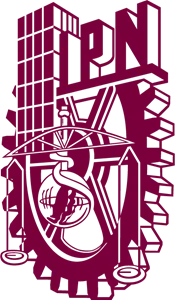
**Instituto Politécnico Nacional Escuela Superior de Cómputo**

Proyecto 1. Diseñar una aplicación móvil que permita, medir en forma remota la distancia numérica entre un imán de neodimio y un sensor de efecto Hall.

Profesor:

Dr. Alejandro Cifuentes A.

Integrantes

Cuevas Nava Dilan Mackdiel

Silvia

Mariela

Jiménez Velázquez José Bryan Omar

Grupo: 7CM4 14/10/2025

Introducción al Proyecto

En la actualidad, la convergencia entre los sistemas embebidos y las tecnologías móviles ha impulsado de manera significativa el desarrollo de aplicaciones orientadas a la Internet de las Cosas (IoT). Este paradigma tecnológico permite la interconexión de dispositivos físicos con plataformas digitales, promoviendo la automatización, el control remoto y la adquisición de datos en tiempo real. Dentro de este contexto, los microcontroladores de nueva generación, como el ESP32, han adquirido gran relevancia debido a su alta capacidad de procesamiento, bajo consumo energético e integración nativa de conectividad inalámbrica, particularmente Wi-Fi y Bluetooth.

El presente proyecto se enfoca en el diseño e implementación de un sistema básico de control remoto bidireccional entre un microcontrolador ESP32 y una aplicación Android. El objetivo principal es controlar el encendido y apagado de un LED mediante comunicación Bluetooth, utilizando una interfaz gráfica sencilla que permita al usuario interactuar de manera intuitiva con el hardware. De esta manera, se busca demostrar el funcionamiento práctico de una conexión inalámbrica confiable, en la cual los comandos enviados desde el dispositivo móvil sean interpretados y ejecutados por el microcontrolador en tiempo real.

A diferencia de las soluciones tradicionales basadas en Arduino UNO o Nano acompañadas de módulos externos como el HC-05 o HC-06, el ESP32 ofrece la ventaja de integrar la funcionalidad Bluetooth Clásico y Bluetooth Low Energy (BLE) directamente en el chip, simplificando la arquitectura del sistema y reduciendo los costos de implementación. Esta característica lo convierte en una plataforma idónea para proyectos educativos, de prototipado rápido o de desarrollo de dispositivos inteligentes. La aplicación Android desarrollada en este proyecto actúa como interfaz de usuario y centro de control del sistema. Desde ella se enviarán los comandos necesarios para modificar el estado del LED (encendido/apagado), al tiempo que recibirá información sobre su estado actual enviada por el ESP32. Esta comunicación bidireccional permite no solo emitir órdenes, sino también obtener retroalimentación del dispositivo físico, estableciendo así un canal de interacción en tiempo real entre el usuario y el entorno electrónico.

En términos educativos y tecnológicos, este proyecto representa una introducción práctica a los fundamentos de la comunicación inalámbrica, la programación de microcontroladores y el desarrollo de aplicaciones móviles nativas. Además, sienta las bases para la expansión hacia sistemas más complejos, como el monitoreo de sensores, el control de actuadores múltiples, o la integración con servicios en la nube dentro del ecosistema IoT.

En conclusión, el desarrollo de este sistema de control Bluetooth entre Android y ESP32 no solo constituye una experiencia de aprendizaje en el ámbito del hardware y software embebido, sino que también evidencia la versatilidad y potencia del ESP32 como herramienta central para el diseño de aplicaciones conectadas e inteligentes.

Objetivos Específicos

* Establecer la Conexión Bluetooth: Configurar una comunicación estable y confiable entre la aplicación Android y el ESP32, empleando el perfil Bluetooth Clásico o BLE según la implementación elegida.
* Control del LED (Android → ESP32): Desarrollar la lógica de la aplicación Android para enviar comandos específicos (‘1’ para encender, ‘0’ para apagar) al ESP32, y programar el firmware del ESP32 para interpretar dichos comandos y modificar el estado físico del LED.
* Retroalimentación de Estado (ESP32 → Android): Implementar en el ESP32 la capacidad de reportar cualquier cambio en el estado del LED hacia la aplicación Android, mostrando esta información en tiempo real dentro de la interfaz de usuario.
* Diseño de Interfaz Gráfica: Crear una interfaz intuitiva y funcional en Android (activity\_main.xml) con controles para encender y apagar el LED, además de indicadores visuales que reflejen su estado actual.

