UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN DE DATOS BIOMÉTRICO PARA LA EVALUACIÓN DE PILOTOS DE COMPETICIÓN

TRABAJO FINAL DEL

Máster Universitario en Sensores para Aplicaciones Industriales

REALIZADO POR

Óscar Bori Fabra

TUTORIZADO POR

Francisco Javier Manjón Herrera
Juan Francisco Dols Ruiz
Javier Catalán Pechuán

CURSO ACADÉMICO: 2019/2020

Agradecimientos

“A mi familia, tutores, y todas las personas que me han apoyado”

Resumen

El presente trabajo plantea el diseño y la implementación de un sistema de adquisición de datos biométrico para la evaluación de pilotos de competición. Este sistema permitirá la monitorización en tiempo real de las señales biométricas del piloto. Con dichos datos se pretende interpretar, mediante la fusión de sensores obtenida, un estado global psicofísico del piloto durante los diferentes eventos que transcurren en la realización de las pruebas orientadas a las situaciones de conducción que sufre un piloto de competición en pista.

Este sistema se va a poner en práctica en el simulador G-UPV de conducción profesional para pilotos de competición, perteneciente al Instituto de Diseño y Fabricación (IDF), el cual cuenta con un cockpit monocasco apoyado en una plataforma dinámica de 6 GDL, con la que se pretende emular la aceleración que experimenta un piloto en una pista real, además de las condiciones del asfalto.

Con este sistema se pretende medir e interpretar una combinación de señales que se consideran potencialmente relevantes para la determinación del estado psicofísico del piloto, entre las cuales se incluyen señales de electrocardiograma, electromiograma, electrodermograma, ritmo respiratorio, temperatura corporal, saturación de oxígeno en sangre y aceleración que experimenta el piloto en los ejes xyz.

Palabras clave: sensores, competición, telemetría, biométrico, pilotos.

Resum

El present treball planteja el diseny i l'implementació d'un sistema de adquisició de dades biomètric per a l'avaluació de pilots de competició. Aquest sistema permet la monitorització en temps real de les senyals biomètriques del pilot. Amb eixes dades, es pretén interpretar mitzançant la fusió dels sensors obtesos, un estat global psicofísic del pilot durant els diferents esdeveniments que transcorren en la realització de les probes orientades a les situacions de conducció que pateix un pilot de competició en pista.

Aquest sistema es va a possar en pràctica en el simulador G-UPV de conducció profesional per a pilots de competició, que perteneix al Institut de Diseny i Fabricació (IDF), que compta amb un cockpit monocasc recolzat en una plataforma dinàmica de 6 GDL, amb la que es pretén emular la acceleració que experimenta un pilot en una pista real, a més, de les condicions del asfalt de la pista.

Amb aquest sistema es pretén medir e interpretar una combinació de senyals que es consideren potencialment relevants per a la determinació del estat psicofísic del pilot, entre les cuals s'inclouen senyals de electrocardiograma, electromiograma, electrodermograma, ritme respiratori, temperatura corporal, saturació d'oxigen en sang i l'acceleració que experimenta el pilot en els eixos xyz.

Paraules clau: sensors, competició, telemetría, biométric, pilots.

Abstract

The present work proposes the design and implementation of a biometric data acquisition system for the evaluation of competition pilots. This system will allow real-time monitoring of the pilot's biometric signals. With these data, the aim is to interpret, by means of the sensor fusion obtained, a global psychophysical state of the pilot during the different events that take place in carrying out tests aimed at the driving situations suffered by a competition pilot on the track.

This system will be implemented in the G-UPV professional driving simulator for competition pilots, belonging to the Institute of Design and Manufacturing (IDF), which has a monocoque cockpit supported on a dynamic 6 GDL platform, with which is intended to emulate the acceleration experienced by a pilot on a real track, in addition to the asphalt conditions.

This system aims to measure and interpret a combination of signals that are considered potentially relevant for the determination of the pilot's psychophysical state, including electrocardiogram, electromyogram, electrodermogram, respiratory rhythm, body temperature, blood oxygen saturation and the acceleration that pilot experiences on the xyz axes.

Keywords: sensors, competition, telemetry, biometric, driver.

Alumno: Óscar Bori Fabra

Valencia, noviembre de 2020

Tutor: Francisco Javier Manjón Herrera

Cotutor: Juan Francisco Dolz Ruiz

Cotutor: Javier Catalán Pechuán

Tabla de contenido

Documento I

Índice de figuras

Documento I

Índice de tablas

Documento I

Documento II

Índice de código

Documento I

Lista de abreviaturas

ECG Electroardiogram

EMG Electromiogram

EDA Electrodermal activity

PZT Piezoelectric

ACC Accelerometer

TcP Temperatura corporal del Piloto

ADC Analog to Digital Converter

PCB Printed Circuit Board

Documento I

Memoria

1. INTRODUCCIÓN

La realización de un Trabajo de Fin de Máster tiene como finalidad encontrar una solución a un problema determinado. Con dicha intención se ha realizado este proyecto para el Instituto de Diseño y Fabricación (IDF) de la Universitat Politècnica de València, cuyo objetivo es el estudio y diseño de un sistema de sensado aplicable a pilotos de competición que permita la monitorización de diferentes variables biométricas, para posteriormente en otros estudios, poder analizar los resultados procedentes de dicha fusión de sensores. Las pruebas este sistema se llevaron a cabo en el simulador G-UPV de conducción profesional para pilotos de competición.

1.1. OBJETIVO DEL TRABAJO FINAL DE MÁSTER

Teniendo en cuenta lo anteriormente mencionado, dado el potencial que ofrece una amplia fusión de sensores, los objetivos que se plantean en este TFM son:

- Determinar la viabilidad del sistema de adquisición de datos diseñado para la monitorización no invasiva de pilotos profesionales de competición.
- Identificar que sensores y/o componentes podrían ser reemplazados en un futuro, debido a problemas y/o insuficiencias que puedan surgir durante el desarrollo de este trabajo.

1.2. ANÁLISIS DEL ESTADO DEL ARTE

Desde que se concibió la idea de competir con vehículos como deporte, es sabido que es una disciplina que requiere de una preparación física y mental óptima, conforme evolucionan los vehículos tecnológicamente. Esta adaptación ha dado lugar a la progresiva profesionalización de este deporte, llegando hoy en día a la contratación de profesionales de la salud y preparadores físicos a tiempo completo por parte de los equipos, para la mejor preparación posible de los pilotos. En este capítulo se hablará de la tendencia de las vías de investigación abiertas en el campo de la monitorización del conductor/piloto para determinar su estado tanto fisiológico como mental (grado de concentración).

1.2.1. Punto de partida

Hay una amplia variedad de estudios que abarcan tanto mediciones invasivas como no invasivas, además de diferencias entre las variables que se desean medir entre estos. Es por ello que hay que determinar un punto de partida para poder progresar en esta investigación; ese punto de partida interesante puede ser el estudio realizado por Potkanowicz y Mendel (2013).

Dicho estudio habla en rasgos generales, acerca de la importancia de la monitorización en tiempo real del piloto para conocer su estado psicofísico, así como, de las facilidades que hay con la tecnología actual, de implementar sistemas miniaturizados integrados en la equipación del piloto. Se remarca la importancia de la planificación y ejecución efectiva del entrenamiento, tanto físico como mental, de los pilotos. Además, se puntualizan los parámetros principales de medición orientativos, para determinar el estado del piloto; concretamente son:

electrocardiograma (ECG), respuesta muscular o electromiograma (EMG), temperatura corporal, aceleración sufrida por el piloto (fuerzas G).

1.2.2. Ampliación del sensado

Hay otro estudio de David P. Ferguson y Nicholas D. Myers (2018), que complementa el anteriormente mencionado, implementando dos parámetros más comunes y uno muy especial para casos muy concretos de pilotos. Los dos parámetros comunes estudiados son: frecuencia de respiración (BR), temperatura de la piel (ST). El parámetro especial mide la concentración de glucosa en sangre, y está destinado especialmente en este estudio a ser analizado sobre un piloto diabético. Para este experimento, se planificó un banco de datos considerable en el cual se monitorizó a un piloto durante todas las vueltas en 47 carreras y durante 3 temporadas, de la IndyCar. Dicho estudio compara los resultados de rendimiento en pista con diferentes niveles de glucosa en sangre, además de que también se analiza la correlación, en este caso negativa, entre la resistencia a la fatiga y el esfuerzo muscular que exige este tipo de deporte.

El estudio evalúa dos puntos clave, el estado de preparación físico de un piloto de competición que padece diabetes de tipo 1, y la examinación de qué modo la glucosa en sangre puede influenciar los parámetros clave relacionados con el éxito en la competición.

1.2.3. Estrés mental

Para completar los dos estudios anteriores, el estudio realizado por Wen et al. (2017), va un paso más allá, analizando la reacción del piloto a estímulos externos para determinar el grado de estrés mental al que está sometido. Para dicho estudio, se implementó como novedad el parámetro de conductividad eléctrica de la piel (EDA) o también llamado en este estudio Galvanic Skin Response (GSR). En dicho estudio los investigadores han observado dos factores claros que afectan a diferentes parámetros, donde la aparición de la fatiga se ve reflejada en una mayor variación de los parámetros ECG y GSR. El parámetro EMG ha sido tomado del musculo Maxilofacial con el que han podido observar correlación con el factor estrés Mental. No obstante, en el estudio se aclara que dicha medición del musculo Maxilofacial puede dar falsas medidas debido a tics que pueda padecer el piloto o el mero hecho de salivar.

El procedimiento para la realización de las pruebas fue, durante la sesión de prácticas antes de la carrera, se realizaron seis vueltas consecutivas sin descanso, para posteriormente registrar la carrera real.

Este estudio concluye que existen diferentes elementos externos que generan estrés al piloto, y que, a su vez, estos generan respuestas fisiológicas dispares.

1.2.4. Una nueva vertiente

El hecho de que la medida del parámetro del músculo Maxilofacial puede verse mínimamente alterada, da lugar a posteriores estudios realizados, donde el grado de estrés mental sufrido o concentración por el piloto ha sido analizado mediante técnica de seguimiento de los ojos (*eye-tracking*), haciendo uso de videocámaras y empleando algoritmos que determinen el grado de concentración, comparando lo que se ve desde el vehículo y donde focaliza la visión el piloto;

además de complementar con el análisis del encefalograma (EEG) en tiempo real. Es el caso de los estudios realizados por Alzu'Bi et al. (2013), Clement et al. (2016) y Hu y Lodewijks (2020).

1.2.5. Enfoque en el software

También es muy interesante mencionar el estudio realizado por Sharma y Bundele (2020), donde teniendo un gran banco de datos de diversos parámetros (EEG, ECG, EMG, etc), han desarrollado un algoritmo basado en el modelo estadístico de K-Medias, con diferentes variantes para comparar rendimiento de cómputo, que les permite clasificar las diferentes señales fisiológicas de una forma más sencilla. Dichas señales se tomaron con una duración cada una de entre tres y cinco minutos de duración, donde participaron 150 conductores. Los datos se transferían a una memoria flash para posteriormente enviarla vía Bluetooth al ordenador portátil.

Después de los experimentos, el estudio concluye que los algoritmos propuestos funcionan muy bien en línea con los enfoques presentes con algunas características seleccionadas de señales fisiológicas como conductancia de la piel, pulso de oximetría y respiración.

1.2.6. La saturación de oxígeno puede ser clave

Como último parámetro destacado para el análisis del estado fisiológico del piloto, diversos estudios como el realizado por Jing et al. (2020), sugieren que la medición del ECG per se, no es suficientemente como indicador del estado del piloto, debido a que esta medición puede verse influenciada por múltiples factores, como puede ser la ingesta de cafeína, exposición al calor o la duración del ejercicio. Por ello, dichos estudios, recurren como indicador complementario al ECG, con la implementación de un oxímetro para poder cuantificar la concentración de oxígeno para determinar la relevancia de cada parámetro frente a los diferentes estímulos a los que se somete el piloto. Con este parámetro se pretende analizar que ocurre dentro del cuerpo cuando se realiza un esfuerzo físico.

Para la realización de las pruebas, se dispuso de la carretera nacional China G109, cuya altitud varía desde 3500 a 4700 m. Las pruebas se subdividieron en dos altitudes, 3500-4100 m y 4100-4700 m. En dichas pruebas participaron cinco conductores varones, de diferentes edades y experiencia en conducción.

El estudio demostró que la frecuencia cardíaca se puede utilizar como un indicador eficaz de la fatiga al conducir, mientras que la saturación de oxígeno de la sangre solo puede ser utilizada como indicador auxiliar. El estudio también confirmó sustancias, como Red Bull y el café, tienen la capacidad de aumentar la resistencia a la fatiga de conducción y retrasarla en entornos de hipoxia y baja energía.

1.2.7. Anotaciones finales del estado del arte

Para finalizar este capítulo, es interesante mencionar, que actualmente fabricantes como OMP (2019) y NTT (2020), están desarrollando diferentes sensores fisiológicos, véase EDA (GSR) junto a pulsioxímetro, y ECG respectivamente, los cuales están compuestos de fibras poliméricas con la característica de que tienen buena conductividad eléctrica, lo que permite captar las tenues diferencias de potencial en el caso del ECG con una gran sencillez en cuanto a cables y confort.

Lo mismo se puede aplicar al EDA en el guante del fabricante OMP junto con el sensor óptico por IR para la medición de concentración de oxígeno en sangre, donde un electrodo convencional reduciría el confort del piloto a la vez que reduciría la sencillez del montaje.

2. MATERIALES Y MÉTODOS

En el siguiente capítulo se procederá a describir los materiales empleados para la realización de este trabajo, así como el diseño de ejecución de las pruebas a realizar para validar el sistema de adquisición de datos. La primera mitad del capítulo hablará sobre los sensores empleados, su tecnología, y sus aplicaciones, además de la descripción del montaje del hardware empleado y el código desarrollado para la aplicación de Matlab y Arduino. La segunda mitad de este capítulo tratará sobre la metodología experimental del diseño y realización de las pruebas a realizar.

2.1. DESCRIPCIÓN DE LAS SEÑALES BIOMÉTRICAS

En el siguiente apartado se describen los sensores que han sido empleados para la realización de este trabajo, para que haya una lectura más clara se describirá cada sensor en un subapartado distinto.

2.1.1. Electrocardiograma (ECG)

El electrocardiograma estándar de 12 electrodos muestra la diferencia de potencial en distintos sitios prescritos de la superficie del cuerpo, los cuales varían durante el ciclo cardiaco. Este refleja las variaciones en el voltaje de las membranas en las células del miocardio, que transcurren durante la polarización y la despolarización en cada ciclo.

Detallando más, el trazo expresa la magnitud y el sentido del vector de despolarización de las señales que producen la excitación de las células del miocardio. Los vectores de polarización más representativos son conocidos como derivaciones. Coincidiendo con lo que se ha nombrado al principio, el número total de derivaciones es 12, siendo 3 bipolares y 9 unipolares. El impulso se obtiene colocando en la dirección de dichos vectores un conjunto de 10 electrodos. El análisis simultáneo de las 12 señales permite determinar con seguridad el estado del corazón del sujeto de análisis.

Cada señal se compone de cinco ondas, representadas en la imagen siguiente:

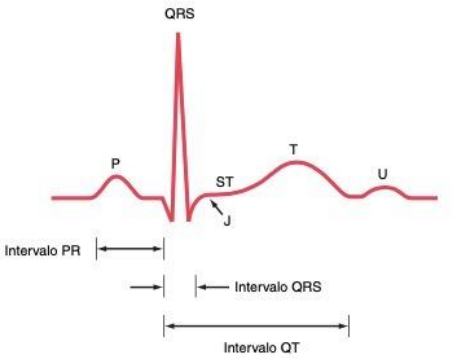


Figura 1. Onda P, complejo QRS, y complejo ST-T-U. Fuente: Harrison, 17 ed.I, pp. 1389.

La onda P representa la despolarización auricular, que precede a la contracción mecánica de las aurículas, permitiendo el paso de sangre a los ventrículos. El complejo QRS representa la despolarización ventricular, antecesora de la contracción ventricular, que permite el bombeo hacia el resto del cuerpo. Por último, el complejo ST-T-U representa la repolarización ventricular, indicando que el sistema está listo para un nuevo ciclo. Debido a que "la despolarización es una onda progresiva de cargas positivas" según Dubin (3 ed.), el acercamiento de una onda de estimulación a un electrodo positivo da lugar a una deflexión positiva en el electrocardiograma, y el alejamiento de estas da lugar a una deflexión negativa.

Fisiológicamente, "las corrientes eléctricas que viajan por el corazón se originan en tres elementos diferentes: las células cardiacas con función de marcapasos, el tejido especializado de la conducción y el propio miocardio." El ECG registra únicamente los potenciales de despolarización y repolarización, que corresponden a estímulo y recuperación de las células del miocardio, respectivamente.

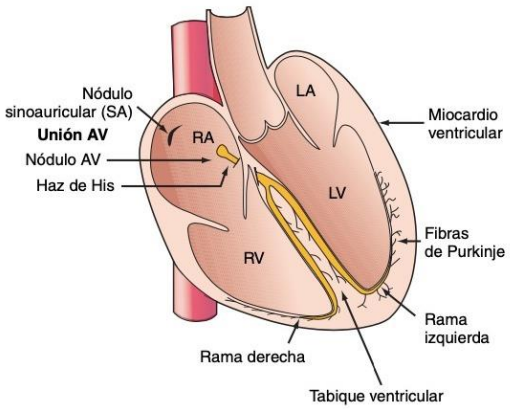


Figura 2. Esquema del sistema de conducción cardiaco. Fuente: Harrison, 17 ed., pp. 1388.

La figura superior muestra el sistema de conducción cardiaco. El latido cardiaco normal (ritmo sinusal) se caracteriza por que "*el estímulo para la despolarización […] se origina en el nódulo sinoauricular (SA)*" según Harrison (17 ed., pp.1388). La observación de la onda P seguidas de un complejo QRS en todas las derivaciones indica que existe ritmo sinusal. La ausencia de ritmo sinusal puede alertar acerca de alteraciones mecánicas o electrofisiológicas. También es importante comprobar la ritmicidad de los complejos QRS, es decir, que los intervalos que los separan sean de idéntica duración. Las arritmias se caracterizan por no cumplir la condición de ritmicidad constante. Sin embargo, con una derivación es posible medir la frecuencia cardiaca, el ritmo, y una estimación de si funciona correctamente el corazón (señales eléctricas indicativas de posibles síndromes coronarios agudos: infarto de miocardio, aneurismas aórticos). La derivación II suele ser la indicada para estos casos, pues el eje de la medición se corresponde con el eje anatómico del corazón.

