



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

SISTEMEN INGENIERITZA ETA
AUTOMATIKA SAILA
DEPARTAMENTO DE INGENIERÍA DE
SISTEMAS Y AUTOMÁTICA

UNIVERSIDAD DEL PAÍS VASCO

ESCUELA DE INGENIERÍA DE BILBAO

BECAS FUNDACIÓN JESÚS DE GANGOITI BARRERA

Diseño y desarrollo de un sistema de apoyo a la diagnosis e identificación de patologías implementado con dispositivos de bajo coste, preciso, fiable y de uso en entornos domiciliarios, residenciales y hospitalarios

Autor:

Imanol Ayude Prieto

Supervisor:

Dr. Eloy Irigoyen Gordo

13 de marzo de 2023

AGRADECIMIENTOS

Mi más sincero agradecimiento a todas las personas que, de un modo u otro, me han ayudado en este camino, y en especial;

A mi familia, que ha sido un apoyo constante y fundamental en toda mi vida, y que, sin su ayuda, no hubiera sido posible desarrollarme, tanto personalmente como formativamente.

A mi director, Dr. Eloy Irigoyen Gordo, que, gracias a su apoyo y motivación casi diaria, así como a todo su conocimiento tanto técnico como conocimiento a la hora de gestión del trabajo, este trabajo ha llegado hasta este punto.

A todas las instituciones involucradas tanto en mi desarrollo como estudiante/investigador, así como las entidades que han ayudado a la elaboración y puesta en marcha de este proyecto. Pero, en especial, al Grupo de Investigación de Control Inteligente y a la Fundación Jesús Gangoiti Barrera.

RESUMEN

Resumen

En este proyecto se plantea el diseño de un dispositivo que sea capaz de recoger varias señales fisiológicas, como el pulso, el nivel de sudoración en piel y el electrocardiograma. Todos los datos adquiridos serán subidos a una base de datos, para que así se pueda tener un amplio historial del usuario y poder ayudar a la diagnosis de diferentes cardiopatías.

Laburpena

Proiektu honetan, hainbat seinale fisiologiko jasotzeko gai izango den gailu bat diseinatzea planteatzen da, hala nola pultsua, azalean izerditzeko maila eta elektrokardiograma. Eskuratutako datu guztiak datu-base batera igoko dira, horrela erabiltzailearen historia zabala eduki ahal izateko eta hainbat kardiopatia diagnostikatzen lagundu ahal izateko.

Summary

This project proposes the design of a device capable of collecting various physiological signals, such as the pulse, the level of perspiration on the skin and the electrocardiogram. All the data acquired will be uploaded to a database, in order to have a comprehensive history of the user and to be able to help in the diagnosis of different heart diseases.

Palabras clave: Pseudo Tiempo Real, IoT, Bioseñales, ESP32, Python

INDICE DE CONTENIDOS

Contenido

1 INTRODUCCIÓN	2
1.1 Motivación	2
1.2 Antecedentes.....	3
2 OBJETIVOS Y ALCANCE.....	6
2.1 Objetivos principales	6
2.2 Objetivos parciales.....	6
2.3 Alcance.....	6
2.3.1 Ámbito tecnológico	6
2.3.2 Ámbito socio sanitario.....	7
2.4 Estructura	7
3 ESTADO DEL ARTE	10
3.1 Sistema circulatorio	10
3.2 Sistema nervioso.....	12
3.3 Dispositivos anteriores	12
4 DISPOSITIVOS Y TECNOLOGÍAS	14
4.1 Dispositivos.....	14
4.1.1 BITalino	14
4.1.2 Arduino	14
4.1.3 ESP32	15
4.1.4 Raspberry Pi.....	15
4.2 Tecnologías	16
4.2.1 Pulso y SPO2	16
4.2.2 EDA/GSR	18
4.2.3 ECG	18
5 METODOLOGÍA	22
5.1 Creación de una Base de Datos	22
5.1.1 Configuración del adaptador USB	22
5.1.2 Configuración del Router.....	23
5.1.3 Configuración de la Máquina Virtual.....	24
5.1.4 Instalación y configuración de PostgreSQL	25
5.1.5 Creación de la base de datos.....	28

5.2	Raspberry Pi	30
5.2.1	Instalación de un OS.....	30
5.2.2	Configuración Raspberry.....	31
5.2.3	Programación Python	32
5.3	ESP32.....	44
5.3.1	Lenguajes de programación.....	44
5.3.2	Instalación y configuración del IDE-Arduino	44
5.3.3	Programación en Arduino	46
5.3.4	Servidor MQTT Raspberry Pi.....	51
5.4	Montaje hardware	55
5.4.1	Conexión Max30102	55
5.4.2	Conección medición GSR.....	56
5.4.3	Montaje completo	58
5.5	Grafana.....	58
6	RESULTADOS	64
6.1	Raspberry Pi	64
6.1.1	Primera versión del programa	64
6.1.2	Segunda versión del programa	69
6.2	ESP32.....	71
6.2.1	Pulsioxímetro	71
6.2.2	GSR	72
6.2.3	ECG	73
6.2.4	Implementación completa.....	73
7	CONCLUSIONES.....	76
7.1	Solución Raspberry PI.....	76
7.2	Solución Esp32	76
7.3	Conclusiones del proyecto	76
7.4	Acciones futuras.....	76
8	REFERENCIAS BIBLIOGRÁFICAS.....	80

INDICE DE FIGURAS

INDICE DE FIGURAS

Figura 3.1.- Esquema de circulación sanguínea	10
Figura 3.2.- Esquema de la inervación autónoma del corazón	11
Figura 3.3.- Diagrama de Wiggers, muestra los eventos durante el periodo cardiaco	11
Figura 4.1.- BITalino (r)evolution Board	14
Figura 4.2.- Arduino nano 33 IoT	14
Figura 4.3.- <i>ESP32</i>	15
Figura 4.4.- Raspberry Pi Zero 2W	15
Figura 4.5.- Raspberry Pi Model 4B.....	16
Figura 4.6.- Observación de la refracción de la luz en la <i>Hb</i> y la <i>HbO₂</i>	17
Figura 4.7.- Tipos de electrodos según su contacto.....	18
Figura 4.8.- Circuito AD8232	19
Figura 5.1.- Dispositivos TL-WN725N (USB) y TL-WR841ND (Router)	22
Figura 5.2.- Instalador de drivers del TL-WN725N	22
Figura 5.3.- Leds encendidos para la configuración del TL-WR841ND.....	23
Figura 5.4.- Puertos LAN	23
Figura 5.5.- Solicitud de usuario y contraseña configuración TL-WN725N.....	23
Figura 5.6.- Configuración de Wireless Settings.....	23
Figura 5.7.- Configuración de Wireless Security	24
Figura 5.8.- Configuración de Network adapter (VMware)	24
Figura 5.9.- Configure Adapters (VMWare)	25
Figura 5.10.- Autoridad y permisos de Data, server.key y server.crt (root)	26
Figura 5.11.- Autoridad y permisos de Data, server.key y server.crt (postgres)	27
Figura 5.12.- Cambio escucha todas las IP (postgresql.conf).....	27
Figura 5.13.- Activación y actualización de localización certificados (postgresql.conf)	27
Figura 5.14.- Whitelist de direcciones IP (pg_hba.conf).....	28
Figura 5.15.- Listado de usuarios	29
Figura 5.16.- Listado y visualización de tabla.....	29
Figura 5.17.- Ventana Advanced Options	30
Figura 5.18.- Esquema de I2C	31
Figura 5.19.- Ventana inicial Raspi-config	31
Figura 5.20.- Ventana habilitación <i>interfaces</i>	31
Figura 5.21.- Logo de Python.....	32
Figura 5.22.- Diagrama multithreading	32
Figura 5.23.- Diagrama Multiprocessing.....	33
Figura 5.24.- Diagrama de activación de una Tarea Periódica.....	33
Figura 5.25.- Registro Protegido	34

Figura 5.26.- Tarea periódica con plazo	35
Figura 5.27.- Tarea periódica simplificada	36
Figura 5.28.- Diagrama de comunicación – Primer programa	37
Figura 5.29.- Función CogerGuardarDatosPOX.....	37
Figura 5.30.- Cálculo del valor GSR.....	38
Figura 5.31.- Función CogerGuardarDatosGSR.....	38
Figura 5.32.- Función COMS	39
Figura 5.33.- Estructura Programa principal (V1 Python).....	39
Figura 5.34.- Estructura tipo usada en el segundo programa	40
Figura 5.35.- Esquema estructura programa total	40
Figura 5.36.- Función generar_buffer_pox	40
Figura 5.37.- Función generar_buffer_gsr	41
Figura 5.38.- Función generar_buffer_ecg	41
Figura 5.39.- Main del programa principal	42
Figura 5.40.- Programa obtener_valores.....	43
Figura 5.41.- Programa connect_bbdd.....	43
Figura 5.42.- Función tipo insertar en bbdd.....	43
Figura 5.43.- Apertura del gestor de tarjetas.....	45
Figura 5.44.- Búsqueda en el gestor de tarjetas	45
Figura 5.45.- Selección de <i>ESP32 Dev Module</i>	46
Figura 5.46.- Logo de Free Real Time Operative Sistem	46
Figura 5.47.- Estructura de Tareas - Programa ESP32	47
Figura 5.48.-Diagrama de flujo del programa en ESP32	47
Figura 5.49.- Valores configuración MAX30102	48
Figura 5.50.- Diagrama de flujo de tarea periódica en ESP32	49
Figura 5.51.- Estructura de datos ESP32	50
Figura 5.52.- Pantalla inicial <i>Node-Red</i>	52
Figura 5.53.- Flujo programado en <i>Node-Red</i>	52
Figura 5.54.- Configuración de nodos	52
Figura 5.55.- Diagrama de flujo funciones Separador.py	53
Figura 5.56.- Decodificado de estructura de datos.....	53
Figura 5.57.- Diagrama de flujo función FiltroRuido.py	54
Figura 5.58.- Diagrama de flujo tarea periódica subida de datos.....	54
Figura 5.59.- Montaje MAX30102 en Raspberry Pi.....	55
Figura 5.60.- Montaje MAX30102 en ESP32.....	56
Figura 5.61.- Amplificador operacional - No inversor	56
Figura 5.62.- Grove Base Hat for Raspberry Pi	57

Figura 5.63.- Montaje del sensor GSR – Raspberry Pi	57
Figura 5.64.- Montaje del sensor GSR – ESP32	57
Figura 5.65.- Montaje completo en Raspberry Pi y ESP32	58
Figura 5.66.- Logo Grafana	59
Figura 5.67.- Interface inicial de Grafana.....	59
Figura 5.68.- Conexión <i>Grafana</i> y base de datos (I)	60
Figura 5.69.- Conexión <i>Grafana</i> y base de datos (II).....	60
Figura 5.70.- Configuración <i>Grafana</i> (Vinculación con <i>BBDD</i>)	60
Figura 5.71.- Proceso añadir un nuevo panel	61
Figura 5.72.- Dashboard generado	61
Figura 6.1.- Pulsioxímetro (1 Hz) - Raspberry Pi.....	64
Figura 6.2.- Pulsioxímetro (10 Hz) – Raspberry Pi.....	65
Figura 6.3.- Pulsioxímetro (50 Hz) - Raspberry Pi.....	65
Figura 6.4.- GSR (50 Hz) - Raspberry Pi	66
Figura 6.5.- GSR (10 Hz) - Raspberry Pi	67
Figura 6.6.- GSR (1 Hz) - Raspberry Pi	67
Figura 6.7.- Cambios introducidos en el programa (RPi - V1)	68
Figura 6.8.- Raspberry Pi - Versión 1.....	68
Figura 6.9.- Ejemplo perdida de una o más muestras.....	69
Figura 6.10.- Cambios introducidos en el programa (RPi – V2).....	69
Figura 6.11.- GSR (1 Hz) - Raspberry Pi (V2).....	70
Figura 6.12.- Figura 6.13.- Pulsioxímetro (2.5 Hz) – Raspberry Pi (v2).....	70
Figura 6.14.- POX.csv un dato por conexión	71
Figura 6.15.- POX.csv cinco datos por conexión	71
Figura 6.16.- Recogida de señal <i>POX</i> constante durante una hora	72
Figura 6.17.- Recogida de señal <i>GSR</i> constante durante una hora	72
Figura 6.18.- Recogida de señal <i>ECG</i> constante durante una hora	73
Figura 6.19.- Grafica de datos recogidos (5 minutos)	74

INDICE DE TABLAS

Tabla 4.1.- Especificaciones generales de placas buscadas.....	16
Tabla 4.2.- Comparativa entre Max30100 y Max30102	17
Tabla 5.1.- Comparación de lenguajes de programación	44
Tabla 6.1.- Frecuencias de muestreo evaluación teórica	68
Tabla 6.2.- Frecuencias de muestreo - Segunda versión	69
Tabla 6.3.- Frecuencias de muestreo final.....	69
Tabla 6.4.- Características de las tareas ESP32.....	73

CAPITULO 1

INTRODUCCIÓN

1 Introducción

1.1 Motivación

A día de hoy, que ya han pasado dos años del comienzo de la pandemia de *COVID-19*, la situación sanitaria sigue estando afectada por esta circunstancia. Pero, no solo se ven afectadas los y las sanitarias y otros trabajadores vinculados al sector socio sanitario, sino que también sigue estando afectada la forma en la que los usuarios de estos servicios, en definitiva, pacientes son atendidos, cuidados y curados.

Una de las grandes causas que ha provocado este efecto negativo sobre el sector, a parte, de que se trataba de una enfermedad desconocida, fue también la falta de recursos, ya no solo humanos (faltaba personal en el sector de la salud) sino que también de material sanitario. Y debido a esta situación, se ha sacado a la palestra la ya obvia necesidad de sistemas que apoyen a la labor sanitaria.

Desde ya hace unos años, la tecnología de *Internet of the Things* se ha desarrollado exponencialmente. Esto ha provocado que, junto a la mejora de la potencia de los microprocesadores la creación de dispositivos que ayuden a la diagnosis prolifere. Mediante este tipo de herramientas se amplía el volumen de gente a la que se puede atender, al mismo tiempo que se ofrece una mejor calidad en esta atención debido a la información que se ha ido recogiendo gracias a estas herramientas.

Pero fuera de esta situación, que lo único que ha hecho ha sido visibilizar el estado del sector socio sanitario, existen otros tipos de situaciones en las que la diagnosis de patologías comunes como pueden ser las cardiovasculares, no se ven atendidas. Y según la OMS, este tipo de patologías son las que más muertes causan por encima de todas. Sobre todo, se destaca que un alto porcentaje de estas muertes se dan en países subdesarrollados a causa de infartos.

Pero según diferentes libros y estudios, una parte de estos infartos pueden ser previamente detectados, ya que suelen estar precedidos por eventos ventriculares mediables mediante diferentes sensores.

Todos estos hechos, junto a la motivación del propio becario y la continuación de la línea de investigación del *Grupo de Investigación en Control Inteligente*, se ha permitido llevar a cabo este proyecto que estará centrado en la elaboración de un dispositivo de adquisición fiable y en tiempo real de diferentes señales biológicas, que caractericen tanto los periodos ventriculares que pueden denotar una cardiopatía futura, como señales que puedan alertar al médico de otro tipo de patologías. El valor de este proyecto, se encontrará en los algoritmos y diseños electrónicos que se elaborarán para la creación de este dispositivo.

1.2 Antecedentes

Como se ha comentado anteriormente, este proyecto se desarrolla en la *Universidad del País Vasco*, para ser concreto dentro del *Grupo de Investigación en Control Inteligente*, de la *Escuela de Ingeniería de Bilbao*. Este grupo de investigación, pertenece al departamento de *Ingeniería de Sistemas y Automatización* en el cual se busca dar soluciones a problemas complejos mediante la aplicación de sistemas de computación inteligente. El director de este proyecto es el Dr. Eloy Irigoyen Gordo, el cual tiene ya una gran experiencia en la dirección de proyectos centrados tanto en la computación inteligente como de toma correcta de señales para el correcto funcionamiento y diseño de estos. Este proyecto se encuentra dentro del campo de la bio-ingeniería, en el cual este grupo de investigación lleva varios años centrado. Dentro de los proyectos elaborados en el grupo, la que más destaca es la Tesis elaborada por la Dra. Raquel Martínez Rodríguez con el título, “*Diseño de un sistema de detección y clasificación de cambios emocionales basado en el análisis fisiológicas no invasivas*”[1] en la cual se estudiaba desde el año 2006 cómo se podía gestionar las señales fisiológicas para detectar alteraciones emocionales que pueda sufrir el paciente, así como poder clasificar esos estados de ánimos para que después puedan ser usados para conocer el estado emocional de cara a por ejemplo, a hacer una intervención quirúrgica. Continuando la línea de investigación abierta por la Dra. Raquel Martínez Rodríguez, el Dr. Unai Zalabarria Pena realizó su tesis cuyo título es “*Identificación del nivel de estrés y relajación en personas basadas en el estudio y procesamiento avanzado de señales fisiológicas relacionadas con la actividad del sistema nervioso autónomo*” [2], con la que se dio otro paso más en la línea de investigación implementando técnicas de inteligencia artificial, así como procesamientos avanzados sobre las señales fisiológicas.

Esta tesis supuso en el grupo la apertura de nuevas líneas de investigación para la obtención, procesado y detección de patologías cardiovasculares. Y centrándose en el procesado para la detección de estas, de nuevo, el Dr. Unai Zalabarria Pena, publicó en la revista llamada *Applied Mathematics and Computation* en artículo de nombre “*Online robust R-peaks detection in noisy electrocardiograms using a novel iterative smart processing algorithm*”[3] que ayudó a iniciar en GICI el estudio e investigación para la extracción robusta y segura de patrones en las señales fisiológicas provenientes de señales características del sistema cardiovascular, como puede ser el electrocardiograma.

Estas señales que derivan del sistema cardiovascular fueron posteriormente estudiadas junto a otras señales como puede ser la sudoración o la respiración para así poder lograr mediante algoritmos inteligentes los parámetros de estas señales fisiológicas, con el fin de que pudieran ser procesados en un dispositivo de bajo costo. Esta investigación finalmente derivó en la publicación del artículo “*A low-cost portable solution for stress and relaxation estimation based on a real-time fuzzy algorithm*” [4] en la revista *IEEE Access*.

El siguiente paso que se dio en la línea de investigación fue de nuevo un estudio encabezado por el Dr. Unai Zalabarria, junto a varios investigadores pertenecientes al grupo de investigación GICI y al Instituto de Tecnologías Biomédicas de la universidad de Tecnológica de Auckland, Nueva Zelanda. En este desarrollo se buscaba la validación de las técnicas desarrolladas en el trabajo anterior, además de realizar un profundo análisis de la vinculación entre la señal pletismográfica y diferentes patologías cardiovasculares. Finalmente, en el año 2020 se publicó un nuevo artículo en la prestigiosa revista *Computer Methods and Programs in Biomedicine* bajo el título “*Diagnosis of atrial fibrillation based on arterial pulse wave foot print detection using artifical neural network*” [5]. En dicho artículo se presentaba la validación de las técnicas inteligentes para la predicción de eventos anómalos o patologías cardiovasculares mediante el correcto procesamiento de los parámetros físicos extraídos de señales como, el ritmo cardíaco o de la presión arterial.

Y siguiendo estos pasos ya dados en la línea de investigación, el trabajo que se ha desarrollado en esta beca, nace del interés del beneficiario por la correcta obtención de las señales fisiológicas, para así ofrecer un valor muy elevado de datos para poder seguir desarrollando los algoritmos y técnicas inteligentes de computación para la detección de patologías. Además, se quiere implementar esta

adquisición de datos en un dispositivo portable y de bajo costo, pero sin perder en ningún momento la seguridad que es necesaria para trabajar con estas señales.

Como se ha comentado en cada uno de los trabajos, tesis o artículo, las señales fisiológicas más usadas para la detección de estas patologías, son la señal de pulso, oxigenación de sangre, y respuesta galvánica de la piel. Por lo que, en este trabajo se quiere que la toma de todas estas señales sean recogidas en los períodos adecuados para la recogida correcta de cada una de las señales, así como que todas ellas sean recogidas de forma fiable.

CAPITULO 2

OBJETIVOS Y ALCANCE

2 Objetivos y alcance

2.1 Objetivos principales

El principal objetivo de este trabajo se centra en la creación de un dispositivo de bajo costo, que sea capaz de recoger diferentes señales fisiológicas de forma fiable, o al menos, garantizando la recogida continuada. Las señales que se quieren adquirir deberán ser de la manera menos invasiva posible, para que así, el paciente no se vea afectado y las medidas recogidas sean lo más precisas posible. Además, se quiere hacer que el dispositivo aparte de ser de bajo costo, también sea lo más portable posible, ya sea mediante una pulsera, o mediante una petaca en el cinto.

Este planteamiento, hace que el sistema pueda ser usado tanto en el sector sanitario (hospitales, residencias) como para uso doméstico. De este modo, se pretende poder hacer una evaluación efectiva de mayor número de gente, usando un menor número de recursos. O simplemente, creando un archivo de datos sobre las señales fisiológicas del paciente, para que después en un menor tiempo puedan ser leídas y mostradas al médico interesado para así poder hacer diagnosticar si existe alguna patología.

Dado que el proyecto que se va a llevar a cabo, trata de obtener y manipular datos personales altamente confidenciales, y estos pueden servir para la identificación del usuario, el trabajo se atenderá a los principios éticos que se pidan en él.

2.2 Objetivos parciales

Para lograr los objetivos principales en este trabajo se han definido los siguientes objetivos parciales:

- Búsqueda bibliográfica y estudio en profundidad del estado del arte. En este se estudiará la fisionomía humana para así poder relacionar qué señales fisiológicas son necesarias para la detección de patologías
- Desarrollo de algoritmos para la lectura de las señales fisiológicas de forma fiable.
- Creación y configuración de un servidor para el correcto almacenamiento de las señales fisiológicas.
- Comunicación segura entre el dispositivo de adquisición de datos y el servidor.
- Testeo del prototipo final mediante la ejecución de este en condiciones de funcionamiento reales.

2.3 Alcance

Debido a que el trabajo que se va a desarrollar por parte del beneficiario trata de aplicar una solución tecnológica a un problema sanitario como puede ser la identificación anticipada de una posible patología en el sistema cardiovascular, el proyecto supone un tema de gran interés en ambos ámbitos.

2.3.1 Ámbito tecnológico

- El desarrollo de una solución avanzada para la recogida autónoma de señales fisiológicas fiables en cuanto a cumplimiento de plazo supondría un hito interesante en este ámbito, ya que,

mediante este dispositivo, se podrá abastecer a diferentes sistemas de clasificación de patologías cardiovasculares de información suficiente para el entrenamiento y validación de ellos.

- El uso de una programación en Pseudo tiempo real a la hora de programar diferentes microcontroladores para la obtención de varias señales fisiológicas de forma simultánea. Lo que supone ser de los primeros sistemas en implementar este tipo de programación para el ámbito socio-sanitario en placas de bajo-costo.
- Creación e implementación de una biblioteca con tareas periódicas basadas en tiempo real para el lenguaje de programación *Python*, desplegable en varios sistemas como puede ser la *Raspberry Pi 4B+*. Con la que se podría usar unos dispositivos de mejores especificaciones para la continua mejora del dispositivo.

2.3.2 Ámbito socio sanitario

- Diseño de un dispositivo portátil para la obtención de datos para la identificación de patologías cardiovasculares y estados de ánimo, lo que supone mejorar la calidad de vida de los usuarios, ya que mediante este se podrá detectar de forma preventiva estos estados o patologías.
- El dispositivo no solo podrá ser usado por un usuario en específico, si no que este podría dar lugar a una implementación en el ámbito sanitario, ya sea en una planta de hospital o en las residencias. Lo que ayudaría a lograr una diagnosis precisa de una forma más rápida.

2.4 Estructura

A continuación, se presenta la estructura del proyecto, explicando de forma general el contenido que se expondrá en cada uno de los apartados.

El Capítulo 3 es el estado del arte, en este se lleva a cabo un estudio y descripción general de las modalidades entre las que cabalga este proyecto, el sistema cardiovascular, sistema nervioso y la programación en pseudo tiempo real.

En el Capítulo 4 se explica los dispositivos y tecnologías normalmente usados en proyectos similares a este. Además, se compararán para determinar cuál convendría ser usado en este trabajo.

El Capítulo 5 estará compuesto por la metodología, en la que se explica todos los pasos dados tanto en el aspecto de programación como en el aspecto de diseño hardware. En él también, se comentará la creación de una base de datos y como los datos de esta pueden ser visualizados.

En el penúltimo capítulo, el 6, se mostrarán los resultados obtenidos en cada una de las fases de desarrollo de los dispositivos.

Y finalmente, en el Capítulo 7 se discuten las conclusiones alcanzadas en torno a los resultados obtenidos en el capítulo anterior. También se incluirá las líneas futuras que este proyecto ha abierto y las que se podría trabajar considerando el gran potencial que podrían tener estas futuras contribuciones.

CAPITULO 3

ESTADO DEL ARTE

3 Estado del arte

Como ya se comentó en el capítulo 1.1, la situación sanitaria a día de hoy no es de las mejores, debido a la pandemia, y sus efectos que se siguen observando actualmente. Pero si se habla de otras enfermedades que tienen una gran mortalidad, son las concernientes al sistema cardiovascular, que generalmente comienzan con cardiopatías.

Desde el año 2020 se estima que al menos el 50% de la población mayor de 60 años tiene una cardiopatía. De estas, el 3% de ellas podrían sufrir un ataque al corazón que resulte con la muerte prematura de la persona [6]. Este valor puede parecer muy bajo, pero teniendo en cuenta que la población mundial actual es de 8.000 millones de personas, aunque sea un porcentaje bajo, el número de personas que podrían fallecer debido a este tipo de cardiopatías es muy grande.

Para conocer cómo se puede detectar este tipo de cardiopatías, en primer lugar, es necesario conocer de una forma más general el funcionamiento del sistema circulatorio, a veces también denominado como sistema cardiovascular, ya que es en este donde se forman las cardiopatías.

El sistema circulatorio es esencial para la vida. Es el encargado de transportar oxígeno y nutrientes a todas las células y órganos, mediante el torrente sanguíneo. Debido a este transporte de los nutrientes, el sistema es capaz de combatir las enfermedades y mantener la homeostasis. Está formado por el corazón, que se trata de un músculo cardíaco que tiene varias diferencias con los músculos esqueléticos, y por un sistema de vasos sanguíneos cerrados llamados, venas, arterias y capilares. Las arterias son las encargadas de transportar la sangre ya oxigenada por todo el cuerpo, a diferencia de las venas, las cuales trasportan la sangre desoxigenada del cuerpo de nuevo al corazón para su reoxigenación. El corazón proporciona un flujo constante de sangre repleta de oxigenación, la cual pasa inicialmente por los pulmones (para abastecerse de oxígeno) y después por todo el cuerpo, como se puede observar en la Figura 3.1[7].

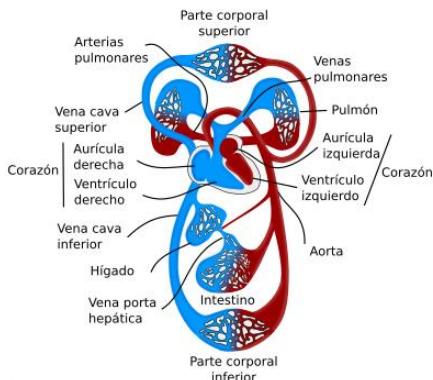


Figura 3.1.- Esquema de circulación sanguínea

Fuente: https://mmejias.webs.uvigo.es/2-organos-a/guiada_o_a_05cardiovascular.php

3.1 Sistema circulatorio

El sistema cardiovascular tiene un papel vital en el correcto mantenimiento de la homeostasis, la cual depende directamente del flujo continuo de sangre a través de la red de miles de kilómetros de capilares que componen todo el cuerpo humano [8]. Existe un gran número de mecanismos para autorregular al sistema cardiovascular, pero están todos asociados al sistema nervioso autónomo y al sistema endocrino, el cual gestiona toda la creación y distribución de hormonas [7].

Además, el sistema circulatorio se encuentra bajo el control del sistema nervioso el cual está compuesto por dos ramas diferentes, la rama simpática, y la rama parasimpática [7]. Estas son las encargadas de controlar el periodo cardíaco, la dilatación de los vasos sanguíneos y la fuerza de contracción y constrictión del propio corazón [8].

