Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías Industriales

Desarrollo de unidad portátil de monitorización de apnea del sueño

Autor: Rosa Inés Chamorro Nieto

Tutor: Ascensión Zafra Cabeza

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

Sevilla, 2013

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Desarrollo de unidad portátil de monitorización de apnea del sueño

Autor:
Rosa Inés Chamorro Nieto

Tutor:
Ascensión Zafra Cabeza
Profesor titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Grado: Desarrollo de unidad portátil de monitorización de apnea del sueño

Autor: Rosa Inés Chamorro Nieto

Tutor: Ascensión Zafra Cabeza

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

Resumen

La apnea del sueño es uno de los trastornos del sueño más comunes y que, sin diagnosticar, puede llegar a tener graves consecuencias en los pacientes. Las pruebas diagnósticas en el domicilio, con dispositivos de tipo III y IV, han sido recomendadas en los últimos años con el fin de facilitar el diagnóstico precoz de la enfermedad.

En este trabajo se implementa un dispositivo de tipo IV utilizando los sensores de flujo de aire, ronquidos y posición del kit de MySignals para Arduino, que permite visualizar fácilmente los valores de los sensores en tiempo real.

El dispositivo se ha complementado con una aplicación de usuario desarrollada con Python que almacena el histórico de las lecturas en un fichero CSV y permite obtener un informe en formato PDF para facilitar su análisis.

Abstract

Sleep apnea is one of the most common sleep disorders, while also being potentially serious. At home testing, using level III and IV devices, has been lately recommended for the purpose of achieving early diagnosis.

This project implements a level IV device employing MySignals' airflow, snoring and body position sensors, and allows to have real time visualization of sensors' values.

In addition, a desktop application has been developed with Python. The application has two main functionalities: it stores the sensors' data in a CSV file and creates a PDF report from from values summarizing results.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

AF	Airflow. Sensor de flujo de aire
SN	Snoring. Sensor de ronquidos
BP	Body Position. Sensor de posición corporal

1 INTRODUCCIÓN

1.1 Descripción de la apnea

La apnea del sueño es un trastorno del sueño potencialmente grave que se caracteriza porque la respiración se interrumpe durante un mínimo de 10 segundos mientras se duerme. Pueden llegar a producirse más de 30 interrupciones a la hora. Hay tres tipos principales de apnea del sueño [1] [2]:

1. La **apnea del sueño obstructiva** es la más frecuente y ocurre cuando los músculos en la parte posterior de la garganta se relajan, lo que ocasiona que las vías aéreas se estrechen o se cierren al respirar. En este caso, está desaconsejado dormir boca arriba, ya que favorece el bloqueo de las vías resiratorias. El nivel de oxígeno en la sangre puede disminuir al no recibir suficiente aire. Cuando el cerebro lo detecta provoca un despertar breve para volver a abrir las vías aéreas.
2. La **apnea del sueño central** ocurre cuando el cerebro deja de transmitir las señales adecuadas a los músculos que controlan la respiración, es decir, no se hace esfuerzo para respirar durante un período breve de tiempo. En general, suele ser secundaria a algún tipo de enfermedad y los pacientes no suelen roncar.
3. La **apnea del sueño mixta**, también denominado síndrome de apnea del sueño compleja o apnea central del sueño emergente del tratamiento. Ocurre cuando alguien tiene apnea obstructiva del sueño y apnea central del sueño.

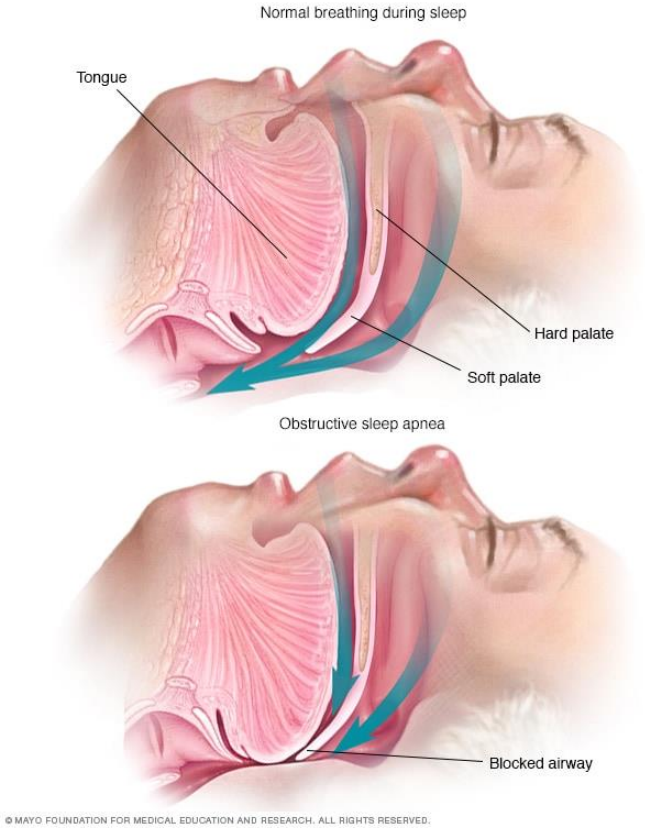


Figura 1-1. Apnea del sueño obstructiva [2]

Entre los síntomas más comunes de la apnea del sueño se incluyen:

4. Ronquidos fuertes
5. Episodios en los que se deja de respirar durante el sueño
6. Jadeos al respirar durante el sueño
7. Despertarse con la boca seca
8. Dolor de cabeza por la mañana
9. Despertarse súbitamente con un ronquido o resoplido
10. Sensación de sueño excesiva durante el día
11. Dificultad para prestar atención mientras estás despierto
12. Irritabilidad

En recientes estudios se estima que 636 millones de adultos de entre 30 y 69 años de todo el mundo tienen apnea del sueño obstructiva leve a severa y 425 millones de moderada a severa. La apnea del sueño sin

diagnosticar puede tener consecuencias negativas importantes sobre los pacientes. Entre otras, está relaciona con enfermedades cardiovasculares, ya que al interrumpirse la respiración durante un periodo de tiempo determinado aumenta la presión arterial y se sobrecarga el sistema cardiovascular para compensar la falta de oxígeno en los tejidos [3].

1.2 Dispositivos usados para su detección

El diagnóstico es fundamental para manejar la enfermedad de forma apropiada y requiere un estudio del sueño que se basa en un monitoreo nocturno de la respiración y otras funciones. La Academia Estadounidense de Medicina del Sueño describe 4 tipos de pruebas para el diagnóstico:

1. Polisomnografía estándar (I): Se realiza en un hospital o clínica especializada. Como mínimo requiere un electroencefalograma, un electrooculograma, un electromiógrafo para la barbilla, un electrocardiógrafo, un sensor de flujo de aire, un sensor de esfuerzo respiratorio y un sensor de saturación de oxígeno en sangre (SpO2). La posición corporal se debe documentar o medir objetivamente. El paciente está vigilado por un técnico experto que puede intervenir en caso de que sea necesario recolocar algún sensor. Aunque aporta más información, obliga a dormir fuera de casa, por lo que puede generar el "efecto laboratorio" con un sueño más superficial y menos representativo del sueño real.
2. Polisomnografía portátil completa (II): Es igual que el anterior, a excepción del electrocardiógrafo, que puede ser sustituido por un sensor de ritmo cardíaco. Además, no se requiere la presencia de un técnico en todos los casos.
3. Monitor de apnea portátil modificado (III): Al menos requiere registrar la respiración con al menos dos canales para el movimiento respiratorio o el movimiento respiratorio y el sensor de flujo de aire, un electorcardiógrafo o un sensor de ritmo cardíaco y un sensor de SpO2. Se requiere de un técnico para la preparación y en algunos casos también para intervenir si fuera necesario.
4. Registro continuo de bioparámetro único o dual (IV): Solo se registran una o dos variables fisiológicas y no requiere de intervención [4] [5].

En los últimos años se ha fomentado la realización de pruebas diagnósticas tipo III y IV en el domicilio para facilitar la detección temprana en el caso de adultos sin complicaciones que muestren síntomas relacionados de apnea obstructiva del sueño. Al contar con un mayor número de sensores, el uso de monitores tipo III es más complejo e intrusivo. Por su parte, los dispositivos tipo IV no tienen estos inconvenientes, pero aún no se conoce con exactitud el número y el tipo de señales más apropiado para utilizar en este caso.

Dos de los sensores más comunes en los dispositivos de tipo IV son el sensor de flujo de aire y el de SpO2. En un estudio de la Universidad de Valladolid, se concluyó que, aunque el uso independiente de la oximetría puede detectar episodios de apnea e hipopnea, sobre todo en casos graves, el uso conjunto de ambas señales aporta un aumento notable del rendimiento [6].

1.3 Motivación y objetivos

En este proyecto se va a desarrollar un aparato para monitorizar el sueño del paciente y detectar eventos de apnea e hipoapnea. Cuenta con un dispositivo portátil con pantalla que permite visualizar los valores de los sensores en tiempo real. Este dispositivo puede conectarse al ordenador para, utilizando la interfaz facilitada, capturar los datos de los sensores y almacenarlos en un archivo CSV.

Además, esta misma interfaz permite generar un informe en formato PDF a partir de los datos de este CSV, en el que se destacarán los periodos en los que la apnea supere el intervalo de tiempo determinado por el usuario en la interfaz.

De esta manera, se pretende crear un dispositivo simple y no invasivo, que facilite tanto al paciente como al médico el diagnóstico de los trastornos de apnea del sueño.

2 HARDWARE

2.1 MySignals HW

Para la implementación de este dispositivo se ha utilizado el kit MySignals HW de Libelium [7], que proporciona una plataforma para desarrollar dispositivos médicos y aplicaciones de eSalud.

El kit incluye 3 elementos principales: el escudo, la pantalla y los sensores.

2.1.1 Escudo

El escudo es la placa principal de MySygnals e incluye:

- Circuito de alimentación
- Conversor analógico digital de 10 bits
- Circuito de conexión de sensores
- Conectores para los sensores
- Conectores para módulo TFT y SD
- Módulo de Bluetooth
- Conectores para módulo de WiFi

La siguiente imagen muestra el escudo y el detalle de los conectores de los sensores. Los conectores recuadrados son aquellos que se corresponden con los sensores utilizados para el desarrollo de esta aplicación:

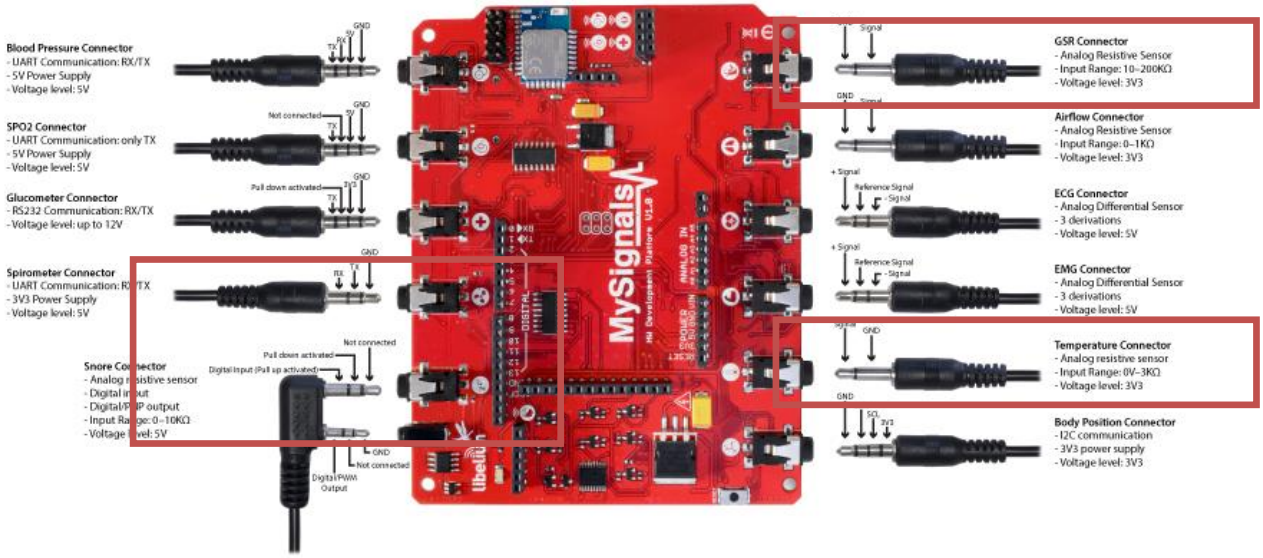


Figura 2-1. Escudo de MySignals y conectores de sensores [7]

2.1.2 Pantalla

El escudo es compatible con la pantalla de Adafruit modelo ILI9341. Es una pantalla TFT LCD, es decir, una pantalla de cristal líquido que utiliza la tecnología de transistor de película delgada, de 2,4 pulgadas con una resolución de 240x320 píxeles RGB. Además, incorpora una pantalla táctil resistiva, un lector de tarjetas SD y un controlador integrado con amortiguador RAM, para minimizar el trabajo realizado por el microcontrolador [8].

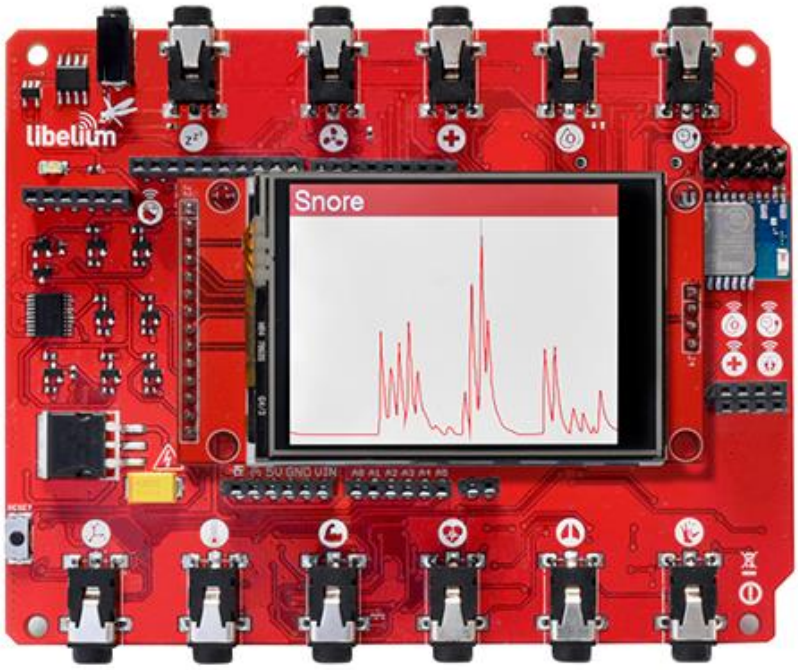


Figura 2-2. Escudo con pantalla conectada [7]

2.1.3 Sensores

Por último, el kit cuenta en total con 11 sensores, de los cuáles, tres se utilizan para crear el monitor:

5. Sensor de resistencia galvánica de la piel
6. **Sensor de flujo de aire**
7. Electrocardiógrafo
8. Electromiógrafo
9. Termómetro
10. **Sensor de posición corporal**
11. **Sensor de ronquido + Generador de pitido + Botón de alerta**
12. Espirómetro
13. Glucómetro
14. Pulsioxímetro
15. Tensiómetro

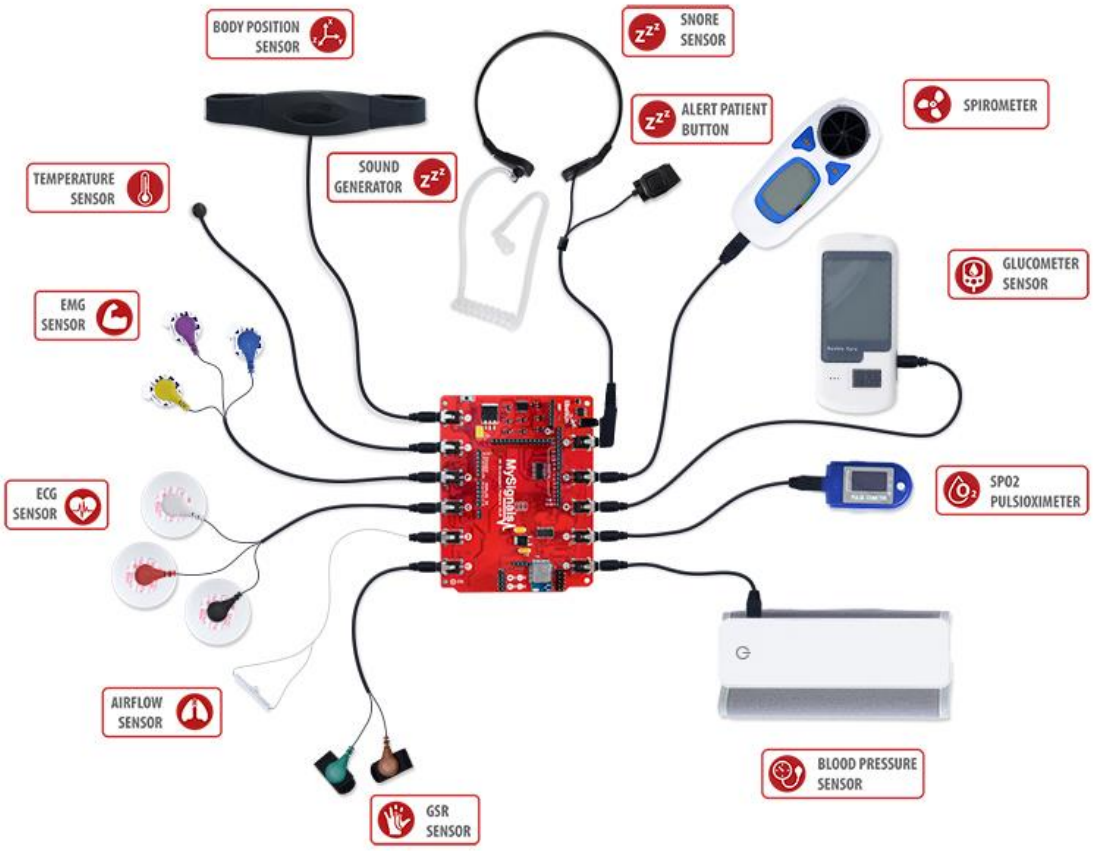


Figura 2-3. Sensores y escudo de Mysignals [7]

En los siguientes apartados se describen los sensores utilizados para implementar el monitor.

2.1.3.1 Sensor de flujo de aire

2.1.3.1.1 Funcionamiento

El sensor de flujo de aire es uno de los sensores más comunes para detectar los episodios de apnea e hipoapnea. Consta de 2 cánulas que se colocan bajo la nariz y otra más larga en dirección a la boca. En su interior contienen unos sensores de termopar que permiten medir el flujo de aire oral y nasal, así como la temperatura del aire.