El último parámetro a tener en cuenta en el ECG es la frecuencia cardiaca, que viene caracterizada por el número de latidos por minuto (en adelante LPM). La frecuencia cardiaca se considera normal cuando se sitúa entre 60 y 100 LPM. La frecuencia cardiaca máxima es función de la edad, siendo la ecuación de Fox y Haskell la más usada para su determinación ($FC_{máx}$ = 220 - edad). En base a esta fórmula, para una persona de 30 años, la $FC_{máx}$ teórica es de 190 LPM.

Puntualizar, que la descripción de esta señal ha sido tan extensa, debido a su comprensible complejidad y extensión del tema.

2.1.2. Electromiograma (EMG)

El electromiograma, llamado por sus siglas en inglés EMG, se basa en el mismo principio que el ECG; detectar las variaciones de potencial que se generan, en este caso al contraer determinado musculo que se quiera monitorizar. Para el caso concreto de este trabajo, el EMG se empleará para medir las contracciones del músculo maxilofacial, para interpretar el estado de estrés psicológico que sufre el piloto, como se menciona en el artículo Wen et al. (2017).

2.1.3. Electrodermograma (EDA)

El electrodermograma, también llamado por sus siglas en inglés, actividad electrodérmica (EDA), mide los cambios de conductividad producidos en la piel debido al aumento de la actividad de las glándulas sudoríparas. Los sitios preferidos para las medidas de EDA se encuentran en las palmas de las manos y plantas de los pies según el artículo Zangróniz et al. (2017). Aunque debido a que se tiene que buscar un equilibrio confort y practicidad para la aplicación en competición, según Wen et al. (2017), el lugar ideal donde hay abundancia de glándulas sudoríparas para la colocación de los electrodos, sin comprometer la medición, es justo detrás de las orejas a la altura del lóbulo. Las glándulas ecrinas segregan sudor por estímulos externos y procesos endógenos (memoria, atención, vigilancia, compromiso motor, etc.), llenando los poros de la piel y aumentando la conductividad.

2.1.4. Respiración (PZT)

La medición del ritmo respiratorio se emplea como un indicador fisiológico, que junto al ECG y al SpO2, que se nombrará más adelante, ayuda a interpretar mejor el análisis del estado del piloto. Estos son indicadores que se retroalimentan, aunque si hay que puntualizar que el predominante y más sensible de estos tres es el ECG. La señal que se obtiene es definida por la variación de la resistencia que experimenta el sensor piezoeléctrico, dando lugar a un potencial de salida de señal variable conforme se deforma.

2.1.5. Saturación de oxígeno en sangre (SpO2)

Como se ha mencionado anteriormente, la saturación de oxígeno en sangre es un indicador valioso para ayudar a interpretar la señal ECG con mayor precisión. No obstante, solo debe tratarse, como un indicador auxiliar al ECG y nunca interpretarse por separado, como se indica en el artículo de Jing et al. (2020), al menos en condiciones de hipoxia, es decir, de gran altitud. El valor de saturación de oxígeno se mide en porcentaje, cuyos valores de saturación normales oscilan entre 92-95 % y 100 %, dependiendo del error del sensor y su precisión.

2.1.6. Temperatura corporal (TcP)

La temperatura corporal del piloto es una medición que hay que considerar, debido a la alta demanda física que sufren en intervalos dinámicos comparable al estudio de Seo et al. (2019); sumándole a esto las diferentes prendas de protección que visten lo cual puede considerar una

aumento de la temperatura corporal, puede que ese incremento se vea reflejado en la fatiga del piloto. No obstante, la fusión de este sensor con los demás, como pueda ser EDA, hace que la interpretación de estos, puede que no indique el grado de fatiga que sufre el piloto; de modo que es importante que un análisis futuro de los resultados de la fusión de todos los sensores determine el peso determinado en cada combinación posible.

2.1.7. Acelerómetro (ACC)

El acelerómetro es indispensable en cualquier proyecto de monitorización en competición. Esto se debe, a que el acelerómetro muestra las diferentes aceleraciones que esta experimentado el piloto en el vehículo, lo que se traduce indirectamente, en cierta medida el estrés físico al que está siendo expuesto. Es un parámetro que puede arrojar muchos detalles sobre los diferentes eventos que experimenta el piloto en pista, como puede ser una apurada de frenada antes de entrar en curva posteriormente de realizar un adelantamiento, donde puede que se vea una relación entre el aumento de ECG, la respiración con la fuerza de deceleración experimentada, e incluso un aumento de la sudoración al aumentar la tensión en el adelantamiento.

2.2. ADQUISICIÓN Y ANÁLISIS DE DATOS BIOMÉTRICOS (SOFTWARE)

En este apartado se procederá a explicar de la forma más resumida posible, pero que se pueda comprender todo el funcionamiento del código escrito en Arduino y Matlab. A modo de introducción, el concepto general de funcionamiento se basa, en que mediante la interfaz creada en Matlab se envía la orden de toma de datos a Arduino con una periodicidad establecida, para posteriormente con diferentes funciones en Matlab, exportar los datos del modo que convenga. En dicha interfaz también se muestran los datos mientras se están recibiendo por parte de Matlab.

2.2.1. Desarrollo Arduino

La primera parte del código Arduino, consiste en la inclusión de librerías y la definición de las variables que se van a usar.

```
#include <DueTimer.h>
#include <Wire.h>
#include "MAX30105.h"
#include "spo2_algorithm.h"

// declaracion de variables
const size_t long_buff = 200;
const size_t long_vect = 4;
byte RecData;
uint8_t t_pilVect[long_vect]; // vector bytes temp piloto (LM35)
volatile float t_pil;
byte valSPO2;
int cont = 0;
int i;
volatile int flgTx = 0;
volatile int flgBuff = 1;
volatile uint16_t buffer1ECG[long_buff];
volatile uint16_t buffer2ECG[long_buff];
volatile uint16_t buffer1EMG[long_buff];
volatile uint16_t buffer2EMG[long_buff];
volatile uint16_t buffer1EDA[long_buff];
volatile uint16_t buffer2EDA[long_buff];
volatile uint16_t buffer1PZT[long_buff];
volatile uint16_t buffer2PZT[long_buff];
volatile uint16_t buffer1ACCx[long_buff];
volatile uint16_t buffer2ACCx[long_buff];
volatile uint16_t buffer1ACCy[long_buff];
volatile uint16_t buffer2ACCy[long_buff];
volatile uint16_t buffer1ACCz[long_buff];
volatile uint16_t buffer2ACCz[long_buff];
volatile uint16_t bff1calACCx[long_buff];
volatile uint16_t bff1calACCy[long_buff];
volatile uint16_t bff1calACCz[long_buff];
volatile uint16_t bff2calACCx[long_buff];
volatile uint16_t bff2calACCy[long_buff];
volatile uint16_t bff2calACCz[long_buff];
int ACCcont = 0;
int j;
byte ACCRecData;
volatile int ACCflgTx = 0;
volatile int ACCflgBuff = 1;
```

Código Arduino 1. Librerías y definición de variables.

```
//Definicion MAX30102-------------------------------------------------------
MAX30105 particleSensor;

#define MAX_BRIGHTNESS 255

#if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168__)
//Arduino Uno doesn't have enough SRAM to store 100 samples of IR led data and red led data in 32-bit format
//To solve this problem, 16-bit MSB of the sampled data will be truncated. Samples become 16-bit data.
uint16_t irBuffer[100]; //infrared LED sensor data
uint16_t redBuffer[100];  //red LED sensor data
#else
uint32_t irBuffer[100]; //infrared LED sensor data
uint32_t redBuffer[100];  //red LED sensor data
#endif

int32_t bufferLength = 60; //data length
int32_t spo2; //SPO2 value

int8_t validSPO2; //indicator to show if the SPO2 calculation is valid
int32_t heartRate; //heart rate value
int8_t validHeartRate; //indicator to show if the heart rate calculation is valid
//Fin definicion MAX30102--------------------------------------------------
```

Código Arduino 2. Definición variables sensor SpO2.

Posteriormente se sitúa la parte de la configuración de Arduino, conocida como una función *Setup*. Donde se crean dos interrupciones distintas que se vinculas a dos funciones, *timerfcn* que es la función de toma de datos principal, y *calACC*, que es la función que solo toma datos del acelerómetro para la calibración.

```
void setup() {
  Timer3.attachInterrupt(timerfcn);
  Timer4.attachInterrupt(calACC);
  //Serial.begin(115200);
  Serial1.begin(115200); //Serial que usa el modulo Bluetooth
  Serial2.begin(115200); //Serial C232HD
  // configuracion sensor SPO2-------------------------------------------------------------------------
  particleSensor.begin(Wire, I2C_SPEED_FAST); //Use default I2C port, 400kHz speed
  byte ledBrightness = 60; //Options: 0=Off to 255=50mA
  byte sampleAverage = 1; //Options: 1, 2, 4, 8, 16, 32
  byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green
  byte sampleRate = 1000; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200
  int pulseWidth = 69; //Options: 69, 118, 215, 411
  int adcRange = 2048; //Options: 2048, 4096, 8192, 16384

  particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth, adcRange);
  //Configure sensor with these settings

  //-------------------------------------------------------------------------------------------------
}
```

Código Arduino 3. Setup.

La función *timerfcn* se divide en dos partes que son idénticas, cuya única diferencia es que se usan *buffers* o vectores distintos, para almacenar los datos de forma continuada y que no haya perdida de datos. Por lo que solo se muestra la toma de datos para el *buffer1*, aunque el *buffer2* es exactamente igual, la única diferencia es que las banderas o *flags* usadas cambian para marcar en que *buffer* tienen que almacenarse los datos cada vez que termina de llenarse uno de ellos.

```
88 // funcion toma de datos principal
89 void timerfcn() {
90
91   if (flgBuff == 1) {
92     analogRead(A0); //se lee primero para que no haya fallos en la lectura
93     buffer1ECG[cont] = analogRead(A0); //lectura pin A0 analogico
94     analogRead(A1);
95     buffer1EMG[cont] = analogRead(A1);
96     analogRead(A2);
97     buffer1EDA[cont] = analogRead(A2);
98     analogRead(A3);
99     buffer1PZT[cont] = analogRead(A3);
00     analogRead(A5);
01     buffer1ACCx[cont] = analogRead(A5);
02     analogRead(A6);
03     buffer1ACCy[cont] = analogRead(A6);
04     analogRead(A7);
05     buffer1ACCz[cont] = analogRead(A7);
06     cont++;
07     if (cont == 200) {
08       //leer el resto de sensores
09       analogRead(A4);
10       int lm_sensor = analogRead(A4); //lectura sensor lm35
11       t_pil = lm_sensor * 330 / 1023.0; // calculo temperatura piloto
12
13       cont = 0;
14       flgTx = 1;
15       flgBuff = 2;
16     }
17   } else {
```

Código Arduino 4. Función timerfcn.

Acto seguido se crea la función de toma de datos para el calibrado del acelerómetro mediante Matlab, con el mismo criterio que la función *timerfcn*.

```
funcion toma de datos para calibrar ACC
void calACC() {

  if (ACCflgBuff == 1) {
    analogRead(A5);
    bfflcalACCx[ACCcont] = analogRead(A5);
    analogRead(A6);
    bfflcalACCy[ACCcont] = analogRead(A6);
    analogRead(A7);
    bfflcalACCz[ACCcont] = analogRead(A7);
    ACCcont++;
    if (ACCcont == 200) {
      ACCcont = 0;
      ACCflgTx = 1;
      ACCflgBuff = 2;

    }
  } else {
    analogRead(A5);
    bff2calACCx[ACCcont] = analogRead(A5);
    analogRead(A6);
    bff2calACCy[ACCcont] = analogRead(A6);
    analogRead(A7);
    bff2calACCz[ACCcont] = analogRead(A7);
    ACCcont++;
    if (ACCcont == 200) {
      ACCcont = 0;
      ACCflgTx = 2;
      ACCflgBuff = 1;

    }
  }
}
```

Código Arduino 5. Función para el calibrado de ACC.

Por último, se pasa al bloque principal, donde se ejecutará el programa mediante las ordenes que lleguen de Matlab y donde se enviarán los datos a este a través de Bluetooth. Hay un bloque para los datos principales, y luego se repite el mismo procedimiento para la calibración del acelerómetro. Por lo tanto, solo será necesario mostrar en este apartado el bloque de los datos principales. Para consultar todo el código de Arduino y Matlab, dirigirse al anexo.

```
void loop() {
  if (Serial1.available() > 0) {

    // Comprobamos si ha llegado algún dato

    RecData = Serial1.read(); // Se lee el dato recibido

    if (RecData == 170) { // Si el dato recibido es 170 se realiza la medida

      Timer3.start(5000); // Interrupcion cada 5ms
    }

    if (RecData == 100) {

      Timer3.stop();
    }
  }
```

Código Arduino 6. Comprobación de ordenes desde Matlab y arranque de la toma de datos.

Una vez se recibe la orden correcta desde Matlab para arrancar la toma de datos, cuando la función *timerfcn* termina con todas las lecturas de los datos de un ciclo, se procede en la siguiente parte a la preparación y el envío de los datos en forma de bytes, a través del puerto serie conectado al módulo Bluetooth. Siguiendo las indicaciones de las banderas o *flags*, para saber de qué *buffer* hay que enviar los datos.

```
200   //envio de datos a traves del puerto serie bluetooth
201   if (flgTx) {
202
203     if ( flgTx == 1) {
204
205       for (i = 0; i < 200; i++) {
206         Serial1.write(buffer1ECG[i] / 256);
207         Serial1.write(buffer1ECG[i] % 256);
208       }
209       for (i = 0; i < 200; i++) {
210         Serial1.write(buffer1EMG[i] / 256);
211         Serial1.write(buffer1EMG[i] % 256);
212       }
213       for (i = 0; i < 200; i++) {
214         Serial1.write(buffer1EDA[i] / 256);
215         Serial1.write(buffer1EDA[i] % 256);
216       }
217       for (i = 0; i < 200; i++) {
218         Serial1.write(buffer1PZT[i] / 256);
219         Serial1.write(buffer1PZT[i] % 256);
220       }
221       for (i = 0; i < 200; i++) {
222         Serial1.write(buffer1ACCx[i] / 256);
223         Serial1.write(buffer1ACCx[i] % 256);
224       }
225       for (i = 0; i < 200; i++) {
226         Serial1.write(buffer1ACCy[i] / 256);
227         Serial1.write(buffer1ACCy[i] % 256);
228       }
229       for (i = 0; i < 200; i++) {
230         Serial1.write(buffer1ACCz[i] / 256);
231         Serial1.write(buffer1ACCz[i] % 256);
232       }
```

Código Arduino 7. Envío de datos de uno de los Buffers, truncado cada dato de 2 bytes en parte alta y parte baja, es decir, una parte es el resultado de la división del número de valores dentro de un byte, y la otra el resto de esa división.

A continuación, la toma de datos del resto de variables.

```
234       //Toma de datos y calculo de SPO2-----------------------------------------------------------
235       for (byte i = 0 ; i < bufferLength ; i++)
236       {
237         while (particleSensor.available() == false) //do we have new data?
238           particleSensor.check(); //Check the sensor for new data
239
240         redBuffer[i] = particleSensor.getRed();
241         irBuffer[i] = particleSensor.getIR();
242         particleSensor.nextSample(); //We're finished with this sample so move to next sample
243       }
244
245       //calculate heart rate and SpO2
246       maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validSPO2,
247
248       if (validSPO2 == 1) {
249         valSPO2 = spo2;
250       } else {
251         valSPO2 = 0;
252       }
253       //Final toma de datos de SPO2---------------------------------------------------------------
254
255       // todas las demas variables
256       byte* t_pilVect = (byte*)&t_pil;
257       Serial1.write(t_pilVect, sizeof(t_pilVect));
258       Serial1.write(valSPO2);
259     }
```

Código Arduino 8. Toma de datos de SpO2 fuera de la interrupción del timer (debido a incompatibilidad de la librería), y el envío de la temperatura.

2.2.2. Desarrollo Matlab

Dado que el código de Matlab ocupa aproximadamente 650 líneas de código, se considera que en este apartado se expliquen únicamente las partes más complicadas y/o relevantes que ayuden a interpretar el código en su conjunto, como la función de lectura de datos que es la más relevante. Como ya se ha mencionado anteriormente, para ver el código completo, consultar el anexo.

Empezando con la interfaz gráfica de usuario de diseño propio mediante Matlab, en la figura que se muestra a continuación, se muestra la interfaz que se ha empleado para el desarrollo de este trabajo. La interfaz muestra la toma de datos en vivo de todos los sensores muestreados, siendo las señales ECG, EMG, EDA, PZT y ACC visibles en formato de gráfica. Para la temperatura corporal del piloto y la saturación de oxigeno en sangre, los valores se muestran de forma numérica en la esquina superior derecha de la interfaz. Esta interfaz cuenta con dos campos rellenables, en los que se introducen el nombre del archivo para exportar los datos, sin la extensión del archivo, y la ruta de guardado del archivo. Esta interfaz, cuenta con seis botones, entre los que se diferencian funciones como, iniciar y para la adquisición de los datos, exportar los datos en formato Excel, salir de la aplicación, calibrar el acelerómetro y reiniciar los datos de calibración del acelerómetro, además de la limpieza de datos en memoria, así como de las gráficas que se muestran.

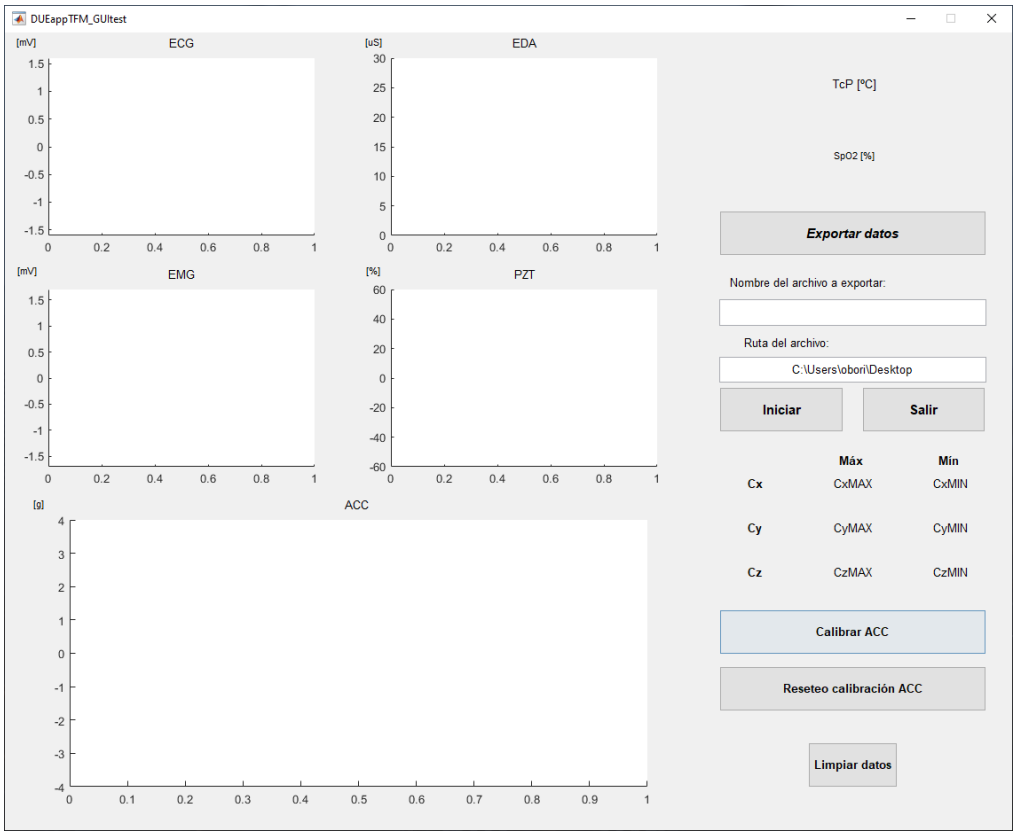


Figura 3. Interfaz gráfica de usuario en Matlab.