El sistema nervioso parasimpático es el encargado de la reducción del pulso cardíaco mediante la segregación de *Acetilcloina*, y el sistema nervioso simpático es el encargado del incremento de la frecuencia cardíaca mediante la segregación de *Norepinefrina*. A todo este proceso de excitación y relajación del pulso cardíaco se le llama *inervación* del corazón el cual se muestra en la Figura 3.2.

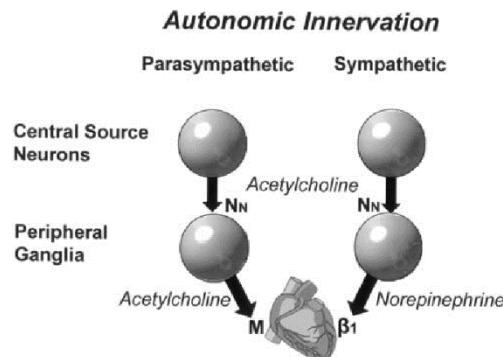


Figura 3.2.- Esquema de la inervación autónoma del corazón

Fuente: *Handbook of Psychophysiology* [7]

Para evaluar el estado del sistema cardiaco se utilizan una gran cantidad de medidas, como pueden ser, el periodo cardíaco, el fujo sanguíneo, la resistencias vascular y el gasto cardíaco [7]. Uno de los más utilizados y que es conocido por la mayoría del mundo, es el periodo cardíaco, mediente el cual se puede estimar no solo el comportamiento del sistema cardiaco, si no que también se puede estimar la actividad del sistema nervioso simpático y parasimpático.

El periodo cardíaco se define como el funcionamiento del corazón entre un latido y el siguiente. Este proceso consta de dos períodos diferentes:

- **Diástole:** cuando el corazón se llena de sangre.
- **Sístole:** cuando el corazón se vacía de sangre debido al bombeo de ella.

En la Figura 3.3 se muestra una gráfica en la cual se representa el diagrama de Wiggers que muestra la presión que hace un corazón sano en la Sístole y en la Diástole.

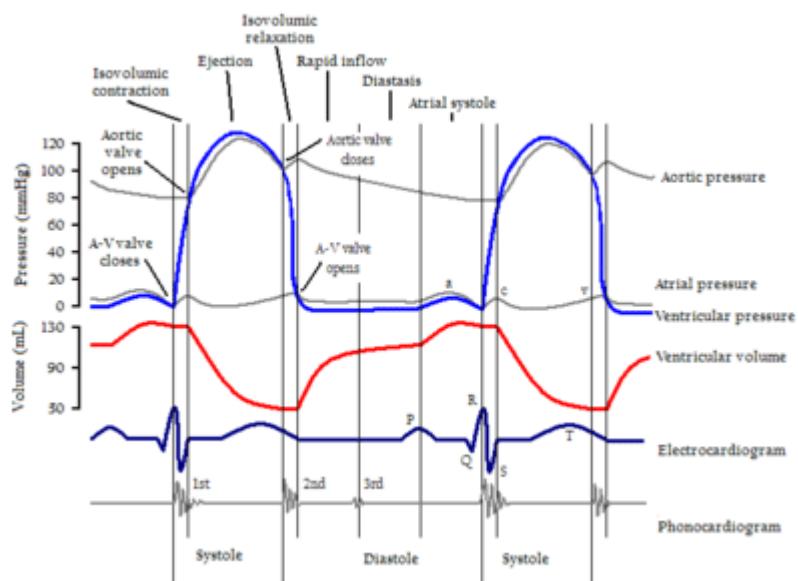


Figura 3.3.- Diagrama de Wiggers, muestra los eventos durante el periodo cardíaco

Fuente: <https://www.aastweb.org/blog/join-us-at-the-aast-fall-course-for-jon-atkinsons-talk-on-cardiac-events>

3.2 Sistema nervioso

Como se acaba de comentar, el sistema nervioso también actúa en el sistema cardiovascular abasteciendo de diferentes sustancias químicas para provocar que el corazón palpite a diferentes frecuencias dependiendo de la situación. Pero este sistema, no solo actúa sobre el cardíaco, si no que actúa sobre todo el cuerpo humano, por lo que existen gran variedad de señales biológicas para la detección del funcionamiento de este.

Uno de los más fáciles de medir y menos invasivo, es la *actividad electro dermal*, ya que es completamente involuntaria y generalmente no puede ser manipulable. Esta actividad se presenta en el cuerpo humano como alteración de la humedad de la dermis, ya sea porque se comienza a sudar en una situación comprometida, o se deja de hacerlo por estar relajado [7].

3.3 Dispositivos anteriores

Con todo esto, existen gran variedad de dispositivos que son capaces de medir señales de los sistemas cardiovascular y nervioso para así poder observar su estado. Estos dispositivos se pueden agrupar en dos ramas diferentes, por un lado, los sistemas profesionales y, por otro lado, los sistemas portables. Esta última rama de dispositivos, se centra en la detección precoz de las cardiopatías, y su característica principal, es que son portables y pueden ser transportados con facilidad. Para dar esa portabilidad al dispositivo generalmente, se suele pensar en colocarlos en la muñeca como si fuera una pulsera [9], o incluso en una camiseta que el usuario pueda quitar y ponérsela cada vez que se quiera hacer uso del dispositivo [10].

Pero este tipo de dispositivos al estar en desarrollo o ser de grandes empresas tecnológicas tiene sus aspectos positivos y sus aspectos negativos, entre los que se encuentra el alto precio que suelen costar los dispositivos de mayor calidad para la detección de estas señales fisiológicas. Ya que para poder obtener una medida de calidad es necesario tener en cuenta aspectos como las no-idealidades de las señales, por ejemplo, del *SPO₂* [11] y la del *Pulsioxímetro* [12], la seguridad con la que estos gestionan los datos que se van recogiendo periódicamente [13], o la calidad de la toma de datos, en cuestión de tiempo-real [14].

Pero este proyecto se quiere centrar en el desarrollo preliminar de un dispositivo que mejore estos aspectos negativos y explorar otras tecnologías. Como por ejemplo utilizar la comunicación vía WIFI, en vez de la comúnmente usada Bluetooth (como se hace en [13]). De esta forma se puede introducir mayor seguridad introduciendo certificados y restricciones de IP como se comenta en [15]. Además, se quiere implementar circuitos propios para el acondicionamiento de señal como se hace en [16], en vez de usar el sensor comercial AD8232 como se hace en la mayoría de dispositivos [17].

Y siguiendo la estructura de bloques que suele ser usada [18], y el posicionamiento habitual de los sensores como se hacen en [9], trabajar en un sistema garantista de plazos, logrando que funcione en Pseudo Tiempo Real usando *MultiThreading* o *Freertos* como se hace en otro tipo de dispositivos[19], [20].

Todo esto quiere ser implementado en otro tipo de tarjetas como puede ser una *Raspberry PI*, que se trata de un ordenador en miniatura con una gran potencia de cálculo, o una *ESP32*, que está más centrada en la toma de datos de una forma más rápida y constante.

CAPITULO 4

DISPOSITIVOS Y TECNOLOGÍAS

4 Dispositivos y tecnologías

En este apartado se resumirán las tecnologías y dispositivos disponibles para la adquisición de señales fisiológicas. Los dispositivos serán placas que tengan la posibilidad de comunicarse con otros dispositivos sin ningún tipo de cable (*WiFi*, *Bluetooth* ...).

4.1 Dispositivos

Existen gran variedad de dispositivos que pueden ser usados para la adquisición de señales fisiológicas, desde placas dedicadas para la adquisición de este tipo de señales, hasta dispositivos profesionales. Tras hacer un estudio de los posibles dispositivos que podían ser usados en este proyecto, se obtuvo una lista de cuatro dispositivos de marcas diferentes.

4.1.1 BITalino

La primera de las placas encontradas, es la *BITalino (r)evolution Board* (Figura 4.1), que es usada en diferentes proyectos [21], [22]. Esta placa está diseñada para la única función de adquirir señales fisiológicas, lo que hace que la capacidad de comunicación con una *Base de Datos* o con algún dispositivo externo se haga solamente por medio de *Bluetooth*, lo que limita en gran medida su uso, por la inseguridad de esa vía. Además, si se observa la Tabla 4.1 destaca frente a otros dispositivos en su tamaño, peso y precio, y su único núcleo en el procesador. Debido a este motivo, la capacidad de hacer que el sistema sea garantista de plazo es menor, ya que no se podrá independizar las tareas en diferentes núcleos. Por todo ello, el uso de este dispositivo para el proyecto fue descartado.

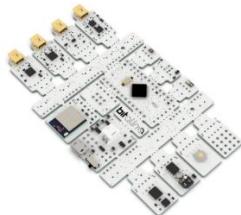


Figura 4.1.- BITalino (r)evolution Board

Fuente: <https://www.pluxbiosignals.com/collections/teaching-kits/products/bitalino-revolution-board-kit-ble-bt>

4.1.2 Arduino

A continuación, se observó que el fabricante más usado para crear dispositivos de adquisición de datos de todo tipo era *Arduino* [23], [24]. Observando el catálogo de esta compañía se llegó a la conclusión de que la placa que más podía concordar con el proyecto que se tenía en mente, era la *Arduino nano 33 IoT* (Figura 4.2), la cual destaca por su tamaño y peso muy reducidos. Además, se tiene la vía de comunicación *WiFi* (como se recoge en la Tabla 4.1), para poder comunicarse con dispositivos externos, y entradas tanto analógicas, para la recogida de señales como el *EDA* (Actividad Electrodérmica) y el *ECG*, como entradas y salida digitales, para recoger la señal del pulsioxímetro.



Figura 4.2.- Arduino nano 33 IoT

Fuente: <https://docs.arduino.cc/hardware/nano-33-iot>

Pero como sucedía en el dispositivo de *BITalino*, la especificación limitante del dispositivo es el procesador. Ya que, de nuevo, se trata de un procesador mono-núcleo, lo que como se ha comentado anteriormente, hace imposible que el dispositivo final sea garantista de plazos con las tres tareas de adquisición de datos y de comunicación. Por lo que se llegó a la conclusión de nuevo, de que esta placa no podía ser usada para el proyecto que se quería llevar a cabo.

4.1.3 ESP32

Continuando con la labor de investigación, se observó que en la línea de las placas de *Arduino*, la empresa *Espressif* había creado una placa de tamaño, peso y precio reducido para proyectos de dispositivos de recogida y gestión de datos, *ESP32* (Figura 4.3). Además de que en el ámbito de investigación estaba siendo usada en diferentes proyectos también [25], [26], pero en este caso, no en el campo de la recogida de señales fisiológicas.



Figura 4.3.- *ESP32*

Fuente: <https://www.espressif.com/en/products/socs/esp32>

Como se puede apreciar en la Tabla 4.1, el dispositivo es capaz de comunicarse con otros dispositivos vía *WiFi* y *Bluetooth*, lo que como se ha comentado anteriormente permite la comunicación segura que se quería crear en el dispositivo final. Además, observando las características del procesador, que fueron las responsables de descartes de placas anteriores, en este caso, se aprecia que este procesador está compuesto por dos núcleos (*DualCore*) independientemente programables. Con lo que se podría ejecutar las tareas de comunicación y adquisición de datos en núcleos diferentes para lograr que el dispositivo fuese garantista de plazos. Por todo esto, esta placa fue una de las elegidas para la elaboración del dispositivo final.

4.1.4 Raspberry Pi

Finalmente se observó que existían ciertas investigaciones donde la labor de adquisición de datos con un periodo alto se hacía con los famosos ordenadores en miniatura del fabricante *Raspberry Pi*. En este caso, al no tratarse de una placa, no solo se podría llevar a cabo la adquisición fiable de las señales fisiológicas, si no que se podrían implementar otros elementos de hardware como pueden ser pantallas, y altavoces.

El ordenador de este fabricante que más encajaba en este proyecto era la *Raspberry Pi Zero 2W* (Figura 4.4) atendiendo a las características mostradas en la Tabla 4.1. Donde destacaba no solo el tamaño, peso y precio, para las opciones que esta ofrece. Sino que cumplía también las restricciones tanto de vías de comunicación (*WiFi* y *Bluetooth*), como de número de núcleos en el procesador. La gran duda que quedaba por resolver era si era posible que un micro-ordenador fuera garantista de plazos en la recogida de datos. Por lo que este dispositivo fue también seleccionado para resolver esta cuestión y ver si se podía desarrollar el proyecto con este ordenador.



Figura 4.4.- *Raspberry Pi Zero 2W*

Pero debido al estado actual del mercado de componentes electrónicos lograr un dispositivo de este modelo era completamente imposible (debido al alza de los precios y los grandes tiempos de envío), se decidió aprovechar material del que ya se disponía, un ordenador en miniatura de este mismo fabricante *Raspberry Pi Model 4B*. Este modelo de ordenador tenía como gran diferencia respecto al seleccionado, el tamaño, peso y precio, que eran superiores. Pero en cuanto a especificaciones del procesador y vías de comunicaciones eran muy similares.



Figura 4.5.- Raspberry Pi Model 4B

Por lo que se decidió trabajar en este ordenador y una vez se tuviera una respuesta a la incógnita de si este dispositivo era capaz de garantizar los plazos en la adquisición de datos, migrar todos los programas y configuraciones a la placa originalmente seleccionada (*Raspberry Pi Zero 2W*).

Tabla 4.1.- Especificaciones generales de placas buscadas

	Placa	Especificaciones						Precio (€)			
		Tamaño (cm)	Peso (g)	Puertos Analógicos	Puertos Digitales	Tipo de Comunicación	Procesador				
							Modelo	nº de Núcleos	Velocidad (MHz)		
Bitalino	BITalino (r)evolution Board Kit BLE/BT		20x20x10	300	4 entrada (10 bits) 2 entrada (6 bits) 1 salida (8 bits)	2 entrada 2 salida	Bluetooth	ATMEGA238P	1	20	228,7
Raspberry Pi	Raspberry Pi Model 4B		10x7x3	50	No tiene	26 entrada/salida	Wifi Bluetooth	CORTEX A72	4	2200	62,95
	Raspberry Pi Zero 2W		1x6.5x3	16	No tiene	26 entrada/salida	Wifi Bluetooth	CORTEX A53	4	400	18,9
Arduino	Arduino nano 33 IoT		1x4.8x1.8	5	8 entrada (10 bits)	14 entrada/salida	Wifi Bluetooth	CORTEX-M0	1	50	20,8
Espressif	ESP32		15.1x2.3	6.8	18 entrada (12 bits)	34 entrada/salida	Wifi Bluetooth	Xtensa LX6	2	240	18,8

4.2 Tecnologías

En este segundo apartado se resumirán los sensores y tecnologías que son comúnmente usados para la adquisición de las señales fisiológicas de *Pulso/SPO2*, *EDA* o *GSR* y por último el *ECG*.

4.2.1 Pulso y SPO2

La tecnología usada en los dispositivos portables para la detección del pulso y del nivel de oxigenación en sangre es mediante la fotopletismografía. Esta tecnología también está siendo usada para detectar la concentración de hemoglobina [27], así como para el seguimiento de personas con asma [28].

El funcionamiento de esta tecnología es bastante sencillo. Se hace que dos leds, uno rojo y otro infrarrojo iluminen generalmente la tercera falange del dedo de una mano. Dependiendo de la intensidad con la que estas luces se reflejen en el dedo se podrá detectar el valor de pulso y de nivel de oxigenación en sangre.

La situación de los leds es fundamental, ya que en esta última falange del dedo la cantidad de capilares es muy alta, por lo que existe una gran circulación de sangre por ella. La sangre que circula por el cuerpo humano, está compuesta entre otras sustancias, de hemoglobina. Que dependiendo de si el corazón está en *Diástole* o en *Sístole*, la cantidad de hemoglobina (*Hb*) y hemoglobina saturada

de oxígeno (HbO_2) es diferente como se puede observar en la Figura 4.6. Cada una de estas sustancias tiene diferentes índices de refracción frente a las longitudes de onda del color rojo e infrarrojo, por lo que matizando lo escrito anteriormente, la luz reflejada no por el propio dedo, si no por la hemoglobina que circula por él se detecta tanto el pulso como el nivel de oxigenación en sangre.

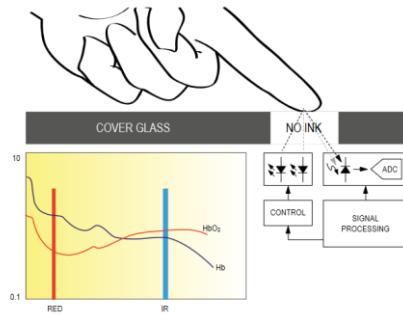


Figura 4.6.- Observación de la refracción de la luz en la Hb y la HbO_2

Fuente: <https://datasheets.maximintegrated.com/en/ds/MAX30100.pdf>

En ambos trabajos comentados anteriormente [27], [28], el sensor usado para la detección de estas dos señales es el *Max30100*. Este sensor incorpora la tecnología de encendido de los **leds**, recogida de la intensidad de luz que se refleja, así como el paso de esta unidad analógica a una digital (por medio de una *ADC* – *Conversor analógico digital*). La comunicación con la placa del microcontrolador es por medio de la tecnología *I2C*.

Tras estudiar en las posibilidades de compra de este componente para que fuese usado en el proyecto, se descubrió la existencia de una versión superior del sensor, el *Max30102*. Este tenía las mismas especificaciones que su versión anterior, pero el conversor analógico digital tenía una mayor resolución, lo que aportaba al dispositivo final una mayor fiabilidad en la recogida de las señales fisiológicas.

Tabla 4.2.- Comparativa entre Max30100 y Max30102

Sensor		Especificaciones				
		Colores LED	Sensor de Temperatura	Comunicación I2C	Resolución ADC	Precio (€)
Max30100		Rojo Infrarrojo	Si	Si	14 bits	4,85
Max30102		Rojo Infrarrojo	Si	Si	18 bits	4,99

Atendiendo al estudio comparativo de la En ambos trabajos comentados anteriormente [27], [28], el sensor usado para la detección de estas dos señales es el *Max30100*. Este sensor incorpora la tecnología de encendido de los **leds**, recogida de la intensidad de luz que se refleja, así como el paso de esta unidad analógica a una digital (por medio de una *ADC* – *Conversor analógico digital*). La comunicación con la placa del microcontrolador es por medio de la tecnología *I2C*.

Tras estudiar en las posibilidades de compra de este componente para que fuese usado en el proyecto, se descubrió la existencia de una versión superior del sensor, el *Max30102*. Este tenía las mismas especificaciones que su versión anterior, pero el conversor analógico digital tenía una mayor resolución, lo que aportaba al dispositivo final una mayor fiabilidad en la recogida de las señales fisiológicas.

Tabla 4.2, se decidió que el sensor que iba a ser usado para el desarrollo del proyecto fuese la versión *Max30102*, ya que ese incremento en la resolución repercutía notablemente en la calidad de la señal recogida, y el precio de este no incrementaba en exceso.

4.2.2 EDA/GSR

La segunda señal fisiológica que se quiere recoger es la Actividad Electrodérmica (EDA) o también llamada, Respuesta Galvánica de la Piel (GSR). Esta señal, mide la conductividad de la piel del ser humano. Esta conductividad varía constantemente debido a los cambios constantes en el estado de ánimo provocados por situaciones cortas o largas.

Dependiendo de la situación a la que se someta al sujeto, su sistema nervioso provocará una variación en la sudoración de la piel. Este sudor afectará a la conductividad de la piel en consecuencia. Por lo que gracias a esta señal se podrá detectar el cambio en el estado de ánimo observando solo las variaciones en la señal *EDA*.

En la investigación se observó que la recogida de la señal *EDA*, no usaba circuitos integrados de fabricantes en concreto [29], si no que al tratarse de una conductancia la medición de esta es muy sencilla. Solo sería necesario aplicar una tensión en un dedo, y medir la tensión en otro dedo. Para aplicar y medir esas tensiones, se usan los electrodos.

Existen dos tipos genéricos de electrodos, los de contacto húmedo, y los de contacto seco (Figura 4.7). Las diferencias entre estos dos tipos, es que uno hace uso de gel que favorece la conducción entre la parte metálica del electrodo y la piel, y el otro no.



Figura 4.7.- Tipos de electrodos según su contacto

Debido a que a lo largo de los trabajos hechos en la línea de investigación la recogida de señal *EDA* se hizo mediante electrodos de contacto seco [1], [4], se decidió que para este proyecto se iba a seguir con esta decisión.

4.2.3 ECG

La tecnología usada para la detección de la señal electrocardiográfica, como ya se ha comentado anteriormente, puede ser con tarjetas específicas que filtran y acondicionan la señal, dejando al usuario el mínimo margen para mejorar la señal recogida, o con circuitos propios diseñados específicamente para un proyecto en particular.

En la investigación llevada a cabo se observó que la mayoría de trabajos en los que se recogía esta señal, se hacía por medio de circuitos prediseñados, como puede ser el *AD8232* [30] (Figura 4.8) o el propio módulo de adquisición presente en la placa *BITalino* [22]. Uno de los aspectos positivos de este circuito prefabricado (El *AD8232*) es que en su envoltorio de fábrica viene incluido los cables de conexión y electrodos necesarios para la correcta detección de la señal electrocardiográfica.



Figura 4.8.- Circuito AD8232

Fuente: <https://eu.robotshop.com/es/products/ad8232-heart-rate-monitor>

Pero, en el intento de seguir desarrollando nuevos caminos en la línea de investigación, se decidió alejarse de este producto prefabricado, e intentar diseñar un circuito y una placa propia para así poder tener mayor margen de ajuste para la correcta obtención de la señal. Así mismo, aprovechando el material comprado por el grupo de investigación y con la intención de aprovechar al máximo los recursos del grupo, también se decidió que los electrodos que se usarían en el proyecto serían los de contacto húmedo.

CAPITULO 5

METODOLOGÍA

5 Metodología

En este quinto capítulo se expondrá y explicará el proceso seguido para llegar a la solución final. Este se dividirá en tres grandes sub-apartados. En primer lugar, el proceso seguido para la creación y configuración de una base de datos en una máquina virtual. El segundo y tercer sub-apartado se centrarán en el dispositivo de adquisición de datos y cómo este se comunica con la base de datos antes creada. Se trata de dos apartados diferentes, ya que como se ha comentado en el Capítulo 0, se decidió intentar desarrollar una solución en dos dispositivos diferentes, una *Raspberry Pi 4B* y una *ESP32*.

5.1 Creación de una Base de Datos

En este trabajo se ha trabajado con una máquina virtual de Linux, pero el proceso se ha probado en dos versiones diferentes del sistema operativo *Ubuntu*, la versión *22.04* y la *18.04 minimal version*. En el caso de que futuros trabajos se desarrollem mediante máquinas virtuales conectadas mediante *ethernet* a la intranet de la universidad, es necesario hacer uso de un adaptador USB para otorgar a los dispositivos de la universidad de conexión wifi.

Para este proyecto se ha hecho uso del dispositivo *TL-WN725N*. Para así poder estar conectado tanto a la intranet de la UPV/EHU, que al fin y al cabo es la que abastece al ordenador de internet, y a la wifi sin internet creada mediante el Router *TL-WR841ND*, a la cual se conectarán tanto la base de datos como el dispositivo de adquisición de datos diseñado en los siguientes apartados.



Figura 5.1.- Dispositivos TL-WN725N (USB) y TL-WR841ND (Router)

Fuente: <https://www.tp-link.com/es/home-networking/adapter/tl-wn725n/>
<https://www.tp-link.com/es/home-networking/wifi-router/tl-wr842nd/>

5.1.1 Configuración del adaptador USB

Para la configuración del *TL-WN725N*, solo sería necesario hacer uso del CD incorporado en el envase donde viene el dispositivo, en él se encuentra un ejecutable llamado *autorun.com* (**¡Error! No se encuentra el origen de la referencia.**) que instalará los drivers necesarios para abastecer al ordenador de la comunicación wifi. Además, dentro de la caja contenedora del dispositivo también se encuentra una guía rápida de instalación, en la cual se dan todas las instrucciones para llevar a cabo la instalación.

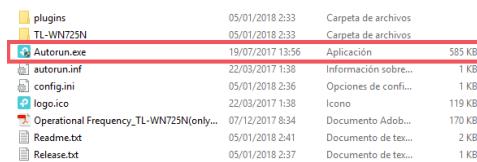


Figura 5.2.- Instalador de drivers del TL-WN725N

A partir de este momento, el ordenador ya dispondrá de comunicación wifi y la conexión a redes será de la forma habitual de Windows.

5.1.2 Configuración del Router

El primero de los pasos para configurar una red privada, en un instante inicial, sin internet, es conectar el dispositivo *TL-WR841ND* a la red eléctrica. Una vez se hayan encendido los dos primeros leds (Figura 5.3), el primero de ellos permanentemente y el segundo de forma parpadeante, se podrá acceder al panel de configuración del fabricante.



Figura 5.3.- Leds encendidos para la configuración del TL-WR841ND

Pero antes de entrar en la configuración será necesario hacer una conexión entre alguno de los cuatro puertos LAN (amarillos) (Figura 5.4) y un ordenador con al menos un explorador web.



Figura 5.4.- Puertos LAN

Mediante este explorador web se accede a la siguiente dirección *192.168.0.1*, se trata de la dirección IP que el fabricante ha dispuesto para la configuración del dispositivo. Eso sí, antes de entrar en el panel de configuración, se solicita una autenticación (Figura 5.5) que como no se ha hecho ningún cambio de credenciales, son el usuario y la contraseña por defecto (*admin/admin*).

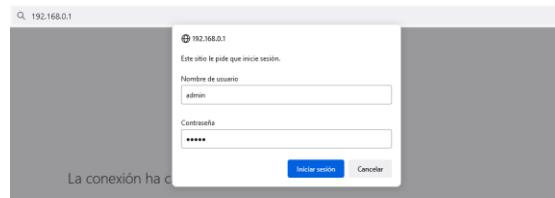


Figura 5.5.- Solicitud de usuario y contraseña configuración TL-WN725N

Dentro de todas las opciones que existen para la configuración, la que se va a utilizar para la creación de una red wifi privada, es la llamada *Wireless*, y dentro de ella, se configurarán tanto *Wireless Settings* como *Wireless Security*. Como se puede apreciar en la Figura 5.6, el campo que se va a modificar es tan solo el nombre de la red (*GICI*).

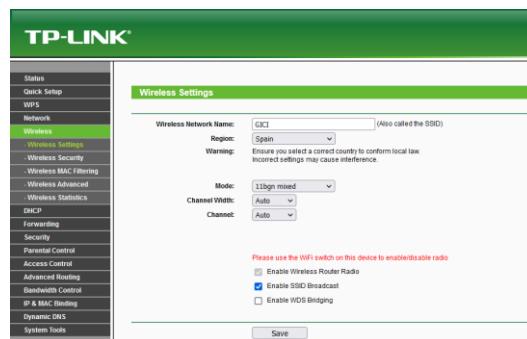


Figura 5.6.- Configuración de Wireless Settings

En cambio, en la pestaña de *Wireless Security*, se configurará como su propio nombre indica la seguridad que se le va a otorgar a la red. Para esta aplicación se ha seleccionado la recomendada por el propio fabricante *WPA2-PSK*, en la que tan solo se va a cambiar la contraseña (*delfingiciehu*). En la Figura 5.7 se muestra la totalidad de los campos para no dejar espacio a las dudas.

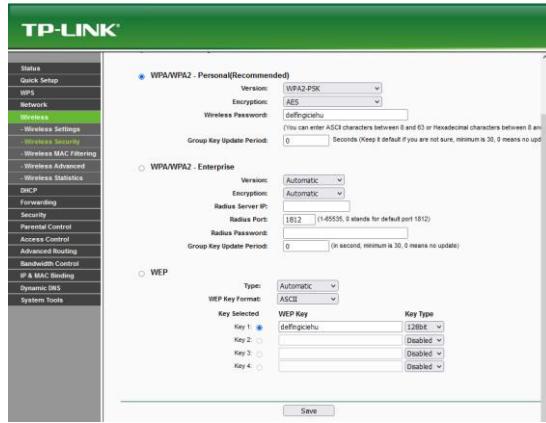


Figura 5.7.- Configuración de Wireless Security

5.1.3 Configuración de la Máquina Virtual

La aplicación usada para la utilización de la máquina virtual es *VMWare Workstation 16* (Descargable en <https://www.vmware.com/es/products/workstation-player.html>) y como se ha comentado anteriormente, para este desarrollo se ha utilizado la versión *Ubuntu 22.04*, que es la última versión estable lanzada públicamente por la propia organización.

El gran problema encontrado para el correcto funcionamiento de la máquina virtual ha sido que esta debería de tener una dirección IP propia y accesible para todos los dispositivos conectados a la red wifi antes creada. Por lo que una vez se haya inicializado la máquina virtual y se haya actualizado todo el sistema, se va a configurar correctamente la máquina virtual para que esta tenga una dirección *IP* diferente a la del ordenador donde se encuentra.

