



*Escuela Superior de  
Ingeniería y Tecnología*

# Trabajo de Fin de Grado

Grado en Ingeniería Informática

---

## Desarrollo de una interfaz para un equipo médico

*Development of an interface for a medical equipment*

Jorge Sierra Acosta

---

La Laguna, 1 de agosto de 2018

D<sup>a</sup>. **Vanesa Muñoz Cruz**, con N.I.F. 78.698.687-R profesora Contratada Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D<sup>a</sup>. **Estefanía Hernández Martín**, con N.I.F. 78.637.430-Q Investigadora adscrita al Departamento de Ciencias Médicas Básicas de la Universidad de La Laguna, como cotutor

## C E R T I F I C A (N)

Que la presente memoria titulada:

*“Desarrollo de una interfaz para un equipo médico”*

ha sido realizada bajo su dirección por D. **Jorge Sierra Acosta**,  
con N.I.F. 79.075.663-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos  
firman la presente en La Laguna a 1 de agosto de 2018

# Agradecimientos

En primer lugar, agradecer a mi tutora Vanesa Muñoz Cruz por el esfuerzo realizado y la gran e inestimable ayuda aportada, sin la cual hubiera sido imposible completar el TFG.

Por otro lado, agradecer también a Estefanía Hernández Martín por la gran ayuda y la información con la que me ha apoyado además de la oportunidad de realizar este trabajo.

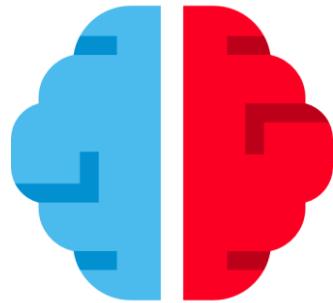
Por último, agradecer a mis compañeros y familiares y a los profesores que se han encargado de guiar el estudio de mi carrera de Ingeniería Informática.

# Licencia



© Esta obra está bajo una [licencia](#) de Creative Commons Reconocimiento-NoComercial 4.0 Internacional

Usted es libre de compartir, copiar y redistribuir el material en cualquier medio o formato y adaptar, remezclar, transformar y crear a partir del material. Bajo las condiciones de reconocimiento adecuado de la autoría, proporcionar un enlace a la licencia e indicar si se han realizado cambios y uso no comercial.



© El icono de la aplicación está bajo una [licencia](#) de Creative Commons Reconocimiento 3.0 Unported

El icono de la aplicación presentada fue realizado por [Freepik](#) desde [www.flaticon.com](http://www.flaticon.com) y no ha recibido cambios. Usted es libre de compartir, copiar y redistribuir el material en cualquier medio o formato y adaptar, remezclar, transformar y crear a partir del material para cualquier finalidad, incluso comercial. Bajo las condiciones de reconocimiento adecuado de la autoría, proporcionar un enlace a la licencia e indicar si se han realizado cambios.

Los datos de los experimentos para la creación de las imágenes y las gráficas que se muestran en la memoria, han sido cedidos por el laboratorio de Neuroquímica y Neuroimagen de la Universidad de La Laguna.

## Resumen

*El objetivo de este trabajo ha sido el desarrollo de una aplicación multiplataforma capaz de recibir las imágenes obtenidas por un hardware especial de espectroscopía de infrarrojo cercano del grupo de Neuroquímica y Neuroimagen de la ULL y desarrollar una interfaz en la que poder visualizar y tratar dichas imágenes, realizando si es posible un análisis estadístico.*

*Estas imágenes representan en un mapa bidimensional los cambios en la oxigenación de los tejidos cerebrales indicando una mayor o menor actividad relativa al punto de referencia, aunque la misma técnica base puede ser empleada en cerebros humanos, en este caso concreto las imágenes representan capturas de cambios hemodinámicos de un cerebro de rata común.*

*Actualmente existen una serie de scripts escritos para Matlab (software matemático) que presentan una solución temporal al problema, pero se busca una solución más definitiva con este proyecto, presentando una interfaz más intuitiva y sencilla que agrupe todos los controles eliminando la necesidad de depender de la creación y edición continua de scripts y del propio Matlab que además requiere una licencia no gratuita.*

*También se busca explorar la posibilidad de sincronizar la nueva aplicación con un Arduino que debe enviar al sistema de captura de imágenes una señal indicando en qué punto es completada la captura de las imágenes que formaran el basal, esta es, la imagen de referencia para medir las activaciones cerebrales siguientes.*

**Palabras clave:** activación cerebral, cambios hemodinámicos, NIRS, interfaz, tratamiento de imágenes

## **Abstract**

*The purpose of this project is to development a multiplatform application which will be receiving images taken by a near infrared spectroscopy hardware by the group of Neuroimaging and Neurochemistry of the ULL and build an interface in which the user may visualize and process such images, applying if possible a statistical analysis.*

*These images represent a bidimensional map of changes in the oxygenation of brain tissues indicating an increment or decrease of the brain activity in that region relative to the initial reference point. Even though the same technique can be applied to human brains, in this specific scenario the images were taken from a common rat one.*

*Currently there are a variety of scripts codded for Matlab (mathematical software) which serve as a temporal solution to the problem to which this project pretends to offer a more robust one, creating a more intuitive and simpler interface grouping all functionality and removing the need to depend on the constant creation and modification of those scripts and the non-free licence needed for such software.*

*Another objective is to explore the possibility to synchronize the new application with an Arduino in charge of sending a signal to the image capturing hardware indicating when the capturing of the basal is completed, this is, the reference image used to measure the different brain activations.*

**Keywords:** brain activation, hemodynamic changes, NIRS, interface, image processing

# Índice general

<b>Capítulo 1 Introducción.....</b>	<b>1</b>
1.1 Activación cerebral.....	1
1.2 Antecedentes.....	1
1.3 Objetivos del proyecto .....	2
1.4 Estado inicial del proyecto.....	3
<b>Capítulo 2 Plan de trabajo y tecnologías utilizadas .....</b>	<b>5</b>
2.1 Plan de trabajo.....	5
2.2 Cronograma y tareas.....	6
2.3 Tecnologías utilizadas .....	7
<b>Capítulo 3 Módulo de back-end o procesamiento.....</b>	<b>11</b>
3.1 Introducción.....	11
3.2 Estructura del archivo del experimento.....	11
3.3 Sistema de comunicación .....	13
3.4 Estructura del módulo .....	15
3.5 Procesamiento.....	16
3.6 Filtros.....	18
3.7 Multithreading.....	19
3.8 Multiplataforma.....	20
<b>Capítulo 4 Módulo front-end o interfaz.....</b>	<b>21</b>
4.1 Introducción.....	21
4.2 Esquema de colores y logo .....	22

4.3	Diseño de la interfaz .....	23
4.4	Guías de diseño WCAG .....	26
4.5	Visor de imágenes.....	28
4.6	Estructura de programa de la interfaz .....	28
<b>Capítulo 5</b>	<b>Problemas y trabajo en desarrollo.....</b>	<b>30</b>
5.1	Problemas y dificultades .....	30
5.2	Análisis estadístico.....	30
5.3	Documentación .....	32
<b>Capítulo 6</b>	<b>Conclusiones y líneas futuras.....</b>	<b>33</b>
6.1	Conclusiones .....	33
6.2	Líneas futuras .....	34
<b>Capítulo 7</b>	<b>Summary and Conclusions.....</b>	<b>35</b>
7.1	Summary .....	35
7.2	Conclusions.....	35
<b>Capítulo 8</b>	<b>Presupuesto.....</b>	<b>37</b>
8.1	Coste del proyecto .....	37
<b>Capítulo 9</b>	<b>Bibliografía.....</b>	<b>38</b>
<b>Capítulo 10</b>	<b>Anexos .....</b>	<b>43</b>
10.1	<i>Frames</i> de calibración.....	43
10.2	Evolución de la interfaz.....	44
10.3	Ejemplos de imágenes de saturación.....	45

# Índice de figuras

Figura 1: Picos de absorción de la oxi- y desoxihemoglobina.....	2
Figura 2: Equipo NIRS del grupo de Neuroquímica y Neuroimagen de la ULL .....	3
Figura 3: Máscara sobre dos imágenes en <i>Matlab</i> .....	5
Figura 4: Máscara sobre dos imágenes en <i>C++</i> .....	6
Figura 5: Estructura de la aplicación .....	11
Figura 6: Flujo de la información.....	13
Figura 7: Ejemplo de uso de una interfaz con <i>QWebChannel</i> en <i>QT</i> .....	14
Figura 8: Diagrama simplificado de clases del módulo de procesamiento. Las clases con la inicial <i>Q</i> pertenecen al <i>framework</i> de <i>QT</i> .....	16
Figura 9: Imagen coloreada de saturación basal (Izquierda), valores normalizados. Imagen coloreada de saturación (Derecha), en este caso se han truncado los valores entre 0.06 y 0.00.....	18
Figura 10: Cambios en la concentración de las moléculas de HbR (desoxihemoglobina) y HbO (oxihemoglobina) .....	18
Figura 11: Aplicación de un filtro paso bajo a una porción del gráfico de concentración .....	19
Figura 12: Tiempo de ejecución relativo (eje ordenadas) con respecto al número de hilos (eje de abscisas) en un sistema con un procesador de 4 núcleos para ~8000 <i>frames</i> . .....	20
Figura 13: Interfaz de la aplicación. Zona principal.....	21
Figura 14: Interfaz de la aplicación. Zona principal. Experimento cargado. ..	22
Figura 15: Logo de la aplicación. Sujeto a las normas de copyright presentadas al principio del documento.....	22
Figura 16: Esquema de colores de la aplicación.....	23
Figura 17: Interfaz de la aplicación. Zona de opciones. .....	24
Figura 18: Interfaz de la aplicación. Zona about. .....	25

Figura 19: Tarjetas de <i>load</i> y <i>filters</i> . Barra de carga en progreso y mensaje de error. (En la aplicación aparecen una sobre la otra, no en horizontal). .....	27
Figura 20: Visor de imágenes con una imagen basal. Modo normal con un área de interés seleccionada (Izquierda). Modo zoom sobre un pixel de valor 0.898 (Derecha).....	28
Figura 21: Estructura simplificada de clases de la interfaz.....	29
Figura 22: Imagen de saturación (Izquierda). Imagen de saturación filtrada con un <i>kernel</i> de tamaño $3 \times 3$ y sigma 1.0. Mismo mapa de color que el aplicado a la imagen de saturación de la figura 9.....	31
Figuras 23 y 24: Ejemplo de <i>frame dark</i> (arriba) y <i>gain</i> (abajo). Explicados en el 1.4.....	43
Figuras 25 y 26: Desarrollo de la interfaz inicial (arriba) y posterior (abajo). El desarrollo final se encuentra en el capítulo 4. ....	44
Figura 27 (Izquierda): Imagen de saturación, correspondiente al <i>frame</i> 1000.45	
Figura 28 (Derecha): Imagen de saturación, correspondiente al <i>frame</i> 2000. .45	
Figura 29 (Izquierda): Imagen de saturación, correspondiente al <i>frame</i> 3000.45	
Figura 30 (Derecha): Imagen de saturación, correspondiente al <i>frame</i> 7000. .45	

# Índice de tablas y ecuaciones

Tabla 1: Cronograma inicial.....	7
Tabla 2: Estructura del archivo .exp .....	12
Tabla 3: Presupuesto.....	37
Ecuación 1: Ley de Beer-Lambert.....	4
Ecuación 2: Concentración de hemoglobina.....	4
Ecuación 3: Cálculo de la saturación.....	17
Ecuación 4: Cálculo del basal.....	17

# Capítulo 1

## Introducción

### 1.1 Activación cerebral

El cerebro supone un reto a la investigación, por su complejidad inherente, mil millones de neuronas y aún más conexiones y por su capacidad de trabajar a diferentes escalas en el espacio y en el tiempo, así como por su gran capacidad de abstracción y conciencia de uno mismo.

Los avances en el estudio específico del cerebro suponen avances en la salud, la biología y la medicina, pero también apoyan un desarrollo paralelo a otras ciencias, como dos de los ejemplos, la informática y las matemáticas en cuanto a desarrollo de modelos matemáticos y aprendizaje automático.

Cambios en la actividad neuronal del tejido cerebral están relacionados especialmente con cambios en el flujo sanguíneo cerebral (FSC), su volumen y oxygenación de la sangre, referidos en su conjunto como *respuesta hemodinámica* [1].

El aumento o disminución de la actividad produce variaciones en la saturación de oxígeno de la hemoglobina (hemoproteína de la sangre encargada del transporte de oxígeno), esta puede estar desaturada de moléculas de oxígeno (desoxihemoglobina o *HbR*) o saturada con ellas (oxihemoglobina o *HbO*).

### 1.2 Antecedentes

Existen varios sistemas para la detección de actividad cerebral. La tecnología más conocida es la **resonancia magnética funcional** (fMRI), no invasiva, que realiza una medida indirecta de la actividad en base al flujo de sangre en las diferentes áreas del cerebro gracias a un fuerte campo magnético generado artificialmente. Debido a la lentitud relativa del flujo de sangre, posee una resolución temporal baja, sin embargo, puede proporcionar imágenes muy detalladas en la dimensión espacial. Un sistema similar, la **tomografía por emisión de positrones** (PET) realiza las medidas del flujo sanguíneo gracias a un radiofármaco administrado [2].

El **electroencefalograma** (EEG) por otro lado, permite medir directamente la actividad eléctrica creada por distintas capas del cerebro (específicamente de regiones de materia gris) usando electrodos colocados en la parte superior de la cabeza. Permite estudiar procesos que ocurren en un tiempo corto posterior al estímulo o estados mentales en largos períodos (motivación, somnolencia, ...). La **magnetoencefalografía** (MEG) captura la actividad cerebral a través de los campos magnéticos generados por la actividad neuronal, creando una buena resolución espacial y temporal [3].

