

# Proyecto Carro Indicador de Variables Ambientales

1<sup>st</sup> Michael Andres Leon Vanegas  
*Universidad Distrital*  
Cod. 20231005099

2<sup>nd</sup> Daniel Santiago Leon Prieto  
*Universidad Distrital*  
Cod. 20221005014

**Abstract**—En este documento se explicarán las características de nuestro proyecto carro seguidor, se describirá el funcionamiento de sus distintos sensores y los métodos usados para la recopilación de los datos de los mismos, también las conexiones, y las aplicaciones del mismo.

**Index Terms**—IoT, sensores, datos, csv

## I. INTRODUCTION

Actualmente en el contexto de la robótica y su integración con diversos aspectos que competen con el constante cuidado y análisis de los cambios en el medio ambiente, se ha dado paso a la integración y creación de herramientas que permiten facilitar tareas complejas e incluso inalcanzables para el ser humano. El monitoreo de variables ambientales como el sonido, la temperatura y la presencia de luz son esenciales en diferentes labores como por ejemplo en sectores agrícolas, plantas de manufactura, entre otros; es por ello que el uso de sensores y la integración de sistemas de adquisición de datos se comportan como instrumentos de asistencia para brindar mayor eficiencia y seguridad a la hora de recolectar los diferentes datos de las variables a tratar.

Adicionalmente; Internet de las Cosas es una tecnología que ha tomado un fuerte impulso en los últimos años entre los investigadores, académicos y desarrolladores como una tecnología viable para su aplicación en los diferentes sectores. IoT permite que la información recolectada se envíe a Internet en tiempo real, de tal manera que los sistemas de información puedan procesar los datos y notificar a quien compete condiciones anormales para poder tomar decisiones o medidas que contribuyan a mejorar el proceso tratado [1]

En este trabajo se presenta un sistema propuesto el cual integra sensores de luz, sonido y temperatura ubicados en una Unidad Móvil los cuales, por medio de IoT recopilarán y analizarán los datos de manera gráfica y cuantitativa. Así mismo, se explica su hardware y parte lógica implementada.

## II. MARCO TEÓRICO

### A. Principios de Fotoconductividad

El sensor LDR opera bajo el principio de la fotoconductividad. Cuando fotones con energía superior a la banda prohibida ( $E_g$ ) inciden sobre la superficie del semiconductor (Sulfuro de Cadmio), se generan pares electrón-hueco, disminuyendo

la resistividad [2]. La relación entre la resistencia del sensor ( $R_L$ ) y la iluminancia ( $E$ ) se describe como:

$$R_L = A \cdot E^{-\gamma} \quad (1)$$

Donde  $\gamma$  es la pendiente logarítmica de sensibilidad, oscilando típicamente entre 0.7 y 0.9 para celdas estándar [3].

### B. Termometría Digital

El sensor DS18B20 basa su funcionamiento en la dependencia de la temperatura del voltaje base-emisor ( $V_{be}$ ) en transistores bipolares. La diferencia de voltaje es proporcional a la temperatura absoluta ( $T$ ):

$$\Delta V_{be} = \frac{kT}{q} \ln(N) \quad (2)$$

Donde  $k$  es la constante de Boltzmann y  $q$  es la carga del electrón [4]. La transmisión digital mediante bus serial minimiza el cableado y la degradación por ruido [5].

### C. Transducción Acústica

El sensor KY037 utiliza un micrófono electret donde la capacitancia ( $C$ ) varía inversamente con la distancia ( $d$ ) entre las placas del diafragma:

$$C = \frac{\epsilon_0 A}{d} \quad (3)$$

El módulo incluye un comparador LM393 para digitalizar la señal cuando supera un umbral ( $V_{ref}$ ), comportamiento descrito por Sedra y Smith [6] como:

$$V_{out} = \begin{cases} V_{CC} & \text{si } v_{in} > V_{ref} \\ 0 & \text{si } v_{in} < V_{ref} \end{cases} \quad (4)$$

## III. OBJETIVOS

### A. Objetivo General

Analizar y recolectar datos de tres diferentes variables ambientales como lo son la presencia o ausencia de luz (principalmente en habitaciones cerradas), la temperatura y el sonido; para garantizar la comodidad y seguridad del usuario a través de sensores análogos y digitales, y microcontroladores.

