****

**UNIVERSIDAD DE OVIEDO**

**ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN**

**INGENIERO TÉCNICO DE TELECOMUNICACIÓN,**

**ESPECIALIDAD TELEMÁTICA**

|  |
| --- |
| **ÁREA DE TECNOLOGÍA ELECTRÓNICA** |

|  |  |
| --- | --- |
| **PROYECTO FIN DE CARRERA Nº** | **3154275** |

**"DESARROLLO DE UN TECLADO VIRTUAL CONTROLADO MEDIANTE SEÑALES ELECTROOCULOGRÁFICAS"**

**AUTOR: D. DAVID YANGÜELA CARBAJOSA**

**TUTOR: D. FRANCISCO JAVIER FERRERO MARTÍN**

**FECHA: JULIO 2016**

**MEMORIA PRESENTADA POR**

**D. DAVID YANGÜELA CARBAJOSA**

**PARA OPTAR AL TÍTULO DE**

**INGENIERO TÉCNICO DE TELECOMUNICACIÓN,**

**ESPECIALIDAD TELEMÁTICA**

**INDICE**

[**0. PRÓLOGO 7**](#_Toc454465728)

[**1. INTRODUCCIÓN 8