

UNIVERSIDAD POLITÉCNICA DE MADRID  
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

**Proyecto Innovador: RoomPi  
Sistema de medida de condiciones  
ambientales en interiores**

Sistemas Digitales II 2020 - 2021

Grupo LT-06

Victoria M. Gullón Alfonso  
Marcos Gómez Bracamonte

## 1. Idea y motivación

En el contexto de la asignatura Sistemas Digitales II del tercer curso del Grado en Ingeniería de Tecnología y Servicios de Telecomunicación, se presenta el Proyecto Innovador “RoomPi”, un sistema de monitorización de las condiciones ambientales en interiores.

La motivación de esta idea surgió a raíz de la situación vivida en plena pandemia por COVID-19 durante la realización de los exámenes finales del cuatrimestre de invierno. Al darse la necesidad de usar a modo de aula espacios diseñados originalmente para desempeñar otra función, las condiciones para realizar estas pruebas no fueron a veces las más adecuadas.

Sin embargo, el sistema RoomPi posee un diseño configurable, por lo que es completamente adaptable a multitud de situaciones y espacios: al hogar, aulas, quirófanos, gimnasios, bibliotecas, y un largo etcétera.



Figura 1: Renderizado preliminar del sistema instalado y en funcionamiento en el entorno de un aula.

## 2. Objetivos y solución

A la hora de llevar a cabo este proyecto, tenemos en mente 4 objetivos muy definidos.

- **Medición:** consideramos necesario medir periódicamente condiciones ambientales como la intensidad lumínica, el nivel de temperatura y humedad y el nivel de CO<sub>2</sub> en el aire.
- **Visualización:** los datos recogidos podrán visualizarse no solo in-situ de forma cíclica mediante un display LCD, sino que además, a través de una interfaz web, será posible

acceder al historial de mediciones, observar la media de los últimos días, realizar análisis estadísticos sobre los datos...

- **Avisos:** se hace obvio que, además de la recogida de medidas, es necesario realizar un procesamiento de estas. Así, en caso de no cumplirse las condiciones definidas como óptimas, se avisará al usuario visualmente con un array de LEDs a modo de semáforo y, en ocasiones muy concretas, también de forma sonora con un zumbido.
- **Personalización:** dada la versatilidad que queríamos otorgarle a nuestro proyecto, el usuario podrá personalizar a una serie de parámetros mediante la interfaz web, para así poder adaptar el sistema a la situación y espacio que necesite.

La solución idónea para implantar un sistema que realice todas estas funciones sería mediante el conexionado de los sensores y actuadores a un microcontrolador, como por ejemplo, el de STM32 y colocar uno por habitación. A su vez, todos los microcontroladores de una misma zona estarían conectados a una Raspberry Pi que actuaría como nodo central, por lo que contendría el sistema de gestión de los microcontroladores de las habitaciones, así como los subsistemas web necesarios para la visualización de los datos y la comunicación con la BBDD.

Sin embargo, para esta asignatura hemos creado un prototipo en el que conectamos directamente los sensores y actuadores a nuestra Raspberry Pi. En la fig. 2 se muestra de manera conceptual cómo se organizaría el sistema.

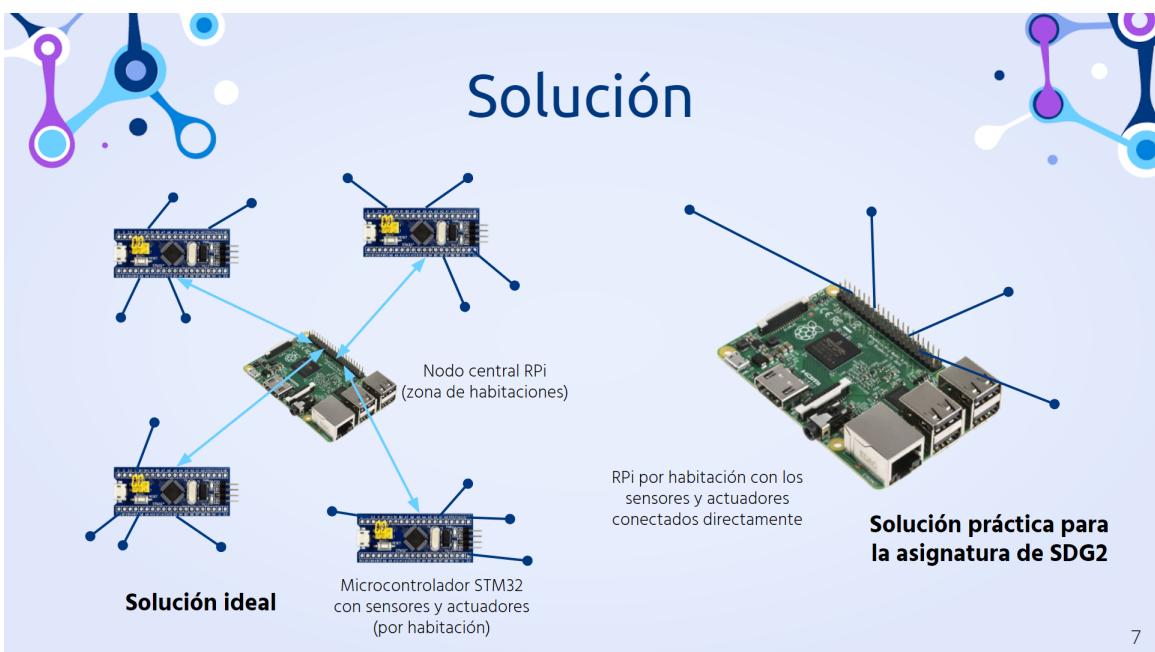


Figura 2: Diagrama de la solución ideal frente a la solución implementada para la asignatura de SDGII. Obtenido de la presentación del proyecto.

### 3. Parte Técnica

Teniendo en cuenta el feedback de mejoras recibido por parte del profesorado mediante el Hito Intermedio y nuestras propias mejoras, se presenta tanto el Hardware como el Software que finalmente conforman nuestro sistema RoomPi.

Destacar que tanto los drivers de los sensores y actuadores como los subsistemas de medida y visualización, que serán explicados a continuación, los hemos desarrollado en el lenguaje C, según el estándar C99.

En lo relativo al funcionamiento de la web, la parte desarrollada por nosotros es la aplicación web que permite realizar los ajustes de los distintos parámetros del sistema. Esta aplicación web está realizada en Python versión 3.8.

#### 3.1. Subsistemas

El sistema general RoomPi está dividido en 3 subsistemas distintos que se centran en distintas tareas:

1. **Subsistema de medida:** Este subsistema está compuesto por los sensores, por las FSMs de cada sensor y por la FSM que procesa las medidas de estos últimos.
2. **Subsistema de visualización:** Este subsistema lo componen los actuadores y las FSMs que los controlan.
3. **Subsistema web:** Este subsistema lo componen las aplicaciones y servicios web del sistema y la base de datos, que se ejecutan como instancias de un sistema de virtualización explicado más adelante.

Una visión esquemática del subsistema puede encontrarse en la fig. 3

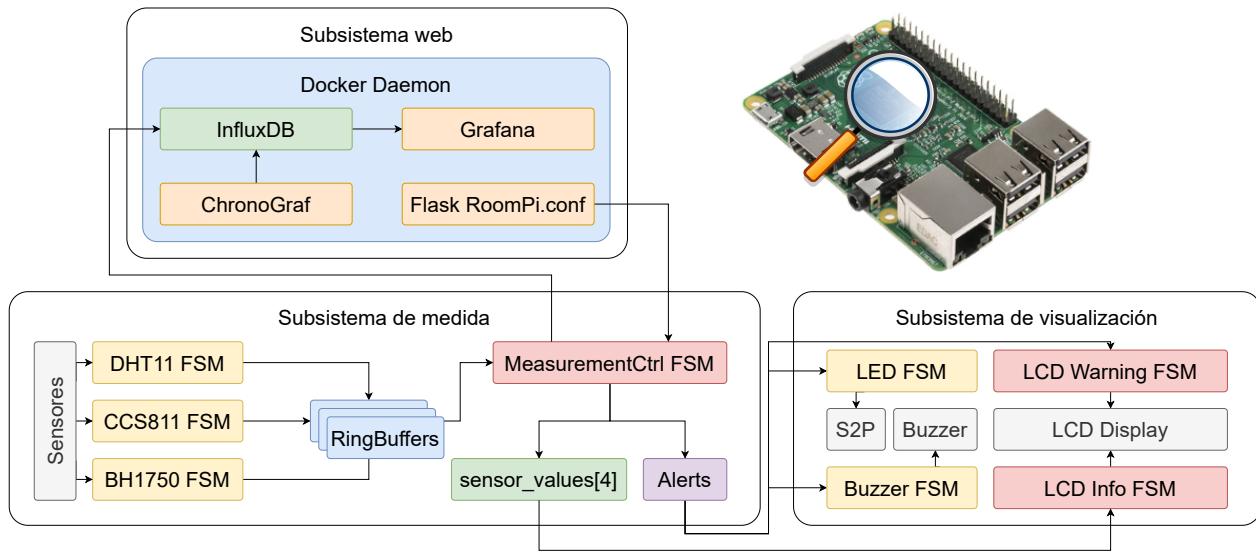


Figura 3: Visión general del proyecto donde se muestran los subsistemas que lo componen, las FSMs incluidas dentro de dichos subsistemas y los sensores o actuadores involucrados así como las estructuras en memoria más destacables.