El procedimiento normal de ejecución de la aplicación es, una vez conectada la placa Arduino mediante Bluetooth al ordenador, se arranca la aplicación. Con el cable FTDI C232HD conectado a la placa, se pulsa el botón de calibrar ACC, y se gira el sensor ACC en los tres ejes lentamente unos 360 grados de giro. En la parte de arriba se muestran los datos del calibrado en vivo. El botón habrá cambiado de color y el texto indicará parar la calibración; se pulsa cuando se haya calibrado y esos datos son almacenados hasta que se cierre la aplicación. Si se pretende reiniciar la calibración con datos teóricos, se puede pulsar el botón Reseteo calibración ACC, que establecerá 1 y -1 g por defecto, como máximo y mínimo en cada eje.

Acto seguido se desconecta de Arduino el cable FTDI C232HD, y se pulsa el botón Iniciar, para la toma de mediciones, este botón cambiará de color y el texto indicará parar. Una vez se concluye la toma de datos, se pulsa el mismo botón anterior, se escribe el nombre del archivo para exportar en Excel, se establece una ruta para este, y se pulsa el botón Exportar datos. Cuando estos hayan sido exportados, emergerá una ventana avisando que ha concluido la exportación, se cierra la ventana emergente. Si se desea repetir nuevas mediciones, se pulsa el botón, Limpiar datos, el cual conserva la calibración del acelerómetro, y se mide nuevamente. Si se desea salir, se pulsa el botón Salir, el cual pedirá confirmación mediante una ventana emergente.

Describiendo la función principal de toma de datos, primero se definen las diferentes variables globales, para poder ser usadas en otras funciones de la aplicación, así como las locales. También se asignan a diferentes conjuntos de datos los bytes que se reciben.

```
206    function Rx_Callback(hObject, eventdata, handles, varargin)
207 -  global SerCOM x j Cont ECG ECGH ECGL ECGval EMG EMGH EMGL EMGval EDA EDAH EDAL EDAval...
208        PZT PZTH PZTL PZTval ACCx ACCxH ACCxL ACCxval ACCy ACCyH ACCyL ACCyval...
209        ACCz ACCzH ACCzL ACCzval TP yTP spo2Val ySPO2 plotECG...
210        Cxmin Cxmax Cymin Cymax Czmin Czmax plotEMG plotEDA plotPZT plotACC...
211        xECG xEMG xEDA xPZT xACCx xACCy xACCz% Declaracion variables globales
212 -  a = 1;
213
214    % SerCOM.BytesAvailable
215 -  Datos=fread(SerCOM,2805); % Se leen los 2805 bytes recibidos
216
217 -  DatosECG = uint16(Datos(1:400)); %convertimos a uint16 para hacer el cast
218 -  DatosEMG = uint16(Datos(401:800));
219 -  DatosEDA = uint16(Datos(801:1200));
220 -  DatosPZT = uint16(Datos(1201:1600));
221
222 -  DatosACCx = uint16(Datos(1601:2000));
223 -  DatosACCy = uint16(Datos(2001:2400));
224 -  DatosACCz = uint16(Datos(2401:2800));
225
226 -  DatosTP = uint8(Datos(2801:2804));
227 -  DatosSPO2 = uint8(Datos(2805));
```

Código Matlab 1. Variables y asignación de bytes.

Acto seguido, de cada conjunto de bytes designado, se vuelven a juntar las partes altas y partes bajas, mencionadas en el código Arduino, para conformar el valor que corresponde a 2 bytes de información. Cuando es conformado el valor recibido, se aplica la función de transformación de cada sensor, para obtener el valor que se pretende mostrar e interpretar, ya sea mV, uS, g de fuerza, etc.

```
229 -  for i=1:200
230 -      ECGH=DatosECG(2*i-1);
231 -      ECGL=DatosECG(2*i);
232 -      ECG(j+i)=256*ECGH+ECGL;
233 -      ECGval(j+i)=((((ECG(j+i)/2^10)-0.5)*3.3)/1100)*1000;
234 -      xECG(j+i)=length(ECGval);
235 -  end
236 -  for i=1:200
237 -      EMGH=DatosEMG(2*i-1);
238 -      EMGL=DatosEMG(2*i);
239 -      EMG(j+i)=256*EMGH+EMGL;
240 -      EMGval(j+i)=((((EMG(j+i)/2^10)-0.5)*3.3)/1009)*1000;
241 -      xEMG(j+i)=length(EMGval);
242 -  end
243 -  for i=1:200
244 -      EDAH=DatosEDA(2*i-1);
245 -      EDAL=DatosEDA(2*i);
246 -      EDA(j+i)=256*EDAH+EDAL;
247 -      EDAval(j+i)=1/(1-(EDA(j+i)/2^10));
248 -      xEDA(j+i)=length(EDAval);
249 -  end
250 -  for i=1:200
251 -      PZTH=DatosPZT(2*i-1);
252 -      PZTL=DatosPZT(2*i);
253 -      PZT(j+i)=256*PZTH+PZTL;
254 -      PZTval(j+i)=((PZT(j+i)/2^10)-0.5)*100;
255 -      xPZT(j+i)=length(PZTval);
256 -  end
```

Código Matlab 2. Conformado de los valores a partir de los bytes recibidos.

```
for i=1:200
    ACCxH=DatosACCx(2*i-1);
    ACCxL=DatosACCx(2*i);
    ACCx(j+i)=256*ACCxH+ACCxL;
    ACCxval(j+i)=(((ACCx(j+i)-Cxmin)/(Cxmax-Cxmin))*2)-1;
    xACCx(j+i)=length(ACCxval);
end
for i=1:200
    ACCyH=DatosACCy(2*i-1);
    ACCyL=DatosACCy(2*i);
    ACCy(j+i)=256*ACCyH+ACCyL;
    ACCyval(j+i)=(((ACCy(j+i)-Cymin)/(Cymax-Cymin))*2)-1;
    xACCy(j+i)=length(ACCyval);
end
for i=1:200
    ACCzH=DatosACCz(2*i-1);
    ACCzL=DatosACCz(2*i);
    ACCz(j+i)=256*ACCzH+ACCzL;
    ACCzval(j+i)=(((ACCz(j+i)-Czmin)/(Czmax-Czmin))*2)-1;
    xACCz(j+i)=length(ACCzval);
end
```

Código Matlab 3. Conformado de los bytes correspondientes al acelerómetro.

El siguiente paso, consta en aplicar el filtrado de señal de media móvil a las señales, para eliminar parte del ruido que hay en cada señal, mediante la función *filter* de Matlab.

```
%Filtrado de señales con media movil
windowSize1 = 5; %5 por defecto funciona bien
b1 = (1/windowSize1)*ones(1,windowSize1);
ECGval = filter(b1,a,ECGval);

windowSize2 = 5;
b2 = (1/windowSize2)*ones(1,windowSize2);
EMGval = filter(b2,a,EMGval);

windowSize3 = 5;
b3 = (1/windowSize3)*ones(1,windowSize3);
EDAval = filter(b3,a,EDAval);

windowSize4 = 5;
b4 = (1/windowSize4)*ones(1,windowSize4);
PZTval = filter(b4,a,PZTval);

windowSize5 = 5;
b5 = (1/windowSize5)*ones(1,windowSize5);
ACCxval = filter(b5,a,ACCxval);
ACCyval = filter(b5,a,ACCyval);
ACCzval = filter(b5,a,ACCzval);
```

Código Matlab 4. Filtrado de señales con media móvil.

Dicho filtro de media móvil se analiza su aplicación en Matlab, en el artículo (Report & Gonz, 2015).

Dicho filtro ha sido aplicado en este trabajo a modo de facilitar la visualización en vivo de los datos en la interfaz de usuario. Este filtro de media móvil se describe en la siguiente ecuación:

$$y[n] = \frac{1}{N}\sum_{i=0}^{N-1} x[n-i]$$

Donde N es el tamaño de la ventana de la media, el cual se estableció en 5 dado sus buenos resultados en este trabajo concretamente. Y[n] es el valor de salida, x[n] es el valor de entrada, x[n-1] es el valor de entrada previo.

Por último, de los bytes asignados, se conforma el valor de la temperatura del piloto y la saturación de oxígeno en sangre (SpO2).

```
304 -    j=j+200;
305
306 -    TP = typecast(DatosTP(1:4), 'single'); %temperatura piloto
307 -    spo2Val = DatosSPO2(1); %valor SpO2
308
309 -    x(Cont)=Cont;
310 -    yTP(Cont)=TP;
311 -    ySPO2(Cont)=spo2Val;
```

Código Matlab 5. Valores TcP y SpO2.

Detallar que, para la conformación del valor de la temperatura del piloto, como este dato consta de 4 bytes, no se puede hacer lo mismo que lo realizado en los otros datos, haciendo parte alta y parte baja, esto solo era para el caso de datos en 2 bytes. Para este caso en concreto se empleó la función *typecast* de Matlab para conformar el valor TcP.

La siguiente parte de la función principal, corresponde a la muestra de datos en la interfaz, lo cual no es relevante para la comprensión de este trabajo. Dicha parte podrá ser consultada en el anexo.

2.3. DESCRIPCIÓN E IMPLEMENTACIÓN DE LOS SENSORES (HARDWARE)

2.3.1. Diagrama de bloques

A continuación, se muestra el diagrama de bloques respecto al montaje general del sistema de adquisición de datos, donde se especifica el tipo de conexiones que se emplean y a que conectores están designados físicamente en la placa Arduino DUE.

DIAGRAMA DE BLOQUES

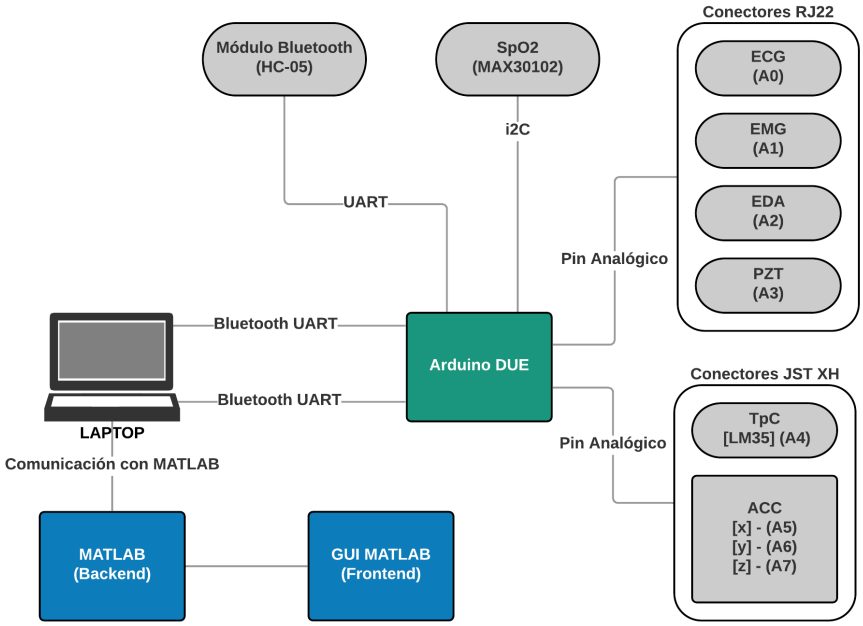


Figura 4. Diagrama de bloques. Fuente: diseño propio.

Para mayor detalle del montaje, consultar el esquema eléctrico del Anexo.

2.3.2. Comunicación

En este subapartado, se exponen brevemente los distintos protocolos de comunicación empleados en el trabajo.

- i2C: es un protocolo de comunicación síncrona, utilizado para transmitir los datos desde el sensor MAX30102, del cual se obtiene el parámetro de saturación de oxígeno en sangre tanto por cien. La configuración de velocidad de transmisión del protocolo i2C se ha modificado hasta alcanzar los 400KHz, mediante la librería del sensor en el IDE de Arduino.
- Bluetooth-UART: Se ha empleado un dispositivo inalámbrico Bluetooth para la transmisión de datos entre la placa Arduino DUE y el ordenador, siguiendo el protocolo de comunicación asíncrona UART. El modelo empleado es el HC-05, el cual puede actuar como esclavo y maestro. La velocidad de transmisión se ha fijado en 115200 baudios, debido a la gran cantidad de datos que se envían con la alta frecuencia de muestreo de los datos.

Con los sensores ECG, EMG, EDA, PZT y ACC, se obtienen valores que oscilan desde 0 hasta 1023, debido a que la resolución del conversor analógico/digital (ADC) es de 10 bits, lo que se traduce en 2^{10} valores digitales, es decir en 1024 valores. Por lo que los bytes necesarios para enviar un valor de cada uno de estos sensores serán de 2 bytes. Para la TcP (LM35), el valor que se obtiene al ser de coma flotante, en Arduino DUE concretamente ocupa 4 bytes. Por último, el sensor MAX30102 (SpO2) al ser conexión digital entrega en Arduino un valor entre 0 y 100, por lo que para enviar este dato solo será necesario de 1 byte de información.

Se pretende hacer un muestreo de 1000 Hz de cada sensor, Arduino DUE tiene una frecuencia de muestreo por defecto en el conversor ADC de aproximadamente 27 KHz, por lo que cumple con creces. Cada sensor, aproximadamente, es muestreado mediante una interrupción del microcontrolador cada 5 milisegundos para completar 200 valores en 1 segundo, considerando que por cada sensor paralelamente se guarda un vector de 200 valores antes de ser enviado al ordenador. Esto se aplica a los sensores ECG, EMG, EDA, PZT y ACC. Al final de cada interrupción se tomará una muestra de la TcP y de la SpO2, es decir, cada 1 segundo. Recordando que los vectores de 200 valores, cada valor es de 2 bytes, la trama final de datos queda del siguiente modo:

400 bytes ECG	400 bytes EMG	400 bytes EDA	400 bytes PZT	4 bytes TcP	400 bytes ACC (x)	400 bytes ACC (y)	400 bytes ACC (z)	1 byte SpO2

Tabla 1. Trama de datos que se envía desde Arduino. Orden: de izquierda a derecha.

2.3.3. Componentes

2.3.3.1. Arduino DUE

Arduino DUE es una placa de microcontrolador basada en la CPU Atmel SAM3X8E ARM Cortex-M3 de 32 bits. Tiene 54 pines de entrada / salida digitales (de los cuales 12 se pueden usar como salidas PWM), 12 entradas analógicas, 4 UART (puertos serie de hardware), un reloj de 84 MHz, una conexión USB OTG compatible, 2 DAC (digital a analógico), 2 TWI (i2C), un conector de alimentación (*power Jack*), un encabezado SPI, un encabezado JTAG, un botón de reinicio y un botón de borrado. Mediante el *power Jack*, se ha alimentado el sistema con una batería de tipo LiPo de 11.1 Voltios y 1500 mAh.

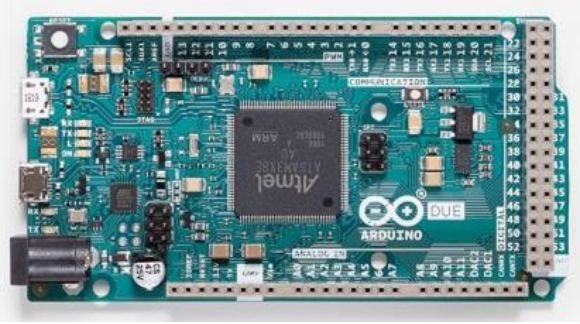


Figura 5. Placa Arduino DUE. Fuente: https://store.arduino.cc/arduino-due

Sus características principales se muestran en la siguiente tabla:

Núcleo	AT91SAM3X8E 32 bits
Frecuencia de trabajo	84 MHz
Vin (V)	7 - 12
Vlogic (V)	3.3
A I/O	12
D I/O	54

UART	4
I2C	2
SPI	2
Flash	512 kB
SRAM	96 kB

Tabla 2. Características principales de la placa Arduino DUE.

2.3.3.2. Modulo bluetooth HC-05

El módulo HC-05 se ha empleado para establecer la conexión de datos entre la placa Arduino DUE y el ordenador, a través del protocolo de comunicación serie UART. La velocidad de transmisión de datos se establece en 115200 baudios.

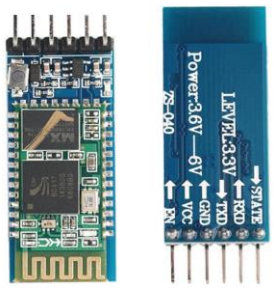


Figura 6. Módulo Bluetooth HC-05.

Sus características son las siguientes:

Rango (m)	< 100
Vin (V)	5
Vout (V)	5
Tipo	Maestro/esclavo
Vtransmisión (baudios)	hasta 460800

Tabla 3. Características del módulo HC-05. Fuente: https://components101.com/wireless/hc-05-bluetooth-module

2.3.3.3. Módulo ECG

Para la medición del electrocardiograma se ha empleado el módulo ECG de la marca Bitalino.

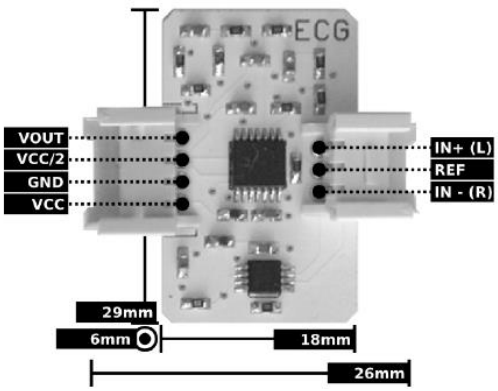


Figura 7. Módulo ECG de Bitalino.