Otra tecnología se basa en medir las concentraciones de la oxihemoglobina y desoxihemoglobina en diferentes regiones del cerebro de forma continua, relativamente barata y no invasiva usando la **espectroscopía de infrarrojo cercano** (NIRS). Esta puede ser usada tanto en el cerebro como en los músculos y es comúnmente utilizada en la investigación en deportes, ergonomía, rehabilitación, monitorización, funcionamiento cerebral, ... Su funcionamiento está basado en las diferencias ópticas en los picos de absorción que presentan las diferentes concentraciones de oxígeno en la hemoglobina (entre los  $500\text{nm}$  y  $900\text{nm}$ , ver ejemplo en la figura 1). Esta técnica ha demostrado proporcionar medidas robustas de respuestas hemodinámicas localizadas en distintas zonas cerebrales [4].

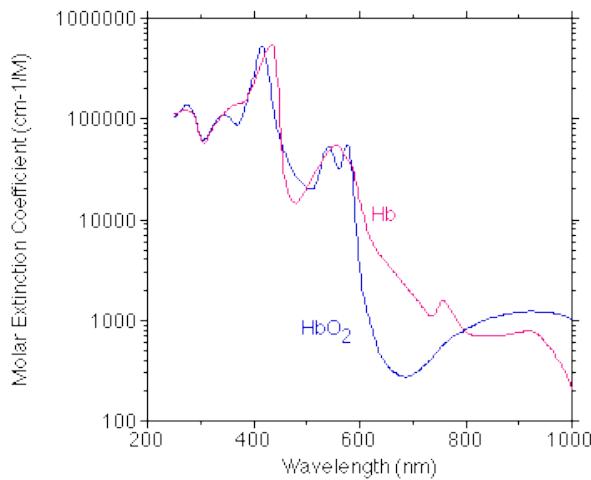


Figura 1: Picos de absorción de la oxi- y desoxihemoglobina

### 1.3 Objetivos del proyecto

El grupo de Neuroquímica y Neuroimagen de la Universidad de La Laguna porta un equipo de espectroscopía de infrarrojo capaz de determinar los cambios hemodinámicos tras una activación. Recientemente fue desarrollado por su

parte un hardware especial para medir las activaciones sobre un cerebro de rata común en un entorno *C++* en Linux. Este hardware está basado principalmente en una fuente de luz blanca, una cámara de alta velocidad y una óptica separadora de imágenes según las longitudes de onda seleccionadas de  $630\text{nm}$  y  $852\text{nm}$  a una velocidad de  $14,188\text{ Hz}$  (figura 2).



Figura 2: Equipo NIRS del grupo de Neuroquímica y Neuroimagen de la ULL

El **objetivo principal** del TFG consistirá en el desarrollo de una aplicación para ese entorno que permita recibir las imágenes capturadas por el hardware y realizar una representación del cambio hemodinámico con respecto al tiempo sobre el cual poder aplicar un análisis sobre un área de interés (ROI [5]) y posteriormente apoyar al trabajo con un análisis estadístico de los resultados obtenidos.

Otros objetivos son el desarrollo de un sistema portable, funcional, intuitivo e interactivo.

## 1.4 Estado inicial del proyecto

En la actualidad el equipo porta un software creado en Linux, en *C++* que permite procesar las imágenes originalmente de  $16\times 16$  píxeles que han sido capturadas y almacenadas en un búfer.

A partir de las dos imágenes capturadas por la óptica separadora a cada longitud de onda y sus coeficientes de extinción molar se permite calcular la concentración de hemoglobina en el medio por la aplicación de la ley de Beer Lamber modificada (Ecuaciones 1 y 2).

$$\mu a(\lambda_j) = OD(\lambda_j) = -\log \frac{\lambda_j - \lambda D}{\lambda R - \lambda D}$$

Ecuación 1: Ley de Beer-Lambert. Dónde  $\mu a(\lambda_j)$  es el coeficiente de absorción,  $\lambda R$  la calibración de referencia,  $\lambda D$  la calibración de ruido del entorno y  $OD(\lambda_j)$  la densidad óptica

$$CH = \mu a * \varepsilon$$

Ecuación 2: Concentración de hemoglobina. Dónde  $\mu a$  es el coeficiente de absorción y  $\varepsilon$  el coeficiente de extinción molar

Previa captura de las imágenes de interés, se obtienen dos *frames* de calibración el *dark* (figura 23 en el apartado 10.1 del Anexo), generado por la cámara en una zona completamente oscura que servirá para eliminar ruido [6] y el *gain* (figura 24 en el apartado 10.1 del Anexo) que permitirá ajustar la ganancia. El procesamiento de dichas imágenes se lleva a cabo mediante una serie de scripts desarrollados en *Matlab* [7] (Herramienta de software matemático).

En este contexto surge la propuesta del TFG cuyo objetivo es desarrollar una aplicación para el procesamiento y análisis de dichas imágenes que cumpla las funciones del actual sistema con una solución más permanente, robusta y sencilla al usuario.

En los siguientes capítulos se describirá el plan de trabajo y las herramientas utilizadas. A continuación, se procederá a explicar el desarrollo de la aplicación, para finalizar con las conclusiones y líneas futuras.

# Capítulo 2

## Plan de trabajo y tecnologías utilizadas

### 2.1 Plan de trabajo

Inicialmente se planteó desarrollar una aplicación en *Java* o *C++* que se comunicara mediante una API con el software de *Matlab*, llamando a los scripts ya existentes y de esta forma siendo sólo necesaria implementar la interfaz en alguno de los lenguajes anteriores. El lenguaje seleccionado fue *C++*.

Posteriormente se decidió que para eliminar completamente las dependencias de software externo se replanteara el diseño de la aplicación, en este caso se aportó una solución que enfocaba la implementación de los scripts en la propia aplicación.

Esta modificación supone un aumento importante de la complejidad inicial de la aplicación y un mayor requerimiento de tiempo. Es importante notar que muchas de las funciones disponibles en *Matlab* son funciones que corresponden a su propia librería y permiten realizar operaciones complejas de forma abstracta y sin necesidad de implementar código adicional.

Como ejemplo, en la figura 3, se puede observar una función bastante sencilla de máscara sobre dos imágenes en *Matlab* comparado con la implementación de la misma operación sobre la aplicación en *C++*, en la figura 4 (en ambos casos, la operación tiene una complejidad temporal de  $O(n^2)$  en el tamaño de la matriz). Otros métodos tienen que ser traducidos a extensiones mucho más grandes de código en la aplicación final.

```
% Imágenes im1, im2
mask = (im1 > 0.9) | (im2 > 0.9);
im1(mask) = 0;
im2(mask) = 0;
```

Figura 3: Máscara sobre dos imágenes en *Matlab*

```

auto matrix1 = img1.getData();
auto matrix2 = img2.getData();

for (int row = 0; row < matrix1.rows(); ++row) {
    for (int col = 0; col < matrix1.cols(); ++col) {
        if (matrix1(row, col) > MASK_VALUE || matrix2(row, col) > MASK_VALUE) {
            img1.set(row, col, 0);
            img2.set(row, col, 0);
        }
    }
}

```

Figura 4: Máscara sobre dos imágenes en *C++*

En este aspecto se requería reconstruir la aplicación desde cero, teniendo que traducir el código en alto nivel de *Matlab* a distintas funciones en el programa final.

Por otro lado, con diferencia del planteamiento inicial de realizar la interfaz de la aplicación utilizando los *frameworks* disponibles para *C/C++* o *Java*, se decidió optar por una solución web para la interfaz. Debido a ellos, la aplicación se ha realizado como dos módulos separados, uno creado exclusivamente para la lectura y procesamiento de las imágenes que será explicado en el capítulo 3 (módulo de back-end o procesamiento) y otro para la interfaz (módulo de front-end o interfaz) mostrado en el capítulo 4.

Este tipo de planteamiento tiene ciertas ventajas con respecto al original. La aplicación puede ser manejada en cualquier máquina siempre que el back-end soporte la multiplataforma (como es el caso) con una interfaz completamente idéntica sin distinciones de la máquina sobre la que se use. Esto provoca además un modelo implícito de desarrollo modelo-vista-controlador [8] (MVC), dónde la lógica de aplicación aparece completamente separada de la interfaz y los cambios en uno de los módulos no afectan a los demás.

Además, se ha trabajado con un sistema de control versiones, GIT, que permite ordenar el trabajo temporalmente y revertir a versiones anteriores en caso de que sea necesario, así como trabajar en nuevas versiones secundarias de código sin alterar la principal.

## 2.2 Cronograma y tareas

Debido a las razones del apartado anterior, el cronograma de las actividades a realizar expuesto inicialmente en la memoria del proyecto, se vio

desplazado. El proyecto, aunque cuenta con una aplicación completa, todavía se encuentra en proceso de desarrollo en cuanto a mejoras y líneas futuras que todavía están en proceso de desarrollo y líneas futuras que se explicarán en el capítulo 6.

Semana 1	Estudio del problema médico abordado y de la metodología seguida
Semanas 2-3	Estudio del prototipo ya existente en <i>Matlab</i> .
Semanas 4-5	Identificación de las carencias del primer prototipo y diseño de la interfaz.
Semanas 6-13	Desarrollo e implementación de la nueva aplicación/interfaz y de las propuestas de mejora (filtro, máscaras, análisis estadístico, etc.)
Semanas 14-16	Sincronización con Arduino

Tabla 1: Cronograma inicial

## 2.3 Tecnologías utilizadas

### 2.3.1 Lenguajes

Se ha utilizado una amplia variedad de lenguajes a la hora de desarrollar la aplicación. En la codificación del back-end de la aplicación, se recurrió por completo al lenguaje ***C++*** por el excelente control que ofrece sobre los datos y las optimizaciones y gran velocidad que puede aportar al programa final inherente a un lenguaje con un nivel de abstracción menor. Se ha completado con el *framework* de ***QT*** que contiene las herramientas necesarias para crear un código portable y acceder a funciones del sistema como el control de las ventanas o el sistema de archivos además de brindar la funcionalidad de un navegador con el módulo de *QtWebEngine*.

Por otro lado, para la creación del interfaz se han utilizado los lenguajes comunes al desarrollo web, ***javascript***, ***html*** y ***css*** inicialmente, haciendo una transición a los preprocesadores de lenguajes ***pug*** y ***sass*** en vez de utilizar directamente ***html*** y ***css*** respectivamente.

A lo largo del desarrollo se ha utilizado el compilador ***MSVC2017*** para Windows y ***MinGW*** para compilaciones en sistemas Linux.

A continuación, se describen brevemente las tecnologías y herramientas

utilizadas en el TFG.

### 2.3.2 Librerías y frameworks utilizados en el TFG

#### EIGEN [9]

Licencia [MPL2-licensed](#)

Eigen es una librería de plantillas en C++ para el álgebra lineal: matrices, vectores, solucionadores numéricos y algoritmos relacionados. Eigen destaca por ser versátil, tamaños de matriz virtualmente infinitos, fijos o dinámicos para todo tipo de números, incluyendo complejos. Ofrece una gran velocidad y una API robusta.

#### DSPFilters [10]

Licencia [MIT](#)

DSPFilters es una librería *C/C++* que resuelve el problema del concepto de los filtros multicanal IIR de orden arbitrario en el procesamiento de señales digitales implementado algunos como el *Butterworth*, *Chebyshev*, *Elliptic*, ...

#### Dygraphs [11]

Licencia [MIT](#)

DSPFilters es una librería *javascript* excelentemente rápida, flexible y *open source* para la creación de gráficas, permite representar y navegar sets de datos muy densos.

#### PIXIJS [12]

Licencia [MIT](#)

Librería gráfica para el renderizado de contenido 2D usando el renderizador WebGL.

#### QT [13]

Licencia [GPL](#)

Framework para la creación de aplicaciones e interfaces de usuario modernas multiplataforma.

#### JQuery [14]

Licencia [MIT](#)

Biblioteca de *javascript* que pretende simplificar el lenguaje.

### 2.3.3 Herramientas

#### **ADOBE PHOTOSHOP [15], GIMP [16]**

Software ampliamente utilizado para la edición de imágenes, diseño gráfico y arte digital.

#### **ADOBE ILLUSTRATOR [17]**

Software ampliamente utilizado para la edición de gráficos vectoriales.

#### **QT CREATOR [18]**

IDE Para el desarrollo rápido de aplicaciones *C/C++* integrado con *QT*.

#### **FLATICON [19]**

Servicio web dedicado a la creación y distribución de iconos digitales.

#### **GITLAB [20]**

Servicio web para el control de versiones y desarrollo de software.

#### **GIT [21]**

Software para el control de versiones.

#### **MTUNER [22]**

Software para el análisis de rendimiento y memoria de aplicaciones *C/C++*.

#### **PALETTON [23], CANVA [24]**

Servicio web para la creación de paletas de colores según patrones de diseño.

#### **ATOM [25]**

Editor de texto altamente personalizable con soporte para gran variedad de lenguajes y además de módulos para el desarrollo web.

#### **CHROME [26]**

Navegador web, usado como *debugger* a la hora del desarrollo de la interfaz.

## **MATLAB [7]**

Software matemático que permite scripting para el tratamiento de datos.

## **JSLint [27]**

Software de análisis estático para la calidad del código *javascript*.

## **SonarQube [28]**

Software de análisis estático el código *C/C++*.

## **SASS [29], PUG [30]**

Preprocesadores de lenguaje que ofrecen una alternativa con una mayor capa de abstracción a los lenguajes de marcado comunes *html* y *css* respectivamente.

## **Doxxygen [31]**

Doxxygen es uno de los estándares más conocidos a la hora de generar documentación de programas en *C++*.