### 3.2. Sensores, actuadores y drivers

El sistema RoomPi hace uso de diversos sensores para obtener las medidas del entorno en el que se encuentra. Cada uno de estos sensores mide magnitudes atmosféricas distintas y tiene características distintas. Los sensores empleados son los siguientes:

1. DHT11

Magnitud medida	Temperatura y humedad en °C y % de humedad relativa
Modelo	DHT11 DIP 1-wire
Fabricante	Aosong Electronics Co., Ltd.
Datasheet	<a href="#">DHT11_Aosong.pdf</a>
Rango de medidas y resolución	0-50 °C, 20-90 % RH +/- 1 °C, 1 % RH
Interfaz	La comunicación con el sensor se realiza a través de un protocolo específico a través de una sola línea de señal. La escritura de comandos y la lectura de medidas se realizan en la misma línea de manera consecutiva. El protocolo empleado se basa en la emisión o recepción de pulsos de determinada longitud que se interpretan posteriormente como una señal digital. Para más detalle sobre el protocolo de comunicación, consultar el datasheet.
Tensión de alimentación	3.3-5.0 V DC
Consumo de corriente	Realizando medida: 1-1.5 mA En espera: 40 μA

## 2. BH1750

Magnitud medida	Flujo luminoso por unidad de área (lux)
Modelo	BH1750FVI SMD en placa breakout con conexiones DIP, compensación por longitud de onda
Fabricante	Rohm Semiconductor Co., Ltd.
Datasheet	<a href="#">bh1750fvi-e-186247.pdf</a>
Rango de medidas y resolución	1-65535 lx Resol. 1 lux, +/- 20 % accuracy
Interfaz	Comunicación por bus I2C, protocolo de boot y configuración de sampling rate. Dirección configurada a 0x23.
Tensión de alimentación	2.4-3.6 V DC
Consumo de corriente	Realizando medida: 120-190 μA En espera: 2 μA

## 3. CCS811

Magnitud medida	Sensor MOX que detecta alcoholes, aldehídos, cetonas, ácidos orgánicos, aminoácidos, hidrocarburos alifáticos y aromáticos y genera una medición de CO <sub>2</sub> equivalente a través de la concentración de gases en el entorno
Modelo	CCS811 en placa breakout con conexiones DIP
Fabricante	ams AG
Datasheet	<a href="#">CCS811_Datasheet-DS000459.pdf</a>
Rango de medidas y resolución	400-8192 ppm Resol. 1 ppm, +/- 30 % accuracy Calibración de 30s antes de cada uso Calibración esporádica entre medidas Periodo de burn-in inicial: 24h
Interfaz	Comunicación por bus I2C, protocolo de boot, configuración de resolución y medida. Dirección configurada a 0x5a.
Tensión de alimentación	1.8-3.3 V DC
Consumo de corriente	Realizando medida: 26 mA En espera: ≈0.6 mA

Asimismo, el sistema precisa de actuadores para visualizar la información correspondiente de los sensores en tiempo real. Los actuadores empleados son los siguientes:

### 1. HD44780

Utilización	Pantalla LCD azul de matriz de puntos con 2 líneas y 16 caracteres por línea
Modelo	HD44780U DIP
Fabricante	Hitachi Co. Ltd.
Datasheet	<a href="#">HD44780.pdf</a>
Tiempo de actualización	Clear e inicialización: 2 s Escritura: 2 MHz
Interfaz	Bus 4 pines digitales, señal de enable y reloj
Tensión de alimentación	LCD Driver: 5-5.5 V DC Backlight: 3.3 V DC
Consumo de corriente	LCD Driver: 200-300 μA Backlight: 2 mA

### 2. SN74HC595

Utilización	Registro de desplazamiento de 8 bits (conversor serie-paralelo) para tira de LEDs de aviso
Modelo	SN74HC595 DIP
Fabricante	Texas Instruments Inc.
Datasheet	<a href="#">scls041i.pdf</a>
Tiempo de actualización	$t_{pd} = 13 \text{ ns (typ.)}$
Interfaz	Serial In, reloj y output enable
Tensión de alimentación	-0.5 - 7 V DC
Consumo de corriente	IC: máx 80 $\mu\text{A}$ Input corriente máxima: 20 mA Corriente output por pin: 35 mA

### 3. Buzzer

Utilización	Buzzer activo con oscilador para señal de alarma
Modelo	SDC1610
Fabricante	TDK Semiconductor Co. Ltd.
Datasheet	<a href="#">electromagnetic_buzzer_sdc_en.pdf</a>
Tiempo de actualización	Freq: $2400 \pm 20\% \text{ Hz}$
Interfaz	N/A
Tensión de alimentación	3-8 V DC
Consumo de corriente	max 40 mA

Además el sistema emplea otros componentes para su correcto funcionamiento como son los siguientes:

#### 1. 8 LEDs

Utilización	Color variado, para visualización de alertas
Tensión de alimentación	3.3 V DC
Consumo de corriente	10 mA cada uno

#### 2. LM117

Utilización	Regulador de tensión ajustable mediante resistencias
Modelo	LM117T TO-220
Fabricante	STMicroelectronics N.V.
Datasheet	<a href="#">lm317.pdf</a>
Tensión de alimentación	5 V DC
Tensión de salida	3.2-3.4 V DC
Corriente de entrada	2.4 A (USB power supply)
Corriente de salida	1.2 A (medido)

El conexionado de los distintos componentes del sistema completo puede encontrarse en la fig. 4 y en el PDF adjunto con mayor detalle.

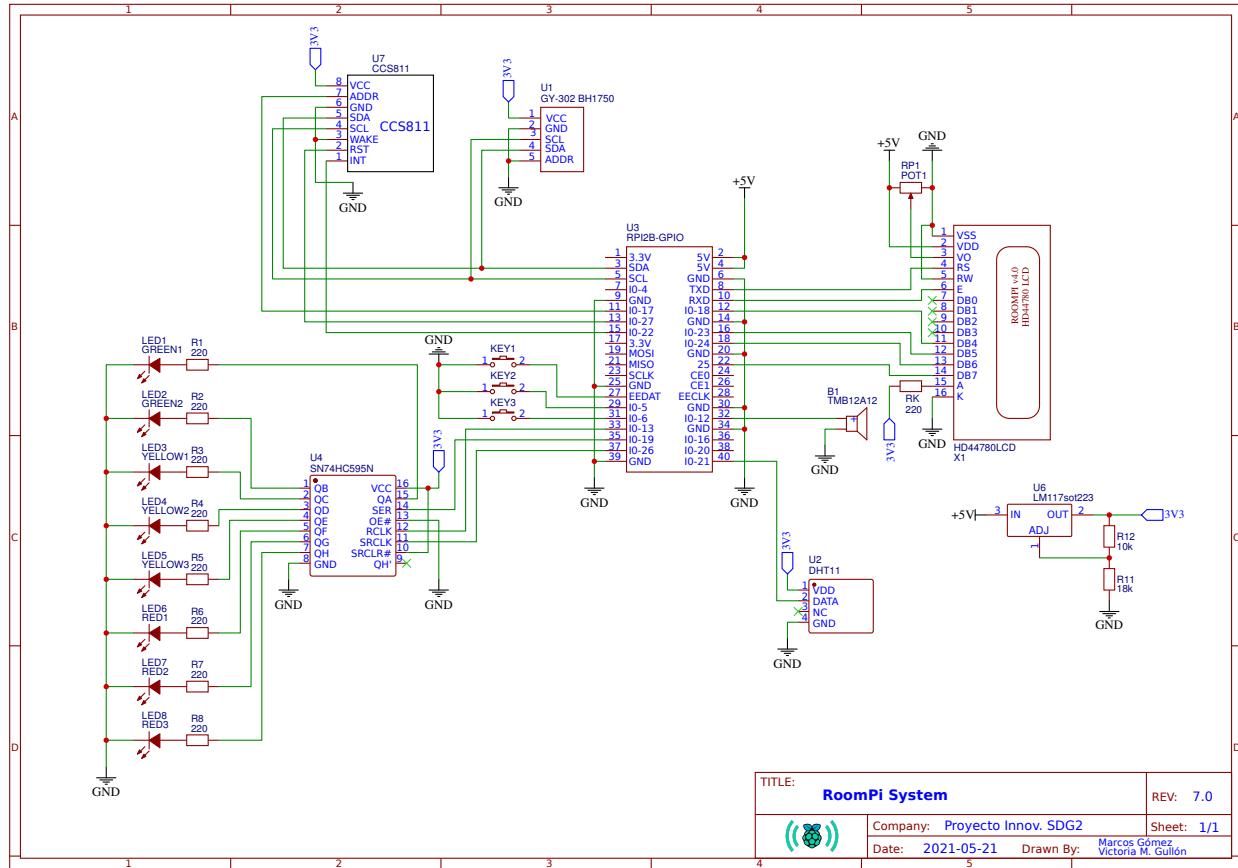


Figura 4: Esquemático de todo el sistema completo, con todos los componentes y sus conexiones