El proyecto "Desarrollo de Aplicaciones Móviles Nativas" integra la electrónica embebida con la interfaz móvil para crear un sistema de medición remota.

El ESP32 como Plataforma IoT

El ESP32 es el núcleo de este sistema. Su principal ventaja sobre microcontroladores clásicos como el Arduino Uno reside en su conectividad inalámbrica integrada. A diferencia del Arduino, que requiere módulos externos (como el HC-05 para Bluetooth o el ESP8266 para Wi-Fi), el ESP32 ofrece:

* SoC (System-on-a-Chip): Integra CPU dual-core, memoria, Wi-Fi (802.11 b/g/n) y Bluetooth (Clásico y BLE) en un solo chip, reduciendo el *hardware* y simplificando el desarrollo.
* ADC de 12 bits: El Conversor Analógico-Digital (ADC) del ESP32, con una resolución de 12 bits, proporciona niveles de lectura, lo que ofrece una mayor precisión en la lectura del sensor KY-024 comparado con los 10 bits ( niveles) de muchos modelos de Arduino.
* Gestión de Energía: El ESP32 puede manejar eficientemente tanto la comunicación Bluetooth como la adquisición de datos del sensor.

La Aplicación Android como HMI

El sistema operativo Android actúa como la Interfaz Hombre-Máquina (HMI). La aplicación, desarrollada en Java usando Android Studio, se encarga de:

1. Descubrimiento y Conexión: Utilizar el BluetoothAdapter y el BluetoothDevice para escanear y emparejar con el ESP32.
2. Manejo del SPP: Usar el BluetoothSocket para abrir un canal de comunicación de flujo de datos (Serial Port Profile).
3. Visualización de Datos: Recibir los datos numéricos de distancia del ESP32 y mostrarlos dinámicamente, típicamente mediante un TextView o un medidor gráfico (como un ProgressBar o una vista personalizada). Esto cumple con el requisito de "mostrar la distancia mínima y máxima".

Fundamentos Teóricos Detallados

2.1. Comunicación Bluetooth Clásico (SPP)

El Serial Port Profile (SPP) es esencial porque emula la comunicación por cable que tradicionalmente se usa con microcontroladores (como la conexión USB-Serial), pero de forma inalámbrica.

* Rol de Dispositivos: En la conexión Android-ESP32, el ESP32 generalmente actúa como el Servidor (o dispositivo periférico) que espera una conexión, mientras que la *app* Android actúa como el Cliente (o dispositivo central) que inicia la conexión.
* UUID Estándar: El UUID 00001101-0000-1000-8000-00805F9B34FB es el identificador reservado globalmente para el servicio SPP. Al usar este UUID, ambas plataformas (Android y ESP32) saben qué tipo de servicio de comunicación se está estableciendo.
* Manejo de I/O: En Android, una vez conectado el BluetoothSocket, se obtiene un InputStream para leer datos del ESP32 y un OutputStream para escribir datos al ESP32 (aunque en la Parte II solo se usa el *Input*).

2.2. Sensor de Efecto Hall (KY-024)

El Efecto Hall es el principio físico donde un campo magnético aplicado perpendicularmente a una corriente que fluye a través de un conductor produce un voltaje transversal a la corriente.

* Modelo KY-024: Este módulo típicamente utiliza un sensor Hall lineal (como el A3144 mencionado en la referencia), que entrega un voltaje analógico proporcional a la densidad de flujo magnético ().
* Relación Distancia-Voltaje:
  + Cuando el imán de neodimio se acerca al sensor, la intensidad del campo magnético aumenta, lo que incrementa el voltaje de salida del sensor (pin AO).
  + La conversión de la lectura del ADC a una distancia en centímetros es una calibración empírica. El *firmware* del ESP32 debe:
    1. Leer el valor del pin GPIO34 (ADC).
    2. *Mapear* o aplicar una función (obtenida experimentalmente) para convertir el valor ADC (0-4095) a una distancia estimada (ej., 0-10 cm).
    3. Transmitir este valor de distancia al Android.
* Uso del GPIO34: Se selecciona el GPIO34 porque, a diferencia de otros pines del ESP32 (GPIO36-39), este puede usarse como entrada analógica (*ADC*) sin la necesidad de desactivar Wi-Fi o Bluetooth, aunque se debe tener cuidado con la calibración de fábrica (VRef) del ADC del ESP32.