### 2.3.4 Descartados

## **BOOTSTRAP [32]**

Framework para el desarrollo web. Fue elegido inicialmente en base a crear una interfaz intuitiva y moderna, pero más adelante se descartó debido a que aportaba un grado de complejidad demasiado extenso cuando la solución base era más sencilla de implementar

# Capítulo 3

## Módulo de back-end o procesamiento

### 3.1 Introducción

Como se ha comentado en capítulos anteriores, la aplicación se ha desarrollado usando dos módulos: el módulo de procesamiento, escrito en *C++* y el módulo de la interfaz, desarrollado con tecnologías web. Este capítulo se centrará en los aspectos del módulo de procesamiento y en realizar una explicación sobre el proceso de comunicación, sin entrar en detalles sobre el funcionamiento de la interfaz, cuya explicación y desarrollo, se puede encontrar en el capítulo 4.

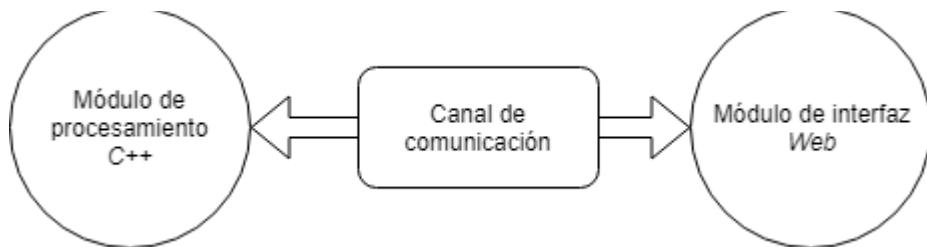


Figura 5: Estructura de la aplicación

El desarrollo de esta estructura provoca una mejora en el rendimiento de la aplicación y una independencia del sistema de procesamiento, al no tener que desarrollar ninguna lógica encargada de control de la interfaz de usuario en este módulo.

### 3.2 Estructura del archivo del experimento

El hardware existente almacena las imágenes capturadas en un archivo binario de extensión *.exp* cuyo diseño sigue el patrón que se muestra en la tabla 2.

Tamaño	Tipo de dato	Descripción
4Bytes	Entero 32 bits	Ancho de los <i>frames</i> en píxeles
4Bytes	Entero 32 bits	Altura de los <i>frames</i> en píxeles
4Bytes	Entero 32 bits	Profundidad de bits (BPP)
4B * Ancho * Alto	Flotante 32 bits	<i>Dark frame</i> , para calibración
4B * Ancho * Alto	Flotante 32 bits	<i>Gain frame</i> , para calibración

Repetir hasta final de archivo:

8Bytes	Entero 64 bits	<i>Timestamp</i> del <i>frame</i> en nanosegundos
2B * Ancho * Alto	Entero 16 bits	<i>Frame</i> de información

Tabla 2: Estructura del archivo .exp

La composición del archivo es simple, pero hay que destacar que el componente de la profundidad de bits, con un valor en el caso del archivo de pruebas de 12, no era utilizado originalmente en el programa en *Matlab*, sino simplemente ignorado y se asumía el tipo de dato que representaría los *frames* sería un entero de 16 bits (para acomodar un *bpp* de 12).

Sin embargo, los *frames* deberían ser leídos con un tipo de dato adecuado dependiendo de la profundidad de bits que indique el archivo. Esta corrección se realiza correctamente en el nuevo módulo implementado, pudiendo utilizar los valores de profundidad de 8, 16, 32 y 64, mostrando un error en caso de un valor mayor.

En este punto, se realiza una transformación de los *frames* leídos, ya sean enteros de 16 bits u otro tipo de datos, al tipo *double* (flotante de 64 bits) para realizar más adelante las operaciones en el cálculo de saturación y no repetir el proceso de transformación cada vez que se requiera un nuevo cálculo. Esta optimización extiende el tamaño en memoria del *frame* sin representar un problema, obteniéndose a cambio una gran mejora de velocidad de procesamiento en cálculos posteriores.

Por otro lado, conocer la frecuencia de muestreo es necesario a la hora de

aplicar los filtros (explicados en detalle en apartados posteriores). Este valor no aparece en el archivo *.exp* así que existirá la posibilidad de introducirlo mediante la interfaz o hacer un cálculo automático del mismo en base a los valores de *timestamp* de los *frames*. Se observa que el cálculo automático para el archivo de prueba tiene un valor de 14.190 *Hz* con respecto a los 14.188 *Hz* esperados.

Cabe destacar que cada uno de los *frames* leídos desde el archivo está compuesto por dos imágenes (las dos longitudes de onda que recoge la óptica separadora). En el caso del archivo de prueba, cada *frame* tiene una dimensión de 260x344 píxeles, y contiene dos imágenes de 260x172 píxeles cada una.

### 3.3 Sistema de comunicación

La información principal sigue un ruta única y omnidireccional a lo largo de todo el módulo de procesamiento, mostrada en la figura 6. A pesar de la explicación realizada y la reafirmación en la estructura de dos módulos, el sistema real implementado, aparenta más una estructura piramidal que horizontal, ya que resulta mucho menos complejo insertar el canal de comunicación sobre el módulo de procesamiento y a su vez que sea este el que soporte al propio módulo de interfaz (y a efectos prácticos no tiene ninguna repercusión negativa).

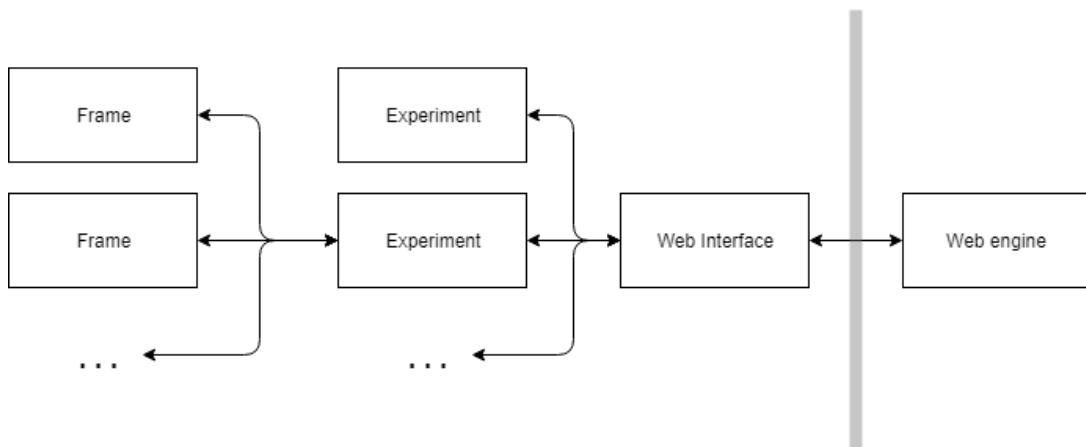


Figura 6: Flujo de la información

De esta forma, el propio *widget* de *QT* sirve de canal de comunicación y administra el módulo de procesamiento. Su funcionamiento básico es el siguiente:

1. Se crea un objeto del tipo *QWebView* y se le asignan los recursos

correspondientes a la interfaz web. Este objeto está basado en un navegador Chromium que permite acceder a las capacidades de *WebGL* que ofrece una gran velocidad en el renderizado. Además, existe la ventaja de poder acceder desde un navegador Chrome a las herramientas de desarrollador y conectar con la propia interfaz para hacer *debugging*.

2. Se crea un objeto u objetos que servirán de canal a la hora de conectar con la interfaz web. Este objeto debe heredar de *QObject* e incluir la macro *Q\_INVOKABLE* previa a cada una de las funciones que puedan ser llamadas desde la interfaz web.
3. Se le indica a *QWebView* que el objeto creado en el punto 2 será el canal de comunicación.

A continuación, en la figura 7, se muestra un pequeño resumen de ejemplo de uso. Al ser una tecnología que pertenece a las nuevas versiones de *QT*, existe documentación al respecto, aunque no es demasiado extensa y no está centralizada en el momento de realizar esta memoria.

El uso de la interfaz desde el código javascript en la web es bastante sencillo de implementar, incluyendo en el documento *html* la librería de *QT* e instanciando un objeto de la clase *QWebChannel*.

```

class Interface {
public:
    Q_INVOKABLE int getValue();
};

QWebView *view = new QWebView();
QWebChannel *channel = new QWebChannel();
Interface *interface = new Interface();
view->setUrl("URL");
view->page()->setWebChannel(channel);
channel->registerObject("web_interface_id", interface);

<script src="qrc:///qtwebchannel/qwebchannel.js"></script>
<script>

    new QWebChannel(qt.webChannelTransport, function(channel) {
        let channel = channel.objects.web_interface_id;
        channel.getVal(val => console.log(val));
    });

</script>
```

Figura 7: Ejemplo de uso de una interfaz con *QWebChannel* en *QT*

## 3.4 Estructura del módulo

Para la realización del módulo de procesamiento de las imágenes se ha creado una estructura de clases, siendo la más importante *Experiment*, cuyas tareas principales son cargar el experimento de archivo y calcular las nuevas imágenes, así como sus valores de saturación a lo largo del tiempo. La información de las imágenes se guarda en instancias de la clase *Frame* que se corresponde con una matriz bidimensional. En la figura 8, se puede observar un diagrama de la jerarquía de clases y dependencias, sin embargo, para facilitar la compresión del esquema, se han eliminado las detalles de las clases así como las clases que heredan de *std::exception* e implementan distintas excepciones, a continuación se listan las excepciones disponibles:

1. *BadIndexForFrameException*: Se ha intentado acceder a un *frame* que no existe en el experimento.
2. *CantWriteFileException*: Se ha producido un problema al generar un archivo (CSV [33], de imagen, ...). Comúnmente se debe a un problema de permisos.
3. *FileNotFoundException*: No se puede acceder a un archivo para su lectura.
4. *FileReadErrorException*: El contenido del archivo no coincide con el esperado o se ha producido un error de lectura.
5. *FrameBPPTooBigException*: El valor de profundidad de píxel de los *frames* (bpp) es demasiado grande (valores superiores a 64).
6. *SizeFrameMissmatchException*: Se ha intentado realizar una operación entre *frames* de distintos tamaños.

También se encuentran disponibles dos clases que siguen el patrón Singleton que también han sido excluidas del diagrama de la figura 8: *Logger* y *Settings* que permiten al programa cargar en un inicio toda la configuración de la última sesión desde un archivo *.ini*, además de llevar un registro de las operaciones realizadas que serán almacenadas en una serie de archivos *.log* dentro de la carpeta *logs*. Estos archivos serán almacenados en la sección de datos de usuario pertinente, cuya ruta que será provista por el *framework* de *QT* según el sistema operativo sobre el que se haya instalado el archivo.

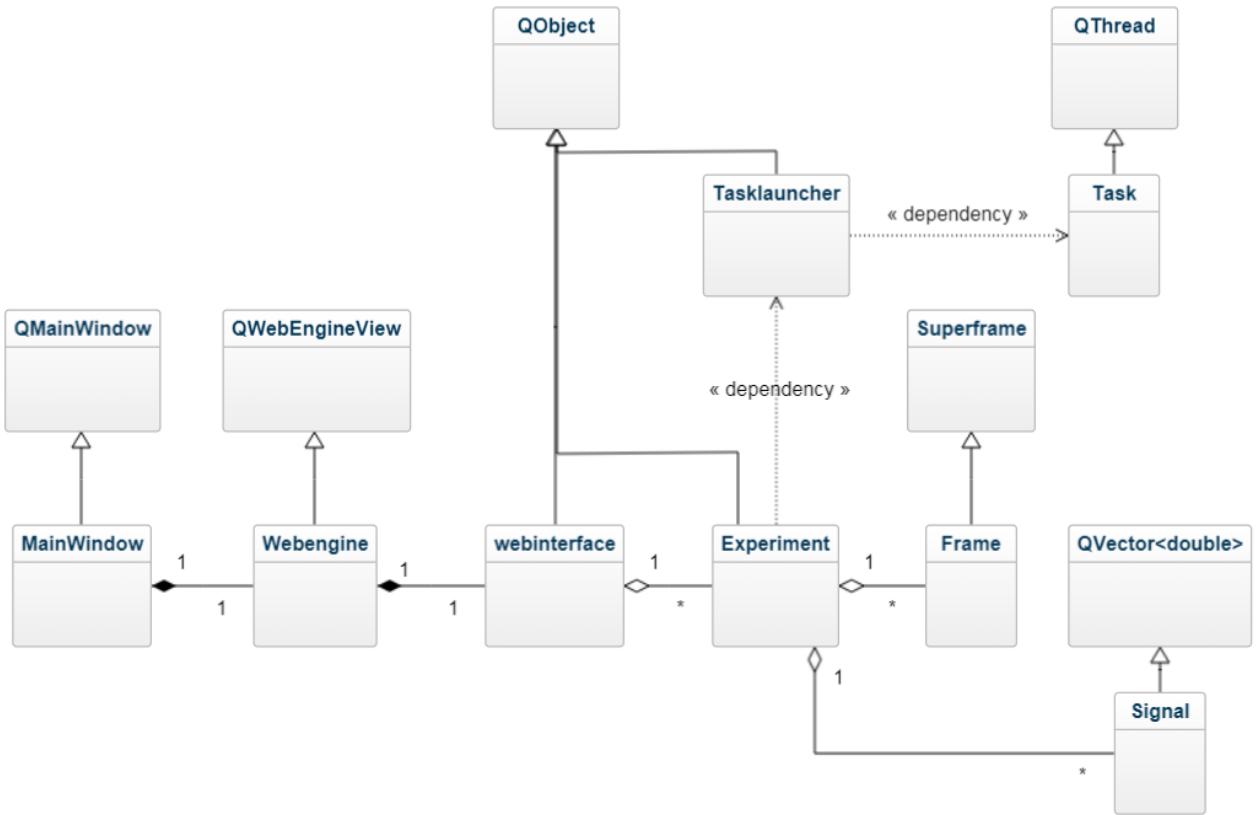


Figura 8: Diagrama simplificado de clases del módulo de procesamiento. Las clases con la inicial *Q* pertenecen al *framework* de *QT*.

