvil (Bolun, 2016) .

El módulo cuenta con dos modos de operación: *modo comando*, utilizado para configurar parámetros como nombre, velocidad en baudios y rol maestro/esclavo; y *modo datos*, utilizado para la transmisión transparente de caracteres (Bolun, 2016) .

IV-M. Sensores de Línea Infrarrojos

Los sensores de línea utilizados están basados en un par emisor–receptor infrarrojo. El LED emisor proyecta luz IR sobre la superficie, mientras que el fototransistor receptor mide la cantidad de radiación reflejada. Debido a la diferencia de reflectancia entre superficies blancas y negras, el sensor presenta dos estados claramente distinguibles: nivel alto al detectar superficie clara, y nivel bajo al detectar superficie oscura o una línea negra (Sharp, 2006) .

V. Procedimiento

V-A. Problema A: Comunicación SPI maestro–esclavo

El objetivo de este apartado es implementar una arquitectura maestro–esclavo utilizando el bus SPI entre dos microcontroladores ATmega328P. El nodo maestro se encarga de leer los sensores (DHT11, LDR y potenciómetro), procesar las variables y enviar tramas de control de 4 bytes hacia el nodo esclavo. Este segundo microcontrolador recibe las tramas a través del módulo SPI en modo esclavo y actualiza los actuadores: LED con PWM, servomotor y buzzer pasivo.

V-A1. Asignación de pines en el maestro SPI: En el ATmega328P configurado como maestro, se utilizaron los pines por hardware del módulo SPI y los canales analógicos correspondientes a los sensores, además de la línea unihilo del DHT11 y el bus I2C para la LCD. La distribución se resume en el Cuadro I.

Cuadro I: Conexiones principales del maestro SPI (ATmega328P).

Señal / Componente	Pin ATmega328P
SPI SCK	PB5 (D13)
SPI MOSI	PB3 (D11)
SPI MISO	PB4 (D12)
SPI SS (selección esclavo)	PB2 (D10)
DHT11 (datos)	PD4 (D4)
LDR (salida analógica)	PC1 (A1)
Potenciómetro	PC0 (A0)
LCD I2C SDA (PCF8574)	PC4 (A4)
LCD I2C SCL (PCF8574)	PC5 (A5)
UART TX (debug)	PD1 (D1)

El maestro inicializa la UART a 9600 baudios para registro de telemetría, el módulo SPI en modo maestro con prescaler $f_{\text{osc}}/64$ y el ADC en referencia AVcc. Para la pantalla LCD se reutiliza la rutina de inicialización en modo 4 bits a través del expansor I2C.

V-A2. Asignación de pines en el esclavo SPI: En el ATmega328P configurado como esclavo, se reservaron los pines SPI por hardware y las salidas PWM necesarias para LED y servomotor, además del pin dedicado al buzzer. La Tabla II resume el conexionado.

Cuadro II: Conexiones principales del esclavo SPI (ATmega328P).

Señal / Componente	Pin ATmega328P
SPI MISO	PB4 (D12, salida)
SPI MOSI	PB3 (D11, entrada)
SPI SCK	PB5 (D13, entrada)
SPI SS	PB2 (D10, entrada)
LED PWM	PD6 (D6, OC0A)
Servo	PB1 (D9, OC1A)
Buzzer pasivo	PD3 (D3)
Pin debug actividad SPI	PD7 (D7)
UART TX (debug)	PD1 (D1)

El esclavo configura el módulo SPI en modo esclavo con interrupción habilitada (`SPIE`). Cada vez que se completa la recepción de un byte (interrupción `SPI_STC_vect`), el dato entrante se almacena en un buffer circular de 4 bytes. Cuando se completan los 4 bytes, se marca una bandera `frameReady` indicando que hay una trama lista para procesar.

V-A3. Definición de la trama de control SPI: La comunicación maestro–esclavo se basa en una trama de 4 bytes:

- Byte 0: *modo* (no utilizado en esta versión, fijado en 0).
- Byte 1: *buzzVal* $(0 \ldots 255)$: valor que se mapea a una frecuencia entre 200Hz y 3000Hz para el buzzer pasivo.
- Byte 2: *ledPWM* $(0 \ldots 255)$: ciclo de trabajo aplicado al LED mediante PWM.
- Byte 3: *servoAng* $(0 \ldots 180)$: ángulo del servomotor en grados.

En el maestro, el potenciómetro se utiliza para derivar simultáneamente *buzzVal* y *servoAng*, mientras que la LDR controla el valor de *ledPWM*. Además, el DHT11 se lee periódicamente y sus valores de temperatura y humedad se muestran en la pantalla LCD junto con el estado actual de los comandos.

V-A4. Simulación y montaje físico: Primero se realizó la simulación del sistema en Proteus, conectando dos ATmega328P mediante las líneas SPI y anexando los componentes virtuales equivalentes (LED, servo, buzzer). Se verificó que, al variar el potenciómetro y la iluminación sobre la LDR, el esclavo modificaba correctamente el brillo del LED, la posición del servo y la frecuencia del buzzer.

Una vez validada la lógica en simulación, se procedió al montaje físico en protoboard, respetando el mismo mapa de pines definido en los Cuadros I y II. Ambos microcontroladores comparten masa común y se alimentan a 5V. El enlace SPI se realizó mediante cables cortos para minimizar ruido, y se agregaron capacitores de desacople cercanos a cada ATmega328P.

V-B. Problema B: Comunicación I2C maestro–esclavo

En este apartado se replica la arquitectura maestro–esclavo del Problema A, pero reemplazando el bus SPI por un bus I2C entre dos ATmega328P. Nuevamente, el nodo maestro integra los sensores (DHT11, LDR y potenciómetro) y la pantalla LCD I2C, mientras que el nodo esclavo se encarga de los actuadores (LED PWM, servo y buzzer). La trama de control y el mapeo de variables se mantienen, lo que permite una comparación directa entre SPI e I2C.

V-B1. Configuración del maestro I2C: El maestro I2C utiliza el módulo TWI del ATmega328P configurado a 100 kHz mediante `TWBR` y `TWSR`. El mismo nodo que en el Problema A lee el DHT11 y las entradas analógicas, construye la trama de 4 bytes y la envía periódicamente al esclavo, ahora mediante una transacción I2C a la dirección de 7 bits $0x12$.

La distribución de pines en el maestro es similar, concentrando las líneas del bus en SDA y SCL:

Cuadro III: Conexiones principales del maestro I2C (ATmega328P).

Señal / Componente	Pin ATmega328P
I2C SDA (bus maestro–esclavo)	PC4 (A4)
I2C SCL (bus maestro–esclavo)	PC5 (A5)
DHT11 (datos)	PD4 (D4)
LDR (salida analógica)	PC1 (A1)
Potenciómetro	PC0 (A0)
LCD I2C (PCF8574, misma SDA/SCL)	Bus I2C compartido
UART TX (debug)	PD1 (D1)

El maestro genera, en cada ciclo, una condición de *START*, envía la dirección del esclavo con bit de escritura, y luego los 4 bytes de la trama (*modo*, *buzzVal*, *ledPWM*, *servoAng*), finalizando con una condición de *STOP*. En paralelo, actualiza la LCD con los valores de temperatura, humedad y los comandos actuales.

V-B2. Configuración del esclavo I2C: El ATmega328P configurado como esclavo I2C se asigna a la dirección de 7 bits $0x12$. Este nodo comparte el mismo esquema de actuadores utilizado en el Problema A: LED PWM sobre Timer0, servomotor en Timer1 y buzzer sobre Timer2. La lógica de interpretación de la trama se mantiene: el segundo byte se convierte en frecuencia de buzzer, el tercero se mapea al ciclo de trabajo del LED, y el cuarto se traduce al ancho de pulso del servo.

Al igual que en el caso SPI, la recepción de la trama se realiza en un buffer de 4 bytes y, una vez completada, se actualizan los actuadores. Se mantuvo la UART a 9600 baudios para depuración, registrando los valores recibidos y el estado de los actuadores a modo de telemetría.

V-B3. Simulación y montaje físico: La simulación en Proteus se llevó a cabo reutilizando la topología del Problema A, pero sustituyendo el enlace SPI por las líneas SDA y SCL con sus respectivas resistencias de pull-up. Se verificó que el maestro enviara tramas válidas a la dirección $0x12$ y que

el esclavo respondiera ajustando el LED, servo y buzzer de acuerdo a los cambios en el potenciómetro y la LDR.

Posteriormente, se montó el sistema en físico utilizando un único bus I2C compartido por el maestro, el esclavo y el módulo PCF8574 de la LCD. Esto permitió observar en la práctica una de las ventajas de I2C frente a SPI: la posibilidad de conectar múltiples dispositivos con solo dos líneas, a costa de una mayor complejidad del protocolo y de la necesidad de gestionar direcciones y colisiones.