Figura 2-4. Sensor de flujo de aire [7]

El sensor se completa con dos cables flexibles que se colocan detrás de las orejas. Para situarlo en la posición adecuada, debe hacerse como se muestra en la siguiente imagen:

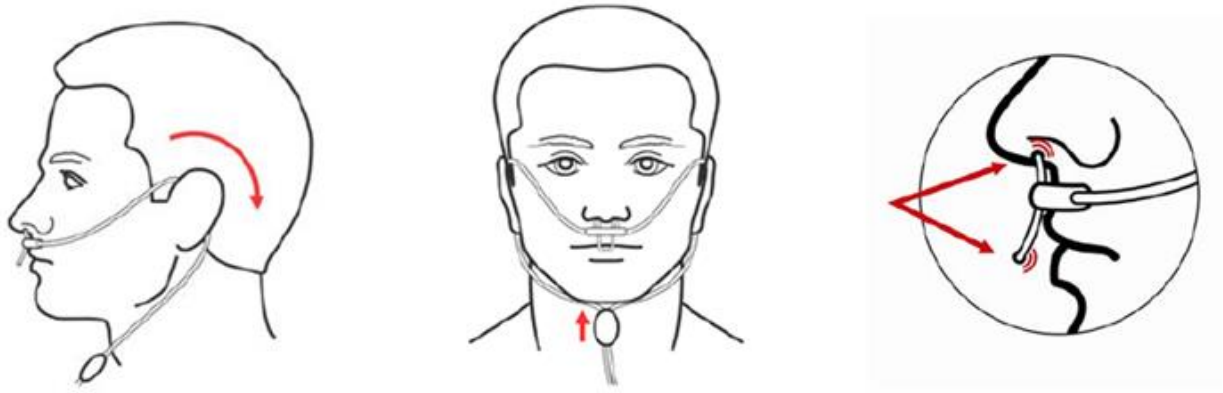


Figura 2-5. Colocación del sensor de flujo de aire [7]

2.1.3.1.2 Rango de Valores

Tabla 2-1. Medidas del sensor de flujo de aire

PARÁMETRO	UNIDADES	RANGO
Ritmo Respiratorio	ppm (picos por minutos)	0 – 60 ppm
Intensidad de la respiración	Voltios	0 – 3,3 V
	Digital	0 – 1023

2.1.3.2 Sensor de ronquidos

2.1.3.2.1 Funcionamiento

Los ronquidos son uno de los síntomas más comunes de la apnea del sueño. Este sensor se coloca en contacto con el cuello para recoger las vibraciones de la garganta y las convierte en una señal analógica. Este método cancela en gran parte el ruido de fondo.

Como complemento, el sensor incluye un botón y un auricular para reproducir una alarma.

Figura 2-6. Sensor de ronquidos [7]

2.1.3.2.2 Rango de valores

Tabla 2-2. Medidas del sensor de ronquidos

PARÁMETRO	UNIDADES	RANGO
Ritmo de ronquido	rpm (ronquidos por minutos)	0 – 60 rpm
Intensidad de la respiración	Voltios	0 – 5 V
	Digital	0 – 1023

2.1.3.3 Sensor de posición

2.1.3.3.1 Funcionamiento

Como se indicó en la introducción, dormir en posición supina (tendido sobre la espalda) puede contribuir al bloqueo de las vías respiratorias, por lo que conocer la posición del paciente es interesante a la hora de analizar los episodios de apnea. El sensor de posición utiliza un acelerómetro de triple eje para clasificar la posición del paciente en 5 posiciones corporales diferentes. El sensor se coloca con una banda elástica alrededor del tronco del paciente.

Figura 2-7. Sensor de posición [7]

2.1.3.3.2 Rango de valores

Tabla 2-3. Medidas del sensor de posición

PARÁMETRO	UNIDADES	RANGO
Posición corporal	Número entero representando una posición corporal	1 – 6

Tabla 2-4 Equivalencia valor-posición

VALOR	POSICIÓN
1	Decúbito prono o ventral
2	Decúbito lateral izquierdo
3	Decúbito lateral derecho
4	Decúbito supino

5	De pie o sentado
6	Indefinido

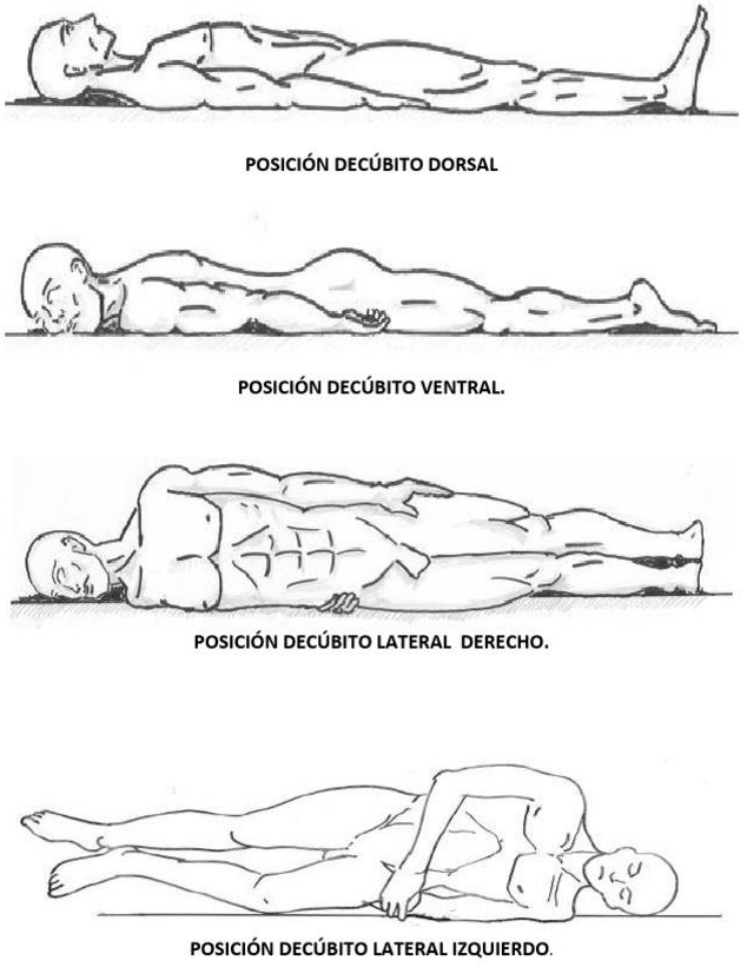


Figura 2-8. Posiciones corporales [9]

2.2 Arduino

Para poder utilizar MySignals HW es necesario conectarlo a un Arduino UNO [10].

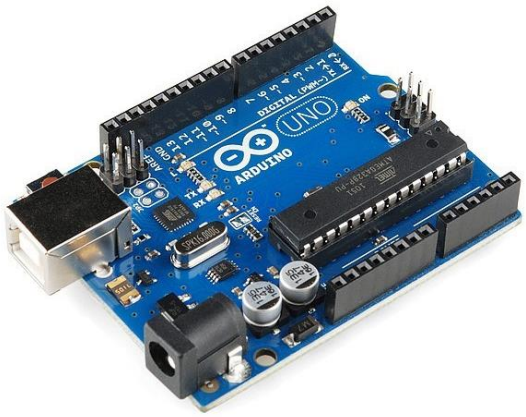


Figura 2-9. Arduino UNO [11]

Tabla 2-5. Características del Arduino UNO

Microcontrolador	ATmega328
Voltaje de operación	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (límites)	6-20V
Pines de E/S digitales	14 (de los cuales 6 proporcionan salida PWM)
Pines de entrada analógica	6
Corriente DC por pin de E/S	40 mA
Corriente DC para 3.3V Pin	50 mA
Memoria Flash	32 KB de los cuales 0,5 KB utilizados por el bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad de reloj	16 MHz

3 DESARROLLO DEL MONITOR DE APNEA

3.1 Descripción

El monitor se basa en dos elementos principales: un dispositivo portátil desarrollado en Arduino para la lectura de los sensores y visualización de los valores en tiempo real, y un programa informático desarrollado con Python para la captura de los datos en formato CSV y análisis de este histórico.

De este modo, el primer dispositivo toma como entrada los datos de los sensores y, simultáneamente, los muestra por pantalla y los envía al puerto serie del Arduino. El segundo módulo puede dividirse a su vez en sus dos funciones principales. La primera, lee los datos que recibe por el puerto serie indicado por el usuario a través de la interfaz y crea un fichero CSV como salida. La segunda, toma estos ficheros como entradas y junto con algunos datos adicionales proporcionados por el usuario, genera un informe en formato PDF.

El siguiente esquema muestra el flujo de datos de la aplicación:

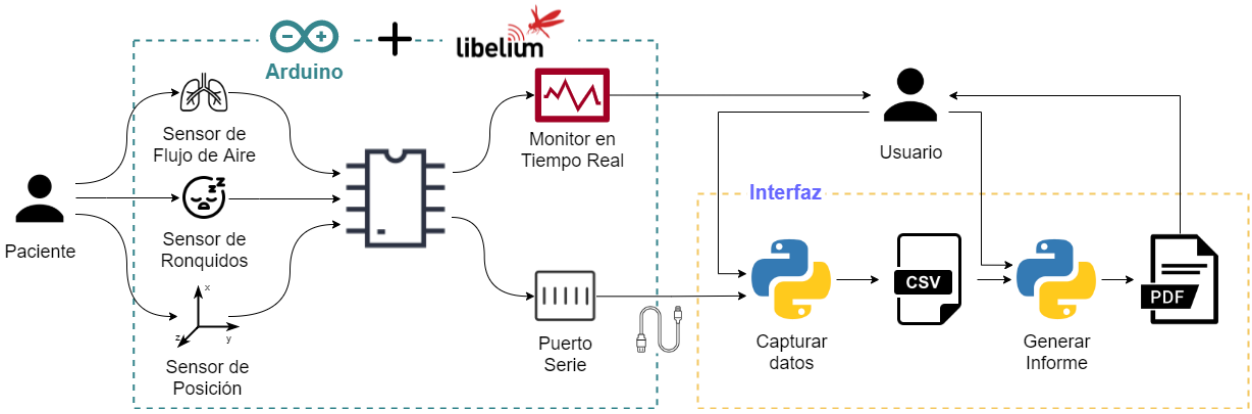


Figura 3-1. Diagrama de flujo de datos

3.2 Funciones

Las funciones que proporciona este dispositivo son tres.

A. Visualización de los datos en tiempo real. Esta función se implementa gracias a la pantalla que se conecta al Arduino. En ella se muestra tanto el valor actual de cada uno de los sensores usados, como una gráfica que recoge el histórico de, aproximadamente, un minuto. En la figura 3-2 puede verse un ejemplo de esta pantalla.

B. Conexión del Arduino a un ordenador, gracias a la interfaz que se muestra en la figura 3-6:

   B.1. Generación de un histórico en formato CSV. Las lecturas de los sensores pueden almacenarse en un fichero CSV para su posterior análisis. Puede verse un ejemplo de los ficheros generados en la figura 3-4.

   B.2. Generación de un informe en formato PDF. A partir de los ficheros generados, se puede obtener fácilmente un informe que identifica los episodios de apnea y permite consultar los valores recogidos por cada sensor. En las figuras 3-5 y 3-6 se muestra un ejemplo del PDF generado.

3.3 Entradas

Las entradas, según los módulos descritos anteriormente, son:

A. Dispositivo portátil

A.1.1. Sensores de flujo de aire, de ronquidos y de posición conectados con el escudo tal y como se muestra en la Figura 2-1

B. Interfaz del ordenador

B.1. Capturador de datos
B.1.1. Número del puerto serie al que conectarse, introducido por el usuario a través de la interfaz
B.1.2. Lectura del puerto serie del Arduino, con los valores de los sensores. Para la lectura del puerto serie es necesario conectar el Arduino al ordenador por USB

B.2. Generador de informe
B.2.1. Fichero CSV generado por el módulo B.1. La ruta en la que se encuentra el fichero puede ser proporcionada por el usuario a través de la interfaz
B.2.2. Nombre del paciente
B.2.3. Apellidos del paciente
B.2.4. Umbral de apnea en segundos

3.4 Salidas

Del mismo modo, las salidas, según los módulos descritos, son:

A. Dispositivo portátil

A.1. Pantalla del dispositivo, que permite visualizar:
A.1.1. Valor de los sensores en tiempo real
A.1.2. Etiqueta correspondiente al valor actual de posición
A.1.3. Gráficas de histórico

A continuación, se muestra un ejemplo de esta pantalla:

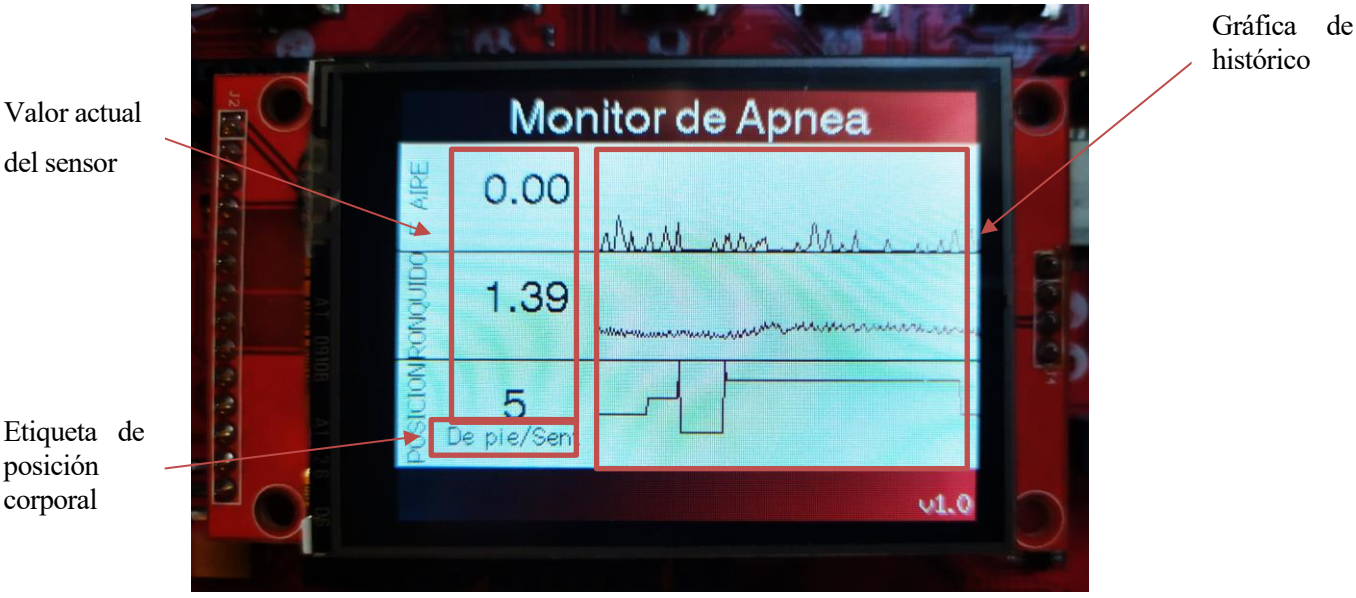


Figura 3-2. Interfaz portátil

A.2. Puerto serie

A.2.1. Los datos de los sensores también se envían al puerto serie del Arduino. Cada vez que se actualiza la lectura de los sensores en la pantalla, se manda una nueva línea con la estructura: *[Valor digital del flujo de aire]*;*[Voltaje flujo de aire]*;*[Valor digital ronquidos]*;*[Voltaje ronquidos]*;*[Valor de posición]*

Si lo vemos a través del monitor serie de la propia aplicación de Arduino:

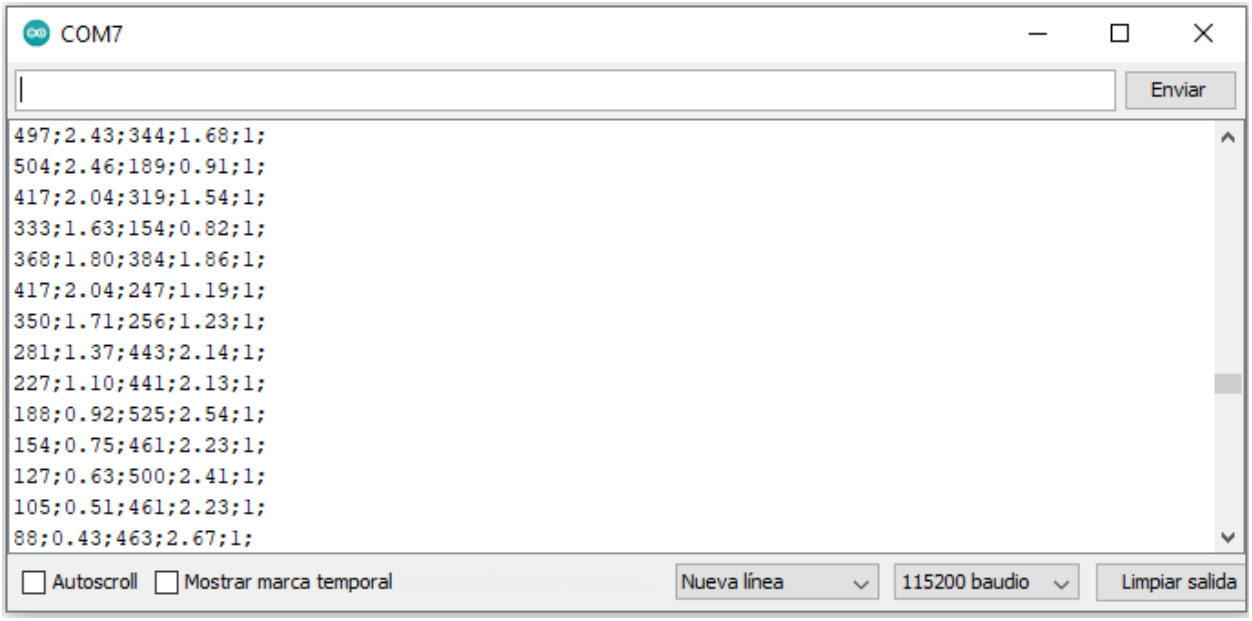


Figura 3-3. Monitor serie de Arduino

B. Interfaz del ordenador

B.1. Capturador de datos

B.1.1. Datos en fichero CSV. En la construcción del fichero, se añade una columna con la fecha y hora del momento en el que se está guardando la lectura. Cada una de las filas del fichero queda, por tanto, con el formato: *[Fecha-hora]*;*[Lectura Puerto Serie]*.

O lo que es lo mismo: *[Fecha-hora]*;*[Valor digital del flujo de aire]*;*[Voltaje flujo de aire]*;*[Valor digital ronquidos]*;*[Voltaje ronquidos]*;*[Valor de posición]*.

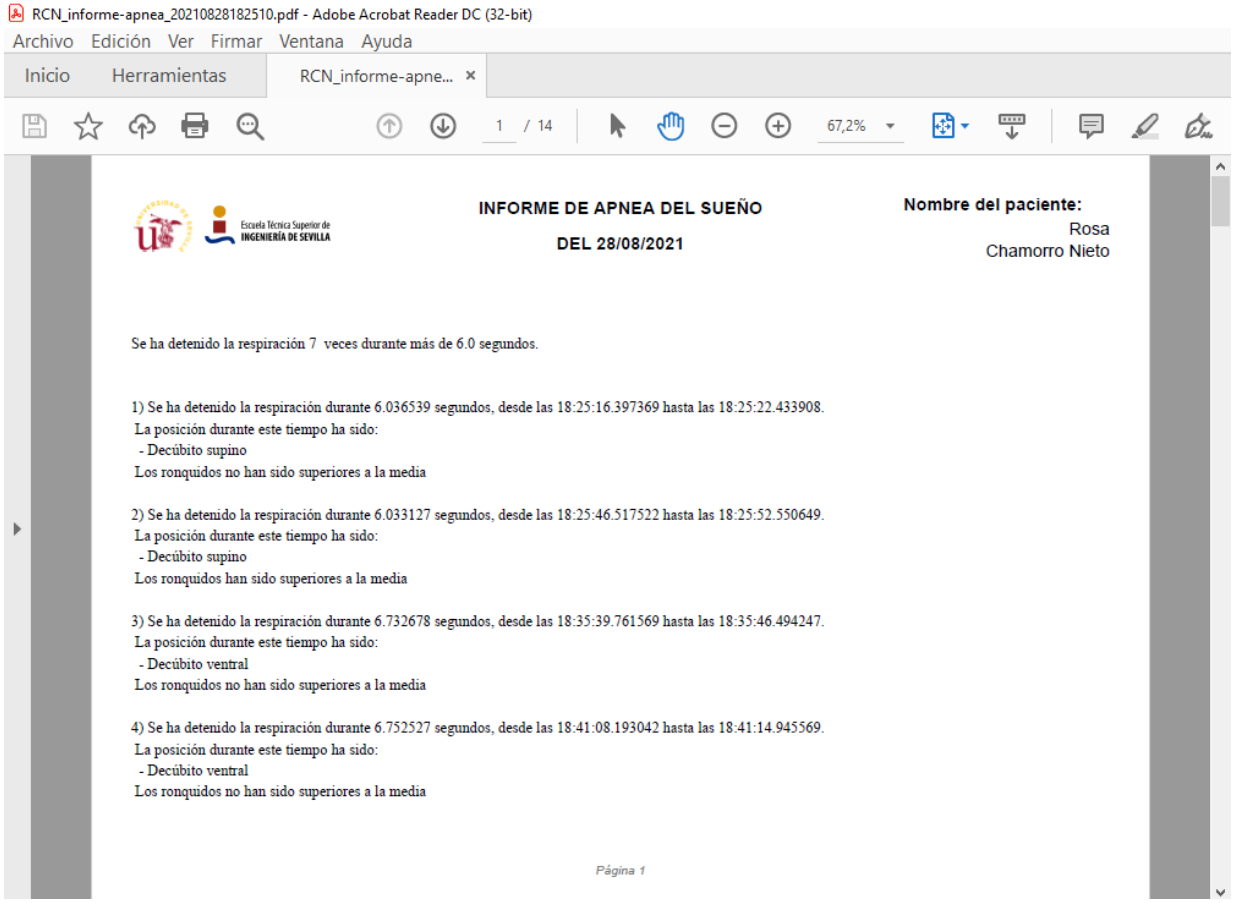
Figura 3-4. Ejemplo de CSV generado

La ruta en la que se crea el fichero se muestra al usuario a través de la interfaz. El nombre del fichero consta del prefijo “monitor-apnea_” seguido de la fecha y hora, hasta los segundos, en la que se ha creado el fichero, para evitar sobreescribirlos.

