Universidad Zaragoza
1542

Trabajo Fin de Grado

Desarrollo de una estación meteorológica mediante Raspberry Pi con predicción del tiempo

Development of a weather station using Raspberry Pi with weather forecasting

Autora

Alba Carmona García

Director

Bonifacio Martín del Brío

GRADO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA

ESCUELA DE INGENIERÍA Y ARQUITECTURA DE ZARAGOZA

2021-2022

Desarrollo de una estación meteorológica mediante Raspberry Pi con predicción del tiempo

RESUMEN

Actualmente, las consultas sobre el clima y su previsión a futuro son recurrentes; simplemente con una conexión a internet, desde nuestro teléfono móvil, tableta u ordenador, podemos acceder a un servidor que nos proporcione esos datos, que en cierto aspecto nos condicionan. Sin embargo, las condiciones meteorológicas pueden cambiar mucho localmente, por ejemplo, en una ciudad en un barrio puede caer una tormenta y en otro barrio no llover en absoluto. Por otro lado, una buena previsión del tiempo, adaptada a las condiciones locales, puede resultar beneficiosa para el ahorro de recursos como el agua, como ocurre en el caso de los huertos urbanos o domésticos.

Por ello, en este proyecto se realiza el diseño de una estación meteorológica doméstica que, además de incluir las funcionalidades básicas que se esperan de estos dispositivos relativas al conocimiento de las condiciones meteorológicas, permita al usuario una monitorización más completa del entorno local y de las variables que definen dichas condiciones en un punto geográfico concreto.

El proyecto se ha llevado a cabo entorno a una *Raspberry Pi*, junto con dos dispositivos compatibles: *Camera Raspberry Pi v2.1* y módulo *SenseHat*, el cual incluye sensores y pantalla matricial. El lenguaje de programación empleado ha sido Python.

Además de las funcionalidades básicas habituales en una estación meteorológica doméstica estándar, en este trabajo se ha trabajado especialmente en tres aspectos más avanzados: i) el diseño de una interfaz que proporciona al usuario datos actuales sobre la temperatura y estado del cielo sin necesidad de acceder a ningún servicio web, simplemente con un vistazo; ii) pronóstico del tiempo local, basado en las medidas recogidas por la propia estación, de los principales fenómenos atmosféricos (lluvia, nubosidad presente y temperaturas); iii) creación de una plataforma, vinculada a internet, que recoge y pone a disposición del usuario toda la información recabada y generada.

Además, mediante una microcámara, se ha implantado un método de grabación del entorno que permite observar la evolución de este a lo largo de un día en un vídeo de corta duración (*timelapse*), con el fin de monitorizar el estado de un terreno, jardín, huerto o del mismo cielo.

ABSTRACT

Nowadays, queries about the weather and its future forecast are recurrent, since only with an internet connection we can access a server that provides us with this data, which in a certain aspect conditions us. However, weather variables depends on local conditions; for instance, in the case of a storm, in a city it can rain in a district but not in another. Thus, for save irrigation water for urban gardens, precise weather forecast based on local measures can be very valuable.

For this reason, this project develops the design of a domestic weather station that, in addition to including the basic functionalities expected of these devices, relating to knowledge of weather conditions, allows the user to monitor the local environment and the variables that define these conditions more specifically and completely.

Among the possible improvements that could be made to a standard weather station, this work has focused on two main aspects: i) the design of a graphic interface that provides the user with current temperature and sky data without the need to access any web service, simply at a glance; ii) a forecast, based on the local measurements collected by the station itself, of the main atmospheric phenomena (rain, cloudiness, present and limit temperatures).

In addition, a method of recording the environment, based on a microcamera, has been implemented, that allows the evolution of the environment over the course of a day to be recorded in a short video (timelapse), in order to be able to monitor the state of a terrain or the sky itself.

The project has been carried out on a *Raspberry Pi*, together with two compatible devices: *Camera Raspberry Pi v2.1* and *SenseHat,* and the programming language used was Python.

Contenido

1. Introducción

1.1. Motivación y objetivos

La meteorología se ha convertido en un campo de gran interés dada la situación de cambio climático, sucediendo fenómenos atmosféricos en temporadas inusuales y de gran magnitud como granizo, grandes tormentas, o lluvias torrenciales. Dada la incertidumbre que esta situación produce, la posibilidad de contar con un dispositivo que monitoriza y advierte de ellos es de gran utilidad.

Por otro lado, es fundamental el ahorro de un bien tan escaso como el agua, por lo que en el caso del riego de jardines y huertos urbanos sería de gran interés disponer de un dispositivo que realice previsiones más ajustadas a las condiciones locales que las genéricas y automatizadas que dan los servicios online disponibles en nuestros teléfonos, las cuales vienen dadas directamente por modelos meteorológicos sin ninguna corrección posterior por un ser humano, por lo que fallan mucho en el caso de fenómenos meteorológicos locales, como situaciones de niebla o tormentas.

La intención de este proyecto incluye, además de la idea de que el usuario disponga de información actualizada sobre el clima presente en una ubicación concreta, así como las previsiones para días posteriores, permitir al usuario realizar grabaciones de video diarias de los elementos o terrenos que sean de su interés. Las aplicaciones de estas grabaciones abarcan desde el control del estado de una planta, jardín o huerto, hasta la visualización del estado del cielo.

La motivación que impulsa el desarrollo de este proyecto es proporcionar una herramienta doméstica de elevadas prestaciones, al alcance de todos los usuarios, que permita el acceso a la información recopilada por la estación meteorológica desde cualquier lugar y en cualquier momento consultando la web que almacena estos datos. La información actual estará, a su vez, disponible de forma gráfica en el propio dispositivo, de tal forma que con un simple vistazo sea posible conocer las condiciones climatológicas actuales. Los objetivos propuestos para el desarrollo del proyecto son:

- Diseñar y construir un prototipo de estación meteorológica basado en una *Raspberry Pi* como centro neurálgico, que controle dos módulos externos: *SenseHat* (que incluye sensores de temperatura, presión y humedad, además de pantalla matricial luminosa 8x8) y *Camera Pi*.

- Diseñar iconos asociados a la situación climática que representados en la pantalla 8x8 del módulo *SenseHat*, de manera que proporcionen al usuario información directa e intuitiva sobre el estado actual del cielo.

- Conocer y entender los posibles modelos de predicción para obtener un pronóstico lo suficientemente ajustado a la realidad.

- Programación de las funcionalidades del dispositivo en lenguaje *Python*, haciendo uso de los paquetes necesarios para el manejo de sensor y cámara, y de librerías dedicadas a trabajar con bases de datos como *pandas* o *csv*, entre otras.

- Validación de las medidas y predicciones obtenidas mediante bases de datos meteorológicas fiables.

- Realizar grabaciones de vídeo de tipo *TimeLapse* mediante la *CameraPi*, que pueda ser cargado en la nube diariamente de forma que sean accesibles para el usuario (por ejemplo, cuando este se encuentra de viaje).

- Desarrollar un servidor web en el que ofrecer la información recopilada y procesada por el dispositivo.

1.2. Estructura del documento y cronograma

La organización que sigue este documento permite explicar el diseño y desarrollo del sistema descrito, tanto a nivel de hardware como de software. El código implementado para crear la aplicación queda reflejado en los anexos, al final de este documento.

La memoria se estructura en capítulos, siguiendo el siguiente orden:

- Capítulo 2. En este capítulo se definen brevemente las variables meteorológicas que son objeto de estudio, junto con los mecanismos de medición comúnmente empleados para estas tareas. Se incluye una introducción a la ciencia de la predicción y su evolución a lo largo de los años.

- Capítulo 3. Listado de los componentes que conforman el hardware del proyecto, incluyendo especificaciones técnicas y futuras funcionalidades dentro de este, que han derivado en la selección de ellos.

- Capítulo 4. Proceso de desarrollo del software necesario para cumplimentar los objetivos estipulados en el apartado anterior. Se incluye el uso de algoritmos de aprendizaje automático para elaborar previsiones locales del tiempo.

- Capítulo 5. Herramientas en la web empleadas para mostrar la información recopilada y generada al usuario.

- Capítulo 6. Análisis de los resultados obtenidos tras días de funcionamiento de la estación meteorológica, estableciendo comparaciones con datos recabados en estaciones estatales de meteorología.

- Capítulo 7. Posibles mejoras a nivel tanto de hardware como de software que suponen un aumento de la eficiencia y disminuyen el porcentaje de error.

En la *Figura 1* se plasman las tareas que se han definido para la realización de este proyecto, en forma de diagrama de Gantt. Dichas tareas abarcan desde el estudio de las posibles soluciones, hasta la comprobación del correcto funcionamiento y el proceso de documentación.

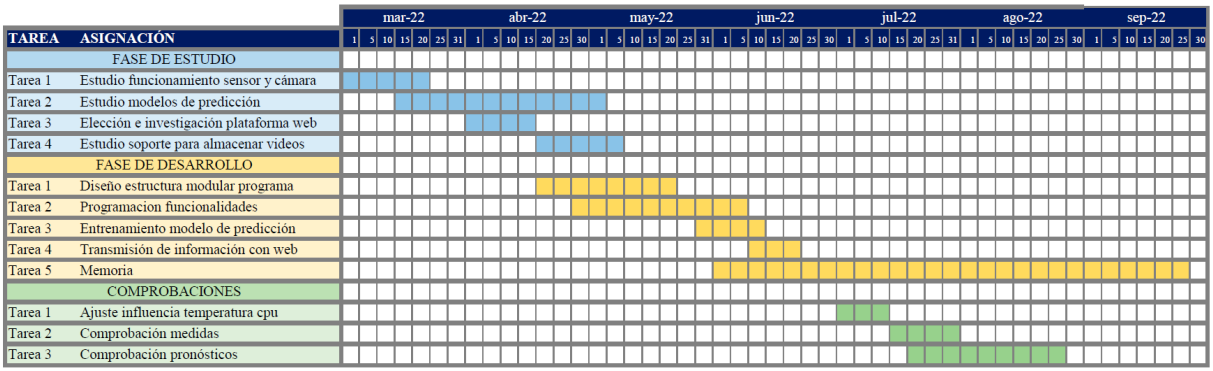


Figura 1. Diagrama de Gantt

2. Marco de referencia.

2.1. Variables meteorológicas y herramientas de medición

Los instrumentos meteorológicos son cada día más precisos y esa precisión permite a los meteorólogos realizar predicciones cada vez más exactas. Entre otras variables, una estación meteorológica proporciona información sobre [1]:

- Presión atmosférica:

El aire presente en la atmósfera tiene un peso, y por lo tanto ejerce una fuerza sobre la superficie de la tierra debida al efecto de la gravedad, esta fuerza por unidad de superficie es a lo que llamamos presión atmosférica. Esta magnitud tiene una gran dependencia con la altitud, dado que a mayor altura menor será la cantidad de aire sobre nosotros, siguiendo la relación que se observa en la *Figura 2*. Depende, además, de otras muchas variables, y es la relación que existe entre esta magnitud y el tiempo en un lugar lo que la hacer una variable fundamental en el ámbito de la meteorología.

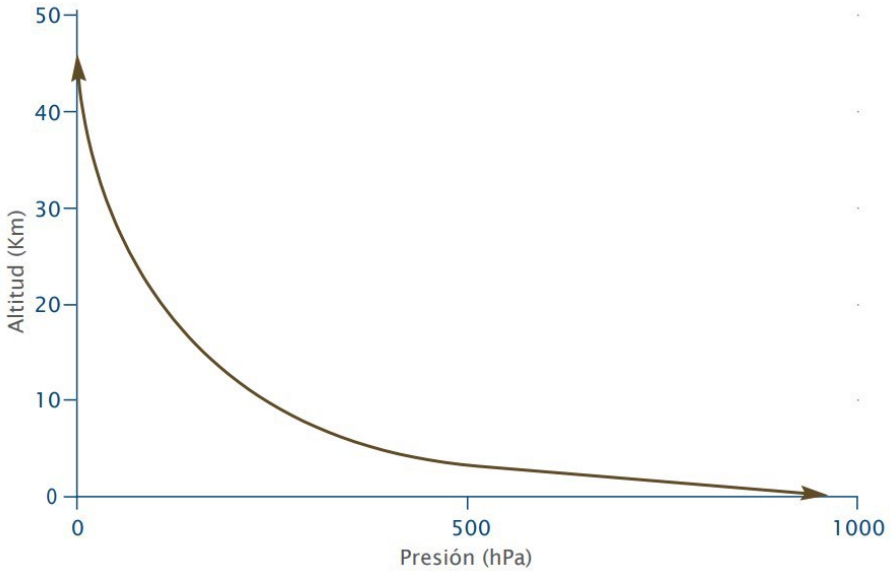


Figura 2. Perfil vertical de la presión atmosférica [1].

Se mide mediante un barómetro, antiguamente eran comunes los de mercurio ya que se consideraban los más precisos, pero actualmente el uso de este tipo de barómetro se desaconseja principalmente por la toxicidad del mercurio.

Por otra parte, existen los sensores de presión, que "simulan" el funcionamiento de los barómetros. Algunos de los tipos que se pueden encontrar son: sensores de presión con tecnología de galgas, capacitivos, piezorresistivos o resonantes. El principio de funcionamiento de las tres primeras tecnologías está basado en la

deflexión de un cuerpo de medición, mientras que la última se centra en los cambios en la frecuencia de resonancia en un mecanismo de detección.

- Temperatura:

Formalmente, la temperatura es la magnitud relacionada con la rapidez del movimiento de las partículas que constituyen la materia. Cuanta mayor agitación presenten éstas, mayor será la temperatura. [2]

Esta magnitud se mide con termómetros, que habitualmente contienen alcohol (antiguamente mercurio). La propiedad de estas sustancias que permite conocer la temperatura es que se contraen con el frio y se expanden con el calor.

En cuanto a los sensores de temperatura, los más comunes son: termopares (generan una corriente entre dos metales que presentan comportamientos diferentes frente a la temperatura), termorresistencias (formados por resistencias cuya conductividad varía en función de la temperatura) o sensores electrónicos (disponen de dispositivos electrónicos que generan una señal dependiente de la temperatura).

- Humedad:

Cantidad de vapor de agua presente en el aire. En meteorología suele hablarse de humedad relativa, esta se expresa en tanto por ciento y da información sobre la cantidad máxima de vapor de agua que puede contener una masa de aire antes de transformarse en agua, de esta forma, un 100% de humedad relativa indicaría que una mayor cantidad de vapor de agua se convertirá en agua líquida o hielo. [2]

Una posible forma de medir esta variable es con un higrómetro, que consiste en una fibra sintética que se estira o contrae en función del vapor de agua existente en el aire. Otra opción para hacer las mediciones con exactitud es el uso de un termómetro seco y otro húmedo, obteniendo la humedad relativa de la diferencia entre ellos.

Los principales tipos de sensores de humedad empleados en la actualidad se diferencian por su principio de funcionamiento y son: de conductividad, miden el nivel de conductividad eléctrica que produce el agua al pasar por unas rejillas, capacitivos, los capacitores sufren cambios en su capacidad dependiendo de la humedad a la que estén sometidos, o resistivos, siguen el principio de conductividad de la tierra (mayor cantidad de agua presente, mayor conductividad de la tierra).

Otras magnitudes como la velocidad del viento, la precipitación o la nubosidad ofrecen también información valiosa sobre el estado del cielo, pero su medida encarece mucho el precio de una estación meteorológica pensada para el ámbito doméstico y con

las variables comentadas anteriormente es posible describir con suficiente exactitud la situación climática actual.

2.2. Predicción meteorológica

Consiste en anticipar el valor que tendrán las variables meteorológicas, que hemos comentado anteriormente, en una cierta región.

La predicción meteorológica puede realizarse mediante técnicas estadísticas, pero la forma más habitual, y la que normalmente ofrece mejores resultados, está basada en la resolución de las ecuaciones matemáticas correspondientes a las leyes físicas que describen el comportamiento de la atmósfera [2]. Conociendo el valor de las variables meteorológicas de un estado inicial, podemos obtener una descripción del futuro estado de la atmósfera tras aplicar dichas fórmulas y, por lo tanto, predecir el tiempo que hará en el futuro que hayamos definido a la hora de resolver las ecuaciones.

Esta metodología de predicción supone un alto costo computacional, ya que se trata de ecuaciones diferenciales en derivadas parciales y no siempre existe una única solución que de valor a las variables de interés. Por ello existen los modelos numéricos de predicción.

Actualmente estos modelos son la principal herramienta de predicción de los servicios meteorológicos modernos. Desde su primera aparición, han evolucionado hasta poder realizar previsiones lo suficientemente próximas a la realidad con un plazo de cinco días. El principal objetivo de esta ciencia es tener la capacidad de conocer, con la máxima antelación y precisión posibles, el inicio de fenómenos atmosféricos.

- Modelos de circulación general o globales

  Se trata de modelos cuya prioridad es la simulación del flujo de circulación general atmosférico, por tanto, su rejilla de integración debe cubrir toda la Tierra. Se usan principalmente para predicciones a medio plazo, estacionales y climáticas ya que no necesitan ningún dato externo (salvo las condiciones iniciales) para realizar las predicciones.