### 3.2.1. Desarrollo de drivers

En el código, los distintos sensores se representan como punteros a estructuras que contienen toda la información necesaria de los distintos sensores, posteriormente se desarrollan una serie de funciones que trabajan sobre los punteros a dichos structs, como pueden ser las de crear el sensor, destruir el sensor y realizar medidas o configuraciones. A modo ilustrativo se adjunta a continuación la declaración del struct del sensor de luz y las funciones asociadas al tipo

```
typedef struct {
    int id; // sensor id
    int addr; // sensor i2c address
    int mode; // sensor operating mode (continuous/one time-hires/lowres)
```

```
int fd; // file descriptor handle representing the i2c device
int lux;

fsm_t *fsm; // FSM that performs a measurement from the light sensor
tmr_t *timer;
// timer that governs a flag used by the light
// sensor measurement FSM (5 s periodic default)
} BH1750Sensor;

BH1750Sensor* BH1750Sensor__create(int id, int addr, int mode);
void BH1750Sensor__destroy(BH1750Sensor* sensor_instance);
int BH1750Sensor__perform_measurement(BH1750Sensor* sensor_instance);
int BH1750Sensor__lux_value(BH1750Sensor* sensor_instance);
```

Para algunas partes del desarrollo del driver del sensor CCS811 se han empleado fragmentos de código de otros desarrolladores. La referencia y licencia a estas porciones de código se puede encontrar en los comentarios al inicio de los archivos pertinentes de código fuente y en el archivo LICENSE del repositorio de GitHub donde se aloja el proyecto.

### 3.3. FSMs

Para la realización de todas las máquinas de estado finito del proyecto se ha utilizado la librería proporcionada en la asignatura.

#### 3.3.1. Subsistema de medida

Este subsistema está compuesto por unas FSMs individuales en cada sensor que realizan medidas individuales y una FSM controladora que realiza el procesamiento de las medidas de los sensores, actualiza la base de datos con los nuevos datos y realiza las activaciones o desactivaciones de los flags de avisos. El subsistema de medida se ha rediseñado completamente desde la versión desarrollada en el hito, lo que se comentará con mayor profundidad más adelante.

##### 1. FSM Individual por sensor

Para cada sensor se ha creado una máquina de estados que realiza una medida periódicamente y cuyo valor se almacena en una variable de tipo SensorValueType. Esta variable, a su vez, se almacena en el búfer circular que corresponde a ese sensor, el cual tiene una capacidad por defecto de 5, aunque esta es una característica configurable por el usuario.

La declaración del struct de SensorValueType se adjunta a continuación.

```
typedef struct {
    enum {
```

```

        is_int, is_float, is_error
    } type;
union {
    int ival;
    float fval;
} val;
} SensorValueType;

```

Los sensores utilizados generan medidas de distintos tipos, floats o enteros, por lo que se han definido los tipos `is_float` e `is_int`. Además, si se da un error en una medida, en lugar de ser descartada se almacenará en una variable del tipo `is_error`.

A continuación se adjuntan en las figuras 5, 6 y 7 las máquinas de estados individuales de los sensores.



Figura 5: FSM individual del sensor de temperatura y humedad: DHT11

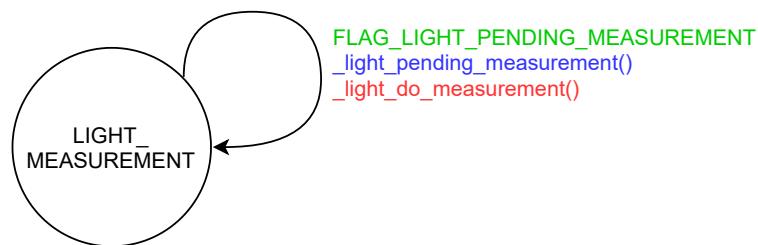


Figura 6: FSM individual del sensor luz: BH1750

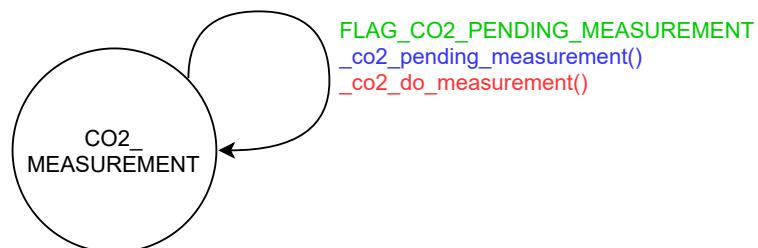


Figura 7: FSM individual del sensor de CO2: CCS811

Todas estas FSMs se componen de un único estado en el que, como indica su nombre, se realiza una medida llamando a la función `_x_do_measurement`. Concretamente, esta función consiste en que la Raspberry Pi lea el valor que el sensor está expresando en ese momento y lo guarde en una variable del tipo `is_float` o `is_int` (dependiendo del sensor) o del tipo `is_error`, como ya se ha explicado con anterioridad, para posteriormente almacenarla en el búfer circular correspondiente a ese sensor. Además, el `FLAG_X_PENDING_MEASUREMENT` será desactivado. No se llevará a cabo otra medida hasta que la función `_x_pending_measurement` detecte que el flag en cuestión ha sido activado por el timer que gobierna esa máquina de estados.

## 2. FSM Control de medidas

Esta FSM consta de 3 estados, en los que se encarga del procesamiento de las medidas almacenadas en los búfers circulares de cada sensor, la generación de alertas si procede y la subida de los datos procesados a la BBDD.

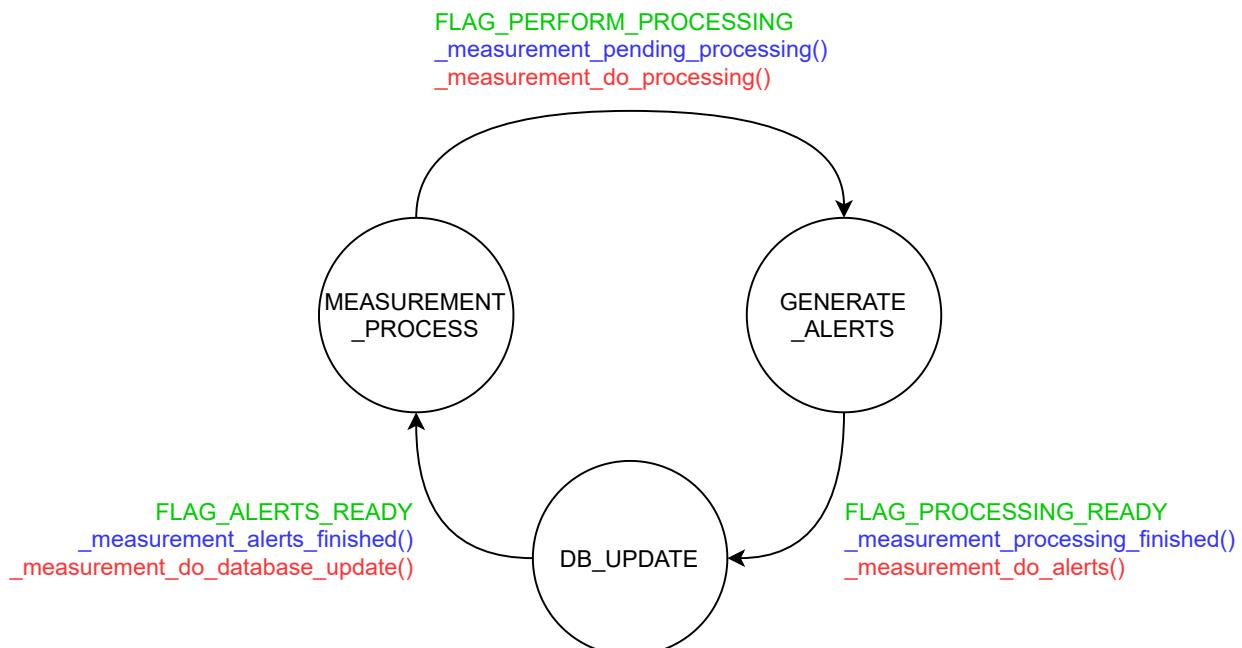


Figura 8: FSM MeasurementCtrl

Cuando nos encontramos en el estado `MEASUREMENT_PROCESS` y la función `_measurement_pending_processing` comprueba que `FLAG_PERFORM_PROCESSING` ha sido activado, significa que la función `_measurement_do_processing` ya puede procesar las

medidas que se encuentran almacenadas en el búfer circular asignado a cada sensor. Concretamente, se descartarán el valor más alto y el más bajo, así como los calificados como error. Con las medidas que no han sido descartadas se realiza el valor medio.