B.2. Generador de informe

B.2.1. Informe en formato PDF. Consta de dos partes. La primera, un resumen de los intervalos de tiempo en los que se ha superado el umbral de apnea. La segunda, unas gráficas de todos los datos extraídos en las que, además, se resaltan los intervalos anteriores. Todas las páginas tienen una cabecera con el logotipo de la Universidad de Sevilla y de la Escuela Técnica Superior de Ingeniería, la fecha de los datos que contiene el informe y nombres y apellidos del paciente.

La ruta en la que se crea este fichero se muestra al usuario a través de la interfaz. El nombre del fichero se construye: *[Iniciales del paciente]* + _informe_apnea_ + *[Fecha del csv con el que se genera el informe]*


RCN_informe-apnea_20210828182510.pdf - Adobe Acrobat Reader DC (32-bit)

Archivo Edición Ver Firmar Ventana Ayuda

Inicio Herramientas RCN_informe-apne... ×

1 / 14 67,2%

Escuela Técnica Superior de INGENIERÍA DE SEVILLA

INFORME DE APNEA DEL SUEÑO

DEL 28/08/2021

Nombre del paciente:
Rosa
Chamorro Nieto

Se ha detenido la respiración 7 veces durante más de 6.0 segundos.

1) Se ha detenido la respiración durante 6.036539 segundos, desde las 18:25:16.397369 hasta las 18:25:22.433908.
La posición durante este tiempo ha sido:
- Decúbito supino
Los ronquidos no han sido superiores a la media

2) Se ha detenido la respiración durante 6.033127 segundos, desde las 18:25:46.517522 hasta las 18:25:52.550649.
La posición durante este tiempo ha sido:
- Decúbito supino
Los ronquidos han sido superiores a la media

3) Se ha detenido la respiración durante 6.732678 segundos, desde las 18:35:39.761569 hasta las 18:35:46.494247.
La posición durante este tiempo ha sido:
- Decúbito ventral
Los ronquidos no han sido superiores a la media

4) Se ha detenido la respiración durante 6.752527 segundos, desde las 18:41:08.193042 hasta las 18:41:14.945569.
La posición durante este tiempo ha sido:
- Decúbito ventral
Los ronquidos no han sido superiores a la media

Página 1

Figura 3-5. Primera parte del informe. Intervalos de apnea

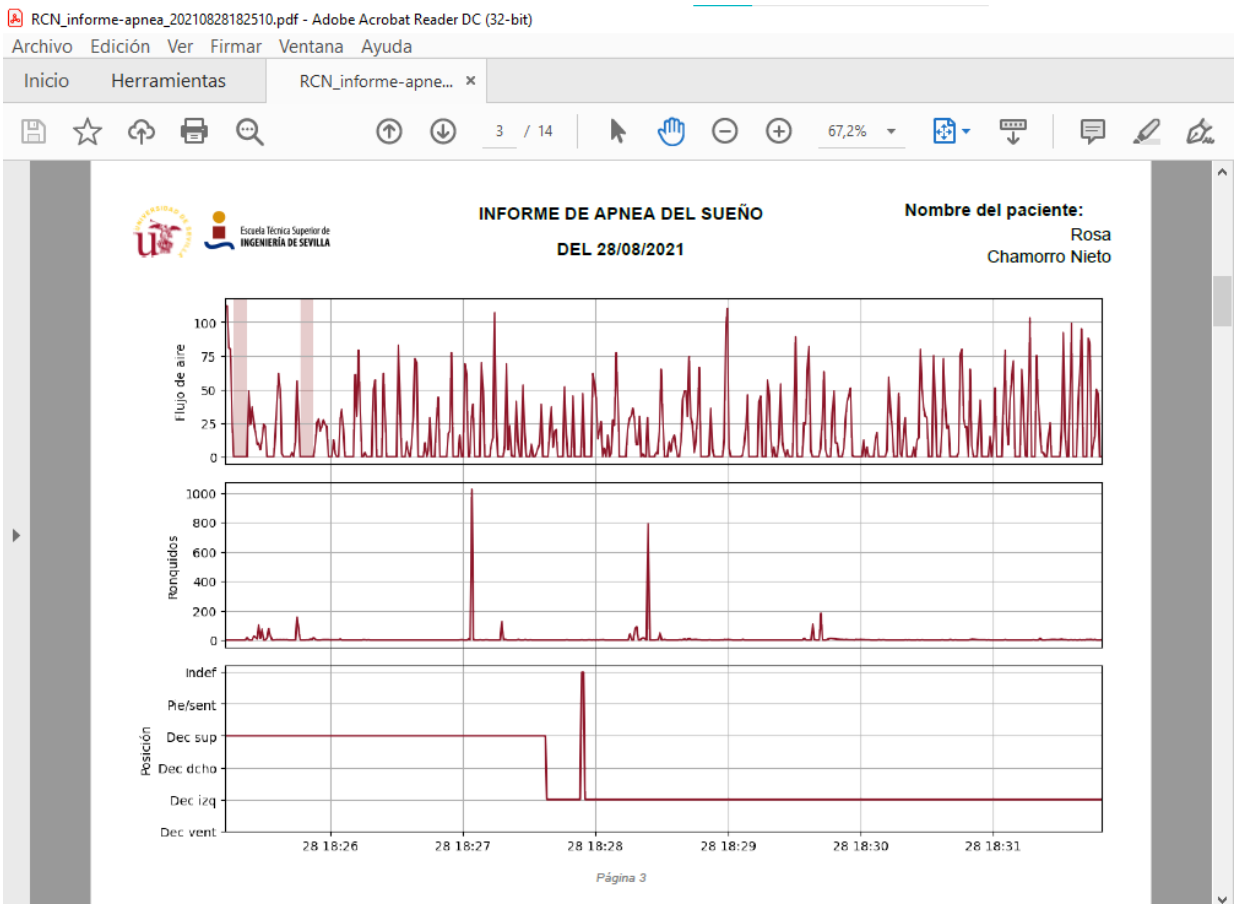


Figura 3-6. Segunda parte del informe. Gráficas de valores

En la siguiente imagen pueden verse los campos de entrada y salida de la interfaz:

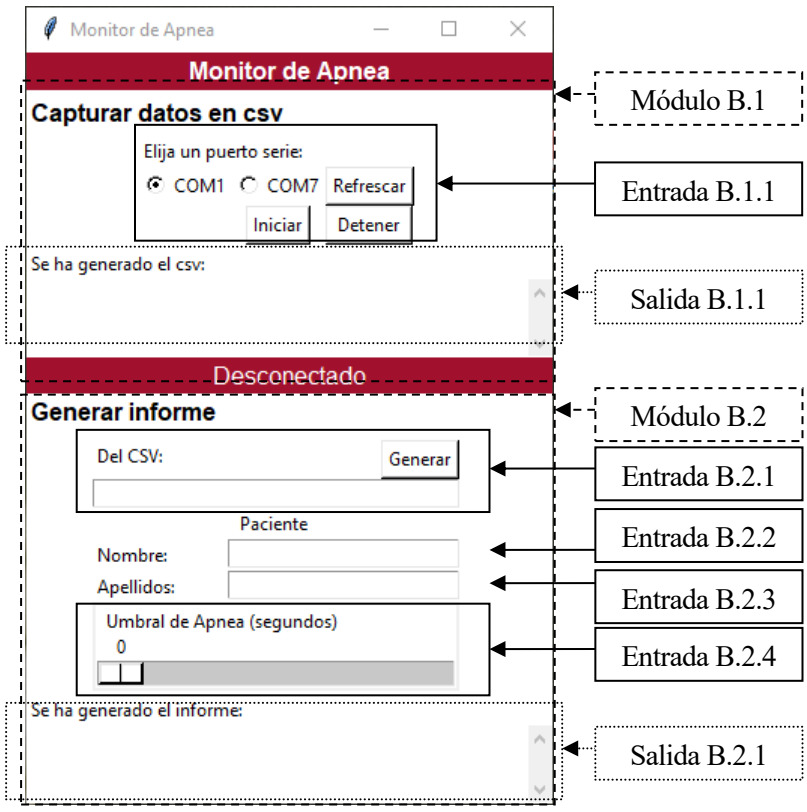


Figura 3-7. Interfaz del ordenador

4 SOFTWARE

Este proyecto emplea dos lenguajes de programación diferentes: C++ para Arduino y Python.

4.1 Interfaz portátil

Dado que la interfaz portátil está implementada en Arduino, el lenguaje de programación que utiliza es una adaptación de C++ a dicha plataforma. La estructura básica de un programa en Arduino divide el código en dos funciones: `setup()` y `loop()`. La primera, se ejecuta solo una vez al inicio de la ejecución y, generalmente, contiene las líneas de configuración. Después, se ejecuta la función `loop()` de manera continuada para actualizar las entradas y salidas.

De forma resumida, en el código desarrollado para este dispositivo se distinguen los siguientes puntos:

1. Se importan las librerías.
2. Se declaran las constantes y variables globales.
3. Función de `setup()`:
    a. Se inicializan el puerto serie, la pantalla y MySignals.
    b. Se pintan en la pantalla los elementos estáticos.
4. Función `loop()`:
    a. Se declaran las variables locales.
    b. Se actualiza la lectura de los sensores.
    c. Se actualiza la pantalla TFT y se envía la lectura al puerto serie.
5. Función `drawGraph()`: es invocada dentro de la función `loop()` y contiene el código necesario para actualizar cada una de las gráficas.

A continuación, se describen con más detalle.

4.1.1 Librerías

El programa utiliza dos librerías. La primera, `Adafruit_ILI9341_AS.h`, específica para el manejo del modelo de pantalla utilizado en este proyecto. Y la otra, `MySignals.h`, para tomar la lectura de los sensores.

Estas librerías son proporcionadas por MySignals en el siguiente enlace:

https://www.cooking-hacks.com/media/cooking/images/documentation/mysignals_hardware/MySignals_HW_SDK_V2.0.2.zip

```
// Importar librerías
#include <Adafruit_ILI9341_AS.h>
#include <MySignals.h>
```

4.1.2 Constantes y variables globales

Se declaran 8 constantes necesarias para delimitar las posiciones de cada parte de la visualización. La pantalla tiene una resolución de 240x320 píxeles, con los ejes X e Y orientados según se indica en la Figura 4-1. Son necesarias las siguientes constantes, que se usarán posteriormente a la hora de implementar el gráfico:

- `graph_left_extrem` y `graph_right_extrem`: Delimitan los extremos en los que comienzan y acaban las gráficas en el eje horizontal (eje X). En el diseño de este dispositivo, se ha dejado un margen a la izquierda para añadir una etiqueta que indique el nombre del sensor en cada caso y el

valor instantáneo de la lectura, por lo que las gráficas comienzan en el píxel x=110 y terminan, al final de la pantalla, en x=320.

- `AF_high_extrem`, `SN_high_extrem` y `BP_high_extrem`: Son los extremos superiores en el sentido vertical (eje Y) de cada sección, para los sensores de Flujo de Aire, Ronquidos y Posición corporal, respectivamente.
- `AF_low_extrem`, `SN_low_extrem` y `BP_low_extrem`: Son los extremos inferiores en el eje Y de cada sección, para los sensores de Flujo de Aire, Ronquidos y Posición corporal, respectivamente.

Se puede apreciar que entre el límite inferior de una sección y el superior del siguiente se deja una línea de un píxel, que es la que se utilizará para marcar la separación. Por ejemplo, AF_low_extrem=89 y SN_high_extrem=91, por tanto, en y=90 se encuentra la línea horizontal que separa la primera sección, de flujo de aire, de la segunda, la del sensor de ronquidos. Lo mismo ocurre con la línea que separa la sección de SN de la de BP en y=150.

Además, el espacio que queda para cada sección son 58 píxeles.

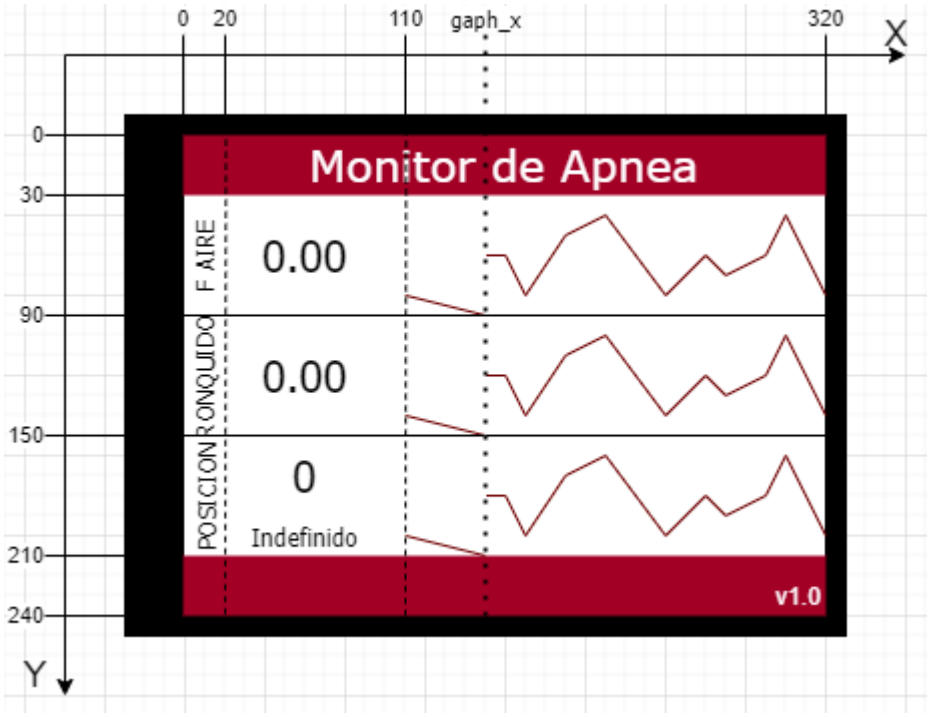


Figura 4-1. Ejes de la intefaz portátil para la orientación 3

En cuanto a las variables globales, se declaran:

- `tft`: Variable de tipo `Adafruit_ILI9341_AS` que se utilizará para definir los aspectos gráficos de la unidad.
- `miversion`: vector de caracteres que contiene la versión del programa para representarlo en la interfaz.
- `AF_prevRead`, `SN_prevRead`, `BP_prevRead`: valor escalado de la lectura anterior sobre el eje Y de los sensores de flujo de aire, ronquidos y posición corporal, respectivamente.
- `graph_x`: posición en el eje X en la que se encuentra la gráfica, que se inicializa al extremo izquierdo de la gráfica. En la figura 4-1 se ha representado un ejemplo.

```
// Constantes globales para límites de gráficas
```

```
#define graph_left_extrem 110
#define graph_right_extrem 320
// Límites superiores
#define AF_high_extrem 31
#define SN_high_extrem 91 //diff 58
#define BP_high_extrem 151
// Límites inferiores
#define AF_low_extrem 89
#define SN_low_extrem 149
#define BP_low_extrem 209

// Variables globales
char miversion[] = "v1.0";
// Instancia para la pantalla TFT
Adafruit_ILI9341_AS tft = Adafruit_ILI9341_AS(TFT_CS, TFT_DC);
// Posición actual de las gráficas en el eje x
uint16_t graph_x = graph_left_extrem;
// Posición en el eje y de cada gráfica en la iteración anterior
uint16_t AF_prevRead;
uint16_t SN_prevRead;
uint8_t BP_prevRead;
```

4.1.3 Función setup()

Como se indicó al inicio de este capítulo, esta función contiene la configuración del dispositivo. En ella:

1. Se inicializa el puerto serie a 115200 baudios.
2. Se inicializa la librería de MySignals con el método `begin()`, así como el sensor de ronquidos, con `initSnore()`.
3. Se inicializa la pantalla TFT con el método `init()`.
4. Se limpia la pantalla, rellenándola de blanco con el método `fillScreen(ILI9341_WHITE)`.
5. Usando la variable `tft`, se dibujan las partes estáticas de la interfaz:
    a. En primer lugar, con la pantalla orientada verticalmente, gracias a `setRotation(2)`, y con el color de la fuente en negro, se escriben las etiquetas de la parte derecha de la interfaz, que indican el nombre del sensor representado en cada sección. Para ello, se usa tres veces el método `drawCentreString()` que recibe como entradas:
        i. El texto que se quiere imprimir.
        ii. La posición en el eje X en la que se encuentra el centro del texto.
        iii. Distancia del margen superior de la pantalla a la parte de superior del texto.
        iv. La fuente que se quiere usar. Para estas etiquetas se usa la fuente pequeña, que en la librería de la pantalla tiene asignado el valor 2.
    b. Después, se orienta la pantalla en formato apaisado con `setRotation(3)` y se añaden la cabecera y el pie de interfaz, así como las líneas horizontales que dividen las secciones.
        i. Para la cabecera, se dibuja un rectángulo que comienza en el punto x=0, y=0, con 320 píxeles de ancho y 30 de alto, del color definido en la librería con la constante `ILI9341_MAROON`.
        ii. En el caso del pie, se utiliza un rectángulo de las mismas características que el anterior que comienza en el punto x=0, y=210.

iii. Con el color del texto en blanco, se escribe el nombre de la pantalla, “Monitor de Apnea” con la función `drawCentreString()` que se ha explicado en el punto anterior, esta vez usando la fuente mediana, que se ha definido como la número 4.

iv. En el pie de la pantalla se escribe la versión del programa, en la esquina derecha, para lo cual se usa la función `drawRightString()`, similar a la anterior, pero en la que la x representa la posición del final del texto, en lugar de la del centro.

v. Por último, se dibujan dos líneas horizontales con el método `drawFastHLine()`. La primera, comienza en el punto x=0, y=90 y la segunda en x=0, y=150. Ambas miden 320 píxeles de longitud y son de color negro.

```
void setup() {
  // Inicialización del puerto serie a 115200 baudios
  Serial.begin(115200);
  // Inicialización de MySignal y del sensor de ronquidos
  MySignals.begin();
  MySignals.initSnore();
  // Inicialización de la pantalla
  tft.init();
  tft.setRotation(2);
  tft.fillScreen(ILI9341_WHITE); // Se limpia la pantalla
  // Se escriben las etiquetas de los sensores
  tft.setTextColor(ILI9341_BLACK);
  tft.drawCentreString("POSICION",60,2,2);
  tft.drawCentreString("RONQUIDO",120,2,2);
  tft.drawCentreString("F AIRE",180,2,2);
  // Se escriben la cabecera y el pie de la pantalla
  tft.setRotation(3);
  tft.fillRect(0,0,320,30,ILI9341_MAROON);
  tft.fillRect(0,210,320,30,ILI9341_MAROON);
  tft.setTextColor(ILI9341_WHITE);
  tft.drawCentreString("Monitor de Apnea",160,5,4);
  tft.drawRightString(miversion,318,222,2);
  // Líneas horizontales de separación
  tft.drawFastHLine(0,90,320,ILI9341_BLACK);
  tft.drawFastHLine(0,150,320,ILI9341_BLACK);
}
```

4.1.4 Función loop()

Tal y como se explica al inicio del apartado 4.1, esta función es la que se encarga de actualizar la lectura de los sensores y enviarlas a las distintas salidas. A continuación, se detalla cómo se ha implementado esta función.

