Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Diseño de Estación Meteorológica para una Planta Fotovoltaica

Autor: Diego Zamora Caballero

Tutor: Manuel Garrido Satué

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Diseño de Estación Meteorológica para una Planta Fotovoltaica

Autor:
Diego Zamora Caballero

Tutor:
Manuel Garrido Satué
Profesor Titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado: Diseño de Estación Meteorológica para una Planta Fotovoltaica

Autor: Diego Zamora Caballero
Tutor: Manuel Garrido Satué

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Gracias a todos los que me han apoyado, tanto los que han estado desde el principio como los que he conocido al final. Gracias a mi tutor, Manuel Garrido Satué, por darme la oportunidad de formar parte de este proyecto.

A mis padres Teresa y Constancio, a mi hermano y ejemplo Gonzalo, y a mi familia, en especial a mi abuela María Antonia.

A mis amigos.

A Dios.

Y a mi abuelo, que esta vez no podrá decirme que me lavé las manos como Pilatos.

Diego Zamora Caballero

Estudiante de Grado en Ingeniería de Tecnologías Industriales

Sevilla, 2024

Resumen

El propósito de este estudio es el de crear un sistema capaz de **recoger, analizar y compartir los datos** obtenidos por varios sensores **meteorológicos**, como serán un sensor de viento, un piranómetro, y un sensor de temperatura y humedad, para así poder trabajar posteriormente con la información aportada. A través de un Arduino Mega 2560, se recopilarán los datos reales de estos sensores, para después poder ser procesados en una Raspberry Pi 4, usando un lenguaje de programación en Python. Se establecerá una conexión cliente-servidor, entre la Raspberry y el equipo donde se quiera visualizar o trabajar con los datos, utilizando el protocolo de comunicación Modbus TCP/IP, permitiendo así a los usuarios el acceso a los valores meteorológicos recogidos, ya sea al momento o recopilados en un histórico. Además, se contará con una pantalla de tipo matriz LED, conectada directamente al Arduino, donde se mostrará la información meteorológica que se esté recogiendo en cada instante, para así contar con una visualización directa en la instalación. A partir de todas estas herramientas, se podrá afrontar el objetivo de este complejo sistema, y en un futuro, ser usado en distintas aplicaciones.

Abstract

The main purpose of this study is to create a system capable of collecting, analysing and sharing the data obtained by different meteorological sensors, such as a wind sensor, a pyranometer, and a temperature and humidity sensor, in order to be able to work with the information provided. Using an Arduino Mega 2560, the real data from these sensors will be collected and then processed on a Raspberry Pi 4, using a Python programming language. A client-server connection will be established between the Raspberry and the computer where the data is to be visualised or worked with, using the Modbus TCP/IP communication protocol, thereby allowing users to access the meteorological values collected, either at the moment or compiled in a data history. In addition, there will be an LED matrix type screen, connected directly to the Arduino, where the meteorological information being collected at any given moment will be displayed, in order to have a direct visualisation in the installation. With all these tools, the objective of this complex system can be met, and in the future, it can be used in different applications.

Índice

Índice de Figuras

Índice de Tablas

1 Introducción

A lo largo de los últimos años, la energía solar ha cobrado una suma importancia en el sector energético dentro de nuestro país, debido a su gran sostenibilidad y abundancia, pero sobre todo, debido al consumo masivo y al agotamiento de los combustibles fósiles. Con ello, el coste de las instalaciones fotovoltaicas se ha visto decrementado con el paso del tiempo, dando una mayor accesibilidad a esta tecnología. Según datos tomados del IRENA[1]: "los costes de las tecnologías renovables disminuyeron un 83 % en el caso de la solar fotovoltaica y un 42 % en el de la eólica terrestre" [26].

Sin embargo, esta fuente de energía requiere de un constante mantenimiento y supervisión, para obtener el máximo rendimiento posible de los paneles solares fotovoltaicos, a través de los cuales se capta la radiación solar. Para ello, es necesario conocer las distintas variables climatológicas que puedan afectar a dicho rendimiento, para así tener un control eficiente de las placas solares y lograr la máxima optimización de estos.

Figura 1.1 Estación Meteorológica Automática AWS310 de la marca *Vaisala* [7].

En la actualidad, existen estaciones meteorológicas, que recopilan la información de los diversos sensores que la componen. Dichos datos meteorológicos, permiten conocer con exactitud valores que afectan directa, o indirectamente, a los paneles solares, como la irradiancia solar, la temperatura o la dirección y la velocidad del viento. Con ello, también se logra una mayor exactitud a la hora de predecir la generación de energía, gracias a una constante monitorización de los datos, por lo que se consigue una gestión anticipada del suministro eléctrico frente a altas demandas de consumo.

[1] IRENA (‘International Renewable Energy Agency’): se trata de la Agencia Internacional de Energías Renovables, una organización intergubernamental que se dedica a promover la adopción y el uso sostenible de todo tipo de energías renovables.

1.1 Sistema Fotovoltaico

El sistema fotovoltaico para el cual se va a instalar una estación meteorológica se encuentra en la zona de los laboratorios de la Escuela Técnica Superior de Ingeniería, en Sevilla, más concretamente, en la azotea del Laboratorio 1, justo encima del departamento de Ingeniería de Sistemas y Automática. Dicho sistema, cuenta con dos robustas estructuras metálicas que soportan y mantienen las placas solares orientadas al sur, con una inclinación de 30°. Con un total de 72 paneles fotovoltaicos, de 420W cada uno, el sistema es capaz de generar una potencia superior a los 30kW. Obviamente, esto solo se daría un día soleado, durante unas pocas horas, y con unas condiciones climáticas estándar, como sería, por ejemplo, una temperatura de 25°C.

Figura 1.2 Sistema Fotovoltaico para el cual se requiere una Estación Meteorológica.

En la figura superior, se puede observar una imagen del sistema completo. Cabe destacar, que aun quedan algunas zonas entre placas donde se podrían colocar más paneles, y una parte de la estructura aun vacía. Además, en la parte inferior izquierda, se tiene una pequeña caseta, en la cual, junto con otros equipos, se colocarán los dispositivos para tratar los datos meteorológicos que se reciban de los sensores, los cuales se situarán a una altura similar a la zona más elevada de las placas fotovoltaicas.

1.2 Objetivos

El objetivo principal que se busca conseguir con este estudio es la monitorización de los distintos datos meteorológicos que pueden afectar de forma significativa al rendimiento de este sistema fotovoltaico, de forma que dicha información sea accesible para cualquier usuario.

Para ello, el sistema se compone de distintas funcionalidades diseñadas para el estudio completo de la información meteorológica recopilada, las cuales funcionan de forma paralela:

- **Acceso instantáneo por parte de un PLC a la información meteorológica a través de una conexión Modbus TCP:**

  Si bien hoy en día se cuenta con diversos métodos fácilmente accesibles para conocer datos meteorológicos, tales como la temperatura o la velocidad del viento, a pesar de su gran exactitud, dicha información ha sido tomada o estimada dentro de un área en concreto. El hecho de poder colocar una estación meteorológica acompañando directamente al sistema fotovoltaico para alcanzar el máximo rendimiento de los paneles solares permite lograr una máxima precisión, por lo que se ha realizado un sistema capaz de devolver dicha información cuando se le pida. La estación utilizada enviará las medidas tomadas por los diferentes sensores a un microcontrolador Arduino, que tratará las distintas señales recibidas y las enviará por puerto serie a una Raspberry Pi. Para establecer la transmisión de información se ha diseñado una comunicación Modbus TCP entre la Raspberry Pi y un PLC que realiza peticiones, creando por tanto una conexión servidor-cliente.

- **Acceso a históricos de datos meteorológicos:**

  La segunda funcionalidad principal incluida en el sistema comprende la recopilación de toda la información tomada, de forma que en el futuro se tenga acceso a ella, para las fechas que sea requerida. Esto, permitiría diseñar un SCADA capaz de controlar todo el sistema fotovoltaico (paneles, baterías, inversores...), logrando así una mayor eficiencia y un menor coste económico, ya que se podría preveer en que momentos es más adecuado el almacenamiento de la energía en las baterías, gestionar el mantenimiento de las placas, si se cuenta con un sensor para monitorizar la suciedad en estas, o simplemente estimar que necesidad de energía de la red haría falta, según la producción eléctrica de los panales y el consumo de las instalaciones que este alimentando. Para ello, se han diseñado las dos siguientes funciones dentro del sistema:

  - **Función de almacenamiento de históricos:** de forma constante, se ejecuta un almacenamiento de los datos meteorológicos enviados desde el Arduino dentro de una base de datos SQL, a través de la Raspberry Pi. De esta manera, la estación meteorológica actuará como un *data-logger* con la información recopilada.

  - **Función de transmisión de históricos:** tras almacenar la información, para poder acceder a esta en un futuro, se ejecutará, mediante un protocolo TCP/IP, un segundo servidor, capaz de crear una conexión con un PC u otro equipo que le realice una solicitud, y transmitir los históricos ya formateados como CSV ('Comma Separated Variables'). De esta forma, el cliente podrá guardar los datos dentro de, por ejemplo, un archivo de texto.

- **Visualización de medidas instantáneas en una matriz LED:**

  Por último, se ha utilizado una pantalla LED para mostrar la información meteorológica instantánea, que puede ser colocada dentro del departamento para su visualización directa por parte de cualquier persona. Esta pantalla se conectará a Arduino, por lo que la placa le estará enviando datos de forma constante, para que así, el panel LED muestre de forma independiente cada medida tomada.

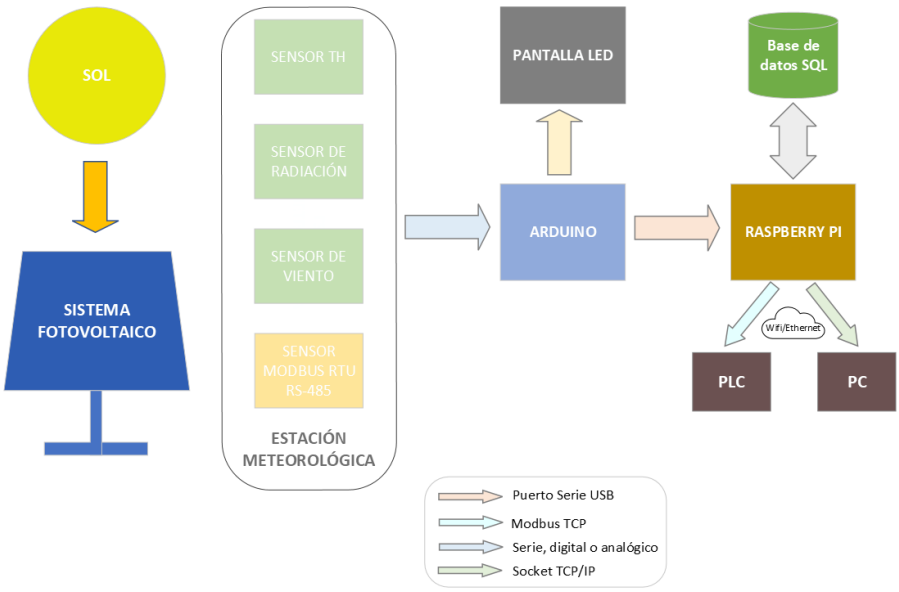

Figura 1.3 Diagrama general del sistema.

1.3 Estructura del documento

El documento se compone de los siguientes capítulos, en los cuales se desarrollan tanto aspectos esenciales utilizados para la realización del proyecto, como pruebas realizadas, resultados y conclusiones obtenidas:

- **Capítulo 1:** Se compone de una breve introducción a las estaciones meteorológicas y al sistema fotovoltaico instalado, junto con los propósitos que tiene este proyecto, además de la estructura básica que sigue este documento.
- **Capítulo 2:** Incluye todos los elementos relacionados con el software y las comunicaciones utilizadas, con distintas descripciones de cada elemento y explicaciones sobre el funcionamiento de estos.
- **Capítulo 3:** Reúne el conjunto de equipos y componentes electrónicos que forman parte del hardware del sistema. Además, también se incluyen pequeños módulos necesarios para adaptar todos los elementos que no estaban preparados en una primera instancia para ser añadidos directamente a los equipos.
- **Capítulo 4:** En este capítulo se recogen las distintas conexiones entre los diferentes equipos del proyecto. Con ello, se incluyen explicaciones gráficas e indicaciones a seguir.
- **Capítulo 5:** En este capítulo se desarrolla la arquitectura del proyecto y se entra en detalle en la implementación de este, teniendo en cuenta todos los componentes explicados previamente, y aportando los resultados obtenidos de distintas pruebas realizadas para comprobar su correcto funcionamiento.
- **Capítulo 6:** En este último capítulo del documento, se recogen las conclusiones obtenidas en base a los resultados aportados tras el desarrollo del proyecto. Además, el esfuerzo de este escrito reside en su futuro uso, por lo que también se exponen los siguientes pasos a realizar para darle funcionalidad real a este proyecto.

2 Software del Sistema

En este Capítulo se van a resumir los distintos lenguajes de programación, protocolos de comunicación y recursos utilizados, para la elaboración del software que compone el proyecto. El orden de los distintos puntos desarrollados viene dado en concordancia con respecto al orden de transmisión de un dato meteorológico, desde que se envía como salida desde un sensor, hasta que un PLC o un PC lo recibe.

2.1 Lenguajes de Programación

2.1.1 Arduino

Arduino es una plataforma electrónica basada en hardware y software muy simple de manejar, de código abierto, y utilizada desde hace años en una multitud de proyectos debido a su cantidad de ventajas. Utiliza el lenguaje de programación *Arduino*, el cual se basa en *Wiring*, y el software Arduino(IDE), el cual se basa en *Processing*:

- **Wiring**: es una plataforma que nos permite programar, generar prototipos con electrónica y controlar varios dispositivos conectados a un microcontrolador. Además, su *Sketch* tiene la misma estructura que Arduino, con una función *setup* y otra función *loop*.
- **Processing**: es un lenguaje de programación de código abierto y entorno de desarrollo basado en Java. Su estructura, aunque en este caso no es igual a Arduino, es bastante similar, pues también cuenta con una función *setup*, y en vez de la función que se ejecuta en bucle *loop*, tiene una función llamada *draw* conla misma funcionalidad.

Por otro lado, entre las múltiples ventajas que presenta Arduino a nivel general, pero también a nivel del proyecto, por las cuales ha sido el entorno elegido para su función en este trabajo, caben destacar las más importantes:

- **Precio:** las placas Arduino son relativamente baratas en comparación con otras plataformas de microcontroladores, y, además, se pueden montar fácilmente a mano.
- **Multiplataforma:** a pesar de que la mayoría de los sistemas de microcontroladores suelen estar limitados a Windows, el software Arduino (IDE) funciona en varios sistemas operativos, como son Windows, Macintosh OSX o Linux.

- **Entorno de programación sencillo y claro:** desde principiantes a profesionales, el software Arduino (IDE) es tan adaptable, que cualquier persona ajena que lo utilicé puede progresar rápidamente, y a su vez también puede ser utilizada por usuarios que ya posean conocimientos avanzados en el entorno, para que puedan desarrollar complejos programas de mayor nivel.
- **Software de código abierto y extensible:** El software Arduino se publica como herramientas de código abierto, por lo que está disponible para ser ampliado mediante librerías C++ por programadores experimentados.
- **Hardware de código abierto y extensible:** ya que los planos de las placas Arduino se publican bajo licencia Creative Commons [22], los diseñadores de circuitos experimentados pueden hacer su propia versión del módulo de las placas de Arduino, ampliándolo y mejorándolo a su disposicón [19].

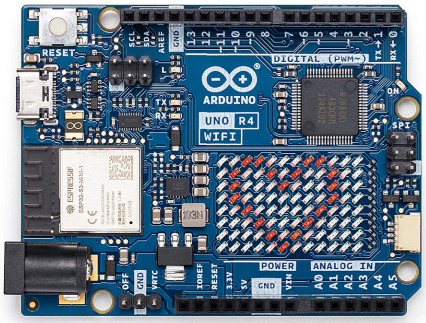

Figura 2.1 Arduino® UNO R4 WiFi [4].

Además, las placas Arduino disponen de salidas analógicas y digitales, siendo también capaces de leer entradas tanto analógicas como digitales, gracias una serie de instrucciones programadas por el microcontrolador de la placa. En este proyecto, se leen como entradas las salidas de los respectivos sensores, que tras ser procesadas en Arduino, corresponden a la información meteorológica específica de cada equipo en el tiempo medido. Posteriormente, desde Arduino, se imprimirá la información recibida para que pueda ser leída por un usuario.

Por último es importante mencionar que para este proyecto se ha ampliado el entorno de Arduino gracias a distintas **librerías**, las cuales proporcionan una nueva funcionalidad dentro del programa creado, y se han incluido tanto librerías que venían instaladas con el IDE, como otras que han sido creadas directamente por terceros. A continuación se describen las que han sido utilizadas:

- ***DHT***: permite trabajar con distintos sensores de temperatura y humedad, en este caso el DHT11.
- ***string* y *stdlib***: *string* se utiliza para manejar cadenas de caracteres y *stdlib* es una biblioteca estándar para funciones de propósito general,
- ***RGBmatrixPanel*, *bit-bmp* y *fonts***: utilizadas para poder trabajar con una pantalla LED, que permite la visualización directa de los datos.
- ***ModbusMaster***: permite implementar el elemento del maestro durante un protocolo de comunicación Modbus RTU.

2.1.2 Python

Python es un lenguaje de programación interpretado, que está orientado a objetos, de un alto nivel y con semántica dinámica. De una sencilla legibilidad, gracias una sintaxis fácil de aprender, por lo que favorece el coste de mantenimiento de los programas. Se caracteriza por unas estructuras de datos incorporadas de alto nivel, que, combinadas con la tipificación dinámica y la vinculación dinámica, permiten un rápido desarrollo de aplicaciones, y además, de poder usarse como lenguaje de *scripting* o pegamento, lo que permite conectar componentes existentes entre sí.

Cuenta con una biblioteca estándar de gran tamaño tanto en formato fuente como en binario, que junto al intérprete de Python, son completamente gratuitos, y de libre distribución.

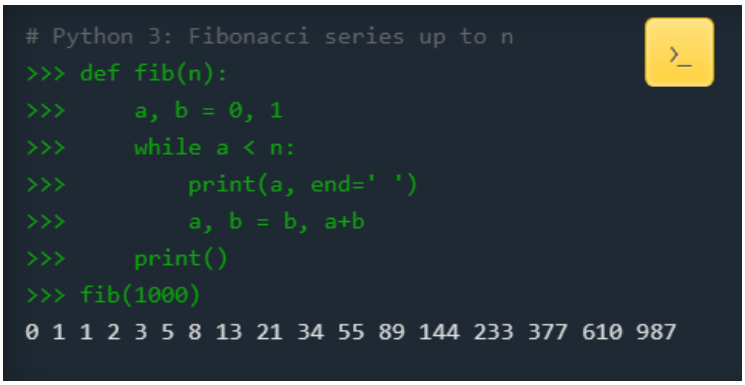

Figura 2.2 Ejemplo de programa en *Python* de la *Sucesión de Fibonacci* [11].