2.3. Arquitectura de Transmisión de Datos

La arquitectura opera en un bucle continuo de adquisición y transmisión:

1. Adquisición: El ESP32 lee el sensor Hall, realiza el promedio de muestras (para mitigar el ruido) y aplica la fórmula de calibración para obtener la distancia.
2. Transmisión (SPP): El ESP32 formatea el dato (ej., "5.3cm\n") y lo envía a través del puerto serie Bluetooth. El carácter de nueva línea (\n) es crucial, ya que actúa como un delimitador de paquete, indicando a la aplicación Android que ha llegado una lectura completa.
3. Recepción (Android): La aplicación Android, ejecutándose en un *hilo* separado (Thread), monitorea el InputStream, lee los datos hasta encontrar el delimitador (\n), y luego actualiza la Interfaz de Usuario.

Código de Arduino

Propósito General

Este programa convierte un ESP32 en un dispositivo Bluetooth que mide distancias usando un sensor magnético (KY-024) y envía los datos por Bluetooth cuando hay cambios significativos.

1. Librerías y Variables Globales

#include "BluetoothSerial.h"

BluetoothSerial SerialBT;

- Incluye la librería para funcionalidad Bluetooth

- Crea un objeto `SerialBT` para manejar la comunicación Bluetooth

const int pinAnalog = 34; // Pin ADC del ESP32 para leer el sensor

int distanciaAnterior = -1; // Almacena el último valor enviado

const int distanciaMin = 2; // Distancia mínima en cm

const int distanciaMax = 20; // Distancia máxima en cm

2. Configuración Inicial (`setup`)

void setup() {

Serial.begin(115200);

SerialBT.begin("ESP32\_Sensor\_Magnetico");

pinMode(pinAnalog, INPUT);

}

- Inicia comunicación serial para depuración

- Inicia Bluetooth con nombre "ESP32\_Sensor\_Magnetico"

- Configura el pin del sensor como entrada

3. Bucle Principal (`loop`)

a) Lectura Promediada del Sensor:

int valorSensor = 0;

for(int i = 0; i < 10; i++){

valorSensor += analogRead(pinAnalog);

}

valorSensor /= 10;

- Toma 10 lecturas del sensor y las promedia

- Esto reduce el ruido y mejora la estabilidad

b) Conversión a Distancia:

int distancia = map(valorSensor, 0, 4095, distanciaMax, distanciaMin);

- Convierte el valor analógico (0-4095) a distancia (2-20 cm)

-Nota importante: El mapeo está invertido porque cuando el sensor detecta algo más cerca, el valor analógico aumenta

c) Envío Condicional por Bluetooth:

if(abs(distancia - distanciaAnterior) >= 1){

if(SerialBT.connected()){

SerialBT.println(distancia);

Serial.println("Distancia enviada: " + String(distancia) + " cm");

delay(10);

}

distanciaAnterior = distancia;

}

- Solo envía datos si la distancia cambió en al menos 1 cm

- Verifica que haya un dispositivo Bluetooth conectado

- Envía la distancia por Bluetooth y también la muestra por Serial

- Actualiza la "distancia anterior" para comparaciones futuras

Flujo de Datos

Sensor KY-024 → Lectura Analógica → Promediado → Conversión a cm →

→ Verificación de Cambio → Envío Bluetooth (si aplica)

Características Importantes

* Eficiencia: Solo envía datos cuando hay cambios significativos
* Estabilidad: Usa promediado para lecturas más confiables
* Comunicación Dual: Muestra datos por Serial y Bluetooth
* Bajo Consumo: Lee cada 100ms, no constantemente

CODIGO MAINACTIVITY JAVA

Propósito General

Esta aplicación Android se conecta vía Bluetooth a un ESP32 que envía datos de sensores y muestra la información recibida en tiempo real con una interfaz visual.

Estructura y Componentes Principales

1. Declaraciones Iniciales

private static final UUID MY\_UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

UUID específico para comunicación Serial Port Profile (SPP), estándar para dispositivos Bluetooth serie como Arduino/ESP32

2. Variables de Bluetooth

BluetoothAdapter bluetoothAdapter; // Adaptador Bluetooth del dispositivo

BluetoothDevice selectedDevice; // Dispositivo seleccionado para conectar

BluetoothSocket socket; // Socket para la comunicación

InputStream inputStream; // Flujo de entrada para recibir datos

3. Elementos de UI

* connectButton: Botón para mostrar dispositivos
* deviceListView: Lista de dispositivos emparejados
* bluetoothStatus: Estado del Bluetooth
* sensorDataText: Muestra datos del sensor
* connectionIndicator, dataIndicator: Indicadores visuales (rojo/verde)