V-C. Control de Animaciones en Matriz RGB mediante UART

El objetivo de este apartado es implementar un sistema capaz de reproducir animaciones en una matriz LED RGB utilizando el microcontrolador ATmega328P, gestionando la actualización de los cuadros de imagen (*frames*) y permitiendo la selección dinámica de contenido visual mediante comandos enviados por comunicación serial (UART). Además de las animaciones, el sistema incluye la inicialización de colores básicos, permitiendo encender toda la matriz en rojo, verde o azul mediante el comando UART correspondiente, lo que facilita la verificación inicial del funcionamiento y del mapeo de colores.

Para el diseño de las animaciones se empleó el software *LED Matrix Studio*, el cual permitió construir de manera visual cada cuadro de la animación y exportar automáticamente los valores RGB de cada LED. Los *frames* generados fueron integrados en el programa como arreglos estáticos, organizados en secuencias que representan las animaciones completas. Cada arreglo contiene los valores RGB de la matriz y puede ser recorrido en tiempo real por el microcontrolador.

El sistema de interacción se basa en la recepción de caracteres por UART. Al recibir el comando '1', el microcontrolador selecciona una animación inspirada en el videojuego *Pac-Man*, en la cual se representa al personaje desplazándose por la matriz y su posterior derrota. Cuando se recibe el comando '2', se activa una animación basada en el juego *Pong*, donde se simula el movimiento de la pelota y el marcador evoluciona en tres etapas: primero 1–0, luego 1–1 y finalmente 2–1. Asimismo, se implementó el comando 'C', encargado de ejecutar una rutina de inicialización cromática que enciende la matriz completa en los colores básicos (rojo, verde o azul), permitiendo comprobar visualmente la correcta transmisión de datos a los LEDs.

La reproducción de los *frames* se diseñó como un proceso no bloqueante, asegurando que la actualización visual no interfiera con la recepción serial. Para cada cuadro, el microcontrolador envía los valores RGB al bus de datos utilizando el protocolo WS2812, respetando los tiempos de señalización necesarios para evitar parpadeos o inconsistencias. La matriz se conectó físicamente al pin digital 8 (PB0) del ATmega328P, manteniendo la misma distribución de pines tanto en pruebas como en la implementación final.

La programación del sistema se organizó en tres etapas principales. En primer lugar, se generaron las animaciones en *LED Matrix Studio* y se integraron los *frames* exportados al código fuente, verificando su correspondencia con los diseños originales. En segundo lugar, se desarrolló el módulo de comunicación UART, que incluye las funciones de selección de animaciones, la inicialización de colores básicos y el comando para apagar la matriz. Finalmente, se llevó a cabo la integración con el controlador WS2812, ajustando la temporización y validando la estabilidad visual de las secuencias. Con esta estructura, el sistema permite alternar entre animaciones y colores de prueba de forma fluida, remota y en tiempo real.

La implementación física se realizó utilizando la misma distribución de pines definida durante las pruebas, conservando la conexión del bus de datos de la matriz LED RGB en el pin 8 del ATmega328P.

V-D. Control de movimiento en matriz LED RGB usando MPU6050 y ATmega328P

En este apartado, se implemento un control de movimiento en una matriz LED RGB utilizando el aceloremetro del MPU6050. El LED debe iniciar en una posición próxima al centro de la matriz, y dependiendo de la inclinación física que detecte el MPU6050 se va a desplazar sobre esta matriz, pero con la restricción que no debe sobrepasar los limites de esta. También se debe contar con un boton, el cual cumple la función de modificar el color del LED al ser presionado.

Una vez comprendidos los requerimientos del apartado, se procedió a diseñar la solución, comenzando por la implementación del control de la matriz 8x8. La matriz LED utiliza el protocolo WS2812B, que requiere una temporización especifica para la transmisión de datos. Cada LED requiere 24 bits de información en formato GRB. El microcontrolador envía estos datos bit a bit mediante:

- Un pulso alto de aproximadamente 0.8µs seguido de 0.45µs en bajo representa un bit '1'
- Un pulso alto de aproximadamente 0.4µs seguido de 0.85µs en bajo representa un bit '0'

,

Para lograr esta temporización precisa se utilizó instrucciones en ensamblador (nop) y se deshabilitaron las interrupciones durante la transmisión, también es necesario la gestion de un buffer de memoria que almacena el estado de cada LED en formato GRB

Posteriomente, se implementó el sensor MPU6050 para lograr el desplazamiento del LED sobre la matriz, el cual se inicializa con un rango de aceleración de $\pm 4g$. Este sensor usa una comunicación I2C a una velocidad de 100 kHz, el proceso de lectura se basa en que el microcontrolador envía la dirección del MPU6050 y el registro a leer, seguidamente se realiza un reinicio repetido, que es una condición especial del protocolo I2C que permite cambiar la dirección o el modo de operación (lectura/escritura) sin liberar el bus, despues se leen los 6 bytes correspondientes a los ejes X, Y, Z. Cada eje está representado por 2 bytes en formato complemento a dos, los valores se convierten de bytes individuales a enteros de 16 bits, estos valores representan un movimiento en la matriz de la siguiente forma:

- Eje X del sensor: Movimiento vertical (arriba/abajo), se utiliza el valor negativo de ax

- Eje Y del sensor: Movimiento horizontal (izquierda/derecha), se utiliza directamente ay

Se estableció un umbral de 1200 unidades para filtrar pequeñas vibraciones. Cuando la aceleración supera este umbral en cualquier dirección, el LED se mueve un píxel en esa dirección. El sistema verifica los limites de la matriz para evitar que el LED los sobrepase.

Por último, se implemento el botón para cmabiar de color, el cual fue configuro con una resistencia pull-up, por lo que su estado en reposo es alto (1). Al presionarlo, el pin lee un estado bajo (0). Los colores predefinidas con los que cuenta el sistema se observan a continuación en el Cuadro.IV.

Cuadro IV: Colores del LED en la matriz

Color	**Valor**
Rojo	0, 255, 0
Verde	255, 0, 0
Azul X	0, 0, 255
Amarillo Y	255, 255, 0
Celeste	255, 0, 255
Naranja	128, 255, 0

V-E. Robot de mini fútbol

El desarrollo del robot de mini fútbol comenzó con la elaboración de un croquis preliminar utilizando el software *Autodesk Fusion 360*. En esta etapa se definió la distribución espacial del chasis, la ubicación de las baterías, los motores, el módulo Bluetooth, los sensores de línea y el mecanismo delantero de metegol. Este croquis permitió anticipar colisiones mecánicas, optimizar la disposición del cableado y verificar que las dimensiones finales cumplieran con las restricciones establecidas por el concurso (25 cm × 25 cm × 20 cm).

Una vez validado el diseño conceptual, se procedió al montaje físico de todos los componentes sobre el chasis acrílico. Se fijaron los motores DC laterales para la tracción diferencial, el motor del metegol y los sensores infrarrojos encargados de detectar los bordes de la cancha. Posteriormente, se realizó el conexionado eléctrico de cada módulo al microcontrolador ATmega328P siguiendo la asignación presentada en el Cuadro V.

Cuadro V: Asignación de pines del robot de mini fútbol

Componente	Puerto AVR	Pin Arduino
Sensor de línea izquierdo	PD2	D2
Sensor de línea derecho	PD7	D7
Motor izquierdo IN1	PD4	D4
Motor izquierdo IN2	PB0	D8
Motor izquierdo PWM	PD6	D6 (OC0A)
Motor derecho IN1	PB1	D9
Motor derecho IN2	PB5	D13
Motor derecho PWM	PD5	D5 (OC0B)
Motor metegol IN1	PC0	A0
Motor metegol IN2	PC1	A1
Motor metegol PWM	PB4	D12
HC-05 RX	PB2	D10
HC-05 TX	PB3	D11
UART Debug RX0	PD0	D0
UART Debug TX0	PD1	D1

Tras establecer la distribución de pines, se conectaron los motores de tracción al puente H principal, utilizando las salidas PWM del temporizador 0 del ATmega328P para controlar la velocidad. El motor delantero del metegol se conectó a un segundo puente H independiente, lo que permitió accionar el mecanismo de golpeo sin interferir con el sistema de movilidad.

Los sensores infrarrojos se instalaron en la parte frontal inferior del robot y se conectaron a las entradas digitales D2 y D7, configuradas con resistencias internas *pull-up*. Esto permitió implementar una lógica de seguridad: si el sensor izquierdo detectaba la línea negra, se bloqueaban los movimientos hacia la izquierda; si el derecho detectaba la línea, se bloqueaban los movimientos hacia la derecha; y si ambos sensores detectaban borde, solo se permitía el retroceso.

Para el control inalámbrico, se conectó el módulo Bluetooth HC-05 mediante *software serial* en los pines D10 y D11. Los comandos enviados desde la aplicación móvil se interpretaron en el microcontrolador a través de la función `processCommand()`, la cual ejecutaba acciones como avanzar, retroceder, girar, realizar diagonales o accionar el metegol con dos niveles de intensidad.

Finalmente, se definió un esquema de alimentación dividido: seis pilas de 1.5 V en serie se utilizaron para la etapa de potencia (motores y puentes H), mientras que una batería de 9 V alimentó exclusivamente la placa Arduino. Se unificó la masa en ambos sistemas para garantizar estabilidad eléctrica y evitar ruido en la comunicación serial. Con la programación final cargada, se realizaron pruebas de funcionamiento que validaron la movilidad, la detección segura de bordes y el correcto accionamiento del sistema de metegol.

