

”Introducción al kit Arduino MKR wifi 1010”

Jostin Agustin Luis

Abstract—

Este trabajo presenta la implementación y pruebas del kit Arduino MKR WiFi 1010 junto con el Carrier MKR IoT Rev2 y sensores externos, a través de cinco prácticas de diferente alcance. En la primera, se desarrolla un juego interactivo con LEDs que cambian de color y estado, demostrando la facilidad de programación y control de salidas digitales. La segunda práctica aborda la monitorización ambiental, integrando mediciones de temperatura, humedad, presión, gases y giroscopio. En la tercera, se exploran las capacidades de detección de gestos y colores, así como la medición de proximidad, destacando el potencial de los sensores incorporados. La cuarta práctica profundiza en la conectividad inalámbrica, estableciendo conexiones Wi-Fi para escanear y seleccionar redes, además de un servicio BLE para lectura y escritura de datos. Finalmente, la quinta práctica emplea sensores externos para medir la humedad del suelo y detectar movimiento con un PIR, demostrando la versatilidad del kit al integrar dispositivos adicionales.

En conjunto, estas prácticas validan lo bueno que es el kit para aplicaciones educativas, investigación y de Internet de las cosas, al simplificar la conexión de sensores, la programación y el despliegue de datos en entornos reales.

Index Terms—

Ambiental, Arduino MKR WiFi 1010, BLE, Carrier MKR IoT Rev2, datos, detección, educativo, evaluación, interactivo, IoT, monitorización, prácticas.

I. INTRODUCCIÓN.

El presente documento describe el uso y las características del kit Arduino MKR WiFi 1010 a través de cinco prácticas que abarcan desde la gestión de LEDs en modo juego hasta la monitorización de variables ambientales, la detección de gestos y colores, la conexión inalámbrica por Wi-Fi y BLE, así como la medición de humedad del suelo y la detección de movimiento mediante un sensor PIR.

El propósito principal de estas prácticas es familiarizarse con la programación y la integración de componentes en un entorno de desarrollo para proyectos IoT y aplicaciones interactivas. Cada sección detalla la configuración inicial, las partes fundamentales del código y los resultados obtenidos. Aunque se emplea Arduino IDE, puede usarse cualquier otra herramienta de preferencia, además, este documento incluye información extraída de la página oficial de Arduino, con el fin de profundizar en el conocimiento de los componentes del kit y facilitar su uso.

Finalmente, se remarca que el objetivo de esta guía es explorar el kit y ofrecer una experiencia, mostrando el potencial de sus características y funcionalidades.o ayudaren la manipulación de mismo a nuevos usuarios.

II. METODOLOGÍA.

A. Partes del kit.

El kit cuenta principalmente con el Arduino wifi 1010 el cual es el cerebro de todo es el encargado de recibir el código y ejecutarlo, como siguiente parte tenemos al Carrier mkr iot rev2 que con sus sensores y componentes integrados es bastante completo, también tenemos un sensor de humedad del suelo y un sensor de detección de movimiento.

1) Arduino MKR WIFI 1010:

Placa de desarrollo que integra Wi-Fi y Bluetooth, permitiendo la creación de dispositivos inteligentes con conectividad inalámbrica. Su microcontrolador SAMD21 Cortex-M0+ de 32 bits trabaja a 48 MHz, con 256 KB de memoria Flash y 32 KB de SRAM, operando a 3.3V y compatible con baterías Li-Po de 3.7V. Cuenta con 7 entradas analógicas 8 salidas digitales algunas con PWM, y soporta protocolos de comunicación como I2C, SPI y UART (Fig. 1). También incluye un botón de reset, un LED integrado y un indicador de carga para monitoreo de batería. Es ideal para proyectos IoT facilitando la creación de sistemas conectados y controlables a distancia [1].

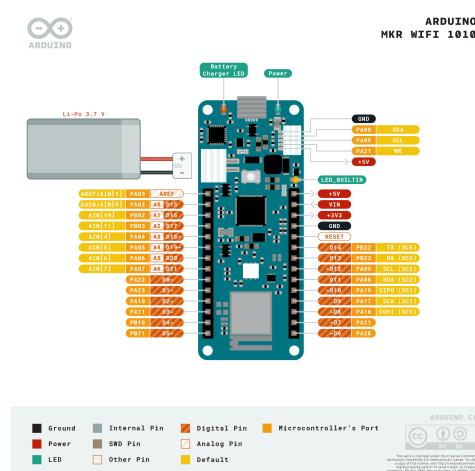


Fig. 1. Pinout Arduino MKR Wifi 1010 [1].

2) Carrier MKR IoT REV2:

Módulo complementario diseñado para potenciar proyectos basados en placas Arduino MKR (Fig. 2). Este Carrier facilita la creación de prototipos IoT al proporcionar herramientas para monitoreo ambiental, control de dispositivos y visualización de datos lo que lo vuelve perfecto para proyectos interactivos y educativos, a continuación profundizaremos en el [1].

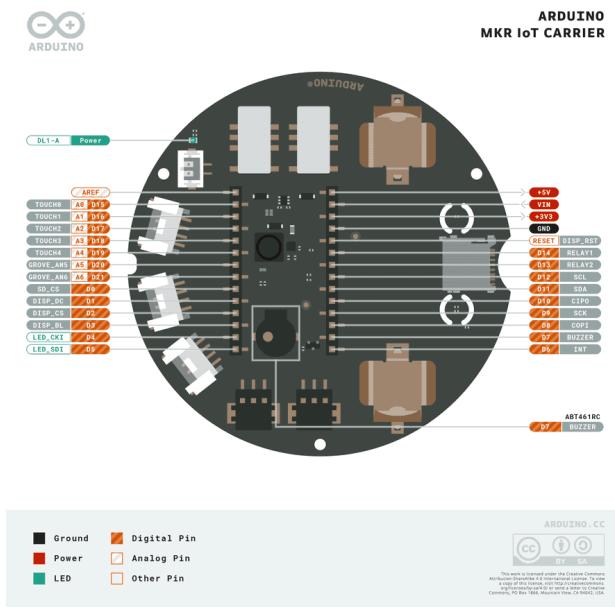


Fig. 2. Pinout Carrier mkr iot rev2 [1].

- Conectores Grove: Cuenta con 3 conectores, dos de ellos son analógicos se identifican con A0 y A6 y uno es para conexión I2C que se identifica como I2C, los cuales sirven para conectar módulos o sensores que complementaran a la placa dependiendo del enfoque (Fig. 3).

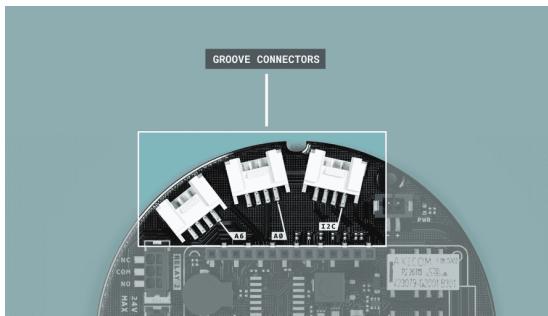


Fig. 3. Puertos Grove [1].

- Sensor BME688: Este sensor permite medir 4 variaciones como lo son:
 - Humedad de 0 - 100 porciento r.H.
 - Temperatura de -40 a +85 °C.
 - Presión de 300 - 1100 hPa.

d) Gas como IAQ, VOC, CO2.

El sensor tiene un consumo de energía muy bajo de apenas 2.1 TMA a 1 Hz y su forma de comunicación es I2C (Fig. 4).



Fig. 4. Ubicación del sensor Bme688 [1].

- LSM6DSOX: Es una unidad de medición inercial IMU el cual cuenta con un acelerómetro y un giroscopio.
 - Acelerómetro: Mide la aceleración lineal en tres ejes (X, Y y Z). Permite detectar el movimiento, la inclinación y la orientación del dispositivo en relación con la gravedad.
 - Rango de medición: ±2g, ±4g, ±8g, ±16g.
 - Resolución: 16 bits.
 - Frecuencia de muestreo: Hasta 6.66 kHz.
 - Ruido de aceleración: 90 µg/Hz.
 - Consumo de energía: 0.55 mA en modo activo.
 - Giroscópico: Mide la velocidad angular en tres ejes (X, Y y Z), permitiendo detectar la rotación del dispositivo y calcular su orientación en el espacio.
 - Rango de medición: ±125, ±250, ±500, ±1000, ±2000 dps.
 - Resolución: 16 bits.
 - Frecuencia de muestreo: Hasta 6.66 kHz.
 - Ruido angular: 3.8 mdps/Hz.
 - Consumo de energía: 0.85 mA en modo activo.
 - RGB y sensor de gestos: El MKR IoT Carrier Rev2 contiene un Broadcom sensores APDS-9960 RGB y Gesture, situado debajo de la pantalla y marcado con un icono de bombilla. El sensor es útil para luz ambiental y RGB detección de color, proximidad detección, y gesto detección (Fig. 5).
 - Relés: Equipado con dos Relés KEMET EE2-5NUL de 5V. Los relés pueden activarse con COM (común), NO (Normalmente abierto) y NC (normalmente cerrado) contactos, y puede ocupar un máximo de 2A y 24 V de entrada cada uno. Las conexiones entre un circuito de alta potencia y los relés se realizarán a través de estos conectores



Fig. 5. Ubicacion del sensor APDS-9960 RGB y Gesture [1].

High power pins. Una vez que los cables se introducen dentro de los conectores, se bloquearán automáticamente en el interior. Para desbloquear un cable y retirarlo del conector, se necesita insertar una herramienta a través del orificio cuadrado superior (Fig. 6).