- Modelos de área limitada (LAM)

  Se emplea cuando el área de interés es muy concreta y, por lo tanto, la rejilla plana cubre solo esa área. Precisan de las predicciones de otro modelo cuya rejilla contenga su rejilla de integración, que considerará condiciones de contorno. Su aplicación está pensada para predicciones a corto plazo, de hasta 48 horas.

Como resultado de aplicar estos modelos, se obtienen las variables del modelo y sus derivadas en los nodos de la rejilla de integración y en cada paso de tiempo hasta definir por completo la predicción planteada.

De estos datos derivan los productos necesarios para la predicción del tiempo, cuyo conjunto se denomina "postproceso del modelo". Algunos de los productos más empleados son los mapas del tiempo o los "meteogramas". En la *Figura 3* presentamos el meteograma de la ciudad de Zaragoza a lo largo de siete días.

En la predicción de fenómenos atmosféricos como la lluvia influyen gran cantidad de variables, además de las tres comentadas anteriormente, como la intensidad del viento y su dirección. Los científicos especializados en esta ciencia, meteorólogos, estudian la evolución de dichos parámetros continuamente para ofrecer predicciones lo más ajustadas a la realidad posibles, pero cabe destacar que el porcentaje de acierto en estos pronósticos es menor cuanto mayor es el espacio temporal que se pretende predecir y cuanta menos información disponible maneja el modelo. En el caso de la precipitación, el mayor porcentaje obtenido hasta ahora no alcanza el 90% para previsiones a seis horas vista [3].

Por otro lado, cabe destacar que las previsiones que ofrecen las aplicaciones que todos consultamos en nuestros teléfonos móviles muestran la salida que directamente da un modelo meteorológico, sin ninguna intervención o "cocinado" por parte de un meteorólogo, motivo por el que estas previsiones a menudo fallan a nivel local, especialmente en situaciones como tormentas o nieblas.

Figura 3. Meteograma Zaragoza [4].

3. Hardware

En este capítulo se presentan los dispositivos que componen el prototipo hardware de esta estación. Todos forman parte de la misma familia (*Raspberry Pi*), de esta forma facilitamos la compatibilidad y comunicación entre ellos. En la *Figura 4* se muestra la estructura incluyendo conexiones y transmisión de datos, en un diagrama de bloques.

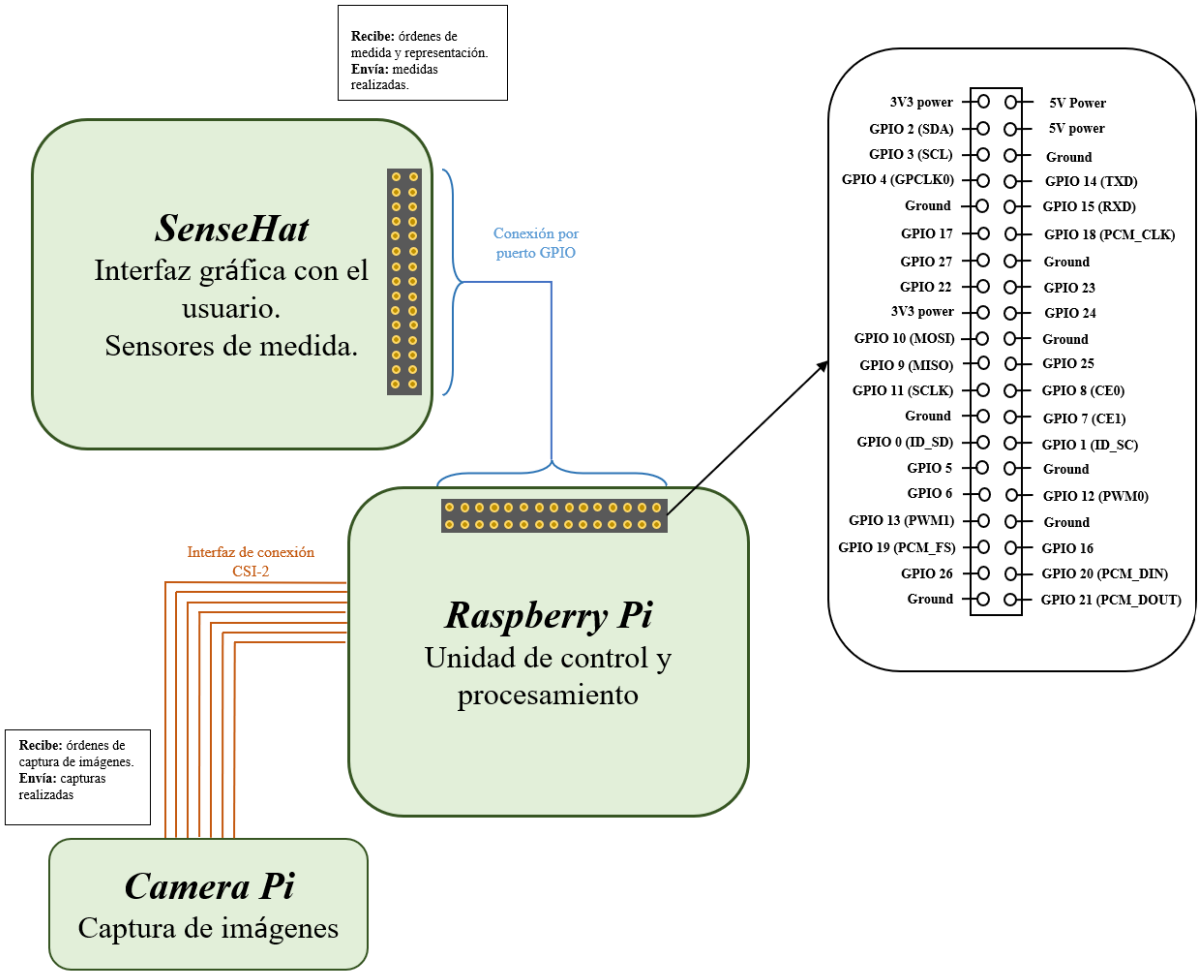


Figura 4. Diagrama de bloques del hardware.

El montaje final de este proyecto, basado en el diagrama de bloques comentado previamente, resulta en la disposición mostrada en la *Figura 5*.Dónde, dado el ángulo de la imagen, elegido para observar los dos módulos conectados a la *Raspberry Pi* correctamente, no se aprecia la conexión entre *Raspberry Pi* y *SenseHat*, pero se visualiza el puerto GPIO que ofrece esta conexión, entendiendo que el segundo dispositivo integra los pines que llevan a cabo esta conexión.

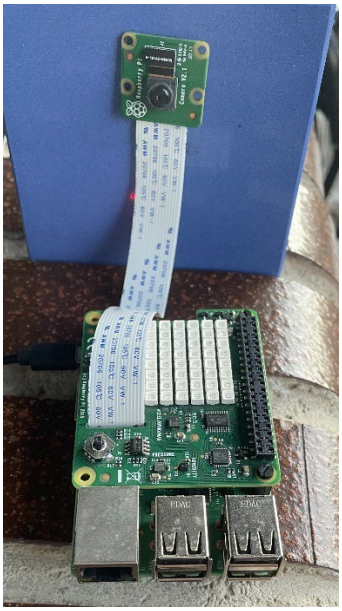


Figura 5. Montaje final del prototipo.

3.1. Raspberry Pi

Se trata de un ordenador monoplaca (*single board computer,* SBC), diseño compacto y bajo coste. Es muy común su uso en el desarrollo de pequeños prototipos. Fue creada con la intención de promover el interés por la informática e introducirla en los centros educativos, además, tiene una política de *hardware* y *software* libres.

Permite la instalación de diversos sistemas operativos, pero *Raspberry Pi* OS (*Operative System*), o *Raspbian*, tiene en cuenta la limitación de potencia y características especiales de estos ordenadores frente a los convencionales, ya que fue desarrollado específicamente para estos dispositivos. Está basado en una distribución de Linux (Debian) y ofrece dos versiones: *Raspberry Pi OS Pixel* y *Raspberry Pi OS Lite*. En este caso se ha optado por la primera opción dado que cuenta con una interfaz gráfica de usuario, lo que supone un manejo más sencillo [5].

El funcionamiento se asemeja al de un ordenador convencional, ya que su composición es similar. Estas placas se basan en un SoC (*System on Chip*) de arquitectura ARM (*Acron RISC Machines*) de buen rendimiento y bajo consumo. En concreto el modelo empleado en este proyecto es la Raspberry Pi 3 B, cuya estructura y componentes se muestran en la *Figura 6*.

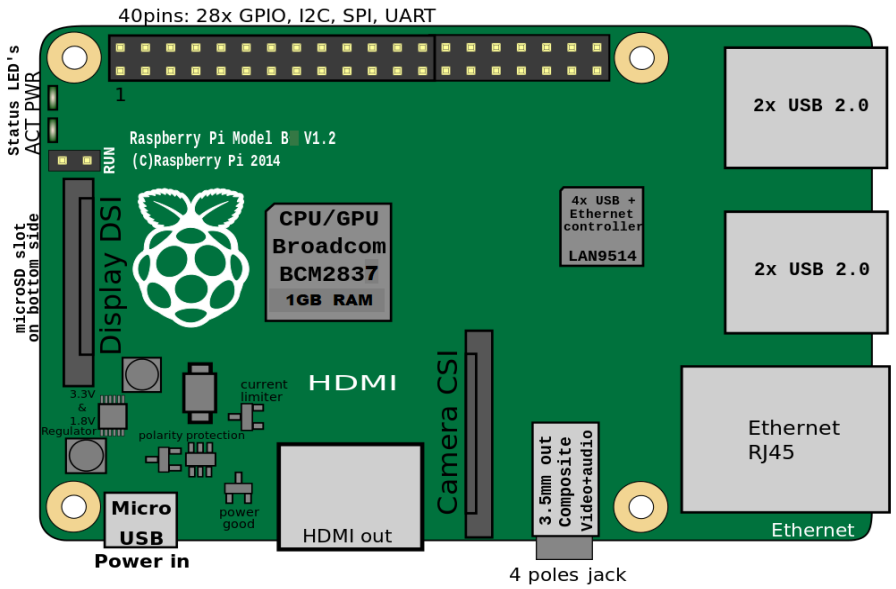


Figura 6. Diagrama Raspberry Pi 3 [6].

Este dispositivo es el ideal en este tipo de proyectos ya que, además de las características comentadas anteriormente relacionadas con el consumo y prestaciones, ofrece una amplia gama de periféricos compatibles con él, de entre los que se han escogido los módulos de cámara y sensor como se comentará posteriormente, al formar parte de la misma familia. Por lo tanto, esta placa conforma la unidad de control de la estación meteorológica y cumple con todos los requisitos necesarios para llevar a cabo el desarrollo de este proyecto [7]:

- Conexión inalámbrica a red.
- Puerto de conexión para módulo de cámara.
- Puerto de conexión para módulo de sensor y *display*.
- Bajo consumo.
- Tamaño reducido.

3.2. Módulo Camera Pi

Tal y como se ha comentado anteriormente en este documento, una de las funcionalidades de esta estación es la posibilidad de visualizar el estado de, por ejemplo, un terreno o el cielo, a lo largo de un día gracias a la creación de un video formado por fotogramas tomados a lo largo de esas veinticuatro horas. Éste es el módulo encargado de capturar dichas imágenes, que una vez procesadas formarán el vídeo a cámara rápida, en formato *TimeLapse*.

Para llevar a cabo esta función se ha empleado la *Raspberry Pi Camera Module 2*, que se muestra en la *Figura 7*, en la que además se puede observar el cable plano que

proporciona la conexión entre la cámara y la *Raspberry Pi* mediante conexión CSI-2. De entre sus especificaciones cabe destacar: capacidad de crear fotografías y vídeos de alta definición (HD – *High Definition*), cuenta con una resolución de 8 *Megapixels* y un sensor de imagen IMX219PQ de Sony, que proporciona imágenes de vídeo de alta sensibilidad y velocidad, además, ofrece la posibilidad de ajustar parámetros de la imagen como la exposición o el balance de blancos.

Figura 7. Raspberry Pi Camera Module 2 [6].

Este módulo no requiere el cumplimiento de exigentes requisitos, dado que su uso se restringe a la captura de imágenes con una resolución suficiente como para observar a grandes rasgos los cambios producidos en el entorno. Por ello, y dadas las especificaciones comentadas anteriormente, este dispositivo cumple con las necesidades básicas para el desarrollo de la parte correspondiente del proyecto [5].

3.3. SenseHat

La medición de las variables meteorológicas es una parte fundamental en el diseño de esta estación. En primer lugar, haciendo referencia al apartado *2.1* de esta memoria, son necesarias un mínimo de tres variables para definir el estado climático. Por lo tanto, será necesario el mismo número de sensores, encargados de realizar las mediciones de dichas variables. En segundo lugar, dadas las condiciones atmosféricas actuales y los rangos que abarcan, en condiciones normales, las variables meteorológicas, estos sensores deberán cumplir una serie de requisitos que aseguren unas mediciones fiables.

Las especificaciones de los sensores que integra el *SenseHat* son las siguientes [8]:

- Sensor de temperatura:
  - Precisión de ±2ºC en un rango de temperaturas comprendido entre 0ºC y 65ºC.
- Sensor de humedad:
  - Precisión de ±4.5% en el rango 20-80% de humedad relativa.
- Sensor de presión:

- Precisión dependiente de la temperatura y presión, bajo condiciones normales se sitúa en ±0.1hPA.
- Abarca un rango de medición de 260hPA a 1260hPA.

Dados los intervalos de medición, se comprueba la funcionalidad de este dispositivo, es decir, si dichos rangos abarcan los valores que alcanzan las variables meteorológicas en la zona de estudio.

Consideramos zona de estudio, para este prototipo de aplicación, la ciudad de Zaragoza. Los valores máximos y mínimos registrados para las tres variables (temperatura, presión y humedad relativa) en esta localización son: [43ºC, -4ºC], [1035mbar, 990mbar] y [90%, 30%] respectivamente. Como puede observarse, de la comparación entre los registros y las especificaciones del sensor, se concluye que éste cumple a grandes rasgos los requisitos necesarios para un correcto funcionamiento de la aplicación.

Por otra parte, el *SenseHat* incluye un *display* compuesto por una matriz de leds de 8x8, que permite al usuario definir el estado de cada led por separado, o funciones más complejas como la posibilidad de mostrar frases. A cada led se le asigna, dentro de la matriz, un vector de tres componentes que definen el color que se pretende representar, siguiendo el código que se muestra en la *Figura 8*.

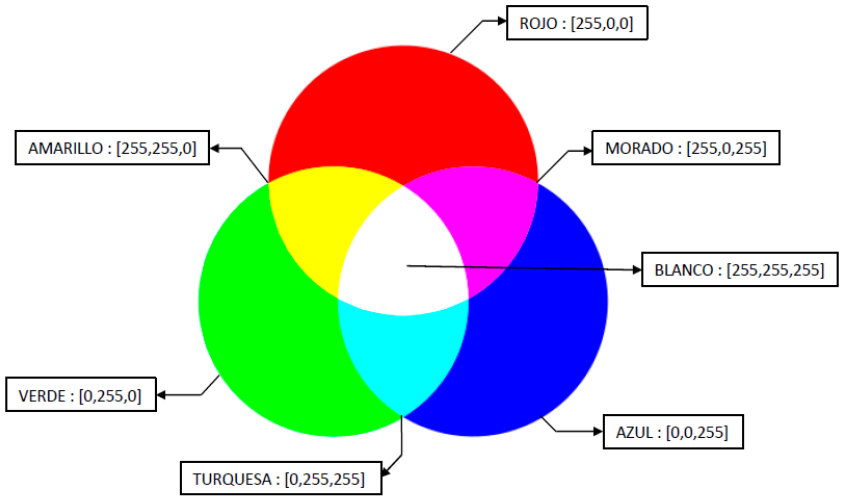


Figura 8. Código de colores.

Recordando las necesidades del proyecto, este dispositivo incluye dos de ellas: los sensores indispensables para la medición de las variables meteorológicas con una precisión suficiente como para aportar datos fiables, y la pantalla en la que se representa la temperatura y estado actual del cielo. De esta forma conseguimos un diseño compacto, al estar los sensores y la matriz de leds integrados en una misma placa, ya que otras

alternativas suponían la selección y uso de estos componentes por separado, con la complejidad de montaje que esto conlleva para conseguir un prototipo físicamente manejable y sencillo. En la *Figura 9* se muestra la vista superior de la placa del *SenseHat*, donde podemos distinguir la matriz y el puerto GPIO (*General Purpose Input/Output*) de conexión con la *Raspberry Pi* entre otros componentes.

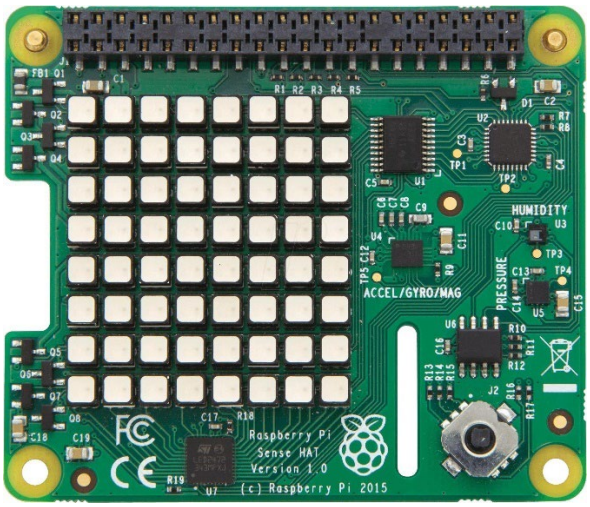


Figura 9. SenseHat [6].