De nuevo, es importante mencionar como se ha ampliado el entorno de *Python*, tal y como previamente se hizo en *Arduino* con librerías, pero en este caso con los llamados **Módulos y Paquetes**, siendo los módulos ficheros *Python*, y los paquetes conjuntos de módulos [30]. Los que han sido incluidos en este proyecto son:

- ***serial***: permite la comunicación por puertos series, en este caso con el Arduino.
- ***sqlite***: crea una interfaz para trabajar con bases de datos SQL, permitiendo almacenar distintos datos.
- ***pyModbusTCP***: permite implementar un servidor modbus TCP.
- ***socket***: permite la creación de conexiones de red a través de la comunicación entre distintos dispositivos mediante sockets.
- ***threading***: permite la ejecución de tareas concurrentes y creación de múltiples hilos (threads) de manera muy simple.
- ***time***: permite trabajar con multiples funciones relacionadas con obtención del tiempo, como pausas u obtención de la hora actual.
- ***json***: habilita la posibilidad de trabajar con datos en formato JSON, muy útil para la transmisión de gran cantidad de información.

2.2 Protocolos de comunicación

A la hora de transmitir datos entre distintos dispositivos, es importante establecer unas normas y reglas, denominadas protocolos de comunicación, que, básicamente, determinen la forma de recibir y transmitir los datos, logrando así una comunicación eficiente y sin fallos. A día de hoy, existen distintos protocolos de comunicación para la transmisión correcta de datos, aunque para este proyecto, solo se ha utilizado **Modbus**

2.2.1 Modbus

Actualmente, Modbus es uno de los protocolos de comunicación más común en aspectos industriales y sistemas de monitorización, utilizándose tanto en redes con pequeños elementos, como en actuadores, controladores y sistema SCADA de gran complejidad. Fue desarrollado y publicado por Modicon[1] en 1979, y a partir de 2004, paso a formar parte de una comunidad de usuarios y fabricantes conocido como Modbus-IDA[2]. Se sitúa en el nivel 7 del Modelo de Referencia OSI[3].

Modbus se utiliza para comunicar dispositivos inteligentes y dispositivos de campo, creando una comunicación cliente/servidor o maestro/esclavo.

Modbus proporciona un medio por el cual un dispositivo puede leer y escribir datos en zonas de memoria concretas de otros dispositivos remotos. Por un lado,un cliente (o maestro) puede realizar varias peticiones a distintos servidores (o esclavos), y por otro, un servidor podrá responder a varias peticiones de los distintos clientes. Sin embargo, que servidor logre procesar las peticiones recibidas de forma paralela dependerá del propio servidor, aunque lo más habitual es que las peticiones se procesen una detrás de otra, según como entren en la llamada cola de peticiones, donde se van recogiendo según le lleguen al servidor.

Este protocolo de comunicación, ha sido elegido para usarse en este proyecto debido a sus múltiples ventajas:

- La documentación del protocolo es pública y gratuita, por lo que no ha sido necesario ningún gasto económico para desarrollar esta parte.
- No incurre en ningún hardware específico para su implementación
- Es un protocolo sencillo.
- Cuenta con librerías para su uso tanto en Python como en Arduino, ambos lenguajes de programación incluidos en el sistema.

A pesar de ello, este protocolo de comunicación tiene distintos inconvenientes, que aunque para sistemas más complejos si pueden ser perjudiciales, no han sido relevantes para este proyecto:

---

[1] Modicon: es una marca de automatización industrial que desarrolló el primer controlador lógico programable (PLC). Actualmente pertenece a Schneider Electric.

[2] Modbus-IDA ('Industrial Device Association') es una organización creada sin ánimo de lucro, que se encarga de la estandarización de los protocolos Modbus, para asegurar que se implemente manera efectiva en equipos de automatización industrial.

[3] Modelo de Referencia OSI: creada por la Organización Internacional de Normalización (ISO) en 1984, es una estandarización de las distintas funciones de una red de telecomunicaciones. Básicamente, divide el proceso de comunicación en siete capas, cada una con funciones específicas y con interfaces definidas, y cada una se construye sobre su capa inferior. Estas capas, de forma ordenada de la primera a la séptima son: Capa Física, Capa de Enlace, Capa de Red, Capa de Transporte, Capa de Sesión, Capa de Presentación y Capa de Aplicación

- No ofrece seguridad contra órdenes no autorizadas o interceptación de datos
- No está pensado para transmitir grandes volúmenes de datos, ya que el tipo y tamaño de los datos que maneja son los disponibles en los PLCs de finales de los años 70.
- No permite la configuración de los dispositivos de la red. Hay que asignar las direcciones manualmente.
- No permite crear variables de red o crear direcciones de grupo [24].

Modbus TCP/IP

Modbus TCP es un protocolo de comunicación Servidor-Cliente, que se establece a través de una red Ethernet TCP/IP, en cual están conectados múltiples dispositivos.

En esta comunicación, un cliente envía una petición a un servidor, el cual le responde con los datos solicitados, enviando tramas Modbus TCP a través de una conexión TCP previamente establecida entre el cliente y el servidor. Para ello, el servidor Modbus escucha a través del puerto 502, el cual está reservado para aplicaciones Modbus, las distintas peticiones de los clientes que desean establecer una nueva conexión con el servidor.

En Modbus TCP, para establecer la comunicación, es necesario para la parte del servidor, el ID y número de puerto del servidor, y para el cliente, la dirección IP del servidor, la ID del cliente y el número de puerto.

La trama de Modbus TCP consiste en una cabecera MBAP ('ModBus Application Protocol') y una capa de aplicación APDU. La cabecera MBAP está formada por los siguientes campos:

- Identificador de transacción. Para relacionar las tramas de petición y respuesta.
- Identificador de protocolo. Siempre tiene el valor cero.
- Longitud. Indica el tamaño en bytes del campo identificador unitario y el APDU.
- Identificador unitario. Es la dirección del dispositivo destino (un esclavo).

Este protocolo de comunicación ha sido el elegido para establecer la transmisión de los datos meteorológicos entre la Raspberry y el usuario que los solicite, siendo por tanto, un factor clave en el proyecto.

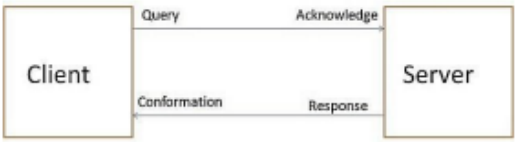

Figura 2.3 Ciclo de mensajes en una comunicación Modbus TCP [29].

Modbus RTU

Modbus RTU es un protocolo de comunicación serie abierta punto a punto. Se utiliza para desarrollar una comunicación Maestro-Esclavo entre dispositivos inteligentes. En Modbus RTU, los dispositivos maestros envían una consulta a los dispositivos esclavos y éstos envían una respuesta a la consulta del maestro de acuerdo con el código especificado.

Como capa física, en Modbus RTU se pueden utilizar RS-232, topología dúplex para transmitir y recibir datos al mismo tiempo, o RS-485, topología en la que los procesos de transmisión y recepción se realizan uno tras otro. Además, estas capas físicas también pueden diferenciarse en función de la velocidad en baudios y la velocidad de transporte de datos. Además, la capa física es responsable de la dirección del esclavo, el bit de inicio, el bit de parada en los datos, el código CRC, el tiempo de espera y la detección de errores de encuadre, mientras que la capa de enlace de datos es responsable del reconocimiento o rechazo del código de función y la reagrupación de datos.

Para este sistema, como se verá más adelante, va a ser necesaria la utilización de una comunicación Modbus RTU, a través de la capa física RS-485, para la transmisión de datos entre un sensor específico y el Arduino, trabajando así en este proyecto con los dos principales protocolos de comunicación Modbus.

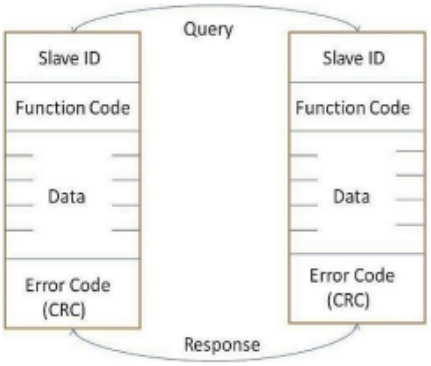

Figura 2.4 Ciclo de mensajes en una comunicación Modbus RTU [29].

2.3 SQLite

SQLite es una biblioteca en lenguaje C que implementa el motor de una base de datos **SQL**[4] pequeño, rápido, autónomo, de alta fiabilidad y que no requiere de un servidor independiente . SQLite es el motor de bases de datos más utilizado del mundo, de tal manera, que está integrado en todos los teléfonos móviles y en la mayoría de los ordenadores, e incluso en aplicaciones y sistemas operativos que se utilizan a diario, como Android, iOS y Windows, Skype y en navegadores como Google Chrome o Mozilla Firefox.

El código de SQLite es de dominio público y, por tanto, de uso libre para todo el mundo. Además, el formato de archivo SQLite es estable, multiplataforma y compatible con versiones anteriores, y los desarrolladores se comprometen a mantenerlo así hasta el año 2050 [6].

[4] SQL: es un lenguaje de computación que sirve para trabajar con conjuntos de datos y relacionarlos entre ellos, creado una base de datos. Es un lenguaje fácil de leer y entender, disponible, entre otros, en Python.

Figura 2.5 SQLite es el motor de bases de datos más utilizado del mundo.

Para este sistema, se va a hacer uso de uno base de datos SQL, dentro de la Raspberry Pi, por lo que se creará en Python. Para ello, se hará uso del módulo *sqlite3*, el cual proporciona una interfaz SQL [18].

2.4 Sockets y Concurrencia de Hilos

Los **sockets** son un medio de comunicación entre distintos programas, que pueden estar situados en distintos PC, pero se encuentran en la misma red. Un socket se define con la dirección IP del equipo donde se ejcuta, el puerto en el que escucha, y el protocolo que utiliza.

Existen dos tipos de sockets: sockets de flujo (*STREAM*)) orientados a conexión y sockets de datagramas (*DGRAM*) no orientados a conexión. Esto depende de si el sevicio utiliza TCP, que es orientado a conexión y fiable, o UDP, respectivamente. Los sockets también se pueden clasificar según la familia de direcciones, pudiendo ser *AF-UNIX* (interconexión local tipo UNIX) o *AF-INET* (TCP/IP). Lo más habitual es usar la familia *AF-INET* y el tipo *STREAM*, y así es como se utilizará en este proyecto.

En python, todas las funciones necesarias para ejecutar un socket se recogen en el módulo *socket*, y su uso es realmente sencillo. Además, una vez es prácticamente como realizar una comunicación Modbus TCP, aunque con distintas funciones.

En cuanto a la **concurrencia de hilos**, esto sucede cuando un proceso posee distintas actividades concurrentes, que comparten el tiempo del procesador existente del sistema, y cada una de ellas se denomina hilo, siendo este por tanto una especie de subproceso. Los hilos son una alternativa más simple a usar distintos procesos para realizar actividades concurrentes, pues la comunicación entre las actividades es más simple, ya que se trabaja con variables globales compartidas, y también es más sencilla la descripción de cada actividad desde el punto de vista del sistema operativo, por lo que el cambio de contexto entre hilos es menos costoso que si fuese entre procesos. Se consigue además, más eficiencia, ya que el coste de concurrencia es menor.

Por otra parte, en python, al igual que los sockets, todas las funciones necesarias para ejecutar un hilo se recogen en un módulo denominado con su nombre en inglés *thread*, y su uso también es muy sencillo. Como se explicará más adelante, la concurrencia de múltiples hilos, sincronizados por exclusión mutua a la hora de trabajar cada uno con variables globales, ha sido la clave para unificar en un solo programa los distintos códigos diseñados para recopilar la información de los sensores, guardarla en un histórico de datos, y poder transferirla de forma instantánea o como conjunto a un PC o un PLC [23].

3 Elección de Componentes

En este Capítulo se van a resumir los distintos lenguajes de programación, protocolos de comunicación y recursos utilizados, para la elaboración del software que compone el proyecto. El orden de los distintos puntos desarrollados viene dado en concordancia con respecto al orden de transmisión de un dato meteorológico, desde que se envía como salida desde un sensor, hasta que un PLC o un usuario lo recibe.

3.1 Placa de Arduino

Existen distintas placas de Arduino, según diversos factores, por lo que para cada proyecto en específico se adecuará mejor una u otra. Para este sistema, a pesar de su sencillez a la hora del conexionado de sensores, como se mencionó previamente, se utiliza una matriz de LEDs a modo de pantalla, lo que va a suponer la necesidad de usar una gran cantidad de entradas y salidas. A continuación se muestran los dos tipos de placas que se han utilizado durante el periodo de pruebas, y se detalla la escogida:

3.1.1 Arduino UNO

Arduino UNO es una placa microcontroladora basada en el *ATmega328P*, el cual es un microcontrolador de un solo chip basado en un núcleo de procesador RISC de 8 bits, de bajo consumo y alto rendimiento que puede ejecutar instrucciones en un solo ciclo de reloj. Esta placa se compone de:

- Catorce (14) pines digitales de entrada/salida, de los cuales 6 se pueden utilizar como salidas *PWM*.
- Seis (6) entradas analógicas.
- Un (1) resonador cerámico de 16 MHz.
- Una (1) conexión USB, para conectase a un ordenador.
- Un (1) conector de alimentación, que puede ser alimentado con una batería (la más óptima, de 9V).
- Un (1) adaptador de CA o CC: con una entrada de 7-12V.

- Un (1) cabezal ICSP.
- Un (1) botón de reinicio (‘*reset*’).

Esta placa microcontrolador, presenta múltiples ventajas electrónicas:

- **Chip reemplazable**: El *ATmega328P* puede sustituirse fácilmente, ya que no está soldado a la placa.
- **Microcontrolador EEPROM**: El *ATmega328P* también dispone de 1kb de EEPROM, una memoria que no se borra cuando se apaga.
- **Alimentación con conector de batería**: El *Arduino UNO* dispone de un conector de barril que funciona perfectamente con una pila estándar de 9V.

En un primer lugar, esta placa básica fue la elegida como componente del sistema, ya que con ella se podía realizar las pruebas con todos los sensores. Sin embargo, como se mencionó anteriormente, la adición de una pantalla LED al proyecto hizo reconsiderar su uso, y dio paso a la utilización de otro tipo de placa.

3.1.2 Arduino MEGA 2560

El Arduino MEGA 2560 está diseñado para proyectos que requieren un mayor número de líneas de Entradas/Salidas, más memoria de *sketch* y más RAM. Con 54 pines de E/S digitales, 16 entradas analógicas y más espacio para su *skecth*, es la tarjeta elegida para impresoras 3D y proyectos de robots. Esto le da a los proyectos mucho espacio y posibilidades, manteniendo siempre la simplicidad y eficiencia de la plataforma Arduino, pues esta placa también se programa utilizando el Software Arduino (IDE).

Sin entrar más detalle, cuenta con los principales elementos y las distintas ventajas explicadas del Arduino UNO, obviando las relacionadas con las Entradas y Salidas del sistema. Además, su microcontrolador EEPROM es de 4kb.

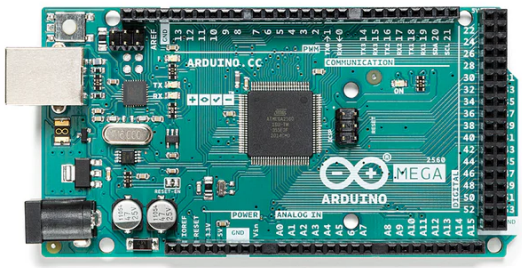

Figura 3.1 Arduino Mega 2560 [3].

Por otro lado, y aunque finalmente no ha sido restrictivo para el sistema, como se verá más adelante, la comunicación realizada entre el Arduino y la Raspberry, ha sido serie, es decir, por puerto serie. Esto, en un principio, con la placa siendo alimentada, a través de un conexión USB,directamente por el PC donde se programaba, era inviable con un Arduino Uno si la conexión se hacía por los terminales serie de la placa (Tx-Rx), ya que es el único puerto hardware serie (UART) que posee dicha placa, y por tanto la información se trasladaría al ordenador, y no a la Raspberry.

Sin embargo, el hecho de utilizar un Arduino Mega, el cual cuenta con cuatro puertos serie, solucionaba el problema pues se podía programar a la placa para que imprimiera la información por el puerto específico conectado a la Raspberry. A pesar de esto, como se verá en el capítulo siguiente, la conexión final entre Arduino y Raspberry fue por USB, por lo que un único puerto hubiese bastado.

En conclusión del apartado, esta ha sido la placa escogida para el proyecto, ya que era la que tenía suficiente número de E/S como para incluir todos los elementos correspondientes.

3.2 Raspberry Pi

Raspberry Pi es un ordenador, del tamaño de una tarjeta, completamente funcional, el cual dispone de un microporocesador con arquitectura ARM, memoria RAM y tarjeta gráfica en un solo chip. No incluye disco duro, por lo que utiliza una tarjeta SD como almacenamiento interno. Además, tampoco cuenta con fuente de alimentación, por lo que siempre tiene que contar con una fuente externa [20].

Según la página oficial de Raspberry, donde se puede encontrar toda la información necesaria para empezar a trabajar con este *ordenador*:

> "Raspberry Pi persigue ayudar en la enseñanza y mejora de las habilidades de la programación en lenguajes como Scratch y Python así como la utilización de sistemas operativos de distribuciones de Linux. Aparte, Raspberry Pi tiene la habilidad de interactuar con el mundo exterior y ha sido utilizado en numerosos proyectos, desde la música, detectores en estaciones meteorológicas, con cámaras infrarrojas para grabación nocturna, robótica, monedas criptográficas o incluso construir una miniconsola o un robot mayordomo. El principal objetivo es que la placa consiga ser utilizada por los jóvenes para el aprendizaje a la programación y entender así cómo trabajan los ordenadores"[14].

Aunque existe un modelo más actual de este *ordenador*, para este proyecto se ha escogido la **Raspberry Pi 4 Model B**. Este modelo destaca por ser el primero en incorporar la posibilidad de manejarse en una doble pantalla, con resoluciones de hasta 4K, a través de un par de puertos micro HDMI.

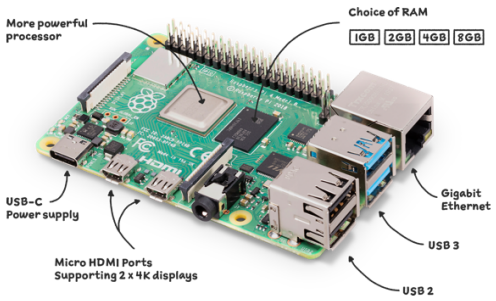

Figura 3.2 Raspberry Pi 4 Model B [12].