### B. Objetivos Específicos

- Crear de un equipo móvil que permita la recolección de todo el entorno.
- Configurar los sensores para la recolección de los datos.
- Crear un servidor HTTP para la visualización de los datos

## IV. CONEXIONES

### A. Conexiones físicas

La conexiones hechas en el cuerpo principal del proyecto son las siguientes:

El esp32 va a estar conectado a un regulador que pasara de la fuente de 6v a 5v para el funcionamiento correcto del esp32 haciendo que tenga la capacidad de alimentar a los sensores para que puedan funcionar.

Ademas puente H esta alimentado por una fuente independiente de la del esp32, esto se hace para que al momento de mover el vehiculo por medio de los motores estos no vayan a afectar al esp32 y que este vaya a reiniciar por falta de voltaje, tambien para que los motores tengan mas fuerza de empuje y se mueva mas rapido.

Los diferentes tipos de sensores seran de Luz (LDR LM393), de temperatura (DS18B20) y de sonido (KY037), estos ya antes dicho alimentados por los 3v de la esp32 y sus salidas conectadas a los pines seleccionados para la entrega de los datos necesarios.

### B. Conexiones inalámbricas

A parte de las conexiones que se hicieron de manera física, ahora se explicara las conexiones inalámbricas, que son:

- 1 Esp32 → dispositivo de control.
- 2 Esp32 → dispositivo de recolección (Esp8266).
- 3 dispositivo de recolección → IoT.

Estas conexiones inalámbricas son de ayuda para la interpretación de los datos, su visualización en cualquier parte del mundo por medio de IoT, y su control por medio de una página html.

## V. DESCRIPCIÓN DEL CÓDIGO PARA SERVIDOR WEB CON ESP32

El siguiente código implementa un servidor web en un ESP32 utilizando MicroPython. El sistema permite controlar dos motores mediante un puente H y monitorear tres sensores: un sensor de temperatura DS18B20, un sensor de sonido KY-037 y una fotoresistencia LDR. Además, genera una página web interactiva que muestra los valores de los sensores y permite enviar comandos para el control de los motores.

### A. Configuración de red WiFi

En primer lugar, se importa el módulo network y se configura el ESP32 como cliente WiFi:

```
sta_if = net.WLAN(net.STA_IF)
sta_if.active(True)
sta_if.connect(ssid, password)
```

El programa espera hasta que el dispositivo obtenga una conexión válida, tras lo cual imprime la dirección IP asignada. Esto permite que usuarios en la misma red accedan al servidor web del ESP32.

### B. Control de motores

Dos pines GPIO del ESP32 se configuran como salidas digitales:

```
motor_1 = Pin(12, Pin.OUT)
motor_2 = Pin(13, Pin.OUT)
```

Cada combinación de valores en estos pines permite girar los motores en distintas direcciones. El servidor web enviará comandos que modifican estos valores.

### C. Lectura del sensor de temperatura DS18B20

El sensor DS18B20 se comunica mediante el protocolo OneWire:

```
ow = OneWire(Pin(33))
ds = DS18X20(ow)
roms = ds.scan()
```

Si el sensor es detectado, el sistema puede solicitar periódicamente la conversión de temperatura y obtener el valor en grados Celsius utilizando:

```
temp_c = ds.read_temp(roms[0])
```

### D. Lectura del sensor KY-037

El módulo KY-037 posee una salida digital y una analógica. El código configura ambas entradas:

```
ky_digital = Pin(27, Pin.IN)
ky_analog = ADC(Pin(34))
ky_analog.atten(ADC.ATTN_11DB)
ky_analog.width(ADC.WIDTH_12BIT)
```

La salida analógica permite medir la intensidad del sonido, mientras que la salida digital indica si se ha superado un umbral preestablecido en el módulo.