VI. Resultados

VI-A. Problema A: Comunicación SPI (Simulado y físico)

En la etapa de simulación en PROTEUS se validó el flujo completo maestro–esclavo por SPI. El maestro leyó en tiempo real el DHT11, el potenciómetro y la LDR, generando a partir de ellos los comandos para el esclavo: valor de frecuencia para el buzzer, ciclo de trabajo PWM para el LED y ángulo

objetivo para el servomotor. Estos valores se empaquetaron en una trama de 4 bytes y se enviaron de forma periódica al esclavo. En la simulación se verificó que, ante cambios en el potenciómetro o en la iluminación de la LDR, el LED variaba su brillo, el servo modificaba su posición y el buzzer cambiaba de tono.

En la Fig. 1 se muestra el montaje simulado del sistema maestro–esclavo SPI en PROTEUS, donde se aprecia el ATmega328P maestro con el DHT11, la LDR, el potenciómetro y la LCD I2C, y el ATmega328P esclavo con el LED PWM, el servo y el buzzer conectados a sus respectivos timers.

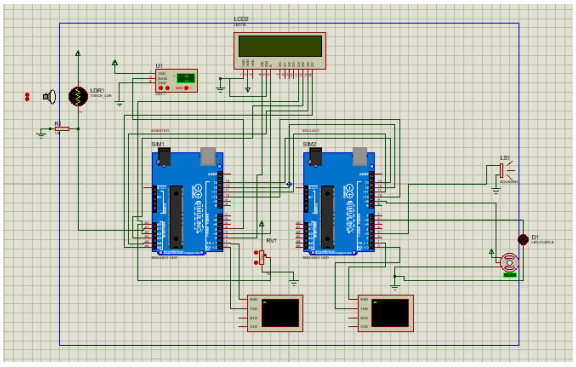

Fig. 1: Montaje simulado del sistema maestro–esclavo SPI en PROTEUS.

Una vez validada la lógica en simulación, se procedió al montaje físico en protoboard. Se respetó el mismo mapeo de pines: en el maestro se conectó el DHT11 al pin digital configurado para la lectura temporal, el potenciómetro y la LDR a las entradas analógicas A0 y A1, y la pantalla LCD mediante el módulo PCF8574 sobre el bus I^2C. En el esclavo se conectó el LED a la salida PWM de Timer0, el servomotor al Timer1 (OC1A) y el buzzer al pin manejado por Timer2.

Durante las pruebas en físico se observó que al modificar el potenciómetro variaba el ángulo del servo y el tono del buzzer, mientras que al tapar o iluminar la LDR cambiaba el brillo del LED. La LCD en el maestro mostraba en tiempo real los valores de temperatura, humedad y las variables de control enviadas al esclavo. En la Fig. 2 se presenta el montaje físico completo del sistema SPI.

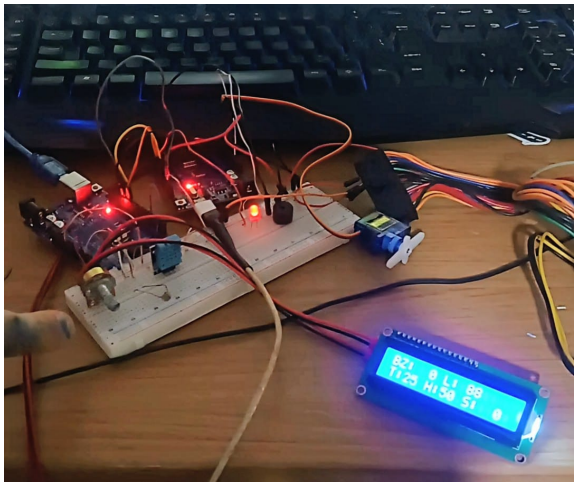

Fig. 2: Montaje físico del sistema maestro–esclavo SPI con dos ATmega328P.

VI-A0a. Explicación del firmware (fragmentos relevantes).: A continuación se documentan brevemente los bloques clave del código para el Problema A.

(1) Envío de la trama SPI en el maestro: El maestro empaqueta `modo`, `buzzVal`, `ledPWM` y `servoAng` en una trama de 4 bytes y la envía bajando el `SS` y transmitiendo los bytes de forma secuencial.

Listing 1: Envío de trama de 4 bytes desde el maestro SPI.

```
static void spi_send4(uint8_t b0,uint8_t b1,uint8_t
    b2,uint8_t b3){
  PORTB &= ~(1<<PB2);         // SS bajo ->
    selecciona esclavo
  spi_txrx(b0);
  spi_txrx(b1);
  spi_txrx(b2);
  spi_txrx(b3);
  PORTB |=  (1<<PB2);         // SS alto -> fin de
    trama
}
```

(2) Recepción de trama en el esclavo (ISR SPI): En el esclavo, cada interrupción `SPI_STC_vect` almacena el byte recibido en un buffer de 4 posiciones. Cuando se completa la trama se levanta la bandera `frameReady`.

Listing 2: Recepción de bytes SPI en el esclavo.

```
volatile uint8_t rxBuf[4];
volatile uint8_t rxIdx=0;
volatile uint8_t frameReady=0;

ISR(SPI_STC_vect){
  uint8_t b = SPDR;
  rxBuf[rxIdx++] = b;
  if(rxIdx>=4){
    rxIdx=0;
    frameReady=1;             // Hay trama completa
  }
  PIND = (1<<PIN_DBG);        // Toggle PD7 para debug
}
```

(3) Actualización de actuadores en el esclavo: Cuando `frameReady` está en 1, el bucle principal copia los bytes de la trama, actualiza servo, LED y buzzer, y vuelve al estado de espera.

Listing 3: Aplicación de la trama SPI a los actuadores.

```
if(frameReady){
  cli();
  uint8_t modo = rxBuf[0];
  uint8_t buzz = rxBuf[1];
  uint8_t lPWM = rxBuf[2];
  uint8_t ang  = rxBuf[3];
  frameReady = 0;
  sei();

  // Buzzer: 0 -> silencio, otro valor -> mapeo a Hz
  if(buzz == 0){
    buzzer_off();
  }else{
    uint16_t hz = 200 + ((uint32_t)buzz*2800UL)/255
    UL;
    buzzer_set_freq(hz);
  }

  led_pwm(lPWM);           // Brillo LED
  servo_write_angle(ang); // ngulo servo
}
```

(4) Mapeo de ángulo a pulso de servo: El esclavo convierte un ángulo en grados en un ancho de pulso en microsegundos dentro del rango permitido por el servomotor y actualiza `OCR1A`.

Listing 4: Conversión de ángulo a pulso de servo.

```
static void servo_write_angle(uint8_t ang){
  if(ang>180) ang=180;
  uint16_t us = SERVO_MIN_US +
                (uint32_t)(SERVO_MAX_US -
    SERVO_MIN_US)*ang/180UL;
  if(us < 400)  us = 400;
  if(us > 2600) us = 2600;
  OCR1A = us * 2;   // Tick de 0.5 us
}
```

VI-B. Problema B: Comunicación I^2C (Simulado y físico)

Para el caso de I^2C se reutilizó la misma lógica funcional del maestro: lectura del DHT11, potenciómetro y LDR, presentación de los valores en la LCD y generación de comandos para actuadores. La principal diferencia fue el reemplazo del enlace SPI por un bus I^2C entre ambos ATmega328P, asignando al esclavo una dirección de 7 bits (por ejemplo, 0x12). El maestro envía tramas de 4 bytes mediante la función de escritura I^2C, y el esclavo las interpreta para actualizar el buzzer, el LED PWM y el servomotor.

En la Fig. 3 se muestra el circuito simulado en PROTEUS para el sistema maestro–esclavo por I^2C. Se observa el bus SDA/SCL compartido entre el ATmega maestro, el módulo PCF8574 de la LCD y el ATmega esclavo, respetando las resistencias de pull-up necesarias para el correcto funcionamiento del bus.

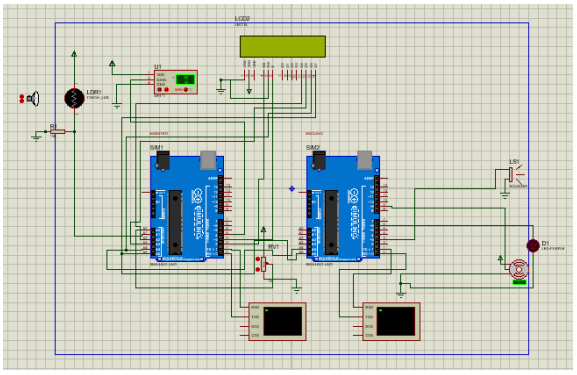

Fig. 3: Montaje simulado del sistema maestro–esclavo por I^2C en PROTEUS.