Sin entrar en detalles muy técnicos, en la siguiente tabla se recogen las principales especificaciones del modelo utilizado:

Tabla 3.1 Especificaciones técnicas de una Raspberry Pi 4 Model B [13].

Procesador	Broadcom BCM2711
Memoria	4GB LPDDR4 RAM
Tarjeta SD	Tarjeta micro SD card slot para el almacenamiento de datos, tras cargar el sistema operativo
GPIO	Estándar: 40 pines GPIO
Conectividad	2.4 GHz y 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, Gigabit Ethernet, 2 puertos USB 3.0, 2 puertos USB 2.0.
Vídeo y sonido	2 puertos micro HDMI, 2 carriles para puerto de un display MIPI DSI, otros dos para puerto cámara MIPI CSI camera port, un puerto de audio y vídeo stereo de 4 polos
Alimentación	5V DC conector USB-C (corriente min. 3A), 5V DC por pines GPIO (corriente min. 3A) o por alimentación a través de Ethernet

3.3 Pantalla LED

Aunque el principal objetivo de este proyecto sea la monitorización de los datos meteorológicos recogidos, junto con sus históricos correspondientes, se ha ampliado sencillamente el sistema incluyendo una pantalla LED, que permite una visualización directa de los datos, de forma instantánea.

Para ello, se ha utilizado un panel con 2048 LEDs integrados, específicamente un **RGB-Matrix-P4-64x32** de la marca *Waveshare*, el cual es compatible tanto con Raspberry Pi, como con Arduino, aunque en este sistema, irá directamente conectado al Arduino Mega 2560 [9].

Figura 3.3 Pantalla LED RGB-Matrix-P4-64x32 [9].

Además, dispone de unos anclajes imantados, lo cual permite su fácil colocación en un cualquier estructura metálica. Se recogen a continuación, en la siguiente tabla, las principales especificaciones de diseño del panel:

Tabla 3.2 Especificaciones del panel LED RGB-Matrix-P4-64x32 [9].

Dimensiones	256mm x 128mm
Píxels	64 x 32 = 2048 DOTS
Tamaño de píxel	4 x 4mm^2
Formato de píxel	RGB
Alimentación	5V DC/ 4A
Consumo	$\leq 20W$

3.4 Sensor de Temperatura y Humedad

La temperatura, junto con la irradiancia, es el parámetro meteorológico que más afecta al rendimiento de las placas meteorológicas. A la hora de elegir la ubicación para un parque fotovoltaico o similar, es importante conocer los niveles de irradiancia y temperatura de dicho lugar, antes de realizar la instalación de un sistema fotovoltaico. Frente a irradiancias parecidas, predomina el emplazamiento de menor temperatura, teniendo siempre de referencia un estándar de 25°C, la cual es la considerada temperatura ideal para las placas fotovoltaicas [27].

Por ello, para esta estación meteorológica se ha incluido un sensor de temperatura y humedad. Este segundo parámetro es de menor relevancia, pero supone una ventaja para la supervisión de los paneles, pues un clima continuamente seco podría implicar un aumento de suciedad en las placas, mientras que la lluvia y la humedad en el ambiente pueden limpiar la superficie de los paneles.

El sensor utilizado es un módulo del sensor **DHT11**, diseñado para establecer una rápida y sencilla comunicación con un Arduino, por lo que, además de ser el sensor de temperatura y humedad de la estación meteorológica dispuesta, ha sido el principal sensor para la realización de pruebas durante el desarrollo y la creación del sistema, pues su falta de complejidad evitaba que se diera problemas en su comunicación con la placa.

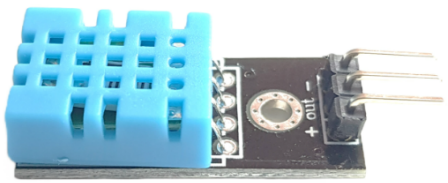

Figura 3.4 Módulo del Sensor de Temperatura y Humedad DHT11 [16].

Este módulo está adaptado de forma que su conexión es muy sencilla, pues se forma de tres pines que lo facilitan: el pin izquierdo Vcc, para conexión a la alimentación del Arduino; su pin central, que corresponde a la salida de datos serie de la temperatura y la humedad; y el último pin, GND, que se conecta a la tierra de la placa.

Sin duda, a pesar de que su sencillez es una ventaja, tiene como inconveniente unas especificaciones muy limitadas. Sin embargo, para el uso elegido y el emplazamiento donde se va a ubicar, no va

a suponer una pérdida de información por falta de rango o de precisión, aunque sería recomendable a futuro que sea remplazado por otro de mejores características. A continuación se muestran en la tabla adjunta las principales especificaciones del sensor:

Tabla 3.3 Especificaciones del Módulo del Sensor de Temperatura y Humedad DHT11 [27].

Alimentación	3.5 a 5.5 V
Corriente de funcionamiento	0.3 mA en medida y 60 μA
Salida	Serie
Rango de temperatura	0-50ºC
Rango de humedad	20-90 %
Resolución	Ambos datos son de 16 bit
Precisión	$\pm$1ºC y $\pm$1 %

3.5 Sensor de Radiación

La radiación solar supone un efecto muy importante sobre el rendimiento de los paneles solares fotovoltaicos debido a su variación en el tiempo. A mayor irradiancia sobre los paneles, mayor energía solar recibirán, es decir, producirán una mayor electricidad. Junto con los 25ºC mencionados en el apartado anterior, como temperatura estándar, la irradiancia estándar para los paneles solares es de entorno a uno 1000 W/m².

La irradiancia recibida y por tanto el rendimiento de las placas, están directamente afectados por variables como: la masa de aire (calculada a partir del ángulo solar), las nubes, el vapor de agua y, en menor medida, la presión superficial, la reflectancia del suelo y la cantidad de ozono.

Además, por supuesto, se tienen que tomar distintas medidas para optimizar la irradiancia recibida a la hora de la instalación de los paneles fotovoltaicos y su cuidado:

- Los paneles se deben orientar hacia el ecuador y con un ángulo de inclinación correcto, para maximizar el máximo tiempo de luz solar durante el año.
- Se debe realizar un mantenimiento preventivo de los paneles solares, para que estén limpios, otro factor fundamental para la máxima captación de luz solar. La acumulación de polvo, hojas o nieve puede reducir la eficiencia. Actualmente, existen equipos que controlan la pérdida de transmisión de luz causada por el polvo, la arena...en los paneles fotovoltaicos. Uno de los más conocidos es el sensor **DustIQ** de Kipp&Zonen, que se caracteriza por no requerir una limpieza rutinaria ni mantenimiento mecánico, y sólo se limpia cuando se limpian los módulos [15]. Como el sistema fotovoltaico en el que se va a instalar la estación meteorológica tiene un tamaño reducido y es fácilmente accesible, no será necesario el uso de este tipo de equipos o similar.
- Por último, las plantas fotovoltaicas obtendrán una mayor radiación solar en ubicaciones donde predomine un clima soleado, como es el caso, en Sevilla, al sur de España.

La irradiancia total sobre un sistema fotovoltaico plano se denomina irradiancia global, la cual, se compone de la radiación solar directa, la radiación difusa procedente del cielo y la radiación reflejada en el suelo. Si el panel está orientado al sol, el haz directo es normal a la superficie de este, y si no es el caso, dicho componente se verá reducido en el coseno del ángulo de incidencia [28].

Para medir dicha irradiancia, se suelen utilizar piranómetros, equipos que miden la radiación solar. Estos dispositivos se clasifican según sus especificaciones, aunque sobre todo según su rango de medida y precisión, los piranómetros se clasifican en Clase A, Clase B o Clase C. Por otro lado, según su colocación en el sistema, los piranómetros se direferencia por la medida de la irradiancia que van a tomar [25]:

- **GHI** ('Global Horizontal Irradiance'): la irradiancia horizontal global se considera la radiación total disponible. Es la suma de la *DNI* y la *DHI*.
- **DNI** ('Direct Normal Irradiance'): irradiancia normal directa, la que indice directamente del sol.
- **DHI** ('Diffuse Horizontal Irradiance'): irradiancia horizontal difusa, la que incide proveniente del cielo.
- **POA** ('Plane of Arrange'): según se indique, hace referencia a alguna de las tres medidas previamente mencionadas, pero directamente sobre el panel solar.
- **Rear-POA** ('Rear Plane of Arrange'): hace referencia a la irradiancia reflejada en el plano posterior del panel, y también puede incluirse como componente de la *GHI*.
- **Albedómetro**: no es una medida como tal, es un equipo que, básicamente, esta formado por dos piranómetros, de los cuales, de forma horizontal o inclinado paralelamente al panel, uno está orientado hacia arriba, y el otro hacia abajo, para medir directamente la irradiancia *GHI* y la reflejada en el suelo.

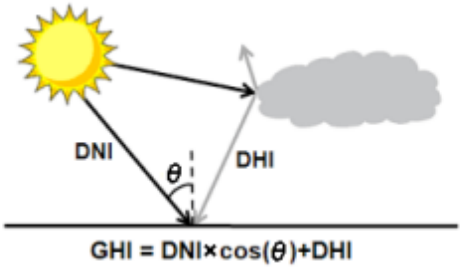

Figura 3.5 Distintas medidas de irradiancia según su incidencia [21].

Para la estación meteorológica del proyecto, se prevé utilizar un piranómetro **LI-200R**, de la marca de *LI-COR*, aunque aún no ha sido incluido. Actualmente, se encuentra instalado el siguiente piranómetro que, a pesar de ser de la misma familia que este, se encuentra descatalogado y sin calibrar, por lo que tendrá que ser reemplazado por el piranómetro LI-COR moderno:

Figura 3.6 Piranómetro LI-COR actualmente instalado en la estación.

Al no contar a tiempo con el piranómetro LI-200R, se ha recurrido, para realizar las pruebas de irradiancia correspondientes, a un sensor solar ISS-D5, que mide el ángulo de un rayo solar en ambos ejes ortogonales (vector solar), la radiación solar y la temperatura, el cual. A continuación se estudian ambos sensores según sus especificaciones.

3.5.1 LI-200R

El piranómetro LI-200R mide la radiación solar global, con un rango de longitud de onda de 400 a 1100 nm, expresado en vatios por metro cuadrado (W m^2). Está diseñado para ser utilizado al aire libre, en aplicaciones meteorológicas, ya que es resistente a la intemperie y alcanza hasta un ángulo de incidencia de 82º. Además, está disponible en diferentes tipos de señales de salida, para su compatibilidad con la mayoría de registros de datos, por lo que no sería un problema su conexión con Arduino [10].

Figura 3.7 Piranómetro LI-200R de la marca *LI-COR* .

El sensor es fácilmente desmontable, lo que simplifica su instalación o su desmontaje. Gracias a ello y a sus buenas características, es un piranómetro muy competitivo en el mercado.

Tabla 3.4 Especificaciones del Piranómetro LI-200R.

Sensibilidad	Típicamente 75 μA por 1.000 W m^-2
Linealidad	Desviación máxima del 1 % hasta 3.000 W m-2
Tiempo de respuesta	Inferior a 1 μs
Rango de temperatura de funcionamiento	-40°C a 65°C
Rango de humedad relativa	0 % a 100 % HR, sin condensación
Detector	Detector fotovoltaico de silicio de alta estabilidad
Inclinación	Sin error inducido por la orientación
Carcasa del sensor	Cuerpo de aluminio resistente a la intemperie con difusor acrílico

3.5.2 Sensor Solar ISS-D5

El sensor solar ISS-D5 mide el ángulo de incidencia de un rayo solar en ambos ejes ortogonales, junto con la radiación solar y la temperatura. La alta sensibilidad alcanzada se basa en las dimensiones geométricas del diseño. Sus características lo convierten en una herramienta adecuada para sistemas de seguimiento solar y posicionamiento de alta precisión, con bajo consumo y alta fiabilidad.. Sin embargo, si no se cuenta con un seguidor solar, su eficacia disminuye drásticamente, ya que el rango de visión en el que mide la irradiancia es muy cerrado, por lo que, a no ser que se incorpore con dicho seguidor, no será funcional para la estación meteorológica del proyecto.

Esto último se debe a que el ISS-D5 mide el ángulo de incidencia de un rayo solar tanto en *acimut* como en elevación, basándose en un cuadrante. La luz solar es guiada hasta el detector a través de una pequeña ventana situada encima del sensor, cuya vista para recibir luz solar está comprendida en unos 10°. En función del ángulo de incidencia, la luz solar induce fotocorriente en los cuatro cuadrantes del detector [17].

Por otro lado, este sensor pertenece a la familia de sensores solares ISS-DX, donde se encuentran hasta tres sensores más. A pesar de que el que se va a utilizar es el más preciso de los cuatro, es el que tiene el campo de visión más cerrado, siendo el ISS-D60 el menos preciso, pero con un campo de visión de 120°, por lo que sería bastante útil para medir la irradiancia.

Figura 3.8 Sensor Solar ISS-D5 de la marca *Solar MEMS*.

Por último, aunque cuenta con unas especificaciones de alto nivel, se comunica por Modbus RTU, a través de RS-485, lo que ha supuesto su conversión a serie para poder ser conectado con el Arduino, usando un módulo de RS-485 a TTL.

Tabla 3.5 Especificaciones del Piranómetro LI-200R.

Consumo	33mA
Rango de tensión de funcionamiento	5÷12 V
Precisón	<0,005° y <10 % (radiación)
Resolución	± 1 W $m^{-}2$
Rango de temperatura industrial	-40°C a 85°C
Radiación máxima	1200 W $m^{-}2$
Protección	IP65

3.6 Sensor de Viento

El viento es un favor climatológico de menor relevancia a la hora de afectar al rendimiento de unos paneles fotovoltaicos, en comparación con la irradiancia o la temperatura. Por supuesto, una fuerte racha de viento puede debilitar la estructura en la que se montan los paneles, o incluso ejercer una carga sobre estos. A pesar de ello, si se realiza un buen montaje, junto con anclaje estructural robusto y un correcto un mantenimiento, el efecto directo del viento sobre los paneles es casi irrelevante.

Sin embargo, el factor del viento sí puede afectar a las placas solares de manera indirecta, tanto para favorecer su rendimiento como para empeorarlo. Por un lado, las ráfagas de viento pueden enfriar los paneles, lo cual reduciría su temperatura, y como ya se ha mencionado previamente, una temperatura más baja favorece el rendimiento. Por otro, si la planta fotovoltaica se encontrase sobre un terreno, las rachas de viento pueden traer escombros, tierra y polvo, y ensuciar las placas, por lo que captarían una menor radiación solar.

El sensor principal para estudiar el viento se denomina anemómetro, y mide la velocidad de este. Otro factor importante a tener en cuenta es la dirección del viento, aunque como este se estudia previa instalación de los paneles, es menos común ver sensores para medir la dirección del viento. El sensor de viento que se va a utilizar en este proyecto mide ambos parámetros, pero un problema mecánico lo ha dejado indispuesto, por lo que se ha recurrido a un anemómetro en su lugar.

3.6.1 Anemoveleta 05103

La anemoveleta 05103 de la marca *Young* se encarga de medir la velocidad y dirección del viento. Su simplicidad y construcción resistente a la corrosión, lo hacen ideal para su uso a la intemperie en estaciones meteorológicas. Este sensor es de tipo veleta con una hélice helicoidal de cuatro palas [1].

Figura 3.9 Anemoveleta 05103 de la marca *Young*.

Actualmente ya se encuentra instalada junto a los paneles fotovoltaicos, a una altura de entorno unos cinco metros por encima del suelo, a la espera de ser reparado.

Tabla 3.6 Especificaciones de la Anemoveleta Young 05103.

Rango de velocidad	0 a 100m/s
Precisión de velocidad	$\pm$0.3m/s
Rango de dirección	0 a 360°
Precisión en la dirección	$\pm$3°
Rango de temperatura operacional	-50°C a 50°C
Radiación máxima	1200 W $m^{-}2$
Salida	Corriente 4-20 mA

3.6.2 Anemómetro IM512CD

El anemómetro IM512CD tipo cazoleta de la marca *METOS* mide la velocidad del viento de forma precisa. Es un sensor de bajo coste y larga duración, aunque su uso sea en intemperie.

Es el sensor de viento con el que se han realizado las pruebas del sistema, sin estar aun instalado en su sitio correspondiente, pero pudiendo aportar datos interesantes sobre la medida del viento [2].

Figura 3.10 Anemómetro IM512CD de la marca *METOS*.

En cuanto a sus especificaciones, es bastante más pobre que la anemoveleta. Su rango de medida es la mitad con respecto a esta, y su precisión es menor. En cuanto a su salida, es igual que el otro sensor de viento, por lo que no se preveen complicaciones cuando sea reemplazada por esta.

Por último, el datalogger que genera dicha salida en voltaje no es de la misma marca, lo cual ha empeorado el resultado, seguramente por un falta de incompatibilidad entre marcas, lo que ha provocado una salida en voltios mucho menor a la esperada. Como solución, aunque el programa se ha diseñado teniendo en cuenta este problema, se prevee utilizar un amplificador de señal que regule correctamente la salida.

3.7 Módulo convertidor MAX485 de RS-485 a TTL

El módulo MAX485 es un transceptor de comunicaciones RS-485, compuesto por un emisor y un receptor para establecer una línea de transmisión de información. Está habilitado para transmitir a una velocidad de hasta 2.5 Mbps. Su consumo es relativamente bajo, entre 120µA y 500µA.

Este módulo es capaz de establecer una comunicación serie, adaptando las señales de comunicación RS-485 a una interfaz serie, como sería en el caso del Arduino la UART del microcontrolador. Por tanto, es capaz de convertir la comunicación RS-485 del sensor solar de Solar MEMS a niveles de TTL, como son las señales TX y RX de un puerto serie de la placa [8].

Como se verá en el capítulo del conexionado de este equipo, es necesario que crear una línea de transmisión completa, pues a pesar de que lo que se quiere solo es recibir datos desde el sensor a la placa, el Arduino necesita enviarle peticiones de solicitud de dicha información, por lo que se tiene que diseñar un programa donde se pueda controlar la transmisión half duplex de este sensor.

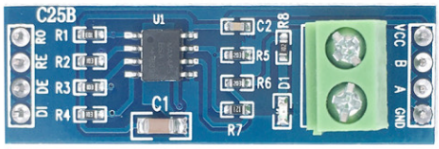

Figura 3.11 Módulo MAX485 convertidor de RS-485 a serie.

4 Diseño del Conexionado

En este Capítulo se van a recoger todas las conexiones de las que se compone el sistema, principalmente al Arduino y a la Raspberry, incluyendo descripciones gráficas que permitan una comprensión más sencilla. Además, también se entra en detalle en aquellas instalaciones de los sensores, donde ha sido necesario un conversor o adaptador.