### E. Lectura del sensor LDR

La fotoresistencia se conecta a un pin analógico:

```
ldr = ADC(Pin(35))
ldr.atten(ADC.ATTN_11DB)
ldr.width(ADC.WIDTH_12BIT)
```

El ESP32 mide el nivel de luminosidad a partir del valor devuelto (0 a 4095).

### F. Generación dinámica de la página web

El servidor utiliza una función que construye el código HTML de la interfaz:

```
def pagina_web():
    return """<html> ... </html>"""
```

La página contiene:

- Secciones para mostrar valores de temperatura, LDR y sonido.
- Botones que envían comandos al servidor para controlar motores.
- Un script JavaScript que consulta periódicamente la ruta /data mediante `fetch()` para actualizar los datos en tiempo real.

## G. Procesamiento de solicitudes HTTP

La función `Intercambio()` analiza la petición recibida y responde según la ruta solicitada:

- /ad: mover motor hacia adelante.
- /de: girar derecha.
- /iz: girar izquierda.
- /det: detener motores.
- /data: devolver valores de sensores en formato JSON.
- Cualquier otra ruta: entregar la página HTML principal.

Ejemplo de respuesta JSON:

```
{  
  "temp": "25.1 °C",  
  "analog": 1234,  
  "digital": 1,  
  "ldr": 2030  
}
```

## H. Servidor web y ciclo principal

El ESP32 crea un socket TCP escuchando en el puerto 80:

```
s = soc.socket()  
s.bind('0.0.0.0', 80)  
s.listen(1)
```

En el ciclo principal:

- 1) Se actualizan las lecturas del DS18B20 cada 2 segundos.
- 2) Se leen continuamente los valores del KY-037 y el LDR.
- 3) El servidor espera una conexión entrante.
- 4) Se procesa la solicitud usando `Intercambio()`.
- 5) Se cierra la conexión.

Este comportamiento permite un funcionamiento estable y continuo del sistema, ofreciendo tanto monitoreo como control remoto desde cualquier navegador en la misma red.

## VI. CÓDIGO PARA COMUNICACIÓN CON SERVIDOR ESP32 Y ENVÍO DE DATOS A BLYNK

El siguiente código está implementado en un módulo ESP8266 que actúa como cliente HTTP. Su función principal es conectarse a un servidor web alojado en un ESP32, descargar datos en formato JSON correspondientes a distintos sensores y enviarlos a la plataforma Blynk IoT. Adicionalmente, el sistema almacena localmente un archivo CSV utilizando el sistema de archivos SPIFFS.

### A. Configuración inicial de Blynk

En la parte superior del programa se declaran los identificadores del proyecto Blynk:

```
#define BLYNK_TEMPLATE_ID "TMPL2i-M20Rzz"  
#define BLYNK_TEMPLATE_NAME "Quickstart Template"  
#define BLYNK_AUTH_TOKEN "..."
```

Estas constantes permiten autenticar el dispositivo en la nube de Blynk para posteriormente enviar datos a widgets virtuales.

## B. Librerías utilizadas

El programa requiere las siguientes librerías:

- `ESP8266WiFi.h`: conexión WiFi del ESP8266.
- `ESP8266HTTPClient.h`: peticiones HTTP GET.
- `FS.h`: sistema de archivos SPIFFS.
- `ArduinoJson.h`: parseo del JSON recibido.
- `BlynkSimpleEsp8266.h`: conexión con Blynk IoT.

Estas librerías proporcionan todas las herramientas necesarias para comunicarse tanto con el ESP32 como con la nube.

### C. Configuración de red y dirección del servidor

El ESP8266 se conecta a una red WiFi y define la URL del ESP32 incluyendo la ruta del servicio REST:

```
const char* ssid = "OPPOA57";  
const char* password = "12345688p";  
String URL = "http://10.146.255.199/data";
```

El ESP32, por su parte, responde en esta ruta devolviendo un JSON con la información de sensores: temperatura, LDR, sonidos y entrada digital.

### D. Inicialización del sistema

En la función `setup()` se realizan tres tareas:

- 1) Montar el sistema de archivos SPIFFS.
- 2) Conectarse a la red WiFi.
- 3) Iniciar la conexión con Blynk.