Dentro de los ajustes de la máquina virtual en cuestión, en este caso la denominada *Ubuntu22.04*, los cambios que se van a hacer son los concernientes a las conexiones de red, que como se puede observar en la Figura 5.8, la pestaña en cuestión, es *Network Adapter*.

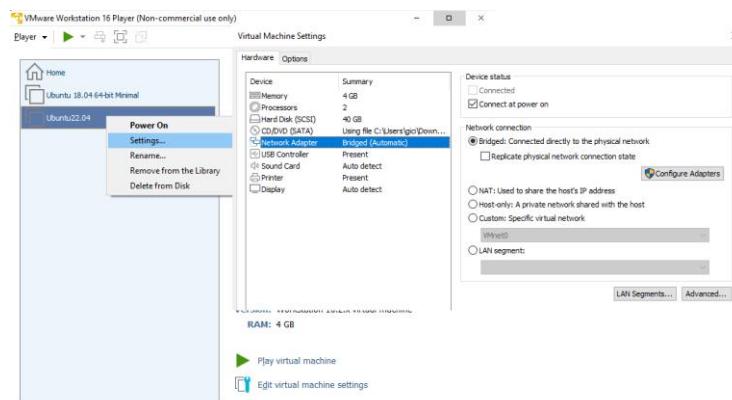


Figura 5.8.- Configuración de Network adapter (VMware)

Dentro de esta pestaña se encuentran varias opciones, la seleccionada por defecto, es la llamada *NAT*, en la que la máquina virtual tiene las mismas direcciones IP que el ordenador físico. El gran problema de esta configuración es, que a la hora de direccionar al dispositivo de adquisición de datos a la IP de la máquina virtual se estaría conectando realmente al ordenador que contiene la máquina virtual.

Por ello, se va a seleccionar la opción *Bridged*, la cual hace que la máquina se conecte directamente a la red física. Además, se deseleccionará la casilla de *replicate physical network connection state*. Pero queda configurar esta opción así que se hará clic en el botón de *Configure Adapters*. Al hacerlo, y conceder los permisos que pide, se abrirá la ventana mostrada en la Figura 5.9 y se elegirá solamente la casilla correspondiente a *TP-Link Wireless USB Adapter*.

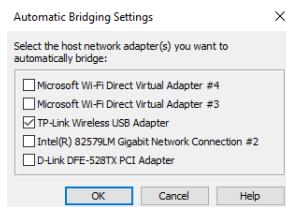


Figura 5.9.- Configure Adapters (VMWare)

Es importante comentar que el cambio entre la configuración *NAT* y *Bridges*, va a ser prácticamente constante, ya que la primera de ellas es necesaria para poder obtener una conexión a internet y así poder descargar paquetes en Linux. *Bridges*, en cambio, será la configuración usada para el levantamiento y gestión de la base de datos.

5.1.4 Instalación y configuración de PostgreSQL

Una vez se tiene configurada la red y la máquina virtual con la que se va a trabajar, solo queda instalar la aplicación que va a ser usada para la creación de la base de datos dentro de ella. En este caso, se usará *PostgreSQL*, que se trata de un sistema de código abierto y gratuito para la administración de bases de datos, cuyo desarrollo se lleva adelante por una gran comunidad de colaboradores.

Antes de instalar *PostgreSQL*, es necesario actualizar el repositorio de apt, mediante la ejecución de la siguiente instrucción en la línea de comando (en Ubuntu se puede abrir mediante Ctrl + Alt + T):

```
sudo apt update
```

A continuación, se puede llevar a cabo la instalación de *PostgreSQL*:

```
sudo apt install -y postgresql
```

Si ya existe una versión instalada se procede a la eliminación y desinstalación:

```
sudo apt remove --purge postgresql-*
```

Una vez se tiene instalada la última versión disponible para el dispositivo en cuestión de *PostgreSQL* se puede comenzar a configurar esta aplicación para que solo acepte conexiones con encriptación *SSL*. Este tipo de encriptación proporciona una conexión segura entre el cliente y el servidor. Además, se trata de un protocolo de internet que encrypta los datos que viajan por la conexión, y así se hacen imposibles de leer para un dispositivo externo.

Para la configuración de la comunicación por medio del protocolo *SSL*, es necesario seguir los pasos establecidos por *Oinatz Aspiazu* en su trabajo fin de máster [31]. El primero de los pasos es cambiar de usuario y establecerse como usuario *ROOT*, un usuario que tiene acceso a todas las carpetas y documentos que existen en el sistema. En segundo lugar, se sitúa en un directorio concreto para así poder crear en tercer lugar una carpeta y situarse de nuevo en ella.

```
sudo su
cd /var/lib/postgresql/
mkdir data
cd data
```

Una vez se tiene creada la carpeta, se ejecuta las instrucciones encargadas de la creación de los certificados y claves necesarias para la encriptación SSL.

```
openssl genrsa -des3 -out server.key 2048
# se solicita una contraseña
chmod 400 server.key
openssl req -new -key server.key -days 36500 -out server.crt -x509 -subj '/C=SP/ST=Bizkaia/L=Bilbao/O=UPV_EHU/CN=rpsensors.com/emailAddress=name@mail.com'
# subj se rellena con datos personales
openssl rsa -in server.key -out server.key
# se elimina la contraseña introducida para proteger el archivo server.key (la pide por última vez)
chmod 600 server.key server.crt
# se establece los permisos necesarios de lectura/modificación/escritura a los certificados
```

Como se han creado tanto la carpeta *data* como los archivos *server.key* y *server.crt* bajo el usuario *root* es necesario cambiar el *Owner* y el *Group* de todos ellos, para que estos puedan ser leídos y utilizados por *PostgreSQL*, que tiene su propio usuario (*postgres*).

Para visualizar los datos de autoridad de archivos y carpetas, se usa la función (*ls -l*). En la Figura 5.10 se puede observar como la carpeta y los archivos recién creados pertenecen al usuario *root*.

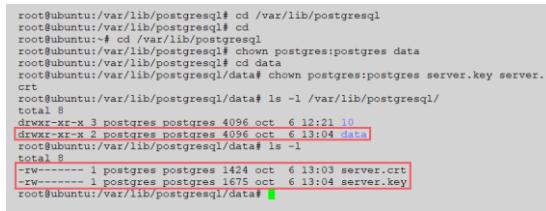
```
root@ubuntu:/var/lib/postgresql# ls -l
total 8
drwxr-xr-x 3 postgres postgres 4096 oct  6 12:21 10
drwxr-xr-x 2 root      root    4096 oct  6 13:04 data
root@ubuntu:/var/lib/postgresql# ls -l data
total 8
-rw----- 1 root      root    1424 oct  6 13:03 server.crt
-rw----- 1 root      root    1675 oct  6 13:04 server.key
root@ubuntu:/var/lib/postgresql#
```

Figura 5.10.- Autoridad y permisos de Data, server.key y server.crt (root)

El comando usado para la modificación tanto de propietario como de grupo es la llamada *chown*:

```
cd /var/lib/postgresql
# se coloca en este directorio para cambiar las propiedades de data
chown postgres:postgres data
# el primero de los términos hace referencia al usuario y el segundo al grupo
cd data
chown postgres:postgres server.key server.crt
```

En la Figura 5.11 se puede observar el resultado de volver a ejecutar el comando (*ls -l*) para comprobar que se ha hecho el cambio de propietario y grupo.



```

root@ubuntu:/var/lib/postgresql# cd /var/lib/postgresql
root@ubuntu:/var/lib/postgresql# cd data
root@ubuntu:# cd /var/lib/postgresql
root@ubuntu:/var/lib/postgresql# chown postgres:postgres data
root@ubuntu:/var/lib/postgresql# chown postgres:postgres server.key server.crt
root@ubuntu:/var/lib/postgresql# ls -l /var/lib/postgresql/
total 8
drwxr-xr-x 3 postgres postgres 4096 oct  6 12:21 10
drwxr-xr-x 2 postgres postgres 4096 oct  6 13:04 data
root@ubuntu:/var/lib/postgresql# ls -l
total 8
-rw----- 1 postgres postgres 1424 oct  6 13:03 server.crt
-rw----- 1 postgres postgres 1675 oct  6 13:04 server.key
root@ubuntu:/var/lib/postgresql# ls -l

```

Figura 5.11.- Autoridad y permisos de Data, server.key y server.crt (postgres)

En este punto, se puede continuar con la configuración de *PostgreSQL*, en primer lugar, se modificará el archivo *postgresql.conf* (para habilitar la comunicación con cifrado SSL) y el archivo *pg_hba.conf* (para determinar las direcciones IP que podrán conectarse a la base de datos), ambos archivos están alojados en el directorio siguiente:

/etc/postgresql/14(version_de_postgresql)/main

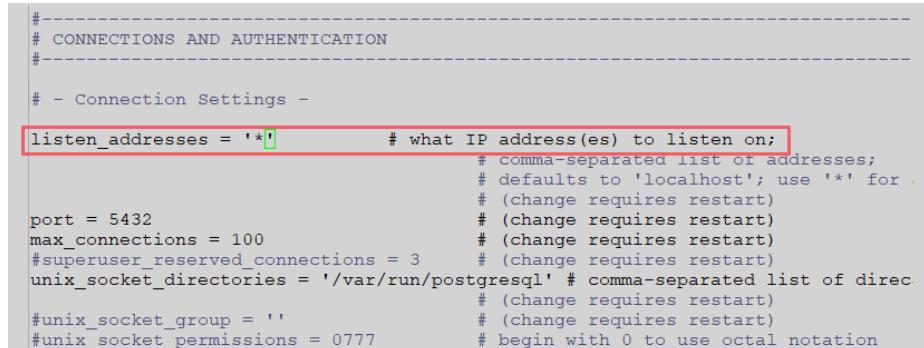
Y para ejecutar las modificaciones se hará uso del comando *nano*, que ejecuta un editor de texto llamado *nano* en el archivo que le sea indicado.

```

nano /etc/postgresql/14/main/postgresql.conf # para el caso del primer archivo
nano /etc/postgresql/14/main/pg_hba.conf    # para el caso del segundo archivo

```

Las modificaciones en el archivo *postgresql.conf*, son las siguientes (Figura 5.12 y Figura 5.13):



```

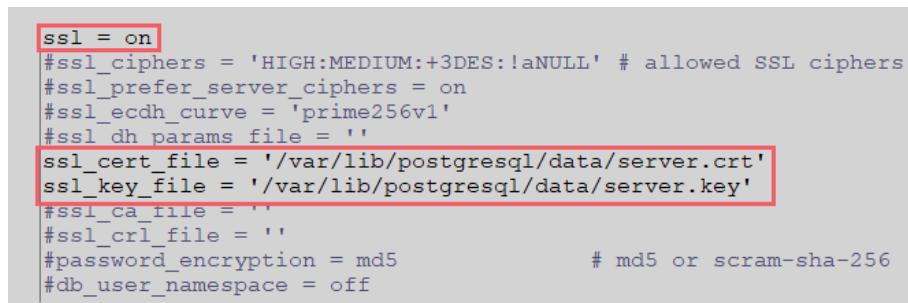
#-----
# CONNECTIONS AND AUTHENTICATION
#-----

# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for .
                                # (change requires restart)
port = 5432                      # (change requires restart)
max_connections = 100            # (change requires restart)
superuser_reserved_connections = 3  # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of direc-
                                                # (change requires restart)
unix_socket_group = ''           # (change requires restart)
unix_socket_permissions = 0777   # begin with 0 to use octal notation

```

Figura 5.12.- Cambio escucha todas las IP (postgresql.conf)



```

ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL' # allowed SSL ciphers
ssl_prefer_server_ciphers = on
ssl_ecdh_curve = 'prime256v1'
ssl_dh_params_file =
ssl_cert_file = '/var/lib/postgresql/data/server.crt'
ssl_key_file = '/var/lib/postgresql/data/server.key'
ssl_ca_file =
ssl_crl_file =
password_encryption = md5           # md5 or scram-sha-256
db_user_namespace = off

```

Figura 5.13.- Activación y actualización de localización certificados (postgresql.conf)

En este punto, la base de datos podría comunicarse con cualquier dispositivo que esté conectado a la misma señal wifi que ella por medio de cifrado SSL. Debido a que el objetivo es lograr que solo los dispositivos que el diseñador quiera puedan conectarse con la base de datos, es necesario introducir las direcciones IP de estos, en una *whitelist*, una lista de direcciones IP con las que se puede conectar.

Esta *whitelist*, se introduce en el otro archivo antes mencionado, *pg_hba.conf*, y los cambios a hacer son los mostrados en la Figura 5.14:

#	TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only					
local	all	all			peer
# IPv4 local connections:					
host	all	all		127.0.0.1/32	md5
 host all all 192.168.0.100/32 md5 host all all 192.168.0.104/32 md5					
# IPv6 local connections:					
host	all	all		::1/128	md5
# Allow replication connections from localhost, by a user with the # replication privilege.					
local	replication	all			peer
host	replication	all		127.0.0.1/32	md5
host	replication	all		::1/128	md5
 # SSL					
hostssl	admin	all		127.0.0.1/32	md5 clientcert=1
hostssl	admin	all		192.168.0.100/32	md5 clientcert=1
hostssl	admin	all		192.168.0.104/32	md5 clientcert=1

Figura 5.14.- Whitelist de direcciones IP (pg_hba.conf)

Donde, la dirección IP referente a la base de datos es, *192.168.0.100*, y la referente al dispositivo al que se le quiere permitir la conexión es, *192.168.0.104*. Además, se ha introducido en una de las columnas la palabra *admin*, que se trata de un usuario creado específicamente para conectarse con la base de datos en *postgreSQL* (La creación de este usuario se mostrará a continuación).

5.1.5 Creación de la base de datos

Una vez ya está configurado todo lo concerniente a la seguridad y conexión con el dispositivo donde se encontrará la base de datos, es el momento de crearla. Para ello tendremos que hacer primeramente los siguientes comandos:

```
sudo su postgres # solicitará contraseña, es la que se usa para iniciar sesión en
psql
```

De aquí en adelante, todas las instrucciones que se escriban serán en lenguaje *SQL*, que se trata de un lenguaje específico de computación para el manejo de grandes conjuntos de datos y las relaciones que existen entre ellos (todas las instrucciones deberán finalizar con “;”).

Lo primero que se va a crear siguiendo las siguientes líneas de comando, es el usuario antes mencionado, *admin*:

```
CREATE USER admin;
ALTER ROLE admin WITH PASSWORD '*1kRzhuYj!$'; # se introduce una contraseña
ALTER ROLE admin WITH SUPERUSER;
\du # (SIN ";") visualización de los usuarios existentes
```

La contraseña elegida para el usuario es muy importante, ya que esta será necesaria para que la conexión entre el dispositivo de adquisición de datos y la base de datos sea correcta. Además, en la última de las instrucciones, se pide al *postgreSQL* que liste todos los usuarios que existen. En este caso, la respuesta del sistema es la siguiente (Figura 5.15):

List of roles			
Role name	Attributes	Member of	
admin	Superuser		{}
postgres	Superuser, Create role, Create DB, Replication, Bypass RLS		{}

Figura 5.15.- Listado de usuarios

El segundo paso es crear una base de datos, ya que en un mismo dispositivo se pueden crear múltiples bases de datos para diferentes quehaceres. Para este proyecto solo se usará una base de datos, y dentro de ella, se introducirán diferentes tablas, una por cada sensor implementado en el dispositivo adquisidor de datos. Las instrucciones a seguir son las siguientes:

```
CREATE DATABASE databio;
\c databio # a partir de esta instrucción se trabaja dentro de la BBDD “databio”
CREATE TABLE pox (acq_time timestamp, red_val int, ir_val int);
```

Donde *pox*, se trata del nombre de la tabla, y dentro del paréntesis se introduce el nombre y el tipo de valor a introducir en cada una de las columnas. Para poder observar, si se ha creado correctamente se sugieren dos instrucciones diferentes, la primera de ellas, “*\dt*”, que listará las tablas dentro de la base de datos y la segunda “*SELECT*FROM pox*”, que mostrará el contenido de esta tabla.

Las respuestas del sistema a estas dos instrucciones serían las mostradas en la Figura 5.16:

List of relations			
Schema	Name	Type	Owner
public	pox	table	postgres

databio=# SELECT * FROM pox;
acq_time red_val ir_val
(0 rows)

Figura 5.16.- Listado y visualización de tabla

Finalmente se otorga todos los privilegios posibles (modificación, escritura, lectura...) para esta base de datos mediante la instrucción *GRANT ALL PRIVILEGES*.

```
GRANT ALL PRIVILEGES ON DATABASE databio TO admin;
```

5.2 Raspberry Pi

Como se ha comentado al inicio de este capítulo, en este segundo sub-apartado referente a la metodología se va a centrar en comentar y explicar el proceso seguido para lograr una primera aproximación al dispositivo objetivo. En el capítulo 0 se comentó que el mini ordenador a usar para implementar una solución era la *Raspberry Pi 4B*. Para que después la solución sea migrada a una placa de menor tamaño y similar potencia (*Raspberry Pi Zero 2W*).

5.2.1 Instalación de un OS

El primero de los pasos a dar en cualquier tipo de dispositivo similar a un ordenador es instalar un sistema operativo (*OS*) para que el mini-ordenador pueda funcionar. Sería como instalar *Windows 11* o *Mac OS*. En este caso, para los dispositivos del fabricante *Raspberry* el sistema operativo mayormente usado es *Raspbian*.

Para la instalación se recomienda el uso de la herramienta *Raspberry PI Imager*. En ella seleccionar el sistema operativo recomendado, ya que para este proyecto está siendo usado un ordenador con potencia suficiente para ejecutar la última de las versiones de *Raspbian OS*.

A la hora de instalar el OS, en la herramienta antes comentada, también se hizo una configuración avanzada de la instalación como se muestra en la Figura 5.17. En ella, se introdujo un nombre personal para este dispositivo, se habilitó la comunicación vía SSH, así como la conexión WIFI.

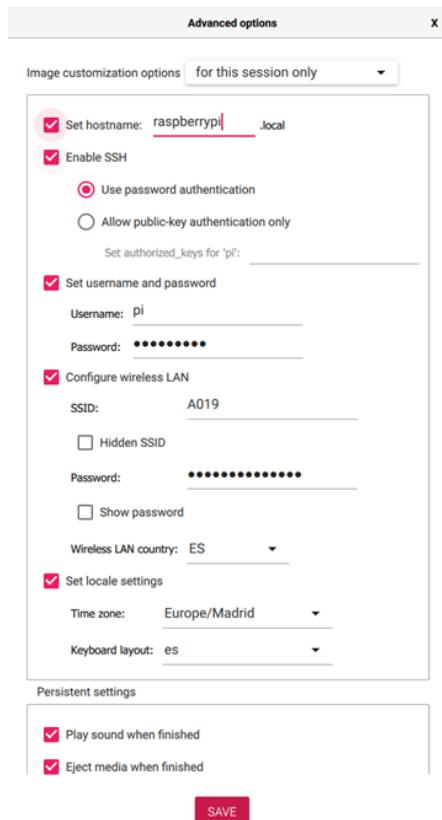


Figura 5.17.- Ventana Advanced Options

5.2.2 Configuración Raspberry

Una vez se tiene instalado un *OS*, solo queda configurar correctamente la *Raspberry*, para que esta pueda funcionar correctamente como un dispositivo de recogida de datos. La configuración para ello será necesario habilitar dos *interfaces*, que este sistema operativo inhabilita por defecto, *I2C* y *VNC*.

- **I2C**

Este tipo de comunicación se trata de un *BUS* de 4 cables (*Vcc*, *Sda*, *Scl*, *Gnd*), que conecta un *Maestro* y un *Esclavo* (Figura 5.18). La comunicación solo podrá ser iniciada por el *Maestro*, por lo que el *Esclavo* solo podrá reaccionar a estas llamadas.

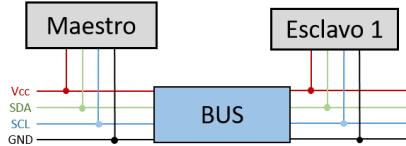


Figura 5.18.- Esquema de I2C

El I2C será usado para comunicar el dispositivo de adquisición de datos con el sensor pulsioxímetro seleccionado en el Capítulo 0, el *MAX30102*.

- **VNC**

Esta comunicación puede ser usada para conectarse al dispositivo remotamente, pero con la *interface* completa, es decir, observando una pantalla y usando un ratón y un teclado. Esta tecnología ha sido usada para los momentos en los que no se disponía de cables HDMI, para usar pantallas, y así poder seguir con el proyecto.

Para la correcta habilitación de estas dos *interfaces*, es necesario abrir la ventana de comandos de *Raspbian* (mediante el atajo *Ctrl + Alt + T*), y en ella escribir la instrucción *sudo raspi-config*. La ventana emergente que aparece es la mostrada en la Figura 5.19 (Se podrá navegar por las opciones con las flechas del teclado).

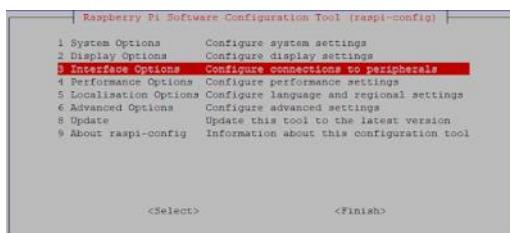


Figura 5.19.- Ventana inicial Raspi-config

Tras seleccionar la tercera de las opciones llamada *Interface Options*, se desplegará una nueva ventana (Figura 5.20), en la que se podrán habilitar las *Interface I2C* y *VNC*.

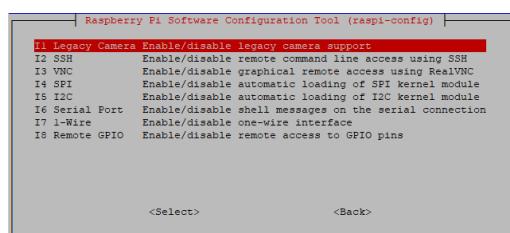


Figura 5.20.- Ventana habilitación *interfaces*

5.2.3 Programación Python

En este punto, la *Raspberry Pi 4B* ya estaría lista para comenzar a programar el funcionamiento para la recogida y subida de datos. Debido a que la *Raspberry* que va a ser usada se trata de un mini-ordenador existen gran cantidad de lenguajes de programación con los que crear la función final del dispositivo.

El lenguaje de programación elegido para el desarrollo de este dispositivo es *Python* (descargable en la Figura 5.21). Este fue seleccionado debido a que se trata de un lenguaje de programación de propósito general, es decir, puede ser usado casi para cualquier cosa. También se trata de uno de los lenguajes más sencillos de programar y aprender, y existen una gran cantidad de bibliotecas y funciones generadas por otros usuarios que facilitan la implementación de sensores, y comunicaciones con otros dispositivos.



Figura 5.21.- Logo de Python

Fuente: <https://www.python.org/downloads/>

5.2.3.1 Soluciones a Tiempo Real en Python

El gran objetivo de este proyecto es que el sistema se comporte como un sistema de tiempo real. Esto implica, entre una gran variedad de restricciones que el sistema debe cumplir plazos. Esto significa que, si se impone una frecuencia de muestreo de diez muestras por segundo, en cualquier situación, cada segundo debe de enviar diez muestras sin perder ninguna. En todos los lenguajes de programación dedicados a ejecución de código en tiempo real se busca que el código ejecute diferentes tareas (funciones generalmente) al mismo tiempo.

El lenguaje Python aporta dos soluciones diferentes en forma de biblioteca para que los programas desarrollados en este lenguaje puedan ejecutar diferentes tareas al mismo tiempo, *Multithreading* y *Multiprocessing*.

- **Multithreading:** se refiere a la capacidad de que un sistema sea capaz de ejecutar diferentes hilos al mismo tiempo, pero ejecutando cada uno de ellos en diferentes núcleos del procesador. En la Figura 5.22 se puede observar un diagrama de cómo se comportaría un procesador de cuatro núcleos como el que va montado en la *Raspberry Pi 4B*.

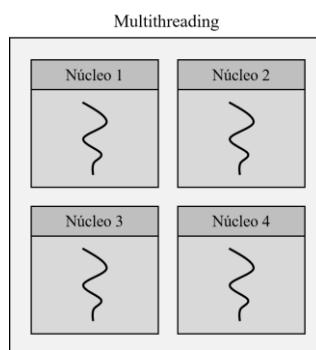


Figura 5.22.- Diagrama multithreading

- **Multiprocessing:** se refiere a la capacidad de que un sistema pueda ejecutar diferentes hilos al mismo tiempo, pero, a diferencia del *multithreading*, en el *Multiprocessing* los hilos se separan por todos los núcleos del procesador, lo que hace que se puedan gestionar mayor cantidad de hilos (siempre y cuando la capacidad del procesador pueda). En la Figura 5.23 se puede observar de nuevo, un diagrama de *Multiprocessing* en un procesador de cuatro núcleos como el que usa la *Raspberry Pi 4B*.

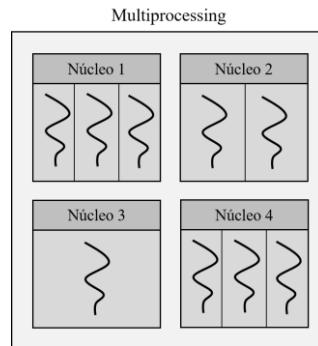


Figura 5.23.- Diagrama Multiprocessing

Observando las dos definiciones se decidió hacer uso de la biblioteca *Multiprocessing*, ya que, al tratarse de un mini-ordenador, la *Raspberry* tendrá diferentes procesos o hilos activos por el mero hecho de estar encendido, y de esta manera, se podría mandar los hilos o tareas generados en el programa a los núcleos que estén menos saturados en cada instante.

5.2.3.2 Componentes de un sistema en tiempo-real

Una vez conocida la biblioteca que iba a ser usada para la generación de un sistema multitarea, se puede comenzar a pensar la estructura que se va a seguir en el programa a diseñar. El objetivo inicial fue que el programa recogiera dos señales diferentes, la señal del Pulsioxímetro y la señal del GSR. Se quiso crear un programa que no distara demasiado de los programas escritos en lenguajes centrados en el tiempo real.

Los tipos de elementos con los que se trabajó en este primer acercamiento a un programa funcional son, las *Tareas Periódicas*, *Registros Protegidos* y *Recursos Pasivos*. Pero es necesario saber cómo funcionan y qué hacen cada uno de estos elementos.

- **¿Qué es una Tarea Periódica?**

Es decir, una sección de código que se repetiría cada cierto periodo de tiempo. Este tipo de tarea está definida por dos parámetros temporales, el *Periodo (P)* y el *Plazo (D)*. La tarea periódica se activará cada *P* tiempo, y su ejecución no podrá retrasarse más de *D* instantes. En el caso de que se supere el instante marcado por el plazo en cualquiera de las activaciones de esta tarea, la tarea no será garantista de plazos.

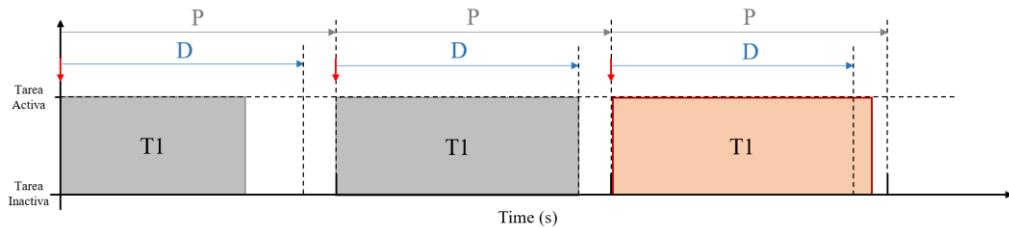


Figura 5.24.- Diagrama de activación de una Tarea Periódica

En la Figura 5.24 se puede observar como una tarea llamada *T1* se activa en tres instantes diferentes, todos ellos separados por el *Periodo(P)*. En la primera y la segunda activación

el tiempo de finalización de la tarea es menor o igual que el *Plazo(D)*, por lo que la tarea funciona correctamente. En cambio, la tercera de las activaciones tiene un tiempo de finalización mayor que el *Plazo (D)*, por lo que como se ha comentado anteriormente, la *T1* tendría un fallo de plazos.

- **¿Qué es un registro protegido?**

Estos recursos son aquellos que necesitan protección debido a que son usados o accedidos por dos o más tareas. La protección trata de que dos tareas no puedan acceder en el mismo instante al recurso, ya que esto podría suponer un mal funcionamiento del recurso o un fallo a la hora de recoger los datos que hay en él. En concreto, un registro protegido es un espacio de memoria donde se guarda más de una variable, y que tiene la protección frente a lecturas simultáneas dando prioridad a la que antes acceda al registro (Figura 5.25).