4.1 Sensores - Arduino

Una vez se concretados los tres sensores que se van a utilizar en el sistema, para reunir los cuatro datos meteorológicos a monitorizar, se puede realizar el conexionado de estos a la placa Arduino.

Como se venía diciendo, se va utilizar otro Arduino Mega único para los sensores, que será el que vendrá conectado a la Raspberry. El hecho de utilizar una nueva placa se explicará más adelante, pero se resume en que habrá dos emplazamientos distintos y que la escritura de datos en la pantalla LED, no es compatible con la lectura de datos por la Raspberry.

Además, como también se menciona previamente, se ha utilizado un convertidor de RS-485 a TTL, lo cual ha implicado la necesidad de trabajar con dos puertos serie en el Arduino, uno hacia la Raspberry y otro hacia este módulo, por lo que tampoco esta vez se ha podido simplificar el proyecto con un Arduino UNO.

A continuación se muestran, de forma independiente para así tener una mejor visualización, los tres sensores conectados a la placa.

4.1.1 Sensor DHT11

Este primer sensor, debido a su simplicidad y a su rutinario uso en Arduino, no ha supuesto ningún problema a la hora de su conexión. Basta con conectar desde el módulo de DTH el positivo Vcc y negativo GND, a los terminales de 5V y tierra de la placa, y además, conectar una resistencia entre la salida del sensor, que se conecta a uno de los pines digitales disponibles (en este caso el pin número 2), y el terminal positivo 5V. Está resistencia tiene un valor de 10 kΩ, y se utiliza como resistencia pull-up, para así garantizar la estabilidad entre la comunicación, asegurando que los niveles de voltaje no se excedan de los rangos permitidos.

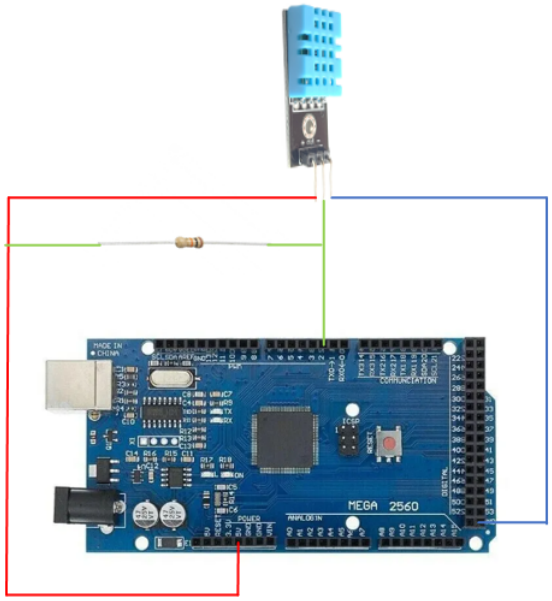

Figura 4.1 Conexión del sensor de temperatura y humedad DHT11.

4.1.2 Sensor Solar

Para el correcto funcionamiento de este sensor de la marca Solar MEMS, ha sido necesario una fuente de alimentación externa de unos 12V, al contrario que el sensor DHT, que se alimentaba directamente de la placa de Arduino. Para lograr el conexionado del sensor solar, se ha recurrido a la utilización de un MAX485 que convierte la salida RS-485 a Serie, que permite una conexión por puerto serie con la placa. Este módulo sí se puede alimentar con el Arduino, y así se establece.

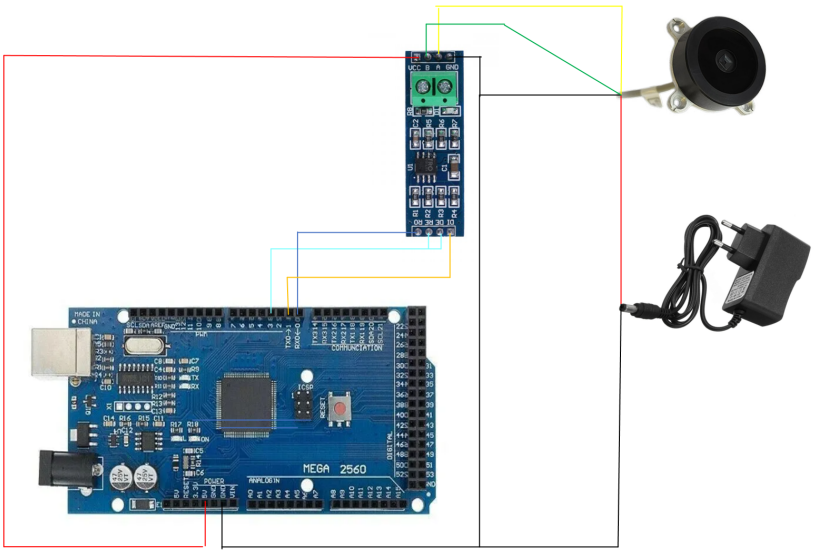

Figura 4.2 Conexión del sensor solar ISS-D5 de Solar Mems.

En cuanto al resto de pines del convertidor, se tiene las siguientes conexiones:

- Pin A: RS-485+ del sensor.
- Pin B: RS-485- del sensor.
- Pin RO: pin RX del puerto serie del Arduino.
- Pin DI: pin TX del puerto serie del Arduino.
- Pin DE: pin 3 del Arduino (establecer orden en línea de transmisión)
- Pin RE: pin 3 del Arduino (establecer orden en línea de transmisión)

4.1.3 Anemómetro

De nuevo, se tiene un sensor que también necesita una fuente de alimentación externa, en este caso de unos 15V. El anemómetro cuenta con un *datalogger* de la marca *Young*, que no se ve reflejada sobre la imagen del conexionado inferior, pero es gracias a este que se muestra la salida del sensor para que pueda ser leída de forma analógica por el Arduino.

El conexionado es muy simple, basta con conectar los terminales positivo y negativo de la fuente y del anemómetro, y la salida del sensor a un pin analógico de la placa, por lo que se ha tomado el pin A7.

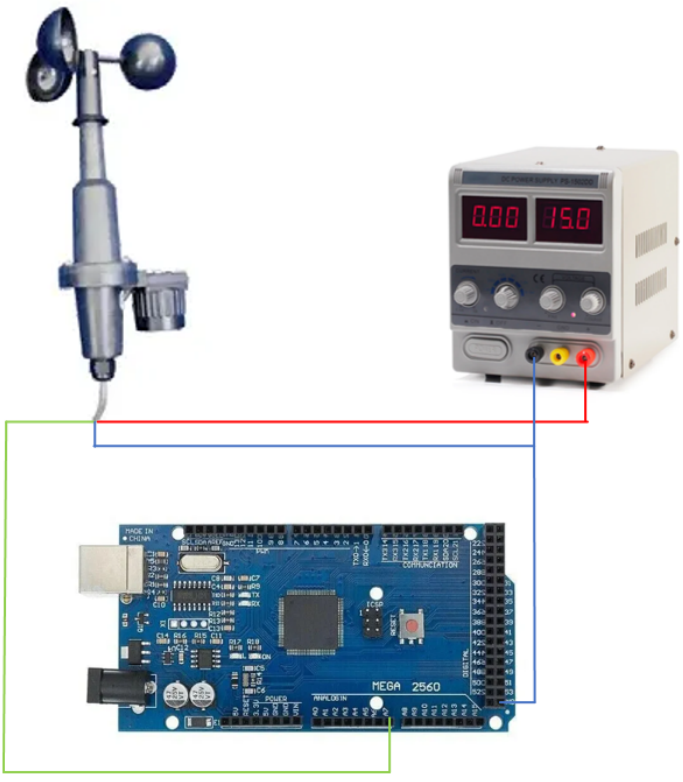

Figura 4.3 Conexión del Anemómetro de METEO.

4.2 Arduino - Pantalla LED

Como ya se venía explicando en capítulos anteriores, la complejidad del cableado del panel LED dio lugar a la necesidad de usar un Arduino Mega 2560.

A la hora de realizar el conexionado, es muy importante distinguir muy bien cada pin de la pantalla. Principalmente, el panel dispone de dos conectores HUB75, uno de entrada y otro de salida, y un conector para la alimentación, como se observa en la siguiente figura.

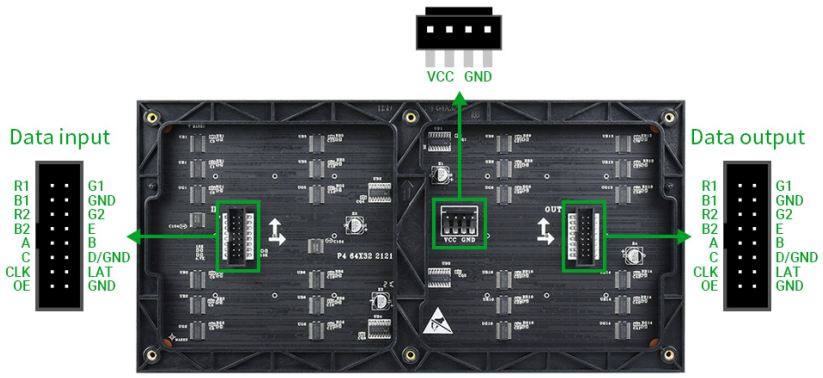

Figura 4.4 Conectores de entrada, de salida y de alimentación del panel RGB-Matrix-P4-64x32 [5].

Tabla 4.1 Especificaciones del panel LED RGB-Matrix-P4-64x32 [9].

Pin	Descripción
VCC	Alimentación 5V
R1	Bit superior rojo
G1	Bit superior verde
B1	Bit superior azul
A	Línea A
C	Línea C
E	Línea E
LAT/STB	Cierre
GND	Tierra
R2	Bit inferior rojo
G2	Bit inferior verde
B2	Bit inferior azul
B	Línea B
D	Línea D
CLK	Reloj
OE	Habilitación de salida

Por un lado, como se mencionó en el Capítulo anterior, se alimentará a 5V, gracias al conector específico correspondiente. Por otro, el conector HUB75 de entrada será el que se conecte a al

Arduino, quedando el de salida libre, ya que este serviría para ampliar el tamaño de la pantalla, pudiendo unirse consecutivamente a otro panel, conectando dicha salida con la entrada del siguiente.

En cuanto a los pines de entrada, es muy importante distinguir muy bien cada pin del conector. Esto no supone un problema, ya que Waveshare especifica como conectarlo, y además, dispone de la siguiente tabla indicativa que recoge todos los pines.

En la siguiente imagen, se detalla cada conexión de la pantalla, sin estar explícitamente ninguna otra conexión correspondiente a otro elemento del sistema, para así evitar la sobrecarga de información en el dibujo. Claramente, las entradas y salidas del Arduino que ya estaban ocupadas por los sensores se han respetado en todo momento y no se han ocupado para este diseño. Teniendo, por tanto, una conexión paralela entre distintas líneas de datos, necesarias para establecer la comunicación, y creando así una interfaz de bus, ya que las señales y los datos se envían a través de estos pines simultáneamente para escribir sobre el panel.

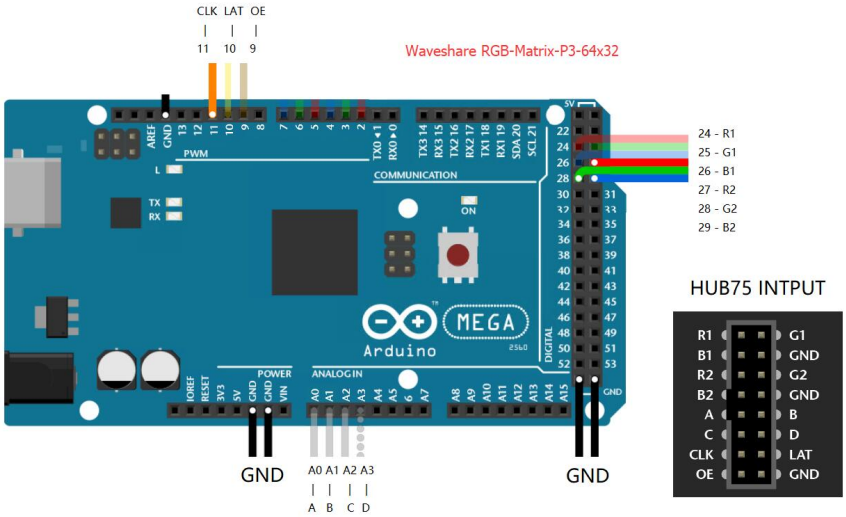

Figura 4.5 Conexionado del panel RGB-Matrix-P4-64x32 al Arduino [9].

4.3 Arduino - Raspberry Pi

El método más efectivo para lograr una conexión eficiente entre la placa de Arduino y la Raspberry Pi, no ha sido otro que la conexión serie mediante USB. Aunque durante mucho tiempo, se contemplo la opción de establecer una conexión serie entre los pines de un puerto serie del Arduino Mega y los GPIO serie de la Raspberry, la mejor solución acabo siendo la conexión directa por el cable. De esta manera, no se tenía problemas de desconexión, pero sobretodo, suponía un ahorro de material, pues se prescindía de la fuente de alimentación externa para el Arduino, pues se alimenta de forma indirecta por esta conexión

4.4 Raspberry Pi - PC/PLC

Actualmente, el diseño del sistema contempla una conexión *Wi-Fi* entre la Raspberry-Pi y cualquier equipo que establezca una comunicación con ella. Para la realización del diseño y las pruebas del sistema, ha sido más que suficiente para transmitir la información de forma efectiva. Sin embargo, en un futuro, cuando los equipos sean asentados en sus respectivos emplazamientos, y se de uso a este sistema, se deberá realizar una conexión mediante fibra óptica, para una mayor eficiencia y velocidad de transmisión.

5 Desarrollo del Sistema

En este Capítulo se va a explicar el desarrollo de la monitorización de la información meteorológica al completo, desde que el primer sensor envía un dato hasta que un PC lo muestra por pantalla, pasando por el Arduino y la Raspberry Pi, y siendo mostrado también en la pantalla LED. Por supuesto, también se explicará como se crea y se almacena un histórico de datos comprendidos entre dos fechas que se quiera.

Además, se van a analizar los datos obtenidos, aunque en este caso, la importancia va a recaer sobre la correcta comunicación y transmisión de la información, no sobre unos datos meteorológicos correctos. Teniendo en cuenta que en un futuro se prevé reemplazar los sensores utilizados por unos más precisos y de mejor calidad, no se va a buscar la exactitud en los valores obtenidos, sino en que todos ellos sean accesibles y bien transferidos.

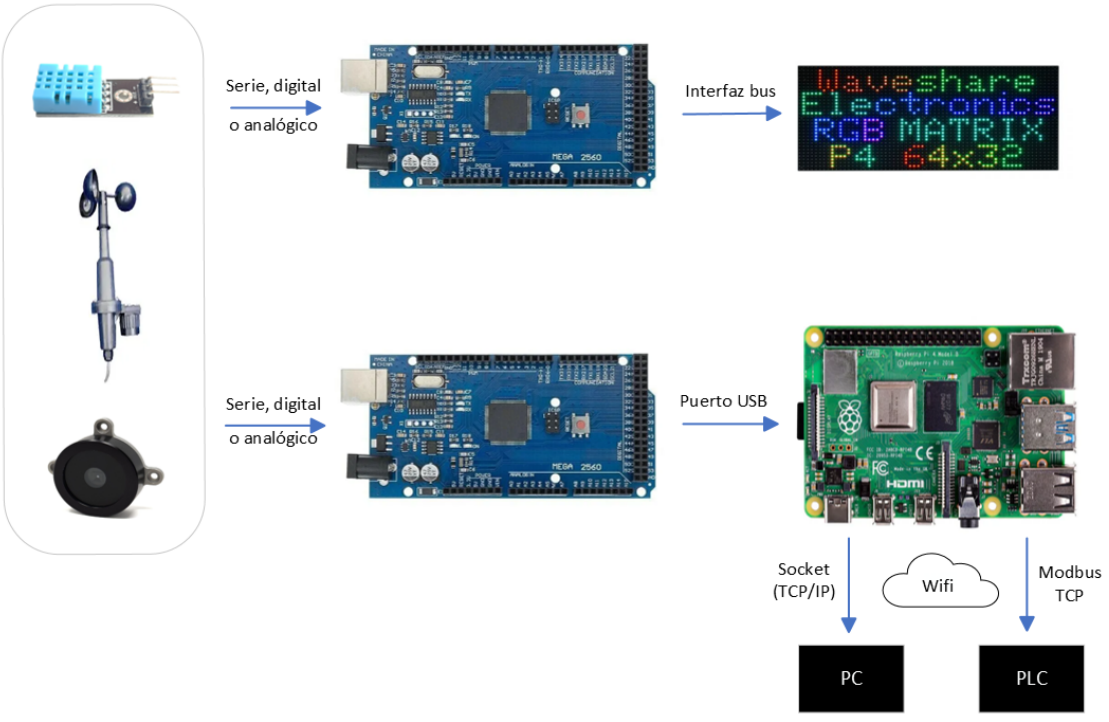

Figura 5.1 Esquema visual del sistema completo del proyecto.

5.1 Recopilación de los datos meteorológicos

En este punto se va a explicar como, a través de Arduino, se obtienen todos los datos meteorológicos, independientemente de la salida que tenga cada sensor, y se van mostrando por pantalla en una misma linea. Esto, no es casualidad, más adelante se verá porque se define una línea con todos los datos de forma sucesiva, además, de ir separados por una coma y mostrarse de la forma 'Clave = Valor'.

Antes de explicar el desarrollo del programa en Arduino, es importante mencionar en único fallo que tiene y que no ha podido ser solucionado. El sensor solar de Solar MEMS, al disponer de una salida RS-485 y usar una librería específica, no permite almacenar valores, solo leer un valor y mostrarlo, lo cual obligaba a crear un código donde se tuviera que estar leyendo dicho dato en un bucle constantemente, sin la posibilidad de almacenarlo. Como la idea es tener un programa general, capaz de leer cualquier tipo de variable, almacenarla y mostrarla cada diez segundos con sus respectivos máximos, mínimos y media correspondientes, se ha preferido no alterar dicho código. Además, como se explicó, este sensor no se pretende usar en el futuro, pues la captación de irradiancia que tiene, a pesar de su precisión, solo es válida si el sensor se incluye con un seguidor solar, el cual, no es el caso, por lo que es cuestión de tiempo que sea reemplazado por un piranómetro sin este problema. Obviamente, el hecho de no poder almacenar los datos de este sensor, impide el cálculo de la irradiancia máxima, mínima y media cada diez segundos, pero el dato de la irradiancia instantánea a los 10 segundos, si será mostrada por pantalla.

Una vez detallado el funcionamiento del programa, se puede explicar la composición y funcionamiento de este. Como todo programa de Arduino básico, se van a diferenciar tres partes imprescindibles: la definición de pines y variables, la función para inicializar *void setup()* y la función en bucle *void loop*.