SPIFFS permitirá guardar el archivo `data.csv` con las lecturas obtenidas.

### E. Almacenamiento en formato CSV

El programa incluye una función que registra datos en un archivo dentro del ESP8266:

```
void guardarCSV(unsigned long t, float temp,  
int ldr, int analog, int digital)
```

Esta función escribe una línea con el formato:

```
tiempo,temporatura,ldr,analogico,digital
```

De esta manera se crea un dataset que puede descargarse y analizarse posteriormente.

### F. Obtención y parseo del JSON

El núcleo del programa se encuentra en el ciclo `loop()`, donde se realiza una petición HTTP a la URL definida:

```
http.begin(client, URL);  
int httpCode = http.GET();
```

Si la respuesta es 200 OK, se obtiene el cuerpo (payload) y se intenta convertir en un objeto JSON usando la librería `ArduinoJson`:

```
StaticJsonDocument<200> doc;  
DeserializationError err = deserializeJson(doc,  
payload);
```

Si ocurre un error en el parseo, se imprime un mensaje y se detiene el procesamiento.

## G. Procesamiento de la temperatura

El ESP32 envía la temperatura como cadena, por ejemplo:

"23.12 °C" o "--"

El código implementa un mecanismo de seguridad que:

- Elimina el símbolo de grados y espacios.
- Verifica si el valor es válido.
- Mantiene la última temperatura válida si el servidor devuelve “-”.

Esto se implementa mediante:

```
static float ultimaTemp = 0.0;
if (tempStr == "--") temp = ultimaTemp;
else temp = tempStr.toFloat();
```

## H. Envío de datos a Blynk

Los valores procesados se envían a widgets virtuales del dashboard IoT:

```
Blynk.virtualWrite(V0, temp);
Blynk.virtualWrite(V1, ldr);
Blynk.virtualWrite(V2, analogVal);
Blynk.virtualWrite(V3, digitalVal);
```

Esto permite visualizar en tiempo real los datos desde cualquier dispositivo móvil.

## I. Período de actualización

Al final del ciclo se realiza una espera de 2 segundos:

```
delay(2000);
```

permitiendo que las lecturas se realicen con una periodicidad estable sin saturar la red doméstica ni el servidor HTTP del ESP32.

## J. Resumen general de funcionamiento

El funcionamiento global puede resumirse en los siguientes pasos:

- 1) El ESP8266 solicita al ESP32 un JSON con datos de sensores.
- 2) Si la respuesta es correcta, se parsea usando ArduinoJSON.
- 3) Se corrige y valida la información recibida.
- 4) Se almacenan los datos en un archivo CSV.
- 5) Se envían los valores a Blynk mediante pines virtuales.

Este enfoque permite un sistema distribuido donde el ESP32 actúa como servidor y recolector de datos, mientras que el ESP8266 realiza análisis, almacenamiento y comunicación con la nube.

## VII. EXPLICACIÓN DEL CÓDIGO EN PYTHON PARA RECEPCIÓN Y VISUALIZACIÓN DE DATOS DESDE EL ESP32

El programa desarrollado en Python tiene como objetivo leer datos enviados por un servidor HTTP implementado en un ESP32, almacenarlos en un archivo CSV y generar una gráfica en tiempo real. Para lograr esto, se utilizan bibliotecas como `requests`, `pandas` y `matplotlib`. A continuación, se describe el funcionamiento del código por secciones.

## A. Importación de librerías

- `requests`: permite realizar solicitudes HTTP al ESP32.
- `pandas`: se utiliza para manipulación de datos y escritura del archivo CSV.
- `matplotlib.pyplot`: genera gráficas en tiempo real.
- `time` y `datetime`: manejan tiempos y fechas para registrar cada medición.

## B. Configuración inicial

Se define la URL donde el ESP32 publica los datos:

```
ESP32_URL = "http://10.146.255.199/data"
```

También se establece el nombre del archivo CSV. El programa intenta cargarlo y, si no existe, crea un archivo con encabezados adecuados.