### 3.5 Procesamiento

El núcleo del módulo de procesamiento es la funcionalidad de realizar el tratamiento de las imágenes recibidas. Esta funcionalidad se divide en varias subtareas o fases distintas.

Por un lado, existe la posibilidad de calcular la saturación de un *frame* arbitrario, compuesto por dos imágenes de la misma región en diferentes longitudes de onda. En un paso inicial, se elimina el ruido y se ajustan sus valores utilizando los *frames dark* y *gain* (vistos con más detalle en el anterior apartado 1.4), para después realizar el cálculo de la saturación siguiendo la fórmula de la ecuación 3, que representa una extensión de la fórmula de Beer-Lambert (ecuación 1). Este es otro ejemplo de uno de los scripts que inicialmente se encontraban planteados en *Matlab* y que fueron traducidos a *C++* para ser integrados en la aplicación.

$$F_{calibrado} = (F_{original} - F_{Dark}) * F_{Gain}$$

$$F_{calibrado} \rightarrow img_1, img_2$$

$$img_{sat} = \frac{\frac{\log img_1}{\log img_2} * \lambda_{HbR_2} - \lambda_{HbR_1}}{\frac{\log img_1}{\log img_2} * (\lambda_{HbR_2} - \lambda_{HbO_2}) + (\lambda_{HbO_1} - \lambda_{HbR_1})}$$

Ecuaciones 3: Cálculo de la saturación

Por otro lado, se genera un basal [34] y su imagen de saturación, es decir, una imagen resultado del promediado de las imágenes iniciales antes de producir un estímulo. Es posible seleccionar ya sea en milisegundos o en *frames* cuál es el rango que va a ocupar en toda la dimensión temporal del experimento el basal. Su cálculo es sencillo y se corresponde con la fórmula de la ecuación 4 (empezando el rango desde el inicio del experimento). Posteriormente se obtiene del *frame* resultado una imagen de saturación del basal, siguiendo la fórmula anterior (ecuación 3).

$$F_{basal} = \frac{1}{n} \sum_{i=inicio}^{final} F_i$$

Ecuación 4: Cálculo del basal

Una vez se obtiene el basal este será sustraído de cada una de las imágenes correspondientes al resto del experimento.

Por último, una de las operaciones básicas necesaria es el cálculo de los valores de concentración de hemoglobina para cada una de las dos imágenes (respectivamente aquella que responde a cambios en la oxihemoglobina y aquella que responde a cambios en la desoxihemoglobina). Este cálculo se realiza haciendo una media de cada una de dichas imágenes después de haber sido calibrada con los *frames dark* y *gain* (sin necesidad de crear una imagen de saturación).

Los resultados de estas operaciones dan lugar a una gráfica de concentraciones de la hemoglobina, así como a distintas imágenes de saturación.

Más adelante en el capítulo 4 se explicará el funcionamiento de la interfaz, pero en las figuras 9 y 10 se pueden observar los resultados en dónde se ha aplicado un mapa de color. Se pueden ver más ejemplos en el apartado 10.3 del Anexo.

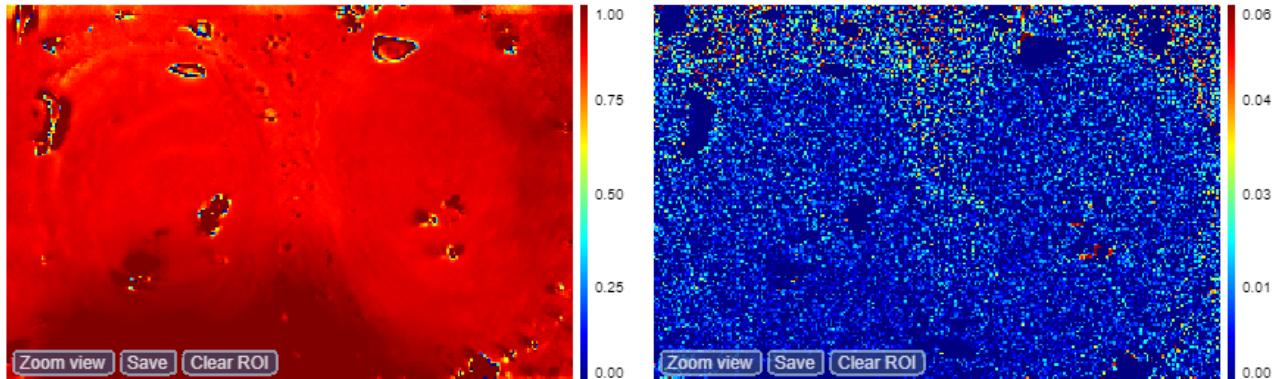


Figura 9: Imagen coloreada de saturación basal (Izquierda), valores normalizados. Imagen coloreada de saturación (Derecha), en este caso se han truncado los valores entre 0.06 y 0.00

Se ha implementado también la posibilidad de realizar todas las funciones anteriores con respecto a un área determinada de la imagen (ROI [5]).

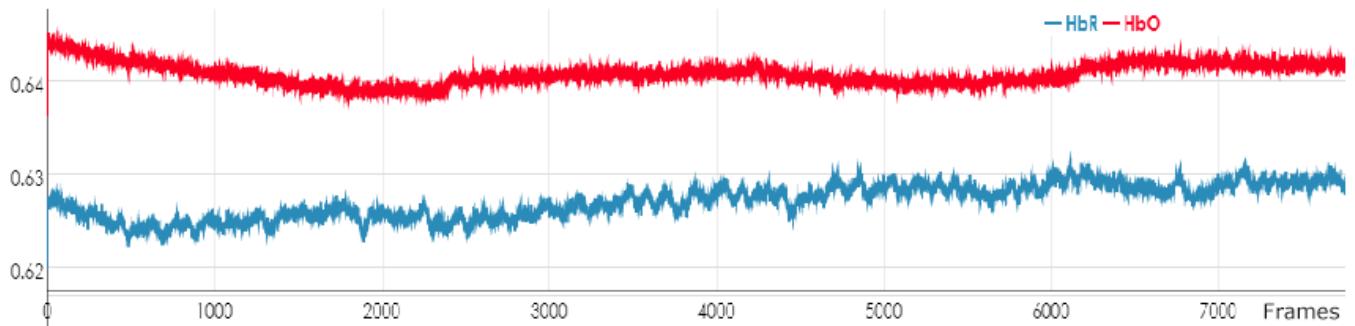


Figura 10: Cambios en la concentración de las moléculas de HbR (desoxihemoglobina) y HbO (oxihemoglobina)

### 3.6 Filtros

Una de las necesidades del sistema es la de poder aplicar filtros de señal a los resultados de las concentraciones de hemoglobina. Para ello se ha recurrido a la librería DSPFilters, ya mencionada en el apartado 2.3 de tecnologías utilizadas, que proporciona una API sencilla para la creación de diferentes filtros de los cuales se han utilizado los siguientes: paso alto, paso bajo, paso banda, elimina banda. Estos filtros están accesibles en el módulo de back-end y permiten procesar las señales bajo demanda. En la figura 11 se puede ver un

ejemplo de funcionamiento de un filtro sobre una porción de los datos de la gráfica. Se comentará la interfaz de filtros más adelante en el capítulo 4.

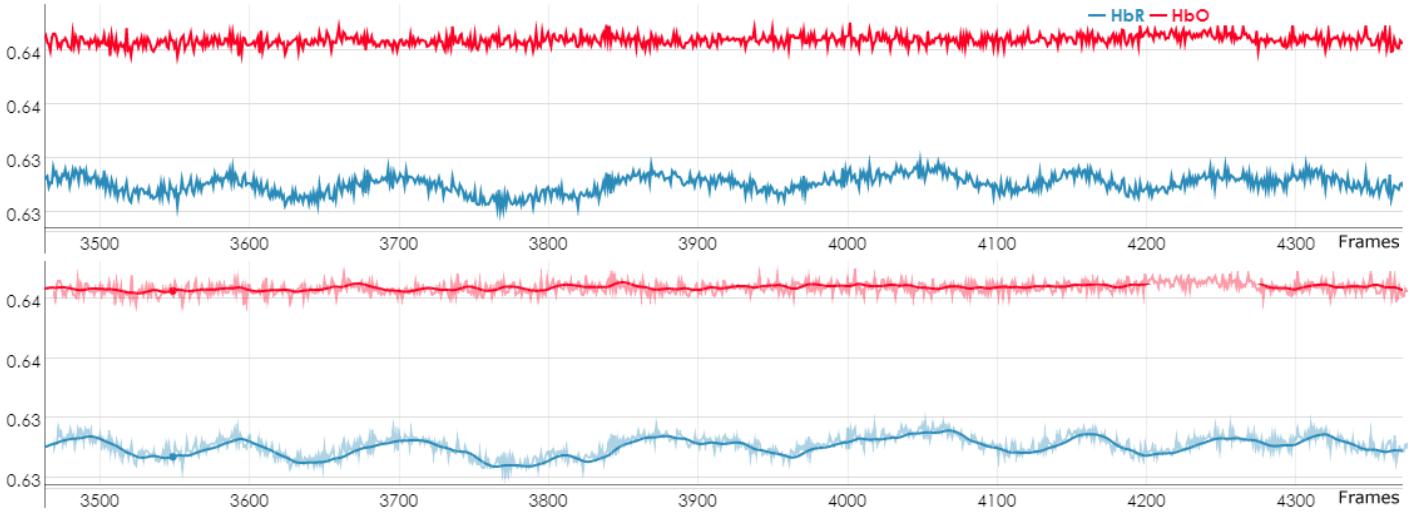


Figura 11: Aplicación de un filtro paso bajo a una porción del gráfico de concentración

### 3.7 Multithreading

Una de las características que debe de tener la aplicación es la de ser interactiva en tiempo real. El sistema de interfaz está construido sobre un *QWebEngineView* que implementa un navegador web y a su vez este incrustado en el módulo de procesamiento como se ha explicado en apartados anteriores. El problema recae en que el módulo de procesamiento debe realizar las tareas pesadas en hilos independientes para evitar que la interfaz permanezca congelada durante el procesamiento ya que el navegador no es ejecutado sobre un proceso distinto (esta decisión viene impuesta por el propio *framework* de *QT*).

La solución se ha desarrollado mediante las clases *TaskLauncher* y *Task* que permiten la creación de tareas independientes del programa principal de forma instantánea. Además, la tarea más pesada computacionalmente del cálculo de los valores de la gráfica se ha paralelizado de forma que su rendimiento aumenta más de un 20% cuando se emplean dos hilos. Sin embargo, la sobrecarga de crear las diferentes tareas y sincronizar la información no permite mejorar el rendimiento incluyendo un mayor número de hilos como se

observa en la gráfica de la figura 12 que muestra las pruebas realizadas.

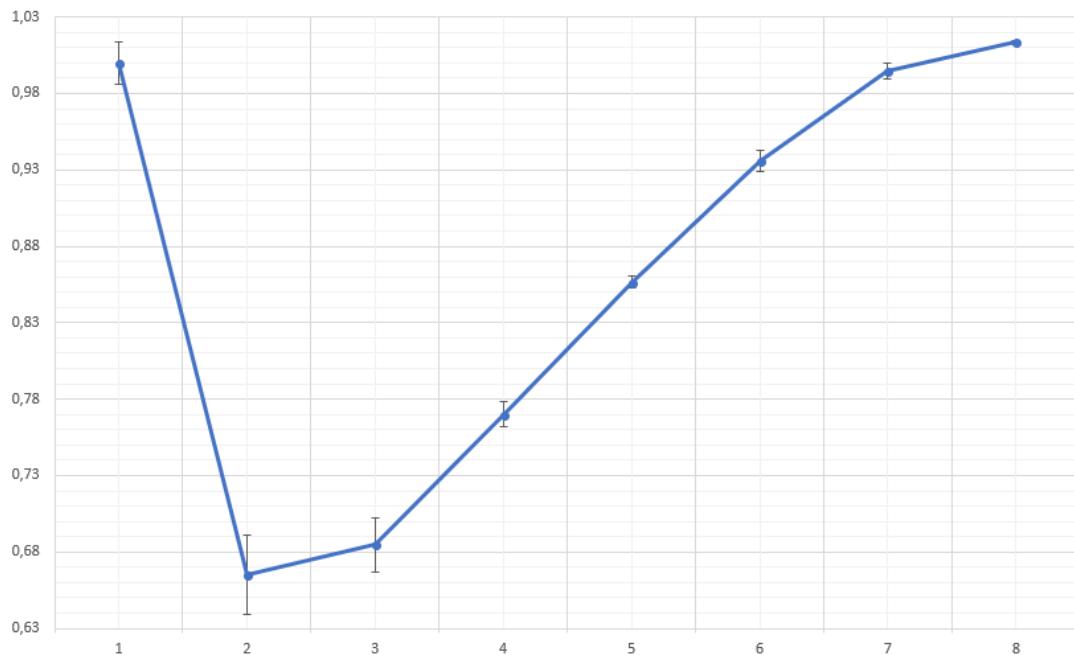


Figura 12: Tiempo de ejecución relativo (eje ordenadas) con respecto al número de hilos (eje de abscisas) en un sistema con un procesador de 4 núcleos para  $\sim 8000$  frames.

Las distintas tareas realizadas en paralelo por el módulo anunciarán su estado por un sistema de señales, lo que permite a la interfaz saber cuándo una tarea está en proceso y qué porcentaje lleva.