Flujo de la Aplicación

1. Inicialización (`onCreate`)

protected void onCreate(Bundle savedInstanceState) {

// Inicializa vistas y verifica estado Bluetooth

checkBluetoothStatus();

}

2. Verificación de Bluetooth (`checkBluetoothStatus`)

private void checkBluetoothStatus() {

if (bluetoothAdapter == null) {

// Dispositivo no soporta Bluetooth

} else if (!bluetoothAdapter.isEnabled()) {

// Bluetooth apagado - solicita activarlo

Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

startActivityForResult(enableBtIntent, 1);

} else {

// Bluetooth listo - muestra estado verde

}

}

3. Mostrar Dispositivos Emparejados (`showPairedDevices`)

private void showPairedDevices() {

pairedDevices = bluetoothAdapter.getBondedDevices();

// Llena la lista con nombre + dirección MAC

deviceListAdapter.add(device.getName() + "\n" + device.getAddress());

}

4. Conexión al Dispositivo (`connectToDevice`)

private void connectToDevice() {

new Thread(() -> {

socket = selectedDevice.createRfcommSocketToServiceRecord(MY\_UUID);

socket.connect();

inputStream = socket.getInputStream();

isConnected = true;

listenForData(); // Inicia escucha de datos

}).start();

}

Importante: La conexión se hace en un hilo separado para no bloquear la UI

5. Recepción de Datos (`listenForData`)

private void listenForData() {

byte[] buffer = new byte[256];

while (isConnected) {

bytes = inputStream.read(buffer);

String message = new String(buffer, 0, bytes).trim();

runOnUiThread(() -> {

sensorDataText.setText(message);

updateSensorDisplay(message); // Actualiza colores

});

}

}

Bucle infinito que escucha constantemente los datos entrantes

runOnUiThread actualiza la interfaz desde el hilo principal

6. Visualización Inteligente (`updateSensorDisplay`)

private void updateSensorDisplay(String sensorValue) {

double value = Double.parseDouble(sensorValue);

if (value > 100) {

sensorDataText.setTextColor(Color.RED); // Valor alto - peligro

} else if (value > 50) {

sensorDataText.setTextColor(Color.ORANGE); // Valor medio - advertencia

} else {

sensorDataText.setTextColor(Color.GREEN); // Valor bajo - normal

}

}

Cambia colores según el valor numérico recibido

Proporciona retroalimentación visual inmediata

Características de Diseño

Indicadores Visuales

* Connection Indicator: Rojo (desconectado) ↔ Verde (conectado)
* Data Indicator Rojo (sin datos) ↔ Verde (datos recibidos)
* Texto del sensor: Cambia color según valores (Verde < 50, Naranja 50-100, Rojo > 100)

Manejo de Hilos

* Conexión Bluetooth: En hilo secundario
* Recepción de datos: En hilo secundario
* Actualización UI: Siempre en hilo principal con `runOnUiThread()`

Manejo de Errores

* Try-catch en conexión y recepción de datos
* Toasts para errores de conexión
* Indicadores visuales para estados de error

Flujo de Comunicación Completo

App inicia → Verifica Bluetooth → Usuario selecciona dispositivo →

→ Conexión en hilo secundario → Bucle de recepción de datos →

→ Actualización UI en hilo principal → Visualización con colores

Compatibilidad con el Código ESP32

Esta aplicación está diseñada específicamente para trabajar con el ESP32 que envía:

Datos numéricos (distancias en cm)

Formato simple: un número por línea

Frecuencia: aproximadamente cada 100ms

CODIGO ACTIVITY\_MAIN.XML

Propósito General

Este XML define una interfaz moderna y atractiva para la aplicación de monitoreo de sensores Bluetooth, con diseño Material Design y elementos visuales intuitivos.

Estructura General

Contenedor Principal

<LinearLayout

android:orientation="vertical"

android:background="@drawable/background\_gradient"

android:padding="20dp">

Layout vertical con gradiente de fondo

Padding para márgenes internos

Secciones de la Interfaz

1. Header con Logo y Título

<LinearLayout

android:orientation="horizontal"

android:gravity="center\_vertical">

<ImageView

android:src="@drawable/ic\_bluetooth"

android:tint="#FFFFFF"/>

<TextView

android:text="Sensor Magnético ESP32"

android:textColor="#FFFFFF"/>

Logo Bluetooth en blanco

Título principal en texto blanco sobre el gradiente

2. Tarjeta de Estado (Doble Indicador)

<androidx.cardview.widget.CardView

app:cardCornerRadius="16dp"

app:cardElevation="4dp">

Indicador Bluetooth:

* connectionIndicator: Círculo rojo/verde para estado de conexión
* bluetoothStatus`: Texto descriptivo del estado
* Icono Bluetooth decorativo

Indicador de Datos:

* dataIndicator: Círculo rojo/verde para recepción de datos
* dataStatusText: Estado de la transmisión
* Icono de datos decorativo

3. Botón de Búsqueda

<com.google.android.material.button.MaterialButton

android:text="BUSCAR DISPOSITIVOS ESP32"

app:icon="@drawable/ic\_search"

app:backgroundTint="#4F46E5">

Botón Material Design con icono de búsqueda

Color azul (#4F46E5) para acción principal

Esquinas redondeadas (12dp)

4. Lista de Dispositivos

<androidx.cardview.widget.CardView

android:layout\_height="0dp"

android:layout\_weight="1">

Altura flexible (usa weight para ocupar espacio disponible)

Header con título "Dispositivos Emparejados"

deviceListView: Lista que muestra dispositivos Bluetooth disponibles

5. Display del Sensor

<androidx.cardview.widget.CardView

app:cardCornerRadius="20dp"

app:cardElevation="6dp">

Esquinas más redondeadas (20dp) para destacar

Mayor elevación para efecto de profundidad

Contenido del Display:

<TextView

android:id="@+id/sensorDataText"

android:text="---"

android:textSize="42sp"

android:textStyle="bold"/>

Texto grande (42sp) para valores del sensor

Placeholder "---" cuando no hay datos

Texto descriptivo "unidades magnéticas"

Características de Diseño

* Sistema de Colores
* Blanco (#FFFFFF): Textos sobre gradiente
* Azul (#4F46E5): Botón principal
* Gris oscuro (#2D3748, #1F2937): Textos principales
* Gris medio (#718096, #6B7280): Iconos y textos secundarios
* Gris claro (#9CA3AF): Textos terciarios

Elementos Visuales

* CardViews con esquinas redondeadas (16dp-20dp)
* Elevación para efectos de sombra
* Indicadores circulares para estados
* Iconografía consistente
* Backgrounds personalizados (@drawable/...)

Jerarquía Visual

* Display del sensor (más prominente)
* Botón de acción (color llamativo)
* Tarjetas de información (fondo blanco)
* Header (sobre gradiente)

Recursos Drawable Referenciados

* @drawable/background\_gradient - Fondo de pantalla
* @drawable/card\_background - Fondo de tarjetas
* @drawable/indicator\_off` - Indicador circular
* @drawable/sensor\_background - Fondo del display
* @drawable/card\_header\_background - Header de lista
* @drawable/ic\_bluetooth, ic\_search, etc. - Iconos

Relación con el Código Java

Elementos Conectados

* connectButton → `showPairedDevices()`
* deviceListView → Lista de dispositivos Bluetooth
* bluetoothStatus → Estado de conexión
* dataStatusText → Estado de datos
* sensorDataText → Valores del sensor
* connectionIndicator → Indicador visual Bluetooth
* dataIndicator → Indicador visual de datos

Feedback Visual Integrado

* Indicadores cambian de rojo a verde según estado
* Texto del sensor cambia color según valores
* Estados de conexión actualizados en tiempo real

A screenshot of a phone

AI-generated content may be incorrect.IMAGEN DE LA APLICACIÓN

Referencias en Formato APA

ESP32 y Programación (Arduino Core)

1. Espressif Systems. (2024). *ESP32 Technical Reference Manual*. Recuperado
2. Arduino Core for the ESP32. (2024). *BluetoothSerial Library Documentation*.

Comunicación Android (Bluetooth SPP)

1. Google Developers. (2024). *Bluetooth | Android Developers*. Recuperado
2. IEEE. (2014). *IEEE 802.15.1: Local and Metropolitan Area Networks – Specific Requirements – Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)*. IEEE Standards Association.

Sensor de Efecto Hall (KY-024) y Principios

1. Llamas, L. (s.f.). *Detectar campos magnéticos con Arduino y sensor Hall A3144*. LuisLlamas.es. Recuperado de <https://www.luisllamas.es/detectar-campos-magneticos-con-arduino-y-sensor-hall-a3144/>].
2. Pons, E. (2018). *Sistemas de Adquisición de Datos*. Editorial Paraninfo.

Desarrollo de Aplicaciones Móviles

1. Deitel, P. J., & Deitel, H. M. (2021). *Android 11 para Programadores: Un Enfoque App-Driven* (4ta ed.). Pearson Education.