4.1.4.1 Variables locales

El primer paso es declarar las variables locales, todas relacionadas con el valor instantáneo de los sensores:

- `AF` y `SN` son variables de tipo `uint16` que almacenan el valor digital de los sensores de flujo de aire y ronquidos respectivamente, por lo que será un valor entre 0 y 1024.
- `BP` es de tipo `uint8` y almacena el valor del sensor de posición, por tanto, valdrá entre 1 y `6`.

  Los valores digitales son los que se utilizarán para representar las gráficas de la interfaz portátil, ya que se requiere un valor entero para poder mapearlo al rango de píxeles disponible.

- `valAF` y `valSN` son variables `float` que registran el voltaje de los sensores.
- Para poder mostrar el valor de los sensores, es necesario tenerlo almacenado como cadena de caracteres, por lo que se han definido `charAF[8]`, `charSN[8]` y `charBP[8]`.

```
//Declaración de variables locales
  // Posición actual en el eje y de cada gráfica
  uint16_t AF;  // Flujo de aire (AirFlow)
  uint16_t SN;  // Ronquidos (SNoring)
  uint8_t BP;   // Posición corporal (Body Position)
  // Variables para lectura analógica de los sensores
  float valAF;
  float valSN;
  // Lectura de sensores en formato texto
  char charAF[8];
  char charSN[8];
  char charBP[8];
```

4.1.4.2 Actualización de los sensores

Para actualizar la lectura de los sensores se utilizan los métodos `getAirflow()`, `getSnore()` y `getBodyPosition()` de la librería de MySignals. En el caso del sensor de flujo de aire y de ronquidos, estos métodos reciben como entrada la constante `DATA` o `VOLTAGE`, definidas en la librería, que determinan si el valor que se devuelve es el valor digital o el del voltaje, en cada caso.

```
//Actualización de los sensores
  AF = (uint16_t)MySignals.getAirflow(DATA);
  valAF = MySignals.getAirflow(VOLTAGE);
  SN = (uint16_t)MySignals.getSnore(DATA);
  valSN = MySignals.getSnore(VOLTAGE);
  BP = (uint8_t)MySignals.getBodyPosition();
```

4.1.4.3 Actualización de las salidas

Finalmente, se procede a actualizar las salidas del programa. Como se explicó en el apartado 3.4 de este documento, la lectura de los sensores se pone a disposición del usuario por dos vías: la pantalla de la interfaz portátil y el puerto serie del Arduino.

Primero, **se envían los datos al puerto serie**. Para facilitar la generación del csv posterior, la escritura se hace respetando este formato. En cada ejecución de la función `loop()` se manda:

- el valor contenido en la variable AF
- un punto y coma (“;”)
- el valor de la variable valAF
- “;”
- el valor de la variable SN
- “;”
- el valor de la variable valSN
- “;”
- el valor de la variable BP
- “;”
- un salto de línea (“\n”)

Después, **se actualiza el valor actual de cada sensor**. Para ello, primero se convierte a caracteres. Posteriormente, se borra el valor anterior de la pantalla pintando un rectángulo blanco sobre el texto. Y, por último, se utiliza la función `drawCentreString()` para escribir el nuevo valor en la pantalla.

En el caso de los sensores de flujo de aire y de ronquidos, se utiliza la función `dtostrf()`para convertir los valores de voltaje de flotante a cadena de caracteres. A esta función se le pasa como entrada el número que se quiere convertir, el tamaño del número en caracteres, el número de decimales que tiene el valor y la variable en la que se quiere almacenar el resultado [12].

Para el sensor de posición, se utiliza `sprintf()` para convertir el valor de entero a carácter. Esta función recibe la variable en la que se almacenará el resultado, el formato de los valores a convertir, en este caso **"%d"**, y el valor a convertir [12].

Además, en la sección de posición, se añade una etiqueta para facilitar la interpretación del valor del sensor. Para ello, se escribe con `drawCentreString()` la etiqueta correspondiente gracias a una estructura condicional `switch-case`.

A continuación, **se actualizan las gráficas**. Es necesario escalar el valor leído por el sensor con la función `map()` para que entre dentro de los límites de la gráfica. Esta función tiene como entradas:

1. El valor a escalar
2. El valor mínimo que puede tener este número
3. El valor máximo que puede tener este número
4. El valor mínimo escalado
5. El valor máximo escalado

Y devuelve el valor escalado, que sobreescribimos en la variable original.

Para actualizar la gráfica se utiliza la función `drawGraph()`, que se explicará posteriormente, enviándole: el valor actual del sensor, el valor anterior almacenado en una variable global y los límites superior e inferior de las gráficas en cada caso. La función nos devuelve el valor del sensor, que utilizaremos para actualizar el valor previo.

El último paso es la actualización de las gráficas de la variable graph x, que se incrementa en cada ejecución. Cuando la variable llega al extremo derecho de la gráfica, vuelve a comenzar desde la izquierda.

Finalmente, se hace una **parada de 100 milisegundos**.

```
// Actualización de las salidas
  // Flujo de aire
    // Envío al puerto serie
    Serial.print(AF);
    Serial.print(";");
    Serial.print(valAF);
    Serial.print(";");
    // Actualización del valor en la pantalla
    dtostrf(valAF, 6, 2, charAF);
    tft.fillRect(21,41,88,30,ILI9341_WHITE);
    tft.setTextColor(ILI9341_BLACK);
    tft.drawCentreString(charAF,65,45,4);
    // Actualización de la gráfica
    AF = map(AF, 0, 250, AF_low_extrem, AF_high_extrem);
    AF_prevRead = drawGraph(AF, AF_prevRead, AF_low_extrem, AF_high_extrem);
```

```
  // Ronquidos
    // Envío al puerto serie
    Serial.print(SN);
    Serial.print(";");
    Serial.print(valSN);
    Serial.print(";");
    // Actualización del valor en la pantalla
    dtostrf(valSN, 6, 2, charSN);
    tft.fillRect(21,101,88,30,ILI9341_WHITE);
    tft.setTextColor(ILI9341_BLACK);
    tft.drawCentreString(charSN,65,105,4);
    // Actualización de la gráfica
    SN = map(SN, 0, 1023, SN_low_extrem, SN_high_extrem);
    SN_prevRead = drawGraph(SN, SN_prevRead, SN_low_extrem, SN_high_extrem);
  // Body position sensor
    // Envío al puerto serie
    Serial.print(BP);
    Serial.print(";");
    Serial.print("\n");
    // Actualización del valor y la etiqueta en la pantalla
    sprintf(charBP, "%d", BP);
    tft.fillRect(21,151,88,58,ILI9341_WHITE);
    tft.setTextColor(ILI9341_BLACK);
    tft.drawCentreString(charBP,65,165,4);
    switch (BP) {
      case 1:
        tft.drawCentreString("Ventral",65,185,2);
        break;
      case 2:
        tft.drawCentreString("Izquierda",65,185,2);
        break;
      case 3:
        tft.drawCentreString("Derecha",65,185,2);
        break;
      case 4:
        tft.drawCentreString("Supino",65,185,2);
        break;
      case 5:
        tft.drawCentreString("De pie/Sent",65,185,2);
        break;
      default:
        tft.drawCentreString("Indefinido",65,185,2);
        break;
      }
    // Actualización de la gráfica
    BP = map(BP, 0, 6, BP_low_extrem, BP_high_extrem);
    BP_prevRead = drawGraph(BP, BP_prevRead, BP_low_extrem, BP_high_extrem);

// Actualización de la x
graph_x++;
if (graph_x == graph_right_extrem)
{
  graph_x = graph_left_extrem;
}
// Parada de 100 milisegundos
```

```
  delay(100);
```

4.1.5 Función drawGraph()

4.1.5.1 Prototipo

La función `drawGraph()` es la que se utiliza para pintar las líneas de las gráficas. Devuelve como salida una variable de tipo `uint16_t`, que se corresponderá con el nuevo valor de la lectura anterior.

Las entradas de la función son:

- `value`: Es el nuevo valor de los sensores.
- `prevRead`: Es el valor de los sensores en la ejecución anterior. De esta forma, se puede pintar la línea desde este valor, hasta el nuevo.
- low_`extrem`: es el margen inferior del espacio asignado a la gráfica.
- `high`_extrem: es el margen superior del espacio asignado a la gráfica.

En Arduino no es necesario declarar el prototipo al inicio del código o en un fichero de cabecera, antes de la propia definición de la función.

```
// Función para la gráfica
uint16_t drawGraph(uint16_t value, uint16_t prevRead,uint16_t low_extrem, uint1
6_t high_extrem)
```

4.1.5.2 Definición

La función realiza 3 acciones que permiten implementar las gráficas:

1. Se comprueba que los valores a representar no sean superiores ni inferiores a los márgenes de la gráfica. Si lo fueran, se actualizarían a los valores de los extremos superiores e inferiores respectivamente, de forma que la gráfica aparecería saturada por alguno de los extremos. Este valor es el que luego se devuelve como salida de la función.
2. Se comprueba que se tienen al menos 2 valores para dibujar la línea correspondiente a la gráfica, tomando como inicio el valor de la lectura anterior, y como fin el de la nueva lectura.
3. Se pinta una línea vertical del color del fondo en la posición de la x que se actualizará en el ciclo siguiente (x+1).

```
  //Rectifica el valor de la Y si se sale de los límites de la gráfica
  if (value < high_extrem)
  {
    value = high_extrem;
  }
  if (value > low_extrem)
  {
    value = low_extrem;
  }
  // Pinta una linea entre el valor anterior del sensor y el nuevo valor
  // a partir de que ya exista al menos 1 valor
  if (graph_x > graph_left_extrem + 1)
  {
```

```
    tft.drawLine(graph_x - 1, prevRead, graph_x, value, ILI9341_MAROON);
  }
  // Barre pantalla pintando una linea del color del fondo, para la siguiente p
osición de la x
  tft.drawLine(graph_x + 1, high_extrem, graph_x + 1, low_extrem, ILI9341_WHITE
);

  SPI.end();
  return(value);
```

4.2 Interfaz de análisis

Para complementar las funcionalidades del proyecto, se ha añadido una aplicación de escritorio programada en Python. El módulo de Python elegido para desarrollar la interfaz de usuario ha sido Tkinter [13] [14] [15]. También son importantes Pandas [16], NumPy [17] y Matplotlib [18], que se han utilizado para el análisis de los datos y la generación de las gráficas. Por último, cabe destacar la librería fpdf [19], que se ha usado para generar el informe en formato PDF.

La interfaz utiliza un sistema de procesamiento multihilo para la generación del CSV, por lo que es significativo nombrar la librería Threading [20].

4.2.1 Interfaz

La interfaz se ha definido en el script **`monitor_apnea.py`**, que se adjunta en el Anexo II. Para su desarrollo, se ha creado una clase de Python llamada `MonitorInterface`, que contiene tanto la definición del aspecto de la interfaz, así como la de las funciones que se ejecutan al pulsar cada uno de sus botones.

En los próximos apartados se va a explicar dicho script.

4.2.1.1 Librerías

En este script, se importan las librerías:

- `tkinter`: Como se ha indicado, se utiliza para el desarrollo de la interfaz.
    - Se importa el módulo `font` de la librería `tkinter`, para personalizar las fuentes de la interfaz.
- `serial`: para la conexión al puerto serie del Arduino.
- `datetime`: para añadir una marca temporal a cada registro del csv.
- `threading`: para crear un hilo cada vez que se comienza a crear un csv.
- `os`: se utiliza para gestionar las rutas en las que se crean los archivos.
- Se importa la clase `PDF`, definida en el script `create_report.py` como parte de este proyecto, que se explicará posteriormente.

```
# Librerías
import tkinter
from tkinter import font
import serial
import datetime
import threading
import os
from create_report import PDF
```

4.2.1.2 Clase MonitorInterface

Esta clase reúne la estructura de la interfaz y los métodos que implementan las distintas acciones asociadas a los botones de la misma. A lo largo de ella, se puede apreciar el uso del objeto `self`, que representa la instancia de la clase. Este objeto permite acceder a los métodos y las variables definidas dentro de la clase desde cualquier punto de esta.

De este modo, por ejemplo, no es lo mismo definir la variable `i`, que `self.i`. La primera, `i`, sólo es accesible en el ámbito local del método en el que se defina, mientras que la segunda, `self.i` será una variable global de la clase.

Para la definición de la estructura gráfica se utilizarán los distintos widgets proporcionados por Tkinter, en concreto:

- `tkinter.Label`: Para añadir títulos a los distintos apartados.
- `tkinter.Frame`: Para contener otros widgets y organizarlos de manera más sencilla.
- `tkinter.Button`: Para añadir los distintos botones de la interfaz.
- `tkinter.Text`: Campos de texto utilizados para indicar las rutas de salida.
- `tkinter.Scrollbar`: Para añadir una barra de desplazamiento en los campos de texto.
- `tkinter.Entry`: Campos de entrada de texto.
- `tkinter.Radiobutton`: Botones de selección. En esta interfaz se usan para permitir al usuario elegir entre uno de los puertos USBs detectados.
- `tkinter.Scale`: Barra deslizable. Se utiliza para seleccionar el número de segundos que se quiere considerar como intervalo de apnea.

Para fijar la distribución de estos widgets se ha usado el gestor de geometría `grid`, que trata el espacio de la ventana como si hubiera en ella una cuadrícula, pudiendo así determinar con exactitud donde se quiere situar cada elemento. Además, permite que haya componentes que se expandan, ocupando más de una fila o columna. La figura 4-2 muestra cómo se han distribuido los widgets en la aplicación.

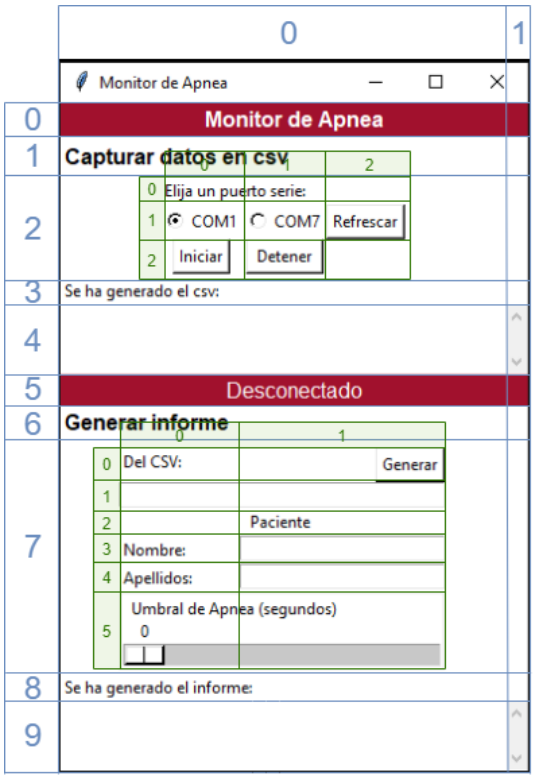


Figura 4-2. Posición de los widgets en la aplicación

4.2.1.2.1 Método _init_()

El método `__init__()` en Python es un constructor que se ejecuta de forma automática después de crear una instancia de la clase. En esta clase se define la estructura gráfica fundamental de la interfaz.

Lo primero que se hace es inicializar cuatro variables que serán necesarias a lo largo de la definición de esta clase:

- `self.csv_threads` será vector que contenga todos los hilos de Python que se crean para generar un csv.
- `self.i` se utiliza para saber en qué columna se deben colocar los distintos elementos.
- `self.file_name` almacena el nombre del último fichero CSV que se haya usado.
- `self.output_path` contiene la ruta a la carpeta en la que se van a generar los ficheros de salida.

Inmediatamente después, se comprueba que esta carpeta exista y si no es así se crea.

```
# Inicialización de variables
self.csv_threads = []
self.i = 1
self.file_name = ""
self.output_path = os.path.abspath("./outputs/")
# Comprobación de carpeta de salida
if not os.path.exists(self.output_path):
    os.mkdir(self.output_path)
    print("Se ha creado la carpeta " + self.output_path)
```

Entonces se comienza a definir y posicionar los distintos elementos de la interfaz:

- Se crea una instancia en el objeto `self.window` de la clase `Tk()`, que hace referencia a la ventana raíz de la aplicación. Todos los elementos que se coloquen en ella se encontrarán dentro de la cuadrícula dibujada en azul en la figura 4-2. Además, se establece el título de la ventana, su tamaño inicial y el color de fondo de la misma.
- Se definen dos fuentes: `self.title_font`, con la letra en negrita, que es la que se usa para los títulos y `self.std_font`, con letra normal, que se usa para la barra de estado. El resto de los elementos utilizan la fuente que tienen asignada por defecto.
- Para homogeneizar el aspecto de la aplicación con el resto de los elementos desarrollados en este trabajo, se añade una franja del color del logotipo de la Universidad de Sevilla con el título de la aplicación en letras blancas.

```
    # Definición de la interfaz
    # Ventana raíz
    self.window = tkinter.Tk()
    self.window.title("Monitor de Apnea")
    self.window.geometry('350x500')
    self.window.configure(bg="white")
    # Fuentes
    self.title_font = font.Font(weight='bold')
    self.std_font = font.Font(weight='normal')
    # Franja de título de la aplicación
    self.title_lbl = tkinter.Label(self.window, text="Monitor de Apnea", bg="#a
1112d", fg="white", font=self.title_font)
    self.title_lbl.grid(column=0, row=0, columnspan=2, sticky=(tkinter.N,tkinte
r.E, tkinter.W))
```

- Por un lado, se definen los componentes relacionados con la captura de los datos:
    - Se añade el título de la sección, “Capturar datos en csv”, con un widget de tipo `tkinter.Label` y se sitúa dentro de la ventana principal.
    - Debajo se coloca un contenedor, `conn_frame`, creado con el widget `tkinter.Frame`. En su interior, se inserta:
        - El selector del puerto serie. Para poder añadir las distintas opciones se utiliza la función `list_sp()`, que:
            - Crea el vector `self.sp_list`, con todos los puertos series que se detectan conectados al ordenador.
            - Añade los selectores correspondientes a la interfaz, y guarda la instancia del objeto correspondiente en el vector `self.rad`.
            - Añade el botón “Refrescar”, que permite actualizar el listado de puertos series.
        - El botón de “Iniciar”. Este botón lanza el método `sp_clicked()` al ser pulsado. Con él se crea un hilo que ejecuta el método `write_csv()`, que se explica en el apartado 4.2.1.2.2, para así evitar que la interfaz se bloquee mientras se está generando el CSV. El constructor del hilo se añade al vector `self.csv_threads`. Antes de iniciar el hilo, se establece como `False` el evento `self.stop`, que comunica cuándo debe dejar de generarse el fichero.

La barra de estado de actualiza, mostrando el texto “Generando CSV”.

- El botón de “Detener”, junto con la creación del evento `self.stop`. Cuando se pulsa este botón se ejecuta el método `stop_clicked()`, en el que se establece `self.stop` como `True`, para notificar a los hilos de que debe finalizar la ejecución.

  La barra de estado recupera el valor “Desconectado” y se escribe en la interfaz la ruta del archivo generado.