### 3.8 Multiplataforma

El código del módulo de back-end está diseñado en su totalidad para ser portable a cualquier sistema Windows o Linux. Las librerías utilizadas también permiten dicha portabilidad.

También la interfaz como se verá en el capítulo 4, puede funcionar en cualquier sistema, ya que la tecnología web utilizada es soportada por un navegador que no aplica cambios al contenido según el sistema sobre el que funcione.

# Capítulo 4

## Módulo front-end o interfaz

### 4.1 Introducción

Una de las principales características del proyecto es el diseño de la Interfaz. En este caso a pesar de ser una tarea con unos límites correctamente definidos dentro del cronograma (Tabla 1), se trabajó en ella de forma iterativa a lo largo de todo el proyecto. En el apartado 10.2 del Anexo se pueden ver dos imágenes que reflejan algunas de las etapas anteriores.

Se ha buscado el desarrollo de una interfaz moderna que presente la información de forma ordenada. Se puede ver la interfaz inicial sin ningún experimento cargado en la figura 13 y con un experimento cargado en la figura 14. En los siguientes apartados se explicará cómo ha sido el proceso de diseño y desarrollo de la interfaz.

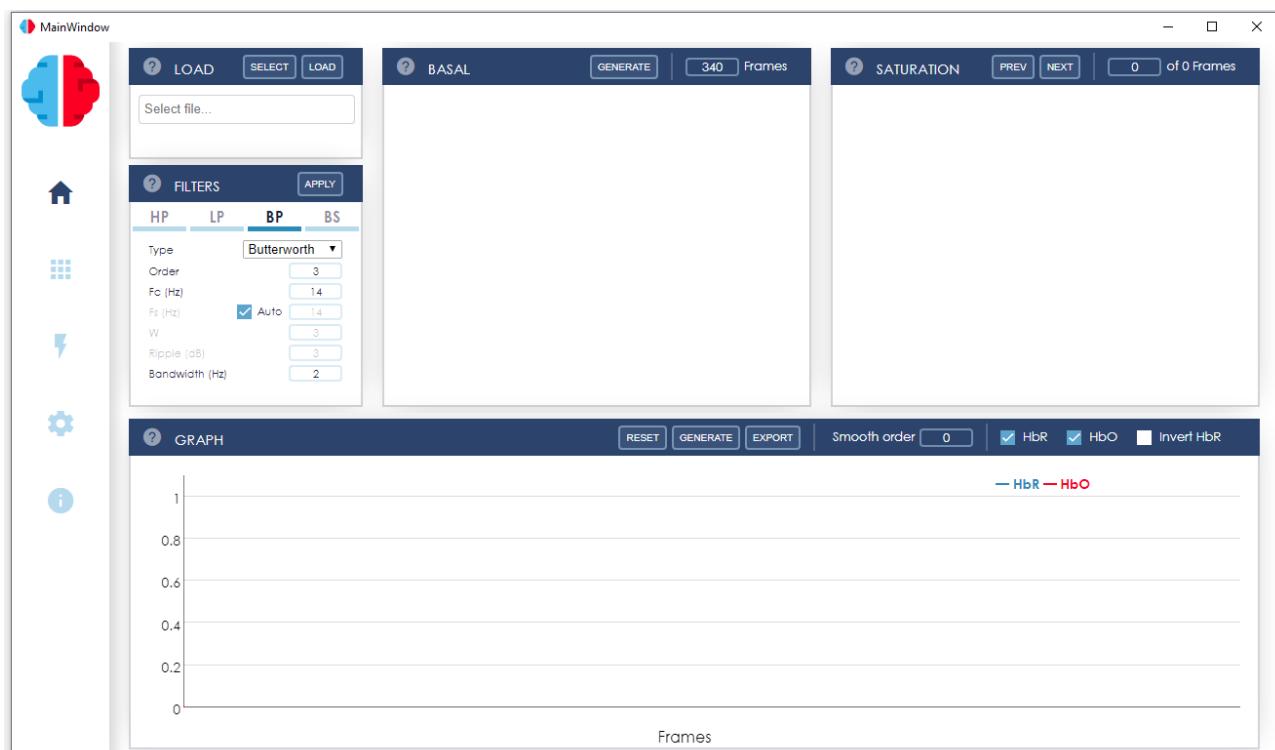


Figura 13: Interfaz de la aplicación. Zona principal.

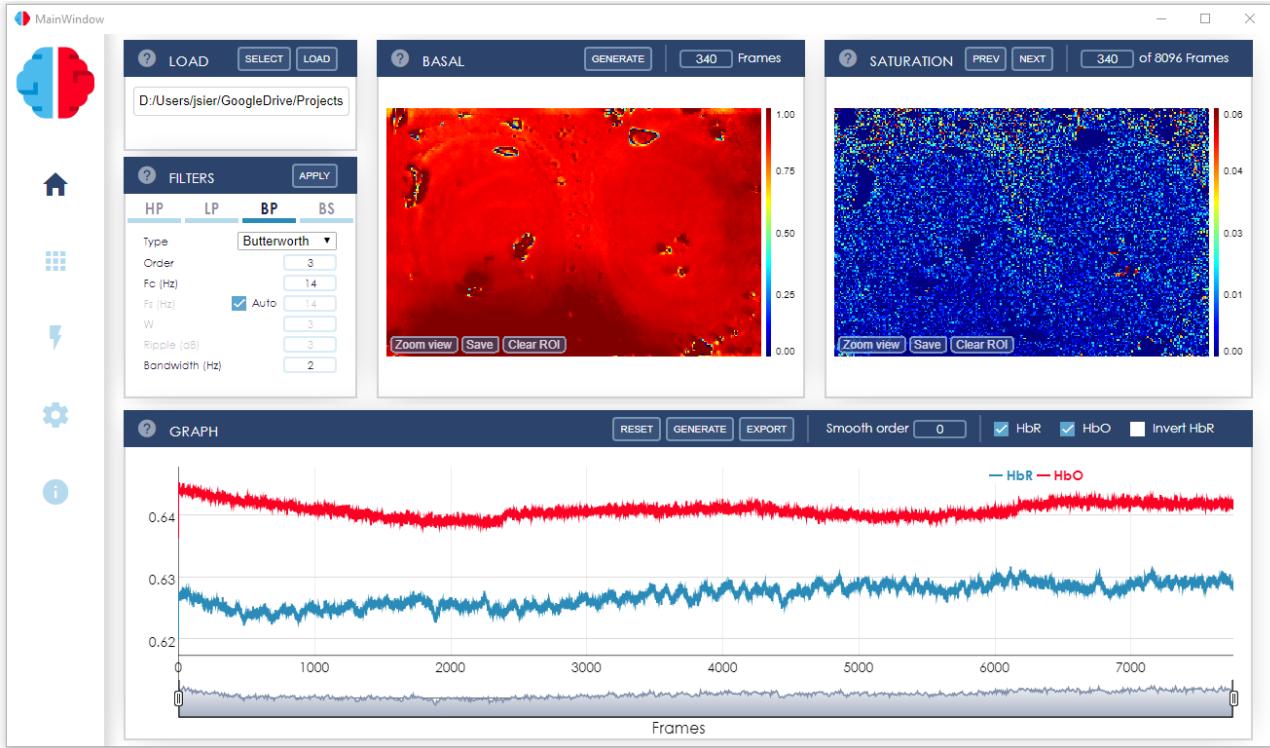


Figura 14: Interfaz de la aplicación. Zona principal. Experimento cargado.

## 4.2 Esquema de colores y logo

Durante la creación de la aplicación primero se llevó a cabo un diseño preliminar de distintos conceptos de logo con la ayuda de Adobe Ilustrador junto a una búsqueda de un logo que reflejara de la mejor forma posible el concepto de la tecnología NIRS o sus aplicaciones a la detección de activaciones cerebrales. Finalmente se decidió utilizar un logo del servicio web Flaticon (figura 15) que sigue el esquema general de una interfaz simple y ordenada.

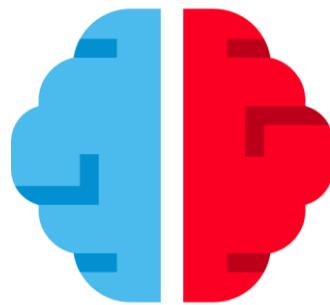


Figura 15: Logo de la aplicación. Sujeto a las normas de copyright presentadas al principio del documento.

Previamente al diseño del logo se había generado una paleta de colores de diferentes tonos de azul, eligiéndose un blanco para el fondo de la aplicación y otros pequeños aspectos, con interpretaciones de calma, orden y confianza [35][36] para una aplicación cuya finalidad son los estudios médicos. Por otro lado, el logo seleccionado completa el esquema con un tono rojo que llama la atención y se interpreta como una activación cerebral, siguiendo la regla del 60% - 30% - 10% para los colores en una aplicación o web (Dónde 60% es color dominante, 30% el secundario y 10% el utilizado para resaltar detalles). El esquema completo de colores se detalla en la figura 16.

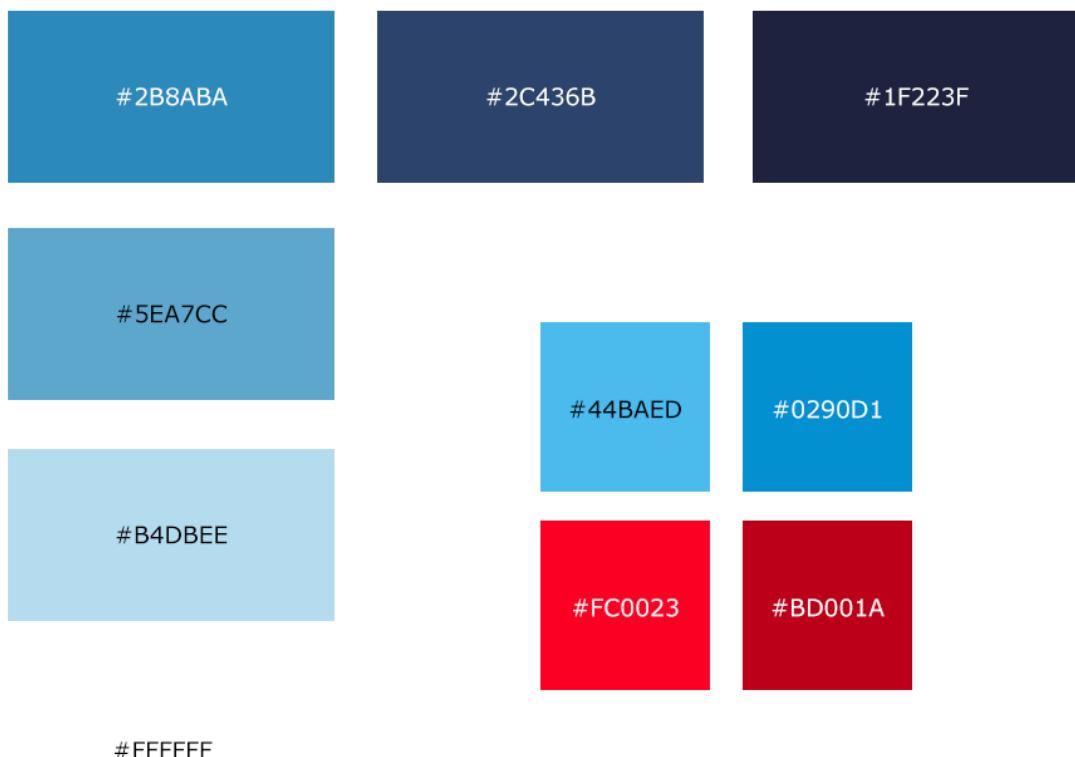


Figura 16: Esquema de colores de la aplicación.

### 4.3 Diseño de la interfaz

La interfaz presenta un diseño moderno y simple con una barra lateral que permite ubicarnos rápidamente a través de un intuitivo menú de iconos, que realizará un *scroll* vertical sobre el contenido para acceder a las diferentes zonas definidas (usando la rueda del ratón o las flechas del teclado), estas son:

- Home: Zona principal, contiene las operaciones básicas a realizar y

cumple con los objetivos generales.

- Análisis estadístico: Esta zona está reservada para llevar a cabo los análisis estadísticos que se definirán más adelante en el apartado de trabajo en desarrollo 5.2 del capítulo 5.
- Arduino: Zona reservada para la sincronización del hardware con una señal de Arduino configurada por la aplicación. Este problema está definido en el apartado de líneas futuras 6.1 del capítulo 6.
- Settings: Zona de configuración de los distintos parámetros disponibles. Se puede ver una imagen en la figura 17. Entre las configuraciones disponibles se puede seleccionar la unidad temporal sobre la que trabajar, *frames* o milisegundos. El tipo de separador de los archivos CSV [33] generados, las localizaciones en el sistema de archivo para almacenar estas opciones y los archivos de *logs*. Y por último los parámetros que definen el comportamiento de los mapas de color.
- About: Zona dedicada a la descripción del proyecto. Se puede ver una imagen en la figura 18.