Una vez realizado esto, la función `_measurement_processing_finished` detectará que el llamado `FLAG_PROCESSING_READY` está activo, por lo que `_measurement_do_alerts` generará las alertas correspondientes. En caso de que el valor medio, resultado del procesamiento, se encuentre dentro del rango definido como óptimo para esa magnitud, no se dará ninguna alerta. En cambio, si este valor está fuera del rango o si está fuera del rango y es un valor muy extremo, se deberá dar una alerta de anomalía o de emergencia, respectivamente.

Finalmente, cuando `_measurement_alerts_finished` compruebe que `FLAG_ALERTS_READY` se ha activado, la función `_measurement_do_database_update` subirá los valores procesados de los diferentes sensores a la BBDD.

### 3.4. Subsistema de visualización

#### 1. FSM LEDs

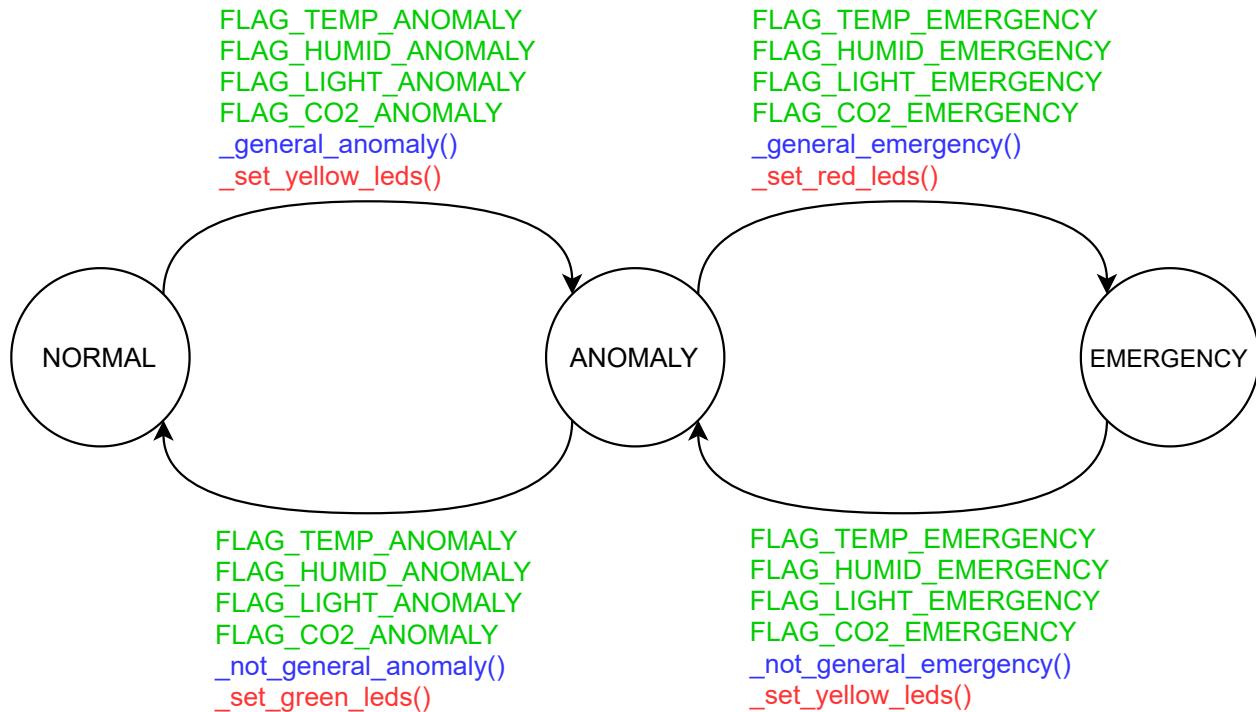


Figura 9: FSM LEDs

Como bien se ha comentado con anterioridad, el array de LEDs actuará a modo de semáforo, poniendo de manifiesto si las condiciones ambientales están dentro del rango considerado como óptimo o si, sin embargo, hay alguna anomalía o incluso existe una situación de emergencia.

Por tanto, observamos que esta FSM se compone de 3 estados en los que se van cambiando los LEDs de color en función de la situación: verde si todo está en orden, amarillo en caso de que algún valor se encuentre fuera del rango óptimo y rojo si se detecta algún valor muy inadecuado.

En caso de que la FSM Control de medidas, anteriormente explicada, dé una alerta indicando que se ha registrado una medida de valor anómalo, activará el flag en cuestión FLAG\_X\_ANOMALY. Esto será detectado por la función general\_anomaly y el semáforo se pondrá en amarillo. Si además de activarse este flag, se activa el FLAG\_X\_EMERGENCY, se estará indicando que el valor registrado no es que únicamente esté fuera del rango óptimo, sino que además es muy valor muy inadecuado, por lo que el semáforo pasará a ser de color rojo. En el momento que la función \_not\_general\_emergency detecte que el FLAG\_X\_EMERGENCY se desactiva o que la función \_not\_general\_anomaly compruebe que no hay ningún flag activo, el semáforo volverá a ponerse de color amarillo o verde, respectivamente.

## 2. FSM Buzzer

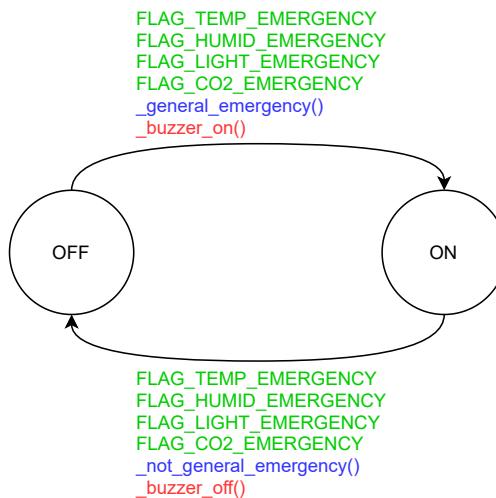


Figura 10: FSM Buzzer

También se comentó que, en algunas ocasiones, el aviso visual iría acompañado de uno sonoro, concretamente de un zumbido. Esto ocurrirá en el caso de que alguna medida sea tan inadecuada que se interprete como una emergencia.

Cuando la máquina de estados que procesa las medidas active alguno de los flags de emergencia, la función `_general_emergency` lo comprobará y se encenderá el zumbador mediante la función de activación `_buzzer_on`. En caso contrario, `_not_general_anomaly` detectará que no hay ningún `FLAG_X_EMERGENCY` activo, por lo que se apagará el aviso sonoro gracias a la función de desactivación `_buzzer_off`.

### 3. FSM LCD información

El display consta de dos filas: la de arriba es utilizada para sacar los avisos por pantalla (anomalía o emergencia) y la de abajo para mostrar los valores procesados de las medidas.

La máquina de estados que se puede ver a continuación es la que controla la fila inferior.