4. Desarrollo del software.

4.1. Programación modular

La programación modular está basada en la técnica de diseño descendente, que consiste en dividir el problema original en diversos subproblemas que se pueden resolver por separado, para después recomponer los resultados y obtener la solución al problema. Un subproblema se denomina módulo y es una parte del problema que se puede resolver de forma independiente. [9]

Un ejemplo de programación modular se muestra en la *Figura 10*.

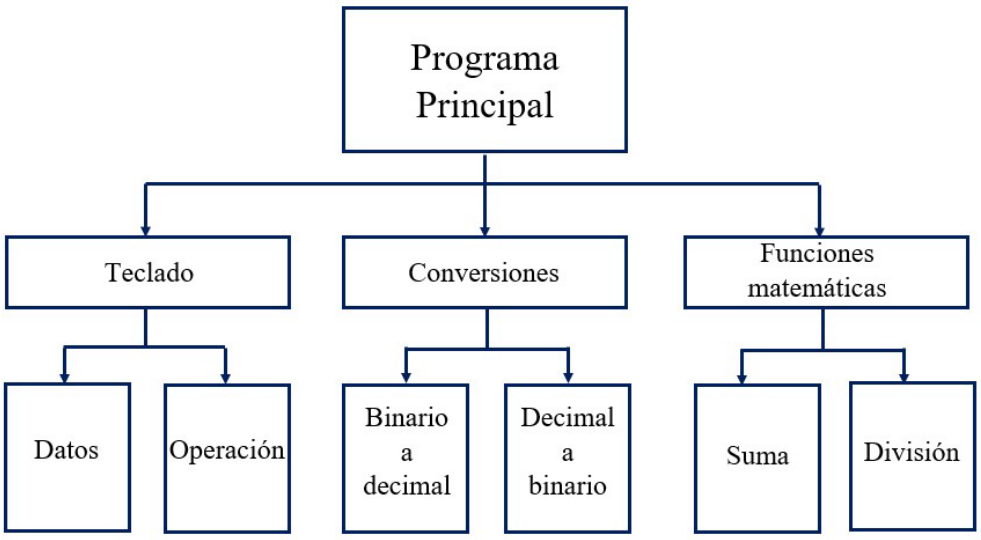


Figura 10. Estructura Programación modular.

Existen dos tipos de programación modular:

- *Top-Down*: conocida como programación descendente, se parte de un módulo genérico y complicado, generando a partir de este otros más simples y pequeños.
- *Bottom-Up*: al contrario de la anterior, se asciende hacia el módulo principal desde los más simples.

Con este tipo de programación es posible reducir la complejidad y el tamaño del problema descomponiéndolo en subproblemas más sencillos de desarrollar. Por otra parte, facilita la comprensión del programa facilitando de esta forma la cooperación entre programadores.

Una de las principales ventajas de este tipo de programación es la reutilización de código, al crearse funciones independientes del problema principal que se quiera resolver,

permite la posibilidad de reutilizar estos módulos en otros proyectos o, en un futuro integrar otros nuevos que puedan suponer una mejora. Otra ventaja a tener en cuenta es la sencillez en la depuración de errores. Esto se debe a que los errores se corrigen empezando por los módulos más simples.

4.2. Estructura del software

En este apartado se presenta la estructura modular que sigue el programa para poner en marcha las funcionalidades que ofrece esta estación meteorológica, cuyo esquema se muestra en la *Figura 11*. Conocidos los componentes que integran el hardware, la identificación de los módulos atiende a la misma clasificación, dado que cada dispositivo se dedica a una tarea específica y las subtareas que estas requieran. A continuación, se indican los procesos que debe llevar a cabo cada uno de los componentes:

- *Raspberry Pi*: Conforma la unidad de control en la que se desarrolla el código, por lo tanto, integra tanto el programa principal como los módulos a los que hace las llamadas. Corresponde al módulo principal en la estructura que estamos definiendo, al ser la encargada del control y administración de llamadas a las funciones, almacenamiento de bases de datos y comunicaciones con el servidor.

- *Camera Pi*: Encargada de generar el video, formato *TimeLapse*. Este proceso se descompone en una serie de subprocesos como la captura y almacenamiento de imágenes, procesamiento de las fotografías para la creación del vídeo y envío del archivo generado al servidor. Estas funciones componen el módulo de imagen.

- *SenseHat*: Sus tareas las definen los elementos que lo constituyen. Cuenta con dos ramas principales, la toma de medidas realizadas mediante los sensores y la representación gráfica por pantalla. Cada una de estas ramas supondrá un módulo, por un lado, se describirán las funciones de mediciones de las variables meteorológicas y el tratamiento de estas, y por otro lado las configuraciones de la matriz de leds.

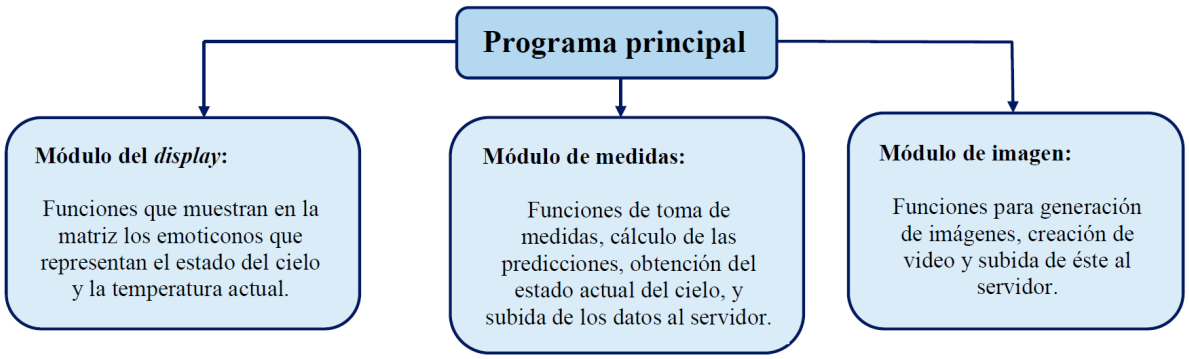


Figura 11. Estructura modular.

4.3. Módulo principal

Integra la estructura organizativa necesaria para realizar las llamadas al resto de módulos cuando se precise la ejecución de alguna de sus funciones. Como se ha mencionado anteriormente, este proyecto cuenta con tareas cíclicas, es decir, cada una de ellas tiene un periodo de tiempo tras el cual debe ponerse en marcha. En este apartado se expone el código que compone este módulo y las decisiones que se han tomado sobre él.

Por norma general, se registran medidas de las variables meteorológicas, asi como las previsiones realizadas, de forma horaria en las instituciones oficiales de meteorología. Con el fin de poder hacer uso de bases de datos externas para completar o mejorar este estudio, los registros que se crean de las mediciones tomadas por el *SenseHat* se rigen por la misma norma.

En una primera fase se estudió poder ofrecer al usuario la posibilidad de decidir cuándo deseaba grabar el *TimeLapse*, habilitando un "pulsador" en la web que proporciona la información obtenida por la estación meteorológica. Pero la herramienta empleada para alojar y mostrar esta información consta de una serie de restricciones, como se explicará en apartados posteriores. Dada la situación, se toma la decisión de programar estas grabaciones de forma diaria, es decir, el inicio y final de la grabación está impuesto y corresponde al inicio y final del día. La intención de este formato de vídeo es poder visualizar lo ocurrido en un largo periodo de tiempo en unos pocos minutos, perdiendo la mínima información posible. Por lo tanto, se estima que, en una duración de veinticuatro horas, se obtiene información suficiente si la captura de imágenes se realiza en intervalos de cinco minutos.

La última tarea por planificar implica la representación gráfica a través de la matriz disponible en el *SenseHat*. Se trata de coordinar la visualización de la temperatura y el estado actual del cielo, de tal forma que estén presentes en la pantalla durante aproximadamente la misma duración, unos treinta segundos, al mismo tiempo que el resto de las tareas siguen su curso.

En la *Figura 12* se muestra el diagrama de flujo de este módulo. Al poner en funcionamiento la *Raspberry Pi* se declaran y definen dos variables que, bien serán pasadas como parámetro a otras funciones (*DataBase)* o bien mediante la llamada a una función propia del paquete importado generan este parámetro (*sense).* Una vez definidas estas variables comienza el bucle infinito que rige el funcionamiento de la estación meteorológica. En primer lugar, se obtiene el estado actual del cielo en términos de nubosidad y precipitación, accediendo a la base de datos generada y almacenada en el directorio indicado en *DataBase,* para poder definir que emoticono debe representarse en el *SenseHat.* Asimismo, gracias a la variable *sense* medimos la temperatura ambiente actual para mostrarla por pantalla una vez haya finalizado la función anterior. Los métodos que crean y muestran los iconos están diseñados para durar aproximadamente treinta segundos ya que algunos de ellos generan iconos dinámicos, por esto se ha establecido una espera de veinte segundos tras mostrar la temperatura, porque en conjunto con el tiempo de ejecución del resto de funciones suman aproximadamente los treinta segundos que consiguen una visualización equitativa de ambos tipos de información.

Tras el inicio del programa se han declarado otro tipo de variables que sirven de referencia temporal para la toma de decisiones, permiten comprobar si desde la última medición ha transcurrido una hora (*ref_1h*) y si la última imagen se capturó hace cinco minutos (*ref_5min*). Para corroborar el cumplimiento de estos intervalos de tiempo, se almacena el instante de tiempo actual (*t*) antes de entrar en las condiciones temporales y mediante una resta entre este y la referencia correspondiente se conoce el tiempo transcurrido. Como cabe esperar una vez se ha verificado la condición, la referencia temporal debe actualizarse.

Sin embargo, la generación y subida de los vídeos a internet está condicionada por la hora actual, es decir, estos se crean y suben una vez ha finalizado el día. De tal forma que en nuestro servidor se almacenaran grabaciones diarias identificadas con la fecha de creación. Puede observarse en la *Figura 12* la última acción antes de reiniciar el bucle es la eliminación de los archivos creados en la memoria de la *Raspberry Pi*, con esto conseguimos evitar futuros problemas de colapso por almacenamiento lleno.

Cabe destacar que las importaciones necesarias para la puesta en marcha, y sin errores de código, son principalmente funciones creadas en el resto de los módulos, y librerías para la creación, definición o manejo de las variables comentadas anteriormente (*SenseHat, time, datetime, pandas)*.

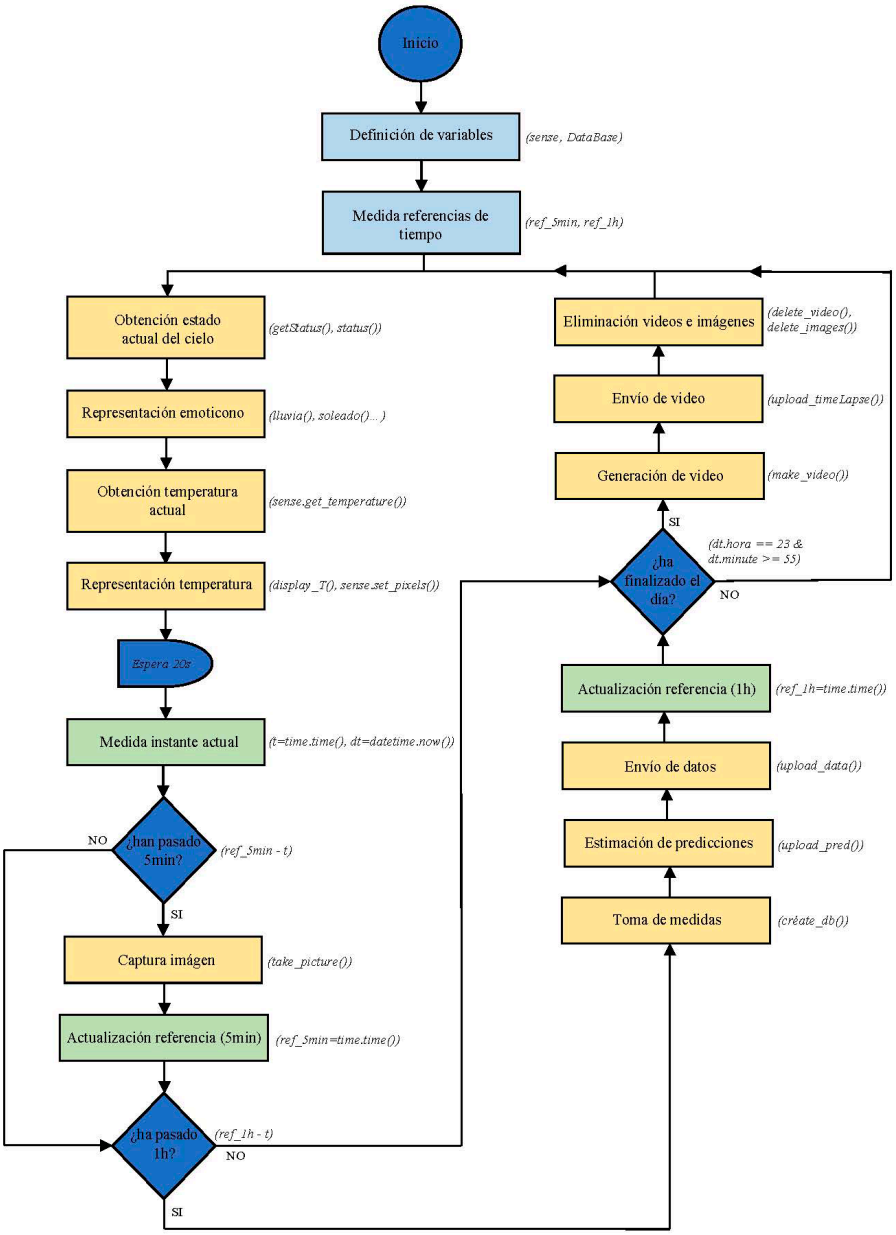


Figura 12. Diagrama de flujo módulo principal.

4.4. Módulo de imagen

El proceso de creación de un *TimeLapse* se divide en tres pasos:

- Captura y almacenamiento de imágenes (*take_picture()):*

El acceso al módulo de la cámara y sus configuraciones lo proporciona el paquete *picamera*, y la conexión con esta se realiza mediante la función *PiCamera()* que se almacena en una variable (*camera*) para mayor facilidad en la llamada a sus métodos.

Antes de tomar la foto se define la resolución deseada, en este caso se ha decidido establecer la máxima resolución permitida por el dispositivo (1920x1080) para apreciar en las imágenes los máximos detalles posibles.

El proceso de captura de la imagen es muy sencillo, basta con recurrir al método *capture*, integrado en la función *PiCamera()*, pero previamente debe indicarse el nombre con el que se quiere almacenar dicha captura en la memoria. El fin de estas imágenes es conformar un vídeo a cámara rápida, por lo tanto, interesa registrar la fecha y hora en que se ha realizado, para posteriormente reflejar esta información en el propio video. La fecha y hora del instante de la foto se obtienen gracias a las librerías *datetime* y *time* [10].

- Creación del vídeo (*create_video()*):

El procesamiento de las imágenes para conformar el vídeo se realiza gracias a la librería *OpenCV*. Es una biblioteca de visión artificial, altamente empleada en proyectos educativos y profesionales al ser de código abierto y contar con implementaciones que abarcan más de 2500 algoritmos, entre ellos el reconocimiento de objetos, reconocimiento de escenarios o clasificación de acciones humanas, por ejemplo.

Su aplicación en este proyecto se limita a la agrupación de las imágenes almacenadas en el transcurso de un día, de tal forma que el resultado de este procesamiento sea un vídeo de corta duración en el que puedan apreciarse todas las capturas durante un tiempo suficiente.

Como se ha mencionado en el apartado previo, las fotografías se almacenan con un nombre que refleja la fecha y hora de la captura, de esta forma, al finalizar el día, se genera una lista de las imágenes ordenada temporalmente gracias a estos nombres. El siguiente paso es crear un objeto de tipo *VideoWriter*, haciendo uso de la librería *OpenCV*, al que se le asignarán nombre, un código de cuatro caracteres del códec utilizado para comprimir las tramas, velocidad de fotogramas del flujo de video creado y tamaño de imagen. El nombre del video corresponde con la fecha del día en el que es creado, permitiendo al usuario reproducir la grabación del día que sea de su interés. Los valores de parámetros *fourcc* (código de cuatro caracteres) y *fps* (velocidad de fotogramas) son 0 y 3.5 respectivamente, porque estamos procesando un vídeo con imágenes en formato *.jpg* y se ha comprobado mediante pruebas de distintos valores que la velocidad óptima para apreciar todos los fotogramas y al mismo tiempo obtener un vídeo de corta duración es de 3.5 *frames* por segundo. El tamaño de la imagen debe ser el mismo que el de las capturas

almacenadas, y haciendo uso de la función *imread()* puede obtenerse esta información.