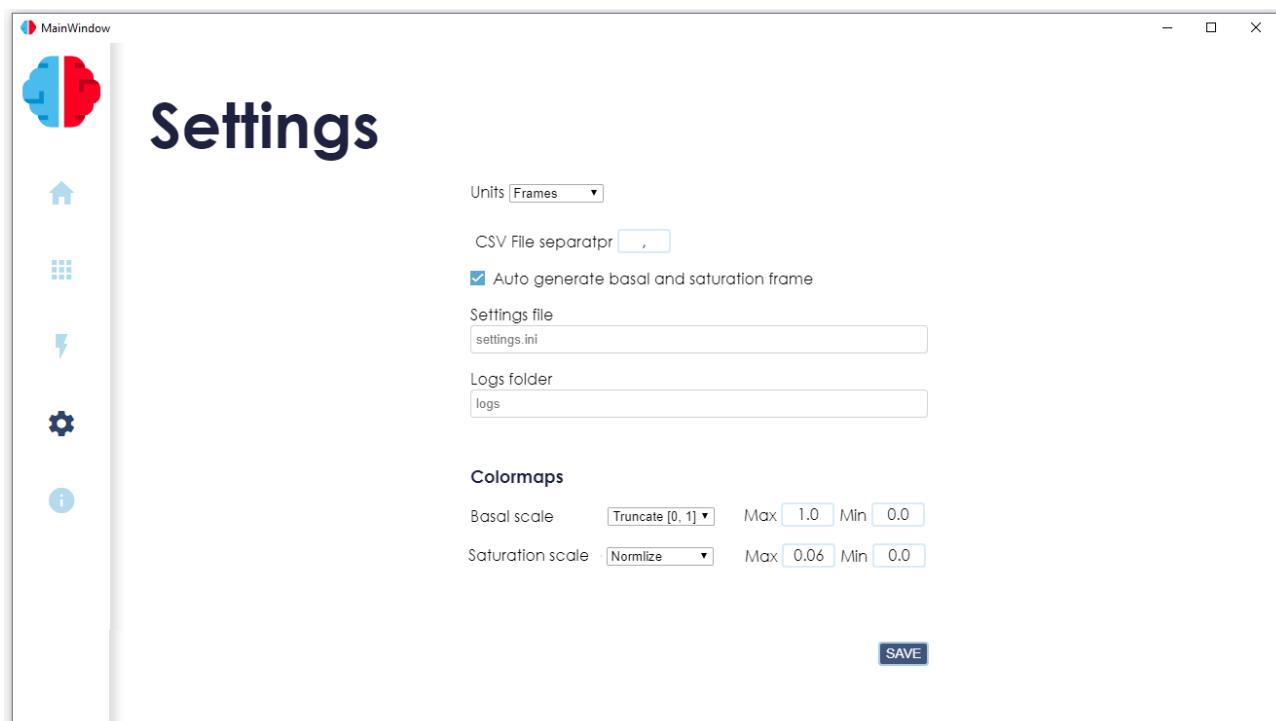


Figura 17: Interfaz de la aplicación. Zona de opciones.

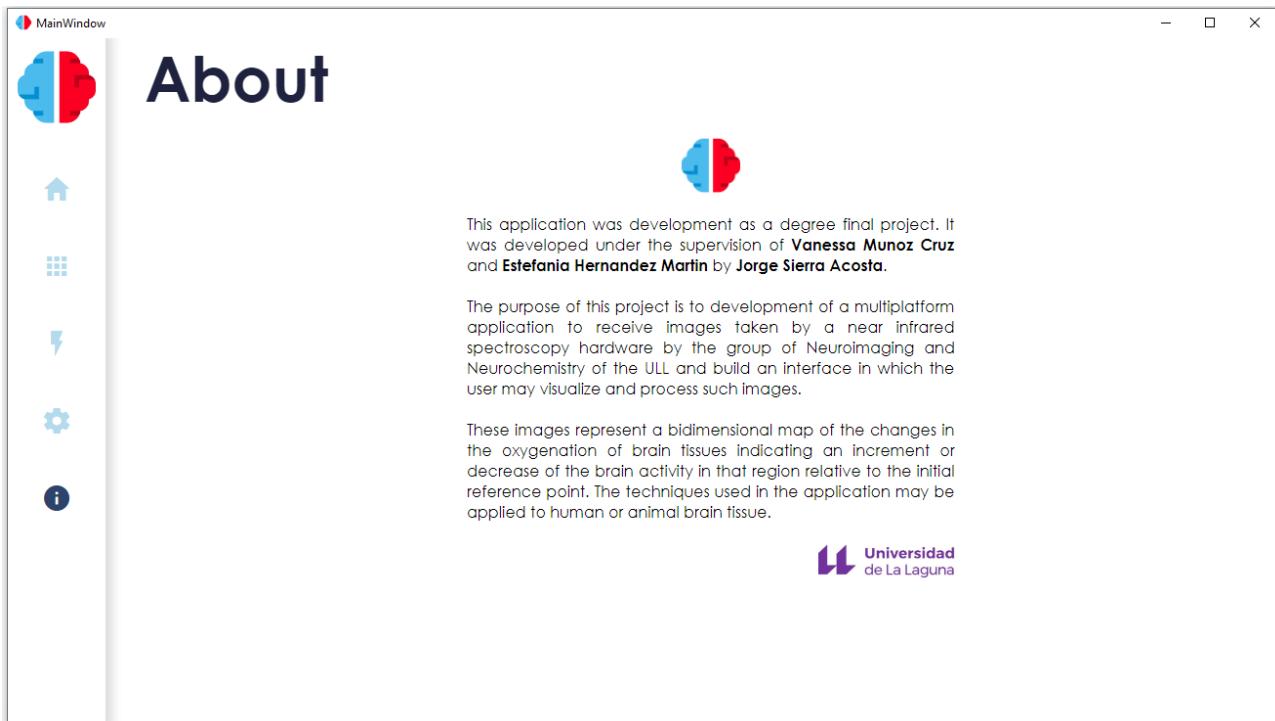


Figura 18: Interfaz de la aplicación. Zona about.

La ventana principal muestra una estructura sencilla diseñada para interactuar de forma ordenada de arriba abajo y de izquierda a derecha. Mediante una clara división por secciones (a las que se refiere comúnmente como tarjetas [37]) se han repartido y concentrado las diferentes funcionalidades. A su vez se han introducido en los títulos los controles inherentes a cada una de ellas que se muestran más adelante.

- Tarjeta *Load*: Permite la carga de archivos de experimento y muestra un mensaje en caso de error.
- Tarjeta *Basal*: Una vez cargado el experimento se generará una imagen basal automáticamente en esta tarjeta. En la parte superior se indica en un recuadro editable el punto final del basal en *frames* por defecto y un botón “generar” que permite volver a generar el basal en caso de actualizar el valor.
- Tarjeta *Saturation*: Tarjeta similar a la anterior, en este caso muestra las imágenes de saturación. El menú superior permite navegar fácilmente entre las distintas imágenes.
- Tarjeta *Graph*: Ocupa toda la zona inferior de la ventana principal, en ella se podrán mostrar de forma conjunta los datos de concentración HbO (oxihemoglobina) y HbR (desoxihemoglobina). El menú superior

contiene las opciones necesarias para mostrar sólo una de las gráficas, o mostrar los valores de HbR invertidos con respecto al eje de las abscisas. Posee botones para generar el gráfico, resetear los valores (eliminando los filtros aplicados) y exportar la información en formato CSV a un archivo. Además, la gráfica, implementada con la librería Dygraphs, permite realizar funciones de zoom y movimiento sobre el gráfico, además del uso de una función de suavizado (equivalente a un filtro paso bajo, aunque de menor calidad).

- Tarjeta *Filters*: muestra los cuatro filtros disponibles (paso alto, paso bajo, paso banda y elimina banda) con todas sus configuraciones posibles. Una vez generados los datos de la gráfica, es posible aplicar uno de estos filtros en esta tarjeta. (ver figura 11 para comprobar el resultado de la aplicación de un filtro y figura 19 para la interfaz). Además, aquí se ofrece la opción de utilizar la frecuencia de muestreo automática (explicada en el apartado 3.2 de la estructura del archivo del experimento) o bien especificar una manualmente.

La mayoría de estas tarjetas, dependiendo de su funcionalidad presentan una barra de carga en su parte inferior, visible en la figura 19, que recibe información sobre el estado del proceso del módulo de back-end.

## 4.4 Guías de diseño WCAG

Siendo una interfaz desarrollada con tecnologías web, se ha decidido seguir las guías de diseño WCAG [38] desarrolladas por W3C [39]. Algunas de las guías de diseño seguidas y aplicadas han sido las siguientes:

1. Percepción
  - a. Proveer de alternativas de texto para contenido no textual.
  - b. Hacer sencillo para los usuarios ver y escuchar el contenido.
2. Operabilidad
  - a. Presentar la funcionalidad disponible con un teclado.
  - b. Dar a los usuarios tiempo para leer el contenido
  - c. No usar contenido que pueda provocar reacciones físicas
3. Comprensibilidad

- a. Presentar un texto comprensible y entendible
  - b. Hacer el contenido operable y aparente en una forma natural
  - c. Ayuda a los usuarios a corregir y evitar errores
4. Robustez
- a. Maximiza la compatibilidad con herramientas futuras.

La perceptibilidad se ha implementado creando una interfaz sencilla e implementando ayudas al contenido no textual con las *tooltips* del ratón. Además, el contraste de los textos se intenta mantener siempre en la mejor de las condiciones utilizando los colores más adecuados de la paleta para alcanzar como mínimo el nivel WCAG AA, e idealmente WCAG AAA en cuanto a calidad. La operabilidad se refleja en una interfaz manejable con el teclado y la comprensibilidad en base a una interfaz intuitiva con textos sencillos que responde a un orden natural de izquierda a derecha y de arriba a abajo.

En la figura 19 se puede observar la tarjeta *load* junto a la tarjeta *filters*, mostrando un mensaje de error que ayuda a la comprensibilidad, pasando el ratón por encima de los símbolos de ayuda se muestra una breve descripción.

Además, se presentan sonidos de “operación completada” y “operación fallida” que ayudan a guiar al usuario o a facilitar tiempos de espera prolongados cuando la cantidad de datos es grande.

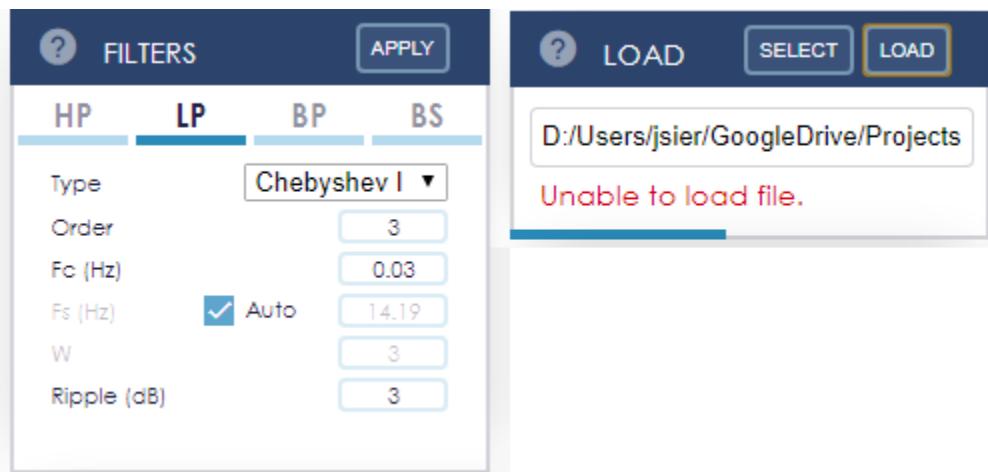


Figura 19: Tarjetas de *load* y *filters*. Barra de carga en progreso y mensaje de error. (En la aplicación aparecen una sobre la otra, no en horizontal).

## 4.5 Visor de imágenes

Para la visualización de las imágenes a color se ha construido desde cero un sencillo visor de imágenes, figura 20. Este funciona sobre un elemento *canvas* de *html5* sobre el cual se realizan funciones de renderizado básicas (texto, formas, escalado, ...) a través de la librería *PixiJS*, que permite aprovechar toda la capacidad de la GPU mediante la tecnología de *WebGL* para gráficos en navegadores web.

Una vez asignada una imagen al visor, este genera su propia escala de color junto a una versión escalada de la imagen. Sobre la imagen se podrá seleccionar un área determinada de trabajo (ROI [5]) para las operaciones posteriores de cálculos de concentración pulsando y arrastrando el ratón sobre la misma. Contiene un menú inferior que permite almacenar la imagen en la máquina, borrar la selección del área de interés o bien entrar en el modo zoom, que permite observar con más detalle las diferentes zonas y ver cuál es el valor del pixel sobre la imagen original que estemos señalando.

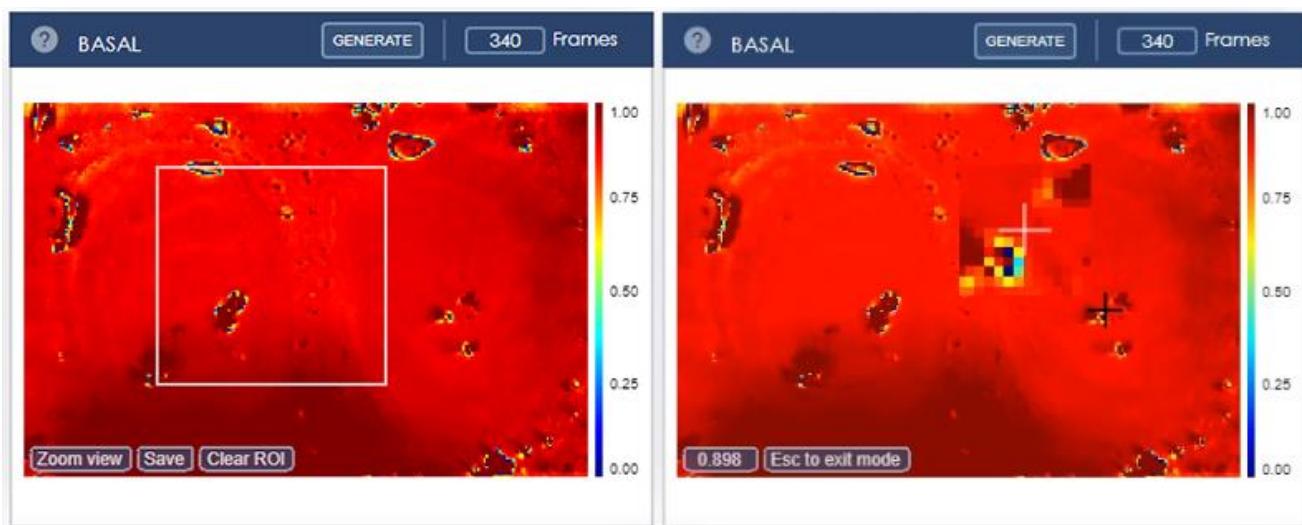


Figura 20: Visor de imágenes con una imagen basal. Modo normal con un área de interés seleccionada (Izquierda). Modo zoom sobre un pixel de valor 0.898 (Derecha).