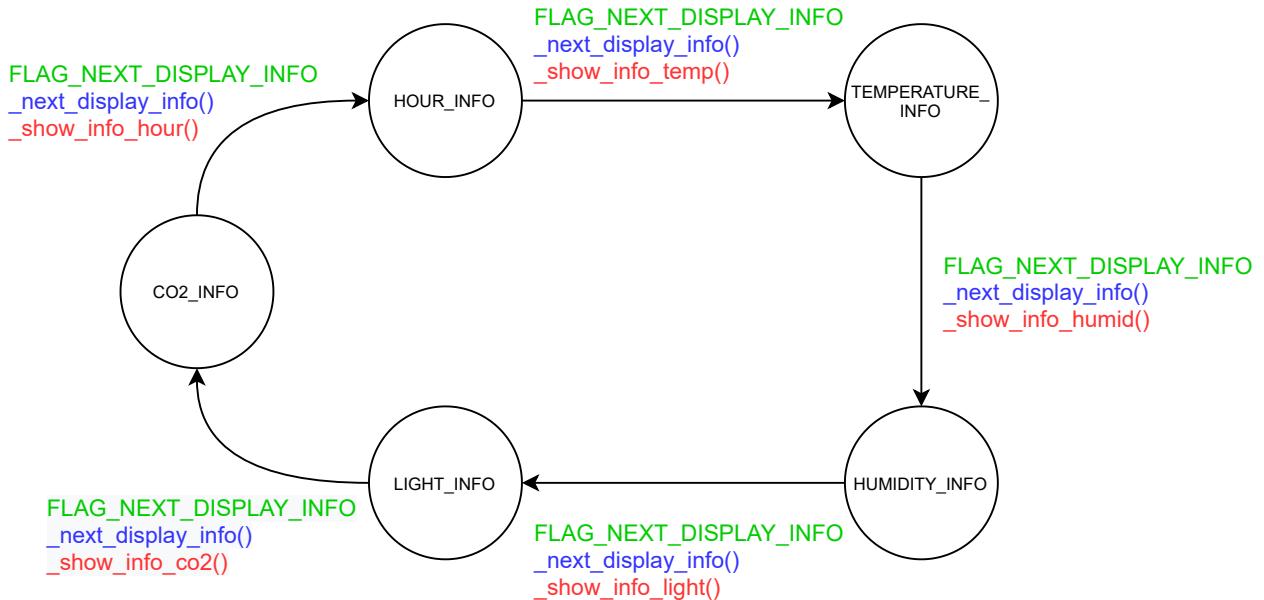


Figura 11: FSM LCD Información

Consta de 5 estados, uno por tipo de información que debe aparecer por pantalla. Cuando el timer que gobierna esta FSM activa `FLAG_NEXT_DISPLAY_INFO` y esto es detectado mediante la función `_next_display_info`, la función `_show_info_x` muestra por pantalla el valor en cuestión.

### 4. FSM LCD Avisos

En este caso, la máquina de estados siguiente es la que controla la fila superior del display LCD, aquella que muestra, si procede, los avisos.

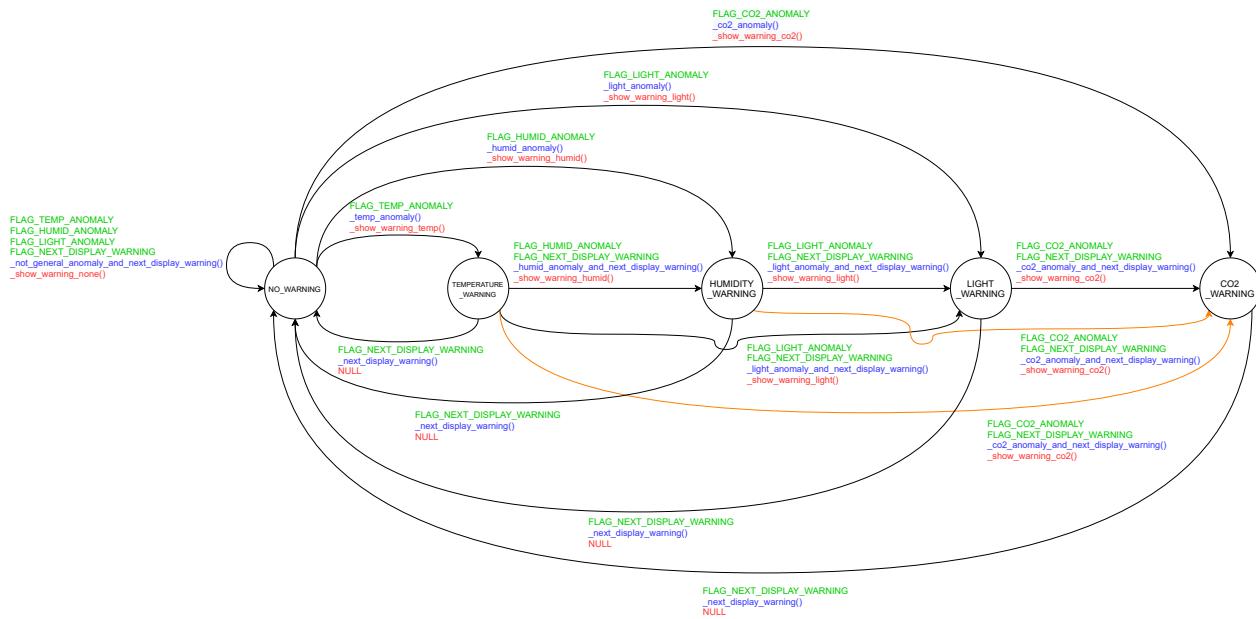


Figura 12: FSM LCD avisos

Esta máquina de estados consta, también, de 5 estados, uno por tipo de aviso que puede darse. En caso de que la función `_not_general_anomaly_and_next_display_warning` detecte que todos los flags de anomalía se encuentran desactivados pero que `FLAG_NEXT_DISPLAY_WARNING` sí está activo, significa que no hay que dar ningún aviso y que ha pasado el tiempo estipulado para refrescar la pantalla, por lo que aparecerá una nueva pantalla con el mensaje “roompi v7.0” gracias a la función `_show_warning_none`.

En caso de estar en el estado `NO_WARNING` y, de repente, la función `_x_anomaly` detecte que un `FLAG_X_ANOMALY` se activa, la función `_show_warning_x` mostrará inmediatamente por pantalla el aviso correspondiente.

Si, en cambio, es necesario mostrar más de un aviso, estos no se mostrarán inmediatamente, sino que después de haber mostrado uno de ellos deberá darse la condición de que el flag de anomalía en cuestión y `FLAG_NEXT_DISPLAY_WARNING` están ambos activos. Es decir, se debe esperar a que haya pasado el tiempo estipulado de refresco de pantalla para dejar de mostrar uno de los avisos y pasar a mostrar el siguiente.

### 3.5. Timers

Después de haber detallado las máquinas de estados que conforman los diferentes subsistemas mencionados, llega el turno de los temporizadores por los que son gobernadas, para cuya realización se ha utilizado la librería proporcionada en la asignatura.

### 3.5.1. Subsistema de medida

#### 1. Timer Individual por sensor

Se hace necesario crear un timer independiente para cada sensor al medir magnitudes muy diversas: la temperatura tiene una inercia mucho mayor (varía más lentamente) que la intensidad lumínica, por lo que tiene sentido que el período de medida del sensor de luz sea mucho mayor. Este es otro parámetro completamente configurable por el usuario.

Estos timer serán los que activen el llamado FLAG\_X\_PENDING\_MEASUREMENT y así marcará cuándo es necesario tomar una medida de ese sensor.

#### 2. Timer Control de medidas

El control de medidas es gobernado por un timer cuyo período es igual o mayor al mínimo común múltiplo de los períodos de medida de los timers de los sensores, este parámetro también es personalizable mediante la interfaz web.

Este temporizador activa el FLAG\_PERFORM\_PROCESSING, que representa el pistoletazo de salida para que la FSM Control de medidas acceda a los búferes circulares, procese las medidas, genere las alertas si procede y suba los valores procesados a la BBDD, como ya se ha comentado con anterioridad.

### 3.5.2. Subsistema de visualización

Dentro de este subsistema solo hemos necesitado utilizar un timer que cada 5 segundos activa FLAG\_NEXT\_DISPLAY\_INFO y FLAG\_NEXT\_DISPLAY\_WARNING para que se refresquen las dos filas que conforman la visualización en la pantalla LCD.

## 3.6. Interrupciones

Como adición a las funcionalidades descritas en el hito y tomando como base las diversas sugerencias propuestas por los profesores en su evaluación, una de las principales nuevas prestaciones es la adición de la posibilidad de controlar e interactuar con el sistema físicamente.

Esta interacción se realiza a través de tres botones situados en la parte exterior del sistema. Las interrupciones están configuradas para lanzarse con el flanco de bajada de los pines de los botones. Esto se debe a que dichos pines están configurados a pull up (PUD\_UP), lo que significa que cuando los botones se pulsan, la tensión baja a GND y mientras no está pulsado, el botón está conectado a 3V3. En las funciones de atención a las subrutinas de las interrupciones se ha implementado un sistema antirrebotes para evitar pulsaciones espurias.