5.1.1 Librerías, Pines y Declaración de Variables

Lo primero de todo es definir las librerías que se van a utilizar en el programa.

```
#include "DHT.h" // Sensor TH arduino
#include <ModbusMaster.h> //Sensor solar mems
```

Aunque lo más habitual es que los sensores se puedan tratar de forma analógica gracias a su salida en voltaje, o en corriente convertida a voltaje, en este caso, solo se podrá trabajar así con el sensor del viento, por lo que se le asigna el pin analógico A7.

Para poder trabajar con el módulo DHT es necesario incluir la librería *DHT*, que permite, una vez se le asigne un pin digital, en este caso el pin 2, y el tipo de sensor DHT que se trata, el DHT11, una lectura de sus datos muy sencilla.

Por último, para el sensor solar, es necesario incluir la librería *ModbusMaster*, la cual se puede instalar directamente desde el paquete de librerías de Arduino. *ModbusMaster* ha sido la librería escogida para poder trabajar con la salida RS-485 del sensor de Solar MEMS.

```
// Crear instancia de ModbusMaster(solar mems)
ModbusMaster node;

// Intervalo de tiempo en milisegundos (10 segundos)
const unsigned long intervalo = 10000;

//Pin TH
#define DHTPIN 2   // Pin sensor Temp&Hum
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// Definir pines para RS485 a TTL del solar Mems
#define RE_PIN 3
#define DE_PIN 3

// Pin analógico para velocidad del sensor de viento
const int velvientoPin = A7;
const float maxVoltage_vviento = 0.150; // Voltaje máximo real en Voltios
```

En esta primera parte, tras definir ambas librerías e indicar el par de pines que se van a utilizar, se declaran una multitud de variables. Como se ve en el código del Apéndice 1, es necesario inicializar: un variable de tiempo con el que se trabajar; las variables de máximo y mínimos de la temperatura, la humedad y la velocidad del viento a valores muy bajos y muy altos respectivamente; varios vectores para trabajar dentro de la función en bucle, de máximos, mínimos, contador de tiempo, suma de valores a lo largo de los diez segundos y de la media, para así ahorrar repetir código y simplificar la forma de añadir nuevos sensores al programa; y por último varios vectores de cadena de caracteres, para comprender todos los tipos de datos, y de nuevo para utilizarse a la hora de imprimir la información por pantalla usando un bucle. De esta manera, si se quiere añadir un nuevo sensor al sistema, bastaría con inicializarlo como cualquier otra variable e incluirla en los vectores, para más adelante ampliar en uno la iteraciones del bucle.

Por último, antes de explicar las dos funciones principales del programa, se tienen que definir dos pequeñas funciones para poder trabajar con el sensor solar. El hecho de incluir un MAX RS-485, establece una conexión serie con el Arduino, de tal manera, que al hacerlo a través del pin declarado número 3, es necesario establecer un orden de comunicación. Si se recuerda, este convertidor opera en modo *half duplex*, por lo que no se pueden enviar y recibir datos simultáneamente, y aunque solo se quiera leer los datos del sensor, sin enviarle nada a este, es necesario crear unas funciones para controlar esta transmisión, pues el Arduino, aunque no mande información, si tiene que enviar la petición al sensor, obligando así a establecer un orden en la transmisión.

```
// Funciones para controlar la transmisión RS485 half duplex del Solar Mems
void preTransmission() { // se llama antes de que el maestro haga la petición.
  digitalWrite(DE_PIN, HIGH);
  digitalWrite(RE_PIN, HIGH);
}

void postTransmission() { //se llama después de que se han enviado los datos.
  digitalWrite(DE_PIN, LOW);
  digitalWrite(RE_PIN, LOW);
}
```

5.1.2 Función de Inicialización

Como es conocida, esta función *set up* solo se ejecuta una vez al iniciar el programa, por lo que en esta se configuran los parámetros iniciales necesarios para que el correcto funcionamiento del sistema. Principalmente, se configuran las comunicación existentes con los sensores y con el sistema Modbus.

Partiendo de lo básico, primero hay que configurar el puerto serial estándar del Arduino para comunicarse con el monitor serial, donde se mostrarán los datos por pantalla, a la velocidad típica de 9600 baudios. Seguido a esto, se inicializa el sensor DHT para poder realizar lecturas de temperatura y humedad, ya que como bien se dijo antes, este funciona con una librería específica. Además, aparece un segundo puerto serial a configurar, el 1, ya que a este es al que se conectará el módulos MAX RS-485, pero a una velocidad de 19200 baudios, pues, como bien se indica en su ficha técnica, esta es la velocidad de comunicación por defecto para el disensor solar [17]. Este puerto serial se usa únicamente para la comunicación Modbus con el sensor de Solar Mems, por lo tanto no será necesario si este es reemplazado por otro tipo de comunicación.

Para terminar esta función, se inicializa la instancia creada previamente para el Modbus, indicada para el puerto serial 1, y con la dirección 1, pues así lo establece la ficha técnica del sensor. Con ello, también es necesario que los pines RE y DE del convertidor se configuren como salidas, para controlar el cambio entre modos de transmisión y recepción, y vienen acompañados de las dos funciones para poder realizar la transmisión de datos.

```
void setup() {
  Serial.begin(9600);
  dht.begin();
  Serial1.begin(19200); // Solar Mems por defecto BIT RATE en 19200

  // Inicializar Modbus (direccion del esclavo 1 por defecto)
  node.begin(1, Serial1);
  pinMode(RE_PIN, OUTPUT);
  pinMode(DE_PIN, OUTPUT);

  node.preTransmission(preTransmission);
  node.postTransmission(postTransmission);
}
```

5.1.3 Función en Bucle

En esta segunda función principal, *loop*, de cualquier estructura de Arduino, todo lo que se encuentre dentro de ella, se estará ejecutando permanentemente hasta el apagado o reinicio de la placa. Dentro de esta, aunque se podía escribir directamente, se ha preferido ejecutar una nueva función denominada *datos*. Esto no es más que para mantener un orden, pues aunque para este proyecto no se han incluido nuevas funcionalidades más que recopilar los datos meteorológicos, se ha preferido trabajar de forma independiente, para así solo tener que incluir nuevas funciones en un futuro.

```
void loop(){
  datos();
}
```

Dentro de esta nueva función, se encuentra el programa encargado de recopilar la información meteorológica e imprimirla por pantalla. Básicamente, la función se divide en un *if-else*. Durante un periodo de 10 segundos, se están leyendo los datos de la temperatura, la humedad y la velocidad del viento, y guardándose en un vector, el cual se va comparando con los vectores de máximos y mínimos, para ver si el valor del dato es mayor o menor respectivamente, y en cuyo caso, actualizar estos vectores con este nuevo valor. Además, cada valor se va sumando dentro de un vector *suma* y por cada análisis de lectura (establecida en un segundo) se incrementa en 1 cada valor del vector *contador*. Estos dos vectores se usarán más adelante para hallar la media a los diez segundos.

```
  if (tiempoActual - tiempoAnterior <= intervalo) { // Si ha pasado el
      intervalo de tiempo
  //Lectura TH
    float T = dht.readTemperature(); //Lee temperatura (ºC)
    float H = dht.readHumidity(); //Lee temperatura (ºC)
  // Leo señal de velocidad del viento (0-1023 por defecto arduino) // Paso a m
      /s (0-50m/s) con la correción de 1V (max) va a ser 0,150V
    int velvientoAnalog = analogRead(velvientoPin);
    float V = (velvientoAnalog / 1023.0) * 50*1/maxVoltage_vviento;

    float dato[]={T,H,V};

    for (int i = 0; i < 3; i++) {
      if (!isnan(dato[i])) { // Verificar si la lectura es válida
        // Actualizar máximo y mínimo
        if (dato[i] > max[i]) {
          max[i] = dato[i];
        }
        if (dato[i] < min[i]) {
          min[i] = dato[i];
        }

        suma[i] += dato[i]; // Acumular valor para calcular la media
        contador[i]++; // Incrementar contador de lecturas
      }
    }
  }
```

Durante el bucle, se entrará diez veces dentro de este, pues al final del programa se define un *delay* de un segundo, y la razón para ejecutarse es que no se haya llegado a los diez segundos. Esto se consigue fácilmente con una variable *tiempoActual* definida previamente como *millis*, por lo que en ella se guarda el tiempo actual en milisegundos, y una variable *tiempoAnterior*, que previa definición a cero, se actualiza más adelante de diez en diez segundos. Se entrará dentro del *if* anterior siempre que la diferencia de estas dos variables no supere el intervalo definido en diez segundos, permitiendo así el cambio de tiempo de muestreo de manera muy sencilla sin alterar el programa.

Ahora bien, justo cuando se supere dicho intervalo de tiempo, se entrará en el *else* de dicho *if*, donde se calculará la media de los tres datos y se volverán a realizar las tres lecturas anteriores, para evitar cualquier pérdida de información, almacenándose en un nuevo vector. Seguido a esto, se imprimirán por pantalla, con la utilización de un *for*, los cuatro vectores donde se han almacenado los distintos datos de estos tres sensores, respetando la forma 'Clave = Valor', y se reiniciarán a los valores iniciales todos los vectores. Es muy importante este reinicio, ya que si no se realizase, en vez de guardar el valor máximo y mínimo cada diez segundos, y ejecutar la media en dicho plazo,

se guardarían y se realizará la media para todo el tiempo de ejecución del programa. De nuevo, este bucle for está pensado para habilitar la opción de incluir nuevos sensores en el programa.

```
else{
  for (int i = 0; i < 3; i++) {
    media[i] = suma[i] / contador[i]; // Calcular media

    float temp = dht.readTemperature(); //Lee temperatura (ºC)
    float hum = dht.readHumidity(); //Lee humedad (%RH)
    int velo = analogRead(velvientoPin); float vel = (velo / 1023.0) * 50*1/
        maxVoltage_vviento; //Lee velocidad del viento en m/s
    float dato[i]={temp,hum, vel};

    // Imprimir los valores calculados
    Serial.print(datosTexto[i]);
    Serial.print(" = ");
    Serial.print(dato[i]);
    Serial.print(", "); //uso lo de la coma porque en un solo print se ralla
        arduino, y necesito separar por comas para la cadena en raspberry
    Serial.print(datosTexto_max[i]);
    Serial.print(" = ");
    Serial.print(max[i]);
    Serial.print(", ");
    Serial.print(datosTexto_min[i]);
    Serial.print(" = ");
    Serial.print(min[i]);
    Serial.print(", ");
    Serial.print(datosTexto_media[i]);
    Serial.print(" = ");
    Serial.print(media[i]);
    Serial.print(", ");

    max[i]=0;
    min[i]=100;
    media[i]=0;
    contador[i]=0;
    suma[i]=0;
  }
```

Por último, una vez se sale de dicho *for*, se leerá y mostrará por pantalla el dato del sensor excluido, el sensor solar. Cabe destacar, como se explicó previamente, que este sensor es un caso excepcional, por como se realiza su lectura, y el hecho de no poder almacenar su información, ha impedido el cálculo de su máximo, mínimo y su media respectiva cada diez segundos. Para esta lectura, se trabaja con comandos especiales de la librería *ModbusMaster*. Primero, se lee el dato de la irradiancia captada, accediendo al registro número 9 de la comunicación (establecido así en el datasheet del sensor de Solar MEMS [17], donde únicamente se encuentra un valor, el de la irradiancia. Después, se verifica la lectura correcta de dicho dato, y en caso afirmativo, se devuelve el primer y único valor recogido, el registro de la irradiancia. Por último, este se muestra por pantalla de la forma establecida anteriormente. En caso de una mala lectura, se imprimirá el dato de irradiancia igual a 226, consecutivamente, que significa que no se pudo leer la información correspondiente. No se imprimirá ningún mensaje de error para evitar un fallo de comunicación con la Raspberry.

Una vez se imprime la irradiancia, se actualiza la variable *tiempoAnterior* al tiempo actual, para así, se pueda repetir el proceso de nuevo.

```
    // Lectura e impresion por pantalla del dato de la irradiancia
    radiacion = node.readHoldingRegisters(9, 1);
    if (radiacion == node.ku8MBSuccess) { // cte. confirma lectura correcta
      int16_t radiacion = node.getResponseBuffer(0); //devuelve el primer valor
      Serial.print("Radiacion");
      Serial.print(" = ");
      Serial.print(radiacion);
      Serial.print(", ");
    } else {
      Serial.print("Radiacion");
      Serial.print(" = ");
      Serial.print(radiacion);
      Serial.print(", ");
    }
    tiempoAnterior = tiempoActual;
    Serial.println();
  }
```

5.2 Tratamiento de la Información

Una vez recopilada la información meteorológica a través de la placa de Arduino, esta es transferida por el puerto USB a la Raspberry Pi, donde, gracias al programa creado usando Python, se trata y se procesa. Dicho programa, al igual que el de Arduino, será el único que se ejecutará, y estará funcionando constantemente, para no perder ningún dato. En este programa se diferenciarán tres funciones, cada una con un rol diferente. Además, en él se trabajará con variables globales y exclusión mutua para acceder a estas, pues posteriormente a su definición, se creará e iniciará cada función en un hilo diferente, creando así una aplicación multi-hilo. Por supuesto, también se incluyen todos los módulos necesarios para el uso correcto de los comandos utilizados.

5.2.1 Almacenamiento de los datos

Antes de entrar en detalle en esta función, ha sido necesario incluir varios aspectos. En esta función se van a leer los datos meteorológicos y almacenarlos, tanto en una variable global *diccionario*, como en una base de datos. Para ello, fuera de la función, se ha definido: la lectura serie por el puerto USB por defecto *ACM0*, a una velocidad de 9600 baudios; un diccionario, que no es más que un vector que permite guardar la información de la forma 'Clave = Valor'; y una variable utilizada para crear un acceso exclusivo al diccionario que impida su uso por parte de más de un hilo a la vez, ya que al estar escribiéndose datos constantemente sobre este, y tener que ser mostrados cuando un usuario lo solicite, podría ocurrir que ambos procesos se den de forma simultánea, por lo que la transmisión de datos sería errónea.

```
#Conexion serie
pser=serial.Serial('/dev/ttyACM0',9600)

#Diccionario almacena datos
datos_meteo = {}

#Mutex de acceso exclusivo para escribir o leer datos en el diccionario
datos_meteo_lock=threading.Lock()
```

Ahora bien, una vez creado el diccionario que se irá actualizando cada diez segundos (pues el tiempo de muestreo dado en Arduino), también es necesario crear la base de datos, donde aquí sí, se almacenen todos los datos. Para ello, se ha creado una base de datos en SQL, y se ha creado una tabla donde se guardan un total de 15 valores, es decir, que esta compuesta por quince columnas. Esto es así ya que se situarán en las dos primeras columnas la fecha y la hora respectivamente, y a partir de estas, los cuatro datos para cada una de las variables temperatura, humedad y velocidad del viento, y un dato de la irradiancia. La fecha se tiene que declarar como un dato tipo *DATE*, la hora como *TIME*, y el resto de valores se han creado como *REAL*

Figura 5.2 Creación de la base de datos donde se guardará la información meteorológica.

Una vez se tiene creada la base de datos, ya se puede proceder al almacenamiento en la función. Dicha función se denomina *almacena_THVR*. Antes de todo se realiza la conexión a la base de datos de SQL, y a partir de esto, en un bucle, comienza a guardar la información. Primero, se toma la fecha y hora respectiva, y se leen los datos que llegan por el puerto serie. Tras esto, usando el mutex de acceso exclusivo, denominado como *lock* en Python, se separan las variables de los valores y se van añadiendo de esta manera al diccionario.

```
def almacena_THVR():
    try:
        # Conecta con la base de datos
        connection = sqlite3.connect('datos_meteorologicos.db')
        cursor = connection.cursor()

        while True:
            #Toma fecha y hora
            fecha = hourSystem.datetime.today().strftime('%Y-%m-%d')
            hora  = hourSystem.datetime.today().strftime('%H:%M:%S')

            #Lee línea puerto serie
            data = pser.readline().decode('utf-8').strip()

            #Separa variables por coma
            data_list = data.split(',')

            #Bucle para separar el nombre de la variable y el valor
            with datos_meteo_lock:
                for i in data_list:
                    data_var = i.split('=')
                    if len(data_var) ==2:
                        variable = data_var[0].strip()
                        valor = data_var[1].strip()
                        #Añade cada variable a mi diccionario
                        datos_meteo[variable] = valor
```

Seguido, como se puede observar en el código respectivo a la Raspberry Pi que se encuentra en el Apéndice 1 del documento, se definen todas las variables para ser incluidas en la tabla de almacenamiento. Esto, aunque podría haberse simplificado con un bucle, se ha dejado de manera individual, pues a la hora de poner la tabla de manera horizontal con los 13 datos, es necesario indicar el nombre de cada uno. Así, además, si en el futuro se leyeran datos distintos, para ver que dato ha cambiado de nombre, se verá más fácil. Para guardar los datos en la base de datos, basta con insertarlos en la tabla creada previamente como *datos_THVR*, usando el comando *execute* y guardando los cambios con el comando *commit*.

5.3 Comunicación con PLC y envío Históricos

Una vez se logra el correcto almacenamiento de la información en la base de datos y la actualización constante de los valores de un diccionario donde se guardan todos los datos cada tiempo de muestreo, es posible comunicar ambos resultados al PC o PLC que lo solicite.

5.3.1 Servidor Modbus

Como se dijo al principio del documento en el apartado de objetivos , el primer propósito de este proyecto era crear un acceso instantáneo a la información meteorológica actual. Para ello, se ha diseñado una segunda función, que trabaja como un hilo, dentro del programa de la Raspberry. Definido como *modbus_PLC*, esta función crea un servidor Modbus TCP, para una IP y un puerto específico, y lo inicia.

```
def modbus_PLC():
    try:
        #Declara IP y puerto
        server_ip = '192.168.1.156'
        server_port = 5020

        # Crea una instancia de ModbusServer
        server = ModbusServer(server_ip, server_port, no_block=True)

        #Inicia servidor
        server.start()
        print(f"Servidor Modbus iniciado en {server_ip}:{server_port}")
```

A partir de ahí, entra en un bucle donde se accede a los datos meteorológicos principales del diccionario, tras pedir acceso exclusivo al *lock*, es decir, los valores instantáneos de las cuatro variables, y los sitúa en cuatro registros distintos de la comunicación. De esta manera, el usuario podrá acceder a la información haciendo una petición de lectura de dicho registro. Cabe destacar, que estos valores se pasarán como enteros, ya que se ha considerado innecesario una correcta precisión de estos datos, pues en realidad, es una petición muy genérica, para saber las condiciones climatológicas en el momento, sin ningún mayor propósito.