- Un widget de tipo `tkinter.Text`, no editable [21], en el que se escribirá la ruta en la que se ha generado el CSV. Se ha elegido este tipo de componente ya que permite visualizar su contenido en múltiples líneas y es seleccionable, por lo que facilita al usuario copiar la ruta en caso necesario. Se acompaña de una barra de deslizamiento (`tkinter.Scrollbar`) para desplazarse por las distintas líneas.
- Finalmente, la barra de estado, mencionada anteriormente y que como valor inicial muestra “Desconectado”.

```
    # Widgets del Capturador de datos
    # Título
    self.csv_lbl = tkinter.Label(self.window, text="Capturar datos en csv", fon
t=self.title_font, bg="white")
    self.csv_lbl.grid(column=0, row=1, sticky=(tkinter.N, tkinter.W))
    # Contenedor 1 - conn_frame
    self.conn_frame = tkinter.Frame(self.window, borderwidth=2, relief="flat",
bg="white")
    self.conn_frame.grid(column=0, row=2)
    self.window.columnconfigure(0, weight=1)
    self.window.rowconfigure(0, weight=1)
    # Selector de puerto serie
    self.sp_lbl = tkinter.Label(self.conn_frame, text="Elija un puerto USB: ",
bg="white")
    self.sp_lbl.grid(column=0, row=0, columnspan=3, sticky=(tkinter.W))
    self.list_sp()
    # Botón de inicio
    self.sp_btn = tkinter.Button(self.conn_frame, text="Iniciar", command=self.
sp_clicked, bg="white")
    self.sp_btn.grid(column=self.i-1, row=3)
    # Botón y evento de parada
    self.stop = threading.Event()
    self.stop_btn = tkinter.Button(self.conn_frame, text=" Detener ", command=s
elf.stop_clicked, bg="white")
    self.stop_btn.grid(column=self.i, row=3)
    # Ruta del salida
    self.file_lbl = tkinter.Label(self.window, text="Se ha generado el csv: ",
bg = "white")
    self.file_lbl.grid(column=0, row=4, sticky=(tkinter.N, tkinter.W))
    # https://www.tutorialspoint.com/python/tk_scrollbar.htm
    self.csv_scroll = tkinter.Scrollbar(self.window)
    self.csv_scroll.grid(column=1,row=5)
    self.name_txt = tkinter.Text(self.window,width=40, height=3, relief=tkinter
.FLAT, yscrollcommand=self.csv_scroll.set)
    self.name_txt.grid(column=0, row=5, sticky=(tkinter.N, tkinter.W, tkinter.E
))
    self.name_txt.config(state=tkinter.DISABLED)
```

```
    self.csv_scroll.config( command = self.name_txt.yview)
    # Barra de estado
    self.status = tkinter.Label(self.window, text="Desconectado", bg="#a1112d",
 fg="white",font=self.std_font)
    self.status.grid(column=0, row=6, columnspan=2, sticky=(tkinter.N,tkinter.E
, tkinter.W))
```

- Por otro, los widgets del generador de informes:
  - Se añade el título de la sección, "Generar informe", con un widget de tipo `tkinter.Label` y se sitúa dentro de la ventana principal.
  - Debajo se coloca un contenedor, `report_frame`. En su interior, se inserta:
    - Campo de tipo `tkinter.Entry` para introducir la ruta del fichero CSV del que se quiere obtener el informe. Como se ha mencionado anteriormente, este campo se autorellena cuando se terminar de crear un CSV, pero también es modificable, por si el usuario quisiera utilizar otro fichero más antiguo.
    - Un campo para introducir el nombre del paciente y otro para los apellidos.
    - El seleccionador del umbral de apnea. de tipo `tkinter.Scale`, con un deslizable que permite elegir un número entero entre el 0 y el 300. Si se selecciona 0, no se remarcarán los intervalos en los que el flujo de aire es 0.
    - El botón de "Generar", que lanza el método `report_clicked()`. Este método se encarga de leer los campos de entrada de la interfaz e instanciar la clase PDF del fichero create_report.py, que se explica en el apartado 4.2.1.3, para generar el informe. Después, se escribe en la interfaz la ruta del archivo generado.
  - Un widget de tipo `tkinter.Text`, no editable, en el que se escribirá la ruta en la que se ha generado el PDF, acompañado de una barra de deslizamiento (`tkinter.Scrollbar`) para desplazarse por las distintas líneas.
- Como último paso del `__init__()`, se utiliza el método `mainloop()`, para generar la ventana y quedar a la espera de que el usuario pulse alguno de los botones.

```
    # Widgets del Generador de Informe
    # Título
    self.report_lbl = tkinter.Label(self.window, text="Generar informe", font=s
elf.title_font, bg="white")
    self.report_lbl.grid(column=0, row=7, sticky=(tkinter.N, tkinter.W))
    # Contenedor 2 - report_frame
    self.report_frame = tkinter.Frame(self.window, borderwidth=2, bg="white")
    self.report_frame.grid(column=0, row=8, sticky=(tkinter.N))
    self.window.rowconfigure(7, weight=1)
    # Campo de entrada Fichero CSV
    self.wrt_lbl = tkinter.Label(self.report_frame, text="Del CSV: ", bg="white
")
    self.wrt_lbl.grid(column=0, row=0, sticky=(tkinter.N,tkinter.W))
    self.file_entry = tkinter.Entry(self.report_frame, bg="white", width=40)
    self.file_entry.grid(column=0, row=1, columnspan=2, sticky=(tkinter.N, tkin
ter.W, tkinter.E))
    # Nombre y Apellidos del paciente
    self.patient_lbl = tkinter.Label(self.report_frame, text="Paciente ", bg="w
hite")
```

```
    self.patient_lbl.grid(column=0, row=2, columnspan=2, sticky=(tkinter.N, tki
nter.W, tkinter.E))
    self.pName_lbl = tkinter.Label(self.report_frame, text="Nombre: ", bg="whit
e")
    self.pName_lbl.grid(column=0, row=3, sticky=(tkinter.N, tkinter.W))
    self.pName_entry = tkinter.Entry(self.report_frame, bg="white", width=25)
    self.pName_entry.grid(column=1, row=3, sticky=(tkinter.N, tkinter.E))
    self.pLast_lbl = tkinter.Label(self.report_frame, text="Apellidos: ", bg="w
hite")
    self.pLast_lbl.grid(column=0, row=4, columnspan=2, sticky=(tkinter.N, tkint
er.W))
    self.pLast_entry = tkinter.Entry(self.report_frame, bg="white", width=25)
    self.pLast_entry.grid(column=1, row=4, sticky=(tkinter.N, tkinter.E))
    # Umbral de Apnea
    self.var = tkinter.IntVar()
    self.scale = tkinter.Scale(self.report_frame,label="Umbral de Apnea (segund
os)" , variable = self.var, from_=0, to=300, bg = "white", orient=tkinter.HORIZ
ONTAL )
    self.scale.grid(column=0, row=5, columnspan=2, sticky=(tkinter.N, tkinter.W
, tkinter.E))
    # Botón Generar
    self.report_btn = tkinter.Button(self.report_frame, text="Generar", command
=self.report_clicked, bg="white")
    self.report_btn.grid(column=1, row=0, sticky=(tkinter.N, tkinter.E))
    # Ruta del salida
    self.endReport_lbl = tkinter.Label(self.window, text="Se ha generado el inf
orme: ", bg = "white")
    self.endReport_lbl.grid(column=0, row=9, sticky=(tkinter.N, tkinter.W))
    self.report_scroll = tkinter.Scrollbar(self.window)
    self.report_scroll.grid(column=1,row=10)
    self.report_txt = tkinter.Text(self.window,width=40, height=3, relief=tkint
er.FLAT, yscrollcommand=self.report_scroll.set)
    self.report_txt.grid(column=0, row=10, sticky=(tkinter.N, tkinter.W, tkinte
r.E))
    self.report_txt.config(state=tkinter.DISABLED)
    self.report_scroll.config(command = self.report_txt.yview)

    # Ejecución de la ventana
    self.window.mainloop()
```

4.2.1.2.2 Captura de datos

La función que genera el fichero CSV está definida en el método `write_csv()` de la clase `MonitorInterface`. Se encarga de detectar el puerto serie seleccionado por el usuario, abrir la conexión y leer las líneas enviadas por el Arduino. A estas les añade un campo inicial con la fecha y hora del momento y las escribe en el fichero correspondiente.

```
def write_csv(self):
    # Lee el puerto seleccionado
    print(self.sp_selected.get())
    sp = self.sp_list[self.sp_selected.get()]
    arduino = serial.Serial(port=sp, baudrate=115200, timeout=None)
```

```
    if arduino.isOpen() is False:
        print("Puerto Erroneo")
        # Si no puede abrir la conexión, queda a la espera de que se pulse Dete
ner
        while not self.stop.is_set():
            pass
    else:
        # Construye el nombre del fichero
        date = datetime.datetime.now().strftime("%Y%m%d%H%M%S")
        self.file_name = self.output_path + "/monitor-apnea_" + date + ".csv"
        print(self.file_name)
        # Mientras que no se pulse Detener
        while not self.stop.is_set():
            # Lee las líneas enviadas al puerto serie del Arduino
            data = arduino.readline().decode("utf-8")
            # Añade un campo con la fecha y hora
            date = datetime.datetime.now().strftime("%Y/%m/%d %H:%M:%S.%f")
            line = date+";"+data
            line = line.replace("\r","")
            print(line)
            # Las escribe en el fichero
            with open(self.file_name, "a") as f:
                f.write(line)
        if arduino.isOpen():
            print("Cerrando puerto")
            arduino.close()
```

4.2.1.3 Generación de informes

El código para generar los informes en PDF se ha definido en el script de Python `create_report.py`. Como se indicó al inicio del apartado 4.2, se utilizan las librerías pandas, NumPy para el tratamiento de los datos y matplotlib para generar las gráficas. También se usa la librería FPDF, como base para generar el documento.

4.2.1.3.1 Método __init__()

En el script se define la clase `PDF()`, que hereda de `FPDF()`. El flujo principal del programa se encuentra dentro del método `__init__()` de la clase. En él se añaden las siguientes entradas a la clase PDF, además de las heredadas:

- `csv_path`: para pasar la ruta del csv del que se quiere obtener el informe.
- `name`: contendrá el nombre del paciente.
- `last_name`: para los apellidos.
- `apnea_TH`: umbral de apnea, es decir, el número de segundos a partir del cual se quieren destacar los periodos sin respiración en el informe. Por defecto, se le ha asignado el valor 10.
- `write_sum`: booleano que determina si se calculan o no los intervalos que superan el umbral de apnea. Si no se recibe como entrada al instanciar la clase, valdrá `True`.
- `output_path`: ruta a la carpeta en la que se quieren guardar las imágenes y el PDF generado. Por defecto, se ha usado "`./outputs/`".

Además, al inicio del mismo, se pasan las entradas de la clase padre gracias a `super().__init__()`. Con ellas se define el formato del papel, A4 apaisado, y se indica que se va a trabajar en milímetros.

También se definen algunos parámetros útiles para el resto del código:

- El ancho y el largo de la página, teniendo en cuenta que se deja 1 cm de margen por cada lado.
- Un diccionario para decodificar la posición detectada por el sensor, en el que la clave es el número que devuelve el sensor y el valor es el `string` que indica el nombre de la posición.
- Un booleano para activar o desactivar fácilmente los bordes de todos los cuadros de texto.

Después se guardan las entradas en variables globales de la clase y se construye la ruta del fichero PDF tomando las iniciales del paciente y la fecha de generación del CSV.

```
class PDF(FPDF):
    def __init__(self, csv_path, name, last_name, apnea_TH = 10, write_sum=True
, output_path = "./outputs/"):
        # Formato del papel: Landscape, A4. Las medidas se usan en mm
        super().__init__('L', 'mm','A4')
        # Parámetros
        self.WIDTH = 277 #297-20 {1cm de margen}
        self.HEIGHT = 190 #210-20
        self.dict_BP = {
            1 : "Decúbito ventral",
            2 : "Decúbito lateral izquierdo",
            3 : "Decúbito lateral derecho",
            4 : "Decúbito supino",
            5 : "De pie o sentado",
            6 : "Indefinido"
        }
        self.b = False # Bordes de los cuadros de texto
        # Entradas accesibles en toda la clase
        self.name = name
        self.last_name =last_name
        self.apnea_TH = np.timedelta64(apnea_TH, 's')
        self.write_sum = write_sum
        self.output_path = output_path

        # Construccion de la ruta del fichero
        # Iniciales del nombre del paciente
        self.initials = ""
        if self.name != "":
            for c in self.name.split(" "):
                self.initials += c[0]
        if self.last_name != "":
            for c in last_name.split(" "):
                self.initials += c[0]
        # Fecha de generación del CSV
        self.csv_date = csv_path.split("/")[-1].split(".")[0].split("_")[-1]
        self.PDF_path = self.output_path + self.initials + "_informe-
apnea_" + self.csv_date + ".pdf"
```

El siguiente paso consiste en leer el CSV como un *dataframe* de pandas usando la función `read_csv()`, almacenándolo en `self.df`, y añadirle una columna con la fecha en formato *datetime* para poder hacer cálculos con ella.

Los últimos pasos del método `__init__()` son invocar al método `create_report()`, definido más abajo, y guardar el fichero con el método `output()`, heredado de `FPDF()`.

```
    # Lectura CSV
    col_names = ["date", "AF_dig", "AF_volt", "SN_dig", "SN_volt", "BP"]
    self.df = pd.read_csv(csv_path, sep=";", names=col_names, index_col=False)
    self.df["date_dt"]=pd.to_datetime(self.df["date"])

    # Construccion de las páginas
    self.create_report()
    # Guargar fichero generado
    self.output(self.PDF_path, 'F')
```

4.2.1.3.2 Gráficas y umbral de apnea

Para las gráficas, se crea un png por cada página del PDF. El **método `create_graph_img()`** es el encargado de generarlas. Dentro de él, si `self.write_sum` tomó el valor `True`, se invoca al método `calc_TH()` para escribir el texto del PDF en el que se indican los intervalos de apnea y añadir al *dataframe* una columna en la que se puedan identificar.

Después, con la función de NumPy `array_split()` se genera un vector de *dataframes*, dividiendo el original en otros más pequeños de aproximadamente 500 líneas. Esto permite que el número de puntos por cada página no sea demasiado elevado para así facilitar la visualización de los datos. En el caso de que el CSV tuviera menos de 500 líneas, simplemente se introduce el *dataframe* original en un vector, para poder realizar el siguiente bucle.

A continuación, se itera en el vector de *dataframes* para generar la imagen correspondiente en cada caso. Cada una de ellas cuenta con 4 gráficas: las 3 principales, que representan las lecturas del sensor de flujo de aire, el de ronquidos y el de posición, y una cuarta que se incluye por detrás de las otras 3, en la que solo es interesante el eje X y su cuadrícula. Esta última gráfica es la que crea las líneas verticales que atraviesan a las otras 3, facilitando la lectura simultánea.

Figura 4-3. Posición de las gráficas

En la gráfica de flujo de aire, si se ha calculado el umbral de apnea, se utiliza el método `span_where()` [22] para remarcar los intervalos de apnea. Por otro lado, para la de posición, se ha usado `set_yticks()` y `set_yticklabels()` para sustituir los números por las etiquetas de posición que le corresponden.

Para terminar, se guardan las gráficas como png y se almacenan las rutas en el vector `graph_name`, que es salida del método descrito.

```
# Se crean los png de las gráficas
def create_graph_img(self):
    mm = (1/2.54)*0.1 # Conversión de pulgadas a mm
    plt.rcParams['axes.xmargin'] = 0 # Elimina el margen por defecto en el eje
X
    i=1
    graph_name =  [] # Vector para las rutas de los gráficos creados
    # Calcula los intervalos que superan el umbral
    if self.write_sum:
        self.calc_TH()

    # Se divide en df de aproximadamente 500 líneas
    if self.df.shape[0]>500:
        df_parts = np.array_split(self.df, int(self.df.shape[0]/500))
    else:
        df_parts = [self.df]
    # Se crea 1 imagen para cada partición
    for df_part in df_parts:
        # Tamaño de la imagen. Con 3 gráficos, distribuidos en 3 filas y 1 colu
mna
        fig, axs = plt.subplots(3, 1, figsize=(self.WIDTH*mm, (self.HEIGHT-
25)*mm))
        # Primera gráfica. Flujo de aire
        axs[0].plot(df_part.index, df_part.AF_dig, color="#8D1528" )
        axs[0].set_ylabel('Flujo de aire')
        axs[0].get_xaxis().set_visible(False)
        axs[0].grid(True, axis="both", which="both")
        if self.write_sum:
            collection = collections.BrokenBarHCollection.span_where(self.df.in
dex, ymin=self.df.AF_dig.min(), ymax=self.df.AF_dig.max(), where=self.df['AF_th
reshold'], facecolor='maroon', alpha=0.2)
            axs[0].add_collection(collection)
        # Segunda gráfica. Ronquidos
        axs[1].plot(df_part.date_dt, df_part.SN_dig, color="#8D1528")
        axs[1].set_ylabel('Ronquidos')
        axs[1].get_xaxis().set_visible(False)
        axs[1].set_xticklabels([])
        axs[1].grid(True)
        # Tercera gráfica. Posición
        axs[2].plot(df_part.date_dt, df_part.BP, color = "#8D1528")
        axs[2].set_xlabel('Tiempo')
        axs[2].set_ylabel('Posición')
        y_BP = [1,2,3,4,5,6]
        y_BP_values = ["Dec vent", "Dec izq", "Dec dcho", "Dec sup", "Pie/sent"
, "Indef"]
```

```
        axs[2].set_yticks(y_BP)
        axs[2].set_yticklabels(y_BP_values)
        axs[2].get_xaxis().set_visible(False)
        axs[2].grid(True)
        # Gráfica de fondo, para las etiquetas y cuadrícula del eje X
        ax3 = fig.add_subplot(111, zorder=-1)
        for _, spine in ax3.spines.items():
            spine.set_visible(False)
        ax3.tick_params(labelleft=False, labelbottom=True, left=False, right=Fa
lse )
        ax3.plot(df_part.date_dt,np.zeros(df_part.date_dt.shape[0]), alpha=0.0)
        ax3.grid(axis="x")
        fig.tight_layout()
        # Se guarda en png
        graph_name +=  [self.output_path + self.initials + '_grafica-
'+str(i)+ "_" + self.csv_date + '.png']
        i += 1
        plt.savefig(graph_name[-1], transparent=True)
        print("El gráfico", graph_name[-1] , "se ha guardado")

    return graph_name
```

Por su parte, el **método `calc_TH()`** añade una nueva columna al *dataframe*, `AF_threshold`, que valdrá `True` cuando el registro esté dentro de un intervalo en el que se supera el umbral de apnea y `False` en caso contrario.

Para determinar cuáles son estos periodos, primero agrupa cada uno de los intervalos en los que, de forma consecutiva, el flujo de aire es cero [23]. Después se filtran, quedándose únicamente aquellos en los que la diferencia entre el instante final y el inicial supera el umbral establecido. Con estos datos se genera el texto del informe que, como último paso de la función se escribe en una nueva página con el método heredado `multi_cell()`, que interpreta los caracteres de salto de línea.

```
#  Cáculo de intervalos de apnea
def calc_TH(self):
    summary = ""
    self.df["AF_threshold"] = False # Nueva columna en el df inicializada a Fal
se
    i = 1
    # Se itera por un listado de Dataframes en los que el flujo de aire es cero
 consecutivamente
    df_group = self.df[self.df['AF_dig'] == 0].groupby((self.df['AF_dig'] != 0)
.cumsum())
    for name, group in df_group:
        # Si la diferencia entre el instante final y el inicial del df supera e
l umbral
        if (group['date_dt'].iloc[-1] -
 group['date_dt'].iloc[0]) >= self.apnea_TH:
            # Se actualiza la nueva columna a True en el df original
            self.df.loc[group.index,'AF_threshold']=True
            # Se genera el texto del informe
            summary += str(i)+") Se ha detenido la respiración durante " + str(
(group['date_dt'].iloc[-1] -
 group['date_dt'].iloc[0]).total_seconds()) + " segundos, desde las " + group['
```

```
date_dt'].iloc[0].strftime("%H:%M:%S.%f") + " hasta las " + group['date_dt'].il
oc[-1].strftime("%H:%M:%S.%f") +".\n"
            summary += "\tLa posición durante este tiempo ha sido:\n"
            for p in group['BP'].unique():
                summary += "\t\t- " + self.dict_BP[p] + "\n"
            summary +="\tLos ronquidos han sido superiores a la media\n" if gro
up['SN_dig'].mean() > self.df['SN_dig'].mean() else "\tLos ronquidos no han sid
o superiores a la media\n"
            summary += "\n"
            i += 1
    summary = "Se ha detenido la respiración " + str(i-
1) + " veces durante más de " + str(pd.Timedelta(self.apnea_TH).total_seconds()
) + " segundos.\n\n\n" + summary
    # Se escribe el texto en el PDF
    self.add_page()
    self.set_font('Times', '', 12)
    self.multi_cell( w = self.WIDTH, h = 6,txt = summary, align ='L', border =
self.b)
```

4.2.1.3.3 Otros métodos

Faltan por mencionar 4 métodos más definidos en la clase PDF().