## 4.6 Estructura de programa de la interfaz

La estructura del programa de la interfaz se ha basado en un documento escrito en lenguaje *pug* y un único documento de estilos en lenguaje *sass* que serán traducidos automática y respectivamente a *html* y *css*. La lógica de interfaz está implementada en *javascript*, pero se ha recurrido a un sistema de clases que permita organizar, mantener y estructurar mejor el código. Se puede ver un diagrama simplificado en la figura 21.

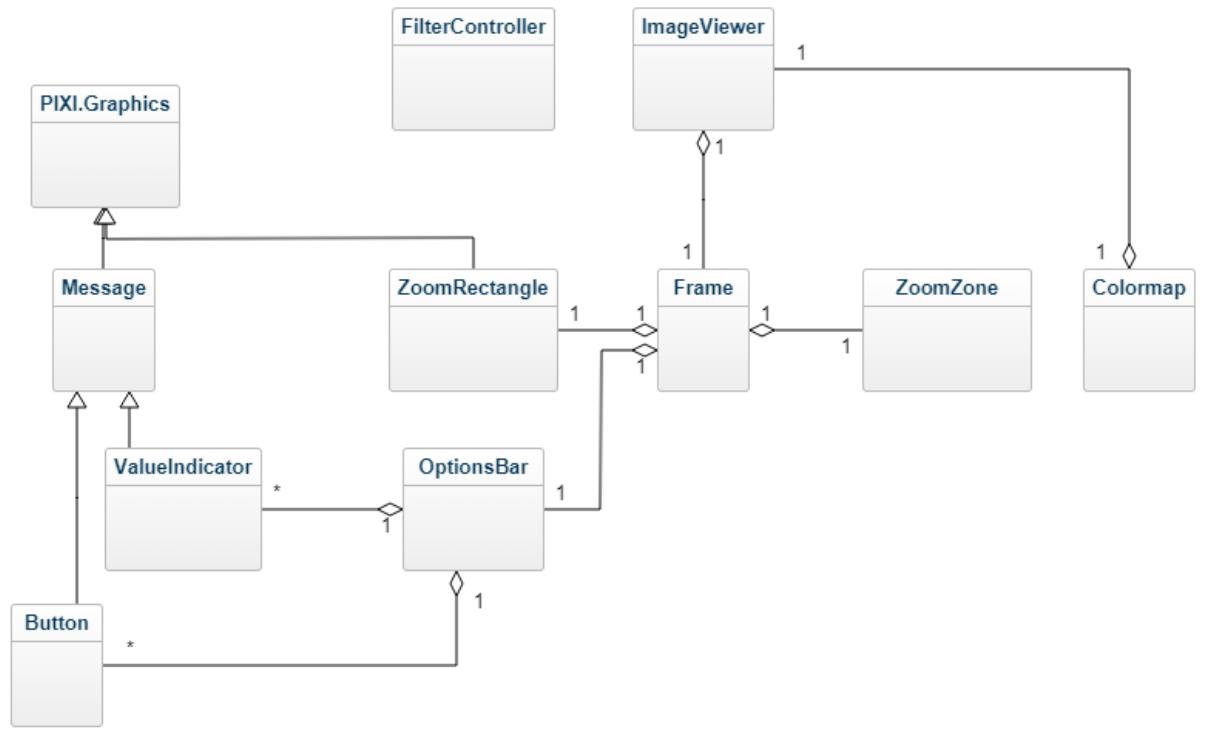


Figura 21: Estructura simplificada de clases de la interfaz.

# Capítulo 5

## Problemas y trabajo en desarrollo

### 5.1 Problemas y dificultades

A continuación, se describen brevemente los principales problemas encontrados en el desarrollo del TFG.

Una de las primeras dificultades fue la de interpretar el formato en que se almacenaban las imágenes en los archivos *.exp* a partir de los scripts de *Matlab*. Fue necesario recurrir a herramientas que facilitar una visualización de un archivo en hexadecimal para estudiar la estructura del archivo.

Seguidamente se comprobó que, de los dos archivos de prueba existentes, uno de ellos se encontraba originalmente corrupto por razones desconocidas y quedó inutilizable. Sin embargo, el primer archivo de pruebas se encontraba en perfecto estado y fue más que suficiente a la hora de realizar y comprobar la aplicación.

Otra dificultad técnica se encontró a la hora de realizar la clase *Frame* en el módulo de procesamiento, ya que utiliza una matriz de la librería Eigen cuyo tipo de dato es aplicado en tiempo de compilación en forma de plantilla. La solución se basó en construir la clase *Frame* a su vez como una plantilla y crear las funciones adecuadas para la conversión entre *frames* de distintos tipos.

Sin embargo, un problema que quedó sin solucionar es la “lenta” reacción que tiene la interfaz a la hora de redimensionar la pantalla, sus elementos parecen redimensionarse y recolocarse de una forma poco fluida. Esto se debe a un error no en el código de la aplicación sino en el *framework* de *QT* [40]. El error aparece sólo en las versiones más recientes y se espera que sea resuelto próximamente.

### 5.2 Análisis estadístico

Debido a lo comentado en el apartado 2.1, no se completaron en el momento de redactar esta memoria el apartado de análisis estadísticos ni la sincronización con Arduino. Sin embargo, se continuará trabajando para completar los análisis estadísticos. Este trabajo se encuentra ahora mismo en

desarrollo y será aplicado en la interfaz sobre la sección “Análisis estadístico” definida en el apartado 4.3.

En un primer lugar antes de poder aplicar un análisis estadístico a las imágenes de saturación, es necesario aplicar un preprocessamiento que consiste en lo siguiente:

1. Corrección del movimiento entre imágenes tomadas del mismo sujeto.
2. Normalización espacial entre imágenes de distintos sujetos (permite alinear las imágenes).
3. Suavizado espacial.

En el caso de estudio de esta aplicación no es necesario implementar las dos primeras operaciones. Sin embargo, sí se requiere la realización de un suavizado espacial. Este suavizado ya se encuentra implementado en el módulo de back-end o procesamiento mediante un filtro paso bajo, en este caso un desenfocado gaussiano [41] con tamaño de *kernel* seleccionable. Se puede ver en la figura 22 un ejemplo de la aplicación del filtro.

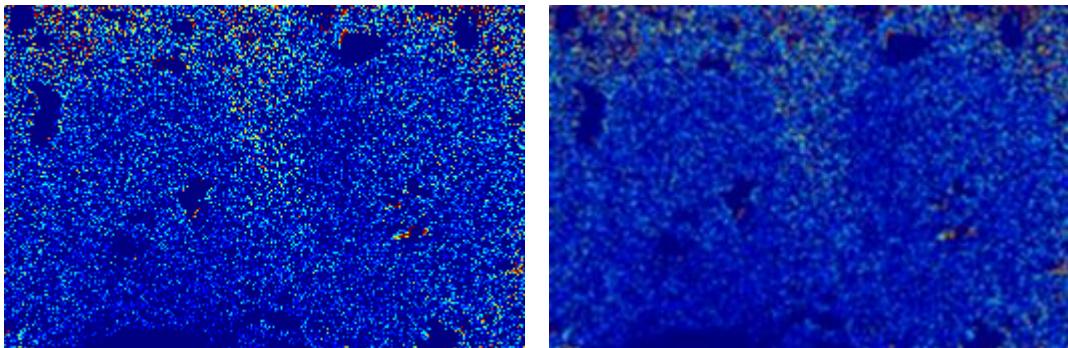


Figura 22: Imagen de saturación (Izquierda). Imagen de saturación filtrada con un *kernel* de tamaño  $3 \times 3$  y sigma 1.0. Mismo mapa de color que el aplicado a la imagen de saturación de la figura 9.

Una vez preprocessadas todas las imágenes se pueden dividir en distintos grupos dependiendo de los diferentes estados de activación o no activación sobre los que se hayan tomado, a partir de cada grupo se obtiene una única imagen al realizar una media de todas aquellas que pertenezcan a él.

Estas últimas son a continuación comparadas entre ellas. Existen varios métodos. Uno de los más sencillos consiste en realizar una simple substracción, sin embargo, esta técnica es muy sensible a movimientos de la imagen. Un mejor resultado se obtiene comúnmente aplicando una prueba de *t de Student* [42] a

cada pixel que permite comprobar cuál es la diferencia de ese pixel con respecto a su grupo (intragrupo) y con respecto a otros grupos (intergrupo). En función a esta variación se le asigna la conocida como *puntuación-t*. Los artefactos generados en la imagen se reducen en comparación la simple substracción [43].

Es este último apartado de agrupación y pruebas *t de Student* el que se encuentra actualmente en desarrollo y se pretende completar posteriormente a la entrega de la memoria.

### 5.3 Documentación

La documentación es parte fundamental de cualquier proyecto. En este apartado se ha recurrido a la herramienta Doxygen, que permite generar la documentación automáticamente a partir de los archivos de código fuente incluyendo además las descripciones añadidas a los métodos, clases, ... a través de los comentarios.

Se ha comprobado efectivamente que se puede generar con normalidad una documentación para el proyecto, pero no se encuentra disponible para su visualización, actualmente está incompleta a la espera de completar análisis estadístico del apartado anterior 5.2.

# Capítulo 6

## Conclusiones y líneas futuras

### 6.1 Conclusiones

Este proyecto se ha centrado en la creación de una interfaz para un sistema ya existente de un equipo médico. Se observa que la solución propuesta plantea una mejora en cuanto a dicho sistema actual.

Ya no es necesario depender de un software externo para las tareas que se realizan de forma común en los estudios pertinentes, el cuál además requiere de una licencia de pago.

Es posible aprovechar ahora un ritmo de trabajo más ordenado que permite acceder a las distintas funcionalidades rápidamente y lo que es más importante, modificar los parámetros sin necesidad de modificar el código sobre el que se construyen los scripts.

Es posible guardar las imágenes de saturación y basales y tratarlas con otras herramientas en caso de que se requiera.

La estructura creada basada en dos módulos ha demostrado funcionar de forma excelente a la hora de dividir las vistas y la lógica de la aplicación. Además, ha demostrado ser bastante robusta en tanto que los fallos de uno de los dos módulos no provocan el completo fallo del otro.

Por otro lado, el diseño final de la interfaz ha revelado ser una solución que aporta un espacio de trabajo adecuado y comprensible al usuario, dividiendo cada uno de los paquetes de trabajo en distintas zonas que siguen un orden natural.

El lenguaje y las tecnologías seleccionadas han sido correctas y además se destaca que, aunque se ha sufrido un retraso considerable al realizar la traducción de los scripts, ya existentes en *Matlab* a la aplicación final, ha sido una decisión adecuada en cuanto a centralizar todo el procesamiento de datos en una única región lógica

## 6.2 Líneas futuras

En cuanto a las líneas futuras se plantea el desarrollo de la sincronización del hardware de captura de imágenes a través de un pequeño dispositivo Arduino. Este dispositivo es un pequeño ordenador, de unos pocos centímetros, muy barato y versátil, programable en diferentes lenguajes (generalmente en *C* ).

La aplicación deberá buscar automáticamente o conectarse de forma manual con un Arduino conectado por USB. Se puede acceder a esta conexión a través de un puerto COM (la conexión por USB con Arduino emula este tipo de interfaz de hardware). Existe un amplio abanico de librerías que permite la adquisición de este tipo de puertos y la transmisión de información, sin ir más lejos, el *framework QT* ofrece una solución con su módulo *QtSerialPort*.

Una vez conectado con el dispositivo este servirá de puente para enviar una señal al hardware de captura de imágenes cuando el usuario decida completar la captura del basal. En este punto, la aplicación podrá automáticamente generar el basal en base a esa referencia sin necesidad de que el usuario introduzca manualmente la duración del basal.

Dependiendo del avance sobre el desarrollo actual presentado en el apartado 5.2, se tratará de implementar esta característica.

El dispositivo Arduino se puede programar a través del IDE de la misma compañía: ArduinoIDE [44].

# Capítulo 7

## Summary and Conclusions

### 7.1 Summary

During the course of this project, a new interface was developed in order to make it easier for the users to work in different studies. The aim was for it to be simpler and easy to use while at the same time providing all the tools needed.

The solution proposed fulfils the initial request and implements a multiplatform robust application with an interactive interface.

The user may be able to extract conclusions about the data based on the different processing functionality that its offered.

### 7.2 Conclusions

The principal objective was the development of a new interface for an already existing system for a medical equipment. The proposed solution brings up an improvement to that system.

The dependency on external software has been removed and a payed licence is not required anymore.

It is instantly possible to take advantage of a faster and better organized workflow which allows the user to access all available functionality faster, and more importantly, the user will be able to modify the parameters of the algorithms without the need to modify the source code where the scripts were originally written.

Now, saturation and basal images can be saved locally and they may be processed with other external tools if needed.

The application structure was built based on two separated modules, approach which has been shown to be very effective. Separating the views from the application logic have also shown to be a very robust alternative to common software. If one of the modules is compromised the other can continue to work normally, even though the user still needs both of them to work with.