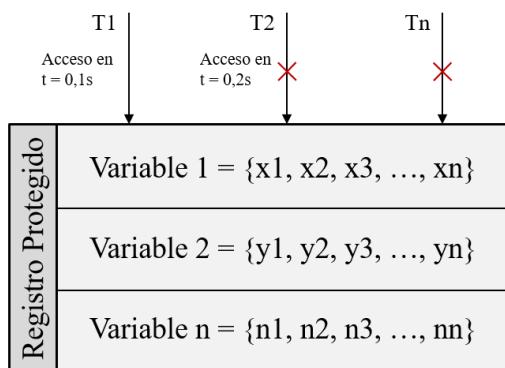


Figura 5.25.- Registro Protegido

- **¿Qué es un recurso pasivo?**

Estos recursos son aquellos en los que el usuario no puede programar en el programa principal. Existe gran variedad de recursos pasivos, como por ejemplo los sensores, de los cuales no se programa como recogen las señales, pero sí la recogida de estas. Al igual que ocurre con las bases de datos, de las cuales se programan como van a estar ordenadas, así como la seguridad con la que van a trabajar, pero no en el programa del dispositivo que se quiere llevar a cabo.

5.2.3.3 Biblioteca Tareas Periódicas

El primer impedimento que se encontró en el desarrollo del programa una vez se tenía clara la estructura que se quería seguir, fue que a pesar de tener una biblioteca que facilitaba que el dispositivo desarrollado fuera multi-tarea, las tareas periódicas no estaban definidas. Por lo que el paso inicial era crear una biblioteca que introdujera las tareas de tipo periódica.

- **Biblioteca Tareas periódicas V1:**

La biblioteca desarrollada para la introducción de las tareas periódicas de forma fácil se le puso el nombre de *BiblioTareasPeriodicas.py*. La creación de estas tareas se basó en las tareas por defecto que se tienen en lenguajes de programación dedicados al tiempo-real (*ADA*). En la Figura 5.26 se muestra el flujograma seguido para la creación de esta función.

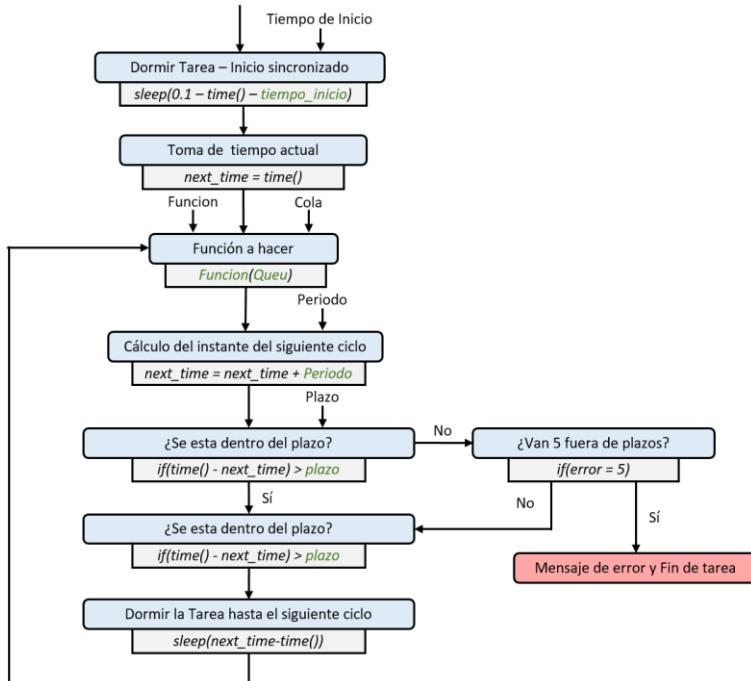


Figura 5.26.- Tarea periódica con plazo

En el flujograma se puede observar como hay ciertas palabras coloreadas en verde, estas palabras realmente son variables que tiene que dar el usuario que haga uso de la biblioteca para que la tarea periódica funcione correctamente.

- Tiempo de inicio:** se trata del instante en el que se llama a las tareas a iniciarse. Este tiempo será usado para llevar a cabo un inicio sincronizado de todas las tareas que se programen. Es decir, que todas las tareas comiencen en el mismo instante de tiempo.
- Función:** este parámetro contendrá el nombre de la función que se ha creado para que se repita cíclicamente. Por lo que lo que esté dentro de esta función será lo que periódicamente se haga.
- Queue:** se trata de la cola, o los parámetros necesarios para que la función antes comentada funcione correctamente.
- Periodo:** es la cantidad de tiempo que pasa entre activación y activación de la tarea periódica en segundos.
- Plazo:** tiempo en segundos a partir del cual se da por perdida la muestra obtenida en ese instante.

Como se aprecia en el flujograma de la Figura 5.26, nada más activar la tarea, se duerme, para poder hacer un inicio sincronizado, una vez se termina ese periodo de tiempo se toma el tiempo en el que este ha acabado,

A partir de este instante la tarea periódica se comporta ya cíclicamente. En primer lugar, lleva a cabo la función para la que está programada. Y se calcula el tiempo en el que se daría la siguiente activación. Inmediatamente después se comprueba si el tiempo tardado en ejecutar estas instrucciones, es menor que el plazo. Si esta comprobación resulta positiva, la ejecución cíclica seguirá. En cambio, si esta comprobación es negativa, y la situación se da más de cinco veces, la ejecución de esa tarea finalizará. Al introducir esta posibilidad de que la tarea se pase del *Plazo* un número mayor que uno, se hace que el sistema deje de ser de tiempo real **estricto**, y pase a ser uno **blando**.

- **Biblioteca Tareas periódicas V2:**

Tras ejecutar gran cantidad de tareas periódicas mediante esta biblioteca, se observó, que el código desarrollado podía ser depurado para que las tareas tuvieran un tiempo de ejecución menor.

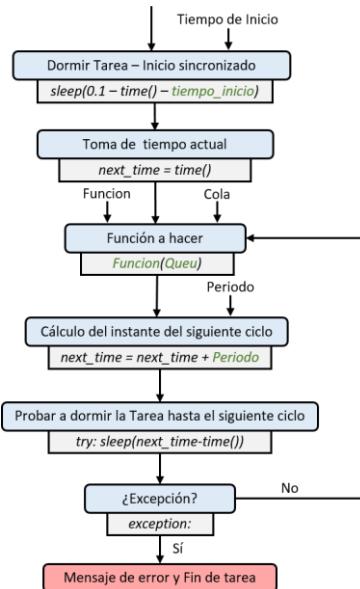


Figura 5.27.- Tarea periódica simplificada

La segunda versión de la biblioteca que aportaba las tareas periódicas al lenguaje de programación *Python*, se basó en el flujograma mostrado en la Figura 5.27. A diferencia del de la primera versión (Figura 5.26), en esta el plazo no es solicitado como parámetro, sino que se trabaja con un *Plazo* igual al *Periodo*. De esta manera, cuando se envía a la tarea a dormir el tiempo restante hasta la siguiente activación, si se da una excepción (que la resta de los tiempos sea negativa) se estaría superando el *Periodo* y el *Plazo*, por lo que finaliza la ejecución de la tarea.

5.2.3.4 Creación del primer programa

Con la biblioteca ***BiblioTareasPeriodicas*** recién generada ya se tendrían disponibles todos los elementos para programar en la *Raspberry Pi* las tres tareas periódicas, el registro protegido y los recursos protegidos. Como este primer acercamiento a una solución mediante Python para la comprobación de la viabilidad de ella, la subida de datos a la base de datos se llevó a cabo de forma ficticia, es decir, los datos se guardarían en la memoria del propio ordenador. Si este programa funcionara correctamente, el siguiente paso sería la implementación de esta subida de datos por *WIFI*.

Con todo esto, se decidió que el programa se dividiese en tres tareas periódicas diferentes, un registro protegido, donde se guardarían los datos obtenidos de los sensores, y tres recursos pasivos. Dando como resultado la Figura 5.28 donde se observa cómo se comunican cada uno de estos componentes entre sí, para lograr el objetivo de funcionamiento del dispositivo.

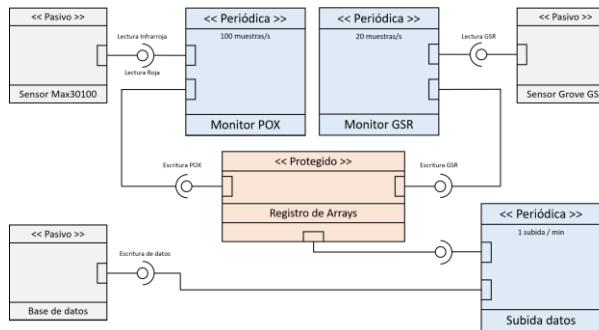


Figura 5.28.- Diagrama de comunicación – Primer programa

Como se ha comentado anteriormente, el programa estará compuesto de tres tareas periódicas, dos de adquisición de datos, y una de comunicación. La primera parte del programa se compondrá de las funciones que regirán el comportamiento de estas tareas. Es decir, el código necesario para que se recojan los datos de los sensores y las instrucciones necesarias para que los datos guardados en el *registro protegido* se conviertan en un archivo *XML* y se introduzca en una carpeta seleccionada del sistema.

- **Función: CogerGuardarDatosPOX**

Como el propio nombre indica, esta función se encargará de recoger y guardar los datos recibidos desde el sensor *MAX30102* en el recurso protegido. El propio fabricante ofrece una biblioteca para recoger estos datos para el lenguaje Python, por lo que para esta función la biblioteca *max30102* es usada.

Entre todas las funciones dentro de la biblioteca, la función *read_fifo()* es la seleccionada, ya que recoge los datos del sensor (el valor del reflejo de la luz roja y la luz infrarroja en el dedo) de forma rápida.

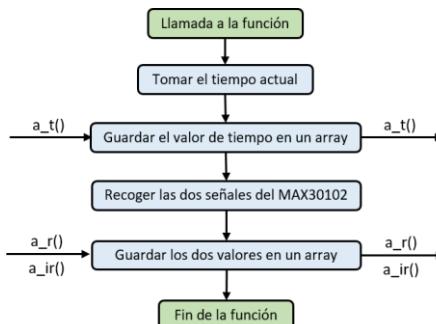


Figura 5.29.- Función CogerGuardarDatosPOX

La estructura de esta función es muy sencilla y se muestra en la Figura 5.29. Como se puede apreciar en ella, en primer lugar, se toma la hora actual para conocer en qué instante se tomó la muestra, e inmediatamente después se guarda en un array, que a la hora de llamar a la función será un *recurso protegido*. A continuación, se recogerán los dos valores del sensor y se guardarán también en dos arrays diferentes que al igual que con el de tiempo serán *recursos protegidos*.

- **Función: CogerGuardarDatosGSR**

La segunda de las funciones creadas, es la encargada de recoger los datos de los electrodos para la obtención del valor *GSR*. En este caso no se necesita la ayuda de ninguna biblioteca, ya que el valor recogido es una tensión. Pero sí que se necesita hacer una serie de cálculos para conocer el valor *GSR* correspondiente a esa tensión.

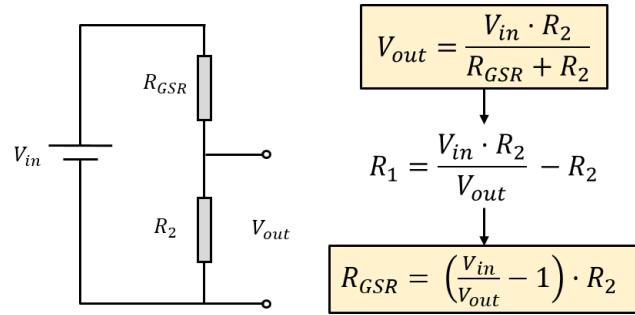


Figura 5.30.- Cálculo del valor GSR

En la Figura 5.30 se puede observar las fórmulas necesarias para calcular el valor de la resistencia. Se trata de un divisor de tensión, donde la resistencia llamada R_{GSR} es la del dedo del sujeto en estudio. Por lo que conociendo la tensión de entrada y el valor de la R_2 , que están ambas a la selección del diseñador, se puede calcular fácilmente el valor de esa resistencia.

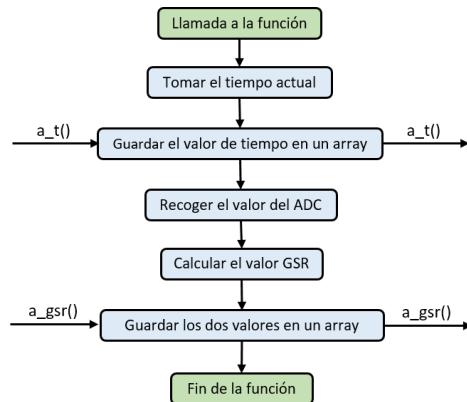


Figura 5.31.- Función CogerGuardarDatosGSR

Una vez se conocen los cálculos necesarios para obtener el valor *GSR*, la estructura de la función es prácticamente igual que la creada para la función de la señal del Pulsioxímetro (Figura 5.31). En primer lugar, se toma la hora actual y se guarda en el array que después se tratará de un recurso protegido. A continuación, se toma la tensión mediante un ADC (Conversor Analógico Digital) y se calcula el valor *GSR*, y este se introduce en otro array protegido.

- **Función:** Coms

La última de las funciones a programar es la concerniente a las comunicaciones. Esta función se encargará de guardar de una forma rápida los datos de los arrays protegidos en archivos CSV en la memoria interna, y después subir estos datos a una *bbdd* local (Base de datos).

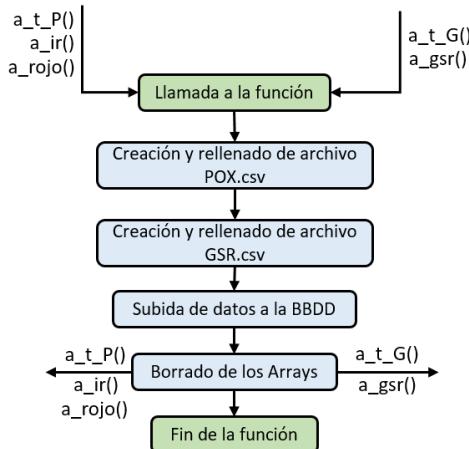


Figura 5.32.- Función COMS

En la Figura 5.32 se puede observar cómo sería la estructura de esta función de forma genérica. Al tratarse de una tarea de comunicaciones, será necesario darle acceso a los arrays generados en las otras funciones, para que esta pueda tomar los datos al inicio de la función y borrarlos a su final.

Para poder llevar a cabo todas las funcionalidades que se piden a esta función, creación de un archivo CSV y subida de datos a una bbdd, es necesario apoyarse en dos nuevas bibliotecas llamadas csv y psycopg2 que permiten al usuario tener instrucciones para crear esas funcionalidades.

■ Programa Principal:

Una vez creadas las tres funciones que determinarán el comportamiento de las tareas periódicas, se creó el programa principal donde se crearían todos los recursos protegidos de cada una de las tareas periódicas.

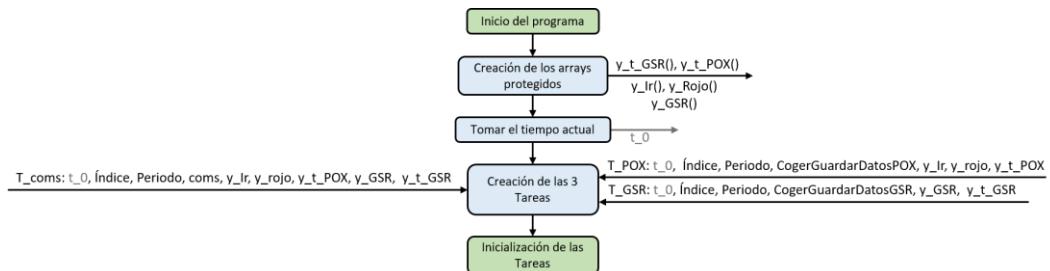


Figura 5.33.- Estructura Programa principal (V1 Python)

En la Figura 5.33 se puede observar la estructura que sigue el programa principal donde como se acaba de comentar se crean los arrays protegidos y las tareas periódicas, así como estas últimas se inicializan con los parámetros seleccionados por el diseñador.

5.2.3.5 Creación del segundo programa

Tras la elaboración de este primer programa se siguió probando cómo hacer que este funcionara de una forma mucho más rápida. La solución se encontró en el trabajo de fin de máster de un compañero del Grupo de Investigación que se centraba en las comunicaciones de este dispositivo [31]. Esta mejora en la velocidad de la toma y subida de los datos se debe a la implantación de *Queue*, un recurso protegido en el que se almacenan datos, donde el primer dato que entra es el primer dato que sale (*FIFO*). Además de la implementación de la función executemany de la biblioteca psycopg2.

En este nuevo programa el diagrama de tiempo real con el que se comenzó a diseñar el programa era algo diferente ya que no se usaba una tarea periódica para llevar a cabo la subida de datos a la

base de datos, si no que se usaría un Proceso de la biblioteca *Multiprocess* que se activaría cada vez que la *Queue* correspondiente llegase a los cien datos. Un comportamiento parecido a lo que serían las *Tareas Esporádicas*.

La estructura en tiempo real básica que fue usada es la mostrada en la Figura 5.34.

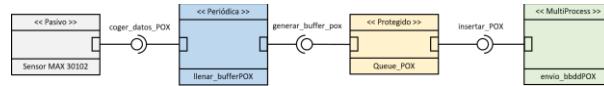


Figura 5.34.- Estructura tipo usada en el segundo programa

Si esta estructura básica se repite tres veces una para cada sensor o señal que se quiere recoger, se tendría el diagrama de comunicación entre las tareas periódicas, objetos protegidos, objetos pasivos y los nuevos procesos llamados *multiprocess*.

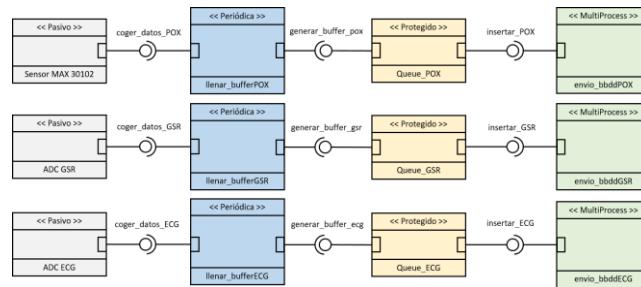


Figura 5.35.- Esquema estructura programa total

En la Figura 5.35 se puede observar el diagrama de comunicación ya finalizado donde cada una de las ramas como se acaba de comentar, se centra en una señal biológica. La primera de ellas, la señal del *Pulsioxímetro*, a continuación, la señal *GSR* y finalmente el *ECG* que no ha sido implementado.

Al igual que se ha hecho en el programa anterior, se comentará la estructura de las funciones, programas y subprogramas usados para el desarrollo de esta segunda versión e Python. De nuevo, el programa principal comienza con la creación de las funciones donde se crean los códigos encargados de tomar los datos desde los sensores y guardarlos en los archivos protegidos.

- **Programa:** main

Se trata del programa principal desde el que se va a llamar a otras funciones situadas en subprogramas. En este programa se encuentran tres funciones, y una parte principal.

- **Función:** generar_buffer_pox

La primera función que se introduce en el programa principal, se encarga de gestionar la obtención y de guardar los datos recibidos de los sensores, en una estructura que finalmente es guardada en la variable *queue* correspondiente a esa señal.

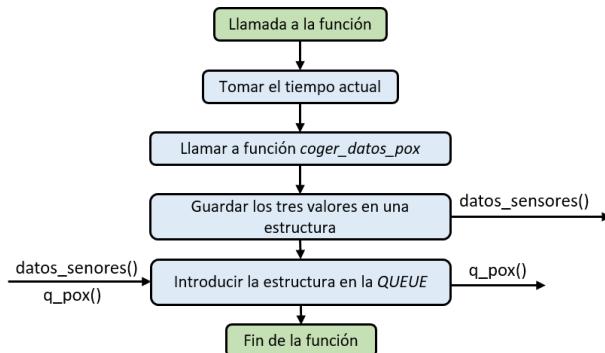


Figura 5.36.- Función generar_buffer_pox

Como se puede observar en la Figura 5.36 esta función llama a otra función de nombre *coger_datos_pox*, que se trata de la función encargada de tomar los datos del sensor *max30102*. Esta función, como ya hacía en la primera versión de la programación, hace uso de la biblioteca del fabricante, y tan solo toma los datos y los devuelve a la función desde donde esta ha sido llamada.

- **Función:** generar_buffer_gsr

La segunda de las funciones, se centra en guardar los datos concernientes a la señal *gsr*. La estructura es exactamente igual que la anterior, la única diferencia es la función a la que esta llama.

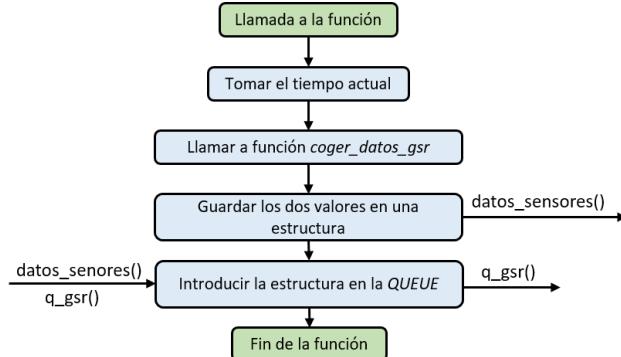


Figura 5.37.- Función generar_buffer_gsr

En este caso la función a la que se llama es la *coger_datos_ecg* (Figura 5.37). En ella se toma el dato del conversor analógico digital y transforma este valor en una tensión y a continuación, en un valor de conductancia, que después se devolverá a la función principal. Como ya hacía la función centrada en el Pulsioxímetro, el dato de la función y el tiempo se introducen en una estructura, que después es guardada en la variable *queue* correspondiente a esa señal.

- **Función:** generar_buffer_ecg

La última función dentro del programa principal, es la que se centra en la última señal fisiológica que queda, el *ecg*. La estructura vuelve a ser exactamente igual que las dos anteriores (Figura 5.38), con la excepción de nuevo, de la función a la que esta llama.

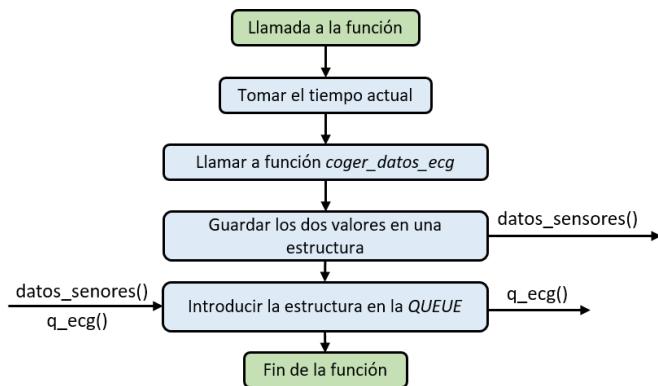


Figura 5.38.- Función generar_buffer_ecg

Como se puede apreciar en la Figura 5.38, la función en este caso tiene como nombre *coger_datos_gsr*. El funcionamiento implementado en esta función no es la del sensor *ECG*, si no que para hacer las pruebas se implementó una adquisición de datos más sencilla. De esta manera se podía comprobar la viabilidad de esta plataforma de una forma más

rápida. Esta función como las anteriores devuelve el dato adquirido a la función principal. Con este dato y con el tiempo que se recoge en el instante inicial, se añaden a una estructura, que finalmente es guardada en la cola (*queue*) correspondiente a esta señal.

- **Main:**

La estructura del *main* de este programa principal es muy parecida a la de la primera versión del código creada, la gran diferencia entre ambas, es que el método de creación de las tareas, al simplificar todos los arrays en una sola estructura *FIFO* (*First In First Out*), se ve simplificada también, ya que solo es necesario introducir esa cola como parámetro (a parte de un identificador, instante inicial, periodo y función a repetir).

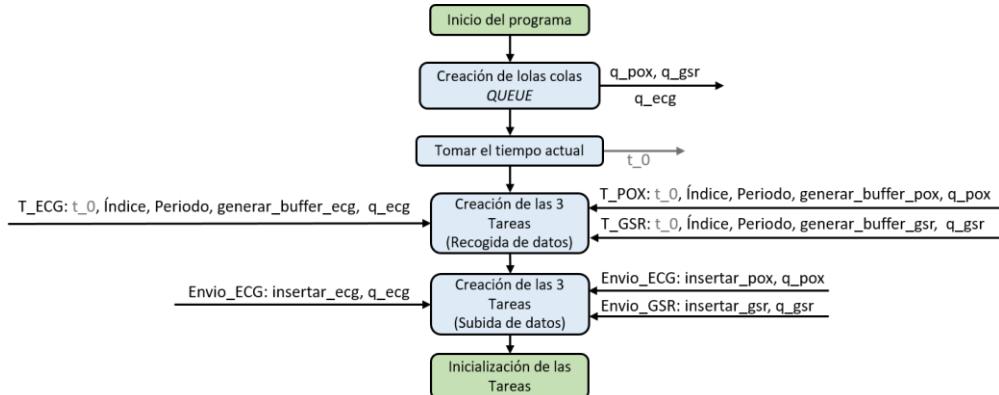


Figura 5.39.- Main del programa principal

En la Figura 5.39 se puede observar ese cambio en la cantidad de parámetros que son necesarios para la creación de las primeras tres tareas. Además, como se comentó al principio del capítulo, esta segunda versión iba a usar un total de seis tareas, las tres periódicas, y otras tres centradas en la subida de datos cuando las colas llegaran a un valor de cien elementos. Debido a esto, en el *main* del programa principal después de la creación de las tareas periódicas, es necesario crear las tres nuevas tareas. Estas solo tienen como parámetros de entrada la función que tiene que llevar a cabo y la cola (*queue*) correspondiente. Finalmente, con todas las tareas creadas, se inicializan todas ellas, para que comience a funcionar el programa.

- **Programa: obtener_valores**

Para mejorar la limpieza de la programación, se decidió que las funciones antes comentadas *coger_datos_pox*, *coger_datos_gsr*, *coger_datos_ecg* estuvieran guardadas en un mismo programa. Ya que todas ellas servían para tomar los datos desde los sensores o conversores analógico-digital. Debido a que este programa solo sirve para tener guardadas las funciones antes listadas, el programa no tiene una parte *main*. Por lo que la estructura del programa es simplemente la definición de las tres funciones (Figura 5.40).

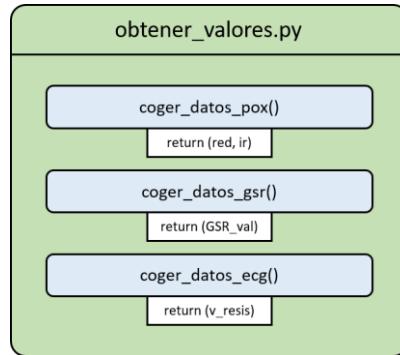


Figura 5.40.- Programa obtener_valores

- **Programa:** connect_bbdd

El último programa que forma parte de la solución es de nuevo un contenedor de funciones. En este caso, las tres funciones que están dentro de este, son las encargadas de enviar los datos introducidos en las colas (*queue*) a la base de datos. Por lo que habrá tres funciones diferentes, una por cada tipo de señal. Todas ellas tienen casi la misma estructura ya que todas toman los datos de la cola, y los suben a la base de datos. La diferencia entre ellas, es que cada una sube los datos a tablas diferentes de la base de datos, por lo que el código necesario para la subida de estos datos es algo diferente en cada una de ellas.

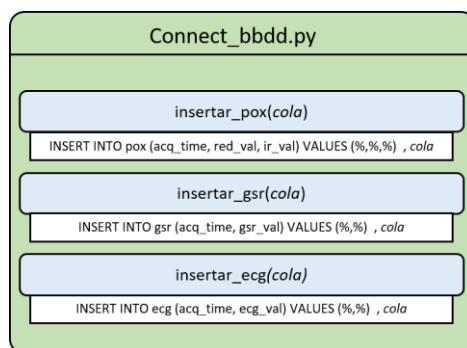


Figura 5.41.- Programa connect_bbdd

En la Figura 5.41 se puede observar la línea de código en la que se diferencian cada una de las funciones. Como se ha comentado anteriormente, este programa tampoco tiene una parte *main*, lo que hace que el programa solo sea un mero contenedor de las funciones para ofrecer un mayor orden en el proyecto.