Las características principales de este módulo se muestran en la siguiente tabla:

Ganancia	1100
Rango [mV]	± 1.5 (con VCC = 3.3V)
Ancho de banda [Hz]	0.5 - 40
Consumo [mA]	4
Impedancia de entrada [GOhm]	100
Electrodos	3 ó 2 (REF virtual)

Tabla 4. Características del módulo ECG de Bitalino. Fuente: (BITalino, 2016b).

La función de transferencia empleada para la conversión de los valores digitales adquiridos a través del ADC, para mostrarlos en valores que oscilarán entre -1.5 y 1.5 mV, es la siguiente:

$$ECG(V) = \frac{\left(\frac{ADC}{2^n} - \frac{1}{2}\right) \times VCC}{G_{ECG}}$$

$$ECG(mV) = ECG(V) \times 1000$$

Donde:

VCC = 3.3V (voltaje de operación)

G_{ECG} = 1100 (ganancia del sensor)

ADC → Valor muestreado del canal

n → Número de bits de resolución del ADC del canal (por defecto 10 bits)

2.3.3.4. Módulo EMG

Para la medición del electromiograma se ha empleado el módulo EMG de la marca Bitalino.

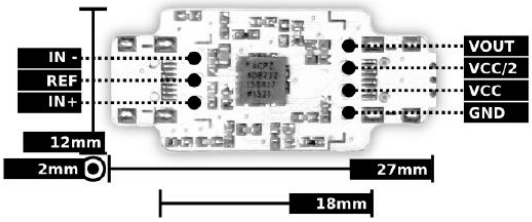


Figura 8. Módulo EMG de Bitalino.

Las características principales de este módulo se muestran en la siguiente tabla:

Ganancia	1009
Rango [mV]	± 1.64 (con VCC = 3.3V)
Ancho de banda [Hz]	25 - 480
Consumo [mA]	0.17
Impedancia de entrada [GOhm]	7.5
Voltaje de entrada [V]	2.0 – 3.5

Tabla 5. Características del módulo EMG de Bitalino. Fuente: (BITalino, 2015b).

La función de transferencia empleada para la conversión de los valores digitales adquiridos a través del ADC, para mostrarlos en valores que oscilarán entre -1.64 y 1.64 mV, es la siguiente:

$$EMG(V) = \frac{\left(\frac{ADC}{2^n} - \frac{1}{2}\right) \times VCC}{G_{EMG}}$$

$$EMG(mV) = EMG(V) \times 1000$$

Donde:

VCC = 3.3V (voltaje de operación)

G_{EMG} = 1009 (ganancia del sensor)

ADC → Valor muestreado del canal

n → Número de bits de resolución del ADC del canal (por defecto 10 bits)

2.3.3.5. Módulo EDA

Para la medición de la actividad electrodérmica se ha empleado el módulo EDA de la marca Bitalino.

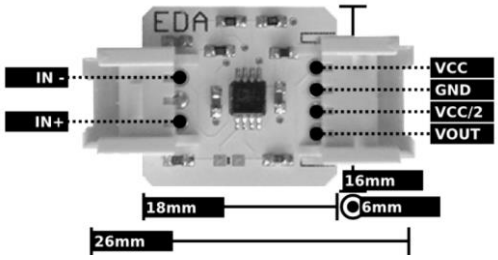


Figura 9. Módulo EDA de Bitalino.

Las características principales de este módulo se muestran en la siguiente tabla:

Ganancia	2
Rango [MOhm]	0 - 1 (con VCC = 3.3V)
Ancho de banda [Hz]	0 - 3
Consumo [mA]	2
Electrodos	2

Tabla 6. Características del módulo EDA de Bitalino. Fuente: (BITalino, 2015a).

La función de transferencia empleada para la conversión de los valores digitales adquiridos a través del ADC, para mostrarlos en valores que oscilarán entre 1 y ∞ µS, es la siguiente:

$$R(MOhm) = 1 - \frac{ADC}{2^n}$$

$$EDA(\mu S) = \frac{1}{R(MOhm)}$$

Donde:

R (MOhm) = Valor de la resistencia del sensor en mega-Ohm

ADC → Valor muestreado del canal

n → Número de bits de resolución del ADC del canal (por defecto 10 bits)

2.3.3.6. Módulo PZT

Para la medición de la respiración ha sido empleado el sensor PZT de la marca Bitalino.

Figura 10. Sensor PZT de Bitalino.

Las características principales de este módulo se muestran en la siguiente tabla:

Ancho de banda [Hz]	0 – 15
Consumo [mA]	4

Tabla 7. Características sensor PZT de Bitalino. Fuente: (BITalino, 2019).

La función de transferencia empleada para la conversión de los valores digitales adquiridos a través del ADC, para mostrarlos en valores que oscilarán entre -50 y 50 %, es la siguiente:

$$PZT(\%) = \left(\frac{ADC}{2^n} - \frac{1}{2}\right) \times 100\%$$

Donde:

ADC → Valor muestreado del canal

n → Número de bits de resolución del ADC del canal (por defecto 10 bits)

2.3.3.7. Acelerómetro (ACC)

Para la medición de la aceleración que experimenta el piloto se ha empleado el módulo ACC de la marca Bitalino.

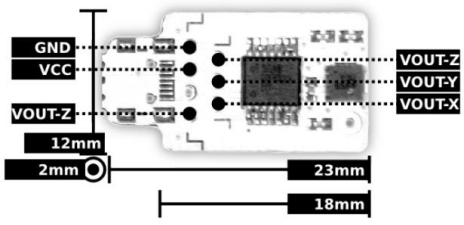


Figura 11. Módulo ACC de Bitalino. Fuente: (BITalino, 2016a).

Las características de este módulo se muestran en la siguiente tabla:

Rango [g]	± 3
Ancho de banda [Hz]	0 - 50
Consumo [mA]	0.35
Voltaje de entrada [V]	1.8 – 3.6

La función de transferencia empleada para la conversión de los valores digitales adquiridos a través del ADC, para mostrarlos en valores que oscilarán entre -3 y 3 g, es la siguiente:

$$ACC(g) = \frac{ADC - C_{min}}{C_{max} - C_{min}} \times 2 - 1$$

Donde:

ADC → Valor muestreado del canal

C_{min} → Valor mínimo de calibración

C_{max} → Valor máximo de calibración

Los valores de calibración se determinan realizando una rotación muy lenta de 360° de la placa del sensor para forzar al acelerómetro a cruzar los -1g y 1g impuestos por la gravedad en cada eje. Se recomienda que se realice un filtrado o un promedio de los datos para eliminar los temblores naturales.

2.3.3.8. Temperatura corporal del piloto (TcP) (LM35)

Para la medición de la temperatura corporal del piloto, se ha empleado un sensor analógico muy común en el mercado y de bajo coste, el LM35 de Texas Instruments.

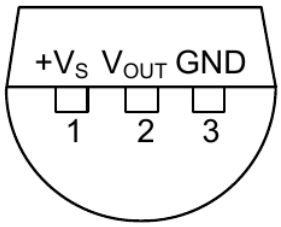


Figura 12. Esquema del sensor LM35 de Texas Instruments.

Algunas de las características principales de este sensor son:

calibración	Directamente en Celsius
Linealidad	+10 mV/ºC de factor de escala
Voltaje de alimentación [V]	4 - 30
Consumo [µA]	Menor de 60
Precisión [ºC]	0.5 (a 25 ºC)

Tabla 8. Características del sensor LM35 de Texas Instruments. Fuente: (Peeters et al., 2010).

La función de transferencia de este sensor en el datasheet oficial, si bien es correcta, para nuestra aplicación no es suficiente para obtener el valor correcto. Por ello la siguiente ecuación que se muestra está adaptada al entorno en el que opera el sensor.

La función de transferencia empleada para la conversión de los valores digitales adquiridos a través del ADC, para mostrarlos en valores que oscilarán entre 2 y 150 °C, es la siguiente:

$$LM35(°C) = \frac{ADC \times (V_{REF} \times 100)}{(2^n - 1)} = \frac{ADC \times 330}{1023}$$

Donde:

ADC → Valor muestreado del canal

n → Número de bits de resolución del ADC del canal (por defecto 10 bits)

V_{REF} → Voltaje de referencia en Arduino DUE = 3.3V

2.3.3.9. SpO2 (MAX30102)

El módulo empleado para la medición de la saturación de oxígeno en sangre es de la marca ARCELI, adquirido en la tienda de Amazon. El corazón principal de esta placa es el módulo integrado MAX30102 de la compañía Maxim Integrated (Maxim Integrated, 2015).

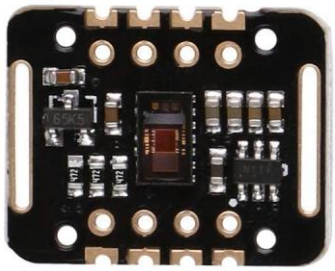


Figura 13. Placa ARCELI, y en el centro el módulo integrado MAX30102.

La comunicación de este sensor con la placa Arduino, se desarrolla de forma digital, mediante el protocolo i2C o también llamado TWI.

Algunas de sus características más destacadas son, la capacidad de operar en entornos desde -40 ºC hasta 85 ºC y su capacidad de configurar una alta tasa de muestreo de los datos. Para una correcta medida, se recomienda que la presión que se ejerza sobre el sensor, al colocarlo en su lugar de funcionamiento, deba ser constante; una buena solución es ajustarlo con una banda elástica.

2.3.3.10. Cable USB a UART FTDI C232HD-DDHSP-0

El cable de la marca FTDI, modelo C232HD, ha sido empleado para conectar un segundo puerto serie del ordenador con la placa Arduino, para hacer el calibrado del sensor ACC (acelerómetro). Esto método se decidió en principio por la simplicidad del código a programar tanto en el propio Arduino como en Matlab. Solo se emplean los cables naranja y amarillo, TXD y RXD.

Figura 14. Cable USB a UART FTDI C232HD-DDHSP-0. Fuente: (Technology & International, 2011).

2.3.3.11. Prototipo *shield* para Arduino DUE

Para conectar todo el sistema de sensado a la placa Arduino DUE, se ha utilizado una PCB perforada para el prototipado del circuito electrónico.

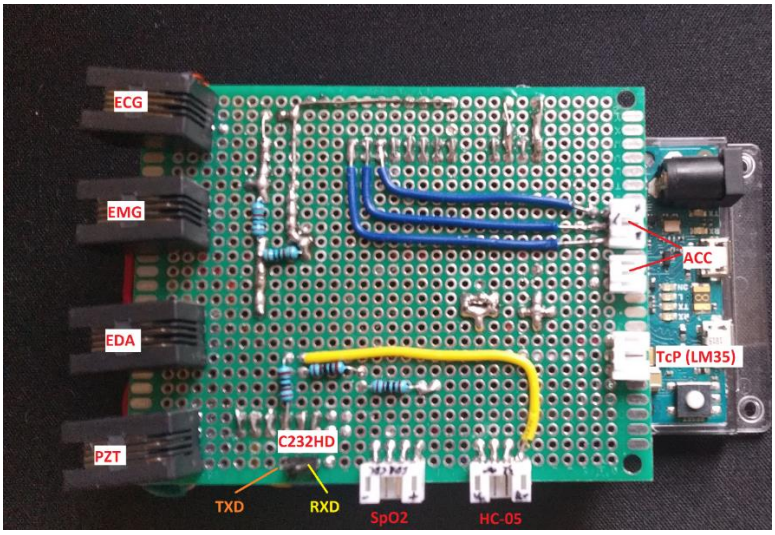


Figura 15. Shield prototipo.

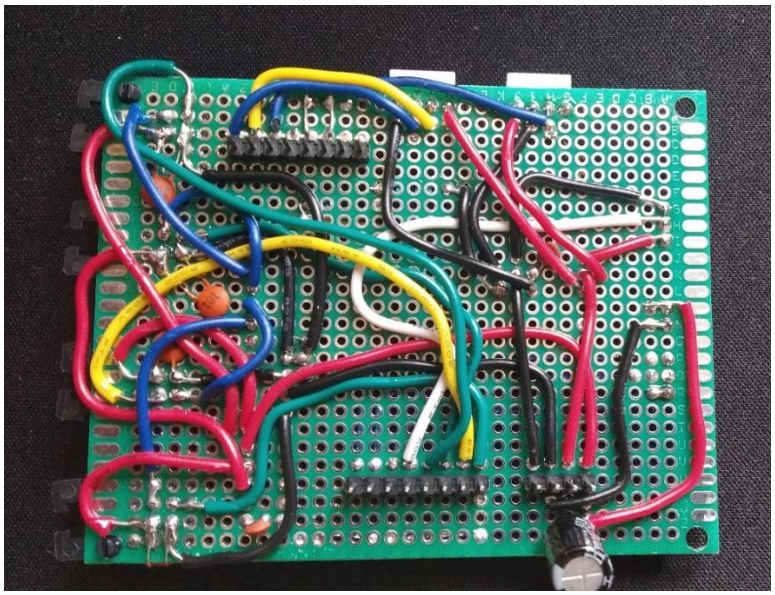


Figura 16. Parte inferior del shield.

2.4. DISEÑO DE EXPERIMENTOS

En este apartado, se procederá a desarrollar como se van a realizar las pruebas con todo el material del que se dispone. Como el objetivo de este trabajo, es observar el funcionamiento general de todo el sistema e identificar sus posibles carencias, las pruebas a realizar si se desarrollan dentro del cockpit, pero sin poner el simulador en marcha. A esto se le ha llamado, pruebas en estático.

Una vez aclarado el rumbo a tomar en el desarrollo de las pruebas, a continuación, se procede a detallar los puntos de los que constará la prueba, para finalmente concluir con las indicaciones de donde hay que colocar los sensores y el resto del sistema. A todo esto, hay que sumarle que también se pondrá a prueba la efectividad de la transmisión por Bluetooth dentro del simulador, dado que el simulador está en una habitación y en una cabina contigua con ventanal está situada la sala de control. A priori, se sospecha que no debería de presentarse ningún problema en este ámbito, debido a la corta distancia entre el simulador y la sala de control.

Figura 17. Simulador G-UPV.

Figura 18. Sala de control del G-UPV.

A continuación, se detallan los pasos a tomar durante el desarrollo de la prueba:

1. Fuera del cockpit, se montarán todos los sensores en el sujeto sin el mono ni el casco. Se analiza cómo responde el sistema en su conjunto. En caso de haber fallos, se prueba cada sensor por separado en el sistema, para ver cuál puede ser la causa del fallo.

2. Repetir la misma operación que el punto anterior, pero esta vez, con el mono y el casco enfundados, y sin entrar aún dentro del cockpit.

3. Probar el acelerómetro andando con el mono y el casco puesto y hacer unas pocas sentadillas. Esto se hace nada más que para comprobar que se captan variaciones, por muy ligeras que puedan ser, debido a que el simulador no se va a poner en marcha.

4. Probar el sistema dentro del cockpit y sin moverse el sujeto. De este modo también se puede analizar el sistema inalámbrico por Bluetooth.

5. Probar dentro del cockpit, mover el volante hacia ambos extremos, aproximadamente 180º de giro es suficiente. Repetir el movimiento diez veces para tener una medición fiable. Se empezará moviendo el volante a la izquierda.

6. Dentro del cockpit, probar el giro del cuello de extremo a extremo, simulando la mirada hacia el vértice de una curva muy cerrada, sin girar el volante. Repetir el proceso diez veces.

7. Dentro del cockpit, combinar los puntos 5 y 6, con el movimiento sincronizado de giro de volante y giro de cuello. Repetir diez veces.

Para las pruebas participaron tres voluntarios de diversas complexiones, para comprobar los resultados obtenidos con diferentes tallas del mono y ajustes.

- Voluntario 1: Estatura 1.79 m; Peso: 68 Kg; Edad: 20 años.
- Voluntario 2: Estatura 1.74 m; Peso: 69 Kg; Edad: 21 años.
- Voluntario 3: Estatura 1.76 m; Peso: 97 Kg; Edad: 21 años.

La disposición de los sensores se muestra en la figura a continuación, junto con algunas imágenes tomadas para detallar la posición exacta.

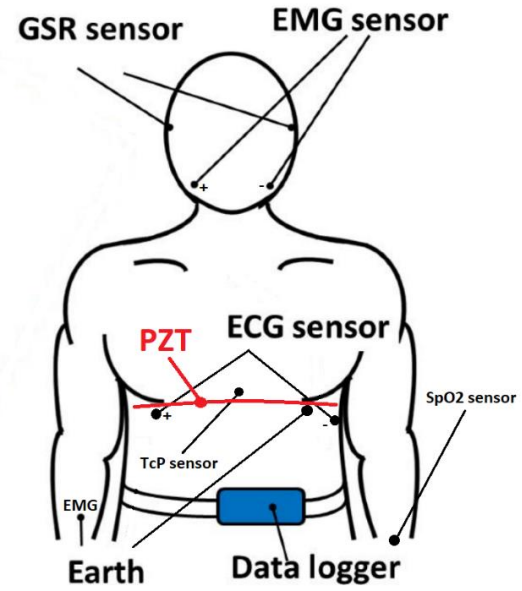


Figura 19. Posicionamiento de los sensores.

Figura 20. Posicionamiento sensor EMG y EDA.

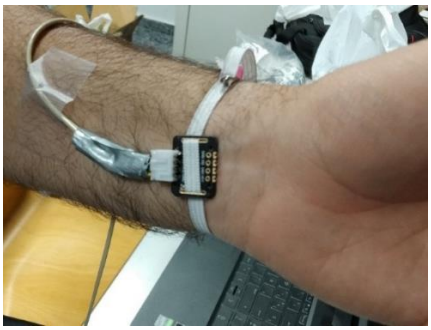


Figura 21. Posicionamiento sensor SpO2.

El acelerómetro por otra parte va montado junto con la placa Arduino con un cableado corto, de este modo se evita demasiado cableado por el cuerpo del piloto que pueda aumentar la probabilidad de fallo por rotura de algún cable, además de la mejora en comodidad debido a la ya gran cantidad de cables usados.

Para terminar la sección, se muestra en la siguiente figura como quedaría el sistema con el piloto, dentro del cockpit del simulador G-UPV.

Figura 22. Piloto dentro del cockpit del G-UPV.

Cabe puntualizar, que se tomará como prueba definitoria para posterior comparación entre voluntarios, el resultado del paso siete de cada prueba realizada a cada voluntario. Se ha tomado este criterio, en referente a que el último paso de las pruebas a cada voluntario, es la combinación de todos los tipos anteriores, dando lugar a lo que puede ser el punto más crítico donde puedan llegar a fallar algunas mediciones.