On the other hand, the final interface design offers an excellent and easy to understand workspace to the user. Isolating different work packages from each other in different zones following a natural order.

The programming languages and technologies adopted have been adequate. It is of great significance to highlight that after the delay caused by the translation of the *Matlab*'s scripts to the application language, it was indeed a good decision to join together all the data processing units in a single space.

# Capítulo 8

## Presupuesto

### 8.1 Coste del proyecto

Se han desarrollado una serie de fases de análisis, diseño e implementación, que de acuerdo a la tabla 1, se pueden diferenciar claramente en el cronograma presentado.

Aunque la fase de diseño se haya realizado de forma iterativa a lo largo de todo el proyecto, se refleja en la tabla 3 de presupuesto como una única fase.

Por otro lado, en la fase de análisis fue necesario estudiar los diferentes scripts de *Matlab* siendo este un programa con una licencia de pago, se recurrió inicialmente a ejecutar los scripts durante las tareas de diseño en una licencia de prueba de dicho software, sin embargo, se ha incluido el precio de una licencia estándar de *Matlab* en la tabla de presupuesto. Se ha considerado un coste de 20€/h para cada una de las fases.

Paquetes	Duración (h)	Coste (€)
Análisis	40 h	800 €
Diseño	40 h	800 €
Implementación	190 h	3800 €
Licencia <i>Matlab</i>	-	800 €
Total	270 h	6200 €

Tabla 3: Presupuesto

# Capítulo 9

## Bibliografía

1. *Begoña Azkarate, Pedro Morrondo, Angel Mendia, Pilar Marco*  
Monitorización del sistema nervioso central. Servicio de Medicina Intensiva. Hospital Aránzazu. San Sebastián. País Vasco. España. CIMC 2000  
<https://www.uninet.edu/cimc2000/cursos/cur1/Begona/Begona.htm>
2. *Jech R.* (2008)  
Functional Imaging of Deep Brain Stimulation: fMRI, SPECT, and PET. In: Tarsy D., Vitek J.L., Starr P.A., Okun M.S. (eds) Deep Brain Stimulation in Neurological and Psychiatric Disorders. Current Clinical Neurology. Humana Press
3. *Fernando Lopes de Silva*  
EEG and MEG: Relevance to Neuroscience, Neuron, Volume 80, Issue 5, 2013,  
Pages 1112-1128, ISSN 0896-6273  
<https://doi.org/10.1016/j.neuron.2013.10.017>
4. *Balardin, J.B.m Zimeo Moraes, G. A., Furucho, R. A., Trambaiolli, L., Vanzella, P., Biasoli, C., & Sato (2017)*  
Imaging Brain Function with Functional Near-Infrared Spectroscopy in Unconstrained Environments. Frontiers in Human Neuroscience, 11, 258.  
<http://doi.org/10.3389/fnhum.2017.00258>
5. *Poldrack, R. A. (2017)*  
Region of interest analysis for fMRI. Social Cognitive and Affective Neuroscience, 2(1), 67 · 70  
<http://doi.org/10.1093/scan/nsm006>
6. *Diffraction Limited*  
Dark Frame Calibration  
[https://diffractionlimited.com/help/maximdl/Dark\\_Frame\\_Calibration.htm](https://diffractionlimited.com/help/maximdl/Dark_Frame_Calibration.htm)

7. Matlab

<https://es.mathworks.com/products/matlab.html>

8. MVC

<http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/mvc.html>

9. Eigen

[http://eigen.tuxfamily.org/index.php?title=Main\\_Page](http://eigen.tuxfamily.org/index.php?title=Main_Page)

10. DSPFilters

<https://github.com/vinniefalco/DSPFilters>

11. Dygraphs

<http://dygraphs.com/>

12. PIXIJS

<http://www.pixijs.com/>

13. QT

<https://www.qt.io/>

14. JQuery

<https://jquery.com/>

15. Adobe Photoshop

<https://www.adobe.com/es/>

16. Gimp

<https://www.gimp.org/>

17. Adobe Illustrator

<https://www.adobe.com/es/>

18. QT Creator

<https://www.qt.io/>

19. Flaticon

<https://www.flaticon.com/>

20. Gitlab

<https://about.gitlab.com/>

21. Git

<https://git-scm.com/>

22. MTuner

<https://github.com/milostosic/MTuner>

23. Paletton

<http://www.paletton.com/#uid=1000u0kllllaFw0g0qFqFg0w0aF>

24. Canva

[https://www.canva.com/es\\_es/](https://www.canva.com/es_es/)

25. Atom

<https://atom.io/>

26. Chrome

<https://www.google.es/chrome/index.html>

27. JSLint

<https://www.jslint.com/>

28. SonarQube

<https://www.sonarqube.org/>

29. SASS

<https://sass-lang.com/>

30. PUG

<https://pugjs.org/api/getting-started.html>

31. Doxygen

<http://www.stack.nl/~dimitri/doxygen/>

32. Bootstrap

<https://getbootstrap.com/>

33. *Wikipedia*

Comma-separated values

[https://en.wikipedia.org/wiki/Comma-separated\\_values](https://en.wikipedia.org/wiki/Comma-separated_values)

34. *Mary S. Ahn, Janis L. Breeze, Nikos Makris, David N. Kennedy, Steven M. Hodge, Martha R. Herbert, Larry J. Seidman, Joseph Biederman, Verne S. Caviness, Jean A. Frazier*

Anatomic brain magnetic resonance imaging of the basal ganglia in paediatric bipolar disorder

Journal of Affective Disorders, Volume 104, Issues 1 · 3, 2007, Pages 147-154

35. *University of Missouri (2014)*

“Logo color affects consumer emotion toward brands, MU study finds”

<https://research.missouri.edu/news/story.php?390>

36. *99designs*

“El negocio del color: el diseño de logo”

<https://99designs.es/logo-design/business-of-color>

37. *Google*

Cards, Material Design

<https://material.io/design/components/cards.html>

38. *WACG*

<https://www.w3.org/TR/WCAG20/>

39. *W3C*

<https://www.w3c.es/>

40. *QT*

*QWebEnginePage* contents size is resize slowly when the window is resizing

<https://bugreports.qt.io/browse/QTBUG-58324>

41. *Wikipedia*

Gaussian Blur

<https://en.wikipedia.org/wiki/Gaussian.blur>

42. *College of Saint Benedict, Saint John's University*

Student's t-Tests

<http://www.physics.csbsju.edu/stats/t-test.html>

43. *Stuart Clare (2006)*

Functional MRI : Methods and Applications, chapter 6.3

<https://users.fmrib.ox.ac.uk/~stuart/thesis/>

44. *Arduino*

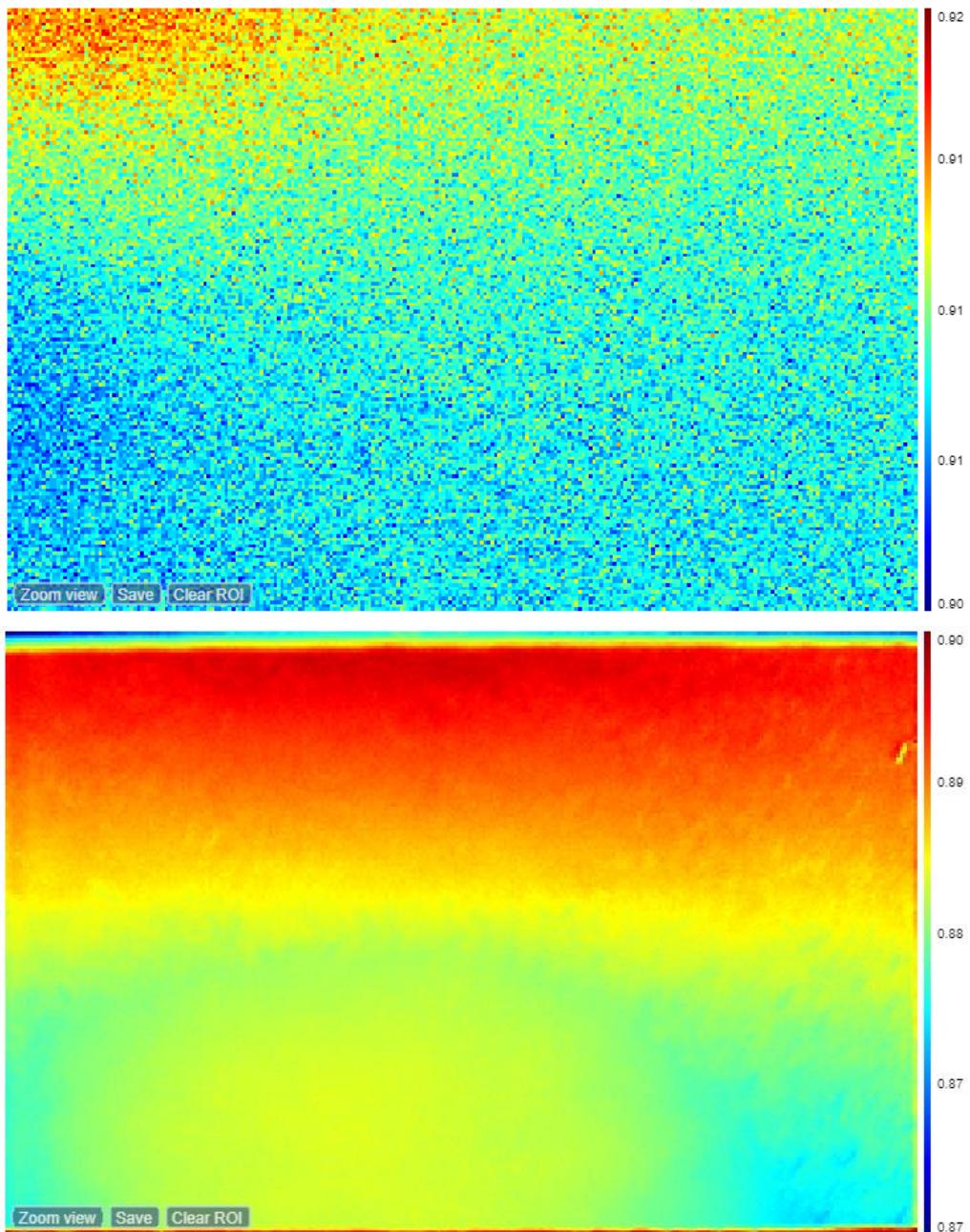
Arduino IDE

<https://www.arduino.cc/en/main/software>

# Capítulo 10

## Anexos

### 10.1 *Frames de calibración*



Figuras 23 y 24: Ejemplo de *frame dark* (arriba) y *gain* (abajo). Explicados en el 1.4.

## 10.2 Evolución de la interfaz



Figuras 25 y 26: Desarrollo de la interfaz inicial (arriba) y posterior (abajo). El desarrollo final se encuentra en el capítulo 4.

## 10.3 Ejemplos de imágenes de saturación

Las siguientes figuras muestran imágenes de saturación del experimento correspondiente al archivo de prueba.

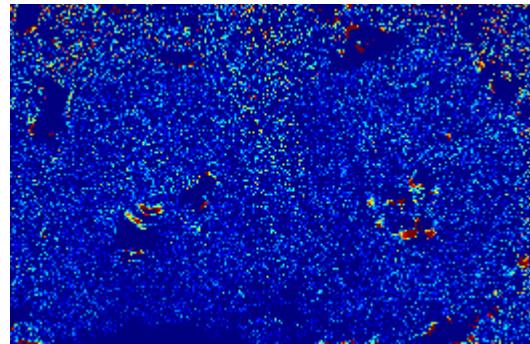
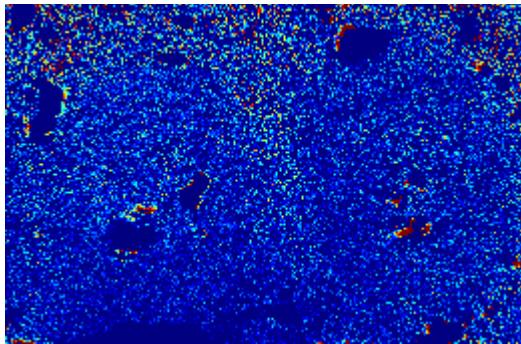


Figura 27 (Izquierda): Imagen de saturación, correspondiente al *frame* 1000.

Figura 28 (Derecha): Imagen de saturación, correspondiente al *frame* 2000.

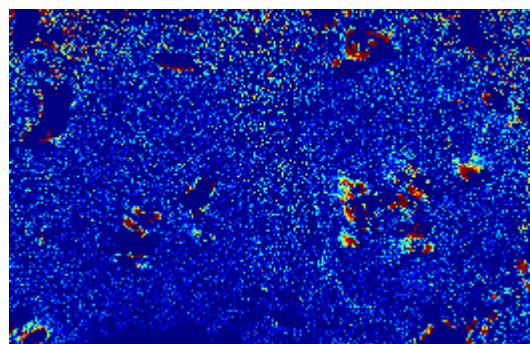
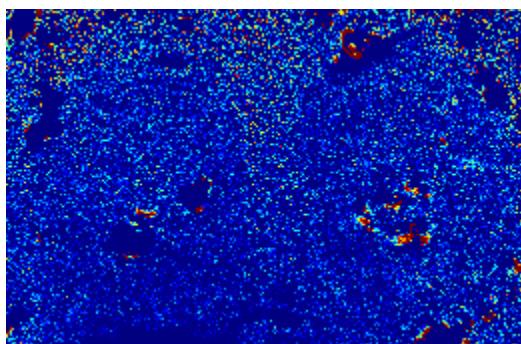


Figura 29 (Izquierda): Imagen de saturación, correspondiente al *frame* 3000.

Figura 30 (Derecha): Imagen de saturación, correspondiente al *frame* 7000.