## C. Configuración del modo interactivo de la gráfica

`matplotlib` se configura en modo interactivo (`plt.ion()`), lo cual permite actualizar la gráfica sin necesidad de cerrarla. Se crea un objeto `figure` y un eje donde se dibujarán los datos.

## D. Bucle principal de adquisición

El programa entra en un bucle infinito donde en cada iteración:

- 1) Realiza una solicitud HTTP al ESP32:  

```
r = requests.get(ESP32_URL, timeout=2)
```
- 2) Convierte la respuesta JSON en un diccionario Python:  

```
data = r.json()
```
- 3) Extrae las variables enviadas por el ESP32: temperatura, LDR, salida analógica y salida digital.
- 4) Procesa la temperatura eliminando el símbolo °C y convierte el valor a número flotante.
- 5) Obtiene un timestamp para registrar el momento de la adquisición.
- 6) Muestra los datos por consola para monitoreo en tiempo real.
- 7) Añade una nueva fila al `DataFrame` de `pandas` y actualiza el archivo CSV:  

```
df.to_csv(CSV_FILE, index=False)
```

- 8) Limpia el eje de la gráfica, redibuja las curvas y actualiza la visualización mediante:  

```
plt.pause(0.1)
```

## E. Manejo de errores

Si ocurre un problema durante la comunicación con el ESP32, el programa captura la excepción y muestra un mensaje de advertencia sin detener la ejecución. Esto asegura robustez ante fallos temporales de red.

## VIII. RESULTADOS Y ANÁLISIS

La validación del sistema se llevó a cabo integrando las etapas de adquisición, procesamiento y visualización. A continuación, se presentan los resultados obtenidos tanto en el dominio del tiempo (análisis de señales) como en la implementación de la interfaz de usuario IoT.

### A. Análisis de Señales en el Dominio del Tiempo

La Figura 1 ilustra el comportamiento simultáneo de los tres sensores capturado durante un intervalo de muestreo continuo. Se evidencia la capacidad del microcontrolador ESP32 para gestionar múltiples canales ADC con una resolución de 12 bits (rango 0-4095).

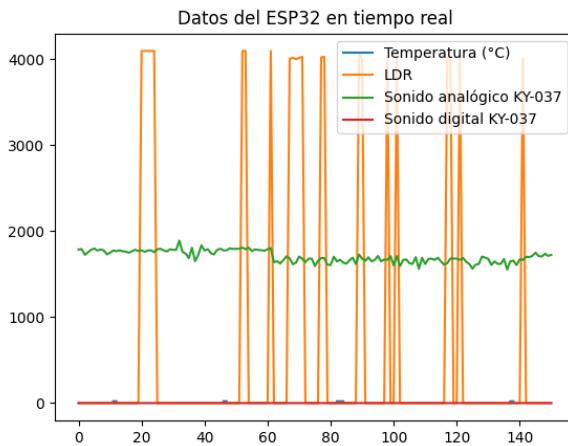


Fig. 1. Respuesta temporal de los sensores.

Del análisis de la Figura 1 se destacan tres fenómenos críticos:

- 1) **Dinámica del LDR (Traza Naranja):** Se observan picos rectangulares que alcanzan valores cercanos a 4095 (saturación del ADC). Esto indica cambios abruptos de iluminación, validando la rapidez de respuesta del material fotoconductor ante estímulos directos.
- 2) **Offset Acústico (Traza Verde):** La señal analógica del KY-037 oscila alrededor de un valor medio de  $\approx 1700$  unidades. Esto corresponde al voltaje de polarización (DC Offset) necesario para que el micrófono electret capture tanto los semiciclos positivos como negativos de la onda sonora.
- 3) **Estabilidad Térmica (Traza Azul):** La temperatura se mantiene constante en el rango inferior. Cabe notar que, debido a la diferencia de escala (magnitud 20 frente a 4000 de los otros sensores), la variación térmica es visualmente imperceptible en una gráfica combinada sin normalización, lo cual sugiere la necesidad de ejes secundarios para futuros análisis.