En el montaje físico se replicó la misma topología: el maestro integra los sensores (DHT11, potenciómetro y LDR) y la pantalla LCD I2C, mientras que el esclavo concentra los actuadores (LED PWM, servomotor y buzzer). Ambos comparten SDA y SCL con resistencias de pull-up a 5V y una masa común.

Durante las pruebas se comprobó que el sistema se comporta de forma equivalente al caso SPI: al variar el potenciómetro se actualiza el ángulo del servo y el tono del buzzer, y al modificar la iluminación sobre la LDR cambia la intensidad luminosa del LED. La LCD muestra los mismos valores de referencia, con la diferencia de que ahora el enlace maestro–esclavo se realiza completamente sobre I^2C. La Fig. 4 presenta el montaje físico de la solución por I^2C.

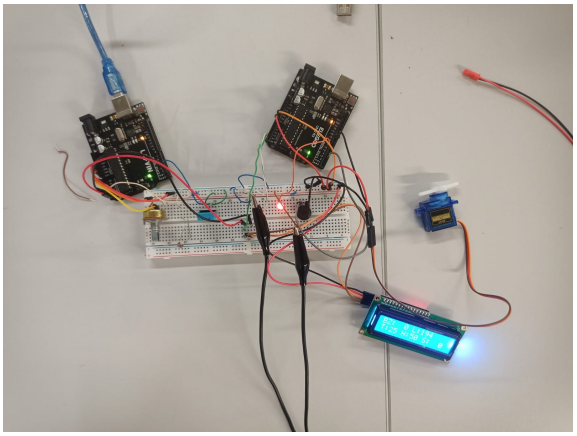

Fig. 4: Montaje físico del sistema maestro–esclavo I^2C con dos ATmega328P.

VI-B0a. Explicación del firmware (fragmentos relevantes).: En este apartado se resumen los bloques más relevantes del código para la solución I^2C.

(1) Envío de la trama I^2C en el maestro: El maestro utiliza la función `i2c_write_frame` para enviar los 4 bytes de control a la dirección 0x12 del esclavo.

Listing 5: Envío de trama I2C desde el maestro.

```
static void i2c_write_frame(uint8_t addr7,
                            uint8_t b0,uint8_t b1,
```

```
                              uint8_t b2,uint8_t b3){
  if(!i2c_start((addr7<<1)|0)) return;  // SLA+W
  i2c_write(b0);
  i2c_write(b1);
  i2c_write(b2);
  i2c_write(b3);
  i2c_stop();
}
```

(2) Construcción de comandos a partir de sensores: En cada iteración, el maestro lee potenciómetro y LDR, actualiza el DHT11 y construye los comandos `buzzVal`, `ledPWM` y `servoAng` antes de enviarlos al esclavo.

Listing 6: Lectura de sensores y construcción de la trama.

```
uint16_t pot = adc_read(0);     // A0
uint16_t ldr = adc_read(1);     // A1

if(++tick>=10){
  tick=0;
  uint8_t t,h;
  if(dht11_read(&t,&h)){ T=t; H=h; }
}

uint8_t buzzVal  = map_u16_to_u8(pot, 0,1023);
uint8_t ledPWM   = map_u16_to_u8(ldr, 0,1023);
uint8_t servoAng = (uint8_t)((uint32_t)pot*180UL/102
    3UL);

i2c_write_frame(0x12, 0, buzzVal, ledPWM, servoAng);
```

(3) Actualización de actuadores en el esclavo I^2C: En el nodo esclavo, la lógica de actualización de actuadores es análoga a la del caso SPI: se interpreta la trama recibida y se ajustan buzzer, LED y servo. El fragmento siguiente ilustra el patrón general de aplicación de los comandos.

Listing 7: Aplicación de comandos en el esclavo I2C.

```
void aplicar_trama(uint8_t modo,
                   uint8_t buzz,
                   uint8_t lPWM,
                   uint8_t ang){
  if(buzz == 0){
    buzzer_off();
  }else{
    uint16_t hz = 200 + ((uint32_t)buzz*2800UL)/255
    UL;
    buzzer_set_freq(hz);
  }
  led_pwm(lPWM);
  servo_write_angle(ang);
}
```

(4) Interfaz de monitoreo en LCD: Tanto en SPI como en I^2C, el maestro utiliza la LCD para mostrar las variables medidas y los comandos enviados, lo que facilitó la depuración en banco.

Listing 8: Actualización de la LCD con variables clave.

```
lcd_goto_xy(0,0);
lcd_printf("BZ:%3u L:%3u   ", buzzVal, ledPWM);
lcd_goto_xy(0,1);
lcd_printf("T:%2u H:%2u S:%3u", T, H, servoAng);
```

VI-C. Control de Animaciones en Matriz RGB mediante UART

En la implementación física, la matriz LED RGB 8×8 se conectó directamente al microcontrolador ATmega328P, utilizando el pin digital 8 (PB0) como línea de datos para el bus WS2812. La alimentación de la matriz se realizó a 5 V desde la misma fuente que el microcontrolador, asegurando una correcta conexión de masa común entre todos los dispositivos. En la Figura **??** se presenta el montaje físico del sistema sobre protoboard, donde se observa la matriz conectada al ATmega328P y al resto del circuito de pruebas.

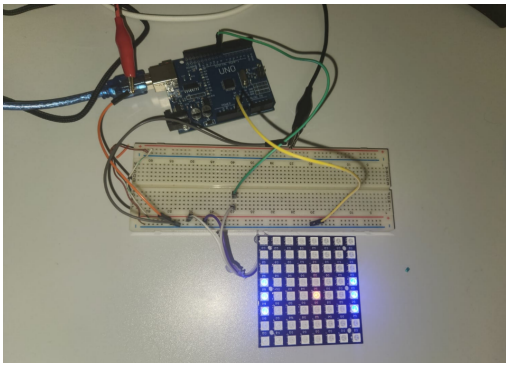

Fig. 5: Circuito en Físico

Durante las pruebas en banco, se verificó que el envío de comandos por UART desde el equipo host (mediante un terminal serie) permitió seleccionar de forma confiable tanto los colores básicos (rojo, verde y azul) como las animaciones predefinidas. El comando `'C'` activó la rutina de inicialización cromática, encendiendo la matriz completa en colores sólidos, lo que permitió comprobar visualmente la correcta correspondencia de los canales RGB y el mapeo de cada LED en la matriz física.

En cuanto a las animaciones, el comando `'1'` activó la animación basada en *Pac-Man*. En la primera parte de la secuencia se observó claramente a Pac-Man representado como una figura amarilla que alterna entre un estado con la "boca" abierta y otro con la "boca" cerrada, simulando el efecto de estar comiendo píxeles a medida que avanza sobre el tablero. En los cuadros siguientes aparece un fantasma de color distinto, que se aproxima hasta superponerse con Pac-Man. A partir de ese instante se ejecuta la animación de la muerte de Pac-Man, donde el personaje se "desarma" progresivamente en varios *frames*, hasta desaparecer por completo de la matriz.

Por su parte, el comando `'2'` habilitó la animación inspirada en el juego *Pong*. En esta secuencia se visualizan las paletas y la pelota desplazándose por la matriz, junto con la representación del marcador del partido. La animación recorre tres estados de puntuación bien diferenciados: primero 1–0, luego 1–1 y finalmente 2–1, dando la sensación de un partido en progreso. En todos los casos, la transición entre cuadros (*frames*) fue fluida, sin parpadeos apreciables, confirmando que la temporización del protocolo WS2812 fue correctamente respetada.

El Listado 9 muestra el bloque principal de definición de constantes, estructuras y variables globales utilizado para gestionar la matriz LED RGB, la comunicación UART y la selección de animaciones. En particular, se configuran

la frecuencia de reloj del microcontrolador (F_CPU) y la velocidad de la UART (BAUD), se definen las macros asociadas al pin de datos del bus WS2812 y se declara la estructura RGB, que almacena los valores de color de cada LED en formato (g, r, b), acorde al orden requerido por los módulos WS2812. El arreglo leds[] actúa como búfer en RAM para la construcción de cada cuadro antes de ser enviado en serie a la matriz.