El proceso de creación del vídeo consiste en recorrer la lista de imágenes mencionada anteriormente, cada una de ellas será "leída" (*imread()*), es decir, almacenada en una variable, modificada mediante la escritura de su nombre (*putText()*) en la propia fotografía, para conocer en la reproducción que momento se está visualizando, y "escrita" (*write()*) en el objeto de video definido previamente. Todas estas funcionalidades al igual que las anteriores forman parte de la biblioteca *OpenCV*. Tras completar la escritura de todas las imágenes se genera el archivo en formato vídeo (*.avi)* y se almacena en la memoria de la *Raspberry Pi* [11].

El lugar de almacenamiento en la nube de estos archivos de vídeo ha sido uno de los principales problemas en el desarrollo de este proyecto. En un primer momento se trató de implementar un mecanismo que permitiera al usuario decidir en qué momentos quería iniciar y detener la grabación, pero la plataforma empleada para el soporte de la aplicación no ofrece esta funcionalidad. Esta plataforma dispone de varios planes, uno gratuito dedicado al sector de la educación y dos versiones de pago con más posibilidades. Para el desarrollo de este proyecto se opta por la versión gratuita, en la que las opciones de las que se dispone para visualizar vídeos son dos: retransmisión de un vídeo perteneciente a la plataforma de *Youtube* directamente en la página que almacena el resto de los datos, o mediante un enlace acceder a una unidad de almacenamiento web en la que guardar los archivos generados.

La primera opción presentaba bastantes problemas, ya que al incluir la reproducción de un video directamente en la página web, únicamente podía visualizarse el vídeo asociado a ese enlace, el resto de *timeLapses* estarían almacenados en el perfil de *Youtube* y para acceder a ellos sería necesario abrir esta aplicación y dirigirse al perfil dedicado para ello. Además, la subida de las grabaciones a *Youtube* se realiza mediante su propia API, y esta presenta restricciones a la hora de enviar archivos de acceso público, es decir, vídeos disponibles para cualquier usuario y no solo para el propietario, ya que es *Youtube* quien estudia la aplicación que se quiere poner en contacto con su servidor y decide si la capacita o no para subir archivos públicos. Para obtener este derecho es necesario rellenar un formulario y esperar el tiempo que la organización estime pertinente para solicitar más información acerca de la aplicación y decidir si se acepta o no la petición [12].

El almacenamiento de las grabaciones en una unidad web, por el contrario, no requería de una solicitud formal y posterior aceptación de la

misma que lo permitiera. Además, de esta forma se proporciona al usuario un enlace en la web de la estación meteorológica en el que encontrará los *timeLapse* generados desde el inicio del dispositivo. La comunicación entre *Raspberry Pi* y servidor en cuanto a implementación de código suponía una complejidad similar en ambas alternativas, por lo tanto, dadas las ventajas e inconvenientes de las dos, se optó por esta última.

En el apartado 5.2 se explica en profundidad el código desarrollado para llevar a cabo esta funcionalidad mediante la API de Google Drive.

4.5. Módulo de medidas

Este es el módulo más importante en el funcionamiento de la estación meteorológica, ya que incluye desde las medidas tomadas por los sensores del *SenseHat* hasta las predicciones del día siguiente y la obtención del estado actual del cielo. Cabe esperar que se trata del más complejo, por lo que se va a subdividir en apartados para una mejor comprensión de todas las funcionalidades que integra.

4.5.1. Medidas en tiempo real

Las tres primeras variables de interés en este proyecto son las variables meteorológicas comentadas en el apartado *2.1*, estas son medidas en tiempo real gracias a los sensores integrados en el *SenseHat* y los valores obtenidos se registran en una base de datos en formato CSV para su posterior consulta.

El paquete importado para el manejo del *SenseHat* integra las funciones necesarias para recopilar la información recogida por los sensores (*get_temperature(), get_pressure(), get_humidity()*) [10], que nos permiten registrar los valores actuales de temperatura, presión y humedad en una base de datos, habiendo acotado antes los decimales de estas variables a dos dígitos. La base de datos creada debe reflejar también el instante en el que se realizan las medidas, dado que se trata de información útil y necesaria en la creación de los modelos de predicción, como se explicará posteriormente. El formato de esta variable temporal debe ser de tipo entero, de esta forma los modelos de predicción lo entenderán como una variable que define el orden de la toma de medidas. La conversión del formato obtenido con la librería *datetime* (YYYY-MM-DD HH:mm:SS) a un formato de tipo YYYYMMDDHHmmSS se lleva a cabo mediante una función llamada *get_current_date()* que almacena los valores de año, mes, día, hora, minutos, y segundos, y los concatena para llegar a ese resultado.

En las pruebas de funcionamiento del *SenseHat* se comprobó que dada la cercanía entre *Raspberry Pi* y *SenseHat*, las medidas recogidas por el sensor de temperatura se ven afectadas por la temperatura que alcanza la CPU (*Central Processing Unit*) que puede ser de hasta 70ºC cuando lleva en funcionamiento un largo periodo de tiempo. Para

compensar este efecto se ha creado una tabla en la que ver reflejados los valores de las siguientes variables: temperatura de la CPU, temperatura medida por el sensor y temperatura real registrada por una estación meteorológica certificada. De esta relación se concluye que agrupando la temperatura de la CPU en pequeños intervalos que definen la cantidad a restar, en ºC a la temperatura medida, puede llegarse a compensar en gran parte su efecto, pero no por completo ya que el grado de ventilación al que esté expuesto el dispositivo agravará o disminuirá este fenómeno, es decir, si la localización de la *Raspberry Pi* es tal que cuenta con un flujo de aire a temperatura ambiente continuo, puede evitar la necesidad de llevar a cabo esta compensación. En el *Anexo 1* se incluye la función implementada para esta tarea, donde pueden verse los intervalos definidos y las modificaciones que estos conllevan en la temperatura medida.

Con cada llamada a esta función (*créate_db()*) se obtienen los valores de temperatura, humedad y presión, y se añaden a una nueva fila en la base de datos junto con la variable temporal formateada.

4.5.2. Predicciones

La predicción meteorológica es un tema muy complejo, nosotros en este trabajo hemos realizado una aproximación sencilla, que proporcione un modelo de predicción básico que pueda funcionar sobre una tarjeta *Raspberry Pi* con una eficacia razonable y que sea aplicable para predicciones locales. No es objetivo de este trabajo realizar un estudio serio y profundo, el cual se podría realizar en el marco de un trabajo fin de máster o una tesis doctoral.

Se han generado una serie de modelos de predicción meteorológicos cuyo objetivo es proporcionar valores, lo suficientemente ajustados a la realidad, de la precipitación (en mm/h), nubosidad (en tanto por ciento), y temperaturas extremas (máxima y mínima en ºC) previstos en 24 horas. La herramienta empleada para la construcción de estos modelos ha sido la biblioteca de software libre y dedicada al *machine leraning* (aprendizaje automático) *scikit-learn*. Esta es una ciencia basada en programar ordenadores de tal forma que sean capaces de aprender de los datos sin la necesidad de programar ese aprendizaje explícitamente. Su uso es especialmente útil en problemas que suponen una gran carga computacional, no existe una única solución aceptable o estudian ambientes en constante evolución. Una de las principales clasificaciones que se pueden aplicar a estos aprendizajes es en base a el grado de supervisión humana que requieren, en este caso se van a estudiar algoritmos supervisados ya que los datos de entrenamiento que se proporcionan conllevan un etiquetado que los identifica. [13]. Se trata de una librería que integra algoritmos de clasificación, regresión, reducción de la dimensionalidad y *clustering* (agrupación) especialmente desarrollados para su aplicación en el análisis predictivo de datos.

En la *Figura 13* pueden observarse los cuatro grupos de estimadores con los que cuenta esta librería y muestra cual se debe escoger en función de las necesidades y propiedades de nuestro proyecto. El funcionamiento general de estos estimadores se basa en el entrenamiento del algoritmo usando una base de datos que queda dividida en dos grupos, datos dedicados al entrenamiento del algoritmo y datos dedicados al testeo de este, es decir, con los que se comprueba el porcentaje de acierto del modelo generado. En el caso que nos ocupa, dado que no ha sido posible generar una base de datos lo suficientemente amplia, debido al tiempo disponible, como para contar con una cantidad que permita obtener un porcentaje de acierto aceptable, el entrenamiento de los modelos se ha llevado a cabo empleando una base de datos proporcionada por un servicio exterior.

Esta base de datos externa ha sido recogida del servidor *WorldWeatherOnline,* que integra una API (*Application Programming* Interface) dedicada a recabar datos meteorológicos por regiones con la intención de ofrecerlos como producto a empresas o desarrolladores particulares [14]. El número de datos recabados asciende a 8784 muestras, recogidas desde mayo de 2021 hasta mayo de 2022, incluyendo cada una de ellas valores de temperatura, presión, humedad relativa, precipitación, nubosidad y temperaturas extremas en la ciudad de Zaragoza, además del orden en que se realizaron las mediciones. Con la intención de predecir las condiciones meteorológicas que se esperan en un plazo de 24 horas, los datos de temperatura, humedad y presión (valores de los que disponemos gracias a los sensores) recabados tienen como fecha inicial el 1 de Mayo, y los de precipitación, humedad y temperaturas extremas (valores a estimar) el 2 de Mayo, de esta forma en una misma fila disponemos de la información obtenida sobre el estado del cielo y la temperatura 24 horas después de haber realizado las mediciones. Por lo tanto, dado que, por un lado, se cuenta muchas más de las cincuenta muestras establecidas como umbral en la *Figura 13*, y tal y como se ha referido en párrafos anteriores se pretende hacer predicciones numéricas, si seguimos los pasos indicados en la *Figura 13* se concluye que son necesarios algoritmos de regresión [15].

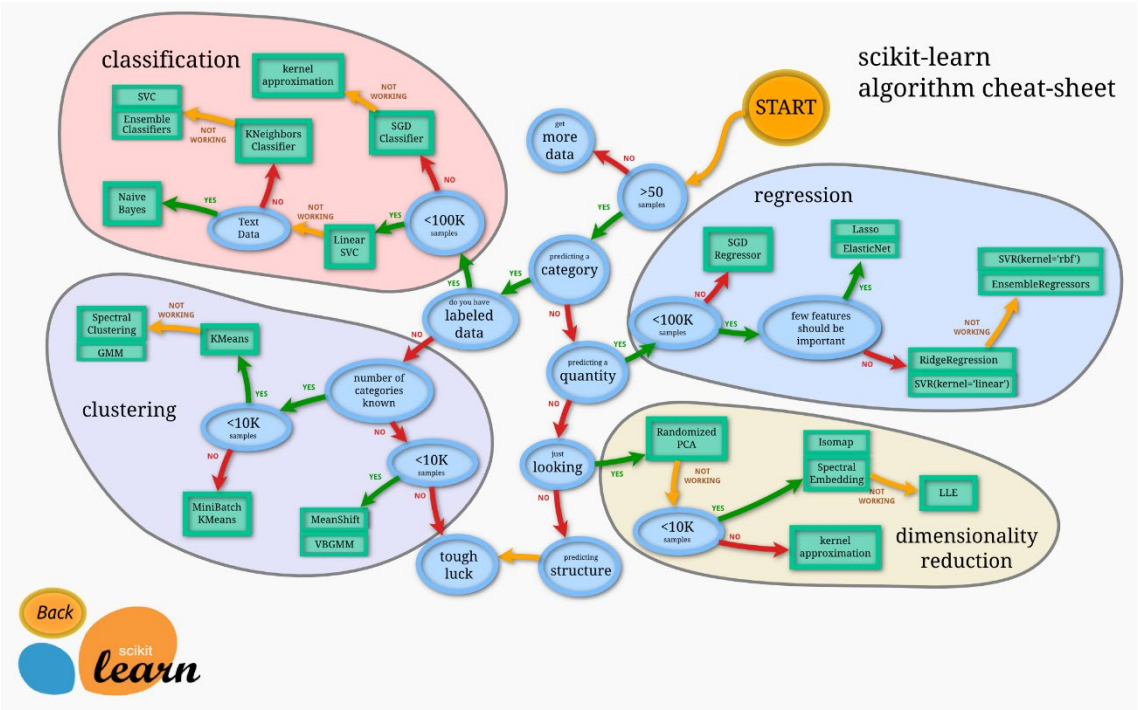


Figura 13. Diagrama de elección del correcto estimador [15].

A continuación, se realiza un breve análisis de los grupos de regresores disponibles para escoger el más adecuado para este proyecto. El lector interesado en más detalles puede consultar la referencia [13].

- Modelos lineales (*Lasso y ElasticNet*)

En este tipo de modelos se espera que el valor a predecir sea el resultado de una combinación lineal de los parámetros conocidos. En términos matemáticos, si ŷ es el valor por predecir:

$$\hat{y}(\omega, x) = \omega_0 + \omega_1 x_1 + \cdots + \omega_p x_p$$

- *Ensemble methods* (métodos de ensamblaje)

Se basan en la combinación de los algoritmos de predicción de varios estimadores con el objetivo de mejorar la robustez obtenida con respecto a un solo estimador. Se distinguen dos familias:

- En los métodos que trabajan con promedios, se construyen varios estimadores independientes y se promedian sus predicciones. De esta forma el estimador resultante suele ser mejor que los que lo constituyen dado que la varianza se reduce.

- Por el contrario, en métodos de refuerzo, los estimadores base se construyen de forma secuencial, tratando de reducir el sesgo del estimador final. El objetivo es combinar modelos débiles y producir uno potente.

- *Support Vector Machines* (máquinas de vectores de apoyo)

Son algoritmos especialmente útiles para la detección, dentro de los modelos de estudio, de valores atípicos.

- *Ridge Regression* (regresión de crestas)

Para entender este tipo de regresión hay que recordar brevemente el método de mínimos cuadrados: se trata de minimizar la suma de los cuadrados de las diferencias entre las predicciones del modelo y los valores reales a ajustar.

La regresión de crestas (ridge) se emplea en casos en los que el método anterior no es apto por la multicolinealidad (fuerte correlación entre variables predictoras). Consiste en añadir a los mínimos cuadrados una penalización por contracción, con la que se pretende minimizar la presencia de las variables predictoras menos influyentes en el modelo [16].

- *Stochastic Gradient Descent* (SGD, descenso de gradiente estocástico)

En este grupo se incluyen los algoritmos basados en el entrenamiento de redes neuronales, actualizan los pesos de la red de tal forma que se minimice el cuadrado de la diferencia entre la salida real y la calculada.

En la *Figura 14* se muestra un esquema de las redes neuronales multicapa que se analizan en estos algoritmos. Las variables representadas en ella corresponden a entradas al sistema (x), neuronas ocultas (h), pesos (w) y salidas calculadas (o). Cada capa oculta representa una unidad perceptrón, que se basa en el cálculo de una combinación lineal de las entradas (valor *net*) que posteriormente se pasa como parámetro a una función de activación y es el resultado de esta función el que define la salida. A continuación, se plasman las fórmulas que rigen este proceso para una comprensión más sencilla:

Función de entrada (*net*)

$\sum_{i=0}^{n} \omega_i x_i$	
Función de activación (σ)	
Linear	$\sigma\,(net) = net$
Sigma (logística)	$\sigma\,(net) = \frac{1}{1+e^{-net}}$
Tangente hiperbólica	$\sigma\,(net) = \frac{e^{2net}-1}{e^{2net}+1}$
Salida	
$o = \sigma\,(net)$	

Tabla 1. Fórmulas que incluyen las capas ocultas.

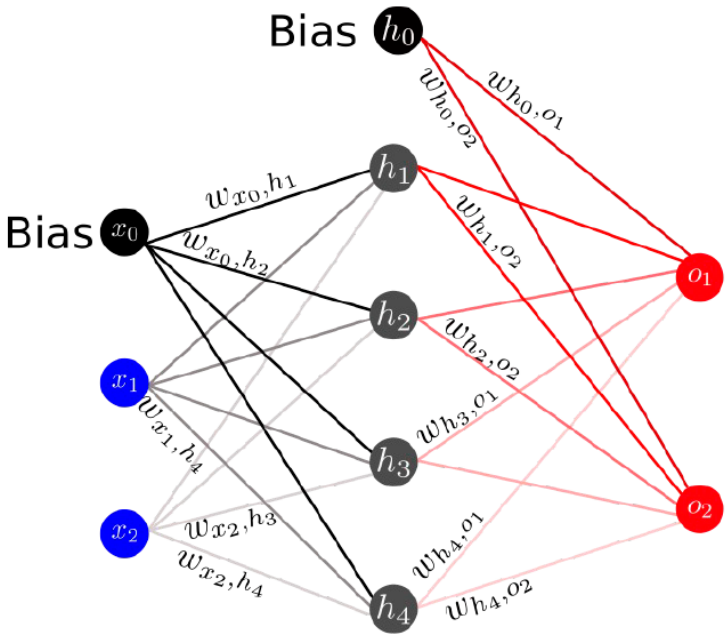


Figura 14. Esquema de red neuronal multicapa [17].

El procedimiento matemático que siguen este tipo de algoritmos comienza por definir el error:

$$E(\vec{w}) = \frac{1}{2} \sum_{s \in D} \sum_{k \in outputs} (t_{ks} - o_{ks})^2)$$

Tamaño de la muestra de entrenamiento

Numero de unidades de salida

Ecuación 1. Error en redes neuronales multicapa [17].