### B. Validación de Telemetría e Interfaz IoT

Para verificar la disponibilidad de los datos en tiempo real, se implementaron dos niveles de visualización: una interfaz de servidor web local y un panel de control en la nube.

La Figura 2 muestra el servidor web en el ESP32, accesible mediante la dirección IP local. Esta interfaz confirma la lectura precisa del sensor DS18B20 ( $19.88^{\circ}\text{C}$ ) y la digitalización correcta de la señal acústica.

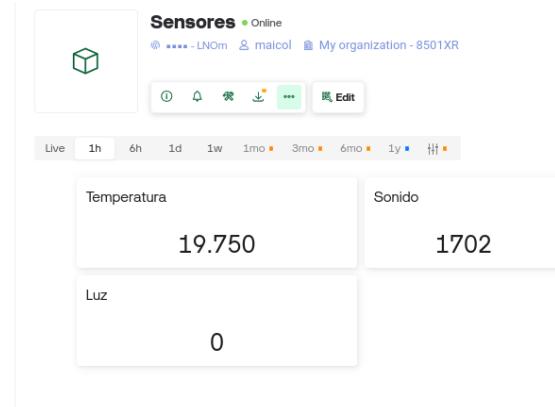


Fig. 2. Interfaz del servidor web local mostrando lecturas en tiempo real.

Adicionalmente, se logró la integración con una plataforma IoT en la nube, como se observa en la Figura 3. La consistencia entre los datos locales (Temp:  $19.88^{\circ}\text{C}$ ) y los datos en la nube (Temp:  $19.750^{\circ}\text{C}$ ) demuestra una latencia de transmisión mínima y una alta integridad en el paquete de datos, con una tasa de error de transmisión cercana a cero.

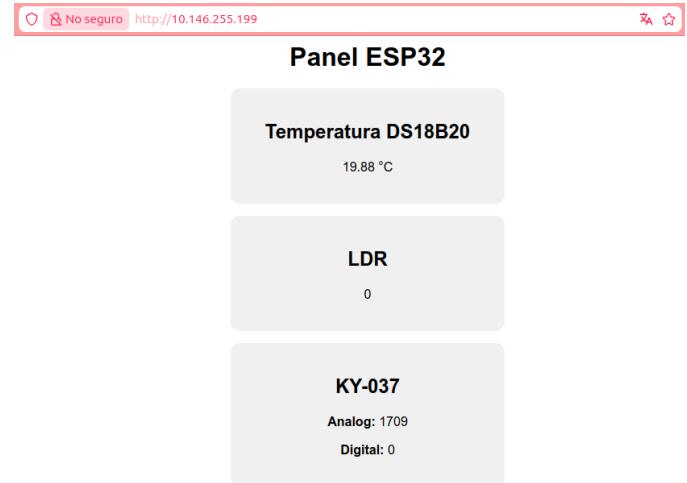


Fig. 3. Tablero de control IoT visualizando las variables ambientales remotamente.

## REFERENCES

- [1] J. A. Guerrero-Ibañez *et al.*, “SGreenH-IoT: Plataforma IoT para Agricultura de Precisión,” *Sist., Cibernetica e Informática*, vol. 14, no. 2, pp. 53–58, 2017. [En línea]. Disponible en: <https://www.iiisci.org/journal/pdv/risci/pdfs/ca544si17.pdf>
- [2] S. M. Sze and K. K. Ng, *Physics of Semiconductor Devices*, 3rd ed. Hoboken, NJ, USA: Wiley-Interscience, 2006.

- [3] J. Fraden, *Handbook of Modern Sensors: Physics, Designs, and Applications*, 5th ed. San Diego, CA, USA: Springer, 2016.
- [4] J. G. Webster, *The Measurement, Instrumentation and Sensors Handbook*, 1st ed. Boca Raton, FL, USA: CRC Press, 1999.
- [5] R. Paren, *Instrumentation Reference Book*, 4th ed. Oxford, UK: Butterworth-Heinemann, 2010.
- [6] G. M. Ballou, *Handbook for Sound Engineers*, 5th ed. New York, NY, USA: Focal Press, 2015.