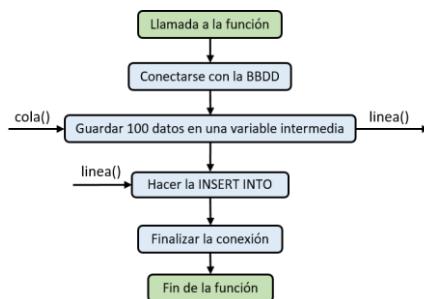


Figura 5.42.- Función tipo insertar en bbdd

El flujograma que siguen las tres funciones es el mostrado en la Figura 5.42. En primer lugar, se conecta el dispositivo a la base de datos. A continuación, se guardan los cien datos de la cola en una variable intermedia, ya que de esta manera se reduce en gran medida los problemas

derivados que se pueden dar cuando dos tareas accedan simultánea y concurrentemente a un mismo registro. Una vez se tiene los datos en la variable intermedia, llamada línea, se hace la instrucción `INSERT INTO` y se finaliza la conexión con la base de datos.

5.3 ESP32

La segunda de las plataformas en las que se decidió implementar este dispositivo fue la placa *ESP32*. A diferencia de las *Raspberry Pi*, los dispositivos *ESP32* no se tratan de ordenadores en miniatura, por lo que no es necesario llevar a cabo ninguna instalación de sistema operativo, y, en consecuencia, tampoco es necesaria una configuración de este.

La configuración de este tipo de dispositivo se hace directamente en el código del programa, por lo que dependiendo de las funcionalidades que se busquen en cada solución, será necesario configurar la *ESP32* de una forma o de otra.

5.3.1 Lenguajes de programación

Todos los modelos de *ESP32* pueden ser programados en dos lenguajes diferentes, *MicroPython*, y en *C* por medio del *IDE-Arduino*. Entre estos dos lenguajes de programación se decidió usar el *Ensamblador* con el *IDE-Arduino*, ya que este tenía más funciones y bibliotecas desarrolladas por otros usuarios específicas para la gestión de Tiempo Real.

Tabla 5.1.- Comparación de lenguajes de programación

Lenguaje		Especificaciones				
		Importar Bibliotecas	Compilación	Velocidad	Bibliotecas Tiempo-Real	Portabilidad (a otros disp.)
MicroPython		Sí, puede	Mediante un sistema de archivos	Más Lento	No tiene	Completa
C (En Arduino-IDE)		Sí, puede	En cada modificación del programa	Más Rápido	Sí, tiene	En circunstancias específicas

En la Tabla 5.1 se puede observar una comparación entre estos dos lenguajes de programación. Lo que decantó la balanza hacia el lado del lenguaje *C* no fue solo las bibliotecas diseñadas específicamente para la gestión del Tiempo-Real, si no que este tipo de lenguaje, al ser de un nivel más bajo, la ejecución de este es mucho más rápida, lo que podría permitir adquirir datos a una frecuencia más alta.

5.3.2 Instalación y configuración del IDE-Arduino

Para poder programar en *C* será necesario el *IDE de Arduino*, que se trata de un programa que convierte lo que el usuario escribe en un lenguaje de programación más intuitivo en el lenguaje *C* como tal, para después compilarlo en la *ESP32* y así ejecutarse nada más se alimente.

La instalación de este programa es muy sencilla. Su instalador puede ser descargado en <https://www.arduino.cc/en/software>. Y una vez descargado solo es necesario seguir los pasos indicados en las instrucciones de instalación dadas en el proceso.

Pero cuando *IDE-Arduino* fue diseñado, no estaba pensado para que fuese usado en otros dispositivos que no fueran modelos *Arduino*. Pero en actualizaciones y versiones posteriores, se habilitó la posibilidad de programar en este *IDE* modelos de otros fabricantes. Debido a que la placa a usar en este caso es la *ESP32*, es necesario configurar y llevar a cabo diferentes acciones para que el programa detecte y sea capaz de compilar código en ella.

El primer paso a dar es abrir el *boards-manager* (Figura 5.43), que se trata del gestor de las tarjetas que el *Arduino-Ide* es capaz de gestionar y compilar.

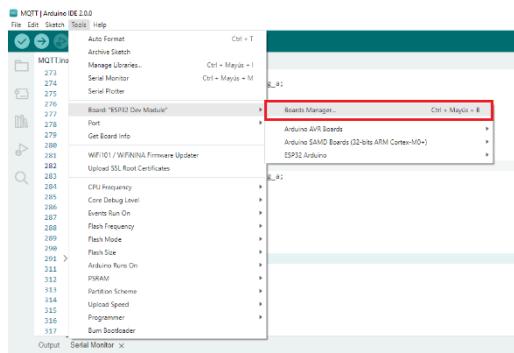


Figura 5.43.- Apertura del gestor de tarjetas

Una vez abierto este *boards-manager* se abrirá una ventana en el lateral izquierdo. En ella se hará la búsqueda “ESP32”. Y se instalará el primer y único resultado “ESP32 Arduino” (Figura 5.44).

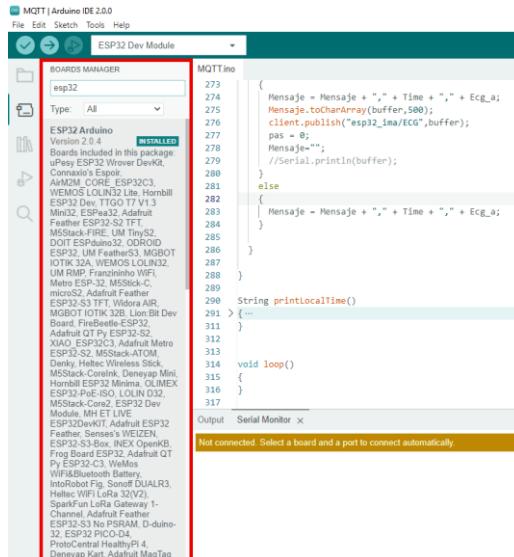


Figura 5.44.- Búsqueda en el gestor de tarjetas

Al instalar este *plug-in* se aumenta la cantidad de modelos de tarjetas que pueden ser programadas mediante el *Arduino-IDE*. Pero para seleccionar la tarjeta que va a ser programada es necesario seleccionarla en la opción *board* de la pestaña *Tools*. El dispositivo que va a ser seleccionado es el llamado “*ESP32 Dev Module*” (Figura 5.45).

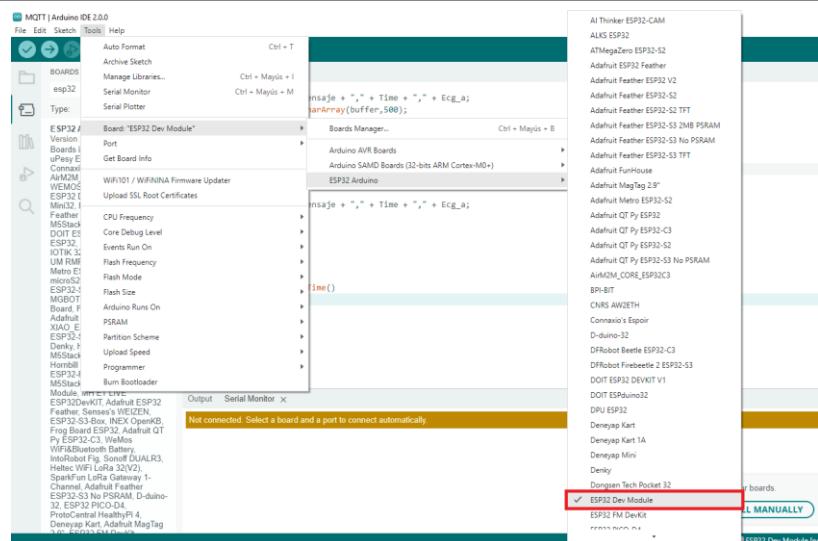


Figura 5.45.- Selección de *ESP32 Dev Module*

En este punto el *IDE-Arduino* ya está completamente configurado para poder crear programas en él, y poder compilarlos en la tarjeta *ESP32*.

5.3.3 Programación en Arduino

La programación de esta placa al igual que en la *Raspberry Pi* está centrada en que la adquisición de todos los datos sea en *Tiempo-Real*. Como se ha comentado cuando se ha explicado de forma general el lenguaje de programación *C (Arduino)* tiene alguna biblioteca que hace que las aplicaciones desarrolladas con ella sean al menos *Pseudo Tiempo-Real*, o lo que es lo mismo, garantista de plazos. Además, al ser este un lenguaje relativamente antiguo y muy usado tiene gran cantidad de bibliotecas que ayudarán al correcto funcionamiento tanto de los sensores como del dispositivo.

5.3.3.1 Soluciones a Tiempo Real en Arduino

La solución encontrada para ofrecer al dispositivo de *Tiempo-Real*, es la del uso de una biblioteca llamada *FreeRTOS* (Figura 5.46). Esta biblioteca se encarga de implementar un sistema operativo en tiempo real en la *ESP32*. Este sistema operativo está diseñado específicamente para dispositivos embebidos.



Figura 5.46.- Logo de Free Real Time Operative Sistem

Fuente: <https://www.freertos.org/>

En específico, esta biblioteca implementa el *Scheduler* (Planificador) que se encarga del orden y de la forma en la que se ejecutan las tareas en el micro de la placa. Además, la biblioteca aporta diferentes funciones como las *Tareas*, *DelayUnitl* (*para la creación de tareas periódicas*) y otras por lo que no es necesario la creación de una biblioteca para completar el funcionamiento del dispositivo.

5.3.3.2 Creación del programa

Debido a que la placa *ESP32* no se trata de un mini-ordenador, esta deberá ser completamente configurada cada vez que se encienda o lo que es lo mismo, cuando se conecte a la corriente. Observando lo hecho en el programa de la *Raspberry Pi* se determina que son cuatro partes las que deben ser configuradas en el instante inicial, *WiFi*, *Reloj Interno*, cliente *MQTT* y configuración del *MAX30102*.

La forma de funcionamiento de este nuevo programa es algo diferente a la desarrollada para la *Raspberry*. Para esta placa, se decidió usar una comunicación *MQTT* con el servidor, y una vez este tuviera los datos, que el propio servidor se encargara de la decodificación de los datos, así como el respectivo guardado de estos en una base de datos.

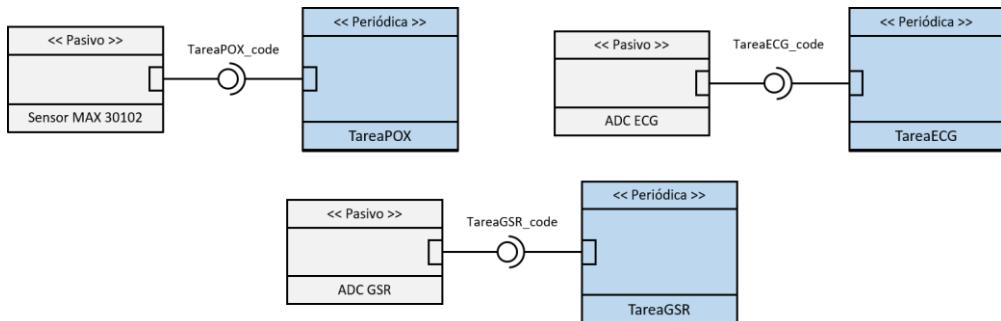


Figura 5.47.- Estructura de Tareas - Programa ESP32

Los datos se recogerán mediante tareas periódicas totalmente independientes unas de otras con diferentes períodos (Figura 5.47). Dentro de cada una de estas tareas no solo recogerá el dato del sensor en cuestión, sino que también se hará la conexión mediante *MQTT* al servidor y se enviarán los datos tomados.

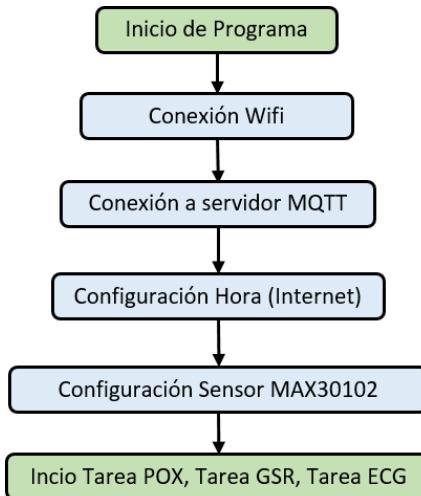


Figura 5.48.-Diagrama de flujo del programa en ESP32

En la Figura 5.48 se presenta el diagrama de flujo seguido para la creación del programa implementado en la *ESP32*. Se pueden observar como se ha comentado anteriormente cuatro configuraciones o conexiones antes de iniciar las tareas diseñadas.

- **Conexión WiFi**

La conexión *WiFi* no es solo necesaria para llevar a cabo la comunicación entre el dispositivo y el servidor, sino que también será usada para obtener la información de la hora en el instante en el que este se conecte a la corriente. Para hacer esta conexión al *WiFi* se hará uso de la

biblioteca *WiFi.h*. Gracias a ella la conexión se hará fácilmente conociendo el *SSID* (nombre de la red) y su contraseña.

▪ Conexión a servidor MQTT

La conexión entre el dispositivo y el servidor será mediante *MQTT*, que se trata de una comunicación *cliente-servidor*, es decir, el cliente publicará los datos que vaya recogiendo en el servidor. Pero para que la comunicación sea correcta, será necesario crear anteriormente un servidor *MQTT* en otro dispositivo (en el caso de este proyecto se implementará en una *Raspberry Pi*).

La conexión a este servidor se hará por medio de la librería *PubSubClient.h*. Gracias a esta librería prediseñada solo será necesario la dirección *IP* del dispositivo con el *broker MQTT* así como el puerto designado como *listener* (el puerto por el que recibe los datos) para conectar el dispositivo con el servidor.

```
client.setServer(mqtt_server_IP, puerto);
client.connect("name");
```

Puede resultar que el intento de conexión con el servidor sea fallido, por lo que la librería tiene una función llamada *client.state* la cual devuelve un número entero que define la casuística del fallo.

▪ Configuración hora

Para la toma de las muestras de las señales fisiológicas, es necesario también conocer el instante en el que estas han sido tomadas, por lo que la configuración de la hora es completamente necesaria. El único modo de configurar la hora de modo correcto es usando internet y tomando la hora de servidores públicos creados solamente con ese fin. Como por ejemplo el usado en este proyecto, *pool.ntp.org*. La biblioteca usada para la obtención de la hora de este servidor es *ESP32Time.h* la cual está dedicada solamente a las placas *ESP32*.

La toma de tiempo solo se hará al iniciar el dispositivo ya que la placa *ESP32* tiene incorporado un reloj en tiempo real, el cual irá calculando la hora en todo instante y gracias a este se podrá ejecutar el programa en tiempo real.

▪ Configuración MAX30102

Finalmente, solo queda configurar el hardware del sensor *MAX30102*, la intensidad de la luz, los colores de led que van a ser usados, el tiempo de muestreo del *ADC*, así como su rango y su amplitud. En la Figura 5.49 se muestran los valores seleccionados para este proyecto.

```
byte ledBrightness = 255; //Options: 0=Off to 255=50mA - 255
byte sampleAverage = 1; //Options: 1, 2, 4, 8, 16, 32
// I have changed the ledmode
byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green
int sampleRate = 1000; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200 - 400
int pulseWidth = 69; //Options: 69, 118, 215, 411 -69
int adcRange = 16384; //Options: 2048, 4096, 8192, 16384 - 16834
```

Figura 5.49.- Valores configuración MAX30102

La biblioteca usada para la configuración es *MAX30105.h* la cual está creada por el propio fabricante. Dentro de esta existen varias funciones, pero la usada para configurar el sensor en el modo seleccionado se hará uso de la función *particleSensor.setup*.

Una vez hecha la configuración de todas las comunicaciones y demás conexiones de la *ESP32*, el siguiente paso es crear el código que será ejecutado en cada una de las tareas. Como se ha comentado al inicio del capítulo, se generarán tres tareas diferentes una por cada señal fisiológica. En ella se tomará la medida y además se enviará mediante la tecnología *MQTT*.

Estas tareas son creadas mediante la biblioteca incorporada por defecto en la placa *freeRTOS*. Esta biblioteca contiene gran variedad de funciones, pero dos de ellas son para la creación de tareas, por un lado, *xTaskCreate* y, por otro lado, *xTaskCreatePinnedToCore*. La diferencia entre estas dos funciones es que la primera de ellas (*xTaskCreate*) el propio código interno de ella selecciona en cuál de los dos núcleos se ejecuta la tarea en cuestión. En cambio, en la segunda de las funciones la selección del núcleo en el que se ejecuta cada tarea está a cargo del programador.

Debido a que se quiere ejecutar la tarea correspondiente al *ECG* en un núcleo y las correspondientes al *POX* y al *ECG* en el otro núcleo, la función seleccionada para la creación de las tareas es, *xTaskCreatePinnedToCore*. La decisión de ejecutar dos tareas en un núcleo y una sola en otro, se tomó en base a los períodos de cada una de ellas. Para que la señal *ECG* pueda ser útil es necesario una frecuencia de muestreo muy grande, por lo que el núcleo en el que se esté ejecutando esta tarea, no podrá hacer más que esta tarea. En cambio, las dos tareas restantes no necesitan una frecuencia de muestreo tan alta para poder tener los valores característicos de las señales fisiológicas por lo que pueden ser ejecutadas en un mismo núcleo.

Pero esta función no crea una tarea periódica que es lo que se necesita para el proyecto. Para crear tareas de este tipo será necesario el uso de una función llamada *DelayUntil()*, cuya traducción literal al castellano es “retrasar hasta”. Esta función permite que una tarea se duerma hasta el instante que se introduzca en ella. Gracias a ello se pueden crear tareas periódicas mediante un bucle infinito cuya primera instrucción sea esta.

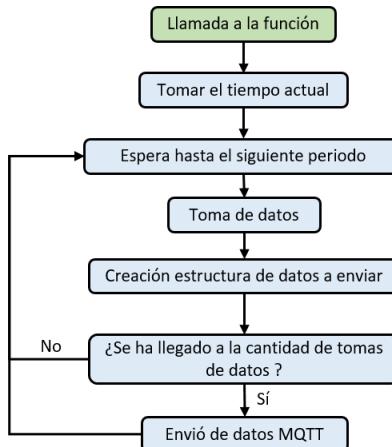


Figura 5.50.- Diagrama de flujo de tarea periódica en ESP32

En la Figura 5.50 se puede observar el diagrama de flujo seguido para la creación de las tres tareas periódicas encargadas de la recogida y envío de las señales fisiológicas. Como se ha comentado, el bucle infinito comienza con la instrucción de espera al siguiente periodo. A continuación, se recoge la señal fisiológica que se esté midiendo. El código en este apartado será diferente en cada una de las tareas debido a que los valores de cada una de las señales se recogen mediante diferentes sensores o de diferentes formas.

Una vez los datos han sido recogidos se introducen en una estructura de datos. La estructura de datos no solo estará compuesta por un instante de medida, sino que, en cada una de las tareas, se tendrán uno o más instantes de medidas en la estructura de datos. De esta manera se puede lograr un menor flujo de datos por medio de la red wifi.



Figura 5.51.- Estructura de datos ESP32

En la Figura 5.51 se observa el modo de estructurar los datos para cada envío, la estructura estará compuesta como se ha comentado por más de un instante de medida y cada instante estará compuesto por un valor de tiempo y otro de la señal fisiológica. Todos estos valores estarán separados por una *coma* (,), para que así el servidor pueda conocer cuándo comienza y acaba cada dato.

▪ Tarea POX

Para la obtención de los valores instantáneos de la señal fisiológica del *Pulsioxímetro*, se hará uso de la antes comentada biblioteca *MAX30105.h*. Y en concreto, *particleSensor.getRed()* y *particleSensor.getIR()*. A continuación, se pasarán los valores enteros obtenidos a una cadena *String* (letras) y se formará la estructura de datos junto a la hora en ese instante.

Para que esta estructura de datos pueda ser enviada, se hará uso también de la función *client.publish()*, de la biblioteca *PubSubClient.h*. En la cual será necesario introducir el mensaje, y el nombre que se haya puesto en *NodeRed* al *MQTT-In* que serán comentados en un capítulo futuro.

▪ Tarea GSR

Para la obtención de los valores instantáneos de la señal fisiológica de la *Respuesta Galvánica de la Piel* no será necesaria ninguna biblioteca. Ya que se hará por medio de uno de los puertos ADC de la *ESP32*. La adquisición de datos se hará mediante la función *analogRead()*, en la que habrá que introducir el número de puerto del que se va a tomar la medida.

El envío de estos datos se hará del mismo modo que la *Tarea POX*, pero la gran diferencia será el número de instantes medidos que se enviarán, ya que como se ha comentado anteriormente, esta señal no necesita una frecuencia de muestreo muy alta, lo que provocará que no sea necesario la gestión del envío de datos.

▪ Tarea ECG

Finalmente, para la obtención de valores de la señal *Electrocardiográfica*, al igual que ocurre con la señal *GSR* no necesitará ninguna biblioteca, debido a que la recogida de datos se hará también por medio de uno de los puertos *ADC* incorporados en la *ESP32*(*analogRead()*).

Esta señal es una señal muy rápida, es decir, necesita una frecuencia de muestreo alta para poder visualizar correctamente la forma de onda real. Debido a esto, la gestión en el envío de datos por medio de la estructura antes comentada es completamente necesaria, ya que sino el sistema podría fallar. Y el envío se hará por medio de la biblioteca y función comentada en la *Tarea POX*.

En este punto, la configuración del dispositivo de recogida de datos y todo su funcionamiento estaría completo, por lo que solo quedaría completar la parte referente a la comunicación *MQTT* del servidor.

5.3.4 Servidor MQTT Raspberry Pi

Toda la gestión de los datos recibidos por medio de *MQTT* se hará en el servidor que, en este proyecto, se encuentra dentro de una *Raspberry Pi*. Por lo que será necesario no solo instalar un *broker MQTT* en la *Raspberry* sino que también habrá que generar un programa que haga la gestión de los datos. Esta gestión estará compuesta por dos partes diferenciadas, la primera de ellas, la decodificación del mensaje y la segunda, la subida de estos datos a la base de datos.

5.3.4.1 Instalación de MQTT y NodeRed

Para la instalación de *MQTT* y en específico su *broker* (que se trata del servidor con el que se comunican los clientes (*ESP32*)) se hará uso de la línea de comando existente en todo sistema operativo basado en Linux. En esta línea de comandos se escribirán las siguientes líneas.

```
$ sudo apt update && sudo apt upgrade
$ sudo apt install -y mosquitto mosquitto-clients
```

Una vez se haya finalizado la instalación será necesario reiniciar el servicio *MQTT* recién instalado mediante la siguiente instrucción en la línea de comandos.

```
$ sudo systemctl restart mosquitto.service
```

Esta línea de comandos tendrá que ser usada siempre y cuando se hagan cambios en la configuración de la comunicación *MQTT*. El modo en el que se hará la comunicación se ajustará en el archivo de configuración *mosquitto.conf* en la ruta */etc/mosquitto/*. Dentro de ese archivo deben aparecer las dos siguientes líneas de texto para que la comunicación funcione correctamente.

```
listener 1883
allow_anonymous true
```

Con todo esto las comunicaciones entre el servidor y el dispositivo de adquisición ya podrían llevarse a cabo, pero para que la *Raspberry Pi* pueda acceder a los datos para hacer la gestión antes comentada, es necesario instalar también la aplicación *Node-Red*.

Node-Red es una herramienta de programación visual (mediante nodos) que ayudará a guardar la estructura de datos compleja procedente de la *ESP32* en archivos *CSV*. Cada una de las publicaciones que se hagan desde el dispositivo de recogida se insertará en una línea nueva del archivo. Debido a que se trabajará con tres señales fisiológicas diferentes, existirán tres archivos *CSV* diferentes.

La instalación de *Node-Red* se hará también por medio de la línea de comandos del sistema operativo. Para esta aplicación las líneas a escribir serían las siguientes.

```
$ bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-
installers/master/deb/update-nodejs-and-nodered)
$ sudo systemctl enable nodered.service # Activación de la aplicación
```

Una vez hecha la instalación de la aplicación, la interface de esta y dónde se desarrollará el proyecto se encontrará en el puerto *1800* de la dirección *IP* del servidor. Por lo que será necesario abrir un navegador web, y en la barra buscadora de este introducir *direcciónIP:1800* para visualizar la pantalla gráfica de la aplicación (Figura 5.52).

La programación de esta herramienta se hace uniendo diferentes nodos, cada uno de estos nodos tienen diferentes funciones. Para llevar a cabo la transformación de los datos recibidos a un archivo *CSV* solo se necesitarán 3 tipos de nodos. En primer lugar, perteneciente a la sección *Network* el nodo de nombre *mqtt in*, en segundo lugar, de la sección *Common*, el nodo *debug*, y finalmente de la sección *Storage* el nodo *write file*.

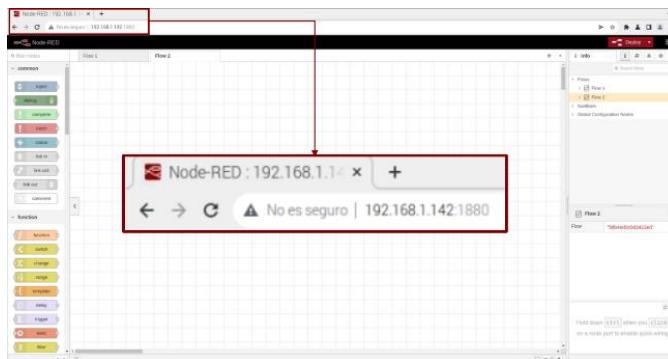


Figura 5.52.- Pantalla inicial Node-Red

La idea es sencilla, y se puede observar en la Figura 5.53 se han creado tres apartados que cumplen la misma función, pero cada uno de ellos centrado en una de las señales fisiológicas.

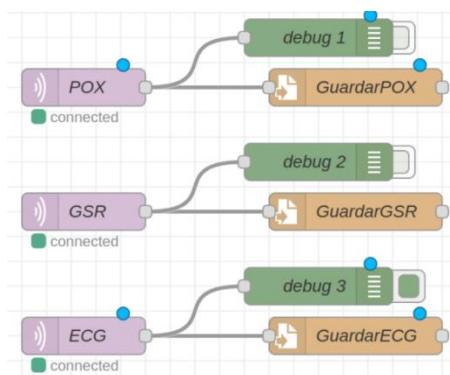


Figura 5.53.- Flujo programado en Node-Red

Mediante el nodo *mqtt in*, se toman las estructuras de datos provenientes de cada una de las tareas periódicas de la *ESP32*. Estas estructuras mediante el nodo *debug*, se podrán visualizar por pantalla, por lo que estos nodos son usados para observar si la comunicación funciona correctamente. Finalmente, mediante el nodo *write file* se escribirán cada una de las estructuras en filas diferentes de un archivo CSV. La configuración de cada uno de estos nodos se muestra en la Figura 5.54.

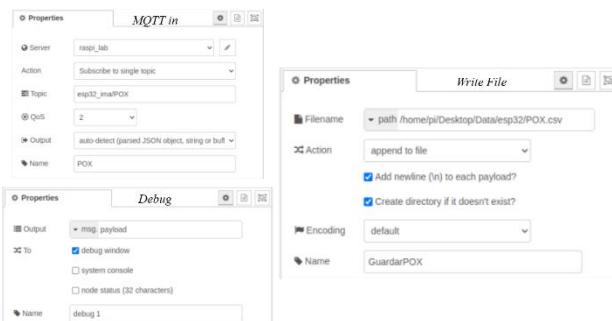


Figura 5.54.- Configuración de nodos

Una vez terminada la configuración y creación del flujo de *Node-Red*, las estructuras de datos se encuentran guardadas en un archivo CSV por filas, en la ruta definida por el diseñador. Pero es necesario decodificar esta estructura, además de filtrar el ruido existente en la señal de *GSR*, así como subir todos los datos a la base de datos dentro del propio servidor.

Estas tres funciones se escribirán en el lenguaje de programación *Python* que como se comentó en capítulos anteriores, se trata de uno de los lenguajes más sencillos y con gran número de bibliotecas sobre todo para la gestión de datos.

5.3.4.2 Función Separador

En este primer programa se definirán dos funciones diferentes cada una de ellas encargada de la decodificación de la estructura de datos provenientes de las señales *POX* y *GSR*. Como se comentó en el Capítulo 5.3.3.2 los datos están separados por comas, por lo que el programa deberá separar cada fila por instantes de toma de datos, para después ordenarlos en orden de adquisición.