3. RESULTADOS Y DISCUSIÓN

Terminadas las pruebas, se procede a mostrar los resultados obtenidos y a su posterior análisis, para determinar si se pueden considerar por válidas las mediciones de estos. Puntualizar, que el eje de abscisas está dividido en muestras y no en tiempo, esto se debe a que aun se desconoce como transmitir los datos temporales de cada medición con precisión, debido a que en Arduino se almacenan 200 valores para enviarlos como un paquete. No obstante, de forma orientativa y suponiendo que cada interrupción son 5 ms, cada muestra puede considerarse como 5 ms aproximadamente. Por lo que, si se tienen 2000 muestras, por ejemplo, habrán transcurrido aproximadamente 10 segundos. Puntualizar, que todas las gráficas comparativas que se

muestran a continuación, las señales de los voluntarios no están sincronizadas en espacio ni tiempo, por tanto, cualquier coincidencia de fase de las señales entre voluntarios deberá considerarse como casual. A continuación, se muestran los resultados por cada sensor, comparando entre los diferentes voluntarios, en el último paso de cada prueba. Con esto lo que se pretende es observar la reproducibilidad y la fiabilidad de las mediciones, al desmontar y montar el sistema en cada voluntario.

3.1. RESULTADOS ECG

Partiendo del sensor, quizás más relevante, se muestra una gráfica de las señales ECG de los tres voluntarios.

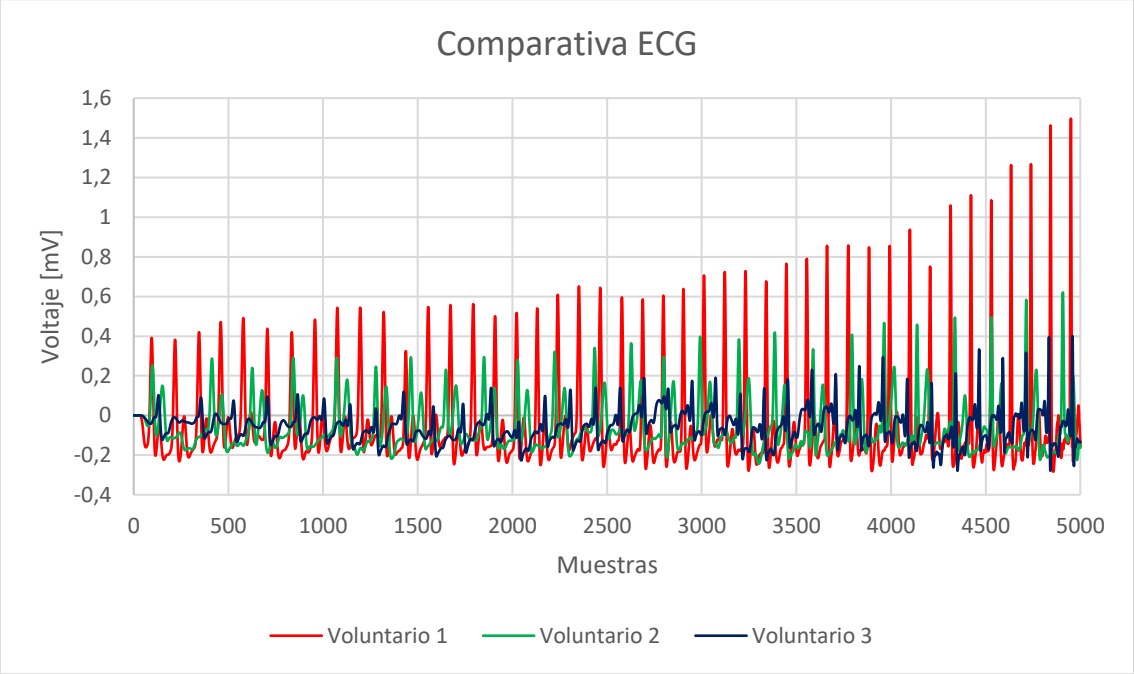


Figura 23. Comparativa ECG.

En el ECG, se observa como los voluntarios 1 y 2 que son de una complexión parecida, aunque los picos tienen amplitudes diferentes, pueden ser fácilmente identificables, no obstante, la señal ECG del voluntario 3, es muy pobre. Considerando que los electrodos se han colocado en el mismo lugar en todas las pruebas, se cree que la complexión de este último voluntario puede haber influido de alguna forma en la medición. Se puede comprobar que la muestra ECG es bastante estable, no obstante, cabe mencionar que sí ha habido alguna de las pruebas aisladas donde se han observado algunos picos anómalos. Estos problemas están seguramente relacionados con el correcto contacto de los electrodos con la piel. Pero en definitiva la señal ECG se puede considerar que satisface las necesidades del trabajo.

3.2. RESULTADOS EMG

En la señal EMG se ha observado que hay interferencia de otra señal en esta, la cual, analizando las demás pruebas, se ha determinado que es el mismo patrón que la señal ECG. Esto es algo que preocupa porque se ve claramente que hay una clara interferencia que no deja mostrar con

claridad la señal EMG. Por el contrario, la interferencia de la señal ECG en la EMG, al superponer las dos señales en la misma figura, se puede apreciar que no es de un orden grande.

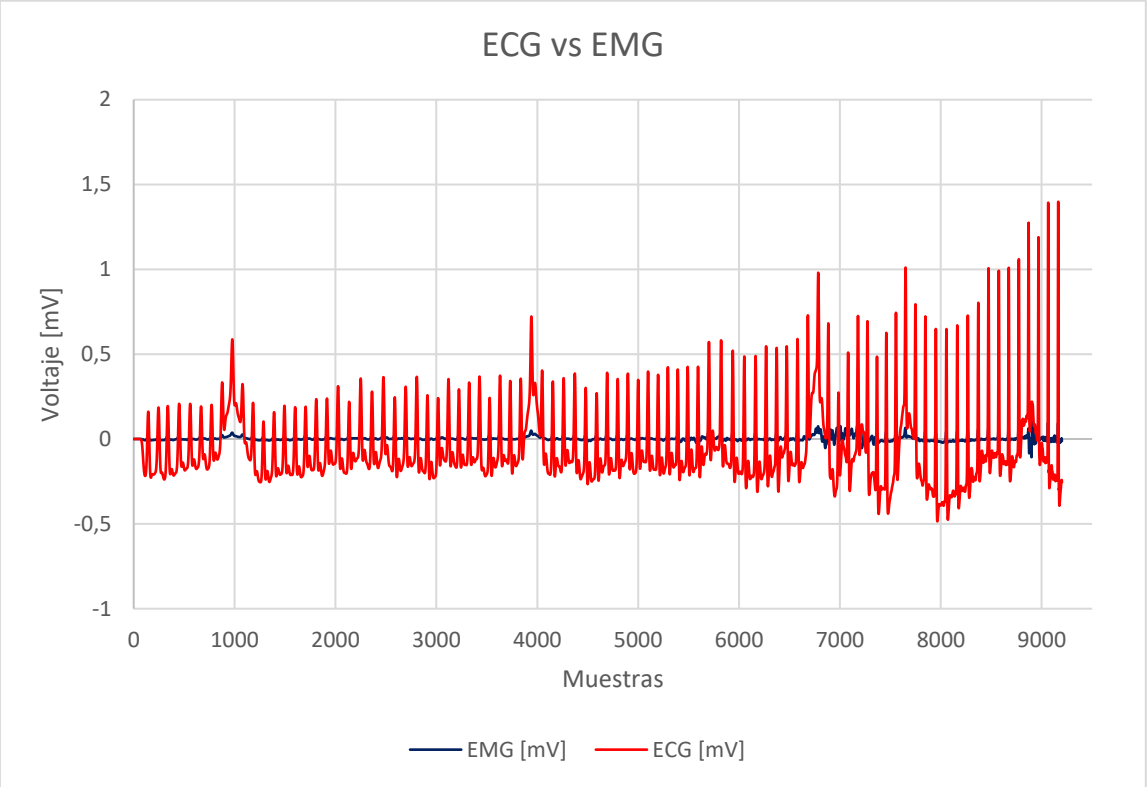


Figura 24. Solapamiento señales ECG y EMG.

Viéndolo de este modo, aunque la toma de la señal es mejorable, parece indicar que el problema no es tan grave como parecía al principio; y que, si bien no se pude afirmar que la señal EMG es definitivamente válida, parece que se puede distinguir, aunque sea con muy poca amplitud, la señal.

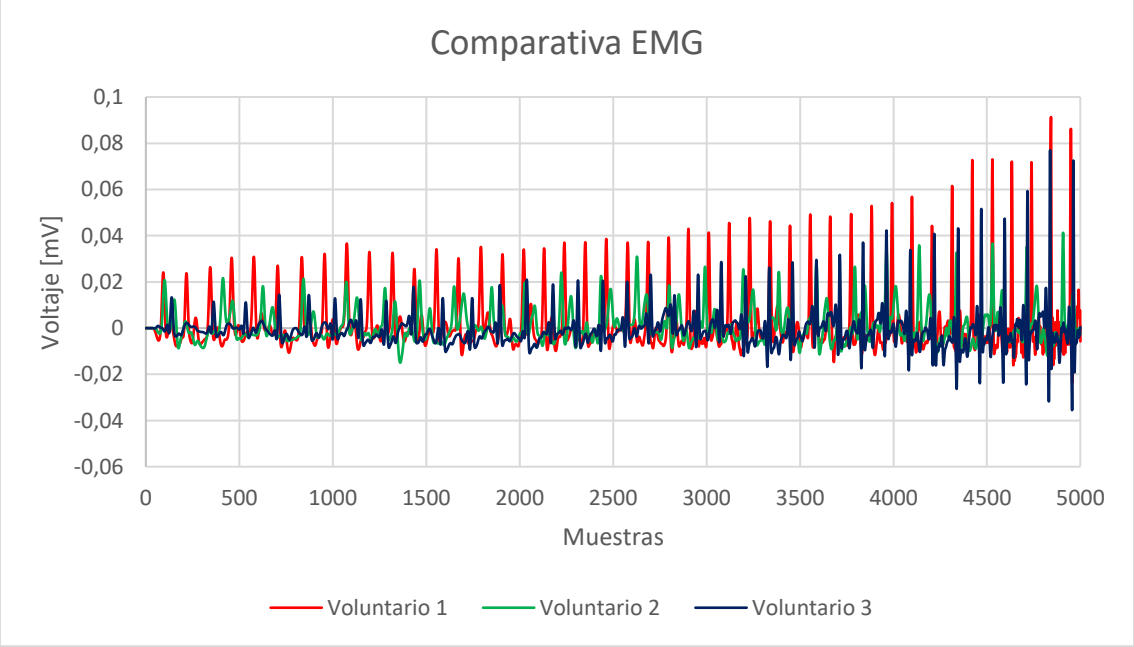


Figura 25. Comparativa EMG.

Con la señal EMG, no se observa demasiada diferencia entre voluntarios, salvo el ya mencionado acoplamiento de la señal ECG. Las mediciones son leíbles, aunque todas tienen muy poca amplitud. Este déficit de amplitud no se está muy seguro de que puede ser, no obstante, si se puede considerar que es más un problema de *hardware* que de *software*. También se desconoce, porque la señal ECG como la EMG, aumentan en amplitud llegando al final de las mediciones, hay ciertas sospechas de que pueda ser relativo al filtrado de media móvil empleado en Matlab, no obstante, no está verificado.

3.3. RESULTADOS EDA

Continuando con la señal EDA, se puede decir que es menos propensa a sufrir interferencias, o si en ocasiones las sufre, son muy poco relevantes en todas las pruebas que se han realizado.

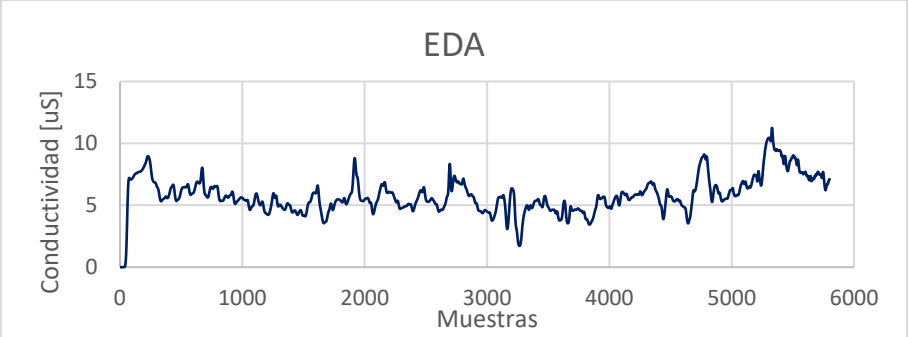


Figura 26. Señal EDA.

Aunque la figura parece que muestra interferencias, se ha comprobado que la mayoría de las oscilaciones se deben a la falta de buen contacto del electrodo con la piel, notándose que esta medida es muy sensible, y se precisa de un buen contacto. En diversas pruebas ocurrió que la medición se perdía, estando los electrodos conectados, y que, si se sujetaban con mayor presión garantizando un mejor contacto, las mediciones volvían a ser correctas. Este problema es posiblemente debido al estado del gel preaplicado en los electrodos autoadhesivos, quizás puede que lleven tiempo sin usarse y pierdan un poco de efectividad. No obstante, las medidas registradas pueden darse por buenas en cierta medida, y que una posible solución futura sería emplear para este sensor unos electrodos reutilizables de plata y aplicando gel conductor a la piel, para comprobar si hay una mejora sustancial.

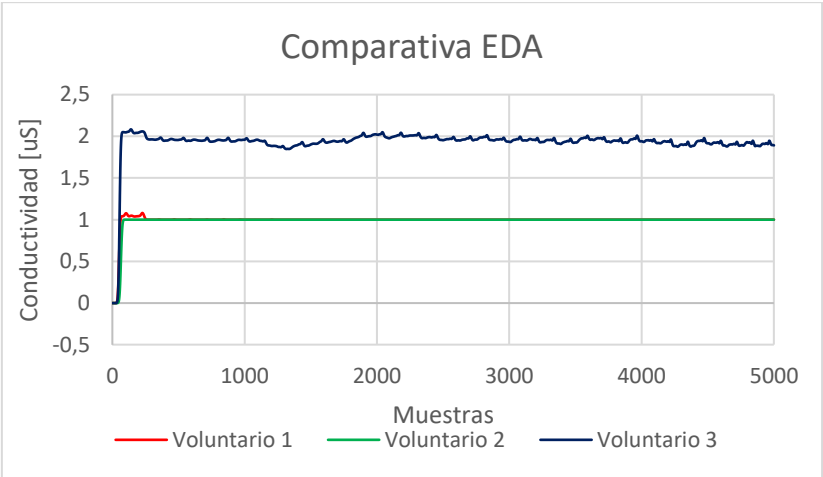


Figura 27. Comparativa EDA.

Analizando los resultados entre los tres voluntarios, sí se puede afirmar, que las mediciones son muy sensibles a perder la conexión en los electrodos, por falta de ajuste al cuerpo. Hay que considerar para este sensor, que el correcto ajuste de los electrodos a la piel es determinante para conseguir las mediciones que se desean, por lo tanto, hay que garantizar el contacto completo del electrodo con la piel. Este fallo de ajuste se observa en la figura anterior, donde el valor de 1 constante es considerado como perdida de conexión de uno de los electrodos.

3.4. RESULTADOS PZT

Continuando con la señal de respiración, PZT, esta se ha comprobado que es correcta, pero se ha observado, que con el sensor que se ha utilizado, ajustando la banda elástica con diferentes presiones no se ha conseguido que aumente la sensibilidad en la medición de forma muy significativa. Puede que también tenga que ver con el modo de respirar de cada voluntario, porque, con alguno de ellos las mediciones no se apreciaban demasiado, y con otros se veían perfectamente las respiraciones, como se muestra en la siguiente figura, y todo ello sin variar la ubicación del sensor.

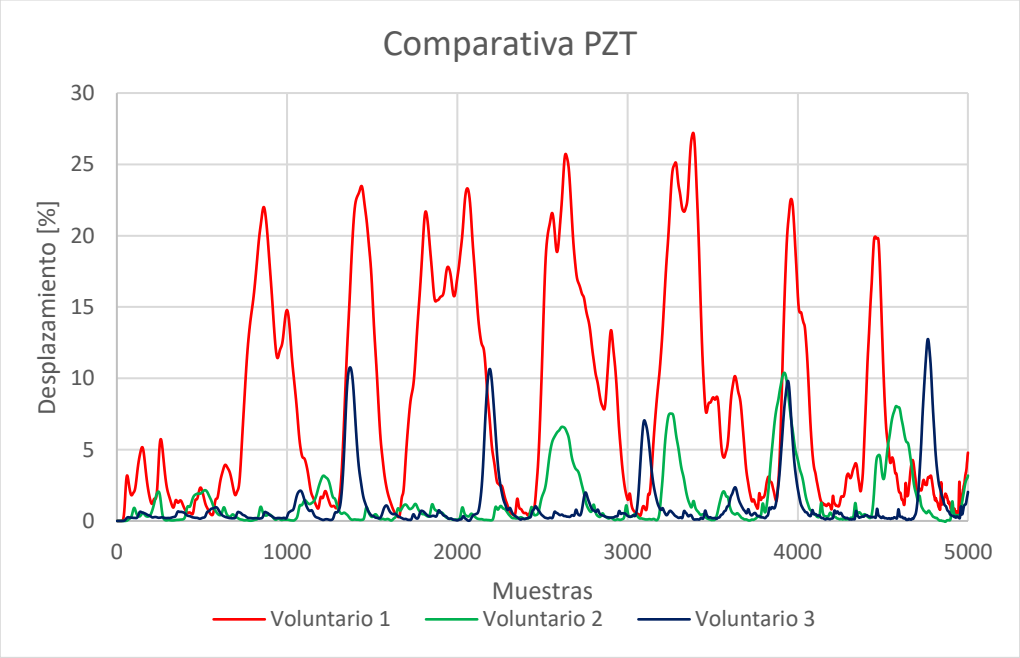


Figura 28. Comparativa PZT.

Respecto PZT, es un sensor que ofrece buenas mediciones, aunque el problema de este radica en el correcto ajuste de la banda elástica alrededor de la caja torácica. Si no se ajusta bien, se obtienen mediciones de menor amplitud. Además, también se ha observado que influye la forma en la que respira cada voluntario, es decir, exagerando en mayor o menor medida la expansión y contracción del pecho.

3.5. RESULTADOS ACC

Con el acelerómetro, ACC, no ha habido ningún problema observable, ha funcionado como se esperaba. La medición en cada voluntario ha sido satisfactoria en todas las pruebas, sin ninguna perdida de conexión o acoplamiento en la señal.