En la *Ecuación 1* se indica el significado de alguno de los términos presentes, pero no todos, previamente se ha comentado que las salidas calculadas se representan con la letra *o* y aquí puede apreciarse la diferencia entre salidas que se pretende minimizar ya que la salida objetivo corresponde a la letra *t.* Dado el nombre de estos regresores cabe esperar que el término que se pretende minimizar sea el gradiente de este error (*Ecuación 2*), y así es. El resto de las ecuaciones que completan este problema iterativo se incluyen, en orden, a continuación. El caso del SGD (*Stochastic Gradient Descent*) toma una única muestra por cada iteración realizada.

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n} \right]$$

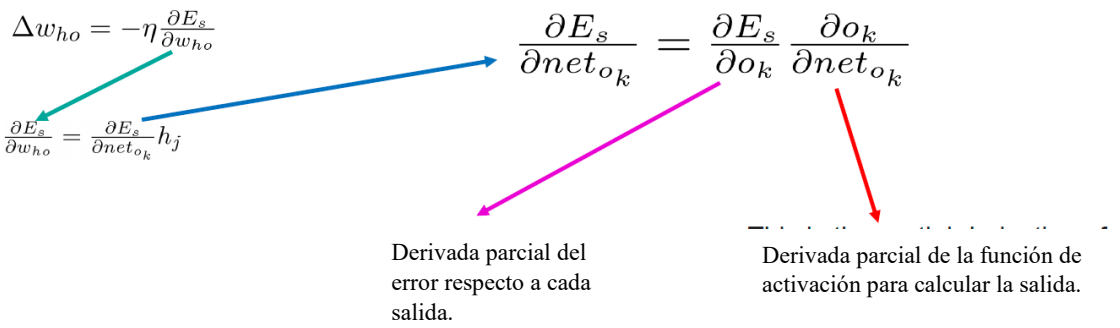
Ecuación 2. Gradiente del error en redes neuronales multicapa [17].

La regla de entrenamiento a aplicar contempla dos casos: cálculo de la variación de los pesos comprendidos entre la oculta y la de salida (*o*) y cálculo entre la capa de entrada (*i*) y la oculta (*h*) (en este orden, lo que se denomina como *backpropagation*). Podemos ver las respectivas expresiones matemáticas en la *Ecuación 3*.

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

$$\Delta w_{ih} = -\eta \frac{\partial E_s}{\partial w_{ih}} \qquad \Delta w_{ho} = -\eta \frac{\partial E_s}{\partial w_{ho}}$$

Ecuación 3. Variación de los vectores de peso [17].



Ecuación 4. Desarrollo de la derivada parcial del error respecto los pesos [17].

La función de activación seleccionada para resolver el problema debe sustituirse correctamente en la *Ecuación 4* para continuar con los cálculos hasta obtener los valores de las variaciones de pesos y añadirlas a los valores iniciales, consiguiendo un proceso cíclico. En esta ecuación y anteriores aparece un nuevo término η que representa el *learning rate* (ritmo de aprendizaje) y será uno de los parámetros a modificar en nuestro modelo, podría decirse que constituye la velocidad a la que aprende el algoritmo [17].

Las variables de predicción no presentan una relación lineal con las variables explicativas, por lo tanto, los modelos de regresión lineal no se ajustan a este modelo, así como los *SVR* (*Support Vector Machine Regression*) que están más orientados a la detección de valores atípicos. Dado que tampoco existen fuertes relaciones de linealidad entre ambos tipos de variable, por ejemplo, el fenómeno de la lluvia no depende únicamente de la temperatura o humedad en el ambiente, sino que influyen en él muchos otros aspectos que no se contemplan en este caso, los modelos de aprendizaje automático más adecuados predicción meteorológica son los métodos de *ensambles* y los perceptrones multicapa o MLP (*feedforward neural networks*) entrenados iterativamente con SGD, que permiten capturar dependencias no lineales.

Comenzando con el proceso de creación de estos modelos, en primer lugar, se distingue entre las variables conocidas (temperatura, humedad, presión y orden de medida), cuyos valores obtenidos de la base de datos se almacenan en una lista de nombre *X_pred,* y las que se pretende predecir (precipitación, nubosidad y temperaturas extremas), creando una lista propia para cada una de ellas (*Y_TMax_pred, Y_TMin_pred, Y_Nub_pred, Y_Prec_pred*), ya que las necesidades de los modelos predictivos pueden variar en función de la variable a predecir, como se explicará posteriormente. Tras esta clasificación, se divide el conjunto de datos dedicando un 70% de estos al proceso de entrenamiento, y un 30% al proceso de testeo, gracias a la función *train_test_split()* integrada en el paquete *sklearn*. Esta función genera cuatro nuevas listas, dos de ellas corresponden al resultado de la división de *X_pred* en *X_train_pred* (contiene el 70% de los valores almacenados en *X_pred*) y *X_test_pred* (contiene el 30% de los valores

restantes), y las otras dos se obtienen de la misma división porcentual aplicada a la lista *Y_TMax_pred* (dividida en *Y_train_TMax_pred y Y_test_TMax_pred*). Este proceso se realiza de forma aleatoria y debe repetirse tantas veces como variables se quieran predecir, es decir, cada modelo debe contar con un conjunto de datos compuesto por valores conocidos y sus "respuestas" que empleará para entrenar el algoritmo, y otro conjunto con las mismas características para comprobar el funcionamiento de este.

Previamente a la creación del modelo deben preprocesarse los datos mediante estandarización (para cada variable, restar su media y dividir por la desviación estándar), necesaria ya que ciertos estimadores podrían tener un mal comportamiento si las características individuales no son similares a datos estándar distribuidos normalmente, como por ejemplo gaussianos con media cero y varianza unitaria.

Los dos grupos de regresores seleccionados incluyen una gran variedad de algoritmos, y habiéndose hecho pruebas de funcionamiento con la mayoría de ellos, se concreta que son necesarios dos tipos de regresores para la predicción de las cuatro variables, obteniendo los mejores resultados de la aplicación de *MLPRegressor* (de la familia de los algoritmos de redes neuronales entrenados con SGD) en la predicción de las temperaturas, y *GradientBoostingRegressor* (de la familia de los regresores basados en *ensambles*) en la predicción de nubosidad y temperatura.

En un primer entrenamiento de estos modelos se obtuvo un porcentaje de acierto en los cuatro casos de alrededor del 70%, siendo considerado un buen porcentaje ya que, como se ha comentado en apartados anteriores, la ciencia de la predicción meteorológica presenta tantas variables dependientes que supone un reto alcanzar porcentajes próximos al 100% (en realidad se trata de un sistema caótico en el sentido físico, por lo que no es posible físicamente llegar a un 100%). Los algoritmos de predicción incluidos en esta biblioteca cuentan con una serie de hiperparametros ajustables para mejorar su funcionamiento. En el caso del *MLPRegressor* son tres los hiperparametros más influyentes: función de activación por defecto ('*rectified linear unit function'*), máximo número de iteraciones (por defecto 200) y ritmo de aprendizaje (por defecto 0.001), y se ha probado que un mayor número de iteraciones (1500) produce una notable mejora en el porcentaje de acierto, pero a costa del mayor riesgo de sobreajuste (*overfitting*), sin embargo, los valores por defecto que aplica a los otros dos parámetros ya proporcionan una respuesta óptima.

GradientBoostingRegressor construye un modelo aditivo que en cada etapa ajusta un árbol de regresión sobre el gradiente negativo de la función de pérdida. Entendido el funcionamiento de este grupo de regresores, los hiperparametros modificados han sido: ratio de aprendizaje (por defecto 0.1), numero de estimadores (por defecto 100) y profundidad máxima alcanzada por los estimadores individuales (por defecto 3). Es esperable que a mayor número de estimadores mejor se ajuste la respuesta a los valores

deseados, y así es, pero se ha observado que en la predicción de la nubosidad un numero de estimadores superior a 150 no producía cambios considerables, por el contrario, la predicción de la lluvia encuentra su máximo en 400 estimadores. El término de la ratio de aprendizaje no tiene una lógica unidireccional, es decir, a mayor valor no es peor el resultado en todos los casos, sino que depende del escenario que se esté estudiando, esto queda reflejado en los valores que proporcionan una mejor respuesta en la predicción de nubosidad y precipitación, 1.0 y 0.1 respectivamente. Por último, se ha aumentado a cuatro la profundidad máxima únicamente en el modelo de la nubosidad, al contemplarse un efecto contrario en el otro modelo.

La elección de estos términos ha seguido un criterio basado en el conocimiento de la lógica que rige estos algoritmos, y el resultado de estos cambios ha resultado en los siguientes porcentajes de acierto: 81.5% (temperatura máxima), 82.9% (temperatura mínima), 83.7% (nubosidad), 68.2% (precipitación).

Una vez generados los modelos es necesario entrenarlos con la función *fit* y guardarlos en formato ". *pkl"* gracias a la herramienta *joblib*, esto nos permite importarlos en el archivo en el que se realizan las predicciones, siendo este el último paso. Las predicciones se realizan pasando como muestra las últimas mediciones de temperatura, humedad y presión, obteniendo de los modelos entrenados los valores de precipitación, nubosidad y temperaturas extremas calculados por estos, este proceso se lleva a cabo en la función *upload_pred()*.

4.5.3. Envío de datos al servidor

De las funciones *upload_pred()* y *créate_db()* se obtienen los valores de temperatura, humedad, presión, precipitación (prevista), nubosidad (prevista) y temperaturas extremas (previstas), y estos son los parámetros que gracias al paquete *request* importado para Python enviamos mediante url a nuestro servidor, del que se dará más información en el apartado 5.1.

4.6. Módulo del display

Las tareas que se llevan a cabo en este módulo incluyen la representación de la temperatura y estado actual del cielo de forma visual (icónica) e intuitiva, los valores numéricos concretos solo se visualizan desde un teléfono móvil, tableta u ordenador. La pantalla en la que se va a visualizar esta información es la matriz 8x8 de leds integrada en el *SenseHat*, por ello, para la primera tarea se deben definir las matrices que representan cada número del cero al nueve en submatrices de 8x4. El uso de estas implica una reducción de código, en lugar de crear una matriz para cada posible valor de la temperatura, se generan dos submatrices, una para las unidades y otra para las decenas, que posteriormente se combinan, mediante llamadas a la función *extend* en un bucle que recorre la matriz, para mostrar el número completo por *display*.

La segunda tarea requiere el diseño de los emoticonos que representan los posibles estados del cielo. Los valores de precipitación y nubosidad definen estos estados, tal y como se indica en la *Figura 15*, obteniendo las siguientes posibilidades, al no ser posible distinguir la intensidad de la lluvia gráficamente: nublado, sol y nubes, sol y nubes con lluvia, soleado y con lluvia.

Precipitación (mm/h)	Estado	Nubosidad (%)	Estado
0	No llueve	[0, 25]	Soleado
(0.1, 2.0]	Lluvia débil	(25, 50]	Sol y nubes
(2.0, 15.0]	Lluvia moderada	(50, ∞)	Nublado
(15.0, 30.0]	Lluvia fuerte		
(30.0, 60.0]	Lluvia muy fuerte		
(60.0, ∞)	Lluvia torrencial		

Figura 15. Definición del estado del cielo [18].

La obtención de estos parámetros se recoge de la base de datos en la que se almacenan las predicciones, teniendo en cuenta que, el último valor almacenado indica la precipitación estimada pasadas veinticuatro horas desde la última medición.

Con la intención de facilitar la comprensión gráfica de estos diseños, algunos de ellos son dinámicos, como muestra se indica la evolución que sigue el caso de sol y nubes con lluvia en la *Figura 16*, el bucle representado en dicha imagen es repetido diez veces antes de salir de la función para cumplir con el objetivo de mostrar alternativamente este emoticono y temperatura cada uno durante aproximadamente treinta segundos.

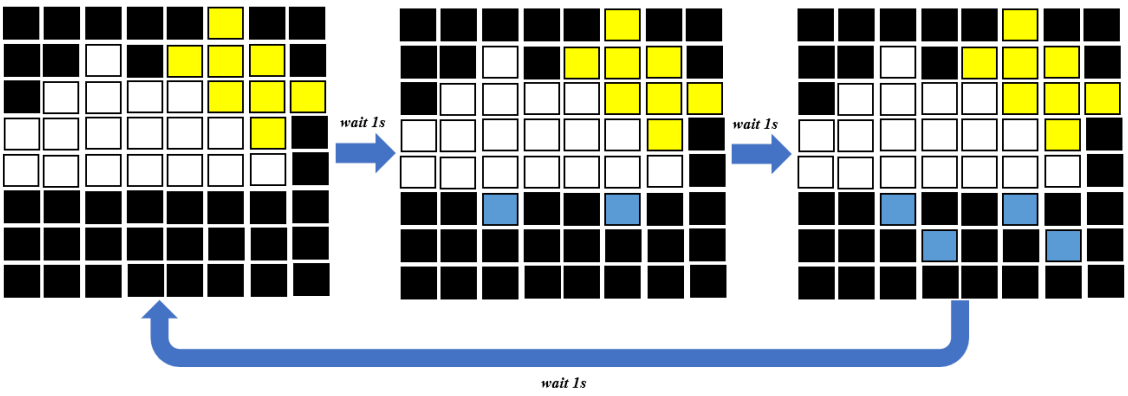


Figura 16. Evolución del emoticono que representa sol y nubes con lluvia.

En la *Figura 17 se muestran los diseños realizados para representar los cinco estados que se han definido previamente, siendo tres de ellos dinámicos: sol y nubes con lluvia, soleado y lluvia.*

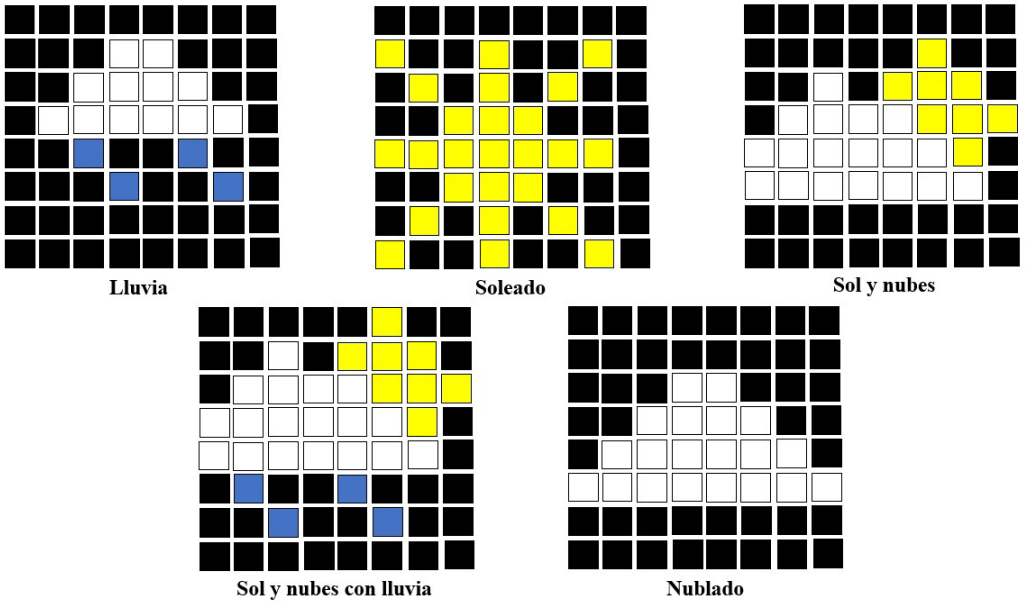


Figura 17. Iconos representados.

Para ilustrar el funcionamiento real de esta tarea se ha realizado una fotografía del dispositivo en funcionamiento en un día soleado, obteniendo la imagen que se muestra en la *Figura 18*. Dada la frecuencia de parpadeo de los leds de la matriz y la intensidad de los mismos, combinada con la resolución de la cámara, el color amarillo que debería apreciarse se convierte en tonos blancos.

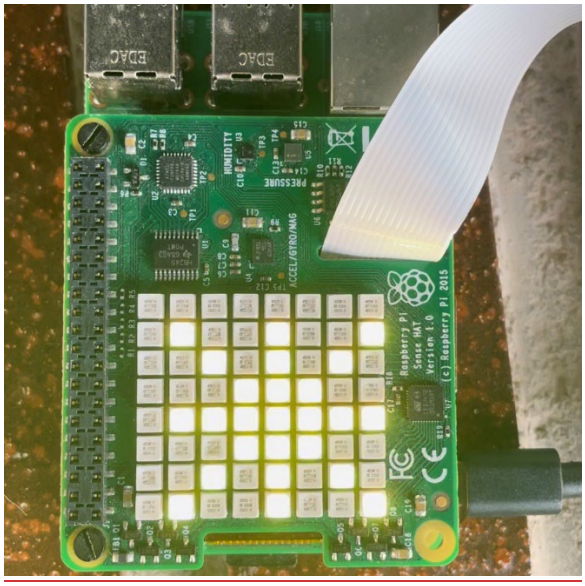


Figura 18. Captura de la representación real que define un cielo soleado.

