PROYECTO FINAL

Martín Felipe Vásquez L.
Sara Catalina González R.
Juan Camilo Moreno Z.

Noviembre 2023

Pontificia Universidad Javeriana
Ingeniería Electrónica
Sistemas embebidos.

Introducción

En el ámbito de la Ingeniería Electrónica, la evolución constante de la tecnología nos impulsa a diseñar sistemas embebidos cada vez más sofisticados y adaptativos. Este proyecto final representa un paso significativo en esta dirección, abordando desafíos contemporáneos en el monitoreo y la gestión de variables ambientales cruciales. Los estudiantes, han colaborado en el desarrollo de un sistema embebido avanzado que no solo establece una base sólida para el monitoreo ambiental, sino que también sienta las bases para un proyecto de grado llamado SOLARI.

En el marco de este proyecto, se ha concebido un sistema de monitoreo de variables ambientales con un enfoque específico en el contexto de la energía solar. Las variables críticas que se miden incluyen la temperatura y la irradiancia UV, dos factores esenciales para comprender y optimizar el rendimiento de los paneles solares. Este proyecto sirve como preludio al proyecto de grado SOLARI, que se centra en el diseño y la implementación de un sistema de monitoreo avanzado para paneles solares, contribuyendo al avance de la eficiencia y la gestión sostenible de la energía solar.

Las pruebas realizadas, que abarcan desde el inicio automático del programa hasta la gestión de desconexiones temporales, validan la robustez y la adaptabilidad del sistema. Los resultados obtenidos confirman la eficacia del sistema embebido, proporcionando insights valiosos para la siguiente fase del proyecto SOLARI.

Resumen

Este proyecto de sistemas embebidos, se destaca por su enfoque avanzado en el monitoreo ambiental. Diseñado como preludio al proyecto de grado SOLARI, se centra en la medición precisa de variables clave, como temperatura e irradiancia UV, para la optimización de paneles solares.

La implementación técnica se presenta a través de diagramas y un código modular en GitHub, destacando la adaptabilidad del sistema. Las pruebas realizadas confirman la robustez, preparando el terreno para futuras fases. En resumen, este proyecto no solo cumple con los objetivos actuales, sino que también allana el camino hacia una gestión eficiente de la energía solar, perfilándose como un logro significativo en ingeniería electrónica.

El objetivo principal del proyecto es desarrollar un sistema embebido avanzado para el monitoreo ambiental, centrándose específicamente en la medición precisa de variables clave, como la temperatura y la irradiancia UV.

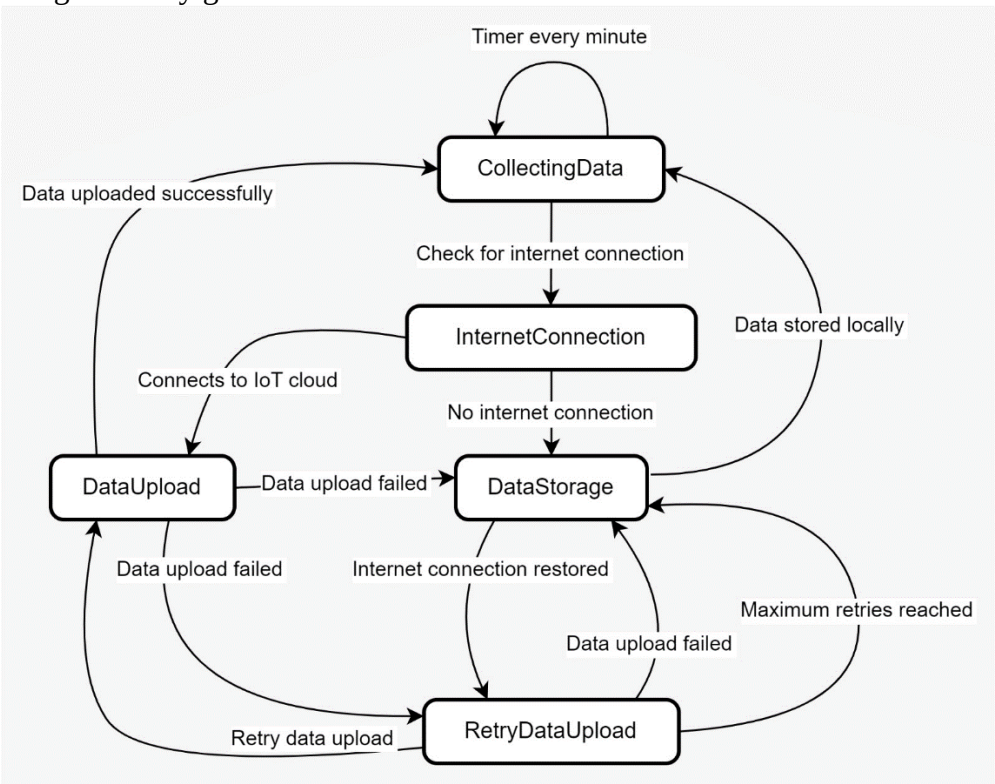
Objeticos Específicos
Desarrollar una comunicación utilizando MQTT para la transmisión efectiva de datos hacia Things Board.