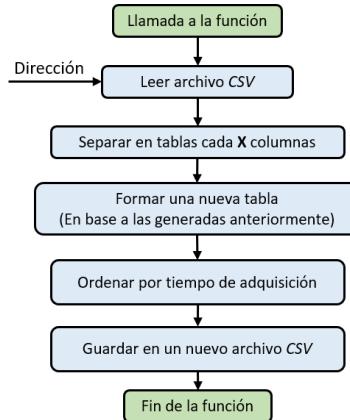


Figura 5.55.- Diagrama de flujo funciones Separador.py

En la Figura 5.55 se muestra el diagrama de flujo de las funciones creadas dentro del archivo *Separador.py*. Para lograr el correcto funcionamiento se utilizó la biblioteca *Pandas*, una biblioteca que contiene gran cantidad de instrucciones para gestionar archivos *CSV* y tablas. Con la instrucción *pd.read_csv()* se recogen los datos de un archivo *CSV* y se convierten en una tabla.

Esta gran tabla se divide por grupos de 3 o 2 columnas (3 → *POX*, 2 → *GSR*) mediante la función *tabla_original.iloc[:, 0:n]*, logrando así superar todos los instantes de toma de datos. Concatenando todas estas nuevas tablas usando la función *pd.concat()* se unirán por filas todas las tablas. En este punto se tendría una muestra por fila, pero no ordenadas. Por lo que antes de guardar la tabla como un nuevo archivo *CSV* mediante la función *tabla_nueva.to_csv()*, será necesario ordenarla mediante *tabla_nueva.sort_values()*.

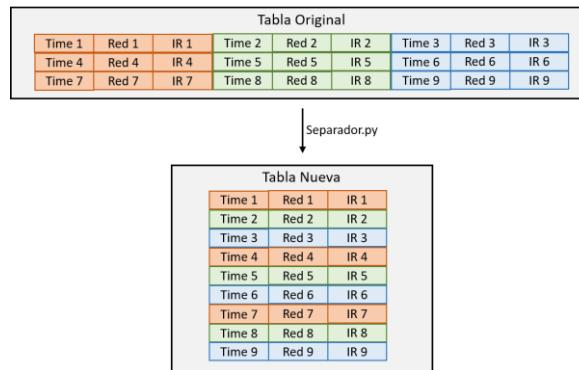


Figura 5.56.- Decodificado de estructura de datos

La Figura 5.56 se muestra un ejemplo de la forma en la que se encuentra la información al inicio del programa *separador.py* y la forma en la que debería estar tras ser decodificado.

5.3.4.3 Función FiltroRuido

El segundo de los programas que también será solo una función creada para filtrar tanto la señal *GSR* como la *ECG*. Estas dos señales necesitan ser filtradas ya que se tratan de señales eléctricas recogidas directamente desde la piel del paciente, y cualquier movimiento de él, o incluso algún otro dispositivo puede generar interferencias en forma de ruido en ellas.

Se introdujo dos filtros diferentes en cada una de las funciones, en primer lugar, el filtro de paso bajo de *Butterworth* y, en segundo lugar, el filtro de *Media Móvil*. El filtro *Butterwoth* de paso bajo lo que hace es solo dejar pasar las frecuencias más bajas a partir de un valor de frecuencia y otro de orden del polinomio. En cambio, el filtro de *Media Móvil* trata de hacer la media de ventanas de los datos. Por ejemplo, si se tienen cien datos, se podrían hacer ventanas de diez, para calcular la media móvil.

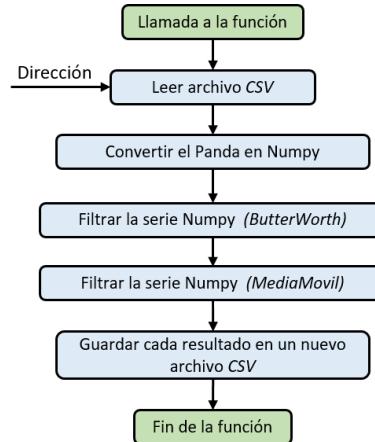


Figura 5.57.- Diagrama de flujo función FiltroRuido.py

La Figura 5.57 muestra el diagrama de flujo seguido para la programación de las funciones dentro del programa *FiltroRuido.py*. Como ocurría en las funciones de *Separador.py*, se necesita dar como entrada la ruta del archivo CSV del que se quiere hacer el filtrado. Las herramientas desarrolladas para el filtrado están dentro de la biblioteca *signal/scipy*. En concreto, la funciones usada serán para el filtro *Butterworth* será *signalfiltfilt()*, y *Datos_or[]rolling().mean()* de la biblioteca *pandas*. Tras crear unas nuevas listas con los datos filtrados, se generan dos archivos CSV nuevos, cada uno con los resultados referentes a los dos modos de filtrado.

5.3.4.4 Programa MoverArchivo

Finalmente, en el programa *MoverArchivo.py* se hacen uso de las funciones creadas en los dos archivos anteriores. Es importante mencionar, que se quería que el trasvase de datos desde los archivos CSV a la base de datos fuera de un tiempo de un minuto. Por lo que se usó la biblioteca de *Tareas Periódicas* creada a lo largo del proyecto. De esta manera se logró que la subida de datos fuera periódica y en *Pseudo tiempo-real*.

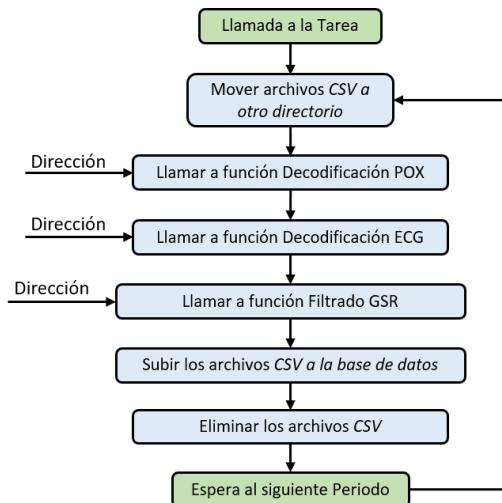


Figura 5.58.- Diagrama de flujo tarea periódica subida de datos

Como se puede apreciar en la Figura 5.58 se observa el fluograma seguido en la creación del código de la tarea periódica. En primer lugar, se mueven los archivos *CSV* a otra carpeta, para evitar el solape y el mal funcionamiento del programa. Una vez movidos a otro directorio, se decodifican los archivos de *POX* y *ECG*, y se filtra el *GSV*. Finalmente, se suben todos los datos de *POX*, *GSR*, *GSR (Median Móvil)*, *GSR (ButterWorth)* y *ECG* a la base de datos montada localmente en la *Raspberry Pi*. Además, una vez finalizado el proceso de subida, se eliminan todos los archivos intermedios usados, para que, en el siguiente periodo, el programa funcione correctamente.

En este punto, ya se tendría configurada y programada la parte de la solución referente al dispositivo de adquisición de datos funcionando en la placa *ESP32* y la parte de comunicación con el servidor y subida a base de datos implementado en una *Raspberry Pi*.

5.4 Montaje hardware

En este nuevo apartado se expondrá el montaje hardware utilizado para que los programas anteriormente comentados funcionen correctamente, tanto en la plataforma *Raspberry Pi*, como en la *ESP32*. El apartado se dividirá en dos diferentes sub apartados, por un lado, la conexión referente al sensor Pulsioxímetro *MAX30102*, y por otro lado el montaje necesario para medir la señal *GSR*.

5.4.1 Conexión Max30102

Para la conexión del sensor de Pulsioxímetro *Max30102* solo son necesarios cuatro cables, ya que como se ha comentado anteriormente, este sensor se comunica con las placas mediante *I2C*. Por lo que dos de estos cuatro cables serán para la comunicación y los dos restantes la alimentación. Es importante mencionar, que la alimentación debe ser a 3,3V por lo que la conexión con la placa deberá ser en los pines que tengan este voltaje.

- Raspberry Pi 4B:

El montaje necesario para establecer la comunicación entre el sensor y el mini-ordenador *Raspberry Pi* es el mostrado en la Figura 5.59.

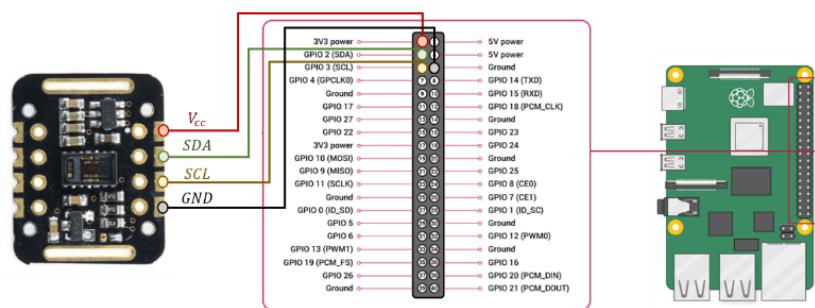


Figura 5.59.- Montaje MAX30102 en Raspberry Pi

En esta se pueden apreciar los cuatro cables antes comentados con diferentes colores. Los encargados de la alimentación del sensor son el rojo (Vcc) y el negro (GND). Los dos restantes, el verde y el marrón (*SDA*, *SCL*), en cambio, son los encargados de establecer la comunicación *I2C*.

- ESP32:

Para llevar a cabo el montaje de este sensor con la placa *ESP32*, se ha seguido el esquema mostrado en la Figura 5.60.

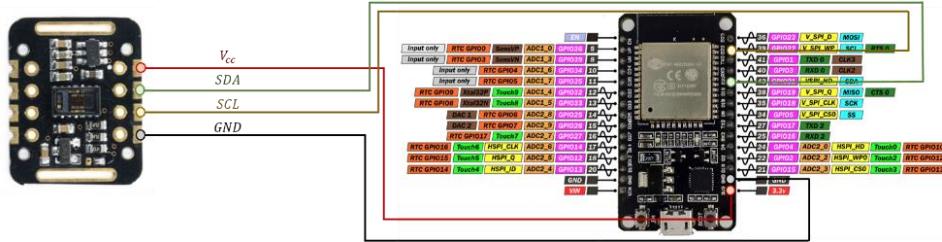


Figura 5.60.- Montaje MAX30102 en ESP32

Al igual que en el montaje para la *Raspberry Pi* y como se ha comentado, solo son necesarios 4 cables, el rojo (*Vcc*) y el negro (*GND*) para la alimentación del sensor y el verde (*SDA*) y el marrón (*SCL*) para la comunicación *I2C*.

5.4.2 Conexión medición *GSR*

A diferencia de la conexión del sensor MAX30102, para poder adquirir la señal *GSR* se necesitan más de un componente electrónico, ya que no se ha optado por no hacer uso de sensor en particular que haga esta medición.

Para comprender el montaje, es necesario comprender que la señal que se va a recoger del cuerpo humano es una tensión, por lo que como se ha comentado en el capítulo 0 y 0 es necesario el uso de un conversor analógico digital (*ADC*).

Además, debido a lo baja que es la tensión que va a ser recogida desde el cuerpo del sujeto, es necesario también un circuito para la amplificación de la señal, para que esta pueda ser leída por el *ADC*.

En cuanto al circuito encargado de recoger la tensión del cuerpo del sujeto en estudio, se hará por medio del circuito mostrado en la Figura 5.30 (Capítulo 0) que se trata de un divisor de tensión básico. Finalmente, para la amplificación de la señal se hace uso en ambos montajes, de un amplificador operacional (*LM358P*) en su configuración no inversora (Figura 5.61). De esta forma variando el valor de las resistencias del circuito se logra incrementar el valor de la tensión de entrada según la fórmula mostrada también en la Figura 5.61

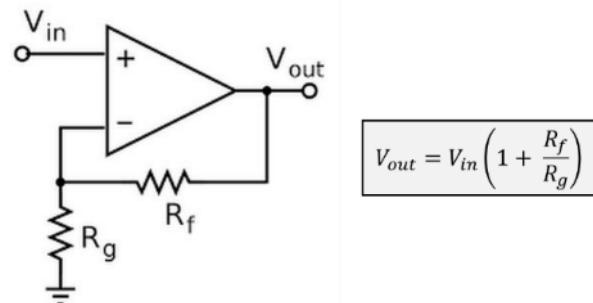


Figura 5.61.- Amplificador operacional - No inversor

- Raspberry pi:

Este dispositivo no tiene incorporado un conversor analógico digital, por lo que será necesario usar uno prefabricado. Para este proyecto, se utilizó el [Grove Base Hat for Raspberry Pi](#) ya que tenía gran cantidad de puertos analógicos con los que medir cualquier otra señal analógica.



Figura 5.62.- Grove Base Hat for Raspberry Pi

Fuente: <https://www.seeedstudio.com/Grove-Base-Hat-for-Raspberry-Pi.html>

Como se puede observar en la Figura 5.62 se muestra en primer lugar este dispositivo, y en segundo lugar cómo es el montaje de este en la propia *Raspberry Pi*. Gracias a este montaje, la conexión entre estos dos dispositivos se hace automáticamente, conectando los pines de la *Raspberry Pi*, con los del *ADC*. Debido a esta conexión estos nuevos pines mantienen las mismas funciones que tenían los pines originales.

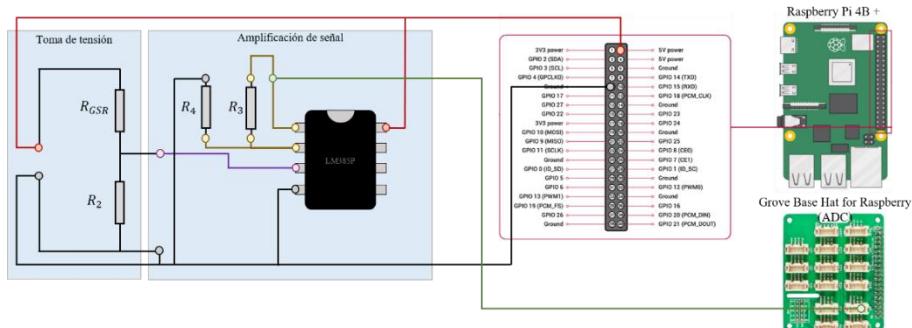


Figura 5.63.- Montaje del sensor GSR – Raspberry Pi

En la Figura 5.63, se puede observar el esquema de conexiones entre todos los elementos que componen el sensor *GSR*. En este caso, solo serán necesarias tres conexiones con la *Raspberry Pi + Base Hat*. Como ya ocurría en el sensor Pulsioxímetro, los cables rojos y negros son los encargados de alimentar tanto el circuito como el chip *LM385P*. Y el cable verde será el que se encargue de tomar la tensión a la salida de la etapa de amplificación y comunicarla al puerto *A0* del *ADC*.

- ESP32

En cambio, la placa *ESP32* sí que incorpora un conversor analógico digital en ella. Por lo que no es necesario un dispositivo extra para poder adquirir este tipo de señales.

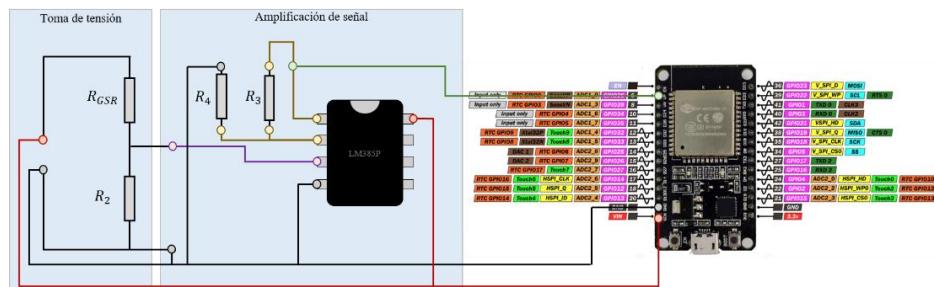


Figura 5.64.- Montaje del sensor GSR – ESP32

En la Figura 5.64 se representa el esquema de conexiones necesario entre todos los componentes que tomar parte en la adquisición de la señal *GSR*. Se puede observar una gran similitud entre los dos montajes, ya que la única diferencia entre los montajes en cada uno de los dispositivos es a qué puertos o pines están conectados los cables rojo (V_{cc}), negro (GND) y verde (V_{out}).

5.4.3 Montaje completo

En este corto sub apartado se va a mostrar el montaje completo en las dos plataformas. En primer lugar, el mini-ordenador *Raspberry Pi* (Figura 5.65, imagen de la izquierda) y en segundo lugar con la placa *ESP32* (Figura 5.65, imagen de la derecha)

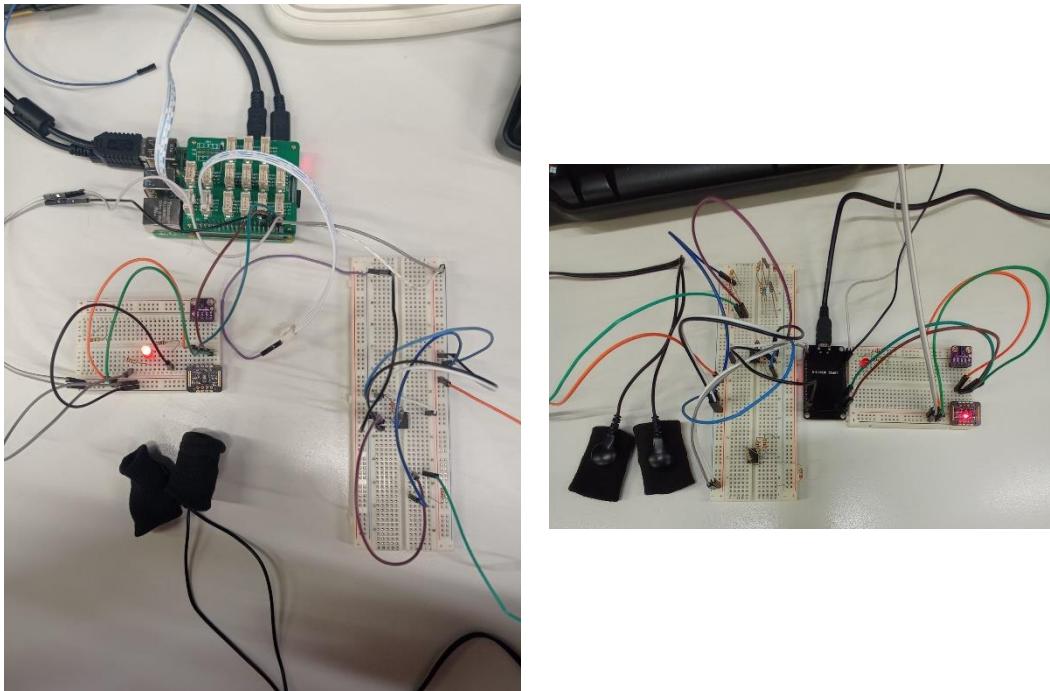


Figura 5.65.- Montaje completo en Raspberry Pi y ESP32

Se puede apreciar que el montaje es exactamente el mismo, la gran diferencia a nivel de montajes son los pines a los que se conectan las tomas de tensión y de tierra, así como en que pin se encuentra el puerto analógico y los puertos de comunicación *SDA* y *SCL* del *I2C*.

5.5 Grafana

Una vez completada la programación, configuración y montaje de las soluciones logradas en el proyecto, se observó que a pesar de que se podrían tener todos los datos en una misma base de datos, también sería necesario implementar alguna herramienta para visualizarlos y observar las cuatro gráficas referentes a las tres señales fisiológicas (La señal *POX* se compone de dos gráficas diferentes).

Una de las herramientas más usadas en la representación gráfica de los datos que existen en una base de datos es *Grafana* (Figura 5.66). Esta herramienta se centra en la representación de series temporales alojadas en cualquier formato de base de datos por lo que puede ser usada en este proyecto.



Figura 5.66.- Logo Grafana
Fuente: <https://grafana.com/>

Para poder trabajar con *Grafana* es necesario instalar esta herramienta en el dispositivo donde se encuentra la base de datos. La instalación es la misma tanto para dispositivos con sistema operativo *Ubuntu* o *Raspbian* y en este caso, se realizó la instalación en una *Raspberry Pi*, siguiendo los pasos descritos en la página oficial de la herramienta (grafana.com/tutorials/install-grafana-on-raspberry-pi/).

En definitiva, solo será necesario introducir las siguientes líneas de comandos, en la consola del sistema operativo.

```
$ wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -
$ echo "deb https://packages.grafana.com/oss/deb stable main" | sudo tee -a /etc/apt/sources.list.d/grafana.list
$ sudo apt-get update
$ sudo apt-get install -y grafana
$ sudo systemctl enable grafana-server
$ sudo systemctl start grafana-server
```

Una vez hecha la instalación, como ya ocurría con la herramienta *MQTT* la interface de su herramienta se encuentra buscando en un navegador de internet la dirección *IP* del dispositivo y el puerto 3000 (*direccionIP:3000*). En la

Figura 5.67 se observa la interface inicial al acceder al puerto de la dirección *IP*.

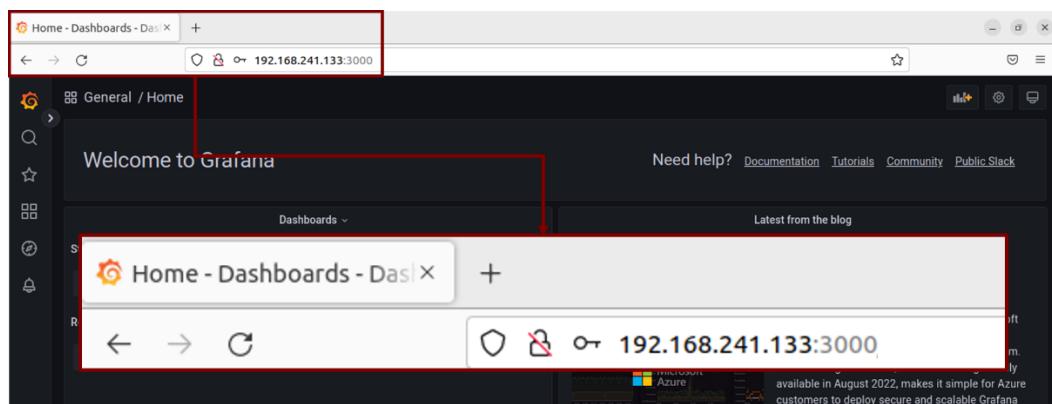


Figura 5.67.- Interface inicial de Grafana

El funcionamiento de *Grafana* se centra en *Dashboards*, es decir se pueden crear diferentes pantallas que pueden ser programadas para visualizar los datos de una base de datos. Para este proyecto se creó una sola pantalla llamada *BIO* de la abreviatura de *señales biológicas*.

A continuación, se deberá conectar *Grafana* a la base de datos, para que la herramienta pueda tener acceso a los datos que se van introduciendo en ella. Para lograr esto, será necesario hacer clic en el símbolo de la rueda dentada, y a continuación, en *Data Sources* como se muestra en la Figura 5.68.

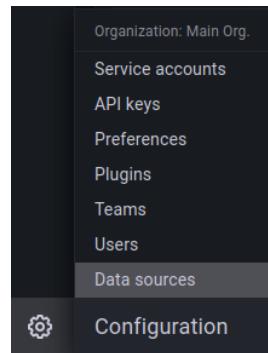


Figura 5.68.- Conexión *Grafana* y base de datos (I)

En la ventana nueva que aparece, se hace clic en el botón azul *Add data source*. A continuación, será necesario seleccionar la aplicación con la que se ha creado la base de datos, por lo que habrá que escribir en la barra de búsqueda *PostgreSQL* y hacer clic en el único resultado de la búsqueda (Figura 5.69).

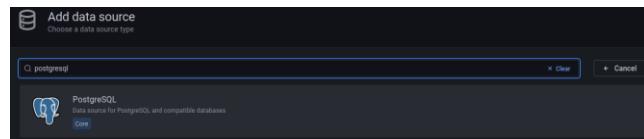


Figura 5.69.- Conexión *Grafana* y base de datos (II)

Antes de finalizar, se tendrá que introducir los datos solicitados para la correcta vinculación de la base de datos y *Grafana*.

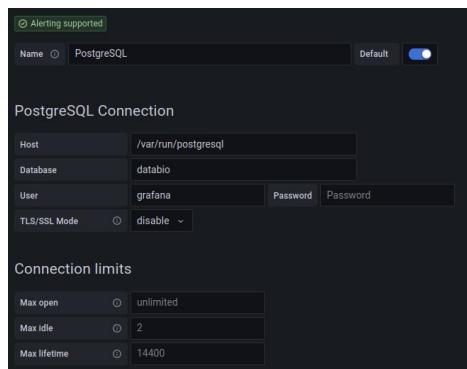


Figura 5.70.- Configuración *Grafana* (Vinculación con BBDD)

En la Figura 5.70 se muestra los datos introducidos para el correcto funcionamiento del proyecto. Uno de los apartados más importantes a llenar es el definido como *User* y *Password*, ya que esto quiere decir que es necesario crear un usuario específico en la base de datos para que *Grafana* pueda acceder a la base de datos. Este usuario deberá tener permisos de superusuario para que funcione correctamente. Para este proyecto, se creó el usuario *Grafana* sin contraseña.

La forma de crear las gráficas es mediante el lenguaje *SQL* que se trata del lenguaje más comúnmente usado para la gestión de base de datos. En primer lugar, se añadirá un nuevo panel (Figura 5.71 -1) y se hará clic en *add a new panel* (Figura 5.71 -2). En consecuencia, aparecerá una nueva pantalla, en la que se dará clic en el botón *EDIT SQL* (Figura 5.71 -3).

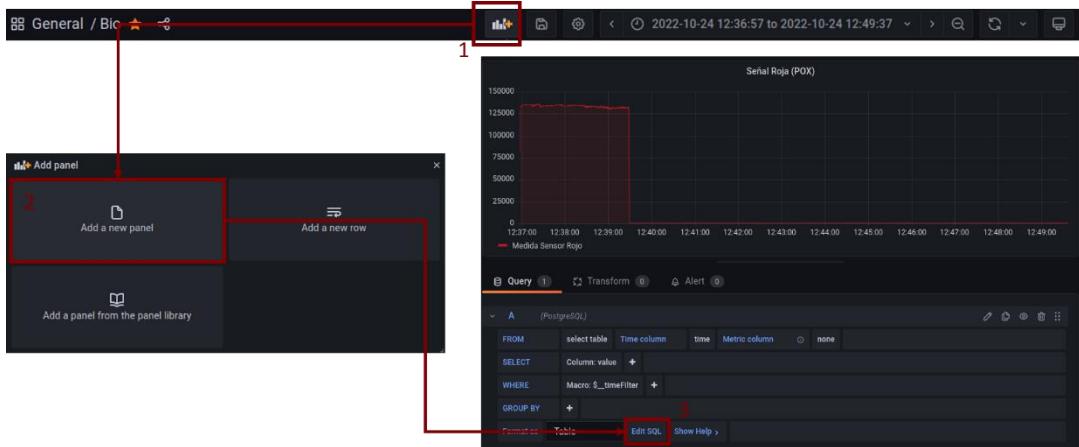


Figura 5.71.- Proceso añadir un nuevo panel

Las líneas de código *SQL* que fueron escritas para obtener los datos de la base de datos a la que *Grafana* estaba conectada son las siguientes.

```

SELECT
    acq_time as "time",
    red_val as "Medida Sensor Rojo"
FROM pox
ORDER BY 1
  
```

Dentro de esta pantalla se introdujeron cuatro gráficas diferentes para así observar los cuatro parámetros característicos de las tres señales fisiológicas que están siendo recogidas. En la Figura 5.72 se presenta la pantalla creada para la visualización de las señales: *Señal Roja (POX)*, *Señal Infrarroja (POX)*, *Sensor GSR* y *Sensor ECG*.

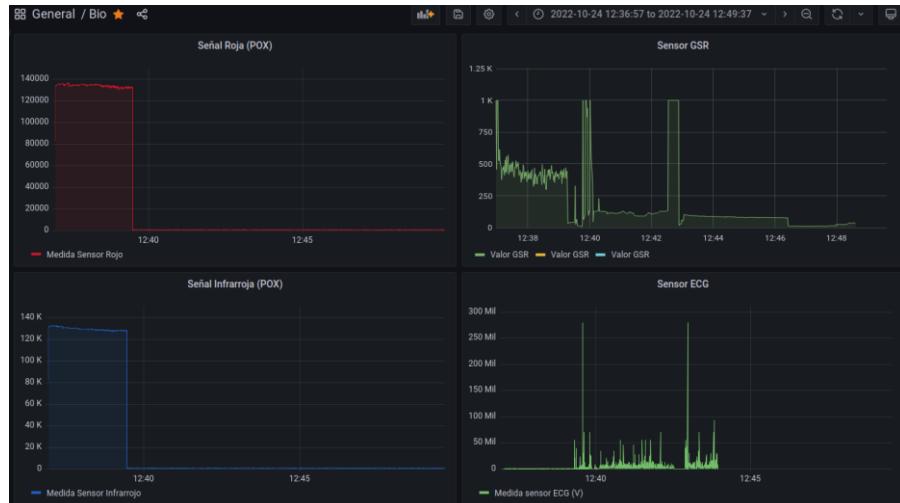


Figura 5.72.- Dashboard generado

Como se puede apreciar, en la gráfica correspondiente a la señal *GSR*, no solo se grafican los resultados directamente recogidos por el dispositivo de adquisición, sino que se representan también la señal filtrada con los dos métodos anteriormente programados, mediante *Butterworth* y mediante una *Media móvil*.