5. Herramientas en la web.

5.1. ThingSpeak

La plataforma empleada para mostrar la información recopilada y generada en los módulos descritos en el punto *4* de este documento, es *ThingSpeak*. Se trata de un servicio de análisis de IoT (*Internet of Things*) que permite añadir, visualizar y analizar flujos de datos en directo en la nube. Proporciona visualizaciones instantáneas de los datos publicados por los dispositivos empleados. Gracias a la capacidad de ejecutar código MATLAB, se puede realizar un análisis y procesamiento en línea de los datos a medida que son recibidos. Es una herramienta común en la creación de prototipos y pruebas de concepto de sistemas IoT que requieren análisis [19].

La comunicación con este servidor sigue un mecanismo muy sencillo. Cada variable es registrada en una casilla, por ejemplo, en este proyecto la temperatura se almacena en la casilla número 1, y la plataforma asocia una llave de usuario para escribir y leer información a cada cuenta registrada. Con estas dos premisas, y gracias a la librería *requests* importada para Python, como se ha comentado anteriormente, podemos acceder al canal creado en esta plataforma para nuestro proyecto. La sintaxis del envío de datos sería la siguiente (*Figura 19*). Sintaxis de envío de datos a servidor., poniendo como ejemplo que únicamente queremos enviar un valor de temperatura a la celda 1:

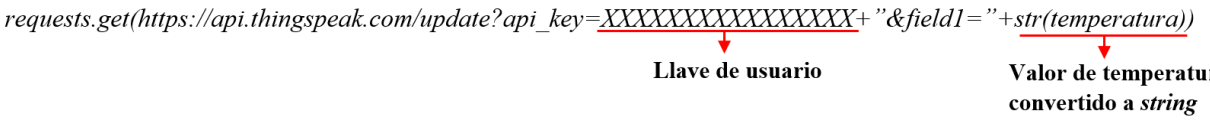


Figura 19. Sintaxis de envío de datos a servidor.

El canal creado para soportar los datos recogidos y calculados en esta estación meteorológica cuenta con siete campos (temperatura, humedad, presión, precipitación +24h, nubosidad +24h, temperatura máxima +24h y temperatura mínima +24h), siendo ocho el número máximo permitido en la versión gratuita. Seis de las casillas se representan en un gráfico lineal generado automáticamente por la plataforma, sin embargo, se ha decidido mostrar la previsión de precipitación en un gráfico de barras ya que es su representación más común e intuitiva. Adicionalmente se ha creado una ventana que indica la localización actual, siendo esta una funcionalidad que ofrece la propia plataforma.

Por último, cuenta con la opción de redirigir al usuario a otra web mediante la pulsación de un botón en la interfaz del canal, este aparece en pantalla con el nombre *"More Information"* y en él es donde se ha incluido el enlace al servidor de Google Drive donde se almacenan los videos diarios. En la *Figura 20* se muestra una vista general del

canal creado para este proyecto, y el enlace que dirige a esta es el siguiente: https://thingspeak.com/channels/1758541 .

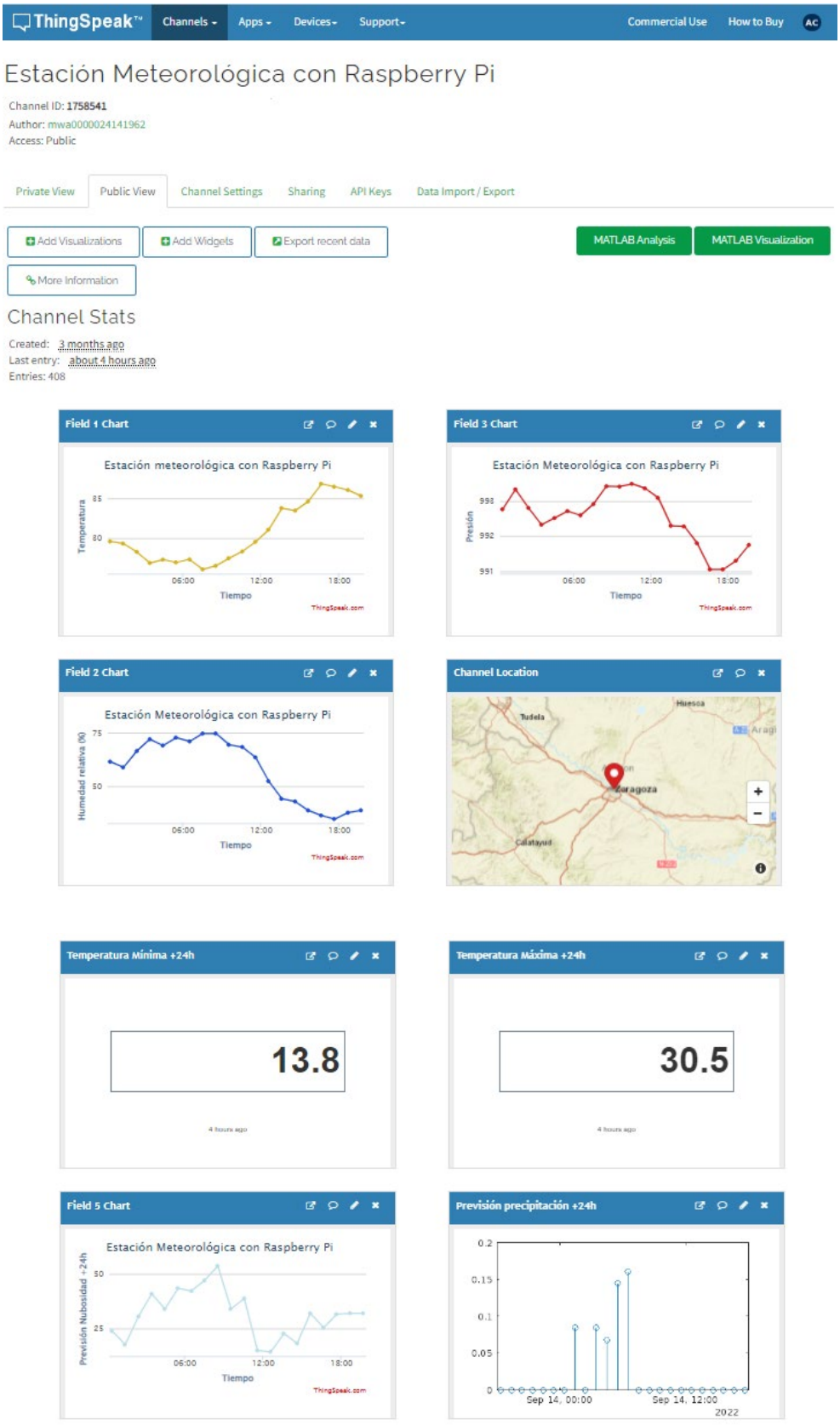


Figura 20. Canal ThingSpeak de la estación meteorológica.

5.2. Google API Drive

La herramienta que permite el envío de videos al repositorio de Google Drive es la API de Google Drive. Su aplicación en lenguaje Python es rápida y sencilla, solamente es necesario importar dos funciones (*GoogleAuth y GoogleDrive)* del paquete *pydrive* que realizan la conexión y subida de archivos. Esta primera se encarga de autenticar al usuario que está tratando de acceder al servidor, por lo que es necesario crear una cuenta con certificación de tipo *Auth2*. Los pasos previos a la creación de esta cuenta pasan por generar un proyecto asociado a esta en *GoogleCloud,* y en esta misma plataforma solicitar el certificado que proporciona un archivo de claves necesario para la comprobación de identidad, es decir, este archivo debe ser integrado en el repositorio de la *Raspberry Pi* para que la función correspondiente tenga acceso a ella a la hora de llevar a cabo la autenticación. Una vez se ha identificado al usuario en el primer envío de archivos realizado mediante la API, esta autenticación se completa automáticamente, por lo que no requiere de un inicio de sesión constante por cada envío de datos.

El siguiente paso es crear un objeto de tipo *GoogleDrive()* y mediante las funciones *CreateFile(),* cuyo parámetro es el título del video a enviar, *SetContentFile(),* al que indicamos el directorio en el que se encuentra dicho archivo, y *upload*, realizar el envío. Estas son funcionalidades propias del objeto creado, y el proceso completo descrito en este apartado se incluye en una función propia llamada *upload_timeLapse()* que solicita como parámetro el título del documento a comunicar. El contenedor final tendrá un aspecto similar al que se muestra en la *Figura 21* [20].

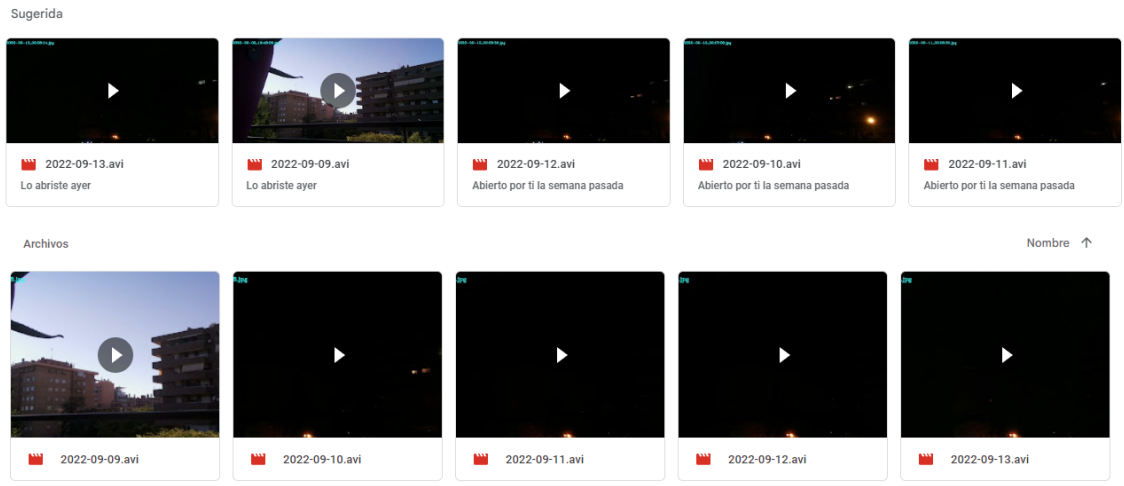


Figura 21.Aspecto unidad de Drive como almacén de TimeLapse.

6. Análisis de resultados.

En este apartado se estudia la fiabilidad de los datos proporcionados al usuario. Para ello se han comparado los valores registrados en la base de datos propia de esta estación meteorológica con los registrados por una estación fiable que aplica certificados de calidad a la información que ofrece antes de hacerla pública, como es la AEMET (Agencia Estatal de Meteorología) [21], habiendo escogido como estación de referencia la situada en el aeropuerto de Zaragoza.

En primer lugar, se estudia la precisión de los sensores que integra este dispositivo comparando valores de temperatura, humedad y presión tal y como queda reflejado en la *Figura 22*.

	Estación meteorológica con Raspberry Pi			Estación meteorológica Aeropuerto de Zaragoza (AEMET)		
Fecha y hora	Temperatura (ºC)	Humedad (%)	Presión (hPa)	Temperatura (ºC)	Humedad (%)	Presión (hPa)
17/09/2022_23:00	23,03	44,37	997	19,9	46	986,1
18/09/2022_00:00	22,59	44,99	997,29	18,7	50	986,7
18/09/2022_01:00	21,88	48,32	997,34	18,2	53	987
18/09/2022_02:00	21,53	48,42	997,35	17,2	56	987,1
18/09/2022_03:00	18,94	46,93	997,52	16,8	56	987,1
18/09/2022_04:00	17,81	47,08	997,23	16,1	58	987
18/09/2022_05:00	16,49	47,74	997,38	15,7	58	986,6
18/09/2022_06:00	16,44	51,37	997,66	15,3	60	986,9
18/09/2022_07:00	16,03	48,57	997,95	15,7	59	987,2
18/09/2022_08:00	16,11	48,69	998,09	14	71	987,6
18/09/2022_09:00	17,23	48,75	998,39	14,2	73	988
18/09/2022_10:00	19,12	43,83	998,47	15,9	61	988,3
18/09/2022_11:00	21,68	45,08	998,47	17,5	57	988,6
18/09/2022_12:00	20,85	45,03	998,36	19,5	56	988,4
18/09/2022_13:00	25,43	42,23	997,53	22	49	987,9
18/09/2022_14:00	27,82	41,52	997,32	24,1	42	987,3
18/09/2022_15:00	28,16	36,52	996,7	25,8	42	986,7
18/09/2022_16:00	29,2	35,63	995,43	26,6	46	985,8
18/09/2022_17:00	30,61	34,41	994,91	27,9	42	985
18/09/2022_18:00	30,19	35,33	994,81	28,2	41	984,6
18/09/2022_19:00	24,5	46,84	995,01	27,9	42	984,6
18/09/2022_20:00	23,3	45,37	995,37	26,7	43	984,6
18/09/2022_21:00	23,15	49,11	995,67	25	48	985,1
18/09/2022_22:00	23,18	51,26	995,97	23,6	50	985,7

Figura 22. Comparativa de datos recopilados por los sensores.

Puede observarse que la presión está desfasada 10hPa con respecto a la estación ubicada en el aeropuerto en la totalidad de los valores analizados, esto puede ser debido a una diferencia de altitud entre ambas estaciones, ya que a mayor altitud menor será la presión registrada y viceversa. Además, habrá una contribución debida a la tolerancia y exactitud propia del sensor de presión usado. Obviando esa variación y fijando la atención únicamente en las variaciones de presión en intervalos de una hora, se están registrando valores prácticamente iguales, es decir, sin tener en cuenta el aumento debido a la diferencia de altitud o exactitud del sensor, la diferencia entre columnas es del orden de décimas de unidad.

Los registros sobre la humedad reflejan una mayor variación que en el caso de la presión, obteniendo un error de menos del 10% en el 83% de los datos registrados, considerándose este como un resultado óptimo teniendo en cuenta que se desconocen las

condiciones a las que está sometida la estación proporcionada por la AEMET y que la humedad relativa puede presentar grandes variaciones en dos puntos separados en el mapa apenas unas decenas de kilómetros debido a la presencia de nubes o zonas de agua. Además, los sensores de humedad no tienen demasiada exactitud (el usado en el proyecto tiene una exactitud de ±4.5%).

Por último, los datos de temperatura son los más irregulares y los que presentan una menor precisión, a pesar de haberse obtenido variaciones de menos de 4ºC entre columnas. Cabe recordar la compensación mediante software que se ha hecho en estas mediciones debido a la temperatura de la CPU, teniendo en cuenta que en ocasiones en las que el dispositivo lleve en funcionamiento un largo periodo de tiempo, esta temperatura lo reflejará, pero si la temperatura ambiente es baja y en presencia de ventilación pueden llegar a haber compensaciones innecesarias, ya que el sensor de temperatura no se verá tan afectado por esta otra en las condiciones mencionadas. Por lo tanto, se pueden esperar errores de ±5ºC, correspondiendo dos de ellos a los referidos por el fabricante del *SenseHat*. Para una aplicación real habría que incluir en el diseño un sensor de temperatura de mayor exactitud y suficientemente alejado de la CPU.

En segundo lugar, se estudia el grado de fiabilidad de las predicciones realizadas, usando el mismo procedimiento que en el primer caso, mediante una tabla comparativa entre las mismas dos estaciones como se aprecia en la *Figura 23*.

	Estación meteorológica con Raspberry Pi				Estación meteorológica Aeropuerto de Zaragoza (AEMET)			
Fecha y hora	Precipitación (mm/h)	Nubosidad(%)	T.Mínima (ºC)	T.Máxima (ºC)	Precipitación (mm/h)	Nubosidad (%)	T.Mínima (ºC)	T.Máxima (ºC)
17/09/2022_23:00	0	19,54	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_00:00	0	16,32	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_01:00	0	23,68	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_02:00	0	22,8	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_03:00	0	20,52	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_04:00	0	18,1	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_05:00	0	19,29	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_06:00	0,023	37,56	12,8	27,6	0	40-60	13,9	28,3
18/09/2022_07:00	0,016	54,35	12,8	27,6	0	80-100	13,9	28,3
18/09/2022_08:00	0	62,29	12,8	27,6	0	80-100	13,9	28,3
18/09/2022_09:00	0,012	56,23	12,8	27,6	0	60-80	13,9	28,3
18/09/2022_10:00	0	56,59	12,8	27,6	0	60-80	13,9	28,3
18/09/2022_11:00	0	43,12	12,8	27,6	0	40-60	13,9	28,3
18/09/2022_12:00	0	24,14	12,8	27,6	0	20-40	13,9	28,3
18/09/2022_13:00	0	23,91	12,8	27,6	0	20-40	13,9	28,3
18/09/2022_14:00	0	14,35	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_15:00	0	13,89	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_16:00	0	13,89	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_17:00	0	13,89	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_18:00	0	27,22	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_19:00	0	19,5	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_20:00	0	21,73	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_21:00	0	19,29	12,8	27,6	0	0-20	13,9	28,3
18/09/2022_22:00	0	26,09	12,8	27,6	0	0-20	13,9	28,3

Figura 23. Comparativa de predicciones con valores reales.