Implementar un mecanismo robusto que asegure la toma continua de datos, incluso en ausencia de conexión, mediante almacenamiento local y reconexión automática.

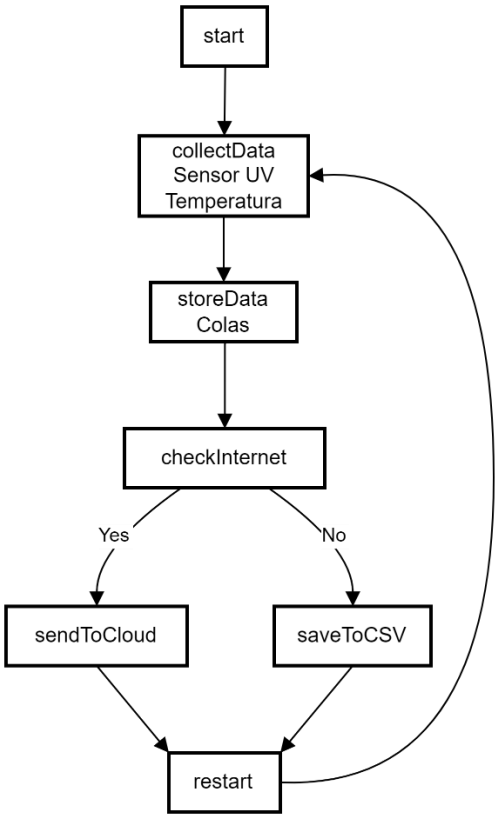
Configurar el sistema embebido para un inicio automático mediante systemd, asegurando que el programa se inicie de manera fiable al arrancar el dispositivo, proporcionando una gestión eficiente desde el inicio.

Desarrollo de la Solución

- **Diagrama de bloques**
  Para nuestro diagrama bloques se realiza de la siguiente manera en el cual se plantean un diagrama de estados las acciones de enviar Internet y corroborar si tiene conexión, también el hecho de recolectar la información cada minuto de organizarla y guardarla de manera local.