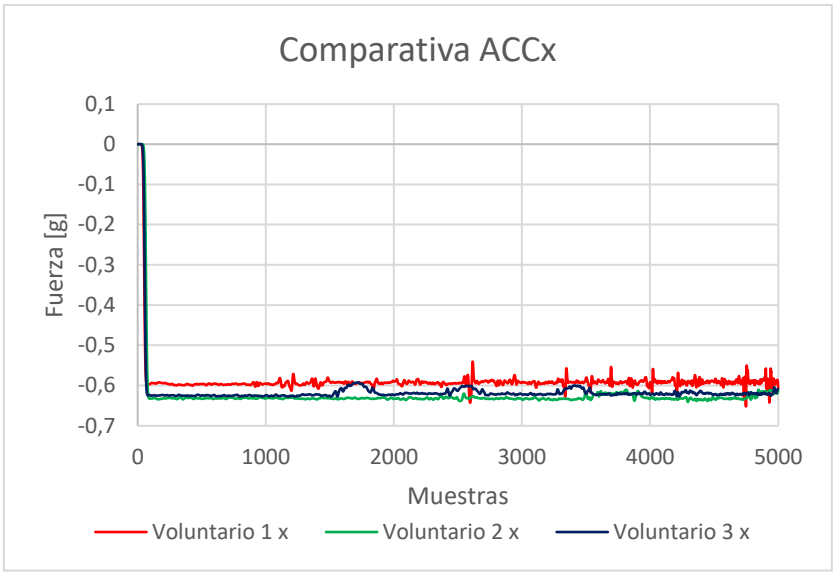


Figura 29. Comparativa ACC eje x.

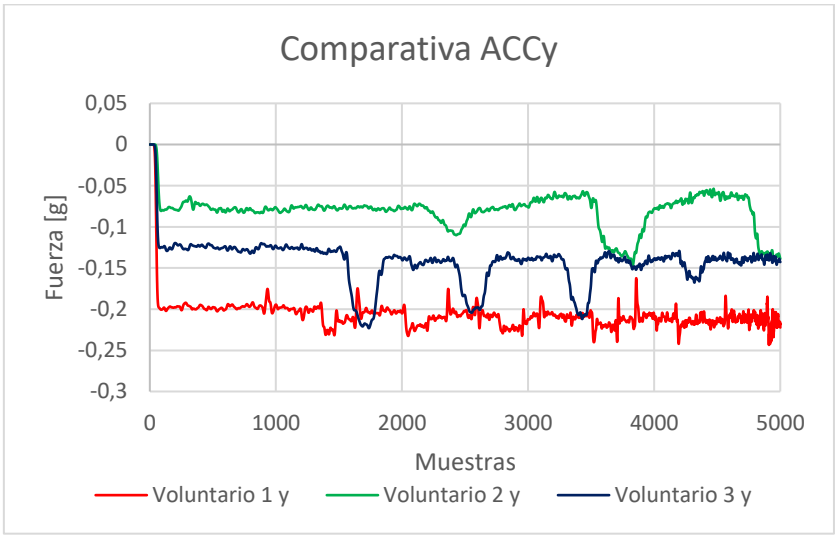


Figura 30. Comparativa ACC eje y.

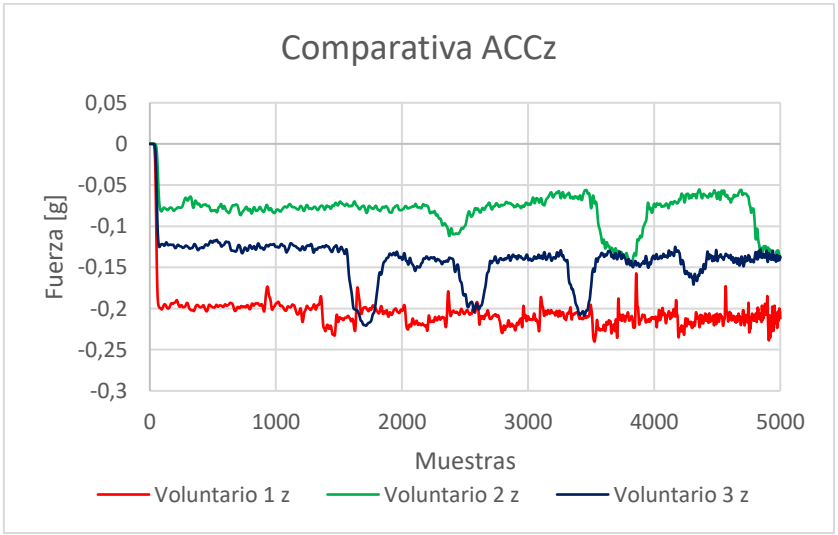


Figura 31. Comparativa ACC eje z.

3.6. RESULTADOS TcP Y SpO2

Para finalizar, se muestra la TcP, donde se ha observado que la medición de la temperatura es estable pero no la correcta, esto se debe a la colocación del sensor de temperatura en el pecho del piloto sujeto con cinta. Se ha comprobado que no es un buen método para sujetar el sensor de temperatura, debido a que este no puede garantizarse que este perfectamente en contacto con la piel, dando lugar a que el sensor este rodeado del segundo mejor aislante térmico si este está en estático, el aire. Por lo que el sensor mide bien, pero hay que encontrar mejores modos se garantizar su sujeción.

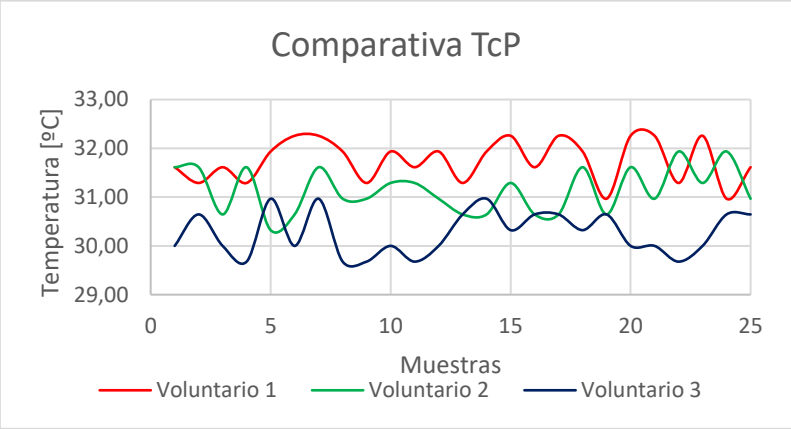


Figura 32. Comparativa TcP.

La temperatura corporal del piloto tampoco ha variado demasiado entre voluntarios. Las variaciones que se observan son debidas al mayor o menor contacto con la piel del sensor en cada caso. Como se ha comentado anteriormente, se precisa mejorar la sujeción del sensor TcP en futuras pruebas.

En cuanto al sensor SpO2, ha sido imposible obtener una medición correcta situando el sensor en su lugar correcto. En el siguiente capítulo se explica el porqué.

4. PROBLEMAS ENCONTRADOS DURANTE EL DESARROLLO DEL TRABAJO

Lo primero que se intentó realizar en este trabajo, fue la implementación directa de una placa BITalino para la toma de datos de todos los sensores. El problema que se observó fue que esta constaba solo de 6 entradas analógicas, y con los sensores empleados se precisaba de 8 entradas analógicas. Por lo tanto, se probó una opción intermedia; existe la posibilidad de emplear BITalino junto con Arduino, cargando un código en Arduino, realizado por los creadores de BITalino, para que la placa BITalino lea parte de los sensores y envíe los datos a Arduino. El primer problema fue que solo era compatible con el código un tipo de placa Arduino vieja llamada Diecimilla, no obstante, como esta era de arquitectura AVR, se probó con una placa Arduino MEGA, y en esta funcionaba, pero como se precisaba de realizar interrupciones en el código con un *timer*, este código era incompatible y bloqueaba la ejecución de toda la programación. Por lo tanto, se decidió emplear exclusivamente una placa Arduino que fuera lo suficientemente potente para el procesamiento de todas las señales. De este modo se decidió por crear todo el código sobre una placa Arduino DUE, y que mediante un *shield* se conectarían

todos los sensores a esta, dando lugar a poder hacer todo a medida. Hay que mencionar que se probó la placa BITalino con Arduino DUE, pero como la arquitectura de esta última es ARM, el código nunca funcionó.

Con el sensor SpO2, el problema que se detectó fue que el código de este, junto con su librería, no era compatible si se ejecutaba dentro de una interrupción, al menos no de una forma fácil o asequible para alguien con conocimientos limitados en electrónica y programación. Por ello esa parte del código debía ejecutarse fuera de la interrupción, teniendo solo 5 milisegundos para ejecutar todo el muestreo para dar un valor. Se probaron diferentes opciones que permiten ajustar el funcionamiento del sensor, tiempo del pulso del led infrarrojo, tamaño del vector de mediciones, etc; ninguna con éxito. Con la configuración que se dispone en el código de Arduino de este trabajo, sí es verdad que se conseguía tomar valores de saturación sin colapsar el programa, es decir, dentro de los 5 milisegundos; pero poniendo el sensor en una superficie a cierta distancia que no reflejaba la distancia que debía llevar al ponerlo en la muñeca del piloto. Probando ajustes por separado solo con el sensor, se conseguían buenas mediciones pero que requerían cierto tiempo extra que no era compatible con el código empleado. Por ello una posible solución a este problema sería, buscar una alternativa a las interrupciones para mejorar la compatibilidad del sensor MAX30102; una de ellas pendiente de estudio podría ser el *free running mode* del conversor analógico a digital, ADC.

5. CONCLUSIONES

De este trabajo se puede concluir que, pese a las dificultades encontradas, los resultados han sido moderadamente satisfactorios, obviando que existe un amplio margen de mejora. Además se puede concluir que, para el ámbito cercano desde el simulador hasta la sala de control, el dispositivo bluetooth ha cumplido su función sin problemas.

Del sensor ECG, se puede concluir que, al ser una de las señales indispensables debido a que es un buen indicador del estado fisiológico del piloto, como punto de partida, es indispensable una medición fiable. Por tanto, con los resultados observados, se puede considerar que el sensor ECG en combinación con todo el sistema, no ha presentado grandes problemas y sus mediciones pueden considerarse como válidas para el objetivo de este trabajo.

Respecto al sensor EMG, considerando que hay acoplamiento de la señal ECG, aunque este sea representado con poca amplitud, debido a la baja amplitud que se ha observado en la adquisición de la señal de EMG dados los resultados, la diferenciación de la señal EMG de la ECG puede verse dificultada, por lo que se concluye que hay margen de mejora y no se pueden dar por válidas las mediciones de EMG.

En relación con el sensor EDA, se concluye que, pese a que las mediciones pueden ser muy sensibles respecto al correcto ajuste de los electrodos, cuando estos están correctamente ajustados, según las observaciones de los resultados, se pueden considerar por válidas.

Del sensor PZT, se puede concluir que, se obtienen buenas mediciones, pero la amplitud de la señal de estas estará condicionada por el correcto ajuste de la banda elástica y la forma en la que respira el sujeto.

Respecto al sensor ACC, se puede concluir que, el sensor satisface las necesidades del trabajo, sin ningún tipo de problema.

En relación con el sensor TcP, se concluye que las mediciones son estables, pero no correctas, debido al tipo de ajuste del sensor al cuerpo. Por ello es preciso buscar nuevos métodos de sujeción que garanticen el mejor contacto del sensor con la piel sin ser invasivos.

Por último, respecto al sensor SpO2, se concluye que presenta problemas de tipo *software*, el cual no se ha encontrado una solución adecuada. Por tanto, es necesario probar nuevas opciones en lo referente al código programado en Arduino, que permitan una medición correcta sin comprometer el funcionamiento de todo el sistema.

No obstante, este es uno de los primeros pasos en una rama de investigación como es la de monitorizar a un piloto de competición, donde no hay apenas cobertura en investigación y existe un largo recorrido de estudios posteriores, ya sea en sistemas de monitorización, como en el estudio de las señales obtenidas para determinar el estado fisiológico del piloto, y en general, el estudio de una fusión de sensores. Por tanto, este trabajo es solo un pequeño paso en un largo recorrido, el cuál humildemente, se espera que pueda servir no como punto de partida, pero si como apoyo de que opciones son mejores a considerar y que problemas pueden surgir.

6. CORRECCIONES Y FUTURAS MEJORAS

Además de las posibles correcciones mencionadas respecto a las señales de los distintos sensores, de forma más global, hay posibles correcciones en lo que se refiere al ruido de las señales; este problema podría desaparecer si yendo un paso más allá se usa una placa PCB diseñada a medida en lugar de la placa prototipo que se ha empleado, seguramente habría una gran mejora en la calidad de algunas señales, así como el uso de cables con un mejor apantallamiento. Una mejora, podría ser, que en lugar de emplear una placa Arduino DUE que es muy genérica con entradas y salidas de sobra, se optase por una más potente que dispone de menor número de pines pero que serían suficiente para satisfacer las necesidades de este trabajo, como podría ser la placa de desarrollo ESP32, la cual es mucho más rápida que la Arduino DUE, y además esta consta de doble núcleo, permitiendo mediciones en paralelo, agilizando todo el proceso.

Figura 33. Espressif ESP32.

Figura 34. Módulo ADS1115.

No obstante, se comenta y algunas personas han comprobado, que una de las pegas de esta placa es que sus conversores ADC no son demasiado buenas, no por resolución de bits, sino por desfases en la medición, comprobándolo con un osciloscopio. Una solución a esto sería emplear módulos ADC externos, como ADS1115 de 16 bit, que es mucho mejor que el ADC de Arduino y el del ESP32.

7. BIBLIOGRAFÍA

Alzu'Bi, H. S., Al-Nuaimy, W., & Al-Zubi, N. S. (2013). EEG-based driver fatigue detection. *Proceedings - 2013 6th International Conference on Developments in ESystems Engineering, DeSE 2013*, *December*, 111–114. https://doi.org/10.1109/DeSE.2013.28

BITalino. (2015a). *Electrodermal Activity (EDA) Sensor Data Sheet*. 0–1. http://bitalino.com/

BITalino. (2015b). *Electromyography (EMG) Sensor Data Sheet*.

BITalino. (2016a). *Accelerometer (ACC) Sensor Data Sheet Accelerometer (ACC) Sensor Data Sheet*.

BITalino. (2016b). Electrocardiography (ECG) sensor data sheet. *PLUX-Wireless Biosignalswireless Biosignals*, 2. https://bitalino.com/datasheets/REVOLUTION_ECG_Sensor_Datasheet.pdf

BITalino. (2019). Respiration (PZT) Sensor Data Sheet. *PLUX-Wireless Biosignals*, 2. https://bitalino.com/datasheets/REVOLUTION_PZT_Sensor_Datasheet.pdf

Clement, F. S. C., Vashistha, A., & Rane, M. E. (2016). Driver fatigue detection system. *Proceedings - IEEE International Conference on Information Processing, ICIP 2015*, *August*, 229–234. https://doi.org/10.1109/INFOP.2015.7489384

David P. Ferguson; Nicholas D. Myers. (2018). *PHYSICAL FITNESS AND BLOOD GLUCOSE INFLUENCE PERFORMANCE IN INDYCAR RACING*. *32*(11), 3193–3206.

Dubin, Electrocardiografía práctica. Lesión, trazado e interpretación. 3ª ed. McGraw-Hill

Harrison, Manual de medicina elemental. 17ª ed.

Hu, X., & Lodewijks, G. (2020). Detecting fatigue in car drivers and aircraft pilots by using non-invasive measures: The value of differentiation of sleepiness and mental fatigue. *Journal of Safety Research*, *72*, 173–187. https://doi.org/10.1016/j.jsr.2019.12.015

Jing, D., Zhang, S., & Guo, Z. (2020). Fatigue driving detection method for low-voltage and hypoxia plateau area: A physiological characteristic analysis approach. *International Journal of Transportation Science and Technology*, *9*(2), 148–158. https://doi.org/10.1016/j.ijtst.2020.01.002

Maxim Integrated. (2015). High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health MAX30102. *Maxim Integrated*, 1–32. https://www.maximintegrated.com/en/products/sensors/MAX30102.html

NTT. (2020). *Supporting the world's top racing car drivers, biometric information solutions*. 1–9.

Peeters, F., Peetermans, M., & Indesteege, L. (2010). Temperature Sensors. *Modern Sensors Handbook*, 347–393. https://doi.org/10.1002/9780470612231.ch8

Potkanowicz, E. S., & Mendel, R. W. (2013). The case for driver science in motorsport: A review and recommendations. *Sports Medicine*, *43*(7), 565–574. https://doi.org/10.1007/s40279-013-0040-2

Press Area (OMP). (2019). *Biometric glove set for F1 debut (OMP)*. *2019*(November), 2019.

Report, T., & Gonz, J. J. (2015). *Filtrado Básico de Señales Biomédicas Trabajo Práctico 2 : Filtrado de Señales Contenido Objetivos*. *May 2014*.

https://doi.org/10.13140/2.1.1757.2168

Seo, Y., Powell, J., Strauch, A., Roberge, R., Kenny, G. P., & Kim, J. H. (2019). Heat stress assessment during intermittent work under different environmental conditions and clothing combinations of effective wet bulb globe temperature (WBGT). *Journal of Occupational and Environmental Hygiene*, *16*(7), 467–476. https://doi.org/10.1080/15459624.2019.1612523

Sharma, M. K., & Bundele, M. (2020). Cognitive Fatigue Detection in Vehicular Drivers Using K-Means Algorithm. In *Advances in Intelligent Systems and Computing* (Vol. 1059). Springer Singapore. https://doi.org/10.1007/978-981-15-0324-5_44

Technology, F., & International, D. (2011). Future Technology Devices International Ltd USB 2 . 0 Hi-Speed to UART Cable Datasheet. *Technology*, *44*(0), 0–17.

Wen, W., Tomoi, D., Yamakawa, H., Hamasaki, S., Takakusaki, K., An, Q., Tamura, Y., Yamashita, A., & Asama, H. (2017). Continuous Estimation of Stress Using Physiological Signals during a Car Race. *Psychology*, *08*(07), 978–986. https://doi.org/10.4236/psych.2017.87064

Zangróniz, R., Martínez-Rodrigo, A., Pastor, J. M., López, M. T., & Fernández-Caballero, A. (2017). Electrodermal activity sensor for classification of calm/distress condition. *Sensors (Switzerland)*, *17*(10), 1–14. https://doi.org/10.3390/s17102324

Documento II

Presupuesto

Presupuesto

En el diseño del sistema de adquisición se incluye el coste del hardware y la mano de obra.

Hardware

Concepto	Cantidad	Precio (con IVA) [€]
Placa prototipo perforada	1	1.00
Resistencia 1k Ω	5	0.105
Surtido condensadores	1	0.50
Surtido cables	1	2.00
Conector RJ22	4	1.00
Conector JST XH	5	0.10
Arduino DUE	1	35
FTDI C232HD	1	30
Sensor ECG	1	25.00
Sensor EMG	1	25.00
Sensor EDA	1	27.50
Sensor PZT	1	98
Sensor ACC	1	24.50
Sensor LM35	1	0.5
Sensor MAX30102	1	8.00
Módulo HC-05	1	6.50
Batería LiPo 11.1V	1	16.00
Total		304.53 €

Tabla 9. Precio hardware

Mano de obra

Partiendo de que gran parte del trabajo se ha realizado en el domicilio particular, se puede calcular que aproximadamente se han trabajado 4h/día de media, 5 días a la semana, por lo tanto:

Concepto	Precio por hora	Horas	Total
Mano de obra	25 €/h	300	7500 €

Tabla 10. Precio mano de obra

Coste total del proyecto

Concepto	Total
Hardware	304.53 €
Mano de obra	7500 €
Precio total	7804.53 €

Tabla 11. Precio total

El precio total del proyecto es de SIETE MIL OCHOCIENTOS CUATRO CON CINCUENTA Y TRES euros.