Los datos reflejados en la tabla representada en la *Figura 23* confirman los porcentajes de acierto de los modelos generados, indicados en el apartado *4.5.2*. Las temperaturas extremas estimadas por estos modelos difieren en menos de un grado con respecto a las temperaturas máxima y mínima reales, y la nubosidad estimada entra dentro de los rangos proporcionados por la AEMET, este tipo de variable no suele proporcionarse en porcentajes exactos sino de forma cualitativa (mayormente soleado,

mayormente despejado, etc.), asignando rangos porcentuales a cada término como puede observarse. En cuanto a la precipitación se ha comprobado, tras varios días de funcionamiento que, tal y como se aprecia en la comparativa, predice con bastante exactitud los días en los que no se esperan precipitaciones, pero no ha sido posible comprobar el acierto de este modelo en días lluviosos, es decir, las condiciones meteorológicas no han permitido establecer una conclusión sobre el acierto de este modelo cuando efectivamente se esperan lluvias.

Dadas las características de este proyecto se han obtenido unos resultados bastante próximos a la realidad, teniendo en cuenta que la exactitud de los sensores empleados en estaciones oficiales es muy superior y sus métodos de predicción tienen en cuenta más parámetros, además de trabajar con algoritmos más complejos que incrementan su porcentaje de acierto.

7. Conclusiones y trabajo futuro.

En primer lugar, podemos concluir que hemos diseñado una estación meteorológica doméstica y construido un prototipo plenamente operativo, que permite medir las variables meteorológicas básicas, hacer predicciones locales con una prometedora exactitud, mostrar en un visualizador luminoso iconos del estado del cielo y observar a distancia (con móvil, tableta u ordenador) los valores de la temperatura, presión y humedad relativa actuales, su evolución en las últimas horas y su predicción. Además, la microcámara permite observar el estado del cielo, de un jardín, huerto, etc., como foto instantánea o como secuencia de video *timelapse*.

Tal y como se ha referido en varias ocasiones a lo largo de este documento, la fiabilidad de las predicciones meteorológicas depende en gran parte de la cantidad de información de la que se disponga, ya que son muchas las variables que influyen en los fenómenos atmosféricos. Las estaciones estatales de meteorología incluyen en el registro de variables meteorológicas, además de las contempladas en este proyecto, la intensidad y dirección del viento, así como un mapa general de la climatología presente en las regiones próximas, lo que les permite obtener un mayor porcentaje de acierto en previsiones del tiempo y a mayor plazo. Además de tener en cuenta estos factores, cuentan con registros generados y actualizados durante grandes periodos de tiempo, es decir, puede estudiarse la evolución de estos parámetros remontándose tantos días atrás como sea de interés.

Este prototipo presenta dos ámbitos de mejora que implican una mayor precisión tanto en las medidas en tiempo real como en las previsiones climatológicas.

El primero corresponde a los sensores, ya que como se ha comentado en el párrafo anterior, la inclusión de un anemómetro supondría una ampliación de la información tratada por la estación meteorológica y, por lo tanto, una nueva variable a considerar en los modelos de predicción, siendo posible un aumento de los porcentajes de acierto en las predicciones de nubosidad y precipitación principalmente, al influir directamente las corrientes de viento en el desplazamiento de las nubes.

Por otra parte, se ha descrito en el apartado *4.5.1* la influencia de la temperatura de la CPU de la *Raspberry Pi* en las medidas realizadas por el sensor de temperatura integrado en el *SenseHat*, y a pesar de haberse compensado esta influencia mediante software, convendría modificar el mecanismo de conexión entre estos dispositivos con el fin de alejar la fuente de calor (CPU) del entorno de medición del sensor, esto se lleva a cabo realizando la conexión mediante un cable GPIO de cuarenta pines macho a hembra. Otra posibilidad sería el uso de un sensor de temperatura de más exactitud, siempre alejado suficientemente de la CPU.

El segundo ámbito de mejora abarca el proceso de generación de los modelos de predicción que, al haber sido entrenados con una base de datos externa, los valores recopilados en esta y los obtenidos por la estación meteorológica propia van a diferir en función de las condiciones en las que se encuentren los sensores, por ejemplo, si la ubicación de estos es cercana a un río o se encuentra en una plataforma con incidencia directa de luz solar. Conviene entrenar los modelos con una base de datos generada por el mismo dispositivo que proporcionará los valores a partir de los cuales se quieren hacer las predicciones, tratando de modificar lo mínimo posible su localización. Pero para ello, se requiere que el dispositivo esté operando y capturando datos durante muchos meses, para luego entrenar los modelos de predicción con los datos locales reales capturados.

Referencias

[1] J. L. M. Zaragoza, «La Meteorología: Conceptos básicos al alcance de todos,» *3Ciencias,* 2013.

[2] Á. B. C. A. P. L. Rosa María Rodríguez Jímenez, «Meteorología y Climatología,» 2004.

[3] J. M. V. Rubio, «Origen y desarrollos actuales de la predicción meteorológica.,» p. 8.

[4] «Freemeteo,» [En línea]. Available: https://freemeteo.es/eltiempo/zaragoza/7dias/meteograma/?gid=3104324&language=spanish&country=spain. [Último acceso: 25 05 2022].

[5] R. ORG, «Raspberry Pi ORG,» [En línea]. Available: https://datasheets.raspberrypi.com/.

[6] «PNGEgg,» [En línea]. Available: https://www.pngegg.com/es. [Último acceso: 06 05 2022].

[7] «PROFESIONALReview,» [En línea]. Available: https://www.profesionalreview.com/2021/07/18/que-es-raspberry-pi/. [Último acceso: 10 05 2022].

[8] «Farnell,» [En línea]. Available: https://www.farnell.com/datasheets/1958037.pdf. [Último acceso: 15 05 2022].

[9] F. B. y. R. Ferrís, «TEMA 5: Subprogramas, programación modular,» 2005.

[10] R. ORG, «Raspberry Pi Documentation,» [En línea]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html.

[11] O. ORG, «OpenCV,» [En línea]. Available: https://opencv.org/. [Último acceso: 03 06 2022].

[12] YouTube, «Youtube Data API,» [En línea]. Available: https://developers.google.com/youtube/v3. [Último acceso: 12 06 2022].

[13] A. Gerón, Hands-On Machine Learning with Sickit Learn, Keras and TensorFlow, O'Reilly Media, 2º edición 2019.

[14] «WorldWeatherOnline,» [En línea]. Available: https://www.worldweatheronline.com/developer/. [Último acceso: 21 04 2022].

[15] «scikit-learn,» [En línea]. Available: https://scikit-learn.org/stable/.

[16] R. S. G. Eduardo Rodríguez Martínez, «Métodos cuantitativos para un modelo de regresión lineal con multicolinealidad.,» *Revista de métodos cuantitativos para la economía y la empresa,* p. 21.

[17] K. Ramirez-Amaro, «Artificial Neuronal Networks,» Göteborg, CHALMERS University, 2021, p. 57.

[18] ElTiempo, «ElTiempo,» [En línea]. Available: https://www.eltiempo.es/noticias/precipitacion-cuando-es-poco-y-cuando-es-mucho. [Último acceso: 05 07 2022].

[19] «ThingSpeak,» [En línea]. Available: https://thingspeak.com/pages/learn_more. [Último acceso: 15 08 2022].

[20] Google, «Google Drive for Developers,» [En línea]. Available: https://developers.google.com/drive. [Último acceso: 12 08 2022].

[21] A. E. d. Meteorología, «AEMET,» [En línea]. Available: https://www.aemet.es/es/eltiempo/observacion. [Último acceso: 09 18 2022].

Indice de figuras

Anexos

Anexo 1: Código desarrollado.

1. main.py

```
import time
import datetime
from sense_hat import SenseHat
import pandas as pd
from Create_DataBase import temperaturaReal, status, create_db, temperaturaReal, upload_pred, get_status, upload_data
from logos_meteo import lluvia, soleado, sol_y_nubes, sol_nubes_lluvia, nublado, display_T
from timeLapse import takePicture, make_video, delete_videos, delete_images
from drive import upload_timeLapse

# Declaración de la variable que llama al SenseHat
sense = SenseHat()

# Declaración de las referencias de tiempo en base a las que se hacen las llamadas a las funciones
ref_pic = time.time()
ref_1 = time.time()

# Declaración de la base de datos sobre la que se realizan las operaciones
DataBase = '/home/pi/Desktop/finales/db_sense.csv'

# Bucle principal
while True:

    [prec,nub] = get_status() # obtención de la precipitacion y nubosidad actuales
    status_sky = status(prec,nub) # definición del estado del cielo dadas las variables anteriores

    # Mostrar icono en SenseHat dado el estado del cielo
    if status_sky[0] == 'no_llueve':
        if status_sky[1] == 'soleado':
            soleado()
        elif status_sky[1] == 'nublado':
            nublado()
        else:
            sol_y_nubes()
    else:
        if status_sky[1] == 'nublado':
            lluvia()
        else:
            sol_nubes_lluvia()

    # Obtención de la temperatura actual y representación en el SenseHat
    temp = sense.get_temperature()
    temp = temperaturaReal(temp)
    mat_T = display_T(temp)
    sense.set_pixels(mat_T)

    time.sleep(20) # mostrar temperatura en el display durante 30s

    # Actualización medidas de tiempo
    dt = datetime.datetime.now()
    t = time.time()

    name_video = str(datetime.date.today())+'.avi'
    # Captura de imagen pasados los 5min
    if (t - ref_pic) >= 300.0: #and (t - ref_pic) < 300.0:
        takePicture()
        ref_pic = time.time()

    # Toma de medidas y actualización de la base de datos y servidor cada hora
    if (t - ref_1) >= 3600.0:#2500.0:
        temp, hum, pr = create_db()
        tMax_pred, tMin_pred, nub_pred, prec_pred = upload_pred(DataBase)
        upload_data(temp,hum,pr,tMax_pred,tMin_pred,nub_pred,prec_pred)
        ref_1 = time.time()

    # Creación y subida al servidor del TimeLapse diario
    if dt.hour == 23 and dt.minute >= 54:
        make_video()
        upload_timeLapse(name_video)
        delete_videos()
        delete_images()
```

2. Create_Database.py

```
import os
import time
import mysql.connector
from sense_hat import SenseHat
import csv
import pandas as pd
from datetime import datetime
import requests
import joblib
from date_list import get_current_date

# Función para obtener la temperatura ambiente real teniendo en cuenta la temperatura de la cpu
def temperaturaReal(tempTotal):

    sense = SenseHat()

    cpu_temp = os.popen('/usr/bin/vcgencmd measure_temp').read()

    if cpu_temp < "45.1":
        result = tempTotal-6.5
    elif cpu_temp >= "45.1" and cpu_temp < "46.2":
        result = tempTotal-8.0
    elif cpu_temp >= "46.2" and cpu_temp < "46.7":
        result = tempTotal-8.5
    elif cpu_temp >= "46.7" and cpu_temp < "48.3":
        result = tempTotal-8.7
    else:
        result = tempTotal-9.0

    return result

# Toma de medidas y actualización de la base de datos del sensor
def create_db():

    # Declaración y actualización de variables temporales
    time_ = datetime.now()
    hour = time_.hour

    # Declaración de la variable que llama al SenseHat
    sense = SenseHat()

    # Obtención de la fecha y hora actual en formato YYYYMMDDhhmmss
    date = get_current_date()

    # Obtención variables meteorológicas con precisión de dos decimales
    temp = sense.get_temperature()
    temp = temperaturaReal(temp)
    temp=round(temp,2)
    hum = sense.get_humidity()
    hum=round(hum,2)
    pr = sense.get_pressure()
    pr=round(pr,2)

    # Escritura de las variables en la base de datos
    myValues = [date,temp,hum,pr]
    with open('/home/pi/Desktop/finales/db_sense.csv','a') as f:
        csv_f = csv.writer(f)
        csv_f.writerow(myValues)

    return temp, hum, pr


# Función para predecir variables meteorológicas a 24h vista y subida al servidor
def upload_pred(file_name):

    db_df = pd.read_csv(file_name) # definición de la base de datos de trabajo

    # Carga de los modelos de predicción entrenados
    modelo_TempMax_pred = joblib.load('modelo_TempMax_pred.pkl')
    modelo_TempMin_pred = joblib.load('modelo_TempMin_pred.pkl')
    modelo_Nub_pred = joblib.load('modelo_Nub_pred.pkl')
    modelo_Prec_pred = joblib.load('modelo_Prec_pred.pkl')

    tMax_pred = modelo_TempMax_pred.predict(db_df.tail(1)) # predicción de la temperatura máxima +24h

    tMin_pred = modelo_TempMin_pred.predict(db_df.tail(1)) # predicción de la temperatura mínima +24h
```

```
    #Adaptación del formato de las predicciones para subir al servidor
    tMax_pred = str(tMax_pred)
    tMin_pred = str(tMin_pred)
    tMax_pred = tMax_pred.replace('[','').replace(']','')
    tMin_pred = tMin_pred.replace('[','').replace(']','')

    # Predicción y adaptación de la variable nubosidad +24h
    nub_pred = modelo_Nub_pred.predict(db_df.tail(1))
    if nub_pred < 0:
        nub_pred = 0
    nub_pred = str(nub_pred)
    nub_pred = nub_pred.replace("[","").replace("]","")

    # Predicción y adaptación de la variable precipitación +24h
    prec_pred = modelo_Prec_pred.predict(db_df.tail(1))
    if prec_pred < 0:
        prec_pred = 0
    prec_pred = str(prec_pred)
    prec_pred = prec_pred.replace("[","").replace("]","")

    # Almacenamiento de las predicciones en base de datos
    myPred = [prec_pred, nub_pred]
    with open('/home/pi/Desktop/finales/db_pred.csv','a') as p:
        csv_p = csv.writer(p)
        csv_p.writerow(myPred)

    return tMax_pred, tMin_pred, nub_pred, prec_pred

# Función para obtener la precipitación y nubosidad actuales
def get_status():

    # Definición base de datos de trabajo
    df_pred = pd.read_csv('/home/pi/Desktop/finales/db_pred.csv')
    predictions = pd.DataFrame(df_pred)

    # Obtención variables actuales de la base de datos
    nub = predictions.loc[len(predictions)-25]['Nubosidad%']
    prec = predictions.loc[len(predictions)-25]['PrecipitacionMM']

            status = [prec,nub]
            return status

        # Función que define el estado del cielo dada la precipitación y nubosidad actuales
        def status(prec, nub):

            stat = ['soleado','no_llueve']

            if prec > 0.1 and prec <= 2.0:
                stat_prec = 'lluvia_debil'
            elif prec > 2.0 and prec <= 15.0:
                stat_prec = 'lluvia_moderada'
            elif prec > 15.0 and prec <= 30.0:
                stat_prec = 'lluvia_fuerte'
            elif prec > 30.0 and prec <= 60.0:
                stat_prec = 'lluvia_muyFuerte'
            elif prec > 60.0:
                stat_prec = 'lluvia_torrencial'
            else:
                stat_prec = 'no_llueve'

            if nub > 25 and nub <= 50:
                stat_nub = 'sol_y_nubes'
            elif nub > 50:
                stat_nub = 'nublado'
            else:
                stat_nub = 'soleado'

            stat = [stat_prec,stat_nub]
            return stat

        #Función para subir todos los datos al servidor
        def upload_data(temp,hum,pr,tMax_pred,tMin_pred,nub_pred,prec_pred):

            thing = requests.get("https://api.thingspeak.com/update?api_key=LDE7IRL5TX7V04L1"+"&field1="+str(temp)+"&field2="+str(hum)
            +"&field3="+str(pr)+"&field4="+str(prec_pred)+"&field5="+str(nub_pred)+"&field6="+str(tMax_pred)+"&field7="+str(tMin_pred))

            thing.close()
```

3. drive.py

```
from pydrive.drive import GoogleDrive
from pydrive.auth import GoogleAuth
import os

gauth = GoogleAuth()

gauth.LocalWebserverAuth()
drive = GoogleDrive(gauth)

path = '/home/pi/Desktop/finales/'

def upload_timeLapse(title):
    video = title

    f = drive.CreateFile({'title':title})
    f.SetContentFile(os.path.join(path,title))
    f.Upload()

    f = None
```

4. Google.py

```
import pickle
import os
from google_auth_oauthlib.flow import Flow, InstalledAppFlow
from googleapiclient.discovery import build
from googleapiclient.http import MediaFileUpload, MediaIoBaseDownload
from google.auth.transport.requests import Request

def Create_Service(client_secret_file, api_name, api_version, *scopes):
    print(client_secret_file, api_name, api_version, scopes, sep='-')
    CLIENT_SECRET_FILE = client_secret_file
    API_SERVICE_NAME = api_name
    API_VERSION = api_version
    SCOPES = [scope for scope in scopes[0]]
    print(SCOPES)

    cred = None

    pickle_file = f'token_{API_SERVICE_NAME}_{API_VERSION}.pickle'

    if os.path.exists(pickle_file):
        with open(pickle_file, 'rb') as token:
            cred = pickle.load(token)

    if not cred or not cred.valid:
        if cred and cred.expired and cred.refresh_token:
            cred.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(CLIENT_SECRET_FILE, SCOPES)
            cred = flow.run_local_server()

        with open(pickle_file, 'wb') as token:
            pickle.dump(cred, token)

    try:
        service = build(API_SERVICE_NAME, API_VERSION, credentials = cred)
        print(API_SERVICE_NAME, 'service created successfully')
        return service
    except Exception as e:
        print('Unable to connect.')
        print(e)
        return None

def convert_to_RFC_datetime(year=1900, month=1, day=1, hour=0, minute=0):
    dt = datetime.datetime(year, month, day, hour, minute, 0).isoformat() + 'Z'
    return dt
```