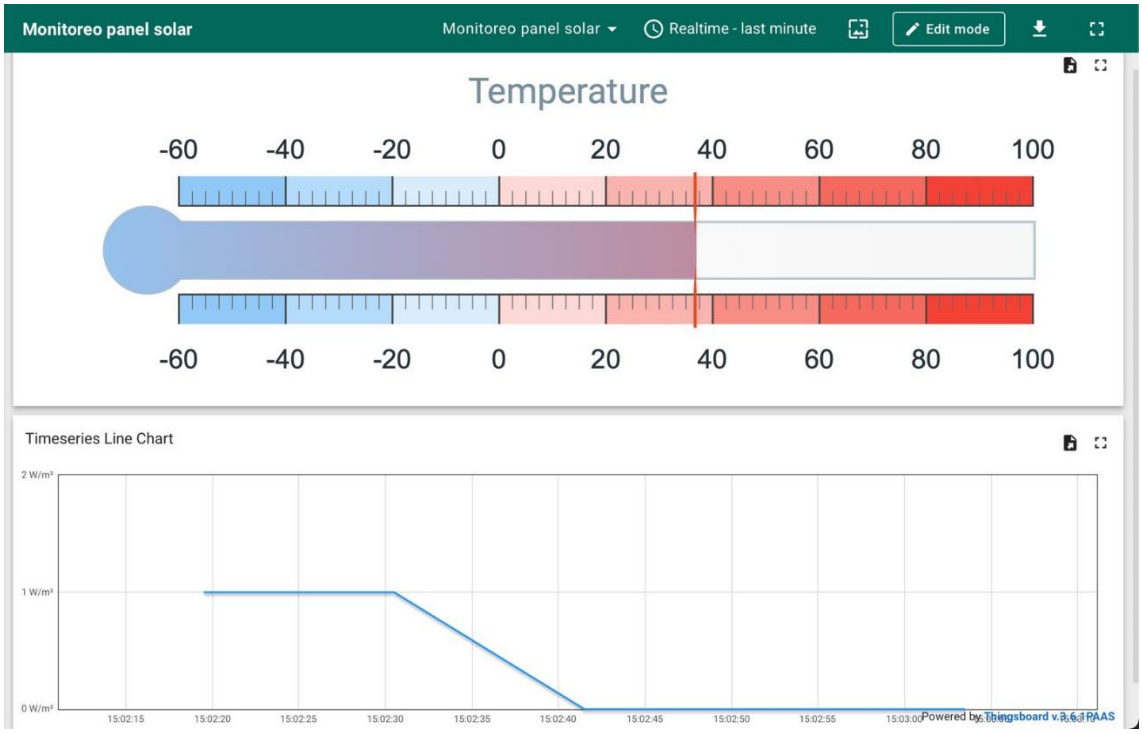


- **Diagrama de flujo**
- Es un diagrama de flujo que inicia después recolecta el valor de temperatura y el valor UV y luego se almacena en una cola después se pregunta la condición si hay Internet y si hay Internet envía los datos de la cola a la nube y si no hay Internet los envía a un archivo CSV y en ambos casos inicié de nuevo el proceso de recolección de datos cada segundo.



- **Dashboard**

Ya identificando nuestro Dashboard el cual se realiza por Things Board nos permite crear gauge Grafica de temperatura y una gráfica de irradiancia UV con la información recolectada del sensor.