El primero, `getOutputPath()` es la implementación de un método *getter* que simplemente devuelve del valor de la variable `self.PDF_path`.

Se definen también la cabecera y el pie de página con los métodos `header()` y `footer()`, respectivamente. En la cabecera se añade al inicio de la página los logos de la Universidad de Sevilla y de la Escuela Técnica Superior de Ingeniería, el título del documento y el nombre del paciente. En el pie de página se escribe el número de página.

Por último, `create_report()` recorre el vector que devuelve `create_graph_img()` con las rutas a las gráficas generadas y, para cada una, añade una nueva página y coloca la imagen.

```
def getOutputPath(self):
    # Devuelve la ruta del fichero generado
    return self.PDF_path

# Definición de la cabecera
def header(self):
    # Añade los logos si existen
    if os.path.exists('assets/logo_US_2.png'):
        self.image('assets/logo_US_2.png', 12, 12, h = 15)
    if os.path.exists('assets/logo_ETSI.png'):
        self.image('assets/logo_ETSI.png', 30, 12, h = 15)
    # Título del fichero
    title = "INFORME DE APNEA DEL SUEÑO\nDEL " + self.df['date_dt'].iloc[0].dat
e().strftime("%d/%m/%Y") \
        if self.df['date_dt'].iloc[0].date() == self.df['date_dt'].iloc[-
1].date() \
        else "INFORME DE APNEA DEL SUEÑO\nDEL" + self.df['date_dt'].iloc[0].dat
e().strftime("%d/%m/%Y") +  " AL " \
```

```
        + self.df['date_dt'].iloc[-1].date().strftime("%d/%m/%Y")
    # Nombre del paciente
    self.set_font('Arial', 'B', 14)
    self.cell(60,h = 20, border = self.b)
    self.multi_cell( w = self.WIDTH -
 120, h = 10,txt = title, align ='C', border =  self.b)
    self.set_xy(self.WIDTH - 50,10)
    self.set_font('Arial', 'B', 14)
    self.cell(60,h = 8, border = self.b, txt = "Nombre del paciente:", align="L
")
    self.set_font('Arial', '', 14)
    self.set_xy(self.WIDTH - 50,18)
    self.multi_cell (w = 60, h = 6, txt = self.name + "\n" + self.last_name, al
ign =  "R", border = self.b)
    self.ln(20)

# Definición del pie de página
def footer(self):
    # Escribe el número de página
    self.set_y(-15)
    self.set_font('Arial', 'I', 10)
    self.set_text_color(128)
    self.cell(0, 12, 'Página ' + str(self.page_no()), 0, 0, 'C')


# Añade 1 página por cada png creado
def create_report(self):
    graph_name = self.create_graph_img()
    print(graph_name)
    for graph in graph_name:
        self.add_page()
        self.image(graph, x=10, y=35, w=self.WIDTH)
```

4.2.2 Generación de ejecutable

Para facilitar el uso de la aplicación de escritorio se crea un fichero ejecutable, de lo contrario, los usuarios deberían tener Python y todas las dependencias instaladas en sus ordenadores para hacer uso de la aplicación.

Para generar el ejecutable se ha utilizado Pyinstaller [24] [25], ejecutando el comando:

```
pyinstaller --onefile --windowed --icon=./assets/sleeping.ico monitor_apnea.py
```

Con esto se genera el archivo monitor_apnea.exe dentro de la carpeta “dist”, que puede ejecutarse en cualquier ordenador con sistema operativo Windows.

5 MANUAL DE USUARIO

A continuación, se describen los pasos a seguir para la instalación del entorno de desarrollo y el uso del dispositivo.

5.1 Instalación

5.1.1 Arduino

Para la implementación del monitor en Arduino, se ha usado el entorno de desarrollo integrado proporcionado por Arduino (Arduino IDE) [26], que puede descargarse en el siguiente enlace:

https://www.arduino.cc/en/software

Una vez instalado el Arduino IDE se deben añadir las librerías proporcionadas por MySignals en un archivo zip [27]:

https://www.cooking-hacks.com/media/cooking/images/documentation/mysignals_hardware/MySignals_HW_SDK_V2.0.2.zip

Para cargar las librerías en el entorno de desarrollo, abre el Arduino IDE y en la barra de opciones pulsa: *Programa* > *Incluir Librería* > *Añadir biblioteca .ZIP*

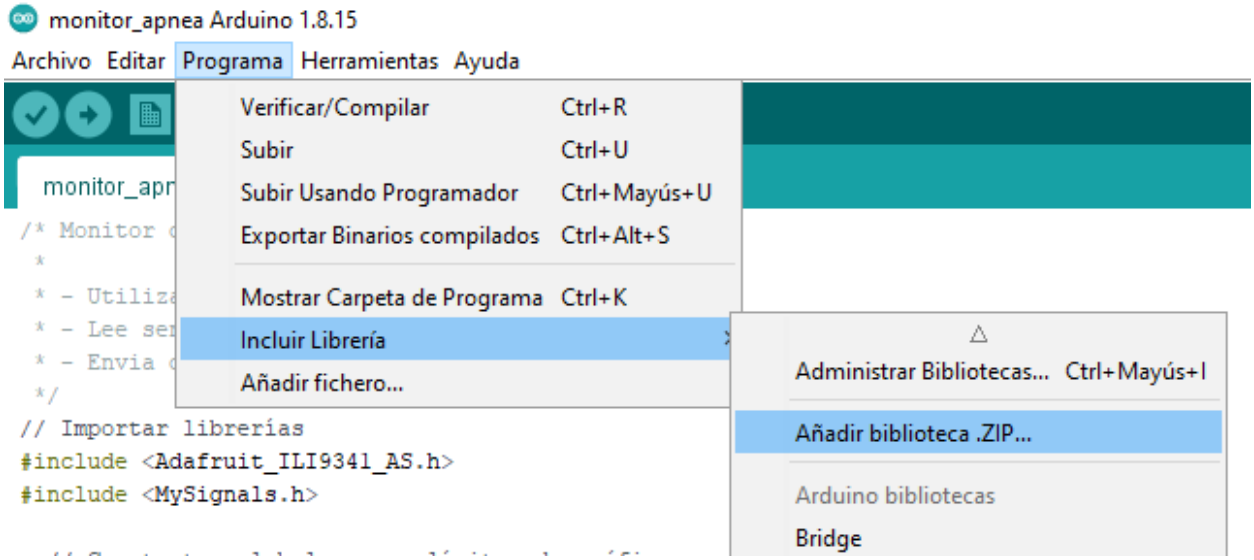


Figura 5-1. Incluir librerías en Arduino IDE

El entorno está listo para subir el fichero monitor_apnea.ino al Arduino.

5.1.2 Python

Para la interfaz de análisis, hay que:

1. Instalar Python. Para el desarrollo de esta aplicación se ha utilizado Python 3.9.4., que puede descargarse en la página oficial de Python [28]:

   https://www.python.org/downloads/

2. Instalar los paquetes de Python no nativos utilizados en el proyecto. Puedes usar el administrador de paquetes de Python. Para ello, ejecuta los siguientes comandos en el terminal de Windows:

   a. `pip install pyserial` [29]

b. `pip install pandas` [30]

c. `pip install matplotlib` [31]

d. `pip install fpdf` [32]

e. pip install pyinstaller

3. Instalar un IDE para Python. En este proyecto se ha usado Visual Studio Code [33]:

   https://code.visualstudio.com/

   a. Instalar la extensión de Python de Microsoft:

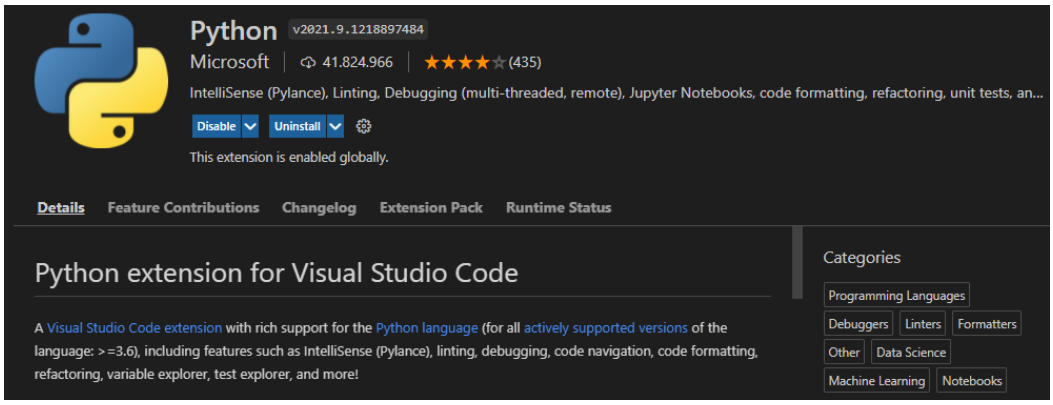


Figura 5-2. Extensión de Python para Visual Studio Code

4. Para ejectuar el código desde VSC, deben estar guardados ambos ficheros, monitor_apnea.py y create_report.py, en el mismo nivel de carpeta. Además, si se desea que aparezcan los logotipos en el PDF generado, deben descargarse y guardarse dentro de la carpeta "assets/".

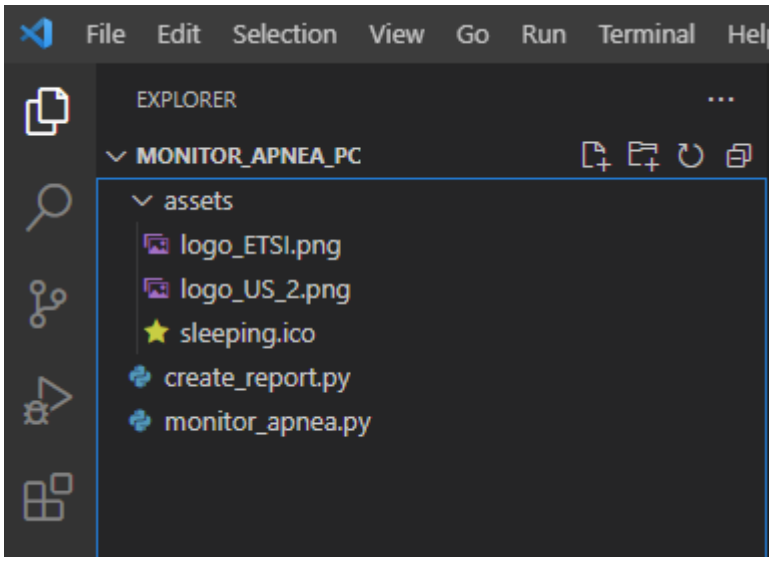


Figura 5-3. Distribución de carpetas para Python

5. Para lanzar la interfaz, ejecuta el código desde el fichero create_report.py.

5.2 Puesta en marcha

Conecte los sensores al dispositivo según se indica en la siguiente imagen:

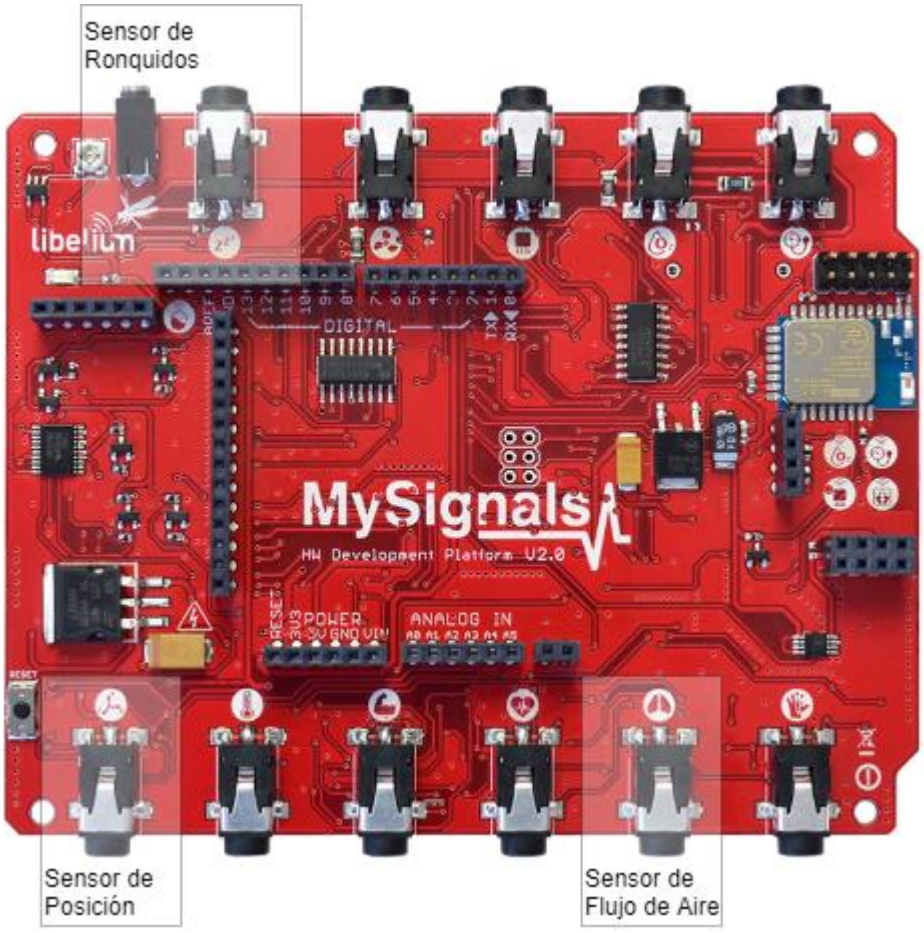


Figura 5-4. Colocación de sensores [7]

Coloque los sensores correctamente en el paciente:

1. El sensor de flujo de aire consta de 2 partes. Tenga cuidado al conectarlas, para respetar la polaridad fíjese en las marcas de los laterales.
2. Para situar el sensor de flujo de aire en la posición adecuada, debe hacerse como se muestra en la siguiente imagen:

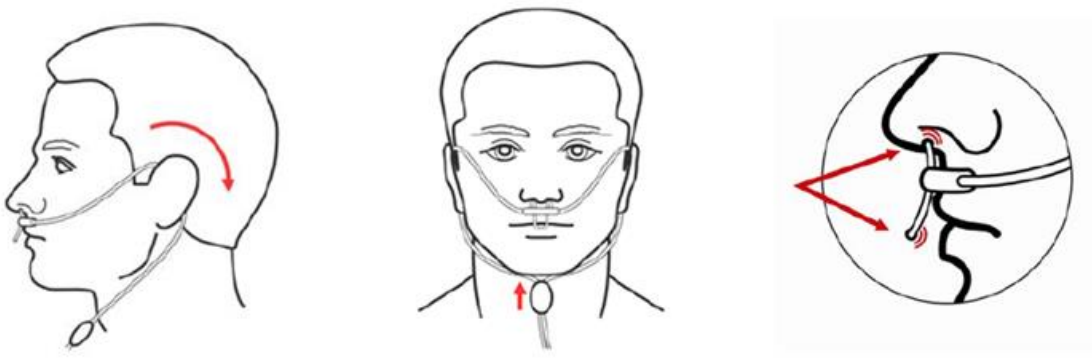


Figura 5-5. Colocación del sensor de flujo de aire [7]

3. Coloque el sensor de ronquidos alrededor del cuello.
4. Sitúe el sensor de posición alrededor del tronco.

Conecte el dispositivo portátil al puerto USB del ordenador usando un cable como el que se muestra en la siguiente imagen.

Figura 5-6. Cable USB para Arduino [34]

Al hacerlo, el dispositivo debe encenderse y cargar la pantalla de monitorización.

En el ordenador, abra la aplicación monitor_apnea.exe haciendo doble clic sobre el icono. En el siguiente apartado se explicará cómo navegar por ella.

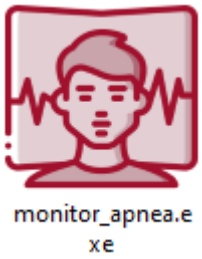


Figura 5-7. Icono de la aplicación de escritorio [35]

5.3 Manual de pantallas

5.3.1 Dispositivo portátil

La pantalla del dispositivo muestra lo siguiente:

1. Valores de los sensores en tiempo de real. Para la posición, también se indica a qué se corresponde el número mostrado.
2. Gráficas con aproximadamente 1 minuto de histórico.

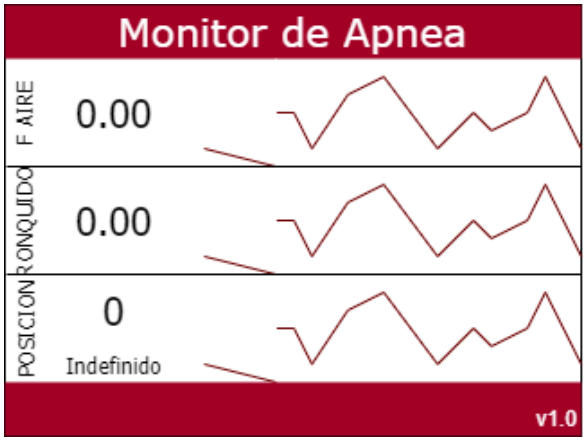


Figura 5-8. Interfaz del monitor portátil

5.3.2 Aplicación de escritorio

Los pasos a seguir para generar un fichero CSV son los siguiente:

1. Seleccione el puerto al que está conectado el dispositivo.
    a. Si abrió la aplicación antes de conectar el dispositivo al ordenador, pulse "Refrescar" para actualizar los puertos que aparecen.
2. Pulse "Iniciar" para comenzar a capturar los datos.
3. El estado cambiará a "Generando CSV"
4. Cuando haya acabado, pulse "Detener".
5. La ruta en la que se encuentra el fichero generado se mostrará en la pantalla. También se rellena en el campo de entrada para generar el informe.
6. El estado volverá a ser "Desconectado".

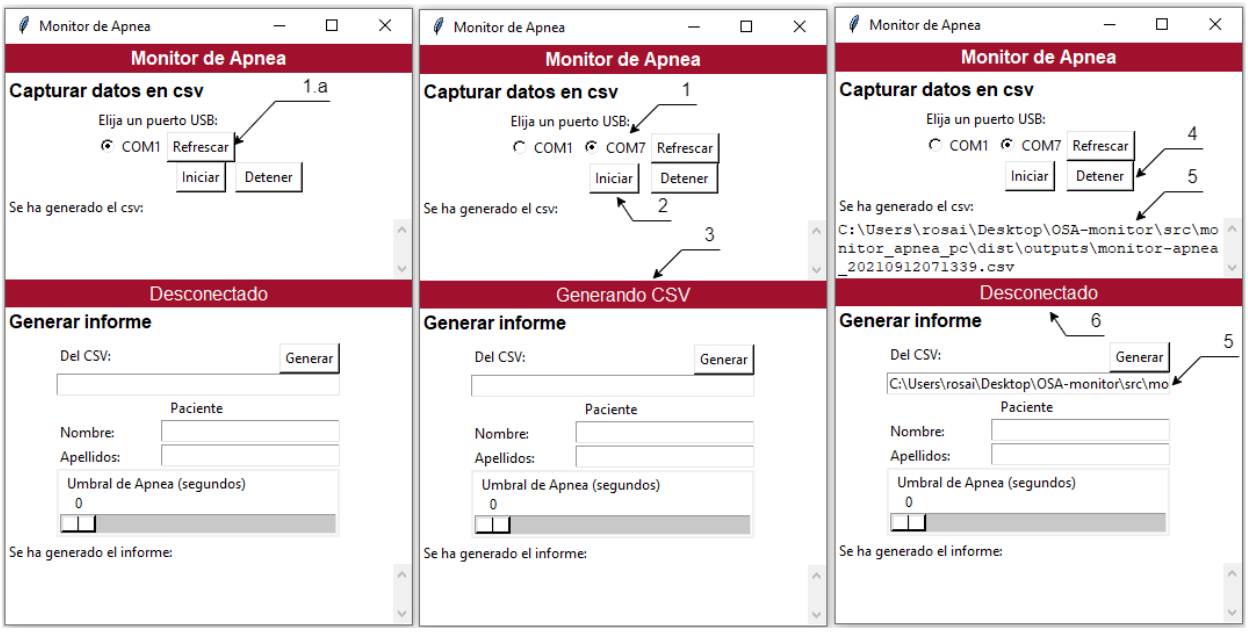


Figura 5-9. Pasos para generar CSV

Para generar un informe:

1. Rellene el campo “Del CSV:” con la ruta al fichero del que quiere obtener el informe.
2. Introduzca el nombre del paciente.
3. Introduzca los apellidos.
4. Seleccione el número de segundos a partir del cual que quiere considerar que el usuario ha entrado en apnea. Si selecciona 0 solo obtendrá las gráficas de los datos, no se buscará ningún intervalo de apnea.
5. Pulse el botón “Generar”.
6. La ruta al fichero generado se mostrará en la pantalla.