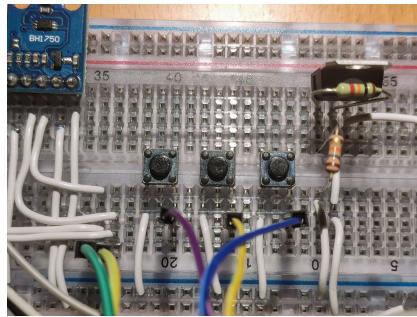


Figura 13: Botones usados para el control físico del sistema

Presionando los diferentes botones se consiguen distintas funcionalidades:

1. Presionando el botón de la izquierda, se ordena al sistema realizar de nuevo un procesamiento de las últimas medidas realizadas por los sensores. Esto se consigue forzando a que se active el flag FLAG\_PERFORM\_PROCESSING y por lo tanto el desencadenamiento de la FSM MeasurementCtrl con las funcionalidades detalladas en los apartados anteriores.
2. Presionando el botón central, se ordena al sistema avanzar a la siguiente pantalla en la visualización del display. Mientras que por defecto las pantallas con los distintos datos se muestran de manera cíclica cada 5 segundos; si se presiona este botón se fuerza la activación del flag FLAG\_NEXT\_DISPLAY del que se nutren las FSMs del subsistema de visualización, avanzando la visualización a la siguiente pantalla.
3. Cada vez que se presiona el botón de la derecha, se habilita o deshabilita un flag en el programa mediante una operación XOR. Si el flag está activo justo antes de la ejecución de la FSM que controla el zumbador, la función de activación se omitirá y se forzará a LOW la salida del pin al que está conectado el zumbador, deshabilitándolo. Cuando este flag interno está desactivado, el funcionamiento del buzzer es el normal.

También se han empleado interrupciones en el pin de datos del sensor DHT11 para mejorar las lecturas, más detalle en la sección de Temporización DHT11.

### 3.7. Web

Como mejora a partir del sistema desarrollado en el hito, se ha implementado un sistema para la visualización y análisis de datos en directo e histórico de datos a través de una interfaz web. Asimismo se ha desarrollado una aplicación web que permite al usuario personalizar los distintos aspectos de configuración del sistema. Para guardar un histórico de los datos de los sensores se emplea una base de datos.

### 3.7.1. Orquestación de servicios de almacenamiento y web

Los diversos servicios que se detallan a continuación están orquestados y gestionados como contenedores, utilizando el famoso servicio de gestión de virtualización de microservicios Docker. En un archivo de esquema de configuración docker-compose.yml se detallan los servicios que se emplean, las imágenes que se utilizan, la red local virtual que emplean los contenedores así como los volúmenes de datos persistentes almacenados en la tarjeta de almacenamiento flash de la Raspberry Pi.

```
pi@adelina:~ $ docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED             STATUS              PORTS     NAMES
c1b8efa3ad0b   docker-compose-influxdb-grafana_flask   "python3 ./app.py"   2 minutes ago    Up 2 minutes
a2d4f94141a7   grafana/grafana:latest                 "/run.sh"            17 hours ago     Up 15 hours
04ffe9451c76   chronograf:latest                   "/entrypoint.sh chro..."  17 hours ago     Up 15 hours
0dc44176dfc1   arm32v7/influxdb:latest                "/entrypoint.sh infl..."  17 hours ago     Up 15 hours
pi@adelina:~ $
```

Figura 14: Contenedores docker corriendo en el sistema tras ejecutar docker ps

### 3.7.2. Almacenamiento de datos

En la FSM MeasurementCtrl se realiza, entre otras cosas, la actualización de la base de datos con las medidas nuevas de los sensores. Para la base de datos se ha empleado InfluxDB, ya que es una BBDD basada en PostgreSQL pero optimizada para el tratamiento de datos de series temporales (TSDB). InfluxDB se instaura como un contenedor persistente en la Raspberry Pi cuyo acceso de lectura y escritura está limitado a la red local virtual de la RPi por motivos de seguridad. Se inicializa con 4 campos disponibles para medidas: “temp”, “rh”, “lux”, “eco2”. En el código fuente del sistema se emplea la librería libcurl para permitir la comunicación con la base de datos a través del protocolo HTTP. Los datos se cargan en la base de datos con una petición POST a la API de InfluxDB detallando la medida que se quiere registrar, el valor de dicha medida y un timestamp (UNIX epoch). En caso de que fallara la transacción HTTP, se reintenta silenciosamente y si vuelve a fallar, se muestra un mensaje por stderr y en el LCD del sistema. La base de datos tiene una política de retención en la que los datos con fecha anterior a 2 meses se eliminan del sistema automáticamente para ahorrar espacio en el almacenamiento SDHC.

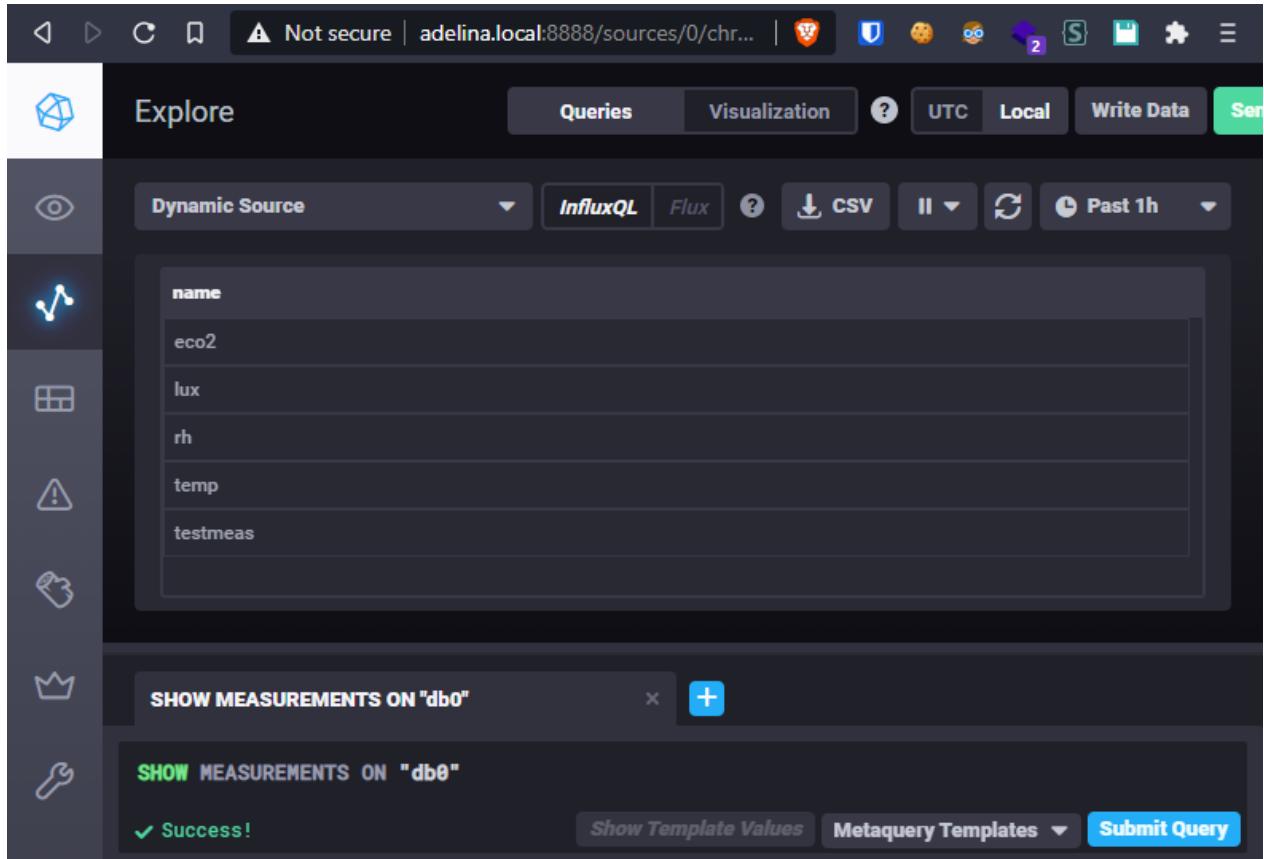


Figura 15: Captura de Chronograf mostrando las medidas de la base de datos

### 3.7.3. Gestión externa de la base de datos

La gestión de la base de datos se realiza con la aplicación Chronograf, diseñada para trabajar con instancias de InfluxDB. El servicio se instaura como un contenedor y su acceso está restringido a la LAN donde se encuentra la RPi.

### 3.7.4. Visualización y análisis de las medidas

La visualización y análisis de las medidas almacenadas en la BBDD se realiza a través de un tablero en la aplicación web Grafana. Grafana es una aplicación que permite la creación de tableros para mostrar datos de distintas fuentes de manera gráfica. Nuestra instancia de Grafana en Docker realiza peticiones a la BBDD InfluxDB.

En el tablero de monitorización se observa un gráfico para cada medida que realiza el sistema donde se aprecian las regiones de valores de emergencia y de aviso para las distintas magnitudes. Asimismo se incluyen distintos paneles que facilitan el análisis rápido de los datos como, por ejemplo, la media de las medidas en los últimos 15 minutos de operación del sistema.