Además, se definen las variables globales animacion_actual y comando_nuevo como volatile, dado que son modificadas dentro de las rutinas de interrupción de la UART. Finalmente, se declaran los arreglos pacman[] y pong[] en memoria de programa (PROGMEM), donde se almacenan los datos de color de cada *frame* de las animaciones. En el código completo, estos arreglos contienen NUM_FRAMES_ANIM1 y NUM_FRAMES_ANIM2 cuadros respectivamente, cada uno compuesto por NUM_LEDS*3 bytes (canales G, R y B por LED).

```
#define F_CPU 16000000UL
#define BAUD  9600
#define MYUBRR ((F_CPU/16/BAUD)-1)

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

#define WS2812_PORT PORTB
#define WS2812_DDR  DDRB
#define WS2812_PIN  PB0

#define NUM_LEDS           64
#define NUM_FRAMES_ANIM1   20
#define NUM_FRAMES_ANIM2   47

typedef struct {
    uint8_t g;
    uint8_t r;
    uint8_t b;
} RGB;

RGB leds[NUM_LEDS];

volatile uint8_t animacion_actual = 0;
volatile uint8_t comando_nuevo    = 0;

/* Declaracion de los arreglos de animaciones
    en memoria de programa */
const uint8_t PROGMEM pacman[NUM_FRAMES_ANIM1
    ][NUM_LEDS * 3];
const uint8_t PROGMEM pong  [NUM_FRAMES_ANIM2
    ][NUM_LEDS * 3];
```

Listing 9: Definiciones principales para el control de la matriz RGB y las animaciones

En el núcleo del programa, la lógica de actualización de la matriz recorre continuamente el búfer de LEDs y, en función del comando recibido por UART, selecciona qué animación reproducir o qué color sólido aplicar. El Listado 10 muestra un fragmento representativo de esta lógica, donde se procesa el último comando recibido y se avanza frame a frame dentro de la animación correspondiente, copiando los datos desde PROGMEM hacia el arreglo leds[] antes de enviarlos por el pin PB0 utilizando el protocolo WS2812.

```
void seleccionar_animacion(uint8_t comando)
{
    switch (comando) {
        case '1':  // Animaci n Pac-Man
            animacion_actual = 1;
            break;
        case '2':  // Animaci n Pong
            animacion_actual = 2;
            break;
        case 'C':  // Colores s lidos de
           prueba (rojo, verde, azul)
            animacion_actual = 3;
            break;
        default:   // Apagar matriz
            animacion_actual = 0;
            break;
    }
}

void reproducir_animaciones(void)
{
    static uint8_t frame = 0;

    if (animacion_actual == 1) {
        // Copiar frame de Pac-Man desde
           PROGMEM a leds[]
        memcpy_P(leds,
                 pacman[frame],
                 sizeof(leds));
        frame = (frame + 1) % NUM_FRAMES_ANIM1
            ;
    }
    else if (animacion_actual == 2) {
        // Copiar frame de Pong desde PROGMEM
           a leds[]
        memcpy_P(leds,
                 pong[frame],
                 sizeof(leds));
        frame = (frame + 1) % NUM_FRAMES_ANIM2
            ;
    }
    else if (animacion_actual == 3) {
        // Colores s lidos (rojo, verde o
           azul) para prueba
        // (la selecci n del color depende
           del comando 'C'
        // y de la l gica adicional
           implementada en el c digo
           completo)
    } else {
        // Apagar matriz
    }

    // Enviar el contenido de leds[] a la
       matriz por WS2812
    enviar_matriz_ws2812(leds, NUM_LEDS);
}
```

Listing 10: Lógica básica de selección de animación y actualización de la matriz

En conjunto, estos bloques de código, junto con los arreglos completos pacman[] y pong[] (omitidos en los listados

por su extensión), permitieron implementar un sistema capaz de reproducir animaciones clásicas de videojuegos y patrones de prueba de color en una matriz LED RGB 8×8, controladas en tiempo real por comandos UART y verificadas tanto en simulación como en el montaje físico.

VI-D. Control de movimiento en matriz LED RGB usando MPU6050 y ATmega328

VI-D1. Montaje: Se procedió a montar en físico el sistema de control de movimiento en una matriz led. A continuación, en el Cuadro.VI se observa la distribución de pines, y en la Fig.6 el circuito montado.

Cuadro VI: Distribución de pines

Componente	Pin ATMEGA328P
Matriz de LEDs	PB0 (PD8)
Botón	PD2
MPU6050 SDA	PC4 (A4)
MPU6050 SCL	PC5 (A5)

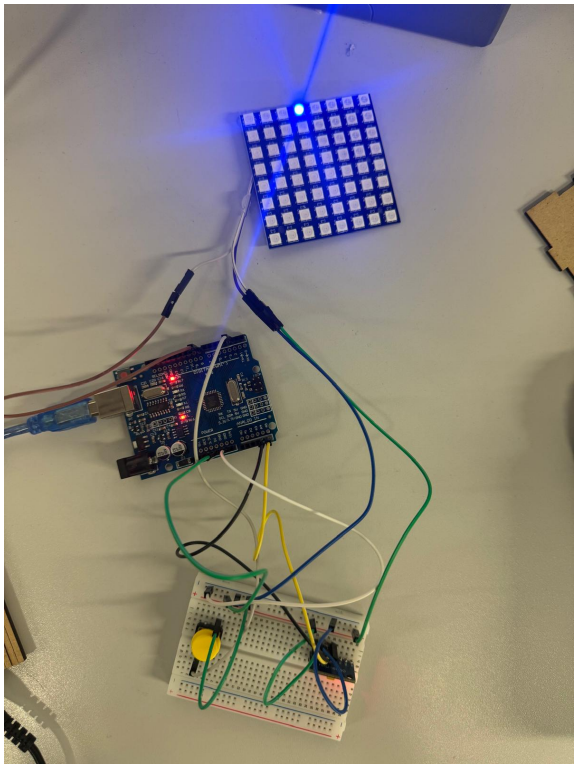

Fig. 6: Ensamblaje en físico control de LED en matriz

VI-D2. Código: A continuación, se mostraran algunos bloques de código necesarios para abarcar correctamente la implementación de la solución:

VI-D3. Comunicación con la Matriz: Se implementó una función de transmisión de bits utilizando instrucciones de ensamblador (nop) para generar retardos exactos de nanosegundos. Esto es importante para cumplir con los tiempos del protocolo de los LEDs, asi los colores se renderizan bien y no hay problemas de compilación.

Listing 11: Comunicación con la Matriz

```
void matriz_enviar_byte(uint8_t byte) {
    cli();  // Deshabilita interrupciones para
    proteger el timing

    for (uint8_t bit = 0; bit < 8; bit++) {
        if (byte & 0x80) {
            // Envo de '1' lgico: pulso alto largo
            PUERTO_MATRIZ |= (1 << PIN_MATRIZ);
            __asm__ __volatile__("nop\n\t" "nop\n\t"
     ... ); // Delays precisos
            PUERTO_MATRIZ &= ~(1 << PIN_MATRIZ);
            // ...
        } else {
            // Envo de '0' lgico: pulso alto corto
            PUERTO_MATRIZ |= (1 << PIN_MATRIZ);
            __asm__ __volatile__("nop\n\t" "nop\n\t"
     ... );
            PUERTO_MATRIZ &= ~(1 << PIN_MATRIZ);
            // ...
        }
        byte <<= 1;
    }
    sei();  // Rehabilita interrupciones
}
```

VI-D4. Datos del Acelerometro: La comunicación con el sensor MPU6050 se realiza mediante el bus I2C. Se destacan la lectura en ráfaga de los registros de aceleración y la operación de desplazamiento de bits necesaria para combinar los bytes altos y bajos, de esa forma se consigue el valor real de 16 bits con signo para cada eje.

Listing 12: Datos del Acelerometro

```
uint8_t mpu6050_leer_aceleracion(int16_t *ax, int16
    _t *ay, int16_t *az) {
    // Inicio de comunicacin I2C

    // Lectura secuencial de 6 bytes (Ejes X, Y, Z -
     Parte Alta y Baja)
    datos[0] = i2c_leer_ack(); // X High
    datos[1] = i2c_leer_ack(); // X Low
    // (lectura de otros ejes)
    datos[5] = i2c_leer_nack(); // ltimo byte sin
    ACK

    // Reconstruccin de variables de 16 bits
    mediante desplazamiento
    *ax = (int16_t)((datos[0] << 8) | datos[1]);
    *ay = (int16_t)((datos[2] << 8) | datos[3]);
    *az = (int16_t)((datos[4] << 8) | datos[5]);

    return 1;
}
}
```

VI-D5. Logica del Sistema: Para traducir la inclinación del sensor al movimiento del LED, se implementó un UMBRAL_MOVIMIENTO. Esto actúa como una "zona muerta"que filtra el ruido natural del acelerómetro y vibraciones pequeñas, asegurando que el LED solo se mueva cuando la inclinación sea intencional.

Listing 13: Logica del Sistema

```
void actualizar_posicion_led(int16_t ax, int16_t ay)
     {
    defineUMBRAL_MOVIMIENTO 1200
```

```
    // Mapeo de ejes del sensor a la orientacin de
    la matriz
    int16_t eje_x = ay;
    int16_t eje_y = -ax;

    // Zona muerta para evitar movimiento por ruido
    if (eje_x > UMBRAL_MOVIMIENTO) {
        nueva_x++;
    } else if (eje_x < -UMBRAL_MOVIMIENTO) {
        nueva_x--;
    }

}
```

VI-E. Robot de mini fútbol

El desarrollo del robot inició con la construcción de un modelo digital mediante el software *Autodesk Fusion 360*. Este croquis permitió definir la distribución interna del chasis, la posición de los motores, sensores de línea, puentes H, baterías y el módulo Bluetooth. Gracias a este modelo, fue posible verificar interferencias mecánicas, alturas de montaje y rutas probables de cableado antes de realizar el ensamblaje físico.

En la Figura 7 se presenta la vista explotada del diseño CAD, donde pueden apreciarse individualmente los módulos electrónicos y mecánicos. Las Figuras 8 y 9 muestran el modelo ensamblado en vistas lateral y superior, respectivamente, evidenciando la distribución final de los componentes y el aprovechamiento del área útil del chasis.