Tras esto, aunque la intención es que nunca se de el caso, si ocurriera algún error, la conexión se cerraría. Obviamente, la idea es que el programa se este ejecutando de forma permanente.

```
        while True:
            with datos_meteo_lock:
                # Accede al diccionario y toma los valores de las 4 variables
                    generales
                temperatura=datos_meteo.get('Temperatura',None)
                humedad=datos_meteo.get('Humedad',None)
                vel_viento=datos_meteo.get('Vel_Viento',None)
                radiacion=datos_meteo.get('Radiacion',None)

                temperatura=float(temperatura) if temperatura is not None else
                    0.0
                humedad=float(humedad) if humedad is not None else 0.0
                vel_viento=float(vel_viento) if vel_viento is not None else 0.0
                radiacion=float(radiacion) if radiacion is not None else 0.0

            #Asigna cada dato a un registro
            server.data_bank.set_holding_registers(0, [int(float(temperatura))])
            server.data_bank.set_holding_registers(1, [int(float(humedad))])
            server.data_bank.set_holding_registers(2, [int(float(vel_viento))])
            server.data_bank.set_holding_registers(3, [int(float(radiacion))])
            sleep(1)
```

5.3.2 Servidor Socket TCP/IP

Para el segundo objetivo a realizar en este proyecto, se ha diseñado una tercera función, que cumple como un segundor servidor. Para ello, esta vez se ha utilizado un socket *STREAM*, el cual se vincula con el cliente y se queda en escucha a la espera de recibir peticiones. Este servidor es compatible con el anterior debido a su simplicidad, pudiendo ambos responder de forma simultánea.

```
def socket_PC():
    try:
        #Declara IP y puerto
        server_ip = '192.168.1.156'
        server_port = 8080

        #Crea Socket, lo vincula y se queda en escucha
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.bind((server_ip, server_port))
        server_socket.listen(5) #servidor permite por ej hasta 5 conexiones
            pendientes en cola antes de empezar a rechazar
        print(f"Servidor Socket escuchando en {server_ip}:{server_port}")
```

A continuación de esto, se entra en un bucle para poder responder constantemente a las peticiones que le lleguen. Tras aceptar la conexión del cliente solicitado, se procede a la transmisión de datos almacenados. Para ello, primero se conecta con la base de datos y queda a la espera de recibir un par de fechas entre las cuales se tiene que comprender la información a enviar. Una vez se tienen, se recopilan todos los datos incluidos entre esas dos fechas, y se la envía al cliente como una cadena de texto.

```
        while True:
            #Acepta conexión del cliente correspondiente
            client_socket, addr = server_socket.accept()
            print(f"Cliente conectado desde {addr}")

            try:
                #Accede a la base de datos meteorológicos
                connection=sqlite3.connect('datos_meteorologicos.db')
                cursor=connection.cursor()

                #Recibe fechas del cliente
                data = client_socket.recv(1024).decode() #maximo recibe hasta
                    1024 bytes
                print("Fechas de la petición recibidas del cliente")

                #Se declaran las dos fechas recibidas
                request = json.loads(data)
                fecha1 = request.get('fecha1')
                fecha2 = request.get('fecha2')

                #Se recopilan todos los datos de la base entre dichas fechas
                cursor.execute("SELECT * FROM datos_THVR WHERE fecha >= ? AND
                    fecha <= ?", (fecha1, fecha2))
                response=cursor.fetchall()

                #Recoge toda la información y se la envía al cliente
                response_str = "\n".join([",".join(map(str,row)) for row in
                    response])
                client_socket.send(response_str.encode())
```

Cabe destacar, que el hecho de enviar la información de esta manera, es para que el cliente decodifique esta cadena de caracteres en bytes, y la escriba por filas en un archivo de texto, el cual almacenará en su propia base de datos. Para ello, se ha diseñado un programa, que se ha incluido en el Apéndice 1, cuya estructura es bastante similar pero al contrario. En dicho programa, se crea un cliente, para la misma IP y puerto, en vez de un servidor, y solicita al usuario dos fechas para mandárselas a este servidor, una vez estén conectados. Después, espera recibir la información enviada, la decodifica, y la guarda en un archivo de texto con un nombre designado.

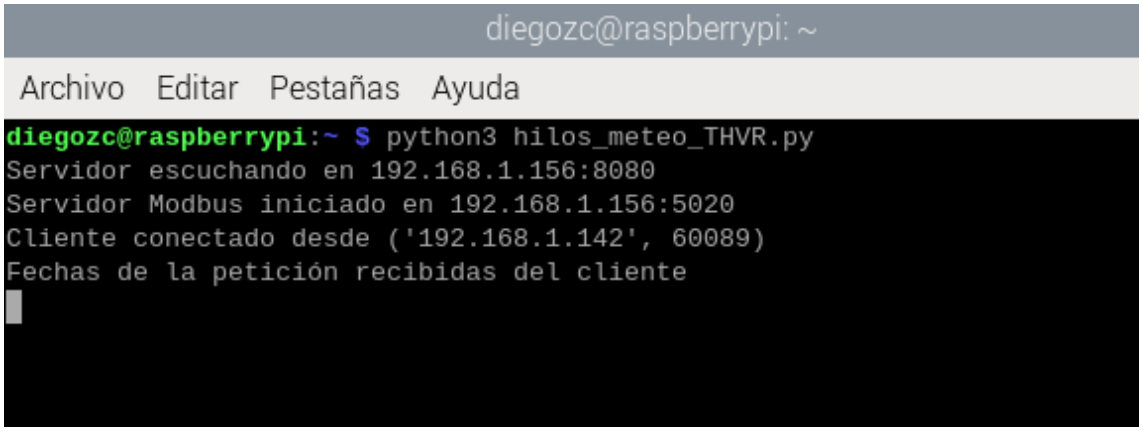

Figura 5.3 Mensajes por pantalla del inicio de ambos servidores y un cliente conectado al servidor Socket.

5.4 Análisis de las Pruebas Realizadas

A partir de los programas explicados en este capítulo, y su descripción al completo en el Apéndice 1 del documento donde se recogen todos los códigos, se han obtenido los resultados esperados, pudiendo así afirmar una comunicación correcta entre servidores y clientes, y obteniendo pues, una primera solución de estación meteorológica para el sistema fotovoltaico instalado.

5.4.1 Visualización de datos en el panel LED

Esta pantalla, será colocada en algún punto fácilmente visible del departamento, o cerca del sistema fotovoltaico, para así mostrar de forma constante la información meteorológica que se está recogiendo a todo aquel que pase a su lado.

Para lograr esta visualización, ha sido necesario una segunda placa Arduino Mega 2560, pudiendo así conectar esta en algún emplazamiento cercano al panel, y evitando por tanto un cableado extenso. Además, se ha ampliado el programa realizado en Arduino, pues era necesario la inclusión de varios comandos.

En primer lugar, se han incluido todas la librerías necesarias, junto con los pines específicos a conexionar, creando así una instancia *matrix* con la que trabajar. Tras esto, se necesitaba una primera inicialización de esta instancia, por lo que así se ha hecho en el *setup*. Una vez iniciada *matrix*, ya era posible su uso, por lo que en el programa, justo después del bucle que imprime los datos por pantalla, pero dentro del *else*, se ha añadido la parte correspondiente a la escritura por la pantalla.

Los pasos seguidos han sido la declaración del tamaño de letra, según el número de píxeles, establecido en 1, y la impresión de los cuatro datos meteorológicos por separado (radiación, temperatura, humedad y viento). El hecho de trabajar esta vez de manera independiente se debe a que hay que indicar en cada caso en que píxel del panel se quiere empezar a escribir, tanto el nombre de la variable como el valor. Además, para cada variable se dispone de la opción de cambiar el color de la letra. Por último, se establece que cada variable se muestre por pantalla durante dos segundos, y acto seguido se borre y aparezca la siguiente.

```
    matrix.setTextSize(1);   // size 1 == 8 pixels high
    matrix.setTextWrap(false); /

    matrix.setCursor(0, 3);
    matrix.setTextColor(matrix.Color333(0,1,5));
    matrix.println("Temperatura");
    matrix.setCursor(11, 18);
    matrix.setTextColor(matrix.Color333(3,3,3));
    matrix.print(temp);
    matrix.drawRoundRect(43, 18, 2, 2, 2, matrix.Color333(3,3,3));
    matrix.println(" C");
    delay (2000);
    screen_clear();
```

El código superior corresponde a impresión por pantalla del dato de la Temperatura. Este formato es el que se ha seguido para la impresión del resto de variables, tal y como se puede observar en el código completo en el Apéndice 1. A continuación, se muestran cuatros imágenes de la pantalla LED mostrando la información meteorológica:

Figura 5.4 Visualización de la información meteorológica en la pantalla LED.

5.4.2 Solicitud de datos por registro Modbus

Una vez iniciado el servidor Modbus, se puede establecer una conexión desde un PC o un PLC, tras previa creación de un cliente Modbus. Fácilmente, se puede solicitar la transmisión de cada dato de forma independiente, utilizando el comando *read_holding_registers*, como es el caso del siguiente programa en Python:

```
from pyModbusTCP.client import ModbusClient
client = ModbusClient(host="192.168.1.156", port=5020)
client.open()
print(client.read_holding_registers(0))
print(client.read_holding_registers(1))
print(client.read_holding_registers(2))
print(client.read_holding_registers(3))
client.close()
```

Este cliente Modbus está conectado al servidor Modbus en las Raspberry, el cual al recibir la cuatro peticiones de lectura, transfiere dicha información al cliente, el cual la imprime por pantalla de la siguiente manera.

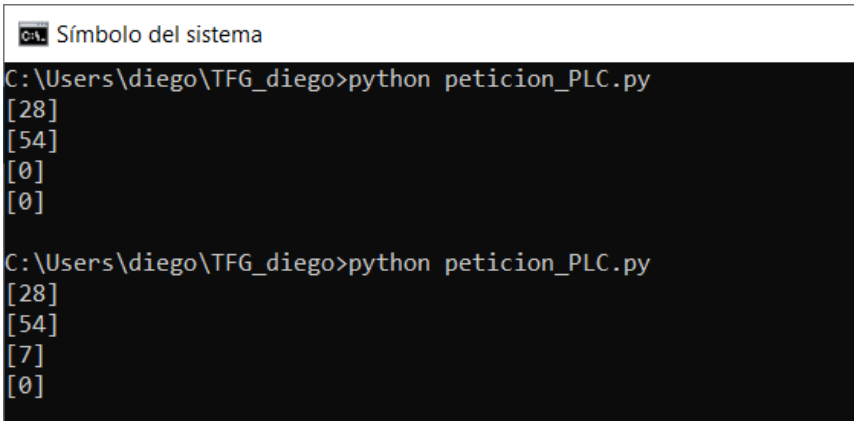

Figura 5.5 Mensajes por pantalla de los cuatro registros del servidor Modbus.

5.4.3 Peticiones de datos de Históricos

De nuevo, se ha diseñado un sencillo programa que actuará como cliente, para realizar la petición de un histórico de datos, entre dos fechas que indique el usuario. Este código está incluido en el Apéndice 1 y se ejecutaría de la siguiente manera:

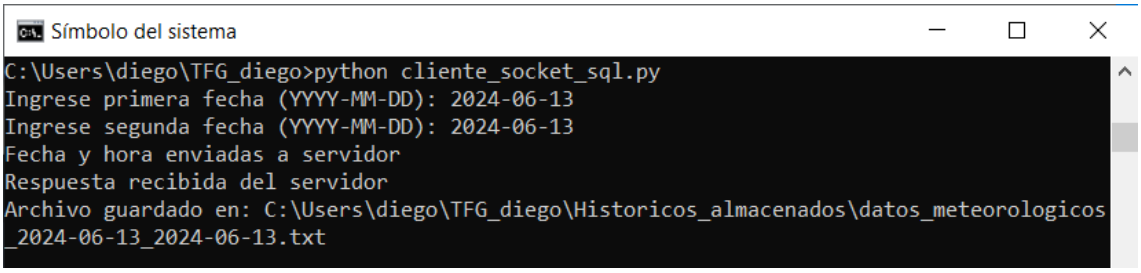

Figura 5.6 Mensajes por pantalla de la solicitud al servidor Socket.

El archivo se ha guardado en una carpeta especificada en el programa, bajo un nombre definido por las dos fechas indicadas previamente. En su interior, se recogen los datos solicitados siguiendo la siguiente estructura:

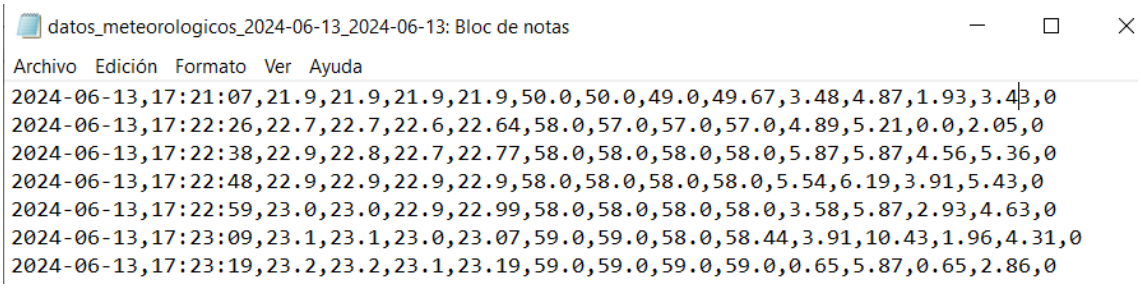

Figura 5.7 Archivo de texto creado con una muestra de datos meteorológicos del 13 de junio.

6 Conclusión y Futuros Trabajos

Después de haber desarrollado con detenimiento el sistema de tratamiento y monitorización de los distintos datos meteorológicos estudiados, se pueden concluir varios aspectos y posibles mejoras a incluir más adelante.

Por un lado, hay que destacar la importancia de no haber podido trabajar con todos los sensores deseados. El hecho de no poder contar con el equipamiento adecuado previa la realización de las pruebas experimentales, ha dado lugar a unos resultados limitados. A pesar de esto, los resultados obtenidos de los sensores con los que sí se ha podido trabajar, y aquellos integrados que simulaban el papel que los sensores con los que finalmente no se ha podido contar, han sido coherentes con la realidad, sin llegar a tener la máxima exactitud a falta de una mejor precisión del sensor.

Con ello, pese a que la importancia del proyecto residía en la parte meteorológica, ya que dicha información afectará directamente a las placas fotovoltaicas, también se ha corroborado el adecuado funcionamiento de la comunicación establecida entre servidores y clientes, a través de la cual es posible el acceso a los datos meteorológicos.

Gracias a este proyecto, se ha dejado la puerta abierta a la posibilidad de crear un estudio real del rendimiento de los paneles solares instalados, quedando aun por delante la inclusión de varios sensores en el proyecto. El hecho de poder controlar y entender la potencia generada por parte del sistema fotovoltaico, permitirá predecir la correcta generación energética de este, habilitando así la posibilidad de una gestión anticipada del suministro eléctrico frente a altas demandas de consumo.

Por último, tras el desarrollo de este sistema, queda pendiente dar paso a un conjunto de posibles mejoras, que darían lugar a un análisis más detallado y realista, permitiendo así llegar a unos resultados más acordes. En cuanto a dichos desarrollos futuros, se parte de la base que los sensores con los que se han realizado las pruebas experimentales no son los adecuados, por lo que habrá que reemplazarlos. A su vez, el hecho de no haber podido contar con más variedad de sensores para medir otro tipo de características climatológicas, impide el estudio completo del efecto de la meteorología sobre el sistema fotovoltaico, por lo que en un futuro, se tendrá que añadir nuevas variables a medir, como la suciedad en el panel o la cantidad de lluvia.

Apéndice A

Códigos

A.1 Arduino

```
#include "DHT.h" // Sensor TH arduino
#include <ModbusMaster.h> //Sensor solar mems


// Crear instancia de ModbusMaster(solar mems)
ModbusMaster node;

// Intervalo de tiempo en milisegundos (10 segundos)
const unsigned long intervalo = 10000;

//TH
#define DHTPIN 2   // Pin sensor Temp&Hum
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// Definir pines para RS485 a TTL del solar Mems
#define RE_PIN 3
#define DE_PIN 3


// Pin analógico al que se conecta las señales de velocidad del sensor de
    viento
const int velvientoPin = A7;
const float maxVoltage_vviento = 0.150; // Voltaje máximo en V que admite el
    sensor de viento (era 1V pero la realidad es entorno a 150mV)


unsigned long tiempoAnterior = 0; // Tiempo de la última actualización
float Tmax = -100; // Inicializar con un valor muy bajo
float Tmin = 100; // Inicializar con un valor muy alto
float Hmax = -100; // Inicializar con un valor muy bajo
float Hmin = 100; // Inicializar con un valor muy alto
float Vmax = -100; // Inicializar con un valor muy bajo
float Vmin = 100; // Inicializar con un valor muy alto
```

```
float sumaT=0, sumaH=0, sumaV=0;
int contadorT=0, contadorH=0, contadorV=0;
float mediaT=0, mediaH=0, mediaV=0;
float max[]={Tmax, Hmax, Vmax};
float min[]={Tmin, Hmin, Vmin};
int contador[3]={contadorT, contadorH, contadorV};
float suma[3]={sumaT, sumaH, sumaV};
float media[3]={mediaT, mediaH, mediaV};
String datosTexto[] = {"Temperatura", "Humedad","Vel_Viento"};
String datosTexto_max[] = {"Temperatura_Maxima", "Humedad_Maxima","
    Vel_Viento_Maxima"};
String datosTexto_min[] = {"Temperatura_Minima", "Humedad_Minima","
    Vel_Viento_Minima"};
String datosTexto_media[] = {"Temperatura_Media", "Humedad_Media","
    Vel_Viento_Media"};


// Funciones para controlar la transmisión RS485 half duplex del Solar Mems
void preTransmission() { // función se llama antes de que el maestro (Arduino)
    envíe petición al esclavo.
  digitalWrite(DE_PIN, HIGH);
  digitalWrite(RE_PIN, HIGH);
}

void postTransmission() { //se llama después de que se han enviado los datos.
  digitalWrite(DE_PIN, LOW);
  digitalWrite(RE_PIN, LOW);
}

void setup() {
  Serial.begin(9600); // Inicializar comunicación serial
  dht.begin();
  Serial1.begin(19200); // Solar Mems por defecto BIT RATE en 19200 pero tambié
      n tiene 9600


  // Inicializar Modbus (direccion del esclavo 1 por defecto)
  node.begin(1, Serial1);
  pinMode(RE_PIN, OUTPUT);
  pinMode(DE_PIN, OUTPUT);

  node.preTransmission(preTransmission);
  node.postTransmission(postTransmission);
}



void loop(){
  datos();
}


void datos() {
  // Variable para la lectura Modbus (entero de 8 bits)
  uint8_t radiacion;
```

```
// Obtener tiempo actual en milisegundos.
unsigned long tiempoActual = millis(); // Obtener tiempo actual en
    milisegundos


if (tiempoActual - tiempoAnterior <= intervalo) { // Si ha pasado el
    intervalo de tiempo
//Lectura TH
  float T = dht.readTemperature(); //Lee temperatura (ºC)
  float H = dht.readHumidity(); //Lee temperatura (ºC)
// Leo señal de velocidad del viento (0-1023 por defecto arduino) // Paso a m
    /s (0-50m/s) con la correción de 1V (max) va a ser 0,150V
  int velvientoAnalog = analogRead(velvientoPin);
  float V = (velvientoAnalog / 1023.0) * 50*1/maxVoltage_vviento;

  float dato[]={T,H,V};

  for (int i = 0; i < 3; i++) {
    if (!isnan(dato[i])) { // Verificar si la lectura es válida
      // Actualizar máximo y mínimo
      if (dato[i] > max[i]) {
        max[i] = dato[i];
      }
      if (dato[i] < min[i]) {
        min[i] = dato[i];
      }

      suma[i] += dato[i]; // Acumular valor para calcular la media
      contador[i]++; // Incrementar contador de lecturas
    }
  }
}

else{
  for (int i = 0; i < 3; i++) {
    media[i] = suma[i] / contador[i]; // Calcular media

    float temp = dht.readTemperature(); //Lee temperatura (ºC)
    float hum = dht.readHumidity(); //Lee humedad (%RH)
    int velo = analogRead(velvientoPin); float vel = (velo / 1023.0) * 50*1/
        maxVoltage_vviento; //Lee velocidad del viento en m/s
    float dato[i]={temp,hum, vel};

    // Imprimir los valores calculados
    Serial.print(datosTexto[i]);
    Serial.print(" = ");
    Serial.print(dato[i]);
    Serial.print(", ");
    Serial.print(datosTexto_max[i]);
    Serial.print(" = ");
    Serial.print(max[i]);
    Serial.print(", ");
    Serial.print(datosTexto_min[i]);
    Serial.print(" = ");
    Serial.print(min[i]);
    Serial.print(", ");
```

```
      Serial.print(datosTexto_media[i]);
      Serial.print(" = ");
      Serial.print(media[i]);
      Serial.print(", ");

      max[i]=0;
      min[i]=100;
      media[i]=0;
      contador[i]=0;
      suma[i]=0;
    }

    // Lectura e impresion por pantalla del dato de la irradiancia
    radiacion = node.readHoldingRegisters(9, 1);
    if (radiacion == node.ku8MBSuccess) { // cte. de la bib. ModbusMasteer que
        confirma lectura correcta si es igual
      int16_t radiacion = node.getResponseBuffer(0); //devuelve el primer valor
          (0) de los registros leidos (solo leo uno)
      Serial.print("Radiacion");
      Serial.print(" = ");
      Serial.print(radiacion);
      Serial.print(", ");
    } else {
      Serial.print("Radiacion");
      Serial.print(" = ");
      Serial.print(radiacion);
      Serial.print(", ");
    }

    tiempoAnterior = tiempoActual;
    Serial.println();
  }

  delay(1000);
}
```