El panel y su distribución es completamente configurable así como el rango temporal de los datos representados en las gráficas, el tiempo de actualización de la interfaz y la configuración de los parámetros del sistema; todos se pueden modificar en la parte superior del tablero con un menú y un botón respectivamente. La interfaz de Grafana está disponible públicamente en modo de espectador mientras que la edición sólo está disponible bajo autenticación.



Figura 16: Captura del tablero de Grafana "Monitorización RoomPi"

### 3.7.5. Configuración de los parámetros del sistema

La configuración de los parámetros del sistema se realiza a través de una aplicación web propia programada en Python empleando la librería Flask. Esta aplicación web presenta al usuario un formulario donde éste puede configurar los parámetros empleados por el sistema RoomPi:

1. En primer lugar se encuentra un desplegable con distintos perfiles preestablecidos de distintos lugares (p.ej. Aula B, biblioteca, hogar, gimnasio...) u órganos oficiales (CSIC, CDC, OMS) para aplicar rápidamente. Estos perfiles modifican los rangos de las magnitudes utilizados para determinar situaciones de anomalía o emergencia.
2. Seleccionando el perfil "Manual" se habilita la edición de los campos numéricos relativos a los rangos mencionados anteriormente. El usuario es capaz ahora de ajustar manualmente cada valor.

3. Seguidamente se puede ajustar el tiempo de los distintos timers de las máquinas de estados o dejar los valores por defecto.
4. Finalmente, pulsando sobre el botón “Aplicar” se genera en el directorio raíz de la RPi el archivo `roompi.conf` que contiene las distintas variables ajustadas. Al arrancar el sistema se lee este archivo y se definen las distintas variables que se utilizan. En caso de que el archivo estuviera mal formado o no existiera el sistema utiliza los valores predeterminados.

Ajuste de parámetros de RoomPi

Selección de perfil...	Temperatura crítica inferior (°C)	HR crítica inferior (%H)	Intensidad lumínica crítica (lux)	Timer FSM MeasurementCtrl (ms)
CSIC Aulas	5	10	150	60000
Default	Temperatura crítica superior (°C)	HR crítica superior (%H)	Intensidad lumínica de aviso (lux)	Timer FSM DHT11 (ms)
Aulas B	33	82	350	5000
Biblioteca	Temperatura de aviso inferior (°C)	HR de aviso inferior (%H)	CO2 equivalente crítica (ppm)	Timer FSM BH1750 (ms)
Hogar Urbano	18	30	2100	5000
Gimnasio/Entrenam.	Temperatura de aviso superior (°C)	HR de aviso superior (%H)	CO2 equivalente de aviso (ppm)	Timer FSM CCS811 (ms)
Quirófano/ICU/Radiolog.	28	70	1200	5000
<b>CSIC Aulas</b>	Aplicar			Timer FSM OutputCtrl (ms)
OMS Trabajo				5000
CDC EEUU				
Custom...				

Figura 17: Captura de la aplicación web de ajuste de parámetros

## 4. Problemas y otras mejoras

### 4.1. Mencionadas en el hito

#### 4.1.1. Temporización DHT11

En el hito se comentó como las lecturas del sensor de temperatura y humedad emplean un protocolo muy sensible al tiempo para el cual el uso de delays era insuficiente debido a que la RPi no puede garantizar la precisión de un sistema RTOS. La comunicación con este sensor se realiza a través de flancos de subida y bajada con duraciones determinadas en una línea dúplex.

En la versión actual, la medición de tiempos se hace de manera mucho más precisa. Esto se ha conseguido mediante el uso de una ISR que responde a las interrupciones disparadas en los flancos de bajada y subida del pin de datos. En cada interrupción se calcula el incremento de tiempo desde la interrupción anterior comparando el timestamp de ésta con el tiempo actual en ns, a continuación se actualiza el timestamp.

Aunque no es una solución perfecta, empleando este nuevo mecanismo en conjunto con el procesamiento de datos de varias medidas, se consigue reducir sustancialmente las situaciones de fallo de lectura del sensor.

#### 4.1.2. Delay de la pantalla

Anteriormente se esperaban, haciendo uso de delay, 2 segundos entre escrituras al display, lo que bloqueaba la ejecución del problema. Actualmente se emplea una variable de timestamp en el struct que representa el display, la cual se actualiza cada vez que se escribe al display satisfactoriamente. Para cada escritura se comprueba si el tiempo transcurrido desde el timestamp al momento actual supera 2 segundos, en ese caso la escritura se realiza, si no la operación se reintenta en la siguiente iteración de las máquinas de estados que controlan el display.

#### 4.1.3. Aviso de errores

Gracias al nuevo tipo `SensorValueType`, si durante una medida de un sensor ocurre algún problema, se almacenará un valor de error en el búfer circular de ese sensor. Por lo que, si en el procesamiento de las medidas no hay suficientes valores válidos en ese búfer circular para hacer un cálculo, como valor procesado se tendrá un error. Así, a la hora de mostrar los datos por pantalla, se mostrará un mensaje de error junto al sensor que tiene problemas y el usuario podrá actuar en consecuencia.

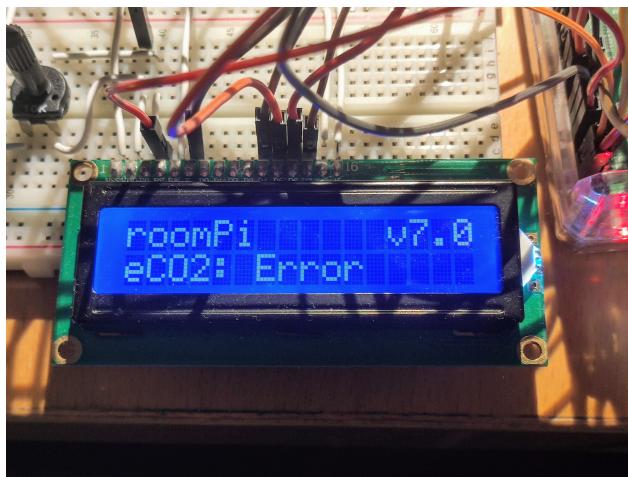


Figura 18: Situación de error de sensor, se ha aprovechado un sensor de CO2 defectuoso que tuvimos anteriormente

#### 4.1.4. Arranque automático del sistema

Anteriormente, para iniciar el sistema era necesario arrancar el ejecutable manualmente desde la terminal. En la versión actual, el sistema arranca automáticamente al iniciarse el SO Raspbian. Para ello se ha creado un servicio en `systemd` que ejecuta el archivo binario compilado tan pronto como haya una conexión de Internet y el servicio Docker haya arrancado.

```
pi@adelina: ~
● roompi.service - RoomPi System Service
   Loaded: loaded (/lib/systemd/system/roompi.service; disabled; vendor preset: enabled)
   Active: active (running) since Fri 2021-05-21 01:16:17 CEST; 11s ago
     Main PID: 21185 (roompi-bin)
       Tasks: 2 (limit: 2096)
      CGroup: /system.slice/roompi.service
              └─21185 /home/pi/roompi-bin

May 21 01:16:17 adelina systemd[1]: Started RoomPi System Service.
pi@adelina:~ $ |
```

Figura 19: Servicio RoomPi ejecutándose correctamente desde systemd

#### 4.1.5. Inicio lento

Este aspecto supuso un problema al principio, y de hecho se solucionó forzando una medida inicial al arrancar el sistema y así no tener que esperar a que los diferentes timers activen sus correspondientes flags y se realicen medidas. Sin embargo, tras la adición del sensor de CO<sub>2</sub> que necesita un tiempo inicial de calibrado y los búferes circulares que necesitan ser rellenados con algunos valores iniciales antes de poder procesarlos, este inicio lento se ha hecho necesario. Por tanto, lo que ocurre ahora es que durante este proceso se le informa al usuario en la pantalla con el mensaje de “inimando...” y “calibrando...”.

#### 4.1.6. Diseño FSM LCD avisos

Somos conscientes del diseño tan caótico de esta máquina de estados, sin embargo, no hemos conseguido ingeniar una solución más optimizada. No obstante, es una solución que es completamente funcional.

#### 4.1.7. Sensor de temperatura y humedad DHT22

Inicialmente teníamos en mente sustituir el sensor de temperatura y humedad DHT11 que estábamos utilizando por su siguiente versión, el sensor DHT22. Sin embargo, al ser este un sistema diseñado para usarse en interiores, no necesitamos medir temperaturas bajo cero, y es que la principal razón por la que habíamos pensado en usar este segundo sensor era el mayor rango de valores que permite medir.