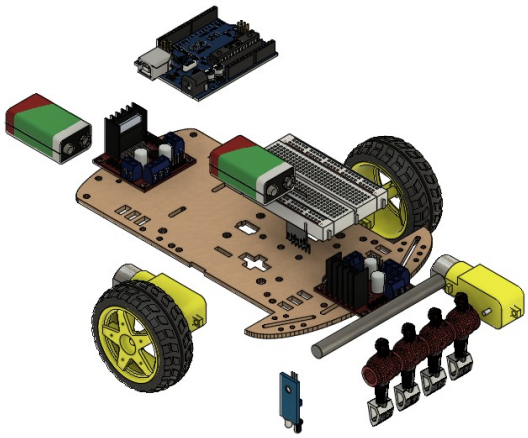

Fig. 7: Vista Explotada del Auto en Fusion 360

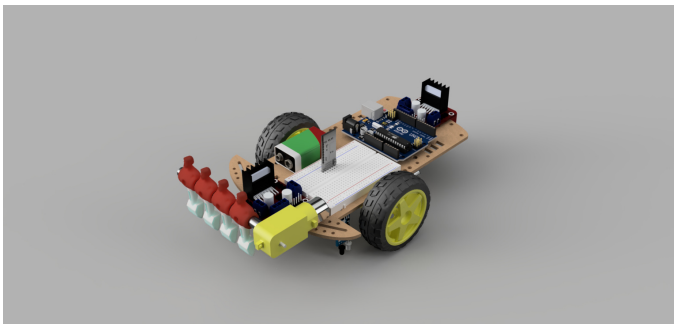

Fig. 8: Vista Laterial del Robot en Fusion 360

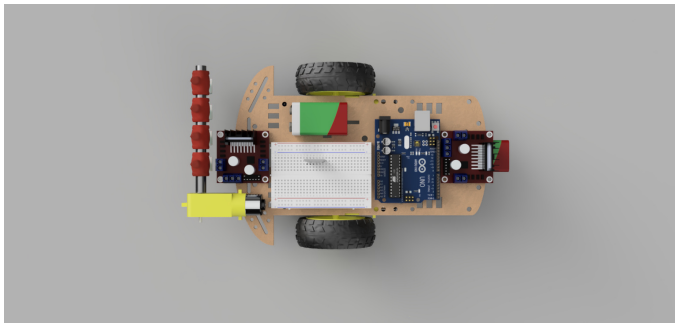

Fig. 9: Vista Superiror del Robot en Fusion 360

Una vez validado el diseño digital, se procedió al armado físico del robot utilizando exactamente los mismos componentes previstos en el modelo. La única diferencia respecto al diseño CAD fue la etapa de potencia: aunque el modelo incluye una batería de 9 V para motores, en la implementación final se utilizaron seis pilas AA en serie (9 V equivalentes) para obtener mayor corriente disponible y estabilidad durante los arranques de los motores DC. El Arduino continuó siendo alimentado por una batería de 9 V independiente, manteniendo la referencia de masa común entre ambos sistemas.

Las siguientes figuras muestran el montaje final en físico en cuatro vistas: frontal, superior, lateral y posterior. Estas imágenes permiten evaluar la correspondencia entre el diseño CAD y el ensamblaje real, así como la correcta ubicación de sensores, puentes H, módulo Bluetooth y sistema de metegol.

Fig. 10: Vista frontal del robot ensamblado en físico.

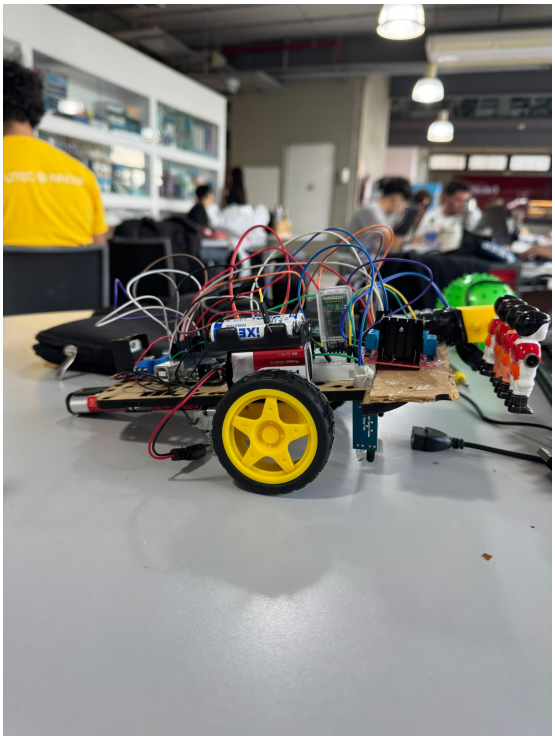

Fig. 12: Vista lateral (costado) del prototipo físico.

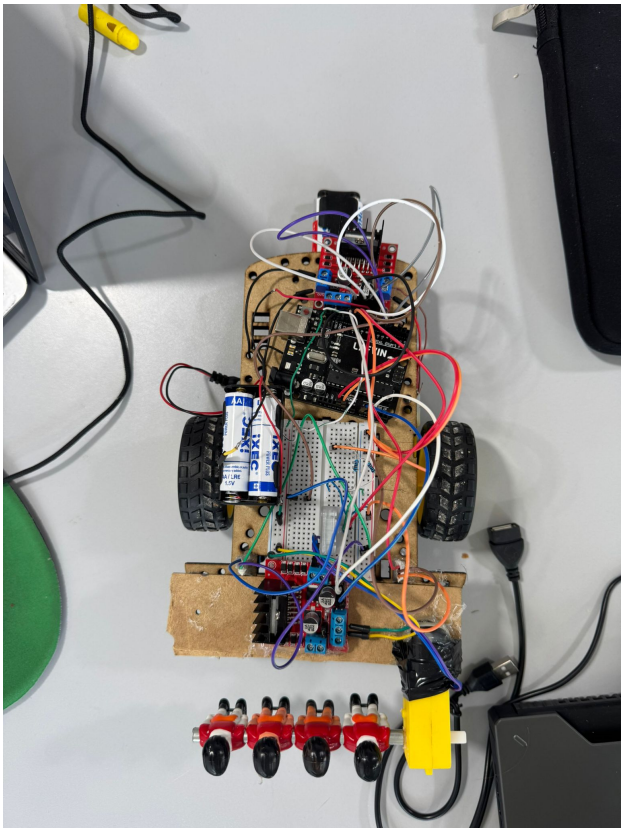

Fig. 11: Vista superior del robot físicamente ensamblado.

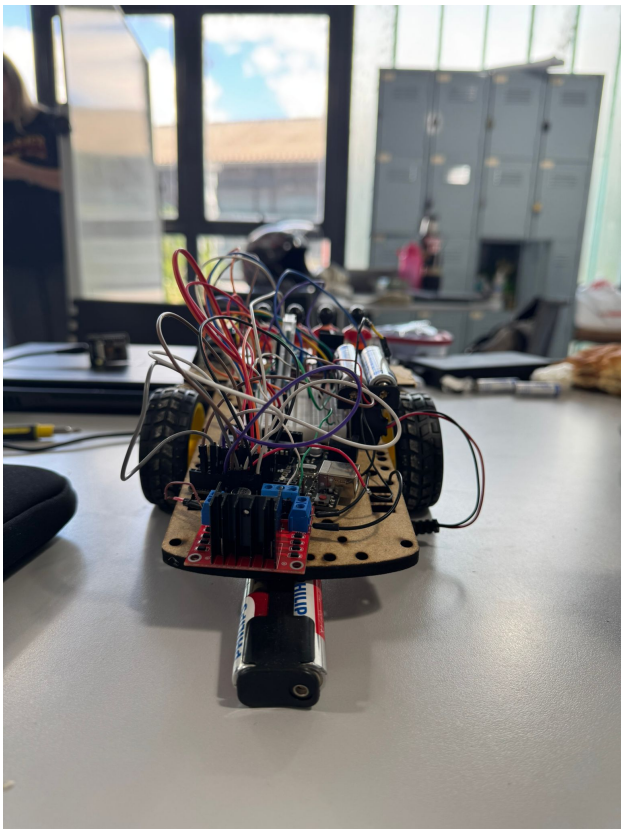

Fig. 13: Vista trasera mostrando la etapa de potencia y distribución de baterías.