5. logos_meteo.py

```
import os
import mysql.connector
import time
import numpy as np
from sense_hat import SenseHat

# Declaración SenseHat
sense = SenseHat()

# Definición de colores presentes en los iconos a mostrar en el SenseHat
n = [0,0,0]         #negro
b = [255,255,255]   #blanco
y = [255,255,0]     #amarillo
a = [0,0,255]       #azul

# Funciones que muestran un icono,que representa el estado actual del cielo, en el display del SenseHat
def lluvia():

    nube = [
    n,n,n,n,n,n,n,n,
    n,n,n,b,b,n,n,n,
    n,n,b,b,b,b,n,n,
    n,b,b,b,b,b,b,n,
    n,n,n,n,n,n,n,n,
    n,n,n,n,n,n,n,n,
    n,n,n,n,n,n,n,n,
    n,n,n,n,n,n,n,n
    ]
    sense.set_pixels(nube)

    for i in range(10):
        sense.set_pixel(2,4,a)
        sense.set_pixel(5,4,a)
        time.sleep(1)
        sense.set_pixel(3,5,a)
        sense.set_pixel(6,5,a)
        time.sleep(1)
        sense.set_pixel(2,6,a)
        sense.set_pixel(5,6,a)
        time.sleep(1)
        sense.set_pixel(2,4,n)
        sense.set_pixel(5,4,n)
        sense.set_pixel(3,5,n)
        sense.set_pixel(6,5,n)
        sense.set_pixel(2,6,n)
        sense.set_pixel(5,6,n)

def soleado():
    sol = [
          n,n,n,n,n,n,n,n,
          n,n,n,n,n,n,n,n,
          n,n,n,n,n,n,n,n,
          n,n,y,y,y,n,n,n,
          n,n,y,y,y,n,n,n,
          n,n,y,y,y,n,n,n,
          n,n,n,n,n,n,n,n,
          n,n,n,n,n,n,n,n
          ]
    sense.set_pixels(sol)

    for i in range(10):
        time.sleep(1)
        sense.set_pixel(1,2,y)
        sense.set_pixel(3,2,y)
        sense.set_pixel(5,2,y)
        sense.set_pixel(1,4,y)
        sense.set_pixel(5,4,y)
        sense.set_pixel(1,6,y)
        sense.set_pixel(3,6,y)
        sense.set_pixel(5,6,y)
        time.sleep(1)
        sense.set_pixel(0,1,y)
        sense.set_pixel(3,1,y)
        sense.set_pixel(6,1,y)
        sense.set_pixel(0,4,y)
        sense.set_pixel(6,4,y)
        sense.set_pixel(0,7,y)
        sense.set_pixel(3,7,y)
```

```
        sense.set_pixel(6,7,y)
        time.sleep(1)
        sense.set_pixel(1,2,n)
        sense.set_pixel(3,2,n)
        sense.set_pixel(5,2,n)
        sense.set_pixel(1,4,n)
        sense.set_pixel(5,4,n)
        sense.set_pixel(1,6,n)
        sense.set_pixel(3,6,n)
        sense.set_pixel(5,6,n)
        sense.set_pixel(0,1,n)
        sense.set_pixel(3,1,n)
        sense.set_pixel(6,1,n)
        sense.set_pixel(0,4,n)
        sense.set_pixel(6,4,n)
        sense.set_pixel(0,7,n)
        sense.set_pixel(3,7,n)
        sense.set_pixel(6,7,n)

def sol_y_nubes():
    sol_nubes = [n,n,n,n,n,n,n,n,
                 n,n,n,n,n,y,n,n,
                 n,n,b,n,y,y,y,n,
                 n,b,b,b,b,y,y,y,
                 b,b,b,b,b,b,y,n,
                 b,b,b,b,b,b,b,n,
                 n,n,n,n,n,n,n,n,
                 n,n,n,n,n,n,n,n]
    sense.set_pixels(sol_nubes)
    time.sleep(30)

def sol_nubes_lluvia():
    sol_nubes_lluvia = [n,n,n,n,n,y,n,n,
                        n,n,b,n,y,y,y,n,
                        n,b,b,b,b,y,y,y,
                        b,b,b,b,b,b,y,n,
                        b,b,b,b,b,b,b,n,
                        n,n,n,n,n,n,n,n,
                        n,n,n,n,n,n,n,n,
                        n,n,n,n,n,n,n,n]
    sense.set_pixels(sol_nubes_lluvia)

    for i in range(10):
        sense.set_pixel(2,5,a)
        sense.set_pixel(5,5,a)
        time.sleep(1)
        sense.set_pixel(3,6,a)
        sense.set_pixel(6,6,a)
        time.sleep(1)
        sense.set_pixel(2,7,a)
        sense.set_pixel(5,7,a)
        time.sleep(1)
        sense.set_pixel(2,5,n)
        sense.set_pixel(5,5,n)
        sense.set_pixel(3,6,n)
        sense.set_pixel(6,6,n)
        sense.set_pixel(2,7,n)
        sense.set_pixel(5,7,n)

def nublado():

    nublado = [n,n,n,n,n,n,n,n,
               n,n,n,n,n,n,n,n,
               n,n,n,b,b,n,n,n,
               n,n,b,b,b,b,b,n,
               b,b,b,b,b,b,b,n,
               b,b,b,b,b,b,b,b,
               n,n,n,n,n,n,n,n,
               n,n,n,n,n,n,n,n]
    sense.set_pixels(nublado)
    time.sleep(30)

# Definición matricial de los números del uno al nueve
uno = [n,n,n,n,
       n,n,n,b,
       n,n,b,b,
       n,b,n,b,
       n,n,n,b,
```

```
       n,n,n,b,
       n,n,n,b,
       n,n,n,b]
dos = [n,n,n,n,
       n,n,b,n,
       n,b,n,b,
       n,n,n,b,
       n,n,n,b,
       n,n,b,n,
       n,b,n,n,
       n,b,b,b]
tres = [n,n,n,n,
        n,n,b,n,
        n,b,n,b,
        n,n,n,b,
        n,n,b,n,
        n,n,n,b,
        n,b,n,b,
        n,n,b,n]
cuatro = [n,n,n,n,
          n,b,n,n,
          n,b,n,b,
          n,b,n,b,
          n,b,b,b,
          n,n,n,b,
          n,n,n,b,
          n,n,n,b]
cinco = [n,n,n,n,
         n,b,b,b,
         n,b,n,n,
         n,b,n,n,
         n,n,b,n,
         n,n,n,b,
         n,n,n,b,
         n,b,b,n]
seis = [n,n,n,n,
        n,n,b,n,
        n,b,n,b,
        n,b,n,n,
        n,b,b,n,
        n,b,n,b,
        n,b,n,b,
        n,n,b,n]
siete = [n,n,n,n,
         n,b,b,b,
         n,n,n,b,
         n,n,n,b,
         n,n,b,n,
         n,b,n,n,
         n,b,n,n,
         n,b,n,n]
ocho = [n,n,n,n,
        n,n,b,n,
        n,b,n,b,
        n,b,n,b,
        n,n,b,n,
        n,b,n,b,
        n,b,n,b,
        n,n,b,n]
nueve = [n,n,n,n,
         n,n,b,n,
         n,b,n,b,
         n,b,n,b,
         n,n,b,b,
         n,n,n,b,
         n,n,n,b,
         n,n,n,b]
cero = [n,n,n,n,
        n,n,b,n,
        n,b,n,b,
        n,b,n,b,
        n,b,n,b,
        n,b,n,b,
        n,b,n,b,
        n,n,b,n]

# Función que muestra en el display del SenseHat la temperatura actual combinando matrices de numeros
def display_T(tempAct):
```

```
    digit1 = int(tempAct//10)
    digit2 = int(tempAct%10)

    if digit1 == 0:
          mat1 = cero
    elif digit1 == 1:
          mat1 = uno
    elif digit1 == 2:
          mat1 = dos
    elif digit1 == 3:
          mat1 = tres
    elif digit1 == 4:
          mat1 = cuatro
    elif digit1 == 5:
          mat1 = cinco
    elif digit1 == 6:
          mat1 = seis
    elif digit1 == 7:
          mat1 = siete
    elif digit1 == 8:
          mat1 = ocho
    else:
          mat1 = nueve

    if digit2 == 0:
          mat2 = cero
    elif digit2 == 1:
          mat2 = uno
    elif digit2 == 2:
          mat2 = dos
    elif digit2 == 3:
          mat2 = tres
    elif digit2 == 4:
          mat2 = cuatro
    elif digit2 == 5:
          mat2 = cinco
    elif digit2 == 6:
          mat2 = seis
    elif digit2 == 7:
          mat2 = siete
    elif digit2 == 8:
          mat2 = ocho
    else:
          mat2 = nueve

    matTemp = []
    for i in range(0,32,4):
            matTemp.extend([mat1[i],mat1[i+1],mat1[i+2],mat1[i+3]])
            matTemp.extend([mat2[i],mat2[i+1],mat2[i+2],mat2[i+3]])
    return matTemp
```

6. status.py

```
# Función que define el estado del cielo dada la precipitación y nubosidad actuales
def status(prec, nub):

    stat = ['soleado','no_llueve']

    if prec > 0.1 and prec <= 2.0:
        stat_prec = 'lluvia_debil'
    elif prec > 2.0 and prec <= 15.0:
        stat_prec = 'lluvia_moderada'
    elif prec > 15.0 and prec <= 30.0:
        stat_prec = 'lluvia_fuerte'
    elif prec > 30.0 and prec <= 60.0:
        stat_prec = 'lluvia_muyFuerte'
    elif prec > 60.0:
        stat_prec = 'lluvia_torrencial'
    else:
        stat_prec = 'no_llueve'

    if nub > 25 and nub <= 50:
        stat_nub = 'sol_y_nubes'
    elif nub > 50:
        stat_nub = 'nublado'
    else:
        stat_nub = 'soleado'

    stat = [stat_prec,stat_nub]
    return stat
```

7. timeLapse.py

```
from picamera import PiCamera
import time
import datetime
import os
import ffmpeg
import subprocess
import RPi.GPIO as GPIO
import subprocess
import cv2
import glob

# Llamdad al módulo de la cámara
camera = PiCamera()

# Función de captura de imagen
def takePicture():

    camera.resolution = (1920,1080) # definición de la resolución de la imagen
    camera.start_preview()

    try:
        # Toma de foto guardandola con fecha y hora de captura
        dt = datetime.datetime.now()
        h = dt.hour
        m = dt.minute
        fecha = datetime.date.today()
        fecha = str(fecha)
        hora = time.strftime("%H:%M:%S")
        camera.capture('/home/pi/Desktop/pruebas/fotos_timeLapse/'+fecha+"_"+hora+".jpg")

    except KeyboardInterrupt:
        camera.stop_preview()

# Función para eliminar todas las imágenes guardadas
def delete_images():
    images = glob.glob('/home/pi/Desktop/pruebas/fotos_timeLapse/*.jpg')

    for image in images:
        try:
            os.remove(image)
        except OSError as e:
            print(f"Error:{e.strerror}")

# Función para eliminar todos los videos creados
def delete_videos():
    videos = glob.glob('/home/pi/Desktop/finales/*.avi')

    for video in videos:
        try:
            os.remove(video)
        except OSError as e:
            print(f"Error:{e.strerror}")

# Función para crear el TimeLapse
def make_video():

    # Definición de rutas y nombres de archivos
    path = '/home/pi/Desktop/pruebas/fotos_timeLapse'
    image_folder = path
    video_name = str(datetime.date.today()) +'.avi'

    images = [img for img in os.listdir(path) if img.endswith(".jpg")]
    images = sorted(images)
    frame = cv2.imread(os.path.join(image_folder, images[0]))
    height, width, layers = frame.shape

    video = cv2.VideoWriter(video_name, 0, 3.5, (width,height))

    # Creación del TimeLapse incluyendo fecha y hora en las imagenes del video
    for image in images:

        img = cv2.imread(os.path.join(image_folder, image))

        font = cv2.FONT_HERSHEY_COMPLEX
        cv2.putText(img, image[:11]+image[11:29].replace('_',':'),(0,50),font,1,(255,255,0),2)


        video.write(img)

    video.release()
```

8. datos_Pred.py

```
from wwo_hist import retrieve_hist_data
import pandas as pd
import csv
from date_list import get_date_list

date_list = get_date_list()

frequency = 1
api_key = '0db87e3eb8534e36848173524221407'
location_list = ['zaragoza']
DataSense = retrieve_hist_data(api_key,
                               location_list,
                               '01-MAY-2021',
                               '20-JUL-2022',
                               frequency,
                               location_label = False,
                               export_csv = False,
                               store_df = True)

t_list = DataSense[0].tempC.astype(float).to_list()
h_list = DataSense[0].humidity.astype(float).to_list()
p_list= DataSense[0].pressure.astype(float).to_list()

df_Sense = pd.DataFrame((zip(date_list, t_list, h_list, p_list)), columns = ['Fecha' ,'Temperatura','Humedad','Presion'])
DataPredict = retrieve_hist_data(api_key,
                               location_list,
                               '02-MAY-2021',
                               '21-JUL-2022',
                               frequency,
                               location_label = False,
                               export_csv = False,
                               store_df = True)

tMAX_list =  DataPredict[0].maxtempC.astype(int).to_list()
tMIN_list =  DataPredict[0].mintempC.astype(int).to_list()
cloud_list = DataPredict[0].cloudcover.astype(int).to_list()
rain_list = DataPredict[0].precipMM.astype(float).to_list()

df_Predict = pd.DataFrame((zip(rain_list, cloud_list, tMAX_list, tMIN_list)), columns = ['PrecipitacionMM','Nubosidad','TempMAX','TempMIN'])

df = pd.concat([df_Sense, df_Predict], axis = 1)
df.to_csv('prueba_pred.csv')
```

9. PRED.py

```
import pandas as pd
import csv
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import Pipeline
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
import joblib

DataBase_pred = 'prueba_pred.csv'

df_pred = pd.read_csv(DataBase_pred, header=0)
df_pred = df_pred.drop(['Unnamed: 0'], axis = 1)

X_pred = df_pred.drop(['PrecipitacionMM','Nubosidad','TempMAX','TempMIN'], axis=1)
Y_TMax_pred = df_pred['TempMAX']
Y_TMin_pred = df_pred['TempMIN']
Y_Nub_pred = df_pred['Nubosidad']
Y_Prec_pred = df_pred['PrecipitacionMM']

X_train_pred, X_test_pred, Y_train_TMax_pred, Y_test_TMax_pred = train_test_split(X_pred, Y_TMax_pred, test_size = 0.3, random_state = 1)
X_train_pred, X_test_pred, Y_train_TMin_pred, Y_test_TMin_pred = train_test_split(X_pred, Y_TMin_pred, test_size = 0.3, random_state = 1)
X_train_pred, X_test_pred, Y_train_Nub_pred, Y_test_Nub_pred = train_test_split(X_pred, Y_Nub_pred, test_size = 0.3, random_state = 1)
X_train_pred, X_test_pred, Y_train_Prec_pred, Y_test_Prec_pred = train_test_split(X_pred, Y_Prec_pred, test_size = 0.3, random_state = 1)

numeric_cols = X_train_pred.select_dtypes(include=['float64','int']).columns.to_list()

numeric_transformer = Pipeline(steps=[('scaler',StandardScaler())])

preprocessor = ColumnTransformer(transformers=[('numeric',numeric_transformer, numeric_cols)], remainder='passthrough')

modelo_TempMax_pred = Pipeline([('preprocessing', preprocessor),('modelo',MLPRegressor(max_iter=1500))])
modelo_TempMin_pred = Pipeline([('preprocessing', preprocessor),('modelo', MLPRegressor(max_iter=1500))])
modelo_Nub_pred = Pipeline([('preprocessing', preprocessor),('modelo',GradientBoostingRegressor(learning_rate = 1.0, n_estimators = 150, random_state = 0, max_depth = 4))])
modelo_Prec_pred = Pipeline([('preprocessing', preprocessor),('modelo',GradientBoostingRegressor(learning_rate = 0.1, n_estimators = 400, random_state = 0))])

modelo_TempMax_pred.fit(X_train_pred, Y_train_TMax_pred)
modelo_TempMin_pred.fit(X_train_pred, Y_train_TMin_pred)
modelo_Nub_pred.fit(X_train_pred, Y_train_Nub_pred)
modelo_Prec_pred.fit(X_train_pred, Y_train_Prec_pred)

print(modelo_TempMax_pred.score(X_test_pred, Y_test_TMax_pred))
print(modelo_TempMin_pred.score(X_test_pred, Y_test_TMin_pred))
print(modelo_Nub_pred.score(X_test_pred, Y_test_Nub_pred))
print(modelo_Prec_pred.score(X_test_pred, Y_test_Prec_pred))

joblib.dump(modelo_TempMax_pred, 'modelo_TempMax_pred.pkl')
joblib.dump(modelo_TempMin_pred, 'modelo_TempMin_pred.pkl')
joblib.dump(modelo_Nub_pred, 'modelo_Nub_pred.pkl')
joblib.dump(modelo_Prec_pred, 'modelo_Prec_pred.pkl')
```