- **Código**

  El código que finalmente se implemento fue el siguiente:

```
import socket
import time
import board
import adafruit_ltr390
from w1thermsensor import W1ThermSensor
import paho.mqtt.client as mqtt
import json
from datetime import datetime
from w1thermsensor import W1ThermSensor
from queue import Queue
from threading import Thread
import csv

# Datos de tu dispositivo y acceso MQTT
THINGSBOARD_HOST = "thingsboard.cloud"
ACCESS_TOKEN = "mobmzp9wjqylrk29bj8h"

# Tema MQTT para enviar datos
MQTT_TOPIC = "v1/devices/me/telemetry"

# Busca autom      ticamente el sensor DS18B20 conectado
sensor = W1ThermSensor()
#Busca automaticamente el sensor LTR390
i2c = board.I2C()
ltr = adafruit_ltr390.LTR390(i2c)

cola_temperatura = queue.LifoQueue()
cola_UV = queue.LifoQueue()
cola_timestap = queue.LifoQueue()

# Bandera para controlar si el cliente MQTT est       conectado
mqtt_conectado = False

def on_connect(client, userdata, flags, rc):
    global mqtt_conectado
    if rc == 0:
        print("Conexi      n exitosa con el servidor MQTT")
        mqtt_conectado = True
    else:
        print(f"Fallo en la conexi      n con c      digo de retorno: {rc}")

def on_disconnect(client, userdata, rc):
    global mqtt_conectado
    print("Desconectado del servidor MQTT")
    mqtt_conectado = False

def conectar_mqtt():
    client = mqtt.Client()
    client.username_pw_set(ACCESS_TOKEN)
    client.on_connect = on_connect
    client.on_disconnect = on_disconnect
```

```
    try:
        client.connect(THINGSBOARD_HOST, 1883, 60)
        client.loop_start()  # Iniciar el bucle de eventos MQTT
    except Exception as e:
        print(f"Fallo en la conexi      n: {e}")

    return client

def leer_datos_sensores():
    global cola_temperatura
    global cola_UV
    global cola_timestap
    UV=ltr.uvs
    cola_UV.put(UV)
    temperature = sensor.get_temperature()
    cola_temperatura.put(temperature)
    timestap = datetime.now().isoformat()
    cola_timestap.put(timestap)
    print("Leyendo sensores")

def nube(client):
    global cola_temperatura
    global cola_UV
    global cola_timestap
    while not cola_temperatura.empty():
        if mqtt_conectado:
            temperatura = cola_temperatura.get()
            irradiancia = cola_UV.get()
            timestap = cola_timestap.get()
            # Crear un diccionario JSON con los datos y el timestamp en formato ISO 8601
            data = {
                "temperature": temperatura,
                "irradiance": irradiancia,
                "ts": timestap  # Agregar el timestamp en formato ISO 8601
            }
            try:
                client.publish(MQTT_TOPIC, json.dumps(data), 1)
                print("Enviando datos")
            except Exception as e:
                print(f"Fallo al enviar datos: {e}")
        else:
            client.reconnect()

# Funci      n para guardar datos en un archivo CSV
def guardar_en_csv():
    global cola_temperatura
    global cola_UV
    global cola_timestap
    timestamp = cola_timestap.get()
    temperatura = cola_temperatura.get()
    irradiancia = cola_UV.get()
```

```
    with open("datos_sensores.csv", "a", newline="") as csvfile:
        fieldnames = ["timestamp", "Temperatura", "UV"]
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        if csvfile.tell() == 0:
            writer.writeheader()  # Escribir encabezados solo si el archivo est      vac     o
        writer.writerow({"timestamp": timestamp, "Temperatura": temperatura, "UV": irradiancia})
    cola_UV.put(irradiancia)
    cola_temperatura.put(temperatura)
    cola_timestap.put(timestamp)

def check_internet_connection():
    try:
        # Intenta establecer una conexi      n con un servidor externo
        socket.create_connection(("www.google.com", 80))
        print("si hay conexion")
        return True
    except OSError:
        pass
    return False

def main():
    client = conectar_mqtt()

    while True:
        try:
            leer_datos_sensores()
            if check_internet_connection():
                nube(client)
            else:
                guardar_en_csv()

            time.sleep(10)
        except KeyboardInterrupt:
            print("Interrupcion de teclado detectada. El script se detiene.")
            client.disconnect()
            print("Programa detenido por el usuario.")
            break


if __name__ == "__main__":
    main()
```

o en el Git Hub
https://acortar.link/Proyecto-final-Embebidos

En este código desarrollamos 8 funciones, las cuales fueron:

- **On_connect:** Nos verifica si nos conectamos al servidor MQTT
- **On_disconnect:** Nos verifica si nos desconectamos del servidor MQTT
- **Conectar_mqtt:** Realiza la conexión al servidor MQTT

- **Leer_datos_sensores:** En esta función leemos los datos de ambos sensores, y en adición generamos el timestamp
- **Nube:** Aquí realizamos él envío, únicamente si estamos conectados al servidor MQTT, si no lo estamos realiza la reconexión al mismo. Además para el caso de envío de datos, se revisa si a cola de cualquiera de los sensores no está vacío, es decir, mientras la cola este llena, entrara en bucle de envío de datos, hasta desocuparla.
- **Guardar_en_csv:** Guarda los datos en el archivo csv
- **Check_internet_connection:** Verifica si estamos o no conectados a internet.
- **Main:** Esta es la función principal de nuestro programa, donde siempre estaremos leyendo los datos de los sensores, pero dependiendo de si estamos o no conectados a internet, se realizara el envío al servidor MQTT o se guardaran los datos localmente.

También el código cuenta con unas colas, las cuales fueron usadas para guardar los datos de los sensores y del timestamp, con la intención de que se fueran guardando todos los datos en la cola tipo LIFO y que al momento de sacar los mismos datos fueran los primeros en haber entrado y así lográramos tanto guardar como enviar los datos, sin necesidad de manipular los datos guardados.

Protocolo de pruebas.