Documento III

Anexos

```
#include <DueTimer.h>
#include <Wire.h>
#include "MAX30105.h"
#include "spo2_algorithm.h"

// declaracion de variables
const size_t long_buff = 200;
const size_t long_vect = 4;
byte RecData;
uint8_t t_pilVect[long_vect]; // vector bytes temp piloto (LM35)
volatile float t_pil;
byte valSPO2;
int cont = 0;
int i;
volatile int flgTx = 0;
volatile int flgBuff = 1;
volatile uint16_t buffer1ECG[long_buff];
volatile uint16_t buffer2ECG[long_buff];
volatile uint16_t buffer1EMG[long_buff];
volatile uint16_t buffer2EMG[long_buff];
volatile uint16_t buffer1EDA[long_buff];
volatile uint16_t buffer2EDA[long_buff];
volatile uint16_t buffer1PZT[long_buff];
volatile uint16_t buffer2PZT[long_buff];
volatile uint16_t buffer1ACCx[long_buff];
volatile uint16_t buffer2ACCx[long_buff];
volatile uint16_t buffer1ACCy[long_buff];
volatile uint16_t buffer2ACCy[long_buff];
volatile uint16_t buffer1ACCz[long_buff];
volatile uint16_t buffer2ACCz[long_buff];
volatile uint16_t bff1calACCx[long_buff];
volatile uint16_t bff1calACCy[long_buff];
volatile uint16_t bff1calACCz[long_buff];
volatile uint16_t bff2calACCx[long_buff];
volatile uint16_t bff2calACCy[long_buff];
volatile uint16_t bff2calACCz[long_buff];
int ACCcont = 0;
int j;
byte ACCRecData;
volatile int ACCflgTx = 0;
volatile int ACCflgBuff = 1;

//Definicion MAX30102-----------------------------------------------------
MAX30105 particleSensor;

#define MAX_BRIGHTNESS 255

#if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168__)
//Arduino Uno doesn't have enough SRAM to store 100 samples of IR led data and red led da
//To solve this problem, 16-bit MSB of the sampled data will be truncated. Samples become
uint16_t irBuffer[100]; //infrared LED sensor data
uint16_t redBuffer[100];  //red LED sensor data
#else
uint32_t irBuffer[100]; //infrared LED sensor data
uint32_t redBuffer[100];  //red LED sensor data
#endif

int32_t bufferLength = 60; //data length
int32_t spo2; //SPO2 value

int8_t validSPO2; //indicator to show if the SPO2 calculation is valid
int32_t heartRate; //heart rate value
int8_t validHeartRate; //indicator to show if the heart rate calculation is valid
//Fin definicion MAX30102-------------------------------------------------


void setup() {
  Timer3.attachInterrupt(timerfcn);
  Timer4.attachInterrupt(calACC);
```

```
  //Serial.begin(115200);
  Serial1.begin(115200); //Serial que usa el modulo Bluetooth
  Serial2.begin(115200); //Serial C232HD
  // configuracion sensor SPO2-------------------------------------------------------------
  particleSensor.begin(Wire, I2C_SPEED_FAST); //Use default I2C port, 400kHz speed
  byte ledBrightness = 60; //Options: 0=Off to 255=50mA
  byte sampleAverage = 1; //Options: 1, 2, 4, 8, 16, 32
  byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green
  byte sampleRate = 1000; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200
  int pulseWidth = 69; //Options: 69, 118, 215, 411
  int adcRange = 2048; //Options: 2048, 4096, 8192, 16384

  particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth, adc

  //--------------------------------------------------------------------------------------
}

// funcion toma de datos principal
void timerfcn() {

  if (flgBuff == 1) {
    analogRead(A0); //se lee primero para que no haya fallos en la lectura
    buffer1ECG[cont] = analogRead(A0); //lectura pin A0 analogico
    analogRead(A1);
    buffer1EMG[cont] = analogRead(A1);
    analogRead(A2);
    buffer1EDA[cont] = analogRead(A2);
    analogRead(A3);
    buffer1PZT[cont] = analogRead(A3);
    analogRead(A5);
    buffer1ACCx[cont] = analogRead(A5);
    analogRead(A6);
    buffer1ACCy[cont] = analogRead(A6);
    analogRead(A7);
    buffer1ACCz[cont] = analogRead(A7);
    cont++;
    if (cont == 200) {
      //leer el resto de sensores
      analogRead(A4);
      int lm_sensor = analogRead(A4); //lectura sensor lm35
      t_pil = lm_sensor * 330 / 1023.0; // calculo temperatura piloto

      cont = 0;
      flgTx = 1;
      flgBuff = 2;
    }
  } else {
    analogRead(A0);
    buffer2ECG[cont] = analogRead(A0);
    analogRead(A1);
    buffer2EMG[cont] = analogRead(A1);
    analogRead(A2);
    buffer2EDA[cont] = analogRead(A2);
    analogRead(A3);
    buffer2PZT[cont] = analogRead(A3);
    analogRead(A5);
    buffer2ACCx[cont] = analogRead(A5);
    analogRead(A6);
    buffer2ACCy[cont] = analogRead(A6);
    analogRead(A7);
    buffer2ACCz[cont] = analogRead(A7);

    cont++;
    if (cont == 200) {
      //leer el resto de sensores
      analogRead(A4);
      int lm_sensor = analogRead(A4); //lectura sensor lm35
      t_pil = lm_sensor * 330 / 1023.0;
```

```
      cont = 0;
      flgTx = 2;
      flgBuff = 1;
    }
  }
}

funcion toma de datos para calibrar ACC
void calACC() {

  if (ACCflgBuff == 1) {
    analogRead(A5);
    bff1calACCx[ACCcont] = analogRead(A5);
    analogRead(A6);
    bff1calACCy[ACCcont] = analogRead(A6);
    analogRead(A7);
    bff1calACCz[ACCcont] = analogRead(A7);
    ACCcont++;
    if (ACCcont == 200) {
      ACCcont = 0;
      ACCflgTx = 1;
      ACCflgBuff = 2;

    }
  } else {
    analogRead(A5);
    bff2calACCx[ACCcont] = analogRead(A5);
    analogRead(A6);
    bff2calACCy[ACCcont] = analogRead(A6);
    analogRead(A7);
    bff2calACCz[ACCcont] = analogRead(A7);
    ACCcont++;
    if (ACCcont == 200) {
      ACCcont = 0;
      ACCflgTx = 2;
      ACCflgBuff = 1;

    }
  }
}


void loop() {
  if (Serial1.available() > 0) {

    // Comprobamos si ha llegado algún dato

    RecData = Serial1.read(); // Se lee el dato recibido

    if (RecData == 170) { // Si el dato recibido es 170 se realiza la medida

      Timer3.start(5000); // Interrupcion cada 5ms
    }

    if (RecData == 100) {

      Timer3.stop();
    }
  }

  //envio de datos a traves del puerto serie bluetooth
  if (flgTx) {

    if ( flgTx == 1) {

      for (i = 0; i < 200; i++) {
        Serial1.write(buffer1ECG[i] / 256);
        Serial1.write(buffer1ECG[i] % 256);
      }
```

```
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer1EMG[i] / 256);
      Serial1.write(buffer1EMG[i] % 256);
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer1EDA[i] / 256);
      Serial1.write(buffer1EDA[i] % 256);
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer1PZT[i] / 256);
      Serial1.write(buffer1PZT[i] % 256);
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer1ACCx[i] / 256);
      Serial1.write(buffer1ACCx[i] % 256);
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer1ACCy[i] / 256);
      Serial1.write(buffer1ACCy[i] % 256);
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer1ACCz[i] / 256);
      Serial1.write(buffer1ACCz[i] % 256);
    }

    //Toma de datos y calculo de SPO2-----------------------------------------------
    for (byte i = 0 ; i < bufferLength ; i++)
    {
      while (particleSensor.available() == false) //do we have new data?
        particleSensor.check(); //Check the sensor for new data

      redBuffer[i] = particleSensor.getRed();
      irBuffer[i] = particleSensor.getIR();
      particleSensor.nextSample(); //We're finished with this sample so move to next sa
    }

    //calculate heart rate and SpO2
    maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2, &v

    if (validSPO2 == 1) {
      valSPO2 = spo2;
    } else {
      valSPO2 = 0;
    }
    //Final toma de datos de SPO2---------------------------------------------------

    // todas las demas variables
    byte* t_pilVect = (byte*)&t_pil;
    Serial1.write(t_pilVect, sizeof(t_pilVect));
    Serial1.write(valSPO2);
  }

  if ( flgTx == 2) {

    for (i = 0; i < 200; i++) {
      Serial1.write(buffer2ECG[i] / 256);
      Serial1.write(buffer2ECG[i] % 256);
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer2EMG[i] / 256);
      Serial1.write(buffer2EMG[i] % 256);
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer2EDA[i] / 256);
      Serial1.write(buffer2EDA[i] % 256);
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer2PZT[i] / 256);
      Serial1.write(buffer2PZT[i] % 256);
```

```
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer2ACCx[i] / 256);
      Serial1.write(buffer2ACCx[i] % 256);
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer2ACCy[i] / 256);
      Serial1.write(buffer2ACCy[i] % 256);
    }
    for (i = 0; i < 200; i++) {
      Serial1.write(buffer2ACCz[i] / 256);
      Serial1.write(buffer2ACCz[i] % 256);
    }


    //Toma de datos y calculo de SPO2-----------------------------------------------
    for (byte i = 0 ; i < bufferLength ; i++)
    {
      while (particleSensor.available() == false) //do we have new data?
        particleSensor.check(); //Check the sensor for new data

      redBuffer[i] = particleSensor.getRed();
      irBuffer[i] = particleSensor.getIR();
      particleSensor.nextSample(); //We're finished with this sample so move to next sa
    }

    //calculate heart rate and SpO2
    maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2, &v

    if (validSPO2 == 1) {

      valSPO2 = spo2;
    } else {
      valSPO2 = 0;
    }
    //Final toma de datos de SPO2---------------------------------------------------

    // todas las demas variables
    byte* t_pilVect = (byte*)&t_pil;
    Serial1.write(t_pilVect, sizeof(t_pilVect));
    Serial1.write(valSPO2);
  }
  flgTx = 0;
}

//A partir de aqui es la calibracion del ACC-------------------------------------
if (Serial2.available() > 0) {


  // Comprobamos si ha llegado algún dato

  ACCRecData = Serial2.read(); // Se lee el dato recibido

  if (ACCRecData == 93) { // Si el dato recibido es 93 se realiza la medida

    Timer4.start(5000); // interrupcion cada 5ms
  }

  if (ACCRecData == 27) {

    Timer4.stop();
  }
}

if (ACCflgTx) {

  if ( ACCflgTx == 1) {

    for (j = 0; j < 200; j++) {
```

```
        Serial2.write(bff1calACCx[j] / 256);
        Serial2.write(bff1calACCx[j] % 256);
      }
      for (j = 0; j < 200; j++) {
        Serial2.write(bff1calACCy[j] / 256);
        Serial2.write(bff1calACCy[j] % 256);
      }
      for (j = 0; j < 200; j++) {
        Serial2.write(bff1calACCz[j] / 256);
        Serial2.write(bff1calACCz[j] % 256);
      }
    }

    if ( ACCflgTx == 2) {

      for (j = 0; j < 200; j++) {
        Serial2.write(bff2calACCx[j] / 256);
        Serial2.write(bff2calACCx[j] % 256);
      }
      for (j = 0; j < 200; j++) {
        Serial2.write(bff2calACCy[j] / 256);
        Serial2.write(bff2calACCy[j] % 256);
      }
      for (j = 0; j < 200; j++) {
        Serial2.write(bff2calACCz[j] / 256);
        Serial2.write(bff2calACCz[j] % 256);
      }
    }

    ACCflgTx = 0;
  }
}
```

```
function varargout = DUEappTFM_v7(varargin)
% DUEAPPTFM_V7 MATLAB code for DUEappTFM_v7.fig
%      DUEAPPTFM_V7, by itself, creates a new DUEAPPTFM_V7 or raises the existing
%      singleton*.
%
%      H = DUEAPPTFM_V7 returns the handle to a new DUEAPPTFM_V7 or the handle to
%      the existing singleton*.
%
%      DUEAPPTFM_V7('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DUEAPPTFM_V7.M with the given input arguments.
%
%      DUEAPPTFM_V7('Property','Value',...) creates a new DUEAPPTFM_V7 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before DUEappTFM_v7_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to DUEappTFM_v7_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help DUEappTFM_v7

% Last Modified by GUIDE v2.5 15-Nov-2020 22:56:08

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @DUEappTFM_v7_OpeningFcn, ...
                   'gui_OutputFcn',  @DUEappTFM_v7_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before DUEappTFM_v7 is made visible.
function DUEappTFM_v7_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to DUEappTFM_v7 (see VARARGIN)

% Choose default command line output for DUEappTFM_v7
```

```
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes DUEappTFM_v7 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

global SerCOM j Cxmin Cxmax Cymin Cymax Czmin Czmax ACCSerCOM q


Puertos_Activos=instrfind; % Lee los puertos activos
if isempty(Puertos_Activos)==0 % Comprueba si hay puertos activos
    fclose(Puertos_Activos); % Cierra los puertos activos
    delete(Puertos_Activos) % Borra la variable Puertos_Activos
    clear Puertos_Activos % Destruye la variable Puertos_Activos
end

Cxmin=341; %nivel digital ADC = 2^10 (10 bit); [0 - 1023]
Cxmax=682;
Cymin=341;
Cymax=682;
Czmin=341;
Czmax=682;

j=0;
SerCOM = serial('COM4'); % Se establece el COM asociado al objeto serial
SerCOM.InputBufferSize = 2805; %tamaño del buffer en bytes que que se recibe
SerCOM.BaudRate=115200; % Se configura la velocidad a 115200 baudios
SerCOM.DataBits=8; % Se establece 8 bits de datos
SerCOM.Parity='none'; % Se configura el puerto serie sin paridad
SerCOM.StopBits=1; % Se establece 1 bit de STOP
SerCOM.FlowControl='none'; % Se configura el puerto serie sin paridad
SerCOM.BytesAvailableFcnCount=2805; % Se configura en n de bytes que debe haber en el
% buffer de RX para empezar el evento Rx_Callback
SerCOM.BytesAvailableFcnMode='byte';
SerCOM.BytesAvailableFcn={@Rx_Callback,handles};

fopen(SerCOM); % Se abre el puerto serie para el bluetooth

% serial com para calibrar el ACC----------------------------------------
q=0;
ACCSerCOM = serial('COM7'); % Se establece el COM asociado al objeto serial
ACCSerCOM.InputBufferSize = 1200;
ACCSerCOM.BaudRate=115200; % Se configura la velocidad a 115200 baudios
ACCSerCOM.DataBits=8; % Se establece 8 bits de datos
ACCSerCOM.Parity='none'; % Se configura el puerto serie sin paridad
ACCSerCOM.StopBits=1; % Se establece 1 bit de STOP
ACCSerCOM.FlowControl='none'; % Se configura el puerto serie sin paridad
ACCSerCOM.BytesAvailableFcnCount=1200; % Se configura en n de bytes que debe haber en↙
el
% buffer de RX para empezar el evento ACCcalRx_Callback
ACCSerCOM.BytesAvailableFcnMode='byte';
ACCSerCOM.BytesAvailableFcn={@ACCcalRx_Callback,handles};
```

```
fopen(ACCSerCOM); % Se abre el puerto serie para el bluetooth


% --- Outputs from this function are returned to the command line.
function varargout = DUEappTFM_v7_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in btnEXIT.
function btnEXIT_Callback(hObject, eventdata, handles)
% hObject    handle to btnEXIT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global SerCOM ACCSerCOM

if handles.btnEXIT.Value==1
    opc=questdlg('¿Desea salir del programa?','Salir','Si','No','No');
    if strcmp(opc,'No')
        return;

    else
        fwrite(SerCOM,100); %se envia el byte de fin de medida
        fclose(SerCOM);
        delete(SerCOM);
        clear SerCOM;

        fwrite(ACCSerCOM,27);
        fclose(ACCSerCOM);
        delete(ACCSerCOM);
        clear ACCSerCOM;

        close all
        clear
        clc
    end
end

% --- Executes on button press in btnSTART.
function btnSTART_Callback(hObject, eventdata, handles)
% hObject    handle to btnSTART (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of btnSTART

global SerCOM Cont x yTP ECG ECGval EMG EMGval EDA EDAval PZT PZTval ACCx ACCxval ACCy↙
ACCyval ACCz ACCzval...
    xECG xEMG xEDA xPZT xACCx xACCy xACCz
```

```
if hObject.Value==1

    fwrite(SerCOM,170); %se envia el byte de inicio de medida

    hObject.String='Parar'; % Se cambia el texto del boton
    hObject.BackgroundColor='Yellow'; % Se cambia el color de fondo del boton
    x=[]; % Se incializa el array de indices
    ECG=[];
    ECGval=[];
    EMG=[];
    EMGval=[];
    EDA=[];
    EDAval=[];
    PZT=[];
    PZTval=[];
    ACCx=[];
    ACCxval=[];
    ACCy=[];
    ACCyval=[];
    ACCz=[];
    ACCzval=[];

    xECG=[];
    xEMG=[];
    xEDA=[];
    xPZT=[];
    xACCx=[];
    xACCy=[];
    xACCz=[];

    yTP=0;
    Cont=1;


else
    fwrite(SerCOM,100); %se envia el byte de fin de medida

    hObject.String='Iniciar'; % Se cambia el texto del boton
    hObject.BackgroundColor='Green'; % Se cambia el color de fondo del boton
end

function Rx_Callback(hObject, eventdata, handles, varargin)
global SerCOM x j Cont ECG ECGH ECGL ECGval EMG EMGH EMGL EMGval EDA EDAH EDAL↙
EDAval...
    PZT PZTH PZTL PZTval ACCx ACCxH ACCxL ACCxval ACCy ACCyH ACCyL ACCyval...
    ACCz ACCzH ACCzL ACCzval TP yTP spo2Val ySPO2 plotECG...
    Cxmin Cxmax Cymin Cymax Czmin Czmax plotEMG plotEDA plotPZT plotACC...
    xECG xEMG xEDA xPZT xACCx xACCy xACCz% Declaracion variables globales
a = 1;

% SerCOM.BytesAvailable
Datos=fread(SerCOM,2805); % Se leen los 2805 bytes recibidos

DatosECG = uint16(Datos(1:400)); %convertimos a uint16 para hacer el cast
```

```
DatosEMG = uint16(Datos(401:800));
DatosEDA = uint16(Datos(801:1200));
DatosPZT = uint16(Datos(1201:1600));

DatosACCx = uint16(Datos(1601:2000));
DatosACCy = uint16(Datos(2001:2400));
DatosACCz = uint16(Datos(2401:2800));

DatosTP = uint8(Datos(2801:2804));
DatosSPO2 = uint8(Datos(2805));

for i=1:200
    ECGH=DatosECG(2*i-1);
    ECGL=DatosECG(2*i);
    ECG(j+i)=256*ECGH+ECGL;
    ECGval(j+i)=((((ECG(j+i)/2^10)-0.5)*3.3)/1100)*1000;
    xECG(j+i)=length(ECGval);
end
for i=1:200
    EMGH=DatosEMG(2*i-1);
    EMGL=DatosEMG(2*i);
    EMG(j+i)=256*EMGH+EMGL;
    EMGval(j+i)=((((EMG(j+i)/2^10)-0.5)*3.3)/1009)*1000;
    xEMG(j+i)=length(EMGval);
end
for i=1:200
    EDAH=DatosEDA(2*i-1);
    EDAL=DatosEDA(2*i);
    EDA(j+i)=256*EDAH+EDAL;
    EDAval(j+i)=1/(1-(EDA(j+i)/2^10));
    xEDA(j+i)=length(EDAval);
end
for i=1:200
    PZTH=DatosPZT(2*i-1);
    PZTL=DatosPZT(2*i);
    PZT(j+i)=256*PZTH+PZTL;
    PZTval(j+i)=((PZT(j+i)/2^10)-0.5)*100;
    xPZT(j+i)=length(PZTval);
end

for i=1:200
    ACCxH=DatosACCx(2*i-1);
    ACCxL=DatosACCx(2*i);
    ACCx(j+i)=256*ACCxH+ACCxL;
    ACCxval(j+i)=(((ACCx(j+i)-Cxmin)/(Cxmax-Cxmin))*2)-1;
    xACCx(j+i)=length(ACCxval);
end
for i=1:200
    ACCyH=DatosACCy(2*i-1);
    ACCyL=DatosACCy(2*i);
    ACCy(j+i)=256*ACCyH+ACCyL;
    ACCyval(j+i)=(((ACCy(j+i)-Cymin)/(Cymax-Cymin))*2)-1;
    xACCy(j+i)=length(ACCyval);
end
for i=1:200
```

```
    ACCzH=DatosACCz(2*i-1);
    ACCzL=DatosACCz(2*i);
    ACCz(j+i)=256*ACCzH+ACCzL;
    ACCzval(j+i)=(((ACCz(j+i)-Czmin)/(Czmax-Czmin))*2)-1;
    xACCz(j+i)=length(ACCzval);
end

%Filtrado de señales con media movil
windowSize1 = 5; %5 por defecto funciona bien
b1 = (1/windowSize1)*ones(1,windowSize1);
ECGval = filter(b1,a,ECGval);

windowSize2 = 5;
b2 = (1/windowSize2)*ones(1,windowSize2);
EMGval = filter(b2,a,EMGval);

windowSize3 = 5;
b3 = (1/windowSize3)*ones(1,windowSize3);
EDAval = filter(b3,a,EDAval);

windowSize4 = 5;
b4 = (1/windowSize4)*ones(1,windowSize4);
PZTval = filter(b4,a,PZTval);

windowSize5 = 5;
b5 = (1/windowSize5)*ones(1,windowSize5);
ACCxval = filter(b5,a,ACCxval);
ACCyval = filter(b5,a,ACCyval);
ACCzval = filter(b5,a,ACCzval);


j=j+200;

TP = typecast(DatosTP(1:4), 'single'); %temperatura piloto
spo2Val = DatosSPO2(1); %valor SpO2

x(Cont)=Cont;
yTP(Cont)=TP;
ySPO2(Cont)=spo2Val;

if length(ECGval) >= 1000
    plotECG = plot(handles.pltECG,xECG(length(xECG)-999:length(xECG)),ECGval(length↙
(ECGval)-999:length(ECGval)));

else
    plotECG = plot(handles.pltECG,xECG,ECGval);

end

if length(EMGval) >= 1000
    plotEMG = plot(handles.pltEMG,xEMG(length(xEMG)-999:length(xEMG)),EMGval(length↙
(EMGval)-999:length(EMGval)));

else
    plotEMG = plot(handles.pltEMG,xEMG,EMGval);
```

```
end

if length(EDAval) >= 1000
    plotEDA = plot(handles.pltEDA,xEDA(length(xEDA)-999:length(xEDA)),EDAval(length↙
(EDAval)-999:length(EDAval)));

else
    plotEDA = plot(handles.pltEDA,xEDA,EDAval);

end

if length(PZTval) >= 1000
    plotPZT = plot(handles.pltPZT,xPZT(length(xPZT)-999:length(xPZT)),PZTval(length↙
(PZTval)-999:length(PZTval)));

else
    plotPZT = plot(handles.pltPZT,xPZT,PZTval);

end

if length(ACCxval) >= 1000 || length(ACCyval) >= 1000 || length(ACCzval) >= 1000

    plotACC = plot(handles.pltACC,xACCx(length(xACCx)-999:length(xACCx)),ACCxval↙
(length(ACCxval)-999:length(ACCxval)),'r',...
        xACCy(length(xACCy)-999:length(xACCy)),ACCyval(length(ACCyval)-999:length↙
(ACCyval)),'g',...
        xACCz(length(xACCz)-999:length(xACCz)),ACCzval(length(ACCzval)-999:length↙
(ACCzval)),'b');

else
    plotACC = plot(handles.pltACC,xACCx,ACCxval,'r',xACCy,ACCyval,'g',xACCz,↙
ACCzval,'b');

end


handles.txtTpil.String = string(TP);
handles.txtSPO2.String = string(spo2Val);


Cont=Cont+1;


% --- Executes on button press in btnCalACC.
function btnCalACC_Callback(hObject, eventdata, handles)
% hObject    handle to btnCalACC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of btnCalACC

global ACCSerCOM calACCx  calACCy  calACCz
```

```
if hObject.Value==1

    fwrite(ACCSerCOM,93); %se envia el byte de inicio de medida

    hObject.String='Parar Calibración ACC'; % Se cambia el texto del boton
    hObject.BackgroundColor='Yellow'; % Se cambia el color de fondo del boton

    calACCx=[];
    calACCy=[];
    calACCz=[];

else
    fwrite(ACCSerCOM,27); %se envia el byte de fin de medida

    hObject.String='Iniciar Calibración ACC'; % Se cambia el texto del boton
    hObject.BackgroundColor='Green'; % Se cambia el color de fondo del boton
end


function ACCcalRx_Callback(hObject, eventdata, handles, varargin)
global ACCSerCOM calACCx calACCy calACCz calACCxH calACCxL calACCyH calACCyL...
    calACCzH calACCzL q Cxmin Cxmax Cymin Cymax Czmin Czmax
a=1;

ACCSerCOM.BytesAvailable
CalDatos=fread(ACCSerCOM,1200); % Se leen los 1200 bytes recibidos

%convertimos a uint16 para hacer el cast
calDatosACCx = uint16(CalDatos(1:400));
calDatosACCy = uint16(CalDatos(401:800));
calDatosACCz = uint16(CalDatos(801:1200));

for i=1:200
    calACCxH=calDatosACCx(2*i-1);
    calACCxL=calDatosACCx(2*i);
    calACCx(q+i)=256*calACCxH+calACCxL;
end
for i=1:200
    calACCyH=calDatosACCy(2*i-1);
    calACCyL=calDatosACCy(2*i);
    calACCy(q+i)=256*calACCyH+calACCyL;
end
for i=1:200
    calACCzH=calDatosACCz(2*i-1);
    calACCzL=calDatosACCz(2*i);
    calACCz(q+i)=256*calACCzH+calACCzL;
end

windowSizeCal = 5;
bCal = (1/windowSizeCal)*ones(1,windowSizeCal);

calACCx = filter(bCal,a,calACCx);
calACCy = filter(bCal,a,calACCy);
calACCz = filter(bCal,a,calACCz);
```