VI-E1. Código y funcionamiento: A continuación se presenta el código final utilizado para controlar el robot, eliminando comentarios para mejorar la legibilidad dentro del informe. El programa integra control de motores mediante PWM, lectura de sensores de línea, comunicación Bluetooth mediante *software serial*, control del motor del metegol y un sistema de bloqueo que impide que el robot salga de la cancha.

```
#ifndef F_CPU
#define F_CPU 16000000UL
#endif

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define SENSOR_IZQ PD2
#define SENSOR_DER PD7

#define MOTOR_IZQ_IN1 PD4
#define MOTOR_IZQ_IN2 PB0
#define MOTOR_IZQ_PWM PD6

#define MOTOR_DER_IN1 PB1
#define MOTOR_DER_IN2 PB5
#define MOTOR_DER_PWM PD5

#define METEGOL_IN1 PC0
#define METEGOL_IN2 PC1
#define METEGOL_PWM PB4

#define HC05_RX_PIN PB2
#define HC05_TX_PIN PB3

#define DEBUG_BAUD 9600
#define DEBUG_UBRR ((F_CPU/16/DEBUG_BAUD)-1)

#define HC05_BAUD 38400
#define HC05_BIT_DELAY 26

#define VEL_AVANCE 200
#define VEL_GIRO 150

#define LEER_IZQ() ((PIND & (1<<SENSOR_IZQ))>>
    SENSOR_IZQ)
#define LEER_DER() ((PIND & (1<<SENSOR_DER))>>
    SENSOR_DER)

char receivedCommand = 0;
uint8_t velocidad_actual = VEL_AVANCE;

void Debug_Init(void){
    UBRR0H = (unsigned char)(DEBUG_UBRR >> 8);
    UBRR0L = (unsigned char)DEBUG_UBRR;
    UCSR0B = (1 << TXEN0);
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}

void Debug_Print(unsigned char data){
    while (!(UCSR0A & (1 << UDRE0)));
    UDR0 = data;
}

void Debug_PrintString(const char* str){
    while (*str) Debug_Print(*str++);
}

void Debug_PrintHex(uint8_t num){
    const char hexDigits[] = "0123456789ABCDEF";
    Debug_Print(hexDigits[(num >> 4) & 0x0F]);
    Debug_Print(hexDigits[num & 0x0F]);
}

void HC05_Init(void){
    DDRB |= (1 << HC05_TX_PIN);
    PORTB |= (1 << HC05_TX_PIN);
    DDRB &= ~(1 << HC05_RX_PIN);
    PORTB |= (1 << HC05_RX_PIN);
}

void HC05_Write(unsigned char data){
    cli();
    PORTB &= ~(1 << HC05_TX_PIN);
    _delay_us(HC05_BIT_DELAY);
    for (uint8_t i = 0; i < 8; i++){
        if (data & 0x01) PORTB |= (1 << HC05_TX_PIN)
            ;
        else PORTB &= ~(1 << HC05_TX_PIN);
        data >>= 1;
        _delay_us(HC05_BIT_DELAY);
    }
    PORTB |= (1 << HC05_TX_PIN);
    _delay_us(HC05_BIT_DELAY);
    sei();
}

void HC05_WriteString(const char* str){
    while (*str) HC05_Write(*str++);
}

uint8_t HC05_Read(char* data){
    uint16_t timeout = 0;
    while ((PINB & (1 << HC05_RX_PIN)) && timeout <
        50000) timeout++;
    if (timeout >= 50000) return 0;
    _delay_us(HC05_BIT_DELAY / 2);
    if (PINB & (1 << HC05_RX_PIN)) return 0;
    uint8_t byte = 0;
    for (uint8_t i = 0; i < 8; i++){
        _delay_us(HC05_BIT_DELAY);
        byte >>= 1;
        if (PINB & (1 << HC05_RX_PIN)) byte |= 0x80;
    }
    _delay_us(HC05_BIT_DELAY);
    if (!(PINB & (1 << HC05_RX_PIN))) return 0;
    *data = byte;
    return 1;
}

void init_pwm(void){
    DDRD |= (1 << MOTOR_IZQ_PWM);
    DDRD |= (1 << MOTOR_DER_PWM);
    TCCR0A = (1<<WGM00)|(1<<WGM01)|(1<<COM0A1)|(1<<
        COM0B1);
    TCCR0B = (1<<CS01);
    OCR0A = 0;
    OCR0B = 0;
}

void set_motor_izq(int16_t velocidad){
    if (velocidad > 0){
        PORTD |= (1 << MOTOR_IZQ_IN1);
        PORTB &= ~(1 << MOTOR_IZQ_IN2);
        OCR0A = (velocidad > 255) ? 255 : velocidad;
    } else if (velocidad < 0){
        PORTD &= ~(1 << MOTOR_IZQ_IN1);
        PORTB |= (1 << MOTOR_IZQ_IN2);
        OCR0A = (-velocidad > 255) ? 255 : -
            velocidad;
    } else {
        PORTD |= (1 << MOTOR_IZQ_IN1);
        PORTB |= (1 << MOTOR_IZQ_IN2);
        OCR0A = 255;
    }
}
```

```
void set_motor_der(int16_t velocidad){
    if (velocidad > 0){
        PORTB |= (1 << MOTOR_DER_IN1);
        PORTB &= ~(1 << MOTOR_DER_IN2);
        OCR0B = (velocidad > 255) ? 255 : velocidad;
    } else if (velocidad < 0){
        PORTB &= ~(1 << MOTOR_DER_IN1);
        PORTB |= (1 << MOTOR_DER_IN2);
        OCR0B = (-velocidad > 255) ? 255 : -
            velocidad;
    } else {
        PORTB |= (1 << MOTOR_DER_IN1);
        PORTB |= (1 << MOTOR_DER_IN2);
        OCR0B = 255;
    }
}

void detener(void){
    set_motor_izq(0);
    set_motor_der(0);
}

void avanzar(void){
    set_motor_izq(velocidad_actual);
    set_motor_der(velocidad_actual);
}

void retroceder(void){
    set_motor_izq(-velocidad_actual);
    set_motor_der(-velocidad_actual);
}

void girar_izquierda(void){
    set_motor_izq(-VEL_GIRO);
    set_motor_der(VEL_GIRO);
}

void girar_derecha(void){
    set_motor_izq(VEL_GIRO);
    set_motor_der(-VEL_GIRO);
}

uint8_t puede_moverse(char comando){
    uint8_t izq = LEER_IZQ();
    uint8_t der = LEER_DER();

    if (!izq && !der) return 1;
    if (izq && der){
        if (comando=='B' || comando=='S') return 1;
        return 0;
    }
    if (izq && !der){
        if (comando=='L' || comando=='Q' || comando
            =='Z') return 0;
        return 1;
    }
    if (!izq && der){
        if (comando=='R' || comando=='E' || comando
            =='C') return 0;
        return 1;
    }
    return 1;
}

void processCommand(char cmd){
    if (!puede_moverse(cmd)){
        detener();
        return;
    }
    switch(cmd){
        case 'F': avanzar(); break;
        case 'B': retroceder(); break;
        case 'L': girar_izquierda(); break;
        case 'R': girar_derecha(); break;
        case 'S': detener(); break;
    }
}

void GPIO_Init(void){
    DDRD |= (1<<MOTOR_IZQ_IN1);
    DDRB |= (1<<MOTOR_IZQ_IN2);
    DDRB |= (1<<MOTOR_DER_IN1)|(1<<MOTOR_DER_IN2);
    DDRC |= (1<<METEGOL_IN1)|(1<<METEGOL_IN2);
    DDRB |= (1<<METEGOL_PWM);
    DDRD &= ~((1<<SENSOR_IZQ)|(1<<SENSOR_DER));
    PORTD |= (1<<SENSOR_IZQ)|(1<<SENSOR_DER);
}

int main(void){
    GPIO_Init();
    Debug_Init();
    HC05_Init();
    init_pwm();
    _delay_ms(1000);
    while (1){
        if (HC05_Read(&receivedCommand)){
            processCommand(receivedCommand);
            _delay_ms(10);
        }
    }
}
```

VI-E2. Análisis del funcionamiento del código: El código se estructura en cinco pilares fundamentales:

- **1. Comunicación Bluetooth por software serial.** El módulo HC-05 se controla mediante una implementación manual del protocolo UART, utilizando temporización precisa con `_delay_us()` para generar los bits de inicio, datos y parada. Esto permite liberar el UART hardware para depuración.
- **2. Control de motores con PWM.** El temporizador 0 se configura en modo Fast PWM, utilizando los pines OC0A y OC0B para controlar la velocidad de los motores izquierdo y derecho. Las funciones `set_motor_izq()` y `set_motor_der()` permiten movimiento hacia adelante, atrás y frenado.
- **3. Sistema de seguridad por detección de líneas.** La función `puede_moverse()` lee los sensores de línea y bloquea movimientos peligrosos. Si el robot detecta borde en el lado izquierdo, se bloquean comandos como `'L'`, `'Q'` y `'Z'`. Si detecta borde en ambos sensores, sólo se habilita el retroceso.
- **4. Motor de metegol con PWM por software.** El motor frontal se controla mediante modulación por software, activando y desactivando el pin `METEGOL_PWM` con temporización en microsegundos. Esto permite dos modos de golpeo: suave y fuerte.
- **5. Procesamiento de comandos Bluetooth.** La función `processCommand()` interpreta letras como:
  - `'F'`: avanzar
  - `'B'`: retroceder
  - `'L'`: girar izquierda
  - `'R'`: girar derecha
  - `'S'`: stop

  Antes de ejecutar cualquier comando, se valida que el movimiento sea seguro mediante `puede_moverse()`.