- **Systemd**
  Para comprobar el funcionamiento de la automatización por systemd, se reiniciará la raspberry pi, y posteriormente cuando ya esté en funcionamiento se preguntará por el estado del servicio, en adición se mirará en el thingsboard si hay transmisión de datos.
- **Guardado local y reconexión**
  Para este apartado como ya tendremos el script corriendo por la automatización de systemd, lo que se hará será apagar el wifi para dejar sin conexión a la tarjeta, se dejara un tiempo sin esto, y después se encenderá otra vez el wifi, se mirara el archivo csv y posteriormente en la plataforma de la nube se observara si se realizó la transmisión de los datos nuevamente.
- **Funcionamiento total**
  Se dejará el dispositivo tomando datos y se verificaran en el dashboard que esté funcionando adecuadamente.

Resultados

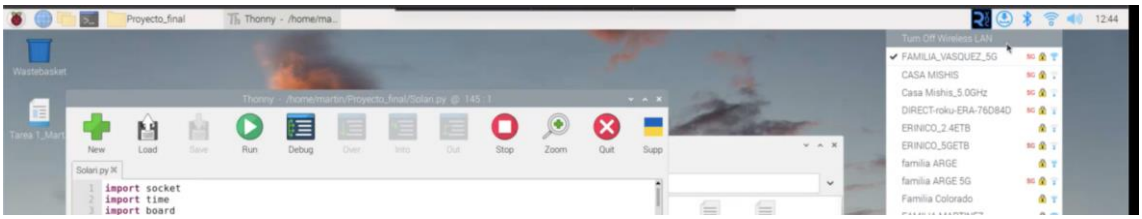
- **Systemd**
  Primero se realizó la conexión por ssh a la raspberry pi, ya con esto se reinició manualmente con sudo reboot, esperando que se reiniciara, se volvió a conectarse e inmediatamente se utilizó el comando ***sudo systemctl status mi_scrript.service,*** mostrando que está activo y en ese momento corriendo desde que se realizó el reinicio, obteniendo así que apenas la raspberry se inicie, nuestro programa se va a ejecutar.

```
martin@raspberrypi:~ $ sudo reboot
client_loop: send disconnect: Broken pipe
martinvasquez@MacBook-Pro-de-Martin ~ % ssh martin@192.168.1.132
Linux raspberrypi 6.1.21-v7+ #1642 SMP Mon Apr  3 17:20:52 BST 2023 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Nov 22 14:32:43 2023
martin@raspberrypi:~ $ sudo systemctl status mi_script.service
● mi_script.service - Mi script
     Loaded: loaded (/etc/systemd/system/mi_script.service; enabled; vendor pre>
     Active: active (running) since Wed 2023-11-22 14:32:52 -05; 1min 22s ago
   Main PID: 814 (python3)
      Tasks: 1 (limit: 1595)
        CPU: 2.637s
     CGroup: /system.slice/mi_script.service
```

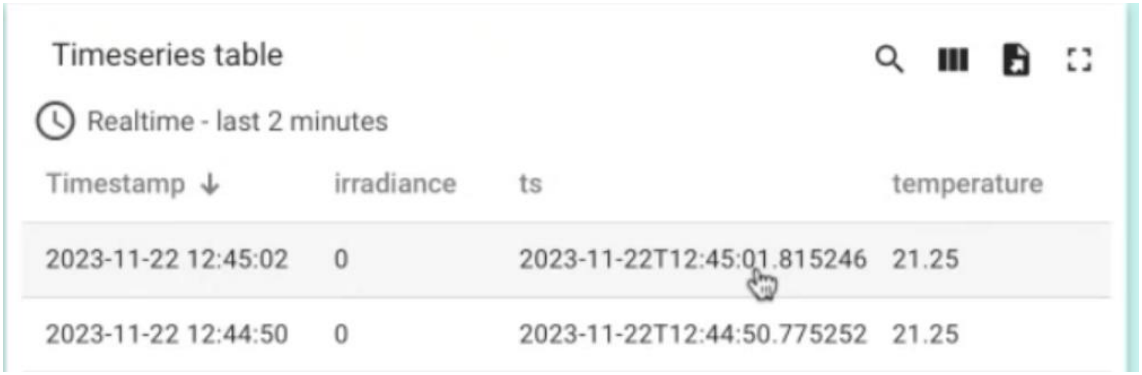
- **Guardado local y reconexión.**
  Primero se realiza el cat al archivo csv, mostrando que el archivo está vacío como se muestra a continuación:

```
martin@raspberrypi:~/Proyecto_final $ cat datos_sensores.csv
martin@raspberrypi:~/Proyecto_final $ cat datos_sensores.csv
```

Posteriormente apagamos el wifi de la tarjeta, dejándola así sin conexión como se muestra a continuación:



Mientras nuestro sistema opera sin conexión, observamos en thingsboard los últimos envíos, los cuales fueron a las 12:45:02 y 12:44:50.



Encendemos el wifi y tenemos que al momento de la conexión en thingsboard recibimos 2 datos al mismo tiempo a las 12:45:56 pero teniendo diferente timestamp, evidenciando que se enviaron los datos exitosamente que no se enviaron antes.

Timeseries table

Realtime - last 2 minutes

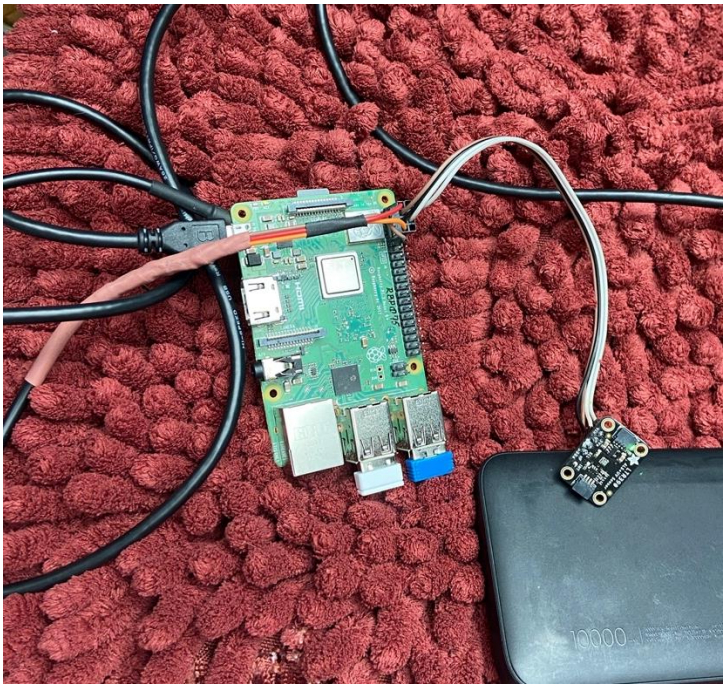
Timestamp ↓	irradiance	ts	temperature
2023-11-22 12:46:07	0	2023-11-22T12:46:07.335273	21.25
2023-11-22 12:45:56	0	2023-11-22T12:45:56.375241	21.25
2023-11-22 12:45:56	0	2023-11-22T12:45:34.615255	21.25
2023-11-22 12:45:02	0	2023-11-22T12:45:01.815246	21.25
2023-11-22 12:44:50	0	2023-11-22T12:44:50.775252	21.25

Por último, revisamos el archivo csv, para observar que, si se guardaron los datos localmente, mientras no tenía conexión a internet

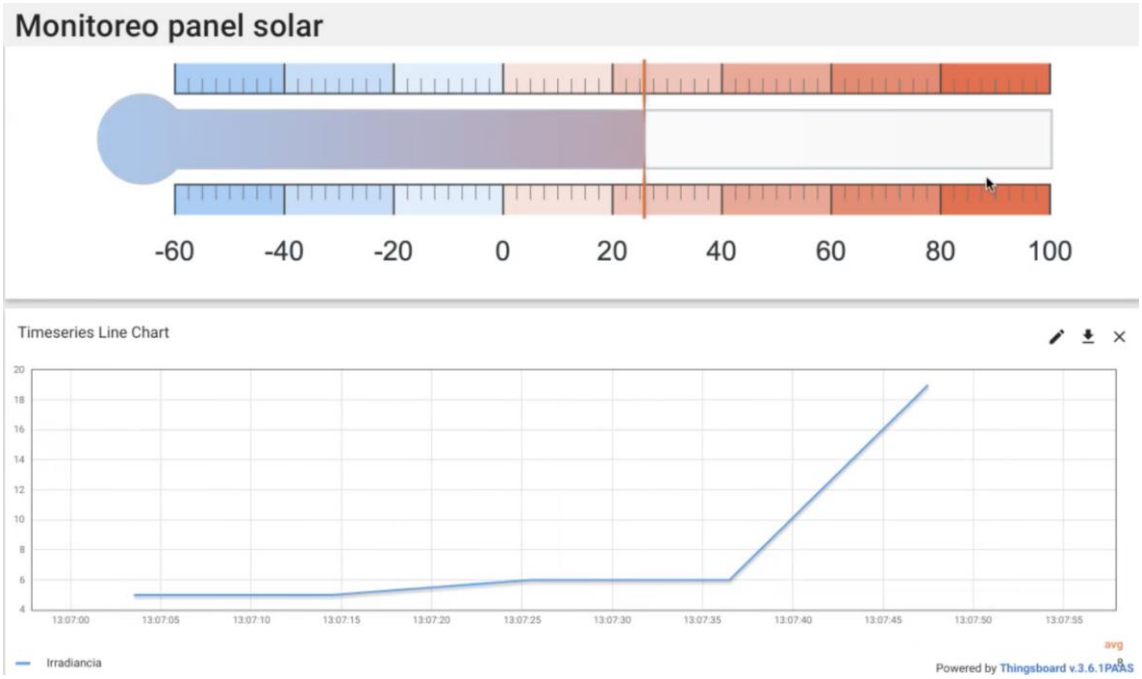
```
martin@raspberrypi:~/Proyecto_final $ cat datos_sensores.csv
timestamp,Temperatura,UV
2023-11-22T12:45:34.855397,21.3125,0
2023-11-22T12:45:56.735195,21.3125,0
```

- **Funcionamiento total.**

  Ya para mostrar funcionando totalmente el dispositivo, se dejó el dispositivo en un lugar tomando datos, también con el fin de poder recibir datos distintos a 0 en la irradiancia, como se muestra en la imagen:



Ya con nuestra tarjeta funcionando, lo que obtuvimos en nuestro dashboard fue lo siguiente:



Conclusiones

- La implementación de la comunicación de MQTT permite una transmisión eficiente y fácil de datos a cualquier servidor, en este caso thingsboard, pero brindando la facilidad de escalar e integrar el sistema a cualquier solución.
- La gestión de los datos tanto con conectividad como sin conectividad brinda una solución robusta, ya que no se pierde ningún dato sin importar la conexión a Internet.
- La reconexión automática nos permite tener la tranquilidad de que a pesar de que se pierda la conexión, el sistema intentara conectarse una y otra vez hasta que sea exitoso, sin alterar la toma de datos de los sensores.
- La implementación modular del código y la arquitectura del sistema resaltan la capacidad para adaptarse y escalar. Esto facilita futuras expansiones, permitiendo la incorporación de nuevos sensores o funcionalidades sin comprometer la estabilidad del sistema existente.
- La capacidad del sistema para cambiar sin problemas entre la transmisión en tiempo real y el almacenamiento local durante períodos de desconexión garantiza una gestión eficiente de los recursos. Esta característica es esencial en entornos donde la conectividad puede ser intermitente.
- La elección de utilizar Things Board como plataforma para el dashboard demuestra la integración efectiva de soluciones externas. Esto no solo simplifica el proceso de visualización de datos, sino que también permite aprovechar las características específicas de plataformas probadas y confiables.
- La aplicación de un protocolo de pruebas que aborda desde el inicio automático hasta la reconexión exitosa, valida la funcionalidad y fiabilidad del sistema. Este enfoque sistemático en las pruebas contribuye a la confianza en la operación del sistema en condiciones del mundo real.

- En cuanto a la propuesta en general se puede decir que, el enfoque en el monitoreo ambiental, particularmente en un contexto solar, anticipa una contribución significativa a la eficiencia energética. La capacidad para entender y optimizar el rendimiento de los paneles solares promete un impacto positivo en la gestión sostenible de la energía solar.