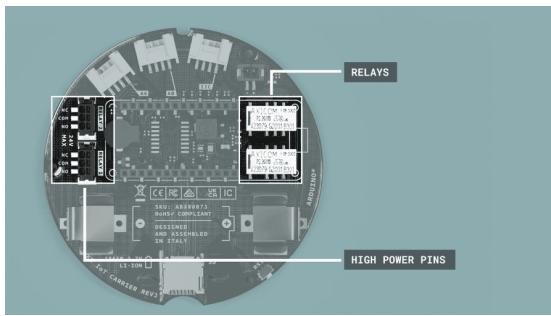


Fig. 6. Relés y puertos [1].

Por otro lado, tenemos los leds L1 y L2 que son los indicadores visuales del estado de los relés. Si los leds están encendidos significa que el COM y el NO el terminal del relé están conectados, y si los leds están apagados significa que COM y NC están conectados (Fig. 7).



Fig. 7. Leds de los Relés [1].

- Pantalla: La pantalla en el MKR IoT Carrier Rev2 es un pantalla TFT redondeada de 1.3", con una resolución de 240 x 240 y un diámetro de 36 x 40 mm (Fig. 8).

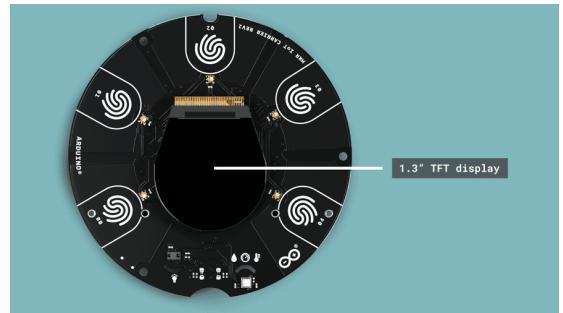


Fig. 8. Pantalla del Carrier [1].

- Botones: El Carrier tiene cinco botones táctiles capacitivos en su lado superior, numerado del 0 al 4. Los botones son sensibles al tacto directo y también pueden detectar el tacto inalámbrico es decir por proximidad (Fig.9).



Fig. 9. Botones del Carrier [1].

- Leds: El Carrier tambien cuenta con 5 LEDS RGB digitales colocado en el lado superior del portador delante de los botones (Fig. 10).



Fig. 10. Leds en el Carrier [1].

- Buzzer: El Carrier está equipado con un buzzer de sonido en el lado inferior del portador el cual funcionara para emitir sonidos programables ya sean zumbidos o melodias (Fig. 11).
- Energía: El MKR IoT Carrier Rev2 se puede alimentar a través de un cable USB conectado a la placa MKR o mediante una batería. La batería utilizada debe ser una batería LI-ION 18650 3.7v.

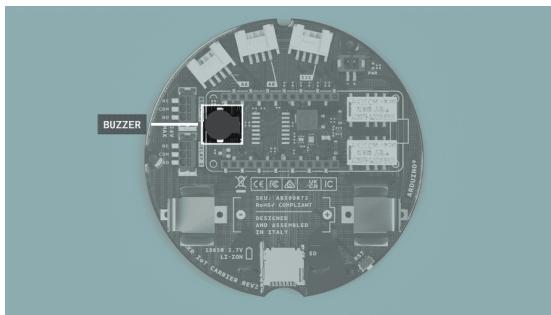


Fig. 11. Buzzer en el Carrier [1].

Para utilizar la alimentación USB para cargar la batería, se necesita un pequeño cable con conectores JST en ambos extremos entre el MKR IoT Carrier Rev2 y la placa MKR. La batería se puede recargar a través de una conexión USB a través de la placa MKR (Fig. 12).

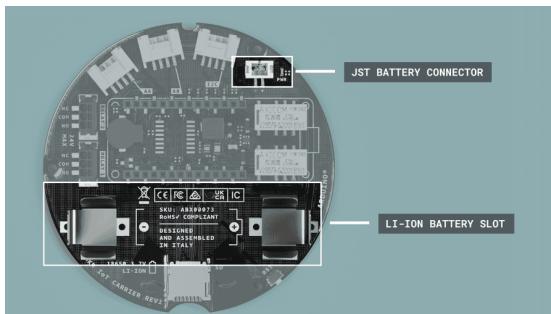


Fig. 12. Formas de alimentación [1].

3) Sensor de humedad de suelo:

Dispositivo diseñado para medir el nivel de humedad en la tierra de una forma sencilla y sin que no sufre corrosión. Es un dispositivo que utiliza principios de capacitancia para detectar la cantidad de agua en el suelo, evitando el desgaste por oxidación, su salida analógica permite obtener lecturas en tiempo real, lo que lo hace ideal para sistemas de control automatizados o para temas didácticos o investigación [1]. (Fig. 13).

- Voltaje de operación: 3.3V - 5V.
- Corriente de operación: menor a 5 mA.
- Salida de señal: Analógica (0 - VCC).



Fig. 13. Pinout sensor de humedad [4].

4) Sensor de detección de movimiento:

Módulo de detección de movimiento que utiliza tecnología de infrarrojos pasivos (PIR) para detectar movimiento mediante el calor emitido, cuenta con sensibilidad ajustable y un retardo de activación configurable, lo que permite adaptarlo a diferentes aplicaciones, además de que su bajo consumo de energía y facilidad de integración lo hacen ideal para sistemas de detección de movimiento en interiores y exteriores [1]. (Fig. 14).

- Tiempo de inicialización: Aprox. 1 minuto tras encenderse
- Voltaje de operación: 4.5V - 20V
- Distancia de detección: 3 - 7 metros ajustable.
- Ángulo de detección: 120°.
- Modo de disparo: Repetitivo y no repetitivo.
- Tiempo de retardo: 0.3 - 5 segundos ajustable.
- Salida de señal: Nivel alto (3.3V) cuando detecta movimiento, nivel bajo (0V) en reposo.

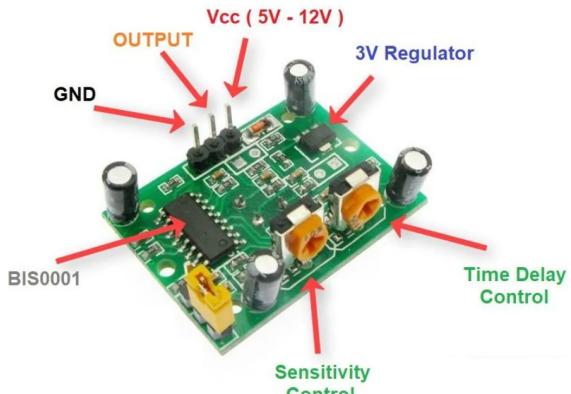


Fig. 14. Pinout sensor PIR [3].

B. Prácticas.

En este apartado se presentarán un total de cinco prácticas, diseñadas para aplicar de manera integral los conceptos previamente abordados. El objetivo principal es evaluar el funcionamiento del kit en distintos escenarios, asegurando su correcto desempeño y verificando que los resultados obtenidos sean óptimos.

1) PRACTICA 1: JUEGO DE LEDS.

El juego consiste en manipular un conjunto de LEDs que pueden adoptar tres estados: apagado, parpadeo y encendido. Además, cambian de color de manera aleatoria.

El objetivo es presionar un botón cuando un LED esté encendido, lo que hará que su color se fije. Si

logras fijar cinco LEDs del mismo color, obtendrás un punto. Para ganar la partida, debes acumular tres puntos, asegurándote de que en cada conjunto de cinco LEDs fijados no se repitan colores.

1. Configuración inicial.

Para comenzar, es necesario incluir las bibliotecas requeridas y declarar el objeto Carrier:

```
#include <Arduino_MKRIoTCarrier.h>
MKRIoTCarrier carrier;
```

La biblioteca Arduino-MKRIoTCarrier es fundamental para que el Carrier funcione correctamente. Asimismo, la línea MKRIoTCarrier carrier; debe incluirse en el código, ya que su ausencia puede impedir el correcto funcionamiento de algunos módulos del Carrier.

2. Estados de los leds.

Los LEDs pueden adoptar los siguientes estados:

- OFF: Apagado.
- BLINKING: Parpadeo.
- ON: Encendido con un color aleatorio.
- FIXED: Color fijado por el jugador.

Inicializamos los estados de los LEDs asignando un estado y un color aleatorio a cada uno:

```
for (int i = 0; i < 5; i++) {
    ledStates[i] = (LedState)random(0, 3);
    ledColors[i] = random(0, 3);
    lastStateChange[i] = millis();
}
```

El cambio de estado de los LEDs se gestiona mediante una estructura switch-case dentro del loop():

```
switch (ledStates[i]) {
    case OFF:
        carrier.leds.setPixelColor(i, 0);
        break;
    case BLINKING:
        if ((millis() / 250) % 2 == 0) {
            carrier.leds.setPixelColor(i, ...
                ...colors[ledColors[i]]);
        } else {
            carrier.leds.setPixelColor(i, 0);
        }
        break;
    case ON:
        carrier.leds.setPixelColor(i, ...
            ...colors[ledColors[i]]);
        break;
    case FIXED:
        carrier.leds.setPixelColor(i, ...
            ...stageColors[stage]);
        break;
}
```

3. Detección de botones.

Cuando el jugador presiona un botón, se mide el tiempo de pulsación:

```
if (carrier.Buttons.getTouch...
    ...((touchButtons)i)) {
    if (buttonPressStart[i] == 0) {
        buttonPressStart[i] = millis();
    } else if (millis() - ...
    ...buttonPressStart[i] >= 3000) {
        ledStates[i] = (LedState)random(0, 3);
        buttonPressStart[i] = 0;
    }
}
```

Si el LED encendido coincide con el color de la etapa actual, su estado cambia a FIXED:

```
if (currentColor == stageColors[stage]) {
    ledStates[i] = FIXED;
}
```

Cuando el jugador logra fijar correctamente cinco LEDs, se avanza a la siguiente etapa.:

```
if (fixedCount == 5) {
    advanceStage();
}
```

4. Mensajes de victoria y sonido.

Al completar una etapa, se muestra un mensaje en pantalla y se activa una melodía:

```
if (stage == 1) {
    carrier.display.setCursor...
    ...(messageX, messageY);
    carrier.display.print("ETAPA 1 WIN");
    playMelody();
} else if (stage == 2) {
    carrier.display.setCursor...
    ...(messageX, messageY);
    carrier.display.print("ETAPA 2 WIN WIN");
    playMelody();
} else if (stage == 3) {
    carrier.display.setCursor...
    ...(messageX, messageY);
    carrier.display.print("YOU WIN");
    playMelody();
}
```

La función playMelody() reproduce una melodía utilizando el buzzer del Carrier:

```
void playMelody() {
    for (int thisNote = 0; thisNote < 4; ...
        ...thisNote++) {
        int noteDuration = 1000 / ...
            ...noteDurations[thisNote];
        carrier.Buzzer.sound(melody[thisNote]);
        delay(noteDuration);
        delay(noteDuration * 0.30);
        carrier.Buzzer.noSound();
    }
}
```

5. Finalización del juego.

Cuando el jugador completa la tercera etapa, el juego finaliza mostrando "YOU WIN" y deteniendo la ejecución del programa:

```
carrier.display.print("YOU WIN");
stopGame();
```

La función stopGame() apaga los LEDs y detiene el código:

```
void stopGame() {
    carrier.leds.fill(0);
    carrier.leds.show();
    while (1);
}
```

2) PRACTICA 2: MONITOREO DEL AMBIENTE.

En esta actividad, desarrollaremos un menú interactivo con cinco opciones: Temperatura, Humedad, Presión, Gases y Giroscopio. Cada una de estas opciones mostrará información diferente en la pantalla del Carrier y podrá ser seleccionada a través de los botones táctiles.

- a) Temperatura: Muestra la temperatura en grados Celsius, Fahrenheit y Kelvin.
- b) Humedad: Indica el porcentaje de humedad en el ambiente.
- c) Presión: Muestra la presión atmosférica en hectopascales (hPa).
- d) Gases: Presenta la resistencia del gas, así como los niveles de compuestos orgánicos volátiles (VOC) y CO₂.
- e) Giroscopio: Muestra las variaciones en los ejes X, Y y Z.

1. Configuración inicial.

Para comenzar, es necesario importar las librerías y declarar el Carrier:

```
#include <Arduino_MKRIoTCarrier.h>
MKRIoTCarrier carrier;
```

Una vez importada la librería, inicializamos el Carrier, mostramos el menú principal e iniciamos el módulo IMU:

```
if (!carrier.begin()) {
    while (1);
}
carrier.display.fillScreen(ST77XX_BLACK);
displayMainMenu();

if (!carrier.IMUmodule.begin()) {
    while (1);
}
```

2. Menú principal.

El menú se mostrará en la pantalla del Carrier con el siguiente formato:

```
void displayMainMenu() {
    carrier.display.fillRect(ST77XX_BLACK);
    carrier.display.setTextColor(ST77XX_WHITE);
    carrier.display.setTextSize(2);
    carrier.display.setCursor(100, 10);
    carrier.display.println("MENU:");

    carrier.display.setTextSize(1);
    carrier.display.setCursor(50, 50);
    carrier.display.println("00: Temperatura");
    carrier.display.setCursor(50, 70);
    carrier.display.println("01: Humedad");
    carrier.display.setCursor(50, 90);
    carrier.display.println("02: Presión");
    carrier.display.setCursor(50, 110);
    carrier.display.println("03: Gases");
    carrier.display.setCursor(50, 130);
    carrier.display.println("04: Giroscopio");
}
```

El usuario podrá seleccionar una opción tocando los botones táctiles del Carrier.

1) Detección de botones: El código detecta la activación de los botones táctiles, permitiendo al usuario navegar entre el menú principal y los submenús:

```
if (carrier.Buttons.getTouch...
...((touchButtons)i)) {
    if (buttonPressStart[i] == 0) {
        buttonPressStart[i] = millis();
    } else if (millis() - ...
...buttonPressStart[i] >= 500) {
        buttonPressStart[i] = 0;

        if (isSubMenu && i != ...
...currentSubMenu) {
            displayMainMenu();
            isSubMenu = false;
            currentSubMenu = -1;
        } else if (!isSubMenu) {
            handleSubMenu(i);
        }
    }
}
```

Si un botón se mantiene presionado durante 500 ms, se activa la opción seleccionada y se muestra el submenú correspondiente. Si el usuario toca otro botón distinto dentro del submenú, regresará al menú principal.

3. Submenús.

Cada botón abre un submenú específico con información relevante:

```
void handleSubMenu(int button) {
    switch (button) {
        case 0:
            displayTemperatureSubMenu();
            currentSubMenu = 0;
            break;
        case 1:
            displayHumiditySubMenu();
            currentSubMenu = 1;
            break;
    }
}
```

```

    case 2:
        displayPressureSubMenu();
        currentSubMenu = 2;
        break;
    case 3:
        displayGasSubMenu();
        currentSubMenu = 3;
        break;
    case 4:
        displayGyroscopeSubMenu();
        currentSubMenu = 4;
        break;
    }
    isSubMenu = true;
}

```

Mientras el usuario se encuentra en un submenú, la pantalla se actualizará cada segundo con las mediciones más recientes del sensor correspondiente:

```

if (isSubMenu) {
    if (millis() - lastUpdate >= 1000) {
        switch (currentSubMenu) {
            case 0:
                updateTemperatureDisplay();
                break;
            case 1:
                updateHumidityDisplay();
                break;
            case 2:
                updatePressureDisplay();
                break;
            case 3:
                updateGasDisplay();
                break;
            case 4:
                updateGyroscopeDisplay();
                break;
        }
        lastUpdate = millis();
    }
}

```

4. Sensores y pantalla.

Cada submenú mostrará información en pantalla según la medición obtenida:

- Temperatura:

```

float temperatureC = ...
...carrier.Env.readTemperature();
float temperatureF = ...
...(temperatureC * 9 / 5) + 32;
float temperatureK = ...
...temperatureC + 273.15;

carrier.display.setCursor(60, 70);
carrier.display.print(temperatureC, 2);
carrier.display.println(" C");
carrier.display.setCursor(60, 110);
carrier.display.print(temperatureF, 2);
carrier.display.println(" F");
carrier.display.setCursor(60, 150);
carrier.display.print(temperatureK, 2);
carrier.display.println(" K");

```

- Humedad:

```

float humidity = ...
...carrier.Env.readHumidity();
carrier.display.setCursor(60, 110);
carrier.display.print(humidity, 2);
carrier.display.println(" %");

```

- Presión Atmosférica:

```

float pressure = carrier.Pressure.readPressure() / ...
carrier.display.setCursor(55, 110);
carrier.display.print(pressure, 2);
carrier.display.println(" hPa");

```

- Gases (VOC y CO₂):

```

float gasResistor = ...
...carrier.AirQuality.readGasResistor();
float volatileOrganicCompounds = carrier.AirQuality...
float co2 = carrier.AirQuality.readCO2();

carrier.display.print("R: ");
carrier.display.print(gasResistor, 2);
carrier.display.println(" ohms");
carrier.display.print("VOC: ");
carrier.display.print...
...(volatileOrganicCompounds, 2);
carrier.display.println(" ppm");
carrier.display.print("CO2: ");
carrier.display.print(co2, 2);
carrier.display.println(" ppm");

```

- Giroscopio:

```

if (carrier.IMUmodule.accelerationAvailable()) {
    carrier.IMUmodule.readAcceleration(x, y, z);
    carrier.display.print("x: "); carrier.display.p...
    carrier.display.print("y: "); carrier.display.p...
    carrier.display.print("z: "); carrier.display.p...
}

```

3) PRACTICA 3: MULTI SENSORES.

En esta tercera actividad, trabajaremos con los sensores de RGB y gestos. Para ello, crearemos un menú interactivo con tres opciones: gestos, colores y proximidad. Cada opción ejecutará una función específica relacionada con los sensores.

- Gestos: El sensor detectará movimientos en las direcciones arriba, abajo, derecha e izquierda, mostrando un ícono correspondiente, como una flecha o un triángulo, indicando la dirección detectada.
- Se identificarán los colores rojo, verde, azul y blanco. Cuando se detecte uno de estos, se mostrará un mensaje en pantalla indicando el color reconocido.
- Proximidad: Los LEDs y el buzzer se activarán según la distancia medida:
 - Verde → Distancia mayor a 200 mm.
 - Amarillo → Entre 120 mm y 200 mm.
 - Rojo → Menos de 120 mm.

1. Configuración inicial.

Para comenzar, importamos las librerías necesarias:

```
#include <Arduino_MKRIoTCarrier.h>
#include <Arduino_APDS9960.h>
#include "pitches.h"
MKRIoTCarrier carrier;
```

La biblioteca Arduino-APDS9960 es fundamental para el correcto funcionamiento del sensor, permitiendo la detección de gestos, colores y proximidad. También se incluye el archivo pitches.h, que contiene las notas musicales necesarias para reproducir sonidos con el buzzer.

A continuación, inicializamos el Carrier y verificamos que el sensor APDS9960 funcione correctamente. En caso de error, se mostrará un mensaje en pantalla:

```
carrier.begin();
carrier.display.fillScreen(ST77XX_BLACK);

if (!APDS.begin()) {
    carrier.display.fillScreen(ST77XX_RED);
    carrier.display.setCursor(10, 60);
    carrier.display.setTextSize(2);
    carrier.display.setTextColor(ST77XX_WHITE);
    carrier.display.println("Error APDS!");
    while (true);
}
carrier.leds.begin();
displayMainMenu();
```

3. Menú principal.

El menú principal se mostrará en la pantalla del Carrier:

```
void displayMainMenu() {
    currentMenu = 0;
    carrier.display.fillScreen(ST77XX_BLACK);
    carrier.display.setTextSize(2);
    carrier.display.setTextColor(ST77XX_WHITE);

    carrier.display.setCursor(50, 70);
    carrier.display.println("01. Gestos");
    carrier.display.setCursor(50, 100);
    carrier.display.println("02. Colores");
    carrier.display.setCursor(50, 130);
    carrier.display.println("04. Proximidad");
    carrier.display.setCursor(50, 160);
    carrier.display.println("00. Volver");
```

4. Detección de gestos.

Cuando el usuario selecciona la opción de gestos, el código espera detectar un gesto y lo muestra en pantalla con una flecha visual:

```
if (APDS.gestureAvailable()) {
    int gesture = APDS.readGesture();
    carrier.display.fillScreen(ST77XX_BLACK);

    switch (gesture) {
        case GESTURE_UP:
            carrier.display.fillTriangle...
            (60, 120, 120, 80, 120, 160, ...
             ...ST77XX_WHITE);
            carrier.display.fillRect...
```

```
(120, 100, 60, 40, ...
 ...ST77XX_WHITE);
 playTone1(NOTE_B0, NOTE_C1);
 break;

case GESTURE_DOWN:
    carrier.display.fillTriangle...
    (180, 120, 120, 80, 120, 160, ...
     ...ST77XX_WHITE);
    carrier.display.fillRect...
    (60, 100, 60, 40, ST77XX_WHITE);
    playTone1(NOTE_B0, NOTE_C1);
break;

case GESTURE_LEFT:
    carrier.display.fillTriangle...
    (120, 180, 80, 120, 160, 120, ...
     ...ST77XX_WHITE);
    carrier.display.fillRect...
    (100, 60, 40, 60, ST77XX_WHITE);
    playTone1(NOTE_B0, NOTE_C1);
break;

case GESTURE_RIGHT:
    carrier.display.fillTriangle...
    (120, 60, 80, 120, 160, 120, ...
     ...ST77XX_WHITE);
    carrier.display.fillRect...
    (100, 120, 40, 60, ST77XX_WHITE);
    playTone1(NOTE_B0, NOTE_C1);
break;

default:
    carrier.display.setCursor(80, 60);
    carrier.display.println...
    ("GESTO NO RECONOCIDO");
    break;
}
```

Cada vez que se detecta un gesto, se dibuja una flecha en la pantalla y se reproduce un sonido con el buzzer.

5. Detección de colores.

Antes de utilizar el sensor de color, se debe calibrar mediante pruebas para establecer rangos adecuados de detección. Luego, el sensor lee los valores RGB y muestra el color detectado en pantalla:

```
if (APDS.colorAvailable()) {
    int r, g, b;
    APDS.readColor(r, g, b);

    if (r > 500 && g > 390 && b > 390) {
        carrier.display.fillScreen...
        ... (ST77XX_YELLOW);
        carrier.display.println("DETECTADO");
    } else if (r > 350 && r < 390 && g > ...
    ... 50 && g < 390 && b > 85 && b < 390) {
        carrier.display.fillScreen(ST77XX_RED);
        carrier.display.println("ROJO");
    } else if (r > 150 && g > 290 && g ...
    ... < 390 && b > 150) {
        carrier.display.fillScreen(ST77XX_GREEN);
        carrier.display.println("VERDE");
    } else if (r > 55 && r < 390 && g > ...
    ... 150 && b < 150) {
        carrier.display.fillScreen(ST77XX_BLUE);
        carrier.display.println("AZUL");
    }
}
```

```

...100 && g < 390 && b > 230 && b < 390) {
    carrier.display.fillRect(ST77XX_BLUE);
    carrier.display.println("AZUL");
}
}

```

6. Detección de proximidad

El sensor mide la distancia de un objeto y enciende los LEDs según los rangos definidos. Además, el buzzer emitirá un sonido correspondiente:

```

if (APDS.proximityAvailable()) {
    int proximity = APDS.readProximity();

    if (proximity > 200) {
        carrier.leds.fill(carrier.-
            leds.Color(0, 255, 0)); // Verde
        playTone1(NOTE_G1, NOTE_A1);
    } else if (proximity > 120) {
        carrier.leds.fill(carrier.-
            leds.Color(255, 255, 0)); // Amarillo
        playTone2(NOTE_D5, NOTE_B4);
    } else {
        carrier.leds.fill(carrier.-
            leds.Color(255, 0, 0)); // Rojo
        playTone3(NOTE_F2, NOTE_E2);
    }
    carrier.leds.show();
}

```

7. Función del buzzer

Para activar el buzzer en diferentes situaciones:

```

void playTone1(int firstNote, ...
...int secondNote) {
    carrier.Buzzer.sound(firstNote);
    delay(50);
    carrier.Buzzer.noSound();
    delay(10);
    carrier.Buzzer.sound(secondNote);
    delay(50);
    carrier.Buzzer.noSound();
}

```

4) PRACTICA 4: WIFI Y BLE.

En esta práctica, exploraremos las conexiones Wi-Fi y Bluetooth Low Energy (BLE) mediante un menú interactivo con dos opciones principales:

- Wi-Fi: Se escanearán las redes Wi-Fi de 2.4 GHz disponibles. Luego, se seleccionará una red a través de los botones táctiles o el puerto serial. Tras ingresar la contraseña vía serial, el sistema intentará conectarse y mostrará la dirección IP obtenida en la pantalla del Carrier si la conexión es exitosa.
- BLE (Bluetooth Low Energy): Se activará un servicio BLE para permitir la comunicación con otros dispositivos. Se publicará un servicio de lectura y escritura, permitiendo conexiones inalámbricas.

1. Configuración inicial.

Para comenzar, importamos las librerías necesarias:

```

#include <Arduino_MKRIoTCarrier.h>
#include <WiFiNINA.h>
#include <ArduinoBLE.h>
MKRIoTCarrier carrier;

```

Las librerías WiFiNINA y ArduinoBLE permiten la inicialización y correcta gestión de las conexiones Wi-Fi y Bluetooth. También se incluye la librería ArduinoMKRIoTCarrier para el manejo de la pantalla y otros módulos.

A continuación, inicializamos la comunicación serial y los módulos del Carrier, Wi-Fi y BLE. Si algún módulo no puede inicializarse, se mostrará un mensaje de error:

```

Serial.begin(9600);
while (!Serial);
if (!carrier.begin()) {
    Serial.println("ERROR CARRIER");
    while (1);
}
if (!BLE.begin()) {
    Serial.println("ERROR INICIANDO BLE");
    while (1);
}
Serial.println(".....");
showMenu();

```

2. Menú principal.

El menú principal se visualizará en la pantalla del Carrier y permitirá seleccionar Wi-Fi o BLE:

```

void showMenu() {
    carrier.display.fillRect(ST77XX_BLACK);
    carrier.display.setTextColor(ST77XX_WHITE);
    carrier.display.setTextSize(2);
    carrier.display.setCursor(90, 40);
    carrier.display.println("MENU");

    carrier.display.setCursor(55, 100);
    carrier.display.println("01. WI-FI");

    carrier.display.setCursor(50, 140);
    carrier.display.println("03. BLE");
}

```

3. Escaneo y conexión Wi-Fi.

Cuando el usuario selecciona la opción de Wi-Fi, el sistema escanea las redes disponibles, mostrando un máximo de 6 redes con la siguiente información:

- Nombre de la red (SSID).
- Intensidad de la señal (RSSI).
- Canal de transmisión.

```

networkCount = min(WiFi.scanNetworks(), 6);
if (networkCount == 0) {
    carrier.display.println...
    ...("NO SE ENCONTRO NADA");
    delay(2000);
    showMenu();
    return;
}

for (int i = 0; i < ...

```

```

...networkCount && i < 6; i++) {
    ssidList[i] = WiFi.SSID(i);
    rssiList[i] = WiFi.RSSI(i);
    channelList[i] = WiFi.channel(i);
}

```

Para seleccionar una red, el usuario utilizará los botones táctiles:

- TOUCH1: Mueve la selección hacia arriba.
- TOUCH3: Mueve la selección hacia abajo.
- TOUCH0: Confirma la selección.

```

if (carrier.Buttons.onTouchDown(TOUCH3)) {
    currentSelection = ...
    ... (currentSelection + 1) % networkCount;
} else if -
(carrier.Buttons.onTouchDown(TOUCH1)) {
    currentSelection = ...
    ... (currentSelection - 1 + networkCount) ...
    ... % networkCount;
} else if -
(carrier.Buttons.onTouchDown(TOUCH0)) {
    networkSelected = true;
}

```

Una vez seleccionada la red, se solicitará la contraseña mediante el puerto serial:

```

Serial.print("CONTRASEÑA: ");
String password = ...
...Serial.readStringUntil('\n');
WiFi.begin(ssidList[index].c_str(), ...
...password.c_str());

```

Si la conexión es exitosa, se mostrará la dirección IP en la pantalla del Carrier:

```

carrier.display.fillScreen(ST77XX_BLACK);
carrier.display.setTextSize(2);
carrier.display.setCursor(70, 80);
carrier.display.setTextColor(ST77XX_GREEN);
carrier.display.println("CONECTADO");
carrier.display.setCursor(20, 120);
carrier.display.println("IP: ");
carrier.display.println(WiFi.localIP());

```

5. Configuración BLE.

Cuando se selecciona la opción BLE, el dispositivo comenzará a transmitir su presencia con la siguiente configuración:

- Se establece el nombre "CARRIER".
- Se crea un servicio BLE con una característica de lectura/escritura.
- Se inicia la transmisión para que otros dispositivos puedan detectarlo.

```

BLE.setLocalName("CARRIER");
BLE.setAdvertisedService(genericService);
genericService.-
addCharacteristic(readWriteCharacteristic);
BLE.addService(genericService);
BLE.advertise();

```

El sistema permanecerá en un bucle hasta que un dispositivo se conecte. Cuando se detecta una conexión, se muestra en pantalla la dirección MAC del dispositivo conectado:

```

while (true) {
    BLEDevice central = BLE.central();
    if (central) {
        bleConnected = true;
        Serial.print("CONECTADO A: ");
        Serial.println(central.address());
    }
}

```

Si el dispositivo BLE se desconecta, se procederá de la siguiente manera:

- Se muestra en pantalla el mensaje "DESCONECTADO".
- La pantalla se pinta de color rojo para indicar la desconexión.
- Se esperarán dos segundos.
- El sistema regresa automáticamente al menú principal.

```

Serial.println("DESCONECTADO");
carrier.display.fillScreen(ST77XX_RED);
carrier.display.setCursor(50, 100);
carrier.display.setTextSize(2);
carrier.display.setTextColor(ST77XX_WHITE);
carrier.display.println("DESCONECTADO");
delay(2000);
showMenu();

```

5) PRACTICA 5: SENSORES EXTERNOS.

En esta última actividad, trabajaremos con sensores externos a través de un menú interactivo que permite seleccionar entre el sensor de humedad del suelo y el sensor PIR (detección de movimiento). La navegación se controla mediante:

- Tecla '1' o botón TOUCH1: Activa el sensor de humedad.
- Tecla '2' o botón TOUCH3: Activa el sensor PIR.
- Tecla 'r' o botón TOUCH2: Regresa al menú principal.

1. Configuración inicial.

Para comenzar, importamos las librerías necesarias e inicializamos el Carrier para garantizar su correcto funcionamiento:

```
#include <Arduino_MKRIoTCarrier.h>
MKRIoTCarrier carrier;
```

A continuación, configuramos los pines de los sensores y mostramos el menú principal:

- Se inicializa la comunicación Serial para depuración.
- Se inicia el Carrier MKR IoT Rev2 y se verifica su correcto funcionamiento.
- Se esperarán dos segundos.

d) Se configura el sensor PIR como entrada digital.

```
Serial.begin(9600);
while (!Serial);
CARRIER_CASE = false;
if (!carrier.begin()) {
    Serial.println("CARRIER ERROR");
    while (1);
}
pinMode(PIR_PIN, INPUT);
```

2. Menú principal.

El menú principal se muestra en la pantalla del Carrier para permitir la selección de sensores:

```
void showState() {
    carrier.display.fillScreen(ST77XX_BLACK);
    carrier.display.setCursor(0, 0);
    carrier.display.setTextSize(3);

    if (currentState == 0) {
        Serial.println("-----");
        Serial.println("MENU:");
        Serial.println("01. Humedad");
        Serial.println("03. PIR");
        Serial.println("-----");

        carrier.display.setCursor(85, 40);
        carrier.display.println("MENU:");

        carrier.display.setCursor(30, 100);
        carrier.display.println("01. Humedad");

        carrier.display.setCursor(30, 140);
        carrier.display.println("03. PIR");
    }
}
```

3. Navegación del menú.

El código gestiona la selección de sensores mediante los botones táctiles del Carrier:

```
if (currentState == 0) {
    if (carrier.Buttons.onTouchDown(TOUCH1)) {
        currentState = 1; // Humedad
    }
    if (carrier.Buttons.onTouchDown(TOUCH3)) {
        currentState = 2; // PIR
    }
} else {
    if (carrier.Buttons.onTouchDown(TOUCH2)) {
        currentState = 0; // Regresar al menú
    }
}
```

4. Sensor de humedad del suelo.

El sensor convierte la lectura analógica en un porcentaje de humedad. Antes de realizar mediciones, es necesario calibrarlo:

```
else if (input == "c") {
    dryValue = analogRead(MOISTURE_PIN);
    Serial.print("Calibracion de suelo ...
    ...seco completada. dryValue = ");
```

```
    Serial.println(dryValue);
}

else if (input == "w") {
    wetValue = analogRead(MOISTURE_PIN);
    Serial.print..."Calibracion de ...
    ...suelo húmedo completada. wetValue = ";
    Serial.println(wetValue);
}
```

- "c": Ajusta el valor del suelo seco.
- "w": Ajusta el valor del suelo húmedo.

Una vez completada la calibración, procedemos a tomar las mediciones en donde:

```
int sensorValue = analogRead(MOISTURE_PIN);
int humedadPercent = map...
... (sensorValue, dryValue, wetValue, 0, 100);

if (humedadPercent < 0) humedadPercent = 0;
if (humedadPercent > 100) humedadPercent = 100;

String estadoSuelo = ...
... (humedadPercent > 50) ? "Humedo" : "Seco";
```

Los resultados se muestran en la pantalla del Carrier y en el puerto Serial:

```
Serial.print(" Valor sensor: ");
Serial.print(sensorValue);
Serial.print(" Humedad: ");
Serial.print(humedadPercent);
Serial.print(" Estado: ");
Serial.println(estadoSuelo);

carrier.display.fillScreen(ST77XX_BLACK);
carrier.display.setCursor(60, 40);
carrier.display.setTextSize(3);
carrier.display.println("HUMEDAD");

carrier.display.setTextSize(2);
carrier.display.setCursor(40, 90);
carrier.display.print("Lectura: ");
carrier.display.println(sensorValue);

carrier.display.setCursor(40, 120);
carrier.display.print("Hum: ");
carrier.display.print(humedadPercent);
carrier.display.println("%");

carrier.display.setCursor(40, 150);
carrier.display.print("Estado: ");
carrier.display.println(estadoSuelo);
```

5. Sensor PIR

El sensor PIR detecta movimiento mediante múltiples lecturas consecutivas:

```
const int numReadings = 5;
const int threshold = 2;
int countHigh = 0;

for (int i = 0; i < numReadings; i++) {
    if (digitalRead(PIR_PIN) == HIGH) {
        countHigh++;
    }
}
```

```
delay(50);
}
```

Si se detecta movimiento, la pantalla cambia a rojo y se muestra el mensaje correspondiente:

```
if (countHigh >= threshold) {
    carrier.display.fillScreen(ST77XX_BLACK);
    carrier.display.setCursor(95, 30);
    carrier.display.setTextSize(3);
    carrier.display.println("PIR");

    carrier.display.fillScreen(ST77XX_RED);
    carrier.display.setTextSize(2);
    carrier.display.setCursor(65, 110);
    carrier.display.println("MOVIMIENTO");
    Serial.println("Hay movimiento");
}
```

Si no hay movimiento, se muestra un mensaje indicando "SIN MOVIMIENTO":

```
else {
    carrier.display.fillScreen(ST77XX_BLACK);
    carrier.display.setCursor(95, 30);
    carrier.display.setTextSize(3);
    carrier.display.println("PIR");

    carrier.display.setTextSize(2);
    carrier.display.setCursor(35, 100);
    carrier.display.println("SIN MOVIMIENTO");
    Serial.println("No hay movimiento");
}
```

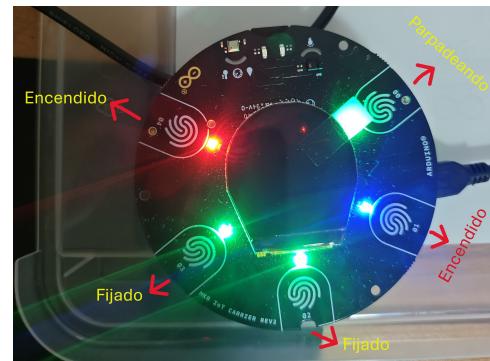


Fig. 16. Estados de LEDs versión 2.

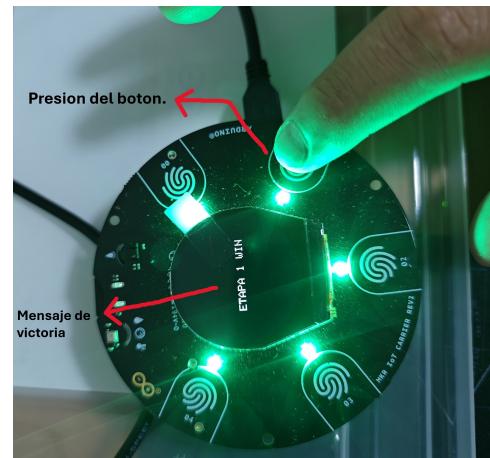


Fig. 17. Mensaje de victoria de la etapa.

III. RESULTADOS.

A. PRACTICA 1:

En las siguientes imágenes se muestran los estados de los LEDs tras cargar el código en el Carrier. Como se puede observar, cada LED asigna su estado de forma aleatoria lo que indica que este a primera vista funciona correctamente.



Fig. 15. Estados de LEDs versión 1.

En la siguiente imagen, se observa que al presionar el último LED en estado de encendido, este queda fijado, completando

así la primera etapa y como resultado, la pantalla muestra el mensaje de victoria previamente definido en el código.

Procedemos a verificar la última etapa, donde confirmamos que el mensaje final se muestra correctamente, lo que indica que el código funciona correctamente.



Fig. 18. Juego completado, mensaje de victoria.

Para finalizar, observamos que los LEDs se apagan y el mensaje de victoria final permanece en la pantalla, sin que los LEDs vuelvan a encenderse, lo que nos indica claramente que el juego ha terminado.



Fig. 19. Juego finalizado.

B. PRACTICA 2:

Ahora veamos el comportamiento de la segunda actividad y vemos que el menú se muestra correctamente en la pantalla del Carrier, permitiendo la selección de opciones de manera.



Fig. 20. Menú principal

Verificamos que los botones funcionen ingresando a la primera opción para asegurarnos de que la navegación dentro del menú se realiza correctamente.



Fig. 21. Submenú de temperatura.

Como se puede observar, el submenú de temperatura funciona correctamente según lo establecido, mostrando la temperatura en Celsius, Fahrenheit y Kelvin.

Continuamos explorando los diferentes submenús, verificando que cada uno funcione correctamente y muestre la información correspondiente en la pantalla del Carrier.



Fig. 22. Submenú de humedad.

El sensor de humedad muestra correctamente el valor y se actualiza de forma periódica.

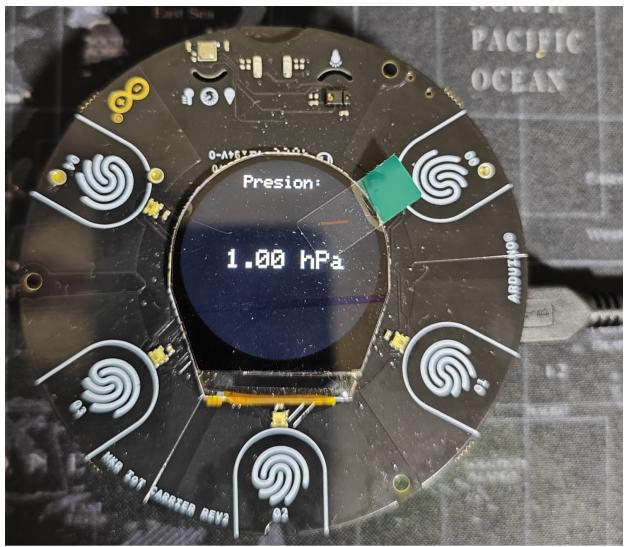


Fig. 23. Submenú de presión.

Para la opción de presión vemos que muestra el valor correspondiente en hectopascales.



Fig. 24. Submenú de gases.

El submenú de gases hace las mediciones solicitadas y muestra los valores como fue indicado.

Para la opción de giroscopio, las siguientes imágenes muestran que los valores de X, Y y Z responden correctamente a los movimientos del Carrier, reflejando cambios en las mediciones en tiempo real. Esto confirma que el sensor integrado funciona de manera adecuada. Además, esta funcionalidad podría expandirse con nuevas aplicaciones en el futuro; sin embargo, por el momento, se ha verificado exitosamente su correcto funcionamiento como vemos a continuación.



Fig. 25. Submenú de giroscopio versión 1.



Fig. 26. Submenú de giroscopio versión 2.

Finalizamos esta actividad confirmando que la navegación entre los submenús funciona correctamente y que la información mostrada en cada uno de ellos se sensa y se muestra correctamente. Incluso en el apartado de giroscopio, se logra detectar la ubicación y visualizar los cambios en los ejes X, Y y Z en tiempo real.

C. PRACTICA 3:

Continuando con la tercera practica los resultados son los siguientes:



Fig. 27. Menú principal.

El menú de opciones se muestra correctamente en la pantalla, y la navegación entre los submenús funciona de manera fluida y correcta.

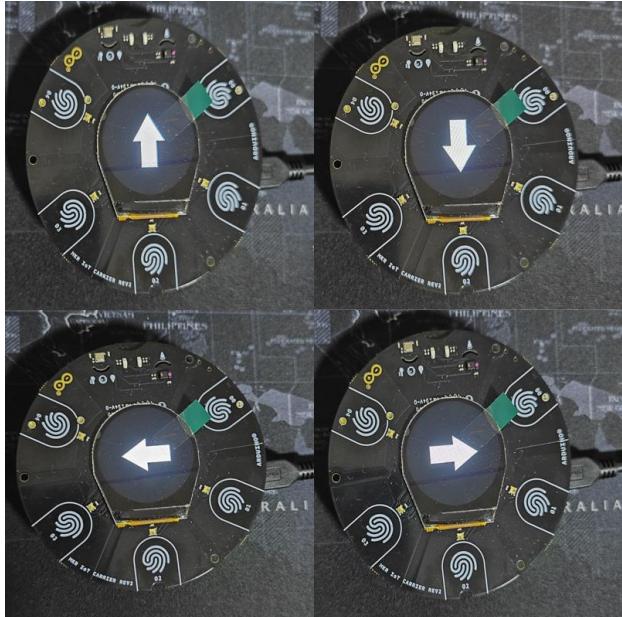


Fig. 28. Detección de gestos

Como se muestra en la imagen anterior, la opción de detección de gestos funciona correctamente, identificando los cuatro gestos establecidos.



Fig. 29. Comparativa de RGB

En la opción de detección de colores, inicialmente verificamos que la detección de los rangos RGB funciona correctamente al comparar mediciones en un entorno con buena iluminación y otro con poca iluminación.

Ahora comprobamos que la detección de colores funciona correctamente, identificando los colores principales del RGB, así como el color blanco.

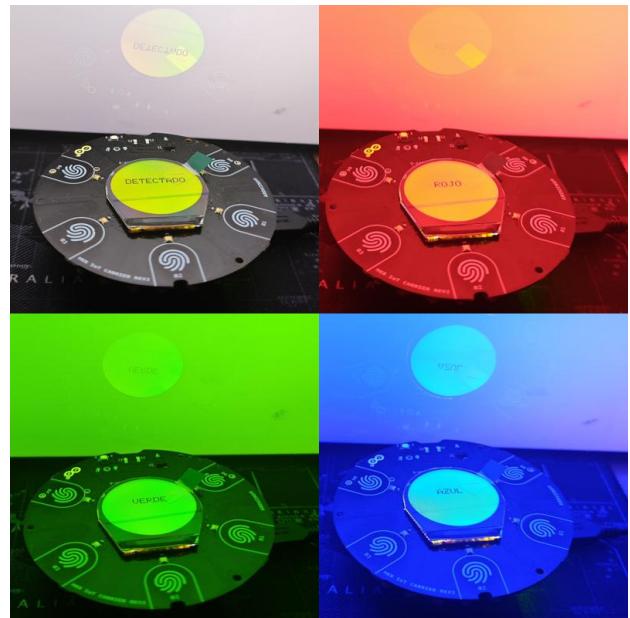


Fig. 30. Comparativa de RGB

Como se observa anteriormente, el sensor funciona correctamente, ya que, al detectar un color, la pantalla del Carrier se pinta con dicho color y muestra un mensaje indicando el nombre correspondiente. En el caso del color blanco, la pantalla muestra únicamente el mensaje **DETECTADO**, confirmando así el correcto desempeño del sensor.

Para finalizar la práctica la última opción que utiliza el sensor de proximidad funciona correctamente. Aunque presenta un pequeño margen de error, su comportamiento es adecuado a la necesidad a atender.



Fig. 31. Escenarios de proximidad.

D. PRACTICA 4:

Continuamos con la práctica 4, en la cual utilizamos los módulos Wi-Fi y BLE del Arduino MKR WiFi 1010. Para ello, se creó un menú interactivo, que se muestra en la pantalla del Carrier como se muestra a continuación.



Fig. 32. Menú de opciones.

Ahora veremos como funciona la primera opción correspondiente a Wi-fi.



Fig. 33. Proceso Wi-fi.

En las imágenes se puede observar que el módulo Wi-Fi se inicializa correctamente, permitiendo la búsqueda de redes disponibles. Además, el menú interactivo para seleccionar la red funciona sin problemas y se cumple con el objetivo de que al conectarse a la red se muestra la dirección IP de la red y en caso de fallo en la conexión, el sistema muestra el mensaje de "FALLO", tal como se evidencia en la imagen.

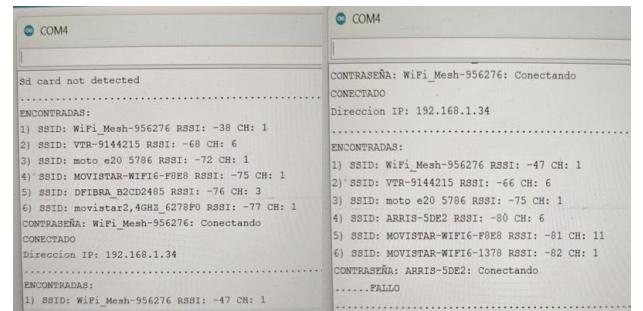


Fig. 34. Puerto serial del Wi-fi.

Ahora continuamos con la segunda opción de BLE, para esta vemos que como paso principal se inicializa el modulo de forma exitosa y publica el servicio de lectura y escritura.

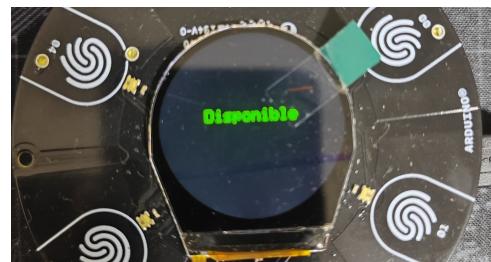


Fig. 35. Inicialización de BLE.

Una vez publicado el servicio comprobamos que este sea visible para dispositivos externos y ademas permitan una conexión con este mismo.

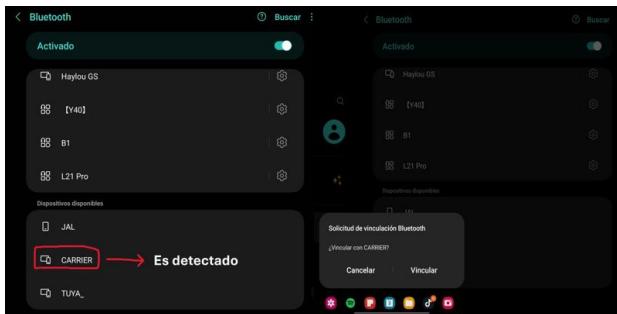


Fig. 36. Búsqueda y conexión.

Una vez que se vincula al dispositivo este manda un mensaje de conexión mostrando la dirección MAC de dispositivo al que esta conectado y lo muestra tanto en el serial como en la pantalla del Carrier.



Fig. 37. Conexión exitosa.

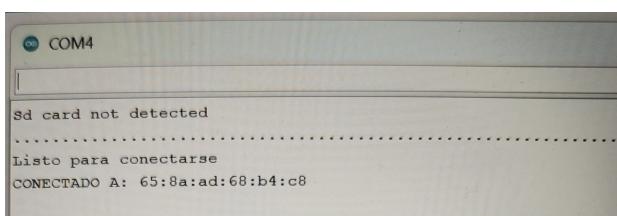


Fig. 38. Datos del puerto serial.

E. PRACTICA 5:

En la última práctica, probamos los sensores incluidos en el kit, los cuales permiten medir la humedad del suelo y detectar movimiento. Como primer paso, se muestra el menú principal, donde se presentan las dos opciones disponibles.



Fig. 39. Menú principal.

A continuación el comportamiento del sensor de humedad del suelo, mostrando los valores registrados y su interpretación en función de la calibración realizada.



Fig. 40. Mediciones con el sensor de humedad.

También podemos ver esta información en el puerto serial, esto con el fin de tener un registro extra.

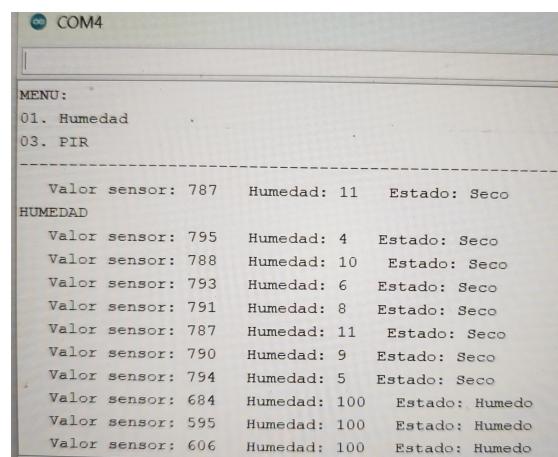


Fig. 41. Puerto serial sensor de humedad.

Ahora la segunda opción referente al sensor PIR podemos ver que este funciona correctamente y que registra los valores tanto en la pantalla como en el puerto serial tal cual se ve a continuación.



Fig. 42. Comportamiento sensor PIR.

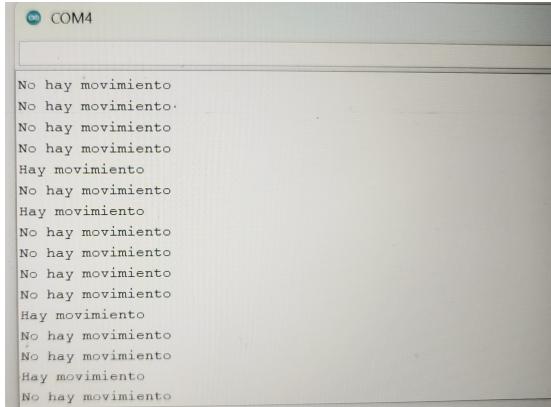


Fig. 43. Puerto serial PIR.

Para finalizar esta actividad, el sensor PIR detecta movimiento según la calibración realizada. Para propósitos prácticos, se configuró para emitir cinco pulsos infrarrojos y, si al menos tres de estas lecturas resultan en ON, se considerará que hay movimiento, según lo definido en el código. Además, para realizar una calibración manual, se realiza lo siguiente.

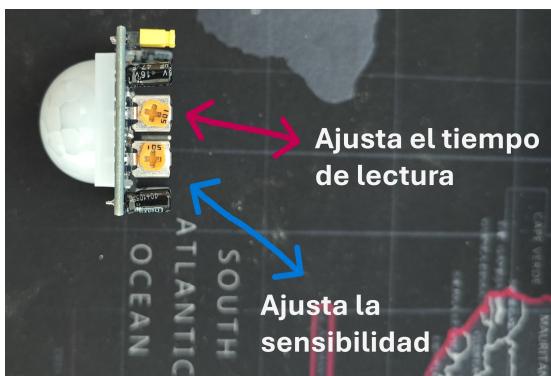


Fig. 44. Potenciómetros del PIR.

Ajusta los potenciómetros según tus necesidades. El potenciómetro relacionado con la sensibilidad también regula la distancia de detección, la cual varía entre 3 metros y 7 metros, en donde para aumentar el rango de detección, gira el potenciómetro hacia la derecha sentido antihorario.

El potenciómetro correspondiente al tiempo determina el intervalo entre detecciones o disparos. Para obtener más lecturas y reducir el delay entre ellas, gírello en sentido antihorario, permitiendo así una detección más rápida y frecuente.

También tenemos los modos de repeat trigger y single trigger los cuales sirven de la siguiente manera:



Fig. 45. Modo single trigger.

El modo single trigger se activa una vez por detección y se apaga tras el tiempo configurado, aunque siga el movimiento. Este es el modo que usamos en el ejemplo.

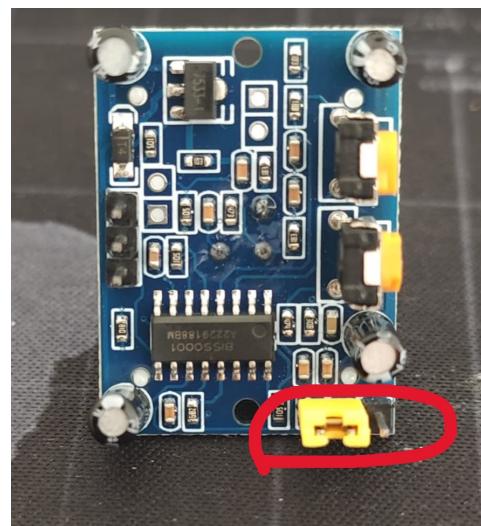


Fig. 46. Modo repeat trigger.

El modo repeat trigger se mantiene activa mientras haya movimiento y deja de continuar hasta que no haya movimiento.

IV. ERRORES Y SOLUCIONES.

A. Botones.

Un problema con los botones fue su alta sensibilidad, lo que provocaba que registraran múltiples toques con un solo toque, para solucionar esto, puedes utilizar lo siguiente:

1) *Estado del botón:* Ocupamos un booleano para que la acción solo cuente si antes el botón estaba en estado false o apagado.

```
bool estadoBoton[5] = {false};

void loop() {
    for (int i = 0; i < 5; i++) {
        bool estadoActual =
            ...carrier.Buttons.getTouch(i);

        if (estadoActual && !estadoBoton[i]) {
            Serial.print("Botón ");
            Serial.print(i);
            Serial.println("funcional");
        }
        estadoBoton[i] = estadoActual;
    }
}
```

2) *Hacer pausa:* Agregamos un pequeño delay después de detectar un toque para evitar una lectura falsa.

```
void loop() {
    for (int i = 0; i < 5; i++) {
        if (carrier.Buttons.getTouch(i)) {
            Serial.print("Botón ");
            Serial.print(i);
            Serial.println("funcional");
            delay(300); // Tiempo
        }
    }
}
```

3) *Sostener el botón:* Hacer el usuario mantenga el dedo en el botón por un tiempo mínimo antes de contar la pulsación.

```
const int tiempoToque = 500;
unsigned long inicioB[5] = {0};
bool presionando[5] = {false};

void loop() {
    for (int i = 0; i < 5; i++) {
        if (carrier.Buttons.getTouch(i)) {
            if (!presionando[i]) {
                inicioB[i] = millis();
                presionando[i] = true;
            }

            if (millis() - inicioB[i] >
                ...tiempoToque) {
```

```
                Serial.print("Botón ");
                Serial.print(i);
                Serial.println("funcional");
                presionando[i] = false;
            }
        } else {
            presionando[i] = false;
        }
    }
}
```

B. Sensores.

1) *BME688:* Otro problema que surgió fue con la inicialización del sensor ambiental BME688, el cual no respondía debido a un fallo en la comunicación I2C. Para solucionar esto, se reinició el carrier presionando dos veces el botón de reset, lo que permitió restablecer la placa y además, en el código se añadieron mensajes de verificación para confirmar el correcto funcionamiento del carrier y finalmente se ajustó la velocidad de transmisión de 9600 a 115200 baudios, ya que es más óptima para esta placa y da menos errores con esta velocidad.

```
void setup() {
    Serial.begin(115200);
    if (!carrier.begin()) {
        Serial.println("ERROR");
        while (1);
    }
    Serial.println("Carrier listo.");
}
```

2) *APDS-9960.:* Se identificó un error en la opción de gestos ya que la detección de gestos en su mayoría eran erróneas. Para solucionar este problema, se puede utilizar lo siguiente:

- **Sensibilidad:** Podemos variar la sensibilidad del sensor siendo 1 el mínimo y 10 el máximo, por las pruebas de 5 a 7 es lo óptimo para lecturas rápidas con un mínimo margen de error.
portador.Gesture.setSensitivity(6);
- **Pausas:** Para garantizar la detección de un gesto podemos colocar pausas entre detecciones.

```
void loop() {
    int gesto = carrier.Gesture.read();
    if (gesto != GESTURE_NONE) {
        Serial.print("Gesto detectado: ");
        Serial.println(gesto);
        delay(500); //
```

Otro error encontrado es en la detección de proximidad ya que su sensibilidad era muy alta, sus lecturas también eran rápidas, para esto se puede optar por lo siguiente:

- Sensibilidad: El sensor es bastante sensible a los objetos cualquier es detectada y mide esa distancia ya que por defecto esta en 8 a 9 de sensibilidad, para esto modificamos su sensibilidad.

```
portador.Proximity.setSensitivity(6);
```

- Promediar: Otra solución es promediar las lecturas y bajo esto tener un menor margen de error.

```
int promedio(int n) {
    int suma = 0;
    for (int i = 0; i < n; i++) {
        suma += carrier.Proximity.read();
        delay(10);
    }
    return suma / n;
}

void loop() {
    int proximidad = promedio(5);
    Serial.print("Proximidad: ");
    Serial.println(proximidad);
}
```

- Lecturas rápidas: Las lecturas del sensor de proximidad son inestables y cambian rápidamente para esto podemos optar por lo siguiente.

```
float lecturaSuavizada = 0.0;
const float suavizado = 0.1;

void loop() {
    int nuevaLectura = ...
    ...carrier.Proximity.read();
    lecturaSuavizada = ...
    ... (suavizado * nuevaLectura) + ...
    ... ((1 - suavizado) * lecturaSuavizada);

    Serial.print("Proximidad filtrada: ");
    Serial.println(lecturaSuavizada);
}
```

C. Comunicaciones

1) Wi-Fi.: Algunos errores que se tuvieron fueron:

- Obtener IP: Para obtener la IP es necesario que este conectado a la red por eso ocupamos un ciclo while.

```
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
}
```

```
Serial.print("IP: ");
Serial.println(WiFi.localIP());
```

- Limitar escáner: Otro problema fue el sobre saturar el sistema teniendo muchas redes ya que causa retrasos para esto es mejor limitar el numero de redes a escanear.

```
int redes = WiFi.scanNetworks();
for (int i = 0; i < min(5, redes); i++)
    Serial.println(WiFi.SSID(i));
```

Una mejora que se podría implementar para mejorar el funcionamiento trabajando con Wi-Fi es hacer que el código no se atasque o bujee para esto podemos ocupar millis.

```
unsigned long tiempoInicio = millis();
const int tiempoMaximo = 10000;

WiFi.begin("Red wifi", "Contraseña");

while (WiFi.status() != ...
...WL_CONNECTED && millis() - ...
...tiempoInicio < tiempoMaximo) {
    delay(500);
    Serial.print(".");
}

if (WiFi.status() == WL_CONNECTED) {
    Serial.println("CONECTADO.");
} else {
    Serial.println("FALLO");
}
```

2) BLE.: En lo que respecta a la comunicación BLE surgió el error de que el dispositivo no era detectado por otros. La solución es ocupar **BLE.advertise()**; esto hará que el nombre sea visible para otros dispositivos.

```
BLE.begin();
BLE.setLocalName("Carrier");
BLE.advertise();
```

Finalizando una mejora a implementar para mejorar la conexión BLE es hacer que este una vez se desconecta del dispositivo vuelva a hacer visible para conectarse a otro dispositivo.

```
void loop() {
    BLEDevice central = BLE.central();

    if (central) {
        Serial.println("Conectado.");
        while (central.connected()) {
            BLE.poll();
        }
        Serial.println("Reiniciando");
        BLE.advertise();
    }
}
```

D. Sensores externos.

En cuanto al sensor externo de de humedad de suelo no se encontraron errores al trabajar con el ya que es bastante sencillo el manipularlo sin embargo si se pueden hacer algunas mejoras en cuanto a código y su comportamiento para esto se propone lo siguiente.

1) Sensor capacitivo de humedad de suelo.:

- Mejorar las lecturas: Promediar varias lecturas antes de reportar el valor final.

```
muestras = 5;
int promedioHumedad(int muestras) {
    int suma = 0;
    for (int i = 0; i < muestras; i++) {
```

```

        suma += analogRead(A0);
        delay(10);
    }
    return suma / muestras;
}

void loop() {
    int humedad = ...
    ...promedioHumedad(muestras);
    Serial.print("Humedad filtrada: ");
    Serial.println(humedad);
}

```

- Mejorar calibración: Hacer mediciones antes con el sensor en suelo seco y húmedo y ajusta los valores en map().

```

const int seco = 800; // SECO
const int humedo = 300; // MOJADO

void loop() {
    int humedad = analogRead(A0);
    int porcentaje = ...
    ...map(humedad, humedo, seco, 100, 0);

    porcentaje = ...
    ...constrain(porcentaje, 0, 100);
    Serial.print("Humedad (%): ");
    Serial.println(porcentaje);
}

```

- Mejorar tiempos de lecturas: Usar un promedio exponencial para hacer la transición más rápida y permitir responder rápido a cambios sin hacer lecturas erráticas para esto varia el ajuste en los rangos de 0.05 a 0.5.

```

float humedadSuavizada = 0;
const float factor = 0.2; // AJUSTE

void loop() {
    int nuevaLectura = analogRead(A0);
    humedadSuavizada = ...
    ... (factor * nuevaLectura) + ...
    ... ((1 - factor) * humedadSuavizada);

    Serial.print("Humedad: ");
    Serial.println(humedadSuavizada);
}

```

En el caso del sensor externo de detección de movimiento infrarrojo PIR fue el que mas reportó errores tales como los que se mencionan a continuación.

2) Sensor detector de proximidad infrarrojo PIR.:

- Movimiento constante: El sensor detecta movimiento cuando no hay nada debido a interferencias electromagnéticas o ruido ambiental. Una solución es usar millis() para asegurarse de que el movimiento es constante antes de reportarlo.

```

const int pinPIR = A6;
unsigned long tiempoDeteccion = 0;
const int tiempoConfirmacion = 500;

void setup() {
    pinMode(pinPIR, INPUT);
}

```

```

    Serial.begin(115200);
}

void loop() {
    if (digitalRead(pinPIR) == HIGH) {
        if (millis() - ...
            ...tiempoDeteccion > ...
            ...tiempoConfirmacion) {
            Serial.println("MOVIMIENTO");
            tiempoDeteccion = millis();
        }
    }
}

```

- Movimiento inestable: A pesar de que ya se detectó movimiento este puede cambiar su lectura y mencionar que no hay movimiento debido a que las lecturas son intermitentes para este caso podemos eliminar los parpadeos y repeticiones innecesarias y confirmar solo la lectura correcta de movimiento.

```

bool movimientoDetectado = false;

void loop() {
    int estado = digitalRead(pinPIR);

    if (estado == ...
        ...HIGH && !movimientoDetectado) {
        Serial.println("MOVIMIENTO");
        movimientoDetectado = true;
    }

    if (estado == LOW) {
        movimientoDetectado = false;
    }
}

```

Estos fueron los errores que se tuvieron a lo largo de las prácticas sin embargo también podemos mencionar una mejora para tener en consideración.

- 1) No colocar el sensor cerca de bombillas incandescentes o radiadores.
- 2) Evitar que el sol le pegue directamente.

V. CONCLUSIONES.

La experiencia adquirida a lo largo de las cinco prácticas confirma la versatilidad del kit Arduino MKR WiFi 1010 en conjunto con el Carrier MKR IoT Rev2 y los sensores externos utilizados. Desde la creación de un juego con LEDs hasta la monitorización de variables ambientales, la detección de gestos y colores, la conexión inalámbrica mediante Wi-Fi y BLE, así como la medición de humedad del suelo y la detección de movimiento con un sensor PIR se demostró la facilidad de integración y el alcance de las herramientas proporcionadas por este kit.

Cada práctica permitió explorar de manera progresiva distintas funcionalidades, mostrando cómo combinar hardware y software para desarrollar aplicaciones IoT e interactivas. Además, la documentación y los ejemplos facilitan la comprensión de cada uno de los componentes, lo cual es valioso para proyectos educativos.

Para finalizar este trabajo evidencia que el kit Arduino MKR WiFi 1010 y el Carrier MKR IoT Rev2 ofrecen una plataforma sólida para la creación de soluciones conectadas, interactivas y personalizables. Ademas en el apartado de referencias podrás encontrar la documentación ocupada y el repositorio de GitHub en donde encontraras los códigos completos de las prácticas [2].

REFERENCES

- [1] Arduino. Mkr iot carrier rev2 - cheat sheet. <https://docs.arduino.cc/tutorials/mkr-iot-carrier-rev2/cheat-sheet/>, 2025. [Último acceso: 28 de enero de 2025].
- [2] Nombre del Autor o Fuente. Título del recurso. <https://github.com/JorgeMarinN/TallerArduinoIoTAC3E> – USM, 2025. [Último acceso : 03defebrode2025].
- [3] MCI Electronics. Sensor pir hc-sr501 - detector de movimiento. <https://mcielectronics.cl/shop/product/sensor-pir-hc-sr501-detector-de-movimiento-28907/>, 2025. [Último acceso: 01 de febrero de 2025].
- [4] Naylamp Mechatronics. Sensor de humedad de suelo capacitivo v1. <https://naylampmechatronics.com/sensores-temperatura-y-humedad/538-sensor-de-humedad-de-suelo-capacitivo-v1.html>, 2025. [Último acceso: 31 de enero de 2025].