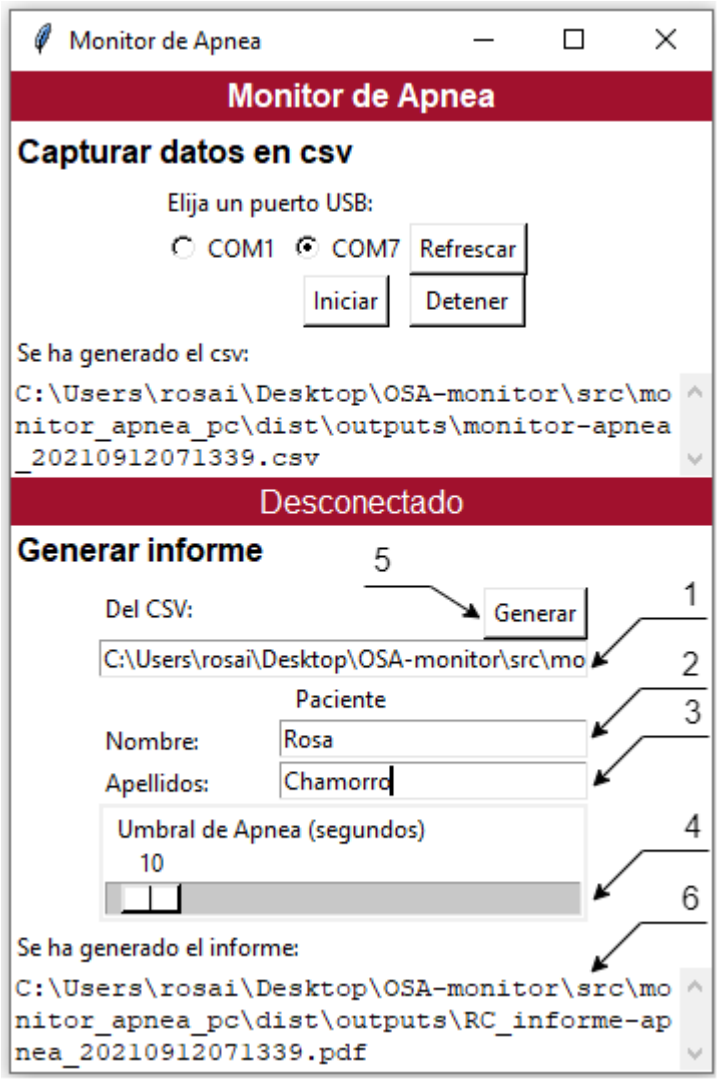


Figura 5-10. Pasos para generar informe

6 CONCLUSIONES Y TRABAJO FUTURO

Como conslusión, se ha hecho uso de los sensores proporcionados, seleccionando aquellos más interesantes para cumplir el propósito de monitorizar la calidad del sueño y detectar los posibles episodios de apnea. Además, se ha podido hacer un análisis de los datos obtenidos con los sensores, cumpliendo así los objetivos del proyecto.

Al mismo tiempo, se han creado dos interfaces de usuario sencillas que facilitan el uso de las herramientas desarrolladas para los usuarios, ya sean médicos o pacientes.

Como posible mejora, hacer que la captura de datos no sea a través del ordenador, sino almacenándolos en una tarjeta SD integrada en la propia unidad haría que el dispositivo fuera más cómodo de usar. Esta idea se ha intentado llevar a cabo como parte de este proyecto, pero no ha sido posible debido a la memoria Flask del modelo de Arduino usado, de tan solo 32 kB. El código que se ha implementado finalmente ocupa un 96% de este espacio, por lo que al intentar añadir la libería para el manejo de la SD se terminaba superando el límite establecido.

Finalmente, la incorporación de otros sensores, como por ejemplo el sensor de saturación de oxígeno en sangre, podría ayudar a enriquecer las muestras de datos tomadas. Al igual que en el punto anterior, la memoria del dispositivo ha sido determinante a la hora de limitar el número de sensores utilizado, ya que el uso de más sensores también repercute en el tamañodel programa.

ANEXO A: MONITOR_APNEA.INO

```
/* Monitor de Apnea v1.0
 *
 * - Utiliza 1 funciones para realizar las 3 gráficas
 * - Lee sensores de Airflow, Snoring y Body Position
 * - Envía datos en formato csv al puerto serie
 */
// Importar librerías
#include <Adafruit_ILI9341_AS.h>
#include <MySignals.h>

// Constantes globales para límites de gráficas
#define graph_left_extrem 110
#define graph_right_extrem 320
// Limites superiores
#define AF_high_extrem 31
#define SN_high_extrem 91 //diff 58
#define BP_high_extrem 151
// Límites inferiores
#define AF_low_extrem 89
#define SN_low_extrem 149
#define BP_low_extrem 209

// Variables globales
char miversion[] = "v1.0";
// Instancia para la pantalla TFT
Adafruit_ILI9341_AS tft = Adafruit_ILI9341_AS(TFT_CS, TFT_DC);
// Posición actual de las gráficas en el eje x
uint16_t graph_x = graph_left_extrem;
// Posición en el eje y de cada gráfica en la iteración anterior
uint16_t AF_prevRead;
uint16_t SN_prevRead;
uint8_t BP_prevRead;

void setup() {
  // Inicialización del puerto serie a 115200 baudios
  Serial.begin(115200);
  // Inicialización de MySignals y del sensor de ronquidos
  MySignals.begin();
  MySignals.initSnore();
  // Inicialización de la pantalla
  tft.init();
  tft.setRotation(2);
  tft.fillScreen(ILI9341_WHITE); // Se limpia la pantalla
  // Se escriben las etiquetas de los sensores
```

```
  tft.setTextColor(ILI9341_BLACK);
  tft.drawCentreString("POSICION",60,2,2);
  tft.drawCentreString("RONQUIDO",120,2,2);
  tft.drawCentreString("F AIRE",180,2,2);
  // Se escriben la cabecera y el pie de la pantalla
  tft.setRotation(3);
  tft.fillRect(0,0,320,30,ILI9341_MAROON);
  tft.fillRect(0,210,320,30,ILI9341_MAROON);
  tft.setTextColor(ILI9341_WHITE);
  tft.drawCentreString("Monitor de Apnea",160,5,4);
  tft.drawRightString(miversion,318,222,2);
  // Líneas horizontales de separación
  tft.drawFastHLine(0,90,320,ILI9341_BLACK);
  tft.drawFastHLine(0,150,320,ILI9341_BLACK);
}

void loop() {
  //Declaración de variables locales
    // Posición actual en el eje y de cada gráfica
    uint16_t AF;  // Flujo de aire (AirFlow)
    uint16_t SN;  // Ronquidos (SNoring)
    uint8_t BP;   // Posición corporal (Body Position)
    // Lectura de sensores en formato texto
    char charAF[8];
    char charSN[8];
    char charBP[8];
    // Variables para lectura analógica de los sensores
    float valAF;
    float valSN;
  // Actualización de los sensores
  AF = (uint16_t)MySignals.getAirflow(DATA);
  valAF = MySignals.getAirflow(VOLTAGE);
  SN = (uint16_t)MySignals.getSnore(DATA);
  valSN = MySignals.getSnore(VOLTAGE);
  BP = (uint8_t)MySignals.getBodyPosition();
  // Actualización de los sensores
    // Flujo de aire
      // Envío al puerto serie
      Serial.print(AF);
      Serial.print(";");
      Serial.print(valAF);
      Serial.print(";");
      // Actualización del valor en la pantalla
      dtostrf(valAF, 6, 2, charAF);
      tft.fillRect(21,41,88,30,ILI9341_WHITE);
      tft.setTextColor(ILI9341_BLACK);
      tft.drawCentreString(charAF,65,45,4);
      // Actualización de la gráfica
      AF = map(AF, 0, 250, AF_low_extrem, AF_high_extrem);
```

```
      AF_prevRead = drawGraph(AF, AF_prevRead, AF_low_extrem, AF_high_ext
rem);
    // Ronquidos
      // Envío al puerto serie
      Serial.print(SN);
      Serial.print(";");
      Serial.print(valSN);
      Serial.print(";");
      // Actualización del valor en la pantalla
      dtostrf(valSN, 6, 2, charSN);
      tft.fillRect(21,101,88,30,ILI9341_WHITE);
      tft.setTextColor(ILI9341_BLACK);
      tft.drawCentreString(charSN,65,105,4);
      // Actualización de la gráfica
      SN = map(SN, 0, 1023, SN_low_extrem, SN_high_extrem);
      SN_prevRead = drawGraph(SN, SN_prevRead, SN_low_extrem, SN_high_ext
rem);
    // Body position sensor
      // Envío al puerto serie
      Serial.print(BP);
      Serial.print(";");
      Serial.print("\n");
      // Actualización del valor y la etiqueta en la pantalla
      sprintf(charBP, "%d", BP);
      tft.fillRect(21,151,88,58,ILI9341_WHITE);
      tft.setTextColor(ILI9341_BLACK);
      tft.drawCentreString(charBP,65,165,4);
      switch (BP) {
        case 1:
          tft.drawCentreString("Ventral",65,185,2);
          break;
        case 2:
          tft.drawCentreString("Izquierda",65,185,2);
          break;
        case 3:
          tft.drawCentreString("Derecha",65,185,2);
          break;
        case 4:
          tft.drawCentreString("Supino",65,185,2);
          break;
        case 5:
          tft.drawCentreString("De pie/Sent",65,185,2);
          break;
        default:
          tft.drawCentreString("Indefinido",65,185,2);
          break;
        }
      // Actualización de la gráfica
      BP = map(BP, 0, 6, BP_low_extrem, BP_high_extrem);
```

```
      BP_prevRead = drawGraph(BP, BP_prevRead, BP_low_extrem, BP_high_ext
rem);

  // Actualización de la x
  graph_x++;
  if (graph_x == graph_right_extrem)
  {
    graph_x = graph_left_extrem;
  }
  // Parada de 100 milisegundos
  delay(100);
}

// Función para la gráfica
uint16_t drawGraph(uint16_t value, uint16_t prevRead,uint16_t low_extrem,
 uint16_t high_extrem)
{
  //Rectifica el valor de la Y si se sale de los límites de la gráfica
  if (value < high_extrem)
  {
    value = high_extrem;
  }
  if (value > low_extrem)
  {
    value = low_extrem;
  }
  // Pinta una linea entre el valor anterior del sensor y el nuevo valor
  // a partir de que ya exista al menos 1 valor
  if (graph_x > graph_left_extrem + 1)
  {
    tft.drawLine(graph_x - 1, prevRead, graph_x, value, ILI9341_MAROON);
  }
  // Barre pantalla pintando una linea del color del fondo, para la sigui
ente posición de la x
  tft.drawLine(graph_x + 1, high_extrem, graph_x + 1, low_extrem, ILI9341
_WHITE);

  SPI.end();
  return(value);
}
```

ANEXO B: MONITOR_APNEA.PY

```
# Librerías
import tkinter
from tkinter import font
import serial
import datetime
import threading
import os
from create_report import PDF

class MonitorInterface:
    def __init__(self):
        # Inicialización de variables
        self.csv_threads = []
        self.i = 1
        self.file_name = ""
        self.output_path = os.path.abspath("./outputs/")
        # Comprobación de carpeta de salida
        if not os.path.exists(self.output_path):
            os.mkdir(self.output_path)
            print("Se ha creado la carpeta " + self.output_path)

        # Definición de la interfaz
        # Ventana raíz
        self.window = tkinter.Tk()
        self.window.title("Monitor de Apnea")
        self.window.geometry('350x500')
        self.window.configure(bg="white")
        # Fuentes
        self.title_font = font.Font(weight='bold')
        self.std_font = font.Font(weight='normal')
        # Franja de título de la aplicación
        self.title_lbl = tkinter.Label(self.window, text="Monitor de Apne
a", bg="#a1112d", fg="white", font=self.title_font)
        self.title_lbl.grid(column=0, row=0, columnspan=2, sticky=(tkinte
r.N,tkinter.E, tkinter.W))

        # Widgets del Capturador de datos
        # Título
        self.csv_lbl = tkinter.Label(self.window, text="Capturar datos en
 csv", font=self.title_font, bg="white")
        self.csv_lbl.grid(column=0, row=1, sticky=(tkinter.N, tkinter.W))
        # Contenedor 1 - conn_frame
        self.conn_frame = tkinter.Frame(self.window, borderwidth=2, relie
f="flat", bg="white")
        self.conn_frame.grid(column=0, row=2)
```

```
        self.window.columnconfigure(0, weight=1)
        self.window.rowconfigure(0, weight=1)
        # Selector de puerto serie
        self.sp_lbl = tkinter.Label(self.conn_frame, text="Elija un puert
o USB: ", bg="white")
        self.sp_lbl.grid(column=0, row=0, columnspan=3, sticky=(tkinter.W
))
        self.list_sp()
        # Botón de inicio
        self.sp_btn = tkinter.Button(self.conn_frame, text="Iniciar", com
mand=self.sp_clicked, bg="white")
        self.sp_btn.grid(column=self.i-1, row=3)
        # Botón y evento de parada
        self.stop = threading.Event()
        self.stop_btn = tkinter.Button(self.conn_frame, text=" Detener ",
 command=self.stop_clicked, bg="white")
        self.stop_btn.grid(column=self.i, row=3)
        # Ruta del salida
        self.file_lbl = tkinter.Label(self.window, text="Se ha generado e
l csv: ", bg = "white")
        self.file_lbl.grid(column=0, row=4, sticky=(tkinter.N, tkinter.W)
)
        # https://www.tutorialspoint.com/python/tk_scrollbar.htm
        self.csv_scroll = tkinter.Scrollbar(self.window)
        self.csv_scroll.grid(column=1,row=5)
        self.name_txt = tkinter.Text(self.window,width=40, height=3, reli
ef=tkinter.FLAT, yscrollcommand=self.csv_scroll.set)
        self.name_txt.grid(column=0, row=5, sticky=(tkinter.N, tkinter.W,
 tkinter.E))
        self.name_txt.config(state=tkinter.DISABLED)
        self.csv_scroll.config( command = self.name_txt.yview)
        # Barra de estado
        self.status = tkinter.Label(self.window, text="Desconectado", bg=
"#a1112d", fg="white",font=self.std_font)
        self.status.grid(column=0, row=6, columnspan=2, sticky=(tkinter.N
,tkinter.E, tkinter.W))

        # Widgets del Generador de Informe
        # Título
        self.report_lbl = tkinter.Label(self.window, text="Generar inform
e", font=self.title_font, bg="white")
        self.report_lbl.grid(column=0, row=7, sticky=(tkinter.N, tkinter.
W))
        # Contenedor 2 - report_frame
        self.report_frame = tkinter.Frame(self.window, borderwidth=2, bg=
"white")
        self.report_frame.grid(column=0, row=8, sticky=(tkinter.N))
        self.window.rowconfigure(7, weight=1)
        # Campo de entrada Fichero CSV
```

```
        self.wrt_lbl = tkinter.Label(self.report_frame, text="Del CSV: ",
 bg="white")
        self.wrt_lbl.grid(column=0, row=0, sticky=(tkinter.N,tkinter.W))
        self.file_entry = tkinter.Entry(self.report_frame, bg="white", wi
dth=40)
        self.file_entry.grid(column=0, row=1, columnspan=2, sticky=(tkint
er.N, tkinter.W, tkinter.E))
        # Nombre y Apellidos del paciente
        self.patient_lbl = tkinter.Label(self.report_frame, text="Pacient
e ", bg="white")
        self.patient_lbl.grid(column=0, row=2, columnspan=2, sticky=(tkin
ter.N, tkinter.W, tkinter.E))
        self.pName_lbl = tkinter.Label(self.report_frame, text="Nombre: "
, bg="white")
        self.pName_lbl.grid(column=0, row=3, sticky=(tkinter.N, tkinter.W
))
        self.pName_entry = tkinter.Entry(self.report_frame, bg="white", w
idth=25)
        self.pName_entry.grid(column=1, row=3, sticky=(tkinter.N, tkinter
.E))
        self.pLast_lbl = tkinter.Label(self.report_frame, text="Apellidos
: ", bg="white")
        self.pLast_lbl.grid(column=0, row=4, columnspan=2, sticky=(tkinte
r.N, tkinter.W))
        self.pLast_entry = tkinter.Entry(self.report_frame, bg="white", w
idth=25)
        self.pLast_entry.grid(column=1, row=4, sticky=(tkinter.N, tkinter
.E))
        # Umbral de Apnea
        self.var = tkinter.IntVar()
        self.scale = tkinter.Scale(self.report_frame,label="Umbral de Apn
ea (segundos)" , variable = self.var, from_=0, to=300, bg = "white", orie
nt=tkinter.HORIZONTAL )
        self.scale.grid(column=0, row=5, columnspan=2, sticky=(tkinter.N,
 tkinter.W, tkinter.E))
        # Botón Generar
        self.report_btn = tkinter.Button(self.report_frame, text="Generar
", command=self.report_clicked, bg="white")
        self.report_btn.grid(column=1, row=0, sticky=(tkinter.N, tkinter.
E))
        # Ruta del salida
        self.endReport_lbl = tkinter.Label(self.window, text="Se ha gener
ado el informe: ", bg = "white")
        self.endReport_lbl.grid(column=0, row=9, sticky=(tkinter.N, tkint
er.W))
        self.report_scroll = tkinter.Scrollbar(self.window)
        self.report_scroll.grid(column=1,row=10)
        self.report_txt = tkinter.Text(self.window,width=40, height=3, re
lief=tkinter.FLAT, yscrollcommand=self.report_scroll.set)
```

```
        self.report_txt.grid(column=0, row=10, sticky=(tkinter.N, tkinter
.W, tkinter.E))
        self.report_txt.config(state=tkinter.DISABLED)
        self.report_scroll.config(command = self.report_txt.yview)

        # Ejecución de la ventana
        self.window.mainloop()

    def list_sp(self):
        # Actualiza los selectores de Puerto Serie
        # Listar puertos serie
        self.sp_list = self.serial_ports()
        print(self.sp_list)
        # Añadir selectores
        self.sp_selected = tkinter.IntVar()
        self.i=0
        self.rad = []
        for sp in self.sp_list:
            self.rad.append(tkinter.Radiobutton(self.conn_frame,text=sp,
value=self.i, variable=self.sp_selected, bg="white"))
            self.rad[self.i].grid(column=self.i, row=1,sticky=(tkinter.N,
tkinter.E, tkinter.W))
            self.i=self.i+1
        # Botón de Refrescar
        self.refresh_button = tkinter.Button(self.conn_frame, text="Refre
scar", command=self.refresh_clicked, bg="white")
        self.refresh_button.grid(column=self.i, row=1)

    def serial_ports(self):
        # Crea la lista de puertos seieres detectados
        ports = ['COM%s' % (i + 1) for i in range(256)]
        result = []
        for port in ports:
            try:
                s = serial.Serial(port)
                s.close()
                result.append(port)
            except (OSError, serial.SerialException):
                pass
        return result

    def sp_clicked(self):
        # Crea hilo que ejecuta write_csv()
        print("Creando hilo")
        self.stop.clear()
        tmp = threading.Thread(target=self.write_csv)
        self.csv_threads.append(tmp)
        tmp.start()
        # Barra de estado
```

```
        self.status = tkinter.Label(self.window, text="Generando CSV", bg
="#a1112d", fg="white",font=self.std_font)
        self.status.grid(column=0, row=6, columnspan=2, sticky=(tkinter.N
,tkinter.E, tkinter.W))

    def stop_clicked(self):
        # Fin de los hilos
        print("Finalizando hilos")
        self.stop.set()
        for csv_thread in self.csv_threads:
            csv_thread.join()
            print(csv_thread, "is joined")
        # Barra de estado
        self.status = tkinter.Label(self.window, text="Desconectado", bg=
"#a1112d", fg="white",font=self.std_font)
        self.status.grid(column=0, row=6, columnspan=2, sticky=(tkinter.N
,tkinter.E, tkinter.W))
        # Escribir ruta al fichero generado
        self.file_name = os.path.abspath(self.file_name)
        self.name_txt.config(state=tkinter.NORMAL)
        self.name_txt.delete("1.0", "1."+str(len(self.file_name)))
        self.name_txt.insert("1.0", self.file_name)
        self.name_txt.config(state=tkinter.DISABLED)
        # También en la entrada para generar informe
        self.file_entry.delete ( 0, last=len(self.file_name) )
        self.file_entry.insert(tkinter.END, self.file_name)

    def refresh_clicked(self):
        print("Refrescando listado de puerto serie")
        for rad_button in self.rad:
            rad_button.destroy()
        self.refresh_button.destroy()
        self.list_sp()

    def report_clicked(self):
        self.file_name = self.file_entry.get()
        print(self.file_name)
        pName = self.pName_entry.get()
        pLast = self.pLast_entry.get()
        TH = self.var.get()
        writeSum = TH != 0
        print(TH, writeSum)
        pdf = PDF(csv_path=self.file_name, name=pName, last_name=pLast, a
pnea_TH = TH, write_sum =writeSum).getOutputPath()
        pdf=self.file_name = os.path.abspath(pdf)
        print(pdf)
        self.report_txt.config(state=tkinter.NORMAL)
        self.report_txt.delete("1.0", "1."+str(len(pdf)))
        self.report_txt.insert("1.0", pdf)
```

```
        self.report_txt.config(state=tkinter.DISABLED)

    def write_csv(self):
        # Lee el puerto seleccionado
        print(self.sp_selected.get())
        sp = self.sp_list[self.sp_selected.get()]
        arduino = serial.Serial(port=sp, baudrate=115200, timeout=None)
        if arduino.isOpen() is False:
            print("Puerto Erroneo")
            # Si no puede abrir la conexión, queda a la espera de que se
pulse Detener
            while not self.stop.is_set():
                pass
        else:
            # Construye el nombre del fichero
            date = datetime.datetime.now().strftime("%Y%m%d%H%M%S")
            self.file_name = self.output_path + "/monitor-
apnea_" + date + ".csv"
            print(self.file_name)
            # Mientras que no se pulse Detener
            while not self.stop.is_set():
                # Lee las líneas enviadas al puerto serie del Arduino
                data = arduino.readline().decode("utf-8")
                # Añade un campo con la fecha y hora
                date = datetime.datetime.now().strftime("%Y/%m/%d %H:%M:%
S.%f")
                line = date+";"+data
                line = line.replace("\r","")
                print(line)
                # Las escribe en el fichero
                with open(self.file_name, "a") as f:
                    f.write(line)
            if arduino.isOpen():
                print("Cerrando puerto")
                arduino.close()

MI = MonitorInterface()
```