CAPITULO 6

RESULTADOS

6 Resultados

En este sexto capítulo, se mostrarán los ensayos y los resultados obtenidos de cada uno de los pasos dados para la obtención de los dispositivos desarrollados. El capítulo se dividirá en dos grandes subapartados: los resultados obtenidos para el desarrollo de la solución implementada en *Raspberry Pi* y los obtenidos en la *ESP32*.

6.1 Raspberry Pi

Como se ha comentado en el capítulo de metodología, para la *Raspberry Pi* se crearon dos versiones diferentes del programa que define el funcionamiento del dispositivo. Debido a esto, en este subapartado se mostrarán los resultados obtenidos en ambas versiones del programa.

6.1.1 Primera versión del programa

Para comprobar el correcto funcionamiento del programa, los primeros ensayos hechos fueron las pruebas de adquisición de datos de los tres sensores de forma independiente. De esta manera, se podrá discernir la frecuencia de muestreo idónea para la captación correcta de las señales fisiológicas.

- Pulsioxímetro:

La señal del pulsioxímetro como ya se ha comentado anteriormente viene representada por dos señales diferentes. Esta señal representa las pulsaciones del corazón del paciente, debido a esto, las frecuencias de muestreo probadas fueron: *1 Hz*, *10 Hz* y *50Hz*.

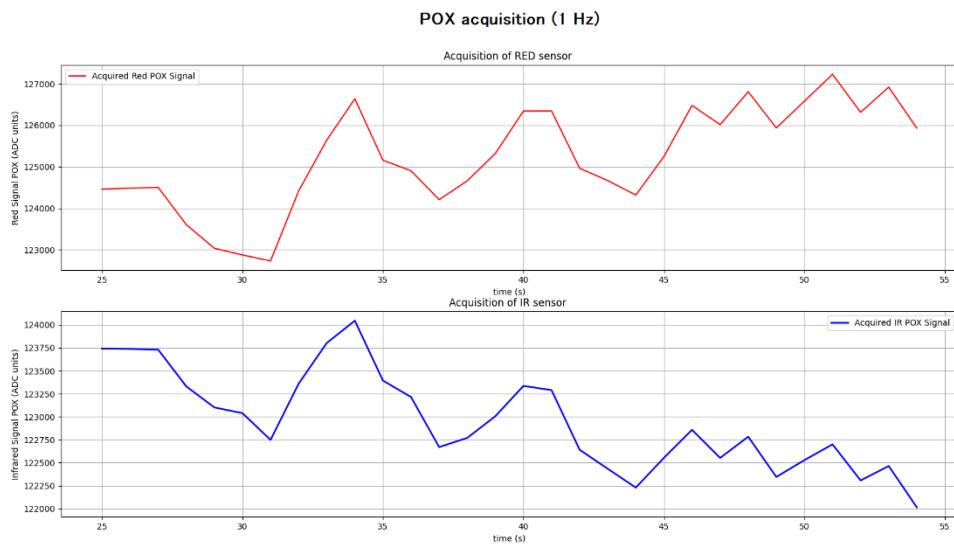


Figura 6.1.- Pulsioxímetro (1 Hz) - Raspberry Pi

En la Figura 6.1 se representa la gráfica resultante tras la toma de datos durante un minuto. Como se puede observar, tanto en la gráfica superior, como en la inferior, no se logra visualizar la forma de onda (dientes de sierra) característica de la señal del pulsioxímetro. Por lo que la frecuencia de adquisición de datos de *1 Hz* no es suficiente ni válida para una correcta toma de datos. Debido a esto se probó a aumentar la frecuencia de muestreo a *10 Hz*.

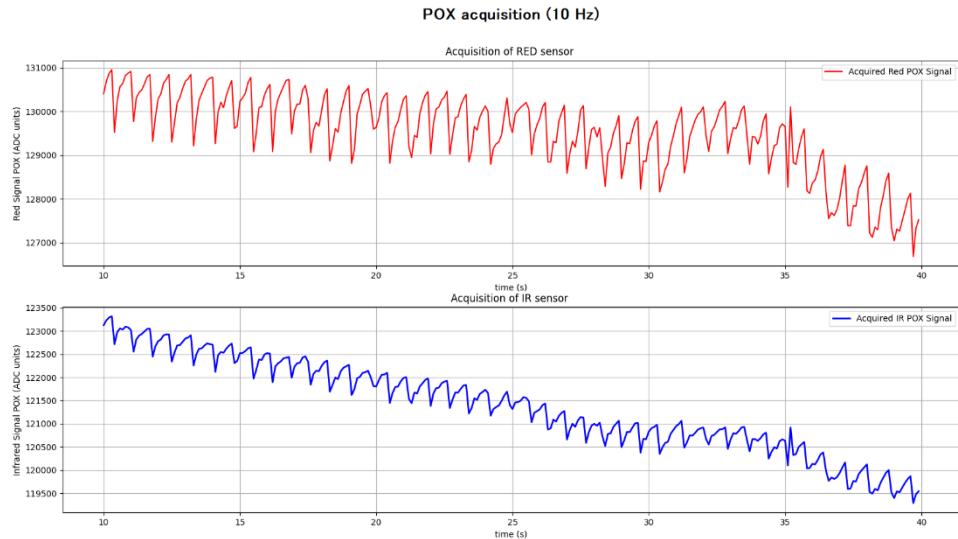


Figura 6.2.- Pulsioxímetro (10 Hz) – Raspberry Pi

Las gráficas que representan los datos obtenidos durante un minuto a una frecuencia de muestreo de 10 Hz se muestran en la Figura 6.2. En este caso en ambas gráficas, se puede apreciar perfectamente la forma de onda característica de las señales de los pulsioxímetros. Debido a esta perfecta representación de la forma de onda se tomó como correcta la frecuencia de 10 Hz para la recogida de la señal del pulsioxímetro. De todas formas, se probó a aumentar hasta los 50 Hz para intentar obtener una señal de mayor resolución.

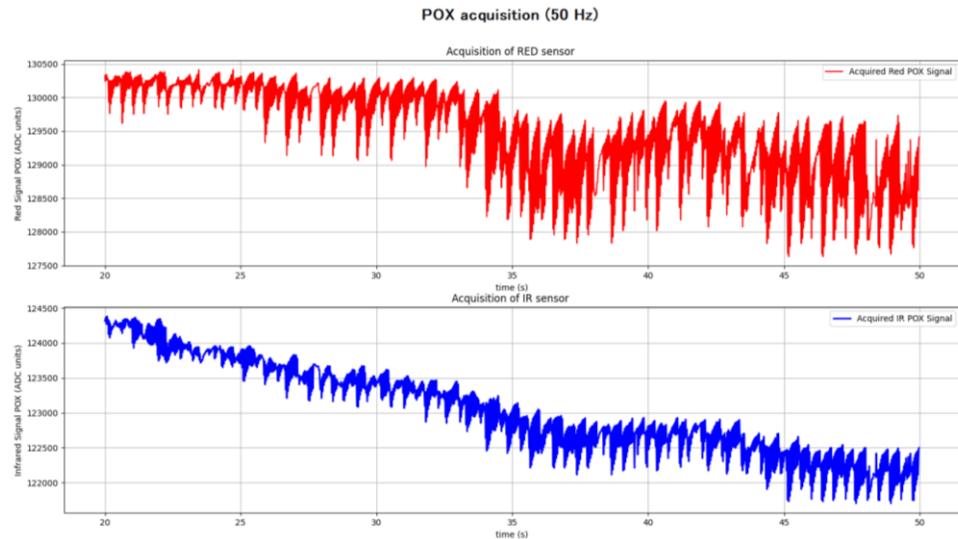


Figura 6.3.- Pulsioxímetro (50 Hz) - Raspberry Pi

Finalmente, en la Figura 6.3 se muestran las gráficas obtenidas tras estar adquiriendo datos durante un minuto a una frecuencia de muestreo de 50 Hz . Se aprecia en este caso que la forma de onda se ve algo perturbada debido al ruido que se introduce debido a esta alta frecuencia. Este ruido es muy difícil de eliminar ya que no en todos los instantes es de la misma intensidad y frecuencia. Debido a esto, se toma como invalida 50 Hz como una frecuencia de muestreo para la correcta toma de esta señal fisiológica.

De este modo, se toma como frecuencia de muestreo idónea para la recogida de datos de la señal del pulsioxímetro es 10 Hz , ya que se trata de una frecuencia a la que no hace falta aplicar ningún filtro de ruido, debido al submuestro, como si ocurre con la frecuencia de 50 Hz .

- Respuesta Galvánica de la Piel:

En cuanto a la señal de la respuesta galvánica de la piel, las frecuencias probadas fueron: $50, 10, 1 \text{ Hz}$. En este caso, se comenzó con una frecuencia de muestreo alta y después se fue reduciendo para observar cuál es la frecuencia mínima con la que se puede trabajar. Es necesario comentar, que en las gráficas de las señales *GSR*, se representan tres series temporales diferentes: en *azul* la señal adquirida, en *rojo* la señal filtrada por medio de un filtro de *Butterword*, y en *verde* la señal filtrada mediante una media móvil.

El ensayo seguido para la comprobación del funcionamiento de este sensor, fue por medio de la medición de un periodo de tiempo, en el que se comienza con un minuto de relajación, una vez terminado, alrededor de un minuto de resolución de un juego que supondría un esfuerzo mental y que provocaría al paciente de estudio cierto nerviosismo, y finalmente otro minuto de relajación.

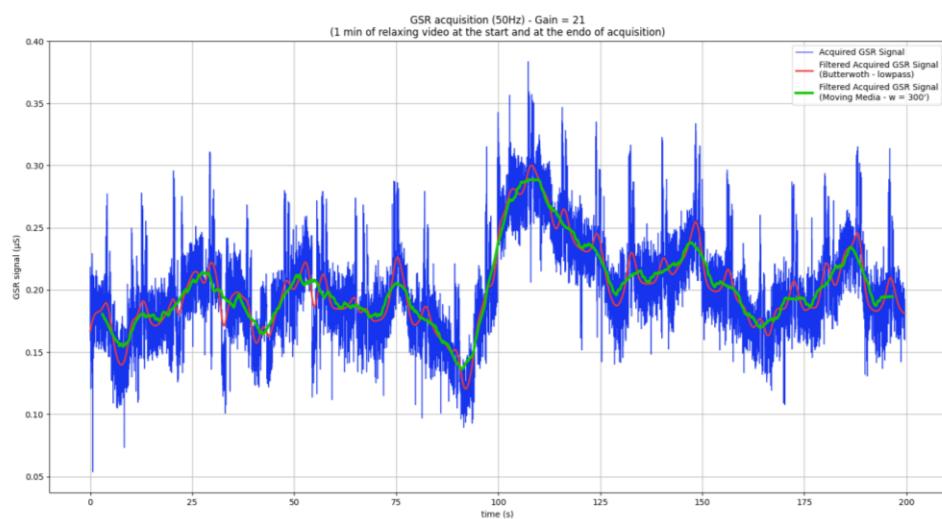


Figura 6.4.- GSR (50 Hz) - Raspberry Pi

La primera gráfica mostrada en la Figura 6.4 muestran los datos adquiridos durante un periodo de ensayo, se puede observar que la señal original (*azul*) contienen bastante ruido, pero aplicando los filtros de *Butterword* y de media móvil, se logra solucionar. Se aprecia una gran resolución en la señal, donde se observa perfectamente el instante de finalización de relajación en el instante 80. Por ello, se determina que la frecuencia de 50 Hz es funcional para la recogida de esta señal.

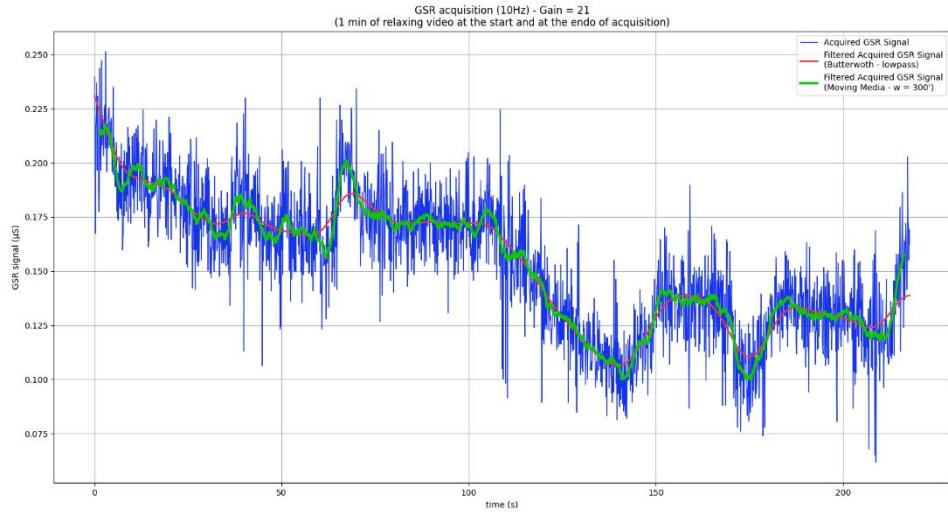


Figura 6.5.- GSR (10 Hz) - Raspberry Pi

Reduciendo la frecuencia de muestreo a los 10 Hz , se obtiene la gráfica mostrada en la Figura 6.5, donde el ruido se ve algo reducido, pero de nuevo, haciendo uso de los filtros se logra una señal más definida, donde se puede apreciar alrededor del instante 60 cómo se da un pico, relacionado con la salida del estado de relajación. Por lo que la frecuencia de 10 Hz también sería una frecuencia funcional para la adquisición.

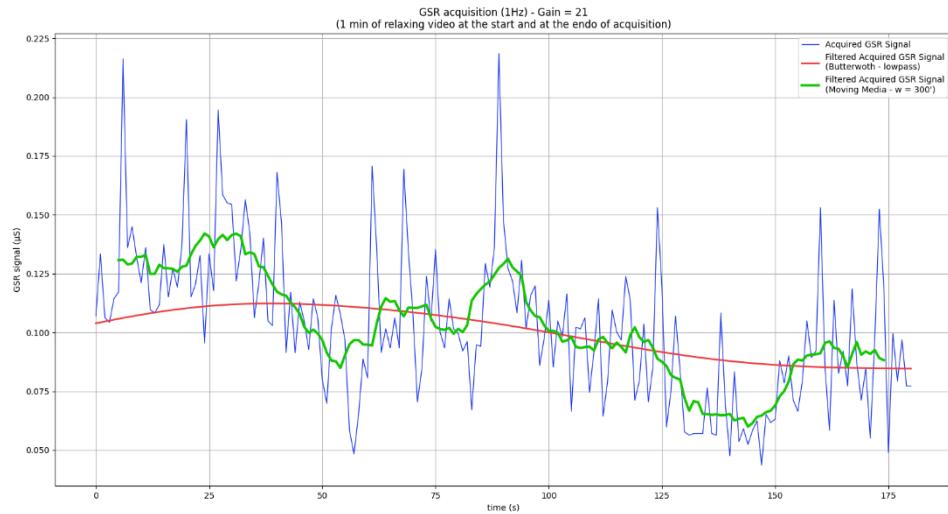


Figura 6.6.- GSR (1 Hz) - Raspberry Pi

Finalmente, se muestra en la Figura 6.6 se muestra la gráfica tras un periodo de ensayo con la última de las frecuencias, y la más baja, 1 Hz , o lo que es lo mismo, una toma de dato por segundo. El ruido sigue estando presente en la señal, y el filtro de media móvil logra eliminarlo, lo que permite apreciar la señal de manera que alrededor de los instantes 60 y 80 se observan unos picos de nuevo referentes a la finalización del periodo de relajación, y una complicación en el juego. Por lo que esta frecuencia también funciona correctamente para adquirir la señal.

Teniendo en cuenta que las tres frecuencias de muestreo son capaces de recoger la señal *GSR* de una forma correcta, y que se quiere reducir al máximo la carga computacional del dispositivo, la frecuencia de muestreo seleccionada para el dispositivo fue 1 Hz .

- Electrocardiograma:

Debido a que no se ha llegado a aplicar el sensor para la adquisición de la señal electrocardiográfica, se ha determinado que la frecuencia de muestreo para la correcta toma de esta señal, sea de *100 Hz*. Esta frecuencia, supone una toma de cien datos por segundo, o lo que es lo mismo seis mil datos por minuto. La frecuencia cardiaca en reposo suele oscilar entre los sesenta y cien pulsos por minuto, por lo que cada latido se vería representado por cien o sesenta datos, que se trata de una resolución completamente viable y suficiente para la correcta adquisición de la señal electrocardiográfica.

Tras finalizar estos tres ensayos, se definió como frecuencias óptimas de muestreo de cada una de las señales fisiológicas para este dispositivo, son las mostradas en la Tabla 6.1.

Tabla 6.1.- Frecuencias de muestreo evaluación teórica

	Señal Pulsioxímetro	Señal GSR	Señal ECG
Frecuencia de muestreo	10 Hz	1 Hz	100 Hz

Por lo que el siguiente ensayo hecho fue la prueba de estas tres frecuencias de muestreo en el programa desarrollado como primera opción, para comprobar que este podría cumplir los plazos, sin fallar en ningún instante. Los cambios realizados en el programa fueron los mostrados en la Figura 6.7, donde se introdujeron las frecuencias de muestreo a cada una de las tareas.

```
tarea1 = mp.Process(target = BiblioTareasPeriodicas.ServerLocal.Tarea_Periodica_Sensores_3Arr, args = (tiempo0, 1, 0.1, 0.1, CogerGuardarDatosPOX, y_Ir,y_Rojo, y_t_POX))
tarea2 = mp.Process(target = BiblioTareasPeriodicas.ServerLocal.Tarea_Periodica_Sensores_2Arr, args = (tiempo0, 2, 0.1, 0.1, CogerGuardarDatosGSR, y_GSR, y_t_GSR))
tarea3 = mp.Process(target = BiblioTareasPeriodicas.ServerLocal.Tarea_Periodica_Sensores_2Arr, args = (tiempo0, 3, 0.01, 0.01, CogerGuardarDatosECG, y_ECG, y_t_ECG))
tarea4 = mp.Process(target = BiblioTareasPeriodicas.ServerLocal.Tarea_Periodica_Sensores_1Arr, args = (tiempo0, 4, 60, 61, coms, y_Ir, y_Rojo, y_t_POX, y_GSR, y_t_ECG))
```

Figura 6.7.- Cambios introducidos en el programa (RPi - V1)

El sistema comenzó a funcionar correctamente, pero después de unos pocos ciclos de adquisición, aparece que la tarea 3, la encargada de tomar los datos correspondientes a la señal *ECG* es abortada (Figura 6.8).

```
3 = 72.06018277168274
4 = 80.06369956701661
Infrarrojo: 100 Rojo: 10
Limpios:

3 = 84.00017275810242
4 = 90.06368532180787
Infrarrojo: 100 Rojo: 10
Limpios:

3 = 96.00016322135926
Abort:T 3
Process Process-4:
Traceback (most recent call last):
  File "/usr/lib/python3.9/multiprocessing/process.py", line 315, in _bootstrap
    self._run()
  File "/usr/lib/python3.9/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/home/pi/GITHUB/Biosignals-HW/Programas/Solucion_Raspberry/1.-Solucion_Imanol/BiblioTareasPeriodicas_ServerLocal.py", line 50, in Tarea_Periodica_Sensores_2Arr
    time.sleep(next_time - time.time())
ValueError: sleep length must be non-negative
```

Figura 6.8.- Raspberry Pi - Versión 1

Se probó en varias ocasiones sin hacer ningún tipo de modificación, y dependiendo de la carga computacional que tuviera el sistema, el fallo de plazo en la señal *ECG* se daba antes o después del tiempo mostrado en la Figura 6.8.

Además, observando los datos subidos a la base de datos en este ensayo final, se observa que, en todos ellos, se pierden datos. Es decir, uno o más de un dato no llega a subirse a la base de datos como se puede observar en la Figura 6.9.

POX.csv	GSR.csv
2023-01-14 19:59:23.237126 129676 122855	2023-01-15 20:44:35.44039 0,132009014329836
2023-01-14 19:59:23.437174 130400 123124	2023-01-15 20:44:37.440473 0,121171396940973
ECG.csv	
2023-01-15 20:46:05.440518 0,147607336544176	
2023-01-15 20:46:07.440445 0,127266206194393	

Figura 6.9.- Ejemplo perdida de una o más muestras

Por lo que, el sistema a esta frecuencia de muestreo y mediante esta primera versión del programa no era fiable. De ahí, la elaboración de una segunda versión del programa buscando otros modos de gestión del tiempo real.

6.1.2 Segunda versión del programa

Implementando las modificaciones mostradas en el capítulo anterior, se creó la segunda versión del programa. Con esta versión se quiso comprobar si se lograba ya no solo que el programa cumpliera con todos los plazos con las frecuencias de muestreo de la Tabla 6.1, sino que también, lograra no perder ninguna muestra en la subida de datos.

La implementación de las frecuencias de muestreo se hizo con los cambios mostrados en la Figura 6.10.

```
llenar_bufferPOX = Process(target=BiblioTareasPeriodicas.Tarea_Periodica_Sensores_1Arr, args =(t0, 1, 0.1,0.1, generar_buffer_pox, q_pox))
llenar_bufferGSR = Process(target=BiblioTareasPeriodicas.Tarea_Periodica_Sensores_1Arr, args =(t0, 2, 1,1, generar_buffer_gsr, q_gsr))
llenar_bufferECG = Process(target=BiblioTareasPeriodicas.Tarea_Periodica_Sensores_1Arr, args =(t0, 3, 0.01,0.01, generar_buffer_ecg, q_ecg))
```

Figura 6.10.- Cambios introducidos en el programa (RPi – V2)

Los resultados obtenidos en este ensayo fueron prácticamente iguales a los de la versión uno del programa. Nada más lanzar a ejecutar el programa, las tareas tres y uno, las referentes al *Pulsioxímetro* y a la señal *Electrocardiográfica*, fallan el primero de los plazos.

```
Lanzamos la query...
Lanzamos la query...
Lanzamos la query...
Fuera de plazo falso --> Tarea 3
Fuera de plazo falso --> Tarea 1
```

Tabla 6.2.- Frecuencias de muestreo - Segunda versión

Debido a este ensayo tan poco producente, la siguiente decisión, fue la reducción de las frecuencias de muestreo de las tareas que fallaron, para así poder observar si esta versión del programa era capaz de no perder ninguna de las muestras recogidas. Tras varios intentos, las frecuencias a partir de las que el programa comienza a funcionar sin ningún fallo son las mostradas en la Tabla 6.3.

Tabla 6.3.- Frecuencias de muestreo final

	Señal Pusioxímetro	Señal GSR	Señal ECG
Frecuencia de muestreo	2.5 Hz	1 Hz	2.5 Hz

Con estas frecuencias, el programa no fallaba ninguno de los plazos, llegando a estar más de una hora recogiendo y subiendo muestras a la base de datos. Donde se ha podido observar las formas de ondas resultantes de cada una de las señales.

- Señal electrocardiográfica:

Como se comentó en el subapartado anterior, la frecuencia mínima que es necesaria para recoger esta señal fisiológica es de *100 Hz*, por lo que la reducción de esta a los *2.5 Hz* hace que este sistema no sea apto para la recogida fiable de esta señal

- Señal GSR:

En este caso, no se ha modificado la frecuencia de muestreo con la que se toman los datos, por lo que la señal, como se puede apreciar en la Figura 6.11, es completamente fiable y utilizable como alimentación a una Inteligencia Artificial.

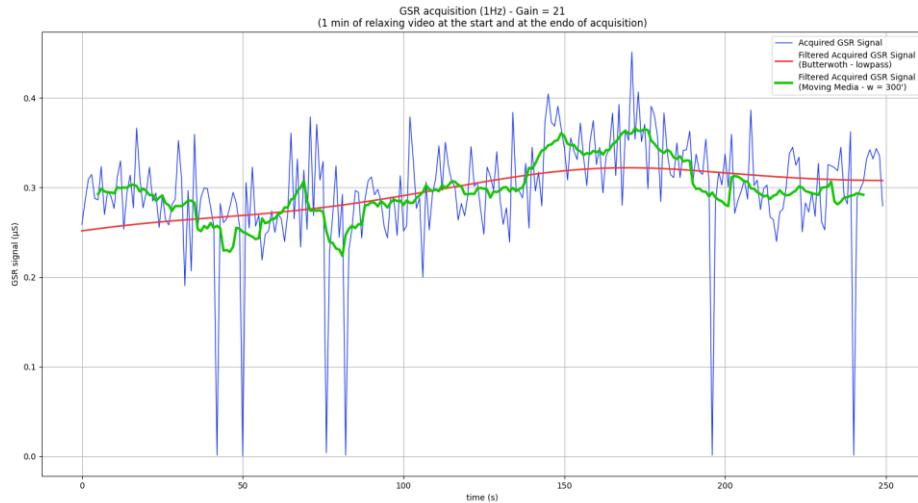


Figura 6.11.- GSR (1 Hz) - Raspberry Pi (V2)

Se puede apreciar en la figura, cómo alrededor del segundo sesenta, se crea un pico en la señal correspondiente a la salida del estado de relajación, y comienzo de un juego de memoria.

- Señal POX:

Finalmente, en el caso de la señal del pulsioxímetro la cual tiene una forma de onda de dientes de sierra, con esta frecuencia se ve simplificada una forma de onda de picos y valles.

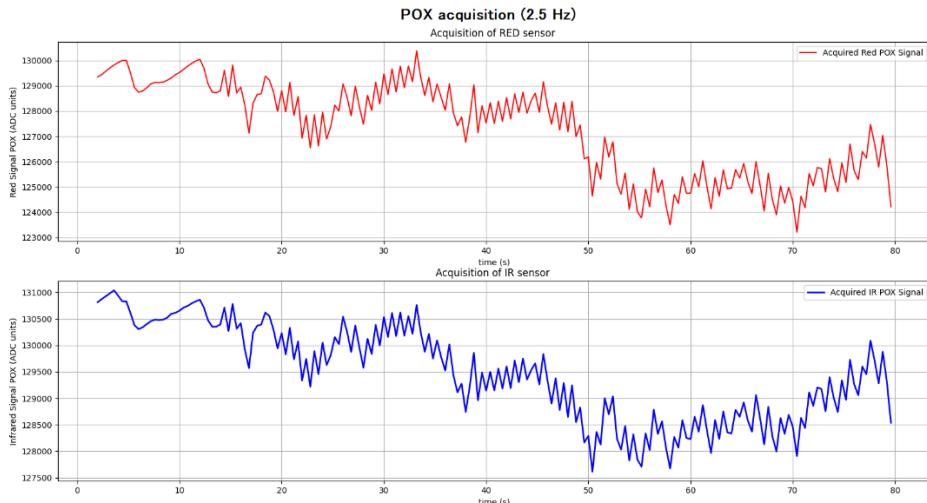


Figura 6.12.- Figura 6.13.- Pulsioxímetro (2.5 Hz) – Raspberry Pi (v2)

Se aprecia en la Figura 6.12, que al haber reducido en un cuarto la frecuencia de muestreo la cantidad de muestras por pico o pulso del corazón es bastante menor, lo que hace que la forma de onda se simplifique de esa manera y no sea funcional para la detección de pulso.

6.2 ESP32

Habiendo llegado a sacar el máximo partido a la solución implementada en la *Raspberry Pi*, como ha sido comentado, se trabajó en una solución aplicada en la tarjeta electrónica llamada *ESP32*.

Los ensayos que se realizaron para discernir si el código desarrollado para la placa *ESP32* es funcional, estuvieron basados en los hechos para la solución implementada en la *Raspberry*. Debido a esto, se tomaron como frecuencias ideales de muestreo, las obtenidas en los ensayos anteriores: para el pulsioxímetro *10 Hz*, para la señal *GSR 1 Hz* y finalmente para la señal *ECG* una frecuencia de *100 Hz*.