En conjunto, estos módulos permiten que el robot se desplace de forma controlada, evite salirse de la cancha, responda a comandos remotos y ejecute golpes de metegol con dos niveles de potencia.

VII. Conclusiones

En relación al Problema A (SPI maestro–esclavo), se concluye que la implementación sobre el ATmega328P permitió integrar de manera efectiva el sensado de variables (temperatura, humedad, luz y posición de potenciómetro) con el control distribuido de actuadores (LED PWM, servomotor y buzzer). El bus SPI proporcionó una comunicación estable y sencilla de configurar, con una trama compacta de 4 bytes que resultó suficiente para transmitir los comandos de control necesarios. La validación en simulación y en montaje físico mostró un comportamiento consistente, sin pérdidas de datos ni desbordes de buffer, evidenciando la robustez del esquema maestro–esclavo con interrupciones en el esclavo.

Respecto al Problema B (I2C maestro–esclavo), la migración de la misma arquitectura al bus I2C demostró la ventaja de trabajar con una topología de dos hilos compartidos por múltiples dispositivos. La posibilidad de conectar en el mismo bus tanto la LCD (mediante PCF8574) como el microcontrolador esclavo facilitó el cableado y redujo el número de pines necesarios en el maestro. No obstante, se constató que la configuración correcta de direcciones y la gestión de *ACK*/*NACK* son aspectos críticos para el funcionamiento del sistema, haciendo al protocolo ligeramente más complejo de depurar que SPI.

En términos de comportamiento de los actuadores, ambos esquemas entregaron resultados equivalentes: el LED respondió adecuadamente al control de brillo, el servomotor siguió el ángulo derivado del potenciómetro y el buzzer generó un rango audible de frecuencias acorde al valor de comando. La pantalla LCD I2C en el maestro proporcionó una interfaz simple pero efectiva para monitorear las variables de sensado y el estado de los comandos enviados.

En cuanto al sistema de animaciones en la matriz LED RGB, se concluye que la integración entre el ATmega328P, el controlador WS2812B y el módulo de comunicación UART permitió desarrollar un sistema de visualización dinámico, estable y coherente con los objetivos planteados. La utilización de *LED Matrix Studio* para la generación de los cuadros de imagen (*frames*) facilitó la creación de animaciones complejas, mientras que su almacenamiento en memoria `PROGMEM` garantizó un uso eficiente de los recursos del microcontrolador.

Respecto al control serial, el sistema respondió de manera inmediata a los comandos enviados por UART, permitiendo alternar entre animaciones sin interrupciones visibles. La animación de *Pac-Man*, con su secuencia caracterizada por la apertura y cierre de la boca y la posterior aparición del fantasma que desencadena la animación de muerte, se reprodujo correctamente en todos los ensayos. De igual forma, la animación inspirada en *Pong* mostró un comportamiento estable, representando adecuadamente el movimiento de la pelota y la evolución del marcador en las etapas 1–0, 1–1 y 2–1.

En términos de precisión visual, la temporización del protocolo WS2812 se mantuvo dentro de los márgenes requeridos, evitando fallos de sincronización o parpadeos entre cuadros. Asimismo, la inicialización de colores sólidos mediante el comando `C` permitió comprobar la correcta correspondencia RGB y verificar el mapeo físico de cada LED antes de ejecutar las animaciones.

Finalmente, se comprobó que la implementación no bloqueante del sistema garantizó una reproducción continua de los *frames* sin interferir con la recepción serial, reafirmando la robustez del diseño y su adecuada adaptación a los requerimientos funcionales del laboratorio.

En cuanto al sistema de control de movimiento en la matriz LED, se concluye que la integración del sensor MPU6050 con el ATmega328P cumple con los requerimientos de interacción física visual. Se logró implementar de forma efectiva la comunicación I2C para la adquisición de datos del acelerómetro en ráfaga, también se aseguró la temporización precisa del protocolo WS2812B mediante instrucciones en ensamblador, garantizando que los colores se rendericen bien sin errores de transmisión.

Adicionalmente, el sistema gestiona la interacción con el usuario mediante un botón, permitiendo cambiar el color del LED entre los colores predefinidos.

Respecto a la validación del comportamiento físico, la lógica de control implementada demostró robustez frente al ruido inherente del sensor. La aplicación de una zona muerta, se filtró eficazmente las vibraciones menores, permitiendo un desplazamiento fluido del LED solo ante inclinaciones intencionales. Finalmente, se verificó el correcto mapeo de los ejes y la restricción de límites, asegurando que el LED este todo el rato dentro de los limites de la matriz.

En cuanto al sistema de control del robot de mini fútbol, se concluye que la integración del microcontrolador ATmega328P con el módulo Bluetooth HC-05, los motores de tracción y el mecanismo de metegol permitió desarrollar un prototipo plenamente funcional dentro de las condiciones del concurso. El control por PWM demostró ser estable y suficientemente responsivo para ejecutar maniobras de avance, retroceso y giros con precisión aceptable, incluso bajo variaciones de carga producidas durante el contacto con la pelota.

La lógica de comunicación inalámbrica implementada mediante *software serial* permitió una interacción fluida entre el controlador remoto y el robot, garantizando que los comandos de movimiento y los modos de golpeo fueran recibidos y ejecutados sin interrupciones. Asimismo, el motor de metegol operó correctamente en sus dos niveles de intensidad, ofreciendo una capacidad realista de disparo acorde con la dinámica del juego.

Respecto al sistema de seguridad basado en sensores de línea, si bien la lectura individual de cada sensor fue adecuada, el comportamiento compuesto no logró reproducir el caso teórico en el que ambos sensores detectan borde simultáneamente. En la práctica, uno de los sensores se activa siempre antes que

el otro por diferencias mínimas en la geometría, el ángulo de iluminación o la distancia al suelo. Esto provoca que el robot ejecute un movimiento semicontrolado donde una rueda queda detenida y la otra activa, impidiendo que el sistema alcance el estado de detección doble. En consecuencia, la lógica de "retroceso forzado" nunca llega a ejecutarse plenamente, lo que representa una limitación funcional del enfoque utilizado.

A pesar de ello, el sistema resultó operativo para evitar la salida del robot de la cancha en la mayoría de las situaciones reales, demostrando robustez general en su comportamiento dinámico. La implementación final cumplió con los requisitos del proyecto, integrando mecánica, electrónica y control en un único prototipo capaz de responder de manera coherente a las condiciones del partido.

VIII. Anexos

1. Link al Repositorio de GITHUB: https://github.com/juanferreiram-cell/Tec.Microprocesamiento
2. Link a la Carpeta de Drive con los videos: https://drive.google.com/drive/folders/1dfyBWJg07bqZ1Sa87PI9gOdNfCK04ClB?usp=sharing

IX. Bibliografía

1. motorola$_{s}piMotorolaInc.$, "$SPIBlockGuide$," $MotorolaSemiconductorApplicationNote$, 1999.
2. microchip$_{a}tmega328pMicrochipTechnologyInc.$, *ATmega328/P 8-bit AVR Microcontroller Datasheet*, 2016.
3. nxp$_{i}2cNXPSemiconductors$, "$I^2C - BusSpecificationandUserManual$," Rev,6, 2014.
4. aosong$_{d}ht11AosongElectronicsCo.Ltd.$, *DHT11 Humidity and Temperature Sensor Datasheet*, 2013.
5. electronic$_{h}andbookR.K.Sharma$, *Electronic Components Handbook*, $4thed.$, $McGraw-Hill$, 2018.
6. horowitz$_{h}illP.HorowitzandW.Hill$, *The Art of Electronics*, $3rded.$, $CambridgeUniversityPress$, 2015.
7. towerpro$_{s}ervoTowerPro$, *SG90 Micro Servo Technical Datasheet*, 2012.
8. nxp$_{p}cf8574NXPSemiconductors$, *PCF8574 Remote 8-bit I/O Expander for I²C Bus*, $DatasheetRev$,7, 2015.
9. hitachi$_{h}d44780HitachiLtd.$, *HD44780U LCD Controller/Driver Datasheet*, 1998.
10. Bolun, Z. (2016). *HC-05 Bluetooth Module Datasheet*. Shenzhen HC Information Technology Co. Recuperado de https://cdn.sparkfun.com/datasheets/Wireless/Bluetooth/HC-05_datasheet.pdf
11. Sharp Corporation. (2006). *Infrared Reflective Sensor Applications and Principles*. Technical Documentation. Recuperado de https://www.sharpsma.com/products/optoelectronics/application-notes
12. Tutorial MPU6050, acelerómetro y giroscopio. (s. f.). Naylamp Mechatronics - Perú. https://naylampmechatronics.com/blog/45_tutorial-mpu6050-acelerometro-y-giroscopio.html
13. Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th ed.). Pearson.
14. LED Matrix Studio. (2020). *LED Matrix Studio Software* [Computer software]. https://apps.microsoft.com
15. dfrobot.com. (s. f.). 8x8 RGB LED Matrix [Vídeo]. dfrobot.com. https://www.dfrobot.com/product-1555.html