ANEXO C: CREATE_REPORT.PY

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.collections as collections
from fpdf import FPDF
import os

class PDF(FPDF):
    def __init__(self, csv_path, name, last_name, apnea_TH = 10, write_su
m=True, output_path = "./outputs/"):
        # Formato del papel: Landscape, A4. Las medidas se usan en mm
        super().__init__('L', 'mm','A4')
        # Parámetros
        self.WIDTH = 277 #297-20 {1cm de margen}
        self.HEIGHT = 190 #210-20
        self.dict_BP = {
            1 : "Decúbito ventral",
            2 : "Decúbito lateral izquierdo",
            3 : "Decúbito lateral derecho",
            4 : "Decúbito supino",
            5 : "De pie o sentado",
            6 : "Indefinido"
        }
        self.b = False # Bordes de los cuadros de texto
        # Entradas accesibles en toda la clase
        self.name = name
        self.last_name =last_name
        self.apnea_TH = np.timedelta64(apnea_TH, 's')
        self.write_sum = write_sum
        self.output_path = output_path

        # Construccion de la ruta del fichero
        # Iniciales del nombre del paciente
        self.initials = ""
        if self.name != "":
            for c in self.name.split(" "):
                self.initials += c[0]
        if self.last_name != "":
            for c in last_name.split(" "):
                self.initials += c[0]
        # Fecha de generación del CSV
        self.csv_date = csv_path.split("/")[-
1].split(".")[0].split("_")[-1]
        self.PDF_path = self.output_path + self.initials + "_informe-
apnea_" + self.csv_date + ".pdf"
```

```
        # Lectura CSV
        col_names = ["date", "AF_dig", "AF_volt", "SN_dig", "SN_volt", "B
P"]
        self.df = pd.read_csv(csv_path, sep=";", names=col_names, index_c
ol=False)
        self.df["date_dt"]=pd.to_datetime(self.df["date"])

        # Construccion de las páginas
        self.create_report()
        # Guargar fichero generado
        self.output(self.PDF_path, 'F')

    def getOutputPath(self):
        # Devuelve la ruta del fichero generado
        return self.PDF_path

    # Definición de la cabecera
    def header(self):
        # Añade los logos si existen
        if os.path.exists('assets/logo_US_2.png'):
            self.image('assets/logo_US_2.png', 12, 12, h = 15)
        if os.path.exists('assets/logo_ETSI.png'):
            self.image('assets/logo_ETSI.png', 30, 12, h = 15)
        # Título del fichero
        title = "INFORME DE APNEA DEL SUEÑO\nDEL " + self.df['date_dt'].i
loc[0].date().strftime("%d/%m/%Y") \
            if self.df['date_dt'].iloc[0].date() == self.df['date_dt'].il
oc[-1].date() \
            else "INFORME DE APNEA DEL SUEÑO\nDEL" + self.df['date_dt'].i
loc[0].date().strftime("%d/%m/%Y") +  " AL " \
            + self.df['date_dt'].iloc[-1].date().strftime("%d/%m/%Y")
        # Nombre del paciente
        self.set_font('Arial', 'B', 14)
        self.cell(60,h = 20, border = self.b)
        self.multi_cell( w = self.WIDTH -
 120, h = 10,txt = title, align ='C', border =  self.b)
        self.set_xy(self.WIDTH - 50,10)
        self.set_font('Arial', 'B', 14)
        self.cell(60,h = 8, border = self.b, txt = "Nombre del paciente:"
, align="L")
        self.set_font('Arial', '', 14)
        self.set_xy(self.WIDTH - 50,18)
        self.multi_cell (w = 60, h = 6, txt = self.name + "\n" + self.las
t_name, align =  "R", border = self.b)
        self.ln(20)

    # Definición del pie de página
    def footer(self):
```

```
        # Escribe el número de página
        self.set_y(-15)
        self.set_font('Arial', 'I', 10)
        self.set_text_color(128)
        self.cell(0, 12, 'Página ' + str(self.page_no()), 0, 0, 'C')

    # Se crean los png de las gráficas
    def create_graph_img(self):
        mm = (1/2.54)*0.1 # Conversión de pulgadas a mm
        plt.rcParams['axes.xmargin'] = 0 # Elimina el margen por defecto
en el eje X
        i=1
        graph_name =  [] # Vector para las rutas de los gráficos creados
        # Calcula los intervalos que superan el umbral
        if self.write_sum:
            self.calc_TH()

        # Se divide en df de aproximadamente 500 líneas
        if self.df.shape[0]>500:
            df_parts = np.array_split(self.df, int(self.df.shape[0]/500))
        else:
            df_parts = [self.df]
        # Se crea 1 imagen para cada partición
        for df_part in df_parts:
            # Tamaño de la imagen. Con 3 gráficos, distribuidos en 3 fila
s y 1 columna
            fig, axs = plt.subplots(3, 1, figsize=(self.WIDTH*mm, (self.H
EIGHT-25)*mm))
            # Primera gráfica. Flujo de aire
            axs[0].plot(df_part.index, df_part.AF_dig, color="#8D1528" )
            axs[0].set_ylabel('Flujo de aire')
            axs[0].get_xaxis().set_visible(False)
            axs[0].grid(True, axis="both", which="both")
            if self.write_sum:
                collection = collections.BrokenBarHCollection.span_where(
self.df.index, ymin=self.df.AF_dig.min(), ymax=self.df.AF_dig.max(), wher
e=self.df['AF_threshold'], facecolor='maroon', alpha=0.2)
                axs[0].add_collection(collection)
            # Segunda gráfica. Ronquidos
            axs[1].plot(df_part.date_dt, df_part.SN_dig, color="#8D1528")
            axs[1].set_ylabel('Ronquidos')
            axs[1].get_xaxis().set_visible(False)
            axs[1].set_xticklabels([])
            axs[1].grid(True)
            # Tercera gráfica. Posición
            axs[2].plot(df_part.date_dt, df_part.BP, color = "#8D1528")
            axs[2].set_xlabel('Tiempo')
            axs[2].set_ylabel('Posición')
            y_BP = [1,2,3,4,5,6]
```

```
            y_BP_values = ["Dec vent", "Dec izq", "Dec dcho", "Dec sup",
"Pie/sent", "Indef"]
            axs[2].set_yticks(y_BP)
            axs[2].set_yticklabels(y_BP_values)
            axs[2].get_xaxis().set_visible(False)
            axs[2].grid(True)
            # Gráfica de fondo, para las etiquetas y cuadrícula del eje X

            ax3 = fig.add_subplot(111, zorder=-1)
            for _, spine in ax3.spines.items():
                spine.set_visible(False)
            ax3.tick_params(labelleft=False, labelbottom=True, left=False
, right=False )
            ax3.plot(df_part.date_dt,np.zeros(df_part.date_dt.shape[0]),
alpha=0.0)
            ax3.grid(axis="x")
            fig.tight_layout()
            # Se guarda en png
            graph_name +=  [self.output_path + self.initials + '_grafica-
'+str(i)+ "_" + self.csv_date + '.png']
            i += 1
            plt.savefig(graph_name[-1], transparent=True)
            print("El gráfico", graph_name[-1] , "se ha guardado")

        return graph_name

    #  Cáculo de intervalos de apnea
    def calc_TH(self):
        summary = ""
        self.df["AF_threshold"] = False # Nueva columna en el df iniciali
zada a False
        i = 1
        # Se itera por un listado de Dataframes en los que el flujo de ai
re es cero consecutivamente
        df_group = self.df[self.df['AF_dig'] == 0].groupby((self.df['AF_d
ig'] != 0).cumsum())
        for name, group in df_group:
            # Si la diferencia entre el instante final y el inicial del d
f supera el umbral
            if (group['date_dt'].iloc[-1] -
 group['date_dt'].iloc[0]) >= self.apnea_TH:
                # Se actualiza la nueva columna a True en el df original
                self.df.loc[group.index,'AF_threshold']=True
                # Se genera el texto del informe
                summary += str(i)+") Se ha detenido la respiración durant
e " + str((group['date_dt'].iloc[-1] -
 group['date_dt'].iloc[0]).total_seconds()) + " segundos, desde las " + g
roup['date_dt'].iloc[0].strftime("%H:%M:%S.%f") + " hasta las " + group['
date_dt'].iloc[-1].strftime("%H:%M:%S.%f") +".\n"
```

```
                summary += "\tLa posición durante este tiempo ha sido:\n"
                for p in group['BP'].unique():
                    summary += "\t\t- " + self.dict_BP[p] + "\n"
                summary +="\tLos ronquidos han sido superiores a la media
\n" if group['SN_dig'].mean() > self.df['SN_dig'].mean() else "\tLos ronq
uidos no han sido superiores a la media\n"
                summary += "\n"
                i += 1
        summary = "Se ha detenido la respiración " + str(i-
1) + " veces durante más de " + str(pd.Timedelta(self.apnea_TH).total_sec
onds()) + " segundos.\n\n\n" + summary
        # Se escribe el texto en el PDF
        self.add_page()
        self.set_font('Times', '', 12)
        self.multi_cell( w = self.WIDTH, h = 6,txt = summary, align ='L',
 border = self.b)

    # Añade 1 página por cada png creado
    def create_report(self):
        graph_name = self.create_graph_img()
        print(graph_name)
        for graph in graph_name:
            self.add_page()
            self.image(graph, x=10, y=35, w=self.WIDTH)
```

REFERENCIAS

[1] Sanitas, «Apnea del sueño,» [En línea]. Available: https://www.sanitas.es/sanitas/seguros/es/particulares/biblioteca-de-salud/prevencion-salud/apnea-del-sueno.html.

[2] Mayo Clinic, «Sleep Apnea,» 28 Julio 2020. [En línea]. Available: https://www.mayoclinic.org/diseases-conditions/sleep-apnea/symptoms-causes/syc-20377631.

[3] A. V. Benjafield, N. T. Ayas, P. R. Eastwood, H. R., M. Ip, M. J. Morrell, C. M. Nunez, T. Penzel, S. R. Patel, T. Penzel, J. L. Pépin, P. Peppard, S. Sinha, S. Tufik, K. Valentine y A. Malhotra, «Estimation of the global prevalence and burden of obstructive sleep apnoea: a literature-based analysis,» *The Lancet Respiratory Medicine,* vol. 7, nº 8, pp. 687-698, 2019.

[4] S. Robert P., «When is ambulatory monitoring for OSA indicated ?,» 6 Agosto 2014. [En línea]. Available: https://www.slideserve.com/quanda/when-is-ambulatory-monitoring-for-osa-indicated.

[5] Hospital Universitario Quirónsalud Madrid, «¿Cómo se diagnostica la apnea del sueño?,» 08 Septiembre 2021. [En línea]. Available: https://www.quironsalud.es/hospital-madrid/es/cartera-servicios/neumologia/escuela-pacientes/sindrome-apnea-sueno/diagnostica-apnea-sueno.

[6] D. Álvarez, A. Cerezo-Hernández, A. Crespo, G. C. Gutiérrez-Tobal, V.-V. Fernando, V. Barroso-García, F. Moreno, A. Arroyo, T. Ruiz, R. Hornero y F. del Campo, «A machine learning-based test for adult sleep apnoea screening at home using oximetry and airflow,» *Scientific Reports,* vol. 10, nº 1, 2020.

[7] Libelium, «MySignals HW v2 - eHealth and Medical IoT Development Platform for Arduino,» [En línea]. Available: https://www.cooking-hacks.com/mysignals-hw-ehealth-medical-biometric-iot-platform-arduino-tutorial.html.

[8] Adafruit Industries, «Adafruit 2.4" TFT LCD with Touchscreen Breakout w/MicroSD Socket,» [En línea]. Available: https://www.adafruit.com/product/2478.

[9] «Posición Decúbito Dorsal,» [En línea]. Available: https://es.scribd.com/doc/178465772/POSICION-DECUBITO-DORSAL.

[10] Descubrearduino.com, «Arduino Uno, partes, componentes, para qué sirve y donde comprar.,» 7 Marzo 2021. [En línea]. Available: https://descubrearduino.com/arduino-uno/.

[11] Bricogeek, «Arduino UNO Rev.3,» [En línea]. Available: https://tienda.bricogeek.com/arduino/305-arduino-uno.html.

[12] L. del Valle Hernández, «Conversión de números a cadenas en Arduino,» [En línea]. Available: https://programarfacil.com/blog/arduino-blog/conversion-de-numeros-a-cadenas-en-arduino/.

[13] Python Software Foundation, «tkinter — Interface de Python para Tcl/Tk,» [En línea]. Available:

https://docs.python.org/es/3.10/library/tkinter.html.

[14] A. Suárez Lamadrid y A. Suárez Jiménez, «Fundamentos de Tkinter,» Marzo 2016. [En línea]. Available: https://python-para-impacientes.blogspot.com/p/tutorial-de-tkinter.html.

[15] Tutorials Point, «Python - GUI Programming (Tkinter),» [En línea]. Available: https://www.tutorialspoint.com/python/python_gui_programming.htm.

[16] The pandas development team, «API reference,» [En línea]. Available: https://pandas.pydata.org/docs/reference/index.html#api.

[17] The NumPy community, «NumPy Reference,» [En línea].

[18] The Matplotlib development team, «API Overview,» [En línea]. Available: https://matplotlib.org/stable/api/index.html.

[19] D. Radečić, «How to Create PDF Reports with Python - The Essential Guide,» 18 Enero 2021. [En línea]. Available: https://towardsdatascience.com/how-to-create-pdf-reports-with-python-the-essential-guide-c08dd3ebf2ee.

[20] Python Software Foundation, «threading - Paralelismo basado en hilos,» [En línea]. Available: https://docs.python.org/es/3/library/threading.html.

[21] DelfStack, «Cómo hacer que el widget Tkinter Text sea sólo de lectura,» 25 Junio 2020. [En línea]. Available: https://www.delftstack.com/es/howto/python-tkinter/how-to-make-tkinter-text-widget-read-only/.

[22] The Matplotlib development team, «Using span_where,» [En línea]. Available: https://matplotlib.org/3.1.0/gallery/lines_bars_and_markers/span_regions.html.

[23] C. Tao, «Pandas DataFrame Group by Consecutive Certain Values,» 5 Julio 2020. [En línea]. Available: https://towardsdatascience.com/pandas-dataframe-group-by-consecutive-certain-values-a6ed8e5d8cc.

[24] Python Software Foundation, «pyinstaller 4.5.1,» [En línea]. Available: https://pypi.org/project/pyinstaller/.

[25] HektorDocs, «Pyinstaller,» 9 Noviembre 2018. [En línea]. Available: https://docs.hektorprofe.net/python/distribucion/pyinstaller/.

[26] Arduino, «Downloads,» [En línea]. Available: https://www.arduino.cc/en/software.

[27] Libelium, «MySignals_HW_SDK_V2.0.2.zip,» [En línea]. Available: https://www.cooking-hacks.com/media/cooking/images/documentation/mysignals_hardware/MySignals_HW_SDK_V2.0.2.zip.

[28] Python, «All releases,» [En línea]. Available: https://www.python.org/downloads/.

[29] Python Software Foundation, «pyserial 3.5,» [En línea]. Available: https://pypi.org/project/pyserial/.

[30] Python Software Foundation, «pandas 1.3.3,» [En línea]. Available: https://pypi.org/project/pandas/.

[31] Python Software Foundation, «matplotlib 3.4.3,» [En línea]. Available: https://pypi.org/project/matplotlib/.

[32] Python Software Foundation, «fpdf 1.7.2.,» [En línea]. Available: https://pypi.org/project/fpdf/.

[33] Microsoft, «Visual Studio Code,» [En línea]. Available: https://code.visualstudio.com/.

[34] e-ika, «Cable USB tipo A/B, de 30cm,» [En línea]. Available: https://www.e-ika.com/cable-usb-tipo-ab.

[35] Freepik Company S.L., «Sleeping free icon,» [En línea]. Available: https://www.flaticon.com/free-icon/sleeping_5512388?term=sleeping%20person&page=1&position=82&page=1&position=82&related_id=5512388&origin=search.