El objetivo era lograr que se recogieran todos estos datos de forma simultánea y sin perder ninguno de ellos. Como ha sido comentado en el capítulo anterior, las tareas encargadas de recoger los datos de las señales de *POX* y *ECG* tienen cierta complicación en el envío mediante internet, ya que se envía más de un dato por cada comunicación por medio de *MQQT*, por lo que será necesario observar con qué cantidad de datos por paquete el programa podrá funcionar.

6.2.1 Pulsioxímetro

Se decidió hacer diferentes pruebas para descubrir cuántos datos deberían de comunicarse en cada conexión con el servidor. La primera de las pruebas hechas fue comunicando un dato por conexión. De esta manera, se podría observar el archivo CSV generado en el servidor, y comprobar si no se pierde ninguno de los datos capturados o si hay algún otro tipo de problema.

2023-01-23 09:52:25.806	127208	128299
2023-01-23 09:52:25.906	127163	128317
2023-01-23 09:52:26.006	127210	128362
2023-01-23 09:52:26.106	127224	128341
2023-01-23 09:52:26.206	127348	128408
2023-01-23 09:52:26.306	127463	128477
2023-01-23 09:52:26.406	127529	128518
2023-01-23 09:52:26.506	127605	128414
2023-01-23 09:52:26.606	126886	128159
2023-01-23 09:52:26.706	126873	128240
2023-01-23 09:52:26.806	126871	128294
2023-01-23 09:52:26.906	126754	128261
2023-01-23 09:52:27.006	127046	128315
2023-01-23 09:52:27.106	127196	128383
2023-01-23 09:52:27.206	127424	128388
2023-01-23 09:52:27.306	127168	128102
2023-01-23 09:52:27.406	126723	127935
2023-01-23 09:52:27.506	126886	128097
2023-01-23 09:52:27.606	127323	128371
2023-01-23 09:52:27.706	127594	128430
2023-01-23 09:52:27.806	127334	128332
2023-01-23 09:52:27.906	127348	128335
2023-01-23 09:52:28.006	127362	128248

Figura 6.14.- POX.csv un dato por conexión

En la Figura 6.14 se puede observar los datos guardados en dos segundos. Se aprecia que ninguno de estos datos está fuera de plazo además de que existe un dato cada décima de segundo, por lo que la recogida de esta señal con una frecuencia de *10 Hz* y de un dato por conexión está garantizada.

Para conocer hasta qué punto se puede reducir la cantidad de conexiones, se aumenta la cantidad de datos que se comunican en cada conexión con la base de datos. El valor máximo encontrado fue de 5 datos por conexión.

2023-1-23 10:19:02.106	1024	1177	2023-1-23 10:19:02.206	1027	1184	2023-1-23 10:19:02.306	1025	1179	2023-1-23 10:19:02.406	1032	1171	2023-1-23 10:19:02.506	1031	1170
2023-1-23 10:19:02.606	1030	1176	2023-1-23 10:19:02.706	1032	1178	2023-1-23 10:19:02.806	1030	1184	2023-1-23 10:19:02.906	1036	1187	2023-1-23 10:19:03.006	1038	1185
2023-1-23 10:19:03.106	1046	1184	2023-1-23 10:19:03.206	1044	1185	2023-1-23 10:19:03.306	1033	1187	2023-1-23 10:19:03.406	1030	1183	2023-1-23 10:19:03.506	1033	1177
2023-1-23 10:19:03.606	1029	1174	2023-1-23 10:19:03.706	1032	1175	2023-1-23 10:19:03.806	1031	1174	2023-1-23 10:19:03.906	1027	1177	2023-1-23 10:19:04.006	1032	1181
2023-1-23 10:19:04.106	1029	1176	2023-1-23 10:19:04.206	1031	1180	2023-1-23 10:19:04.306	1044	1193	2023-1-23 10:19:04.406	1037	1181	2023-1-23 10:19:04.506	1036	1180
2023-1-23 10:21:27.106	1030	1187	2023-1-23 10:21:27.206	1027	1187	2023-1-23 10:21:27.306	1034	1188	2023-1-23 10:21:27.406	1036	1180	2023-1-23 10:21:27.506	1035	1176
2023-1-23 10:21:27.606	1036	1178	2023-1-23 10:21:27.706	1034	1180	2023-1-23 10:21:27.806	1039	1178	2023-1-23 10:21:27.906	1033	1172	2023-1-23 10:21:28.006	1038	1174
2023-1-23 10:21:28.106	1029	1183	2023-1-23 10:21:28.206	1036	1188	2023-1-23 10:21:28.306	1030	1191	2023-1-23 10:21:28.406	1039	1180	2023-1-23 10:21:28.506	1031	1175
2023-1-23 10:21:28.606	1040	1183	2023-1-23 10:21:28.706	1038	1185	2023-1-23 10:21:28.806	1035	1191	2023-1-23 10:21:28.906	1035	1188	2023-1-23 10:21:29.006	1031	1182
2023-1-23 10:21:29.106	1028	1180	2023-1-23 10:21:29.206	1038	1179	2023-1-23 10:21:29.306	1033	1183	2023-1-23 10:21:29.406	1029	1174	2023-1-23 10:21:29.506	1032	1176
2023-1-23 10:21:29.606	1031	1184	2023-1-23 10:21:29.706	1030	1185	2023-1-23 10:21:29.806	1039	1193	2023-1-23 10:21:29.906	1034	1190	2023-1-23 10:21:30.006	1032	1191

Figura 6.15.- POX.csv cinco datos por conexión

Como se puede apreciar en amarillo en la Figura 6.15, el sistema falla en la recogida de datos durante 2 minutos, tras observar en el ordenador lo sucedido, se apreció que el sistema se reinició debido a que, en el instante siguiente, el 2022 1-23 10:19:05:606 estaba fuera de plazo.

Finalmente, la cantidad de datos con la que se logró que el sistema funcionara más de una hora de forma continua fue de tres datos por conexión. De esta forma, se reducía la cantidad de conexiones hechas al servidor, consiguiendo que el sistema garantizara todos los plazos.

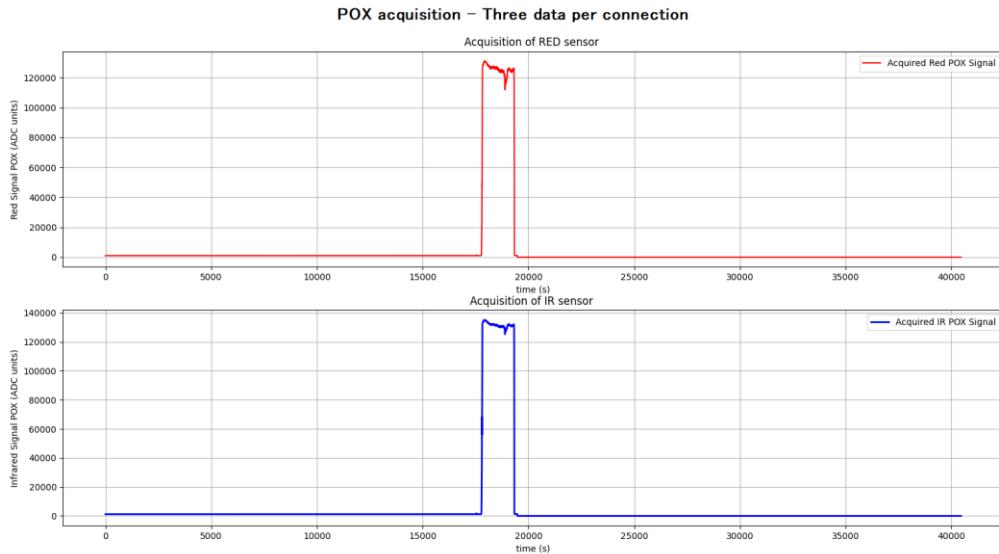


Figura 6.16.- Recogida de señal *POX* constante durante una hora

En la Figura 6.16 se observa las gráficas logradas en esta hora de recogida de datos de forma continua. Es necesario comentar, que estas gráficas no contienen información característica, ya que, en el ensayo, se dejó al dispositivo recoger medidas, pero sin ningún sujeto del que recogerlas.

6.2.2 GSR

En cuanto a las medidas recogidas por los electrodos, se decidió que al tener una frecuencia de muestreo tan baja, comparada con las demás, la cantidad de datos por conexión con el servidor fuera de uno. De esta forma se logró que el dispositivo lograse recoger medidas por más de una hora.

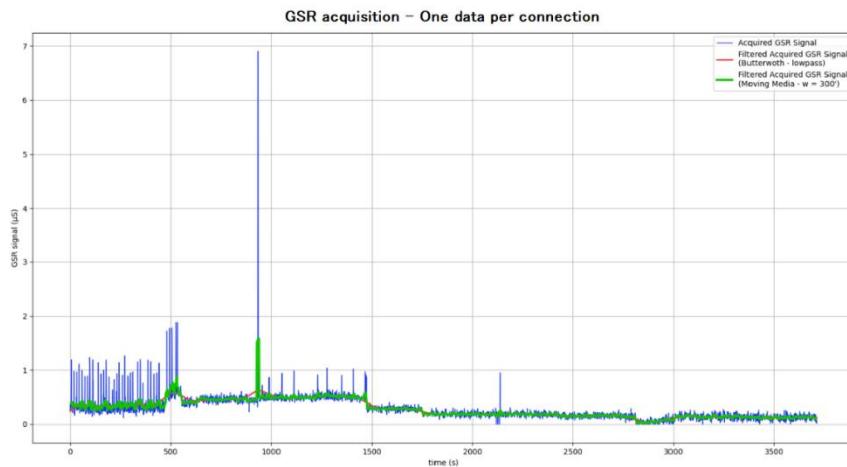


Figura 6.17.- Recogida de señal *GSR* constante durante una hora

En la Figura 6.17 se observa los datos en forma de gráfica del ensayo de más de una hora de duración. Al igual que ocurría con la medición de *POX*, las formas de ondas mostradas en ella, no son características, ya que la recogida de datos no tenía ningún sujeto al que hacérselas.

6.2.3 ECG

Finalmente, para la tarea encargada de la recogida de la señal fisiológica electrocardiográfica, se decidió probar con una comunicación de cinco datos por conexión, haciendo así, que cada conexión se hiciera cada 50 ms , rebajando de una forma bastante notable la cantidad de conexiones a lo largo de un ensayo.

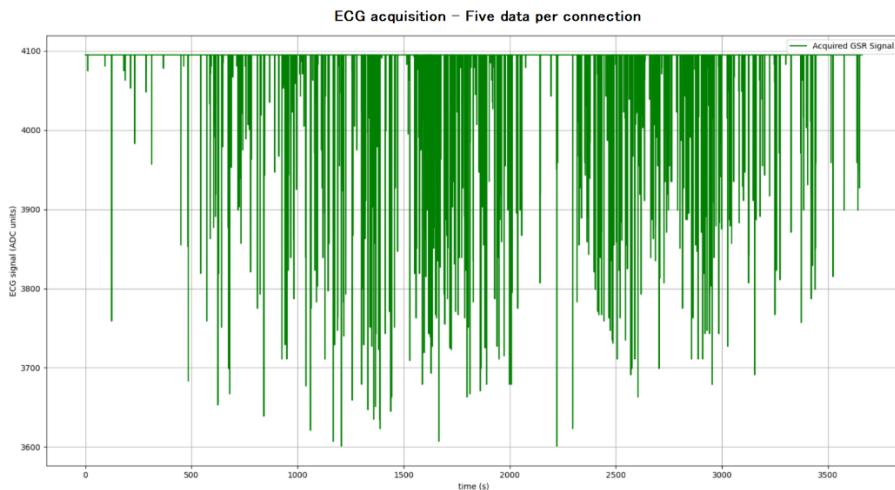


Figura 6.18.- Recogida de señal *ECG* constante durante una hora

En la gráfica representada en la Figura 6.18 se muestran los datos recogidos en un ensayo de una hora. La forma de onda observable es simplemente una señal constante en el valor 4095 con cierto ruido. Esta medida es correcta ya que al no tener implementado en sensor *ECG*, el puerto analógico del que se está tomando las medidas se conectó a una tensión de 3.3V , la tensión máxima que mide el conversor analógico-digital.

6.2.4 Implementación completa.

Una vez se conocen los valores óptimos de datos por conexión de cada una de las tareas, así como a la frecuencia a la que estas van a recoger datos, el último ensayo es el de validación del sistema en una situación de funcionamiento para lo que está diseñado. Es decir, una medición de las señales fisiológicas del sensor *Pulsioxímetro*, señal de la *Respuesta galvánica de la piel (GSR)*, y finalmente la señal *Electrocardiográfica* que en este solo recogerá datos a la frecuencia de 100Hz . Por lo que las características con las que se van a recoger datos son las mostradas en la Tabla 6.4.

Tabla 6.4.- Características de las tareas ESP32

	Señal POX	Señal GSR	Señal ECG
Frecuencia de muestreo	10 Hz	1 Hz	100 Hz
Datos por conexión	3	1	5

Conociendo estas frecuencias de muestreo, el ensayo con el que se comprobó el correcto funcionamiento del dispositivo fue, mediante una recogida de todas las señales durante cinco minutos. Las gráficas representativas de los datos recogidos durante estos instantes se muestran en la Figura 6.19.

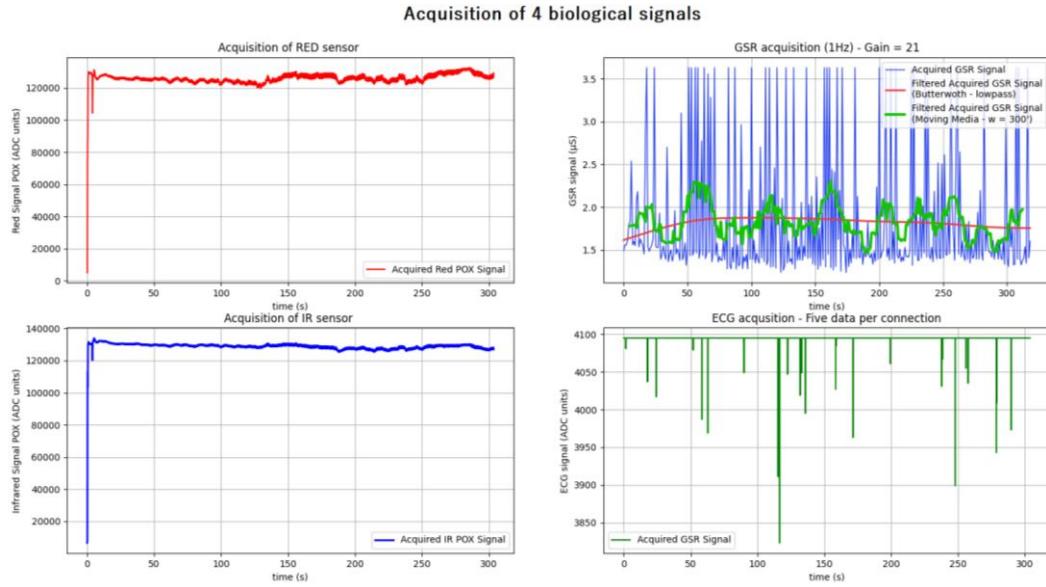


Figura 6.19.- Grafica de datos recogidos (5 minutos)

Como se puede observar en la figura anterior, las cuatro señales recogen datos durante los cinco minutos que duró el ensayo. La señal recogida desde el sensor *Max30102* se representa en las gráficas situadas a la izquierda de la ella. La señal *GSR* se representa en la gráfica superior derecha, y finalmente, la señal *ECG* se muestra en la gráfica inferior derecha.

Esta última, como se ha comentado, no ofrecen ningún tipo de información real, ya que el sensor electrocardiográfico no se llegó a implementar. Pero el puerto analógico al que este iría conectado, se conectó a una tensión de 3.3V, de ahí que, en la gráfica correspondiente a esta señal, sea una línea casi constante en el valor 4096 (Valor máximo del conversor analógico digital).

CAPITULO 7

CONCLUSIONES

7 Conclusiones

En este último capítulo se ahondará en las conclusiones obtenidas a lo largo del desarrollo de este trabajo. No solo se ofrecerán conclusiones generales sobre el proyecto, sino que también se darán conclusiones sobre las soluciones en los dos dispositivos en los que se ha desarrollado la programación. Además, se comentarán las posibles acciones o líneas de investigaciones futuras a las que ha dado pie este proyecto.

7.1 Solución Raspberry PI

Con el dispositivo *Raspberry PI* se ha logrado adquirir los datos de tres señales diferentes de forma simultánea. Pero este hecho se logró rebajando las frecuencias de muestreo, lo que produce una descontextualización de la señal recogida. Es decir, no se recoge correctamente los parámetros característicos de las señales en estudio. A pesar de ello, tanto los programas como las librerías creadas a lo largo de la implantación de esta solución, pueden ser útiles para la recogida de señales de ciclos más largos.

Por lo que la gran conclusión que puede ser sacada de este primer apartado del proyecto, es que se ha logrado desarrollar un dispositivo capaz de medir tres señales, de una frecuencia mayor de 1 Hz . Al mismo tiempo los datos recogidos son subidos mediante *Wi-Fi* a una base de datos, y todo ello de forma simultánea.

7.2 Solución Esp32

En cuanto a la solución implementada en el dispositivo *ESP32*, sí que ha logrado cumplir con los objetivos marcados al inicio del proyecto. La solución logra recoger tres señales fisiológicas como la señal del *Pulsioxímetro*, la señal *GSR* y la señal *Electrocardiográfica* (de forma experimental) caracterizándolas de forma idónea para el siguiente procesado de datos.

Al igual que con la solución anterior, los datos recogidos se envían mediante *Wi-Fi* a una base de datos que puede estar localizada en cualquier dispositivo con sistema operativo Ubuntu ya que el proceso de instalación y configuración es el mismo que se ha hecho para la *Raspberry Pi*.

En conclusión, la solución propuesta para la problemática inicial cumple con los requisitos establecidos, tanto de resolución como de comunicación y guardado de los datos.

7.3 Conclusiones del proyecto

Debido a que el desarrollo del proyecto se ha centrado en varios lenguajes de programación se puede concluir que se trata de un proyecto que ha observado las diferentes opciones o soluciones que se puede aportar a una misma problemática.

De cada una de estas soluciones se ha hecho una comprobación concienzuda de los resultados obtenidos no solo para determinar si estas soluciones eran consistentes en lo que al muestreo de la señales concierne, si no que los dispositivos en sí fueran funcionales, es decir, que se tratase de un dispositivo ligero que cumpliera la característica de portabilidad determinada en los objetivos iniciales.

7.4 Acciones futuras

Gracias al desarrollo de este proyecto se han podido observar diferentes acciones futuras que se pueden llevar a cabo para seguir desarrollando el dispositivo y la línea de investigación en la que se centra el proyecto.

La primera de las acciones en las que se podría trabajar es en la implementación del sensor para la señal *electrocardiográfica*, ya sea mediante un circuito electrónico desarrollado personalmente, o mediante un sensor prediseñado y creado para la recogida de esta señal.

Otras de las acciones futuras se centrarían en la mejora del comportamiento del dispositivo desarrollado. Como puede ser la introducción de diferentes pulsadores con los que se pueda gestionar el encendido y apagado del dispositivo.

También se podría desarrollar diferentes modos de funcionamiento del dispositivo para así poder establecer la duración de la recogida de datos. Para facilitar el uso del dispositivo una acción futura también podría ser la implementación de una interface gráfica en la que visualizar los datos recogidos y el modo usado.

Pero todo esto debe ser implementado en un encapsulado que pueda ser portable y que no comprometa de ninguna manera la adquisición de las señales fisiológicas. Además de buscar la posición más idónea para la colocación de este encapsulado logrando que para el usuario no sea una carga.

CAPITULO 8

REFERENCIAS BIBLIOGRÁFICAS

8 Referencias bibliográficas

- [1] R. Martinez Rodriguez, «Diseño de un sistema de detección y clasificación de cambios emocionales basado en el análisis de señales fisiológicas no intrusivas», UPV/EHU, 2016. Accedido: nov. 15, 2022. [En línea]. Available: <https://dialnet.unirioja.es/servlet/tesis?codigo=112122&info=resumen&idioma=SPA>
- [2] U. Zalabarria Pena, «Identification of the stress and relaxation level in people, based on the study and the advanced processing of physiological signals related to the activity of the autonomic nervous system», 2020. Accedido: nov. 15, 2022. [En línea]. Available: <http://addi.ehu.es/handle/10810/50668>
- [3] U. Zalabarria, E. Irigoyen, R. Martinez, y A. Lowe, «Online robust R-peaks detection in noisy electrocardiograms using a novel iterative smart processing algorithm», *Appl Math Comput*, vol. 369, mar. 2020, doi: 10.1016/J.AMC.2019.124839.
- [4] U. Zalabarria, E. Irigoyen, R. Martinez, M. Larrea, y A. Salazar-Ramirez, «A Low-Cost, Portable Solution for Stress and Relaxation Estimation Based on a Real-Time Fuzzy Algorithm», *IEEE Access*, vol. 8, pp. 74118-74128, 2020, doi: 10.1109/ACCESS.2020.2988348.
- [5] U. Zalabarria, E. Irigoyen, y A. Lowe, «Diagnosis of atrial fibrillation based on arterial pulse wave foot point detection using artificial neural networks», *Comput Methods Programs Biomed*, vol. 197, p. 105681, dic. 2020, doi: 10.1016/J.CMPB.2020.105681.
- [6] M. I. Ahmad, M. J. Singleton, P. D. Bhave, H. Kamel, y E. Z. Soliman, «Atrial cardiopathy and stroke mortality in the general population», doi: 10.1177/1747493019876543.
- [7] «Handbook of Psychophysiology», *Handbook of Psychophysiology*, ene. 2007, doi: 10.1017/CBO9780511546396.
- [8] B. Medić, «The role of autonomic control in cardiovascular system: Summary of basic principles», *Medicinski podmladak*, vol. 67, n.º 1, pp. 14-18, oct. 2016, doi: 10.5937/MEDPODM1601014M.
- [9] Y. Zhang, J. Cui, K. Ma, H. Chen, y J. Zhang, «A wristband device for detecting human pulse and motion based on the Internet of Things», *Measurement (Lond)*, vol. 163, oct. 2020, doi: 10.1016/J.MEASUREMENT.2020.108036.
- [10] S. Sakphrom, T. Limpiti, K. Funsian, S. Chandhaket, R. Haiges, y K. Thinsurat, «Intelligent Medical System with Low-Cost Wearable Monitoring Devices to Measure Basic Vital Signals of Admitted Patients», *Micromachines 2021, Vol. 12, Page 918*, vol. 12, n.º 8, p. 918, jul. 2021, doi: 10.3390/MI12080918.
- [11] D. Berwal, A. Kuruba, A. M. Shaikh, A. Udupa, y M. S. Baghini, «SpO2Measurement: Non-Idealities and Ways to Improve Estimation Accuracy in Wearable Pulse Oximeters», *IEEE Sens J*, vol. 22, n.º 12, pp. 11653-11664, jun. 2022, doi: 10.1109/JSEN.2022.3170069.
- [12] E. Kałamajska, J. Misiurewicz, y J. Weremczuk, «Wearable Pulse Oximeter for Swimming Pool Safety», *Sensors*, vol. 22, n.º 10, may 2022, doi: 10.3390/S22103823.
- [13] R. Saha *et al.*, «Internet of Things Framework for Oxygen Saturation Monitoring in COVID-19 Environment», *IEEE Internet Things J*, vol. 9, n.º 5, pp. 3631-3641, mar. 2022, doi: 10.1109/JIOT.2021.3098158.

Referencias bibliográficas

- [14] F. Luna-Perejón, L. Muñoz-Saavedra, J. M. Castellano-Domínguez, y M. Domínguez-Morales, «IoT garment for remote elderly care network», *Biomed Signal Process Control*, vol. 69, p. 102848, ago. 2021, doi: 10.1016/J.BSPC.2021.102848.
- [15] G. Arfaoui, X. Bultel, P.-A. Fouque, A. Nedelcu, y C. Onete, «The privacy of the TLS 1.3 protocol», *Proceedings on Privacy Enhancing Technologies*, vol. 2019, n.º 4, pp. 190-210, 2019, doi: 10.2478/popets-2019-0065.
- [16] A. Karimlou y M. Yavari, «A Low-Power Delta-Modulation-Based ADC for Wearable Electrocardiogram Sensors», *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, n.º 9, 2022, Accedido: nov. 01, 2022. [En línea]. Available: <https://ieeexplore.ieee.org.ehu.idm.oclc.org/document/9790719/>
- [17] A. Ahmed, M. M. Khan, P. Singh, R. S. Batth, y M. Masud, «IoT-based real-time patients vital physiological parameters monitoring system using smart wearable sensors», *Neural Comput Appl*, 2022, doi: 10.1007/S00521-022-07090-Y/FULLTEXT.HTML.
- [18] M. A. Akkaş, R. SOKULLU, y H. Ertürk Çetin, «Healthcare and patient monitoring using IoT», *Internet of Things (Netherlands)*, vol. 11, sep. 2020, doi: 10.1016/J.IOT.2020.100173.
- [19] T. Triwiyanto *et al.*, «Embedded Machine Learning Using a Multi-Thread Algorithm on a Raspberry Pi Platform to Improve Prosthetic Hand Performance», *Micromachines (Basel)*, vol. 13, n.º 2, feb. 2022, doi: 10.3390/MI13020191.
- [20] D. Ramegowda y M. Lin, «Energy efficient mixed task handling on real-time embedded systems using FreeRTOS», *Journal of Systems Architecture*, vol. 131, p. 102708, oct. 2022, doi: 10.1016/J.SYSARC.2022.102708.
- [21] R. Erna Wagner *et al.*, «Validation of a Low-Cost Electrocardiography (ECG) System for Psychophysiological Research», *Sensors 2021, Vol. 21, Page 4485*, vol. 21, n.º 13, p. 4485, jun. 2021, doi: 10.3390/S21134485.
- [22] N. Ahmed y Y. Zhu, «Early detection of atrial fibrillation based on ecg signals», *Bioengineering*, vol. 7, n.º 1, mar. 2020, doi: 10.3390/BIOENGINEERING7010016.
- [23] S. Kumar *et al.*, «A Low-Cost Multi-Sensor Data Acquisition System for Fault Detection in Fused Deposition Modelling», *Sensors*, vol. 22, n.º 2, ene. 2022, doi: 10.3390/S22020517.
- [24] G. Ehrmann, T. Blachowicz, S. V. Homburg, y A. Ehrmann, «Measuring Biosignals with Single Circuit Boards», *Bioengineering*, vol. 9, n.º 2. MDPI, feb. 01, 2022. doi: 10.3390/bioengineering9020084.
- [25] O. Panagopoulos y A. A. Argiriou, «Low-Cost Data Acquisition System for Solar Thermal Collectors», *Electronics (Switzerland)*, vol. 11, n.º 6, mar. 2022, doi: 10.3390/ELECTRONICS11060934.
- [26] M. J. A. Baig, M. T. Iqbal, M. Jamil, y J. Khan, «Design and implementation of an open-Source IoT and blockchain-based peer-to-peer energy trading platform using ESP32-S2, Node-Red and, MQTT protocol», *Energy Reports*, vol. 7, pp. 5733-5746, nov. 2021, doi: 10.1016/J.EGYR.2021.08.190.
- [27] C. Pintavirooj, B. Ni, C. Chatkobkool, y K. Pinijkij, «Noninvasive portable hemoglobin concentration monitoring system using optical sensor for anemia disease», *Healthcare (Switzerland)*, vol. 9, n.º 6, jun. 2021, doi: 10.3390/healthcare9060647.
- [28] S. R. Anan, M. A. Hossain, M. Z. Milky, M. M. Khan, M. Masud, y S. Aljahdali, «Research and Development of an IoT-Based Remote Asthma Patient Monitoring System», *J Healthc Eng*, vol. 2021, 2021, doi: 10.1155/2021/2192913.

Referencias bilbiográficas

- [29] W. Romine, T. Banerjee, y G. Goodman, «Toward sensor-based sleep monitoring with electrodermal activity measures», *Sensors (Switzerland)*, vol. 19, n.^o 6, mar. 2019, doi: 10.3390/S19061417.
- [30] S. Briginets, A. Volkov, G. Martinov, y A. Veselkov, «Development of a mobile heart monitor based on the ECG module AD8232», *AIP Conf Proc*, vol. 2015, sep. 2018, doi: 10.1063/1.5055087.
- [31] Aspiazu Ituarte. O, «Desarrollo de un sistema de apoyo a la diagnóstico e identificación de patologías implementado con dispositivos de bajo coste, preciso, fiable y de uso en entornos domiciliarios residenciales y hospitalarios.», 2022.