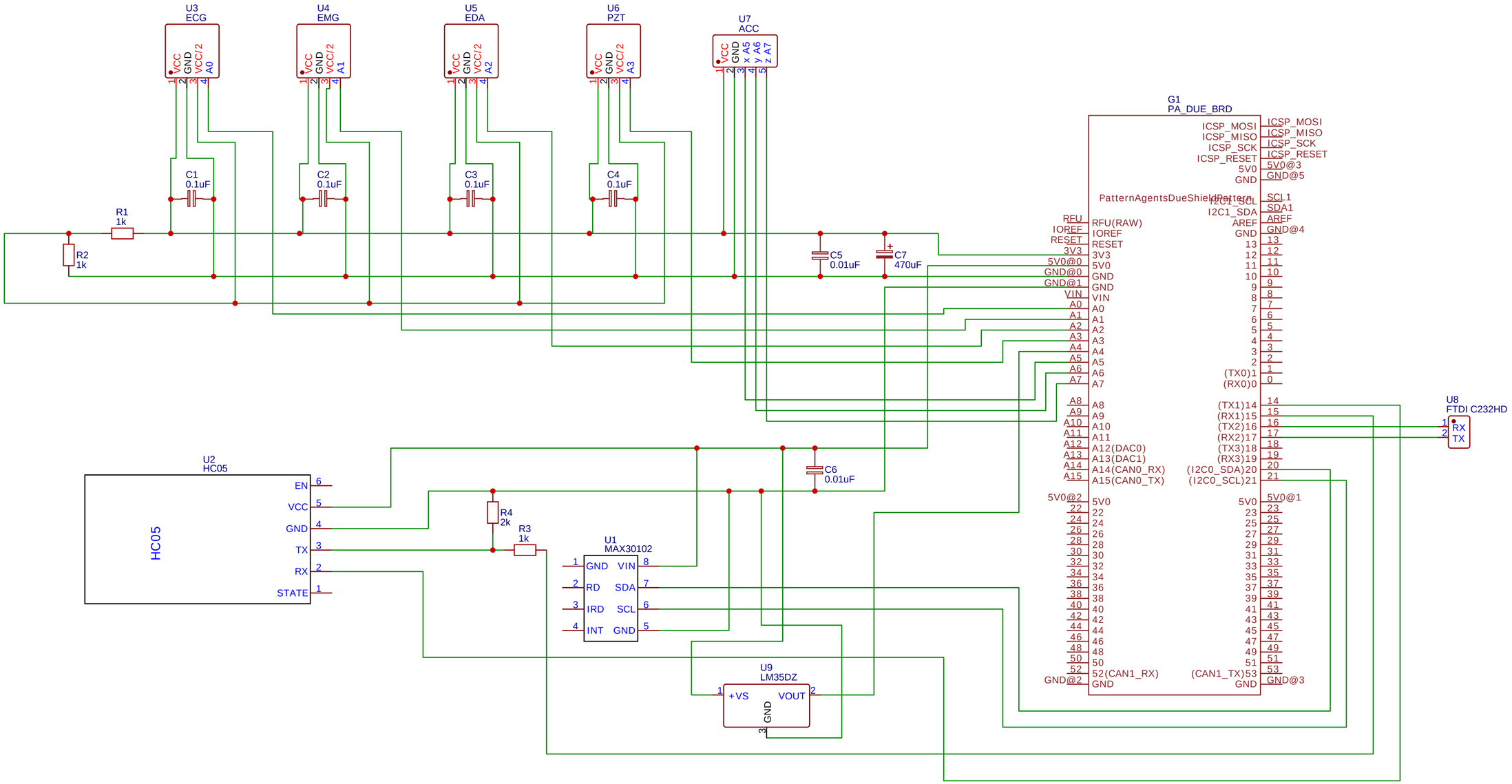
```
q=q+200;

%aqui calcular maximos y minimos
if q > 2000
    Cxmin = min(calACCx(1001:length(calACCx)-1000));
    Cxmax = max(calACCx(1001:length(calACCx)-1000));

    Cymin = min(calACCy(1001:length(calACCy)-1000));
    Cymax = max(calACCy(1001:length(calACCy)-1000));

    Czmin = min(calACCz(1001:length(calACCz)-1000));
    Czmax = max(calACCz(1001:length(calACCz)-1000));
else
    Cxmin = min(calACCx);
    Cxmax = max(calACCx);

    Cymin = min(calACCy);
    Cymax = max(calACCy);

    Czmin = min(calACCz);
    Czmax = max(calACCz);
end

%aqui se muestran los maximos y los minimos de ACC
handles.txtCxMIN.String = string(Cxmin);
handles.txtCxMAX.String = string(Cxmax);

handles.txtCyMIN.String = string(Cymin);
handles.txtCyMAX.String = string(Cymax);

handles.txtCzMIN.String = string(Czmin);
handles.txtCzMAX.String = string(Czmax);

% --- Executes on button press in btnCleanACC.
function btnCleanACC_Callback(hObject, eventdata, handles)
% hObject    handle to btnCleanACC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global ACCSerCOM q Cxmin Cxmax Cymin Cymax Czmin Czmax

if handles.btnCleanACC.Value==1
    opc=questdlg('Data adquisition still runnig,¿Do you want to exit↙
now?','EXIT','Yes','No','No');
    if strcmp(opc,'No')
        return;

    else

        q=0;

        clear calACCx calACCy calACCz calACCxH calACCxL calACCyH calACCyL...
            calACCzH calACCzL
        Cxmin = 341;
        Cxmax = 682;
```

```
        Cymin = 341;
        Cymax = 682;
        Czmin = 341;
        Czmax = 682;

        %aqui se muestran los maximos y los minimos de ACC
        handles.txtCxMIN.String = string(Cxmin);
        handles.txtCxMAX.String = string(Cxmax);

        handles.txtCyMIN.String = string(Cymin);
        handles.txtCyMAX.String = string(Cymax);

        handles.txtCzMIN.String = string(Czmin);
        handles.txtCzMAX.String = string(Czmax);

        fwrite(ACCSerCOM,27); %se envia el byte de fin de medida


    end
end


% --- Executes on button press in btnExport.
function btnExport_Callback(hObject, eventdata, handles)
% hObject    handle to btnExport (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global x ECGval EMGval EDAval PZTval ACCxval ACCyval ACCzval yTP ySPO2...
    xECG xEMG xEDA xPZT xACCx xACCy xACCz% Declaracion variables globales

if handles.btnSTART.Value==0 && handles.btnExport.Value==1
    ECGval = ECGval.';
    EMGval = EMGval.';
    EDAval = EDAval.';
    PZTval = PZTval.';
    ACCxval = ACCxval.';
    ACCyval = ACCyval.';
    ACCzval = ACCzval.';
    xECG = xECG.';
    xEMG = xEMG.';
    xEDA = xEDA.';
    xPZT = xPZT.';
    xACCx = xACCx.';
    xACCy = xACCy.';
    xACCz = xACCz.';

    x = x.';
    yTP = yTP.';
    ySPO2 = ySPO2.';

    pathfile = handles.txtPathExport.String;
    if ~exist(pathfile, 'dir')
        mkdir(pathfile);
    end
    excelfilename = strcat('Reporte_',handles.txtExport.String,'.xlsx');
```

```
    filename = fullfile(pathfile, excelfilename);

    header1 = {'x','ECG [mV]','EMG [mV]','EDA [uS]','PZT [%]','ACCx [g]',...
        'ACCy [g]','ACCz [g]'};
    header2 = {'x','TP [°C]','SPO2 [%]'};



    xlswrite(filename,header1,'Data1','A1');
    xlswrite(filename,xECG,'Data1','A2');
    xlswrite(filename,ECGval,'Data1','B2');
    xlswrite(filename,EMGval,'Data1','C2');
    xlswrite(filename,EDAval,'Data1','D2');
    xlswrite(filename,PZTval,'Data1','E2');
    xlswrite(filename,ACCxval,'Data1','F2');
    xlswrite(filename,ACCyval,'Data1','G2');
    xlswrite(filename,ACCzval,'Data1','H2');

    xlswrite(filename,header2,'Data2','A1');
    xlswrite(filename,x,'Data2','A2');
    xlswrite(filename,yTP,'Data2','B2');
    xlswrite(filename,ySPO2,'Data2','C2');

    dlgExportExit;


end


function txtExport_Callback(hObject, eventdata, handles)
% hObject    handle to txtExport (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txtExport as text
%        str2double(get(hObject,'String')) returns contents of txtExport as a double


% --- Executes during object creation, after setting all properties.
function txtExport_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtExport (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in btnReset.
function btnReset_Callback(hObject, eventdata, handles)
% hObject    handle to btnReset (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global q x j Cont plotECG plotEMG plotEDA plotPZT plotACC
if handles.btnReset.Value==1
    clear ECG ECGH ECGL ECGval EMG EMGH EMGL EMGval EDA EDAH EDAL EDAval...
    PZT PZTH PZTL PZTval ACCx ACCxH ACCxL ACCxval ACCy ACCyH ACCyL ACCyval...
    ACCz ACCzH ACCzL ACCzval TP yTP spo2Val ySPO2...
    xECG xEMG xEDA xPZT xACCx xACCy xACCz calACCx calACCy calACCz calACCxH...
    calACCxL calACCyH calACCyL calACCzH calACCzL

j=0;
q=0;
x=0;
Cont=1;

delete(plotECG);
delete(plotEMG);
delete(plotEDA);
delete(plotPZT);
delete(plotACC);


handles.txtTpil.String = "---";
handles.txtSPO2.String = "---";
end



function txtPathExport_Callback(hObject, eventdata, handles)
% hObject    handle to txtPathExport (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txtPathExport as text
%        str2double(get(hObject,'String')) returns contents of txtPathExport as a↙
double


% --- Executes during object creation, after setting all properties.
function txtPathExport_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txtPathExport (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get↙
(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function dlgExportExit
    d = dialog('Position',[300 300 250 150],'Name','Reporte de sesión');

    txt = uicontrol('Parent',d,...
               'Style','text',...
               'Position',[20 80 210 40],...
               'String','Datos exportados con éxito');

    btn = uicontrol('Parent',d,...
               'Position',[85 20 70 25],...
               'String','Cerrar',...
               'Callback','delete(gcf)');
end
```



TITLE: Sistema de adquisicion de datos

REV: 1.0

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

INSTITUTO DE DISEÑO Y FABRICACIÓN

Company: IDF - UPV

Sheet: 1/1

Date: 2020-11-20 Drawn By: Oscar Bori Fabra