#### 4.1.8. Sonómetro

Se ha experimentado con circuitos compuestos por micrófonos electret de condensación con amplificadores, filtros y procesamiento, pero las medidas que daban eran poco fiables y no realistas. Mientras tanto, los sonómetros IC que están a la venta para integrar en proyectos tienen un precio muy elevado, por lo que decidimos prescindir de este sensor.

## 4.2. Problemas y mejoras adicionales

### 4.2.1. Subsistema de medida

Anteriormente, el subsistema de medida se componía únicamente de los dispositivos sensores y de una sola FSM, cuyo diagrama se adjunta a continuación:

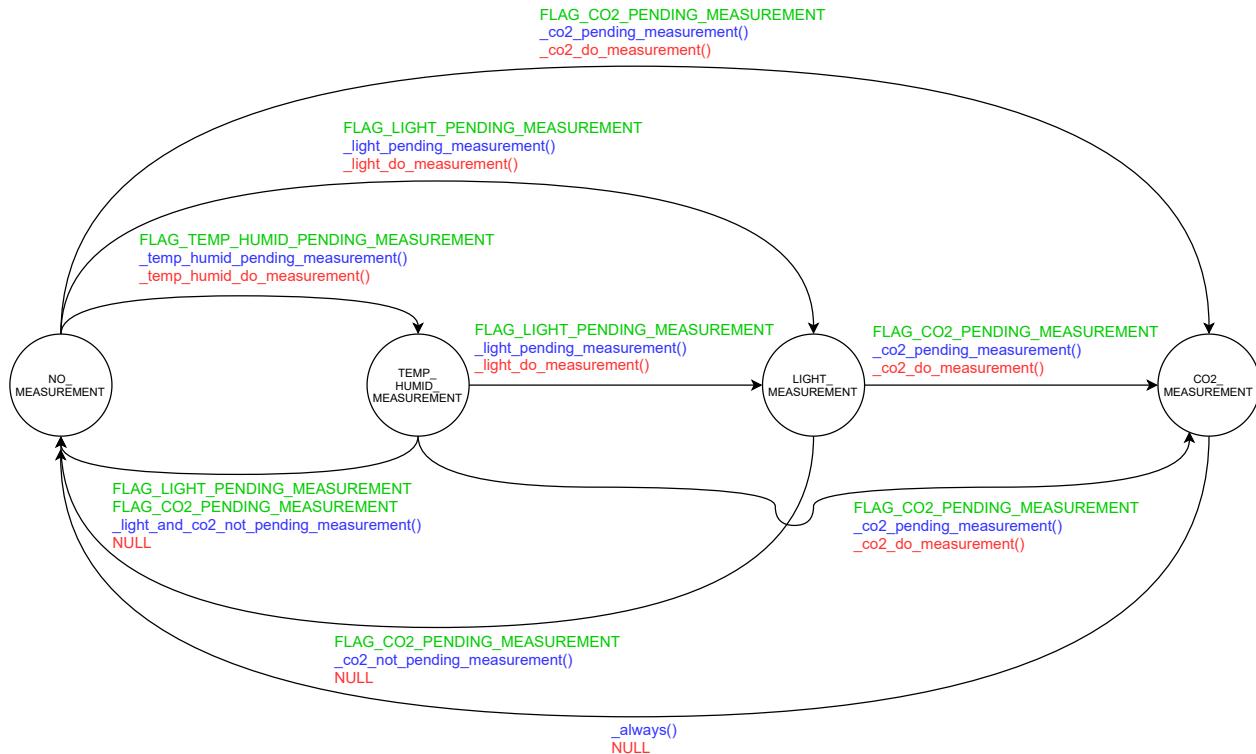


Figura 20: Antigua FSM MeasurementCtrl

Lo que primero salta a la vista es el diseño tan caótico que tenía, y es que era muy difícil de escalar, por lo que al añadir el sensor de CO<sub>2</sub> quedó un diagrama muy complicado. Sin embargo, para la forma de medir que realizábamos por aquel entonces, esta máquina de estados era completamente funcional. El principal problema aconteció cuando intentamos implementar el sistema de medidas múltiples actual, para así poder procesarlas y obtener un valor medio como resultado. Fue entonces cuando decidimos rediseñar todo el subsistema de medidas para acabar resultando el diseño actual, con una máquina de estados por sensor y otra que procesa las medidas de estos. Además, con este nuevo diseño nos ha sido posible también tener períodos de medida diferentes para cada sensor.

#### 4.2.2. Corriente insuficiente

Al añadir el sensor de CO<sub>2</sub> después del desarrollo realizado para el hito se presentó un problema: los sensores estaban conectados al pin 3V3 y según las especificaciones de la RPi nunca se puede extraer más de 16 mA por cualquier pin GPIO de la Raspberry Pi.

Si se conecta dicho sensor, ya superamos el límite cuando está realizando medidas pues en ese estado consume en torno a 26 mA, generando lecturas erróneas. Asimismo se manifestaron problemas de inestabilidad de corriente anteriores a la adición del sensor y nos percatamos de la insuficiente protección del IC de la RPi frente a posibles situaciones de excesiva disipación de potencia así como una general inestabilidad en las tensiones proporcionadas a lo largo del circuito.

Se ha realizado un análisis de consumo de corriente en una situación de máximo consumo para los dispositivos que necesitan una cantidad no despreciable de corriente:

$$20 \text{ mA (LEDs y SN74HC595)} + 26 \text{ mA (CCS811)} + 1.5 \text{ mA (DHT11)} + 2.3 \text{ mA (LCD)} + 0.2 \text{ mA (BH1750)} = 50 \text{ mA}$$

Este valor está justo al límite del valor de corriente máxima que puede suministrar el pin de alimentación de 3V3: 50 mA.

Ante esta situación se ha decidido emplear un regulador de tensión para proporcionar la alimentación 3V3 a los distintos sensores. El regulador tiene como input el pin de 5V, que a su vez está enganchado directamente a la fuente de alimentación USB. La condición en esta situación es que la suma de corriente consumida por los dispositivos más 700 mA que consume de media la placa RPi sean suficientemente suministrada por la fuente de alimentación del sistema. La fuente que empleamos para el proyecto proporciona 5V a 2.4 A, proporcionando un margen más que suficiente para nuestro proyecto.

Para obtener la tensión de salida deseada se ha dispuesto el regulador con determinadas resistencias que pueden consultarse en el esquemático del proyecto.

Para detalles sobre la ecuación de cálculo de las resistencias, consultar el datasheet del regulador de tensión.

#### 4.2.3. Conexionado

Anteriormente, teníamos un conexionado de los cables bastante caótico y, lo más importante, con gran probabilidad de malas conexiones. Actualmente, hemos empleado cable blanco unifilar para realizar el conexionado en lo relativo a la protoboard mientras que para las conexiones de la protoboard a pines de la Raspberry Pi se emplean cables DuPont macho-hembra.

### 4.3. Líneas de Continuación

#### 4.3.1. FSM LCD avisos

Como ya se ha comentado, aunque sea completamente funcional, esta máquina de estados es muy difícil de escalar. En el momento que se requiere añadir un sensor más, el diseño se hace aún más caótico.

#### 4.3.2. BBDD

Como mejora futura podríamos definir un sistema de políticas de retención de la base de datos, de tal forma que se realice un diezmado de los datos almacenados según su antigüedad.

#### 4.3.3. Histéresis

La implementación de un sistema de histéresis nos permitiría evitar rebotes por pequeñas caídas o subidas de magnitudes en los bordes de los umbrales.

#### 4.3.4. Mayor ajuste y personalización

El usuario podría tener aún más acceso a ciertos parámetros, para así realizar un sistema todavía más configurable y versátil. Algunos de estos parámetros podrían ser los rangos de histéresis de las medidas, parámetros intrínsecos de los sensores como su sensibilidad y tiempo de muestreo o incluso poder calibrar los sensores bajo demanda.

#### 4.3.5. Sistema de notificación

Una mejora adicional sería la realización de un sistema de notificación de los avisos vía SMS, email o incluso mediante una aplicación móvil.

#### 4.3.6. Predicción de medidas futuras

Basada en un algoritmo de Machine Learning, empleando técnicas de regresión (p.ej. Lasso Regression) con la librería scikit-learn en Python.

## 5. Links de interés

- Carpeta con archivos adjuntos y vídeo-presentación - [https://upm365-my.sharepoint.com/:f/g/personal/marcos\\_gomezb\\_alumnos\\_upm\\_es/EqGdml0N9WBBnNn1me0-pjIBa-wj2g2h0XMDXE0Q?e=9grveu](https://upm365-my.sharepoint.com/:f/g/personal/marcos_gomezb_alumnos_upm_es/EqGdml0N9WBBnNn1me0-pjIBa-wj2g2h0XMDXE0Q?e=9grveu)

- Repositorio en GitHub del proyecto - <https://github.com/margobra8/RoomPi>