A.2 Raspberry Pi

```
import threading

import serial
import time
import datetime as hourSystem
import sqlite3

import traceback
from pyModbusTCP.server import ModbusServer, DataBank
from time import sleep

import socket
import json
import os
import csv

#Conexion serie
pser=serial.Serial('/dev/ttyACM0',9600)

#Diccionario almacena datos
datos_meteo = {}

#Mutex de acceso exclusivo para escribir o leer datos en el diccionario
datos_meteo_lock=threading.Lock()


def almacena_THVR():
    try:
        # Conecta con la base de datos
        connection = sqlite3.connect('datos_meteorologicos.db')
        cursor = connection.cursor()
        while True:
            #Toma fecha y hora
            fecha = hourSystem.datetime.today().strftime('%Y-%m-%d')
            hora  = hourSystem.datetime.today().strftime('%H:%M:%S')

            #Lee línea puerto serie
            data = pser.readline().decode('utf-8').strip()

            #Separa variables por coma
            data_list = data.split(',')

            #Bucle para separar el nombre de la variable y el valor
            with datos_meteo_lock:
                for i in data_list:
                    data_var = i.split('=')
                    if len(data_var) ==2:
                        variable = data_var[0].strip()
                        valor = data_var[1].strip()
                        #Añade cada variable a mi diccionario
                        datos_meteo[variable] = valor

            #Creo las variables
```

```
            Temperatura=datos_meteo.get('Temperatura',None)
            Temperatura_maxima=datos_meteo.get('Temperatura_Maxima',None)
            Temperatura_minima=datos_meteo.get('Temperatura_Minima',None)
            Temperatura_media=datos_meteo.get('Temperatura_Media',None)
            Humedad=datos_meteo.get('Humedad',None)
            Humedad_maxima=datos_meteo.get('Humedad_Maxima',None)
            Humedad_minima=datos_meteo.get('Humedad_Minima',None)
            Humedad_media=datos_meteo.get('Humedad_Media',None)
            Vel_Viento=datos_meteo.get('Vel_Viento',None)
            Vel_Viento_maxima=datos_meteo.get('Vel_Viento_Maxima',None)
            Vel_Viento_minima=datos_meteo.get('Vel_Viento_Minima',None)
            Vel_Viento_media=datos_meteo.get('Vel_Viento_Media',None)
            Radiacion=datos_meteo.get('Radiacion',None)

            # Inserta la informacion en la tabla
            cursor.execute("INSERT INTO datos_THVR (fecha, hora, Temperatura,
                Temperatura_Maxima, Temperatura_Minima, Temperatura_Media,
                Humedad, Humedad_Maxima, Humedad_Minima, Humedad_Media,
                Vel_Viento, Vel_Viento_Maxima, Vel_Viento_Minima,
                Vel_Viento_Media, Radiacion) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?,
                ?, ?, ?, ?, ?, ?)", (fecha, hora, Temperatura,
                Temperatura_maxima, Temperatura_minima, Temperatura_media,
                Humedad, Humedad_maxima, Humedad_minima, Humedad_media,
                Vel_Viento, Vel_Viento_maxima, Vel_Viento_minima,
                Vel_Viento_media, Radiacion))
            connection.commit() #guarda cambios

    except Exception as e:
        print(f"Error almacenando datos: {e}")
        traceback.print_exc()


def modbus_PLC():
    try:
        #Declara IP y puerto
        server_ip = '192.168.1.156'
        server_port = 5020

        # Crea una instancia de ModbusServer
        server = ModbusServer(server_ip, server_port, no_block=True)

        #Inicia servidor
        server.start()
        print(f"Servidor Modbus iniciado en {server_ip}:{server_port}")


        while True:
            with datos_meteo_lock:
                # Accede al diccionario y toma los valores de las 4 variables
                    generales
                temperatura=datos_meteo.get('Temperatura',None)
                humedad=datos_meteo.get('Humedad',None)
                vel_viento=datos_meteo.get('Vel_Viento',None)
                radiacion=datos_meteo.get('Radiacion',None)
```

```
                temperatura=float(temperatura) if temperatura is not None else
                    0.0
                humedad=float(humedad) if humedad is not None else 0.0
                vel_viento=float(vel_viento) if vel_viento is not None else 0.0
                radiacion=float(radiacion) if radiacion is not None else 0.0

            #Asigna cada dato a un registro
            server.data_bank.set_holding_registers(0, [int(float(temperatura))])
            server.data_bank.set_holding_registers(1, [int(float(humedad))])
            server.data_bank.set_holding_registers(2, [int(float(vel_viento))])
            server.data_bank.set_holding_registers(3, [int(float(radiacion))])
            sleep(1)

    except Exception as e:
        print(f"Error en el servidor modbus: {e}")
        traceback.print_exc()

    finally:
        #Cierra la conexion
        connection.close()


def socket_PC():
    try:
        #Declara IP y puerto
        server_ip = '192.168.1.156'
        server_port = 8080

        #Crea Socket, lo vincula y se queda en escucha
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.bind((server_ip, server_port))
        server_socket.listen(5) #servidor permite por ej hasta 5 conexiones
            pendientes en cola antes de empezar a rechazar
        print(f"Servidor Socket escuchando en {server_ip}:{server_port}")

        while True:
            #Acepta conexión del cliente correspondiente
            client_socket, addr = server_socket.accept() #addr:contendrá ip y
                puerto del cliente, devuelve esos dos datos
            print(f"Cliente conectado desde {addr}")

            try:
                #Accede a la base de datos meteorológicos
                connection=sqlite3.connect('datos_meteorologicos.db')
                cursor=connection.cursor()

                #Recibe fechas del cliente
                data = client_socket.recv(1024).decode() #maximo recibe hasta
                    1024 bytes
                print("Fechas de la petición recibidas del cliente")

                #Se declaran las dos fechas recibidas
                request = json.loads(data)
                fecha1 = request.get('fecha1')
                fecha2 = request.get('fecha2')
```

```
                #Se recopilan todos los datos de la base entre dichas fechas
                cursor.execute("SELECT * FROM datos_THVR WHERE fecha >= ? AND
                    fecha <= ?", (fecha1, fecha2))
                response=cursor.fetchall() #fetchall: recupera todos los
                    resultados de una consulta SQL (el cursor.execute anterior)

                ## (Imprime por pantalla)
                #for row in response:
                #   print(row)

                #Recoge toda la información y se la envía al cliente
                response_str = "\n".join([",".join(map(str,row)) for row in
                    response])
                client_socket.send(response_str.encode()) #encode: convierte
                    cadena de texto en python a cadena de textos en bytes

            except Exception as e:
                print(f"Error al tomar fechas: {e}")
                traceback.print_exc()

            finally:
                #Cierra socket y el acceso a la base de datos
                client_socket.close()
                connection.close()

    except Exception as e:
        print(f"Error en el servidor socket: {e}")
        traceback.print_exc()


if __name__ == "__main__":
    #Creacion de hilos daemon (hilos que se ejecutan en segundo plano)
    hilo_almacena_THVR = threading.Thread(target=almacena_THVR, daemon=True)
    hilo_modbus_PLC = threading.Thread(target=modbus_PLC, daemon=True)
    hilo_socket_PC = threading.Thread(target=socket_PC, daemon=True)

    #Arranca los 3 hilos creados
    hilo_almacena_THVR.start()
    hilo_modbus_PLC.start()
    hilo_socket_PC.start()

    #Enlaza hilos con el mutex creado para el diccionario
    hilo_almacena_THVR.join()
    hilo_modbus_PLC.join()
    hilo_socket_PC.join()
```

A.3 Arduino con Panel LED

```
// Librerias para el panel
#include "RGBmatrixPanel.h"
#include "bit_bmp.h"
#include "fonts.h"
#include <string.h>
#include <stdlib.h>

// Libreria Modbus RTU
#include <ModbusMaster.h> //Sensor solar mems

// Librería para el Sensor DHT
#include "DHT.h"

// Define pines del panel
#define CLK 11
#define OE  9
#define LAT 10
#define A   A0
#define B   A1
#define C   A2
#define D   A3
RGBmatrixPanel matrix(A, B, C, D, CLK, LAT, OE, false, 64);

// Crear instancia de ModbusMaster(solar mems)
ModbusMaster node;

// Intervalo de tiempo en milisegundos (10 segundos)
const unsigned long intervalo = 10000;

//TH
#define DHTPIN 2   // Pin sensor Temp&Hum
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// Definir pines para RS485 a TTL del solar Mems
#define RE_PIN 3
#define DE_PIN 3

// Funciones para controlar la transmisión RS485 half duplex del Solar Mems
void preTransmission() { // función se llama antes de que el maestro (Arduino)
    envíe datos al esclavo.
  digitalWrite(DE_PIN, HIGH);
  digitalWrite(RE_PIN, HIGH);
}

void postTransmission() { //se llama después de que se han enviado los datos.
  digitalWrite(DE_PIN, LOW);
  digitalWrite(RE_PIN, LOW);
}

// Pin analógico al que se conecta las señales de velocidad del sensor de
    viento
const int velvientoPin = A7;
```

```
const float maxVoltage_vviento = 0.150; // Voltaje máximo en V que admite el
    sensor de viento (era 1V pero la realidad es entorno a 150mV)


unsigned long tiempoAnterior = 0; // Tiempo de la última actualización
float Tmax = -100; // Inicializar con un valor muy bajo
float Tmin = 100; // Inicializar con un valor muy alto
float Hmax = -100; // Inicializar con un valor muy bajo
float Hmin = 100; // Inicializar con un valor muy alto
float Vmax = -100; // Inicializar con un valor muy bajo
float Vmin = 100; // Inicializar con un valor muy alto
float sumaT=0, sumaH=0, sumaV=0;
int contadorT=0, contadorH=0, contadorV=0;
float mediaT=0, mediaH=0, mediaV=0;
float max[]={Tmax, Hmax, Vmax};
float min[]={Tmin, Hmin, Vmin};
int contador[3]={contadorT, contadorH, contadorV};
float suma[3]={sumaT, sumaH, sumaV};
float media[3]={mediaT, mediaH, mediaV};
String datosTexto[] = {"Temperatura", "Humedad","Vel_Viento"};
String datosTexto_max[] = {"Temperatura_Maxima", "Humedad_Maxima","
    Vel_Viento_Maxima"};
String datosTexto_min[] = {"Temperatura_Minima", "Humedad_Minima","
    Vel_Viento_Minima"};
String datosTexto_media[] = {"Temperatura_Media", "Humedad_Media","
    Vel_Viento_Media"};




void setup() {
  Serial.begin(9600); // Inicializar comunicación serial
  dht.begin();
  Serial1.begin(19200); // Solar Mems por defecto BIT RATE en 19200 pero tambié
      n tiene 9600

  // Inicia el panel
  matrix.begin();

  // Inicializar Modbus (direccion del esclavo 1 por defecto)
  node.begin(1, Serial1);
  pinMode(RE_PIN, OUTPUT);
  pinMode(DE_PIN, OUTPUT);

  node.preTransmission(preTransmission);
  node.postTransmission(postTransmission);
}


void loop() {
  // Función imrpime todo los datos
  datos();
}


//Clear screen
```

```
void screen_clear()
{
  matrix.fillRect(0, 0, matrix.width(), matrix.height(), matrix.Color333(0, 0,
      0));
}


void datos(){
  // Variable para la lectura Modbus
  uint8_t radiacion;

  // Obtener tiempo actual en milisegundos
  unsigned long tiempoActual = millis();


  if (tiempoActual - tiempoAnterior <= intervalo) { // Si ha pasado el
      intervalo de tiempo
  //Lectura TH
    float T = dht.readTemperature(); //Lee temperatura (ºC)
    float H = dht.readHumidity(); //Lee temperatura (ºC)
  // Leo señal de velocidad del viento (0-1023 por defecto arduino) // Paso a m
      /s (0-50m/s) con la correción de 1V (max) va a ser 0,150V
    int velvientoAnalog = analogRead(velvientoPin);
    float V = (velvientoAnalog / 1023.0) * 50*1/maxVoltage_vviento;

    float dato[]={T,H,V};

    for (int i = 0; i < 3; i++) {
      if (!isnan(dato[i])) { // Verificar si la lectura es válida
        // Actualizar máximo y mínimo
        if (dato[i] > max[i]) {
          max[i] = dato[i];
        }
        if (dato[i] < min[i]) {
          min[i] = dato[i];
        }
        suma[i] += dato[i]; // Acumular valor para calcular la media
        contador[i]++; // Incrementar contador de lecturas
      }
    }
  }

  else{
    for (int i = 0; i < 3; i++) {
      // Calcular media
      media[i] = suma[i] / contador[i];

      float temp = dht.readTemperature(); //Lee temperatura (ºC)
      float hum = dht.readHumidity(); //Lee humedad (%RH)
      int velo = analogRead(velvientoPin); float vel = (velo / 1023.0) * 50*1/
          maxVoltage_vviento; //Lee velocidad del viento en m/s

      float dato[i]={temp,hum, vel};

      // Imprimir los valores calculados
      Serial.print(datosTexto[i]);
```

```
      Serial.print(" = ");
      Serial.print(dato[i]);
      Serial.print(", ");
      Serial.print(datosTexto_max[i]);
      Serial.print(" = ");
      Serial.print(max[i]);
      Serial.print(", ");
      Serial.print(datosTexto_min[i]);
      Serial.print(" = ");
      Serial.print(min[i]);
      Serial.print(", ");
      Serial.print(datosTexto_media[i]);
      Serial.print(" = ");
      Serial.print(media[i]);
      Serial.print(", ");

      // Reiniciar variables para el próximo intervalo
      max[i]=0;
      min[i]=100;
      media[i]=0;
      contador[i]=0;
      suma[i]=0;
    }
    matrix.setTextSize(1);  // size 1 == 8 pixels high
    matrix.setTextWrap(false); // Don't wrap at end of line - will do ourselves

    radiacion = node.readHoldingRegisters(9, 1);
    Serial.println(radiacion);
    if (radiacion == node.ku8MBSuccess) { // cte. de la bib. ModbusMasteer que
        confirma lectura correcta si ees igual
      int16_t radiacion = node.getResponseBuffer(0); //devuelve el primer valor
          (0) de los registros leidos (solo leo uno)
      Serial.print("Radiacion");
      Serial.print(" = ");
      Serial.print(radiacion);
      Serial.print(", ");

      matrix.setCursor(5, 3);
      matrix.setTextColor(matrix.Color333(0,1,5));
      matrix.println("Radiacion");
      matrix.setCursor(14, 18);
      matrix.setTextColor(matrix.Color333(3,3,3));
      matrix.print(radiacion);
      matrix.println(" W/m2");
      delay (2000);
      screen_clear();
    } else {
      Serial.print("Radiacion");
      Serial.print(" = ");
      Serial.print(radiacion);
      Serial.print(", ");
    }

    Serial.println();
    tiempoAnterior = tiempoActual;
```

```
    matrix.setCursor(0, 3);
    matrix.setTextColor(matrix.Color333(0,1,5));
    matrix.println("Temperatura");
    matrix.setCursor(11, 18);
    matrix.setTextColor(matrix.Color333(3,3,3));
    matrix.print(temp);
    matrix.drawRoundRect(43, 18, 2, 2, 2, matrix.Color333(3,3,3));
    matrix.println(" C");
    delay (2000);
    screen_clear();

    matrix.setCursor(12, 3);
    matrix.setTextColor(matrix.Color333(0,1,5));
    matrix.println("Humedad");
    matrix.setCursor(3, 18);
    matrix.setTextColor(matrix.Color333(3,3,3));
    matrix.print(hum);
    matrix.println(" % RH");
    delay (2000);
    screen_clear();

    matrix.setCursor(14, 3);
    matrix.setTextColor(matrix.Color333(0,1,5));
    matrix.println("Viento");
    matrix.setCursor(8, 18);
    matrix.setTextColor(matrix.Color333(3,3,3));
    matrix.print(vel);
    matrix.println(" m/s");
    delay (2000);
    screen_clear();
  }

  delay(1000);
}
```

A.4 Petición de Clientes

A.4.1 Cliente Modbus

```
from pyModbusTCP.client import ModbusClient
client = ModbusClient(host="192.168.1.156", port=5020)
client.open()

print(client.read_holding_registers(0))

print(client.read_holding_registers(1))

print(client.read_holding_registers(2))

print(client.read_holding_registers(3))

client.close()
```

A.4.2 Cliente Socket

```
import os
import socket
import json

def get_user_input():
    fecha1 = input("Ingrese primera fecha (YYYY-MM-DD): ")
    fecha2= input("Ingrese segunda fecha (YYYY-MM-DD): ")
    return {"fecha1": fecha1, "fecha2": fecha2}

def receive_all(socket):
    buffer_size = 4096
    received_data = b"" #Inicialmente es una cadena de bytes vacía.
    while True:
        data = socket.recv(buffer_size)
        if not data:
            break
        received_data += data
    return received_data

def main():
    # Ruta de la carpeta donde se guardarán los archivos
    directory = 'C:\\Users\\diego\\TFG_diego\\Historicos_almacenados' #Usar
        barras dobles (\\) o un barra simple (/) en rutas en python
    if not os.path.exists(directory):
        os.makedirs(directory)

    # Servidor: ip y puerto (poner siempre aparte para cuando cambie)
    server_ip = '192.168.1.156'
    server_port = 8080

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((server_ip, server_port))
```

```
    # Solicitar al usuario que ingrese la fecha y la hora
    solicitud = get_user_input()

    # Enviar solicitud al servidor en formato JSON
    client_socket.send(json.dumps(solicitud).encode())
    print("Fecha y hora enviadas a servidor")

     # Recibir respuesta del servidor
    response = receive_all(client_socket).decode()
    print("Respuesta recibida del servidor")

    # Guardar el archivo recibido con un nombre específico en la carpeta
        especificada
    filename = f'datos_meteorologicos_{solicitud["fecha1"]}_{solicitud["fecha2
        "]}.txt'
    filepath = os.path.join(directory, filename)

    # Guardar el archivo recibido con el nombre especificado
    with open(filepath, 'w') as file:
        file.write(response)
    print(f"Archivo guardado en: {filepath}")

    client_socket.close()

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

Apéndice B
Guía de uso

La siguiente guía de uso ha sido desarrollada de cara a la instalación futura de este proyecto, que incluirá nuevos sensores no conexionados ni programados para este diseño, por lo que su adición, junto con los pasos a seguir para el funcionamiento del sistema, han sido descritos a continuación.

B.1 Inclusión de nuevos sensores

Como se ha visto, no todos los sensores son fáciles de añadir al sistema, no solo en cuenta a nivel de programación, sino también a nivel hardware. Por ello, se va a explicar como conectar sensores con salida en voltaje o en corriente, pues son lo más habitual de ver, y además, los sensores DHT11 y el solar con salida RS-485 utilizados, son casos excepcionales y ya se han descrito en el documento.

B.1.1 Salida en voltaje

En este proyecto únicamente el sensor de viento mostraba una salida en voltaje gracias al datalogger de la marca Young. Sin embargo, su programación correspondiente se ha visto alterada debido a una falta de amplificación de la salida, ya que esta no era acorde con lo esperado.

Para leer este tipo de señales, la salida del sensor se conecta a un pin analógico del Arduino. Este, por defecto de la placa, leerá la señal con el comando *analogRead* en un rango comprendido de 0 a 1023, ya que el microcontrolador posee un convertidor analógico-digital de 10 bits, teniendo por tanto 2 elevado a la 10 niveles, es decir, 1024 niveles posibles. Seguido a esto basta con convertir el valor a las unidades correspondientes, usando el comando *map*.

A continuación se muestra un ejemplo de como se leerían la velociad y la dirección del viento, para un rango de 0 a 100 m/s y de 0 a 360°:

```
// Pines analógicos conectados las señales de velocidad y dirección del viento
const int velvientoPin = A0; // Sea A0 la velocidad del viento
const int dirvientoPin = A1; // Sea A1 la dirección del viento

void setup() {
  Serial.begin(9600); // Iniciar comunicación serie a 9600 bps
}

void loop() {
  // Lee señal de velocidad del viento (0-1023 por defecto arduino)
  int velvientoAnalog = analogRead(velvientoPin);
  // Convierto valor analógico a velocidad del viento
  float velviento = map(velvientoAnalog, 0, 1023, 0, 100); //de 0 a 100 m/s (o
      el rango que tenga)

  // Leo señal de dirección del viento (0-1023)
  int dirvientoAnalog = analogRead(dirvientoPin);
  // Convierto valor analógico a dirección del viento
  float dirviento = map(dirvientoAnalog, 0, 1023, 0, 360); //de 0 a 360 grados

  // Imprimo valores por el puerto serie
  Serial.print("Velocidad del viento: ");
  Serial.print(velviento);
  Serial.println(" m/s");
  Serial.print("Dirección del viento: ");
  Serial.print(dirviento);
  Serial.println(" grados");

  delay(1000);
}
```

B.1.2 Salida en intensidad

Para tratar señales cuya salida es en corriente, el proceso será igual que en el apartado, pues se usará un convertidor de corriente a voltaje. La salida más habitual en módulos de corriente es de un rango de 4 a 20 mA, por que el convertidor será capaz de transformar este rango de intensidad a rangos de voltaje de 0 a 3.3, 5 o 10 V.

Suponiendo que se tiene un sensor con salida en 4-20mA, que se alimenta con una fuente externa de 24V y un módulo convertidor habitual, se debe conectar de la siguiente manera:

- Terminal positivo del sensor al positivo de la fuente.
- Terminal negativo del sensor a I+ del convertidor.
- Terminal negativo de la fuente de alimentación al I- del convertidor
- Salida del convertidor Vcc a 5V de la placa de Arduino.
- Salida del convertidor Vout a pin analógico de la placa de Arduino.
- Salida del convertidor GND a GND de la placa de Arduino.

Además de las dos entradas en corriente y las tres salidas en voltaje, estos convertidores se componen de más elementos. Empezando desde la entrada, hasta la salida, se tiene un potenciómetro ZERO, un potenciómetro SPAN y dos jumpers. Por un lado, los potenciómetros sirven para calibrar el convertidor de la siguiente manera: primero, con una corriente de 4mA, se ajusta el potenciómetro ZERO hasta que Vout sea igual a 0V, y segundo, se pasa a 20mA, y se ajusta SPAN hasta que Vout sea igual a 5V. Por otra parte, los dos jumpers sirven para definir el rango de voltaje a la salida. Si solo se conecta el primer jumper, la salida será de 0-10V, si solo se conecta el segundo, será de 0-2.5V, si no se conecta ninguno, será de 0-3.3V, y si se conectan ambos, la salida será de 0-5V.

B.2 Efecto de un nuevo sensor sobre los códigos

Los programas utilizados en este proyecto se han diseñado para que la adición de nuevos sensores al sistema sea muy sencilla, tanto en Arduino como en la Raspberry, por lo que se podrán remplazar o añadir nuevos equipos sin necesidad de mucho esfuerzo.

Empezando desde el microcontrolador, tanto en el Arduino Mega conectado a la Raspberry como en el Arduino Mega conectado a la pantalla LED, se realizan los siguientes cambios:

1. Si el sensor, que mide X, lo requiere, añadir nuevas librerías, y además, definir un pin de conexión, seguramente analógico si la señal del sensor era voltaje o corriente.

2. Para medir X, inicializar, de igual manera que el resto de variables, Xmin, Xmax, sumaX, contadorX, mediaX, y añadirlos al final de los vectores correspondientes, además de añadir la cadena de caracteres respectivas a los cuatro vectores *String*.

3. Dentro de *setup*, inicializar, si es necesario, la nueva instancia, pin o puerto serie.

4. Dentro de la función en bucle, en el *if*, realizar la lectura del sensor y añadirla al final del vector *dato*, e incrementar el bucle en uno.

5. Dentro de la función en bucle, en el *else*, incrementar el bucle en uno, leer la nueva señal y ampliar el vector dato al final con la nueva variable.

6. En el caso de la escritura por el panel LED *matrix*, añadir la estructura diseñada para el resto de variables, alterando los *print* del nombre, de la variable y de las unidades, y adaptando, de forma experimental, el comando *setCursor*[1] para centrar la información en el panel.

Además, es importante un detalle para visualización en la pantalla LED, el tiempo que se muestra cada variable. Actualmente se define en unos dos segundos, tras los cuales se borra y aparece la siguiente. Como el tiempo de muestreo del sistema se ha definido en diez segundos, se podrán mostrar cinco variables (una cada dos segundos) medidas en el mismo tiempo de muestreo. Si se amplia el número de señales a más de cinco, se tendrá que cambiar el tiempo en el que se visualizan estas en la pantalla, para que la suma de todos ellos no supere 10 segundos, pues no daría tiempo a mostrar todas las variables

[1] *matrix.setCursor(a,b)*: este comando indica la posición del píxel en la pantalla, más elevado, desde donde se empezará a escribir. El primer término 'a', es el número de píxeles horizontales, y el segundo, 'b', el número de píxeles verticales (dejar en 3 por defecto)

En cuanto al código de la Raspberry, también será relativamente sencillo la adaptación de un nuevo sensor

1. Fuera del código principal, como se explicó en el capítulo 5, crear una nueva tabla en *sqlite3*, por ejemplo, datos_THVRX, donde se añadirá al final la nueva variable: X REAL.

2. En el primer hilo, *almacena_THVR*, cuando se crean las variables, justo después de la inclusión de los datos al diccionario, crear o eliminar las variables con las que se este trabajando, de la misma forma que están el resto actualmente.

3. Justo después, al insertar la información en la base de datos, cambiar el nombre de la tabla a datos_THVRX, añadir un nuevo signo '?', y al final del todo, el nombre de la nueva variable X creada.

4. Dentro del segundo hilo, cuando se accede al diccionario, añadir una línea que tome el valor de la nueva variable de este, de igual forma que el resto. Seguido, añadir una nueva línea que asigne un nuevo registro (que no ese utilizado) a esta variable, de igual forma que con el resto de datos.

5. En el tercer hilo, solo habrá que actualizar el nombre de la nueva tabla, cuando se accede a la base de datos.

Cabe destacar, que estos pasos a seguir para añadir un nuevo sensor al sistema, se tienen que seguir de manera contraria, si se quiere eliminar una variable que ya se este midiendo en el sistema.

B.3 Guía rápida de uso

Para que el software del sistema funcione correctamente, con los sensores con los que se cuenta actualmente, se tienen que seguir los siguientes pasos:

1. **Conexiones de los sensores**: realizar todas las conexiones explicadas en el Capítulo 4, recordando conectar las señales de los sensores a ambas placas de Arduino Mega.

2. **Arduino Mega conectado a la Raspberry Pi**: descargar la librería ModbusMaster en la aplicación Arduino del ordenador, si no esta instalada, desde las librerías de Arduino, y cargar el primer código del Apéndice 1.

3. **Arduino Mega conectado a la Pantalla LED**: descargar la librería ModbusMaster en la aplicación Arduino del ordenador, si no esta instalada, desde las librerías de Arduino, y cargar el segundo código del Apéndice 1.

4. **Raspberry Pi**: instalar todos los módulos necesarios y ejecutar en una terminal el primer código Python del Apéndice 1, tal y como se hace en el apartado 5.3.

5. **Solicitud Modbus**: cargar en un terminal del PC el cuarto código del Apéndice 1, e instalar python si fuera necesario.

6. **Solicitud Socket**: cargar en un terminal del PC el quinto código del Apéndice 1, e instalar python si fuera necesario.

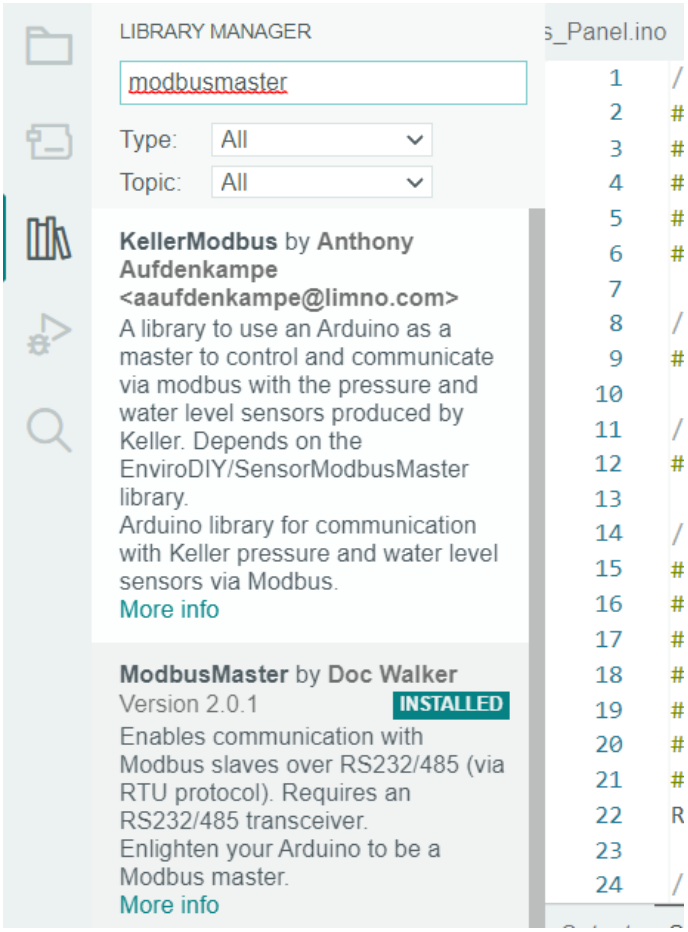

Figura B.1 Instalación de la librería ModbusMaster en Arduino.

Para instalar correctamente los paquetes y módulos utilizados en el programa en Python, se debe ejecutar desde una terminal de la Raspberry distintos comandos. Primero se actualiza el sistema, ante posibles cambios, seguido se instala Python, y por último el módulo que se vaya a utilizar, si este no viene por defecto con Python. Se muestra el caso de la instalación de *serial* en la Raspberry Pi:

```
sudo apt update
sudo apt upgrade

sudo apt install python3
sudo apt install python3-pip

pip3 install pyserial
```

Bibliografía

[1] *Anemoveleta Young 05103 con un datalogger de Campbell Scientific*, https://www.campbellsci.es/05103.

[2] *Anemómetro IM512CD de la marca METOS*, https://metos.global/es/wind-sensors/.

[3] *Arduino Mega 2560*, https://store.arduino.cc/products/arduino-mega-2560-rev3/.

[4] *Arduino UNO R4 WiFi*, https://store.arduino.cc/products/uno-r4-wifi/.

[5] *Conectores de entrada, salida y alimentación de la pantalla LED RGB-Matrix-P4-64x32*, https://www.waveshare.com/rgb-matrix-p3-64x32.htm?amazon.

[6] *Documentación sobre una base de datos en SQLite*, https://sqlite.org/.

[7] *Estación Meteorológica Automática AWS310 de la marca Vaisala*, https://www.vaisala.com/en/products/weather-environmental-sensors/automatic-weather-stations.

[8] *Ficha Técnica del Módulo Convertidor MAX485 de RS-485 a TTL*, https://www.analog.com/media/en/technical-documentation/data-sheets/max1487-max491.pdf.

[9] *Pantalla LED RGB-Matrix-P4-64x32*, https://www.waveshare.com/wiki/RGB-Matrix-P4-64x32#Overview.

[10] *Piránometro LI-200R de la marca LI-COR*, https://www.licor.com/env/products/light/pyranometer.

[11] *Python*, https://www.python.org/.

[12] *Raspberry Pi 4 Model B*, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[13] *Raspberry Pi 4 Model B Datasheet*, https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf.

[14] *Raspberry Pi Documentation*, https://www.raspberrypi.com/documentation/.

[15] *Sensor de monitoreo de suciedad en los paneles solares DustIQ de la marca KippZonen*, https://www.kippzonen.com/Product/419/DustIQ-Soiling-Monitoring-System.

[16] *Sensor de Temperatura y Humedad DHT11*, https://diyables.io/products/dht11-temperature-and-humidity-sensor-module.

[17] *Sensor Solar ISS-D5 de la marca Solar MEMS*, https://solar-mems.com/solar-tracking/iss-dx/.

[18] *Uso de una base de datos SQL en Python, a través de sqlite3*, https://docs.python.org/3/library/sqlite3.html.

[19] *Getting Started - What is Arduino?*, February 2018, https://www.arduino.cc/en/Guide/Introduction/.

[20] Eugenio López Aldea, *Raspberry pi. fundamentos y aplicaciones*, RA-MA Editorial, 2017, Available: https://www.ra-ma.es/libro/raspberry-pi-fundamentos-y-aplicaciones_83263/.

[21] Mohamed Khalifa Boutahir, Yousef Farhaoui, Mourade Azrour, Imad Zeroual, and Ahmad El Allaoui, *Effect of Feature Selection on the Prediction of Direct Normal Irradiance*, Big Data Mining and Analytics **5** (2022), no. 4, 309–317.

[22] Cameron, *Wired on Arduino and open source computing*, https://creativecommons.org/2008/10/22/wired-on-arduino-and-open-source-computing/.

[23] Departamento de Ingeniería de Sistemas y Automática, *Apuntes de Sockets y Concurrencia de Hilos de la asignatura Informática Industrial*, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla (2023-2024).

[24] Departamento de Ingeniería de Sistemas y Automática, *Apuntes de MODBUS de la asignatura de Domótica*, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla (2023-2024).

[25] Michael Gostein, Bill Stueve, Kendra Passow, and Alex Panchula, *Evaluating a model to estimate GHI, DNI, DHI from POA irradiance*, (2016), 0943–0946.

[26] IRENA, *Renewable power generation costs in 2022*, , International Renewable Energy Agency, Abu Dhabi (2023).

[27] M. Izadi N.I. Ahmad M.A.M Radzi N.H. Zaini, M. Z. Ab Kadir and N. Azis, *The Effect of Temperature on a Mono-crystalline Solar PV Panel*, IEEE Conference on Energy Conversion (CENCON) (2015), 249–253.

[28] C. Riordan P. Faine, Sarah R. Kurtz and J. M. Olson, *The influence of spectral solar irradiance variations on the performance of selected single-junction and multijunction solar cells*, Solar Energy Research Institute, 1617 Cole Blvd., Golden, CO 80401 (U.S.A.) (1990), Solar Cells, 31 (1991) 259–278.

[29] Tamboli, Sadik and Rawale, Mallikarjun and Thoraiet, Rupesh and Agashe, Sudhir, *Implementation of Modbus RTU and Modbus TCP communication using Siemens S7-1200 PLC for batch process*, 2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM) (2015), 258–263.

[30] Guido van Rossum, *What is Python? Executive Summary*, 1998, https://www.python.org/doc/essays/blurb/.