



Enseñar la explotación de la tierra,
no la del hombre



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Aprende y Crea: Kit Arduino Explore IoT Rev2

Jiménez Hernández Cecilia Margarita
Enero 2025



Resumen—Este documento presenta una serie de actividades diseñadas para explorar las capacidades del Arduino MKR IoT Carrier, desde el manejo básico de LEDs y sensores hasta aplicaciones más avanzadas como la detección de gestos, colores y proximidad, además de la integración con Wi-Fi, Bluetooth y sensores externos. A lo largo de estas actividades, se trabajó con diferentes estructuras de programación, manejo de displays interactivos, uso de botones capacitivos y optimización de código mediante máquinas de estados.

Cada sección incluyó no solo la implementación del código, sino también una explicación detallada de su funcionamiento, errores comunes y sus respectivas soluciones, permitiendo un aprendizaje progresivo y aplicable a futuros proyectos.

I. INTRODUCCIÓN

El Arduino MKR WiFi 1010 es una placa de desarrollo que facilita la implementación de proyectos de Internet de las Cosas (IoT), permitiendo la comunicación inalámbrica mediante WiFi y la integración con múltiples sensores. Su uso junto con el MKR IoT Carrier simplifica el desarrollo de aplicaciones interactivas, gracias a su pantalla TFT integrada, botones táctiles, sensores ambientales y capacidad de control de dispositivos externos.

Este documento tiene como objetivo servir como una guía en la exploración de las capacidades del Arduino MKR WiFi 1010, junto con su complemento, el MKR IoT Carrier, a través de una serie de actividades diseñadas para fomentar un aprendizaje progresivo y basado en la experiencia práctica. A lo largo de este material, compartiré mi experiencia al trabajar con este kit, mostrando no solo los conceptos y aplicaciones fundamentales, sino también los errores que encontré en el proceso, las soluciones que implementé y mi manera de abordar cada desafío.

El contenido abarca desde conceptos básicos, como la interacción con LEDs y sensores, hasta aplicaciones más avanzadas, como la detección de gestos, colores y proximidad. En cada sección, además de proporcionar explicaciones detalladas y ejemplos de código, incluiré reflexiones sobre las dificultades con las que me encontré, los ajustes que realicé y las soluciones que encontré para lograr que cada proyecto funcionara correctamente.

II. ESPECIFICACIONES TÉCNICAS

El kit incluye los siguientes componentes:

- Arduino MKR WiFi 1010.
- Sensor PIR.
- Cable micro USB.
- Sensor de humedad.
- Cables plug-and-play para los sensores externos.
- Carcasa de plástico para fijar y proteger el hardware.
- Arduino MKR IoT Carrier Rev2, que incorpora:
 - Dos relés de 24V.
 - Soporte para tarjeta SD.
 - Cinco botones táctiles capacitivos.
 - Conectores Grove plug-and-play.
 - Sensor de temperatura.
 - Sensor de humedad.
 - Sensor de presión.

- Sensor de gas (COV).
- Sensor de luz ambiental.
- Sensor de color RGB.
- Sensor de gestos.
- Acelerómetro.
- Pantalla redondeada RGB de 1,20".
- Soporte para batería recargable 18650 Li-Ion.
- Cinco LEDs RGB.
- Zumbador (buzzer).

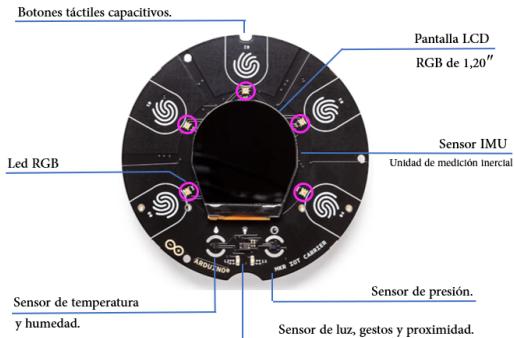


Figura 1: Vista frontal del MKR IoT Carrier.

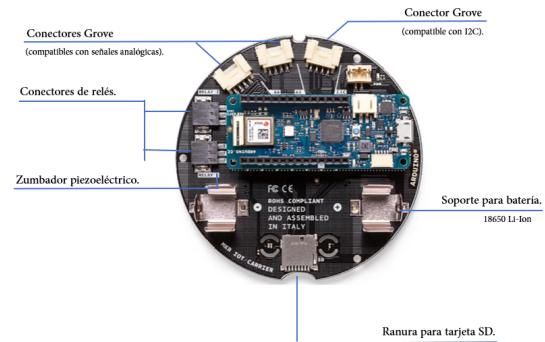


Figura 2: Vista trasera del MKR IoT Carrier.

III. ACTIVIDADES

III-A. Juego LEDs

Propósito: Implementar un sistema interactivo para sincronizar los LEDs del MKR IoT Carrier en un mismo color y acumular puntos.

Estados de los LEDs:

- **Apagado:** El LED cambia de color internamente en intervalos de 10 ms de manera cíclica o pseudo-aleatoria.
- **Parpadeando:** Alterna entre encendido y apagado con una frecuencia perceptible al ojo humano.
- **Encendido:** Se fija en el color seleccionado hasta que el juego se reinicie.

Reglas del Juego:

- Un punto se obtiene al sincronizar todos los LEDs en un mismo color.
- Cada vez que se gana un punto, el buzzer emite un sonido calibrado en frecuencia y duración.
- No se puede obtener puntos repetidos con la misma combinación de colores.
- Al alcanzar tres puntos (colores rojo, azul y verde), el display mostrará "YOU WIN!" el juego se reiniciará.

Consideraciones de diseño: - Implementar un sistema de anti-rebote para evitar múltiples pulsaciones involuntarias. - Ajustar los tiempos de transición y parpadeo para mejorar la experiencia de usuario.

Solución

Algorithm 1 Configuración del Juego de LEDs

```

1: Iniciar Arduino MKR IoT Carrier
2: Definir estados de LEDs: OFF, BLINKING, ON
3: Definir colores LED con intensidad reducida
4: Iniciar variables: puntaje = 0, juego_ganado = falso
5: procedure SETUP
6:   Si el Carrier no inicia correctamente, detener ejecución
7:   Apagar todos los LEDs y limpiar pantalla
8:   Mostrar mensaje: "INICIA EL JUEGO!"
9:   Esperar 2 segundos
10:  Establecer todos los LEDs en estado OFF
11: procedure LOOP
12:   while Ejecutando do
13:     Actualizar botones táctiles
14:     for Cada LED do
15:       if Botón del LED es presionado then
16:         Cambiar estado del LED (OFF →
17:           BLINKING → ON)
18:         if Estado == OFF then
19:           Cambiar color del LED
20:         end if
21:       end if
22:       Manejar estado LED:
23:       if Estado == OFF then
24:         Apagar LED
25:       else if Estado == BLINKING then
26:         Alternar entre encendido y apagado
27:       else if Estado == ON then
28:         Mantener LED encendido
29:       end if
30:     end for
31:     Mostrar cambios en los LEDs
32:     if Todos los LEDs tienen el mismo color y
33:       están encendidos then
34:       Incrementar puntaje
35:       Reproducir sonido de puntuación
36:       if Puntaje ≥ 3 then
37:         Mostrar "YOU WIN!"
```

```

1: while Ejecutando do
2:   Detectar botón presionado
3:   Cambiar estado del LED correspondiente
4:   if Todos los LEDs sincronizados then
5:     Incrementar puntaje
6:     if Puntaje ≥ 3 then
7:       Mostrar "YOU WIN!"
8:       Reproducir sonido de victoria
9:       Reiniciar juego
10:    else
11:      Mostrar puntaje actual
12:    end if
13:  end if
14: end while
15: procedure CHECKWINCONDITION
16:   Definir color objetivo como color del primer LED
17:   for Cada LED do
18:     if LED no está encendido o tiene color diferente
19:       return falso
20:     end if
21:   end for
22:   if Color ya fue puntuado then
23:     return falso
24:   end if
25:   Marcar color como puntuado
26:   return verdadero
27: end procedure
28: procedure PLAYPOINTBUZZER
29:   Emitir tono de 2000 Hz durante 300ms
30: end procedure
31: procedure PLAYWINBUZZER
32:   Emitir tono de 1000 Hz durante 500ms
33: end procedure
34: procedure RESETGAME
35:   Reiniciar puntaje a 0
36:   Restablecer estado de LEDs a OFF
37:   Restablecer colores de LEDs
38:   Mostrar mensaje "INICIA EL JUEGO!"
39: end procedure=0
```

Errores comunes y soluciones

En el desarrollo del juego de LEDs con el MKR IoT Carrier, pueden surgir diversos errores que afectan la interacción con los botones, la visualización de los colores y la lógica del juego. A continuación, se presentan algunos de los problemas más comunes junto con sus posibles soluciones.

1. Los LEDs no cambian de estado al tocar los botones

- **Possible causa:** No se está actualizando el estado de los botones táctiles correctamente en la función `loop()`.
- **Solución:** Verificar que la función `carrier.Buttons.update();` está presente en cada ciclo de `loop()`. Sin esta línea, los

botones táctiles no detectan los toques.

■ **Ejemplo correcto:**

```
void loop() {
    carrier.Buttons.update();
    // Debe llamarse en
    // cada ciclo de loop()
}
```

2. Los LEDs no muestran el color correcto

- **Possible causa:** El índice del color (`currentColor[i]`) no se actualiza correctamente al cambiar de estado.
- **Solución:** Asegurar que `currentColor[i]` solo cambia cuando el LED pasa a estado OFF.
- **Ejemplo correcto:**

```
if (ledStates[i] == OFF) {
    currentColor[i] =
        (currentColor[i] + 1) % 3;
}
```

3. El juego no detecta la condición de victoria

- **Possible causa:** La función `checkWinCondition()` no está validando correctamente los estados de los LEDs.
- **Solución:** Asegurar que `checkWinCondition()` verifica tanto el estado ON de los LEDs como el color.
- **Ejemplo correcto:**

```
bool checkWinCondition() {
    int targetColor =
        currentColor[0];
    for (int i = 0; i < 5;
        i++) {
        if (ledStates[i] != ON || currentColor[i]
            != targetColor) {
            return false;
        }
    }
    return true;
}
```

4. El sonido no se escucha al ganar un punto o al ganar el juego

- **Possible causa:** `carrier.Buzzer.sound(frecuencia);` no está llamándose con una frecuencia válida.
- **Solución:** Asegurar que la frecuencia está dentro del rango audible (ej. 1000 Hz o 2000 Hz).
- **Ejemplo correcto:**

```
void playPointBuzzer() {
    carrier.Buzzer.sound(2000);
    delay(300);
    carrier.Buzzer.noSound();
```

}

5. El juego no se reinicia correctamente después de ganar

- **Possible causa:** `resetGame()` no está reiniciando todas las variables necesarias.

- **Solución:** Asegurar que:

- `score = 0;` reinicia la puntuación.
- `colorMatched[]` se restablece para permitir nuevos colores.
- `ledStates[]` y `currentColor[]` vuelven a su estado inicial.

■ **Ejemplo correcto:**

```
void resetGame() {
    score = 0;
    for (int i = 0; i < 3; i++) {
        colorMatched[i] = false;
    }
    for (int i = 0; i < 5; i++) {
        ledStates[i] = OFF;
        currentColor[i] = i % 3;
    }
}
```

Conceptos Aprendidos en el Juego

- **Manipulación de LEDs RGB:** Control de colores y gestión de estados de los LEDs en el MKR IoT Carrier.
- **Interacción con Botones Táctiles:** Detección de toques y asignación de funciones específicas a cada botón.
- **Uso de Estructuras de Control:** Implementación de ciclos `for` y estructuras condicionales `if-else` para la lógica del juego.
- **Gestión de Estados:** Implementación de una máquina de estados mediante la enumeración:
`enum LedState { OFF, BLINKING, ON };`
- **Generación de Sonidos:** Uso del buzzer integrado en el MKR IoT Carrier para reproducir tonos al ganar puntos.
- **Uso de Arrays para Organización del Código:** Manejo de arreglos como `currentColor[]`, `ledStates[]` y `colorMatched[]` para estructurar la lógica del juego.

Resultados



Figura 3: Estado inicial del juego.



Figura 4: Cambio de estado en LEDs.



Figura 6: Mensaje de victoria.

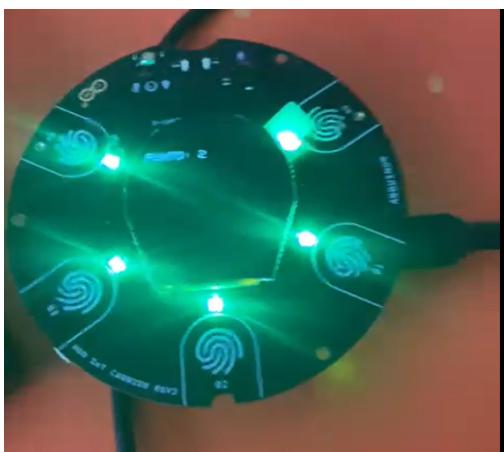


Figura 5: Sincronización de colores.

III-B. Monitor de Ambiente

Propósito: Implementar un sistema de monitoreo que permita visualizar en tiempo real las mediciones de distintos sensores ambientales del MKR IoT Carrier.

Estados del Monitor:

■ Menú Principal (DEFAULT):

- Permite seleccionar qué sensor visualizar.
- Se accede a cada sensor pulsando un botón táctil específico.

■ Submenús de Sensores:

Cada sensor cuenta con un menú que actualiza periódicamente sus mediciones.

Lecturas de Sensores:

■ 00 - Temperatura:

- Muestra la temperatura en grados Celsius ($^{\circ}\text{C}$), Fahrenheit ($^{\circ}\text{F}$) y Kelvin (K).
- Se actualiza automáticamente cada cierto intervalo de tiempo.

■ 01 - Humedad:

- Muestra la humedad relativa del aire en porcentaje (%).
- La lectura se actualiza en tiempo real.

■ 02 - Presión Atmosférica:

- Indica la presión ambiental en hectopascales (hPa).

■ 03 - Calidad del Aire (Gases):

- Resistencia de gas en ohms.
- Concentración de Compuestos Orgánicos Volátiles (VOC).
- Nivel de CO en partes por millón (ppm).

■ 04 - Giroscopio:

- Muestra la orientación en los ejes X, Y y Z.
- Indica la inclinación y el movimiento del dispositivo.

Reglas de Navegación:

- El usuario puede navegar entre sensores mediante los botones táctiles.
- Si se encuentra en un submenú y presiona cualquier botón, regresará al menú principal.

Consideraciones de Diseño:

- Uso de la función `millis()` para actualizar los valores en tiempo real sin bloquear la ejecución del código.
- Manejo de eventos con botones táctiles para mejorar la experiencia del usuario.

Solución

Algorithm 2 Inicialización del Monitor de Ambiente

```
1: Iniciar Arduino MKR IoT Carrier
2: Definir estados del sistema: MENÚ, SUBMENÚ
3: Iniciar variables:
4:   submenú_activo = false, índice_submenú = -1
5: procedure SETUP
6:   if Carrier no inicia correctamente then
7:     Detener ejecución
```

```
1: Limpiar pantalla y mostrar menú principal
2: if Módulo IMU no inicia correctamente then
3:   Detener ejecución
4: end if
5:
6: procedure LOOP
7:   while Ejecutando do
8:     Actualizar botones táctiles
9:     for Cada Botón i en {0, 1, 2, 3, 4} do
10:      if Botón es presionado then
11:        if Es la primera vez then
12:          Guardar tiempo de presión
13:        else if Tiempo de presión  $\geq$  500 ms then
14:          Reiniciar tiempo del botón
15:        if Está en submenú y botón diferente al
16:          actual then
17:            Regresar al menú principal
18:          else
19:            Acceder al submenú correspondien-
              te
20:          end if
21:        end if
22:      end if
23:    end for
24:    if Está en un submenú y han pasado 2 segundos
      then
25:      Actualizar datos del sensor activo
26:    end if
27:  end while
28: end procedure
29: procedure MOSTRAR_MENU
30:   Limpiar pantalla y cambiar fondo a amarillo
31:   Mostrar título "MENÚ."
32:   Mostrar opciones:
33:     ■ 00 - Temperatura (Naranja)
34:     ■ 01 - Humedad (Negro)
35:     ■ 02 - Presión (Magenta)
36:     ■ 03 - Gases (Azul)
37:     ■ 04 - Giroscopio (Rojo)
38:
39: end procedure
40: procedure MANEJAR_SUBMENU(botón)
41:   if botón == 0 then
42:     Mostrar submenú de Temperatura
43:   else if botón == 1 then
44:     Mostrar submenú de Humedad
45:   else if botón == 2 then
46:     Mostrar submenú de Presión
47:   else if botón == 3 then
48:     Mostrar submenú de Gases
49:   else if botón == 4 then
50:     Mostrar submenú de Giroscopio
51:   end if
52:   Activar estado de submenú
53: end procedure=0
```

```

1: procedure ACTUALIZAR_SENSOR
2:   if Sensor == Temperatura then
3:     Leer temperatura en °C, °F y K
4:   else if Sensor == Humedad then
5:     Leer humedad relativa (%)
6:   else if Sensor == Presión then
7:     Leer presión (hPa)
8:   else if Sensor == Gases then
9:     Leer Resistencia, VOC, CO2
10:  else if Sensor == Giroscopio then
11:    Leer valores X, Y, Z
12:  end if
13:  Mostrar valores en pantalla
14: end procedure

```

Errores Comunes y Soluciones

1. La pantalla no muestra nada

- **Possible causa:** El carrier.begin() no se ejecutó correctamente.
- **Solución:** Verificar la inicialización del Carrier en la función setup().
- **Ejemplo correcto:**

```

void loop() {
  carrier.Buttons.update();
  // Debe llamarse en
  // cada ciclo de loop()
}

```

2. Los botones táctiles no responden

- **Possible causa:** No se está llamando carrier.Buttons.update(); en loop().
- **Solución:** Asegurar que la actualización de los botones táctiles está presente en cada ciclo de ejecución.

```

// Solución correcta
void loop() {
  carrier.Buttons.update();
  // Actualiza los botones táctiles
}

```

3. Los sensores no actualizan los valores

- **Possible causa:** La función millis() no está gestionando bien la actualización periódica.
- **Solución:** Usar correctamente el temporizador para actualizar cada 2 segundos.

```

// Solución correcta
if (millis() - lastUpdate >= 2000) {
  updateTemperatureDisplay();
  lastUpdate = millis();
}

```

4. El giroscopio siempre muestra 0 en X, Y y Z

- **Possible causa:** El módulo IMU no se ha inicializado correctamente.
- **Solución:** Asegurar la inicialización en setup().

```

// Solución correcta
if (!carrier.IMUmodule.begin()) {
  while (1); // Detener el programa
  si la IMU no funciona
}

```

Resultados



Figura 7: Menú principal



Figura 8: Datos de temperatura



Figura 9: Datos de humedad



Figura 12: Funcionamiento del giroscopio

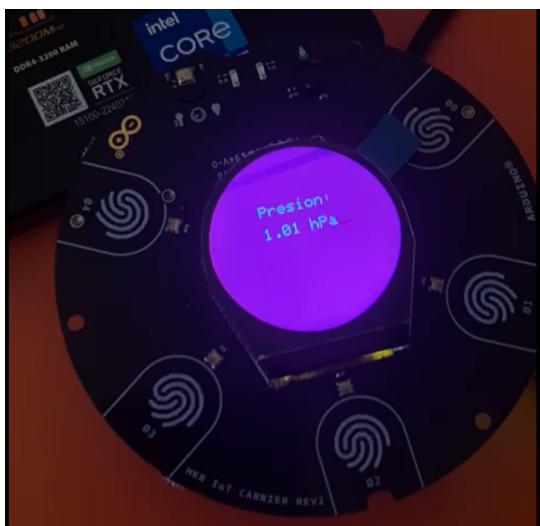


Figura 10: Datos de presión



Figura 11: Toma de datos de los gases

Propósito: Implementar un sistema interactivo que utilice los sensores de gestos, color y proximidad del *MKR IoT Carrier*, proporcionando retroalimentación visual y sonora.

Estados del Sistema:

- **Estado Base:** Muestra en pantalla "*LEYENDO MOVIMIENTO*". Permite detectar gestos o cambiar a detección de color (TOUCH2) o proximidad (TOUCH4).
- **Detección de Gestos:** Captura movimientos en cuatro direcciones (arriba, abajo, izquierda, derecha), muestra flechas en pantalla y emite un zumbido breve.
- **Detección de Color:** Solicita al usuario mostrar color blanco y detecta si los valores de R, G y B están en un rango alto y similar.
- **Detección de Proximidad:** Mide la distancia en mm y usa LEDs para indicar la cercanía del objeto:
 - **Verde:** Mayor a 200 mm.
 - **Amarillo:** Entre 120 y 200 mm.
 - **Rojo:** Menor a 120 mm.

Reglas del Sistema:

- El sistema vuelve al estado base tras 1 segundo en detección de gestos.
- La detección de color finaliza si se detecta blanco o si el usuario presiona TOUCH2 nuevamente.
- La detección de proximidad finaliza al presionar TOUCH4.

Consideraciones:

- **Anti-rebote:** Implementar filtros para evitar lecturas erróneas en los botones táctiles.
- **Buzzer:** Emitir zumbidos al detectar gestos y en condiciones críticas (ejemplo: proximidad extrema).
- **Código Modular:** Organizar el código en una máquina de estados para facilitar su mantenimiento.
- **Sensibilidad:** Calibrar los sensores para obtener mediciones más precisas.
- **Lectura Continua:** Los sensores deben realizar lecturas constantes mientras el sistema esté en el estado correspondiente.

Solución

Algorithm 3 Configuración del Sistema de Detección

```

1: Iniciar Arduino MKR IoT Carrier
2: Definir estados del sistema: BASE, GESTOS, COLOR, PROXIMIDAD
3: Iniciar sensores: APDS9960 para gestos, color y proximidad
4: Iniciar variables: estado_actual = BASE
5: procedure SETUP
6:   if Carrier no inicia correctamente then
7:     Mostrar error en pantalla
8:     Detener ejecución
9:   end if
10:  if APDS9960 no inicia correctamente then
11:    Mostrar error en pantalla
12:    Detener ejecución
13:  end if
14:  Mostrar menú inicial en pantalla
15: end procedure
16: procedure LOOP
17:   Actualizar botones táctiles
18:   if Botón TOUCH1 presionado then
19:     Llamar detectarGestos()
20:   else if Botón TOUCH2 presionado then
21:     Llamar detectarColor()
22:   else if Botón TOUCH4 presionado then
23:     Llamar detectarProximidad()
24:   else if Botón TOUCH3 presionado then
25:     Llamar mostrarMenuPrincipal()
26:   end if
27: end procedure
28: procedure DETECTARGESTOS
29:   Cambiar estado a GESTOS
30:   Mostrar en pantalla: "GESTOS"
31:   while true do
32:     Actualizar botones táctiles
33:     if Botón TOUCH3 presionado then
34:       Regresar al menú principal
35:       Salir del bucle
36:     end if
37:     if Gesto detectado por APDS9960 then
38:       Mostrar flecha en pantalla según dirección
39:       Emitir sonido en buzzer
40:       Esperar 1 segundo
41:       Mostrar "GESTOS" nuevamente
42:     end if
43:   end while
44: end procedure
45: procedure DETECTARCOLOR
46:   Cambiar estado a COLOR
47:   Mostrar "MUESTRE COLOR BLANCO"
48:   while true do
49:     Actualizar botones táctiles
50:     if Botón TOUCH3 presionado then
51:       Regresar al menú principal
52:       Salir del bucle
53:     end if

```

```

1: if Lectura de sensor RGB disponible then
2:   if Valores de R, G, y B son altos y similares then
3:     Mostrar "COLOR BLANCO DETECTADO"
4:     Esperar 1 segundo
5:     Regresar al menú principal
6:   end if
7: end if
8:
9:
10: procedure DETECTARPROXIMIDAD
11:   Cambiar estado a PROXIMIDAD
12:   Mostrar "MIDENDO PROXIMIDAD"
13:   while true do
14:     Actualizar botones táctiles
15:     if Botón TOUCH3 presionado then
16:       Regresar al menú principal
17:       Salir del bucle
18:     end if
19:     if Lectura de proximidad disponible then
20:       Mostrar distancia en mm
21:       if Distancia >= 200 mm then
22:         Encender LED Verde
23:       else if 120 mm ≤ Distancia ≤ 200 mm then
24:         Encender LED Amarillo
25:       else
26:         Encender LED Rojo
27:         Emitir zumbido de advertencia
28:       end if
29:     end if
30:   end while
31: end procedure
32: procedure MOSTRARMENUPRINCIPAL
33:   Cambiar estado a BASE
34:   Limpiar pantalla
35:   Mostrar opciones del menú:
36:   Opción 01: Gestos
37:   Opción 02: Color
38:   Opción 04: Proximidad
39:   Opción 03: Regresar
40: end procedure=0

```

Errores Comunes y Soluciones

1. El sensor no detecta gestos

- **Possible causa:** El sensor APDS9960 no está inicializado correctamente o su sensibilidad no es suficiente para reconocer gestos.
- **Solución:** Verificar que la función APDS.begin() se ejecuta correctamente en la función setup() y ajustar la sensibilidad del sensor si es necesario.
- **Ejemplo correcto:**

```

void setup() {
  if (!APDS.begin()) {
    Serial.println("Error

```

```

al iniciar el sensor
APDS9960");
while (true);
// Detener ejecución
si falla
}
}
```

2. El sensor de color no detecta correctamente el color blanco

- **Possible causa:** Los valores RGB no se comparan correctamente o los umbrales para detectar el color blanco no son adecuados.
- **Solución:** Asegurar que los valores de R, G y B son altos y tienen diferencias mínimas entre ellos para confirmar que se trata de un color blanco.
- **Ejemplo correcto:**

```

if (r > 200 && g > 200 &&
b > 200 &&
abs(r - g) < 50 &&
abs(g - b) < 50 &&
abs(r - b) < 50) {
Serial.println("Color
Blanco Detectado");
}
```

3. El sensor de proximidad no responde

- **Possible causa:** La función de lectura del sensor de proximidad no está siendo llamada correctamente en el loop().
- **Solución:** Asegurar que la función APDS.proximityAvailable() verifica la disponibilidad de datos antes de leer la proximidad.
- **Ejemplo correcto:**

```

if (APDS.proximityAvailable
()) {
int proximity = APDS.
readProximity();
Serial.print
("Proximidad: ");
Serial.println
(proximity);
}
```

4. Los LEDs no cambian de color según la proximidad

- **Possible causa:** No se están comparando correctamente los valores de proximidad para asignar los colores de los LEDs.
- **Solución:** Usar comparaciones adecuadas para determinar si se debe encender el LED rojo, amarillo o verde según la distancia detectada.
- **Ejemplo correcto:**

```

if (proximity > 200) {
carrier.leds.fill
(carrier.leds.Color

```

```

        (0, 255, 0)); // Verde
    } else if
(proximity > 120) {
    carrier.leds.fill
(carrier.leds.
Color(255, 255, 0));
// Amarillo
} else {
    carrier.leds.fill
(carrier.leds.
Color(255, 0, 0));
// Rojo
}

```

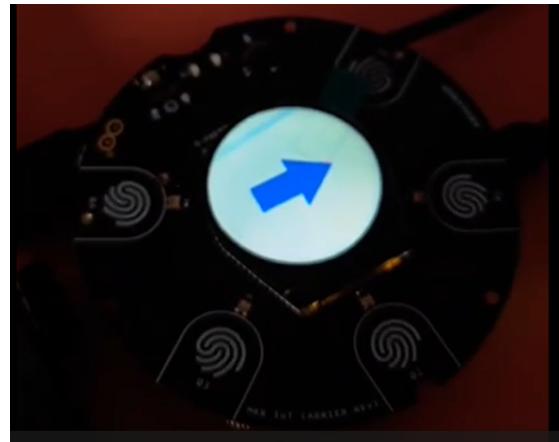


Figura 14: Gesto 1 detectado.

5. El buzzer no emite sonidos al detectar eventos

- **Possible causa:** No se está llamando correctamente la función de activación del buzzer al detectar gestos o proximidad extrema.
- **Solución:** Verificar que la función de sonido está siendo invocada dentro de las condiciones correctas.
- **Ejemplo correcto:**

```

if (proximity < 120) {
    carrier.Buzzer.sound(1000);
// Sonido de alerta
delay(300);
carrier.Buzzer.noSound();
}

```

Resultados



Figura 15: Gesto 2 detectado



Figura 13: Menú principal del juego.



Figura 16: Estado de proximidad.



Figura 17: Estado de alerta para proximidad.



Figura 18: Detección de color con ayuda de dispositivo electrónico.

III-D. Conexión Wi-Fi y Bluetooth Low Energy (BLE)

Propósito: Implementar un sistema que permita la conexión a redes Wi-Fi y dispositivos Bluetooth Low Energy (BLE) mediante el *MKR IoT Carrier*.

Menú Inicial

- El programa debe mostrar un menú con dos opciones:
 - **Conexión Wi-Fi.**
 - **Bluetooth Low Energy (BLE).**

Conexión Wi-Fi

- Escanear redes Wi-Fi disponibles en la banda de 2.4 GHz.
- Mostrar las redes encontradas con la siguiente información:
 - **SSID:** Nombre de la red.
 - **RSSI:** Intensidad de la señal.
 - **Canal:** Frecuencia de transmisión.
- Permitir seleccionar una red Wi-Fi o regresar al menú con la tecla '**m**'.
- Solicitar la contraseña de la red seleccionada.
- Intentar la conexión Wi-Fi y:
 - Mostrar la dirección IP si la conexión es exitosa.
 - Regresar al menú principal si la conexión falla.
- Permitir desconectar el dispositivo ingresando '**m**'.

Bluetooth Low Energy (BLE)

- Activar el módulo BLE y publicar un servicio genérico con una característica de lectura y escritura.
- Indicar en el monitor serial que el dispositivo está disponible para conexiones BLE.
- Detectar y mostrar información del dispositivo central conectado.
- Permitir regresar al menú principal ingresando '**m**', cerrando la conexión BLE.
- Para pruebas, se recomienda utilizar una aplicación de conexión BLE.

Interacción con el Usuario

- Proporcionar retroalimentación clara y concisa a través del monitor serial.
- Permitir regresar al menú principal desde cualquier estado con la tecla '**m**'.

Estabilidad y Robustez

- Implementar validaciones para entradas inválidas del usuario.
- Prevenir bucles infinitos en estados de espera mediante controles adecuados.

Solución

Algorithm 4 Configuración BLE Y WI-FI

- 1: **Iniciar** Arduino MKR IoT Carrier
 - 2: **Iniciar** módulo Wi-Fi
 - 3: **Iniciar** módulo BLE
-

1: **Llenar pantalla** con color azul
2: **Establecer texto en color blanco**
3: **Mostrar título** "MENU"
4: **Mostrar opciones:**
5: Wi-Fi: 01
6: BLE: 04
7: **Mostrar en monitor serial:**
8: "1) Escanear Wi-Fi"
9: "4) Escanear BLE"
10: .^Escribe 'm' para volver al menú"
11: **Llenar pantalla** con color naranja
12: **Mostrar mensaje** ".^ECANEANDO"
13: **Escanear redes Wi-Fi disponibles** (máximo 6)
14: **if** No se detectan redes **then**
15: **Mostrar mensaje** ÍNEXISTENTE"
16: **Esperar** 2 segundos
17: **Volver al menú principal**
18: **else**
19: **for** cada red detectada **do**
20: **Almacenar** SSID, RSSI y canal
21: **end for**
22: **Mostrar en el monitor serial** las redes disponibles
23: **Solicitar selección** de una red (1-6) o 'm' para volver
 al menú
24: **end if**
25: **Solicitar clave de acceso**
26: **Intentar conexión Wi-Fi**
27: **for** máximo 6 intentos **do**
28: **Esperar** 1 segundo
29: **Imprimir punto en monitor serial**
30: **end for**
31: **if** Conexión exitosa **then**
32: **Mostrar mensaje** CONECTADO"
33: **Mostrar dirección IP en pantalla y monitor serial**
34: **else**
35: **Mostrar mensaje** ".^EROR: No se pudo conectar"
36: **Esperar** 2 segundos
37: **Regresar al menú principal**
38: **end if**
39: **Activar módulo BLE**
40: **Publicar servicio genérico BLE**
41: **Mostrar mensaje** ".^Eperando conexión BLE..."
42: **while** esperando conexión BLE **do**
43: **if** Dispositivo BLE detectado **then**
44: **Mostrar mensaje** "Conectado a [dirección]"
45: **Actualizar pantalla** a estado ".^ENLAZADO"
46: **while** Dispositivo BLE sigue conectado **do**
47: **if** Usuario ingresa 'X' **then**
48: **Cerrar conexión BLE**
49: **Regresar al menú principal**
50: **end if**
51: **end while**
52: **Mostrar mensaje** "DESVINCULADO"

-
- 1: Esperar 2 segundos
 - 2: Regresar al menú principal
 - 3:
 - 4: =0
-

Errores Comunes y Soluciones

1. No se muestran redes Wi-Fi detectadas

- **Possible causa:** No se están escaneando correctamente las redes Wi-Fi.
- **Solución:** Verificar que la función WiFi.scanNetworks() se ejecuta antes de intentar mostrar las redes disponibles.
- **Ejemplo correcto:**

```
int networkCount = WiFi.scanNetworks();
if (networkCount == 0) {
    Serial.println("No se encontraron redes.");
}
```

2. El dispositivo no se conecta a Wi-Fi

- **Possible causa:** La clave ingresada es incorrecta o el SSID no está disponible.
- **Solución:** Implementar un sistema de reintentos y validar la contraseña ingresada.
- **Ejemplo correcto:**

```
WiFi.begin(ssid, password);
int attempts = 0;
while (WiFi.status() != WL_CONNECTED
&& attempts < 6) {
    delay(1000);
    attempts++;
}
```

3. El monitor serial no responde a los comandos de entrada

- **Possible causa:** El programa no está verificando si hay datos disponibles en el serial antes de leerlos.
- **Solución:** Asegurar que Serial.available() es mayor que 0 antes de leer la entrada del usuario.
- **Ejemplo correcto:**

```
if (Serial.available() > 0) {
String input = Serial.
readStringUntil('\n');
}
```

4. El dispositivo BLE no es detectado por otras aplicaciones

- **Possible causa:** El servicio BLE no está siendo correctamente anunciado.
- **Solución:** Verificar que se llama a BLE.advertise() después de agregar el servicio BLE.

■ Ejemplo correcto:

```
BLE.setLocalName("CARRIER");
BLE.addService(genericService);
BLE.advertise();
```

5. El dispositivo BLE se desconecta inmediatamente después de conectar

- **Possible causa:** No se mantiene activo el loop de espera para la conexión BLE.
- **Solución:** Verificar que se mantiene la conexión mientras el dispositivo esté enlazado.
- **Ejemplo correcto:**

```
while (central.connected()) {
    // Ejecutar lógica BLE aquí
}
```

Resultados



Figura 19: Menú principal para wi-fi y BLE

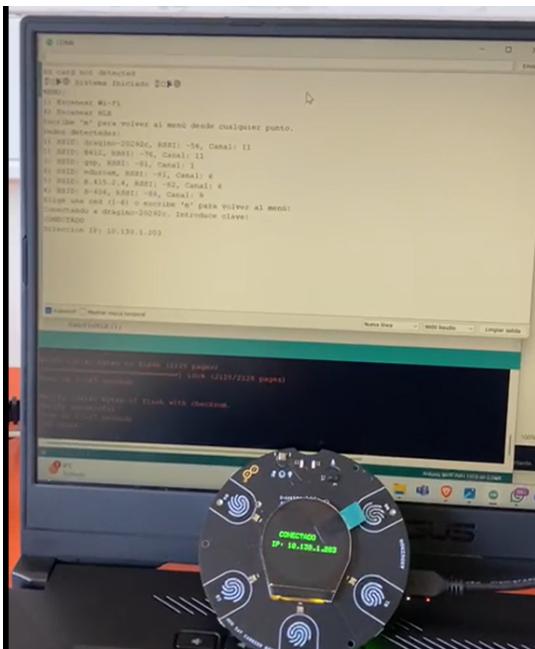


Figura 20: Conexión a red wi-fi.

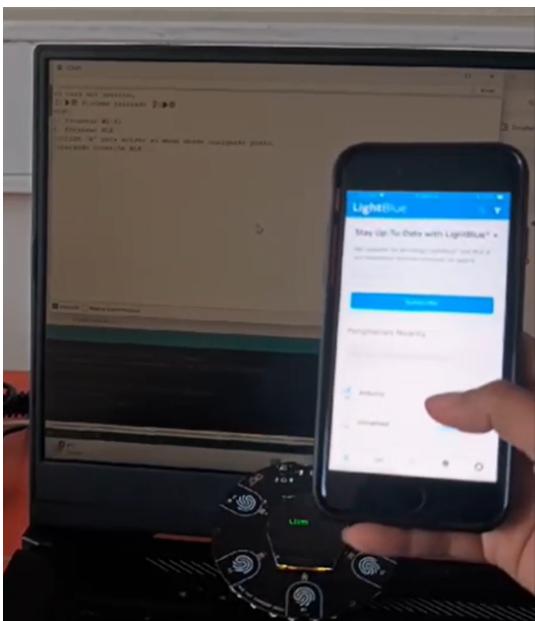


Figura 21: Conexión a BLE con ayuda de dispositivo móvil

III-E. Sensores Externos

Propósito: Implementar la lectura de sensores externos conectados al *MKR IoT Carrier*, específicamente un sensor de humedad del suelo y un sensor PIR, proporcionando información clara en el monitor serial.

Hardware Utilizado

- **Sensor de humedad del suelo:** Conectado al pin analógico **A0**.
- **Sensor PIR:** Conectado a un pin digital.

Funcionamiento General

- Se mostrará un menú principal en el monitor serial donde el usuario podrá seleccionar entre:
 - **Sensor de Humedad del Suelo.**
 - **Sensor PIR.**
- Se permite regresar al menú enviando '**r**' a través del teclado serial.

Modo Sensor de Humedad del Suelo

- **Calibración Inicial:** Se debe colocar el sensor en suelo seco y húmedo para registrar los valores de referencia.
- **Realiza lecturas cada 10 segundos.**
- El monitor serial indicará si el suelo está **seco** o **húmedo** con un porcentaje de humedad (0 % a 100 %).

Modo Sensor PIR

- Detecta movimiento en su rango de cobertura.
- **Realiza lecturas cada 2 segundos.**
- Muestra en el monitor serial si hay **movimiento** o si no se ha detectado actividad.

Control y Navegación

- Se selecciona el sensor con las siguientes opciones:
 - '1': Activa el Sensor de Humedad del Suelo.
 - '2': Activa el Sensor PIR.
- Para regresar al menú principal, se debe ingresar '**r**' en el monitor serial.

Requisitos Adicionales

- El monitor serial debe estar configurado a una velocidad de **9600 baudios**.
- Se debe verificar la correcta conexión de los sensores antes de ejecutar el código.

Calibración del Sensor de Humedad

- Al ingresar al modo de humedad, el sistema permitirá al usuario registrar valores de referencia para suelo seco y suelo húmedo manualmente.

Frecuencia de Muestreo

- **Sensor de Humedad:** Cada **10 segundos**.
- **Sensor PIR:** Cada **2 segundos**.

Plataforma de Desarrollo

- Se utilizará el **Arduino IDE** con soporte para la placa **Arduino IoT Explore Kit Rev2**.

Formato de Datos

- Los resultados se mostrarán en el monitor serial con una estructura clara:
 - **Menú principal con opciones numéricas.**

- **Resultados de lectura de sensores con etiquetas descriptivas.**
- **Indicaciones de estado para el usuario (por ejemplo, "Sensor activo", "Esperando lectura", etc.).**

Solución

Algorithm 5 Configuración de Sensores Externos

Inicio: Configurar el MKR IoT Carrier y sensores de humedad y PIR.

- 1: **Iniciarizar** MKR IoT Carrier
- 2: **Definir** pines de sensores:
- 3: Sensor de Humedad en **A0**
- 4: Sensor PIR en **A6**
- 5: **Iniciarizar** variables de estado
- 6: **Establecer** tasas de muestreo:
- 7: Humedad cada **10 segundos**
- 8: PIR cada **2 segundos**
- 9: **Configurar** monitor serial a 9600 baudios
- 10: **Mostrar** menú inicial
- 11: **Mostrar** opciones en el monitor serial:
- 12: **1:** Sensor de Humedad del Suelo
- 13: **2:** Sensor PIR
- 14: '**r**': Regresar al menú principal
- 15: **Esperar** entrada del usuario
- 16: **if** Usuario ingresa '**1**' **then**
- 17: Ir al modo Sensor de Humedad
- 18: **else if** Usuario ingresa '**2**' **then**
- 19: Ir al modo Sensor PIR
- 20: **else if** Usuario ingresa '**r**' **then**
- 21: Volver al menú principal
- 22: **end if**
- 23: **Calibración inicial:** Registrar valores de suelo seco y húmedo
- 24: **Iniciar** lecturas cada **10 segundos**
- 25: **loop**
- 26: Leer valor analógico del sensor de humedad
- 27: Mapear valor a porcentaje (**0 % a 100 %**)
- 28: **if** Humedad >=50% **then**
- 29: **Mostrar:** "Suelo Húmedo"
- 30: **else**
- 31: **Mostrar:** "Suelo Seco"
- 32: **end if**
- 33: **end loop**
- 34: **Iniciar** lecturas cada **2 segundos**
- 35: **loop**
- 36: Leer estado del sensor PIR
- 37: **if** Movimiento detectado **then**
- 38: **Mostrar:** "Hay movimiento"
- 39: **else**
- 40: **Mostrar:** "No hay movimiento"
- 41: **end if**
- 42: **end loop**
- 43: **if** Usuario presiona botón táctil en menú principal **then**

-
- 1: Cambiar estado a sensor correspondiente
 - 2:
 - 3: **if** Usuario presiona botón táctil en sensor activo **then**
 - 4: Regresar al menú principal
 - 5: **end if=0**
-

Errores Comunes y Posibles Soluciones

1. El sensor de humedad no muestra valores correctos

- **Possible causa:** El sensor no está calibrado correctamente.
- **Solución:** Realizar una calibración manual antes de comenzar las lecturas. Se deben registrar los valores de suelo seco y húmedo utilizando los comandos correspondientes ('c' para seco y 'w' para húmedo).

■ **Ejemplo correcto:**

- Ingresar 'c' cuando el sensor esté en suelo seco.
- Ingresar 'w' cuando el sensor esté en suelo húmedo.

2. El sensor PIR no detecta movimiento

- **Possible causa:** La sensibilidad del sensor puede estar configurada incorrectamente o la conexión de hardware es incorrecta.
- **Solución:**

- Revisar que el cableado esté correctamente conectado al pin A6.
- Verificar que el sensor no esté bloqueado por objetos externos.
- Asegurar que se está realizando la lectura cada 2 segundos.

3. El monitor serial no muestra información de los sensores

- **Possible causa:** La comunicación con el monitor serial no está configurada correctamente o la velocidad de baudios es incorrecta.
- **Solución:**

- Verificar que en el código se haya incluido la línea `Serial.begin(9600);` en la función `setup()`.
- Asegurar que el monitor serial en Arduino IDE está configurado a **9600 baudios**.

4. No se puede regresar al menú principal

- **Possible causa:** La detección del comando 'r' en el puerto serie no está funcionando correctamente.
- **Solución:**

- Asegurar que la función `handleSerialCommand()` está correctamente implementada y que el código está procesando la entrada del usuario en cada iteración de `loop()`.
- Verificar que la estructura condicional incluya el caso para el comando 'r'.

5. La pantalla no muestra los valores de los sensores

- **Possible causa:** No se está llamando correctamente a la función que actualiza el estado de la pantalla.

- **Solución:**

- Asegurar que la función `showState()` es llamada cada vez que cambia el estado del sistema.
- Confirmar que los datos de los sensores están siendo leídos y actualizados en pantalla.

Resultados

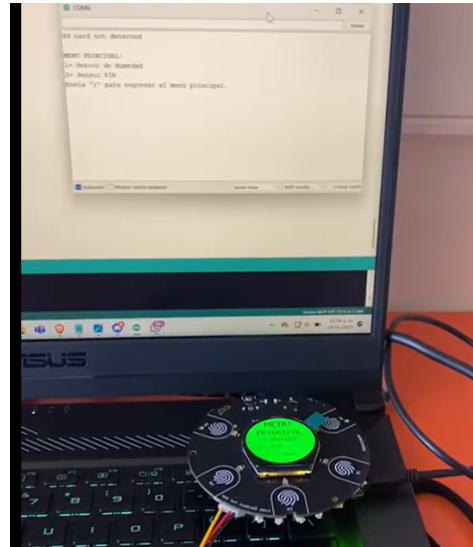


Figura 22: Menú principal para uso de sensores.



Figura 23: Demostración de sensor de humedad.



Figura 24: Demostración de sensor PIR.

IV. CONCLUSIÓN

Explorar el Arduino MKR IoT Carrier ha sido una experiencia enriquecedora, ya que no solo nos permitió entender su hardware y programación, sino que también nos obligó a resolver problemas reales a medida que avanzábamos en las actividades. Desde prender un LED hasta conectar el dispositivo a Wi-Fi y Bluetooth, cada paso fue un pequeño reto que nos llevó a mejorar habilidades en programación y electrónica.

Este documento es solo el inicio de lo que se puede lograr con el MKR IoT Carrier, ya que sus capacidades permiten crear desde simples sistemas de monitoreo hasta aplicaciones más complejas en el mundo del Internet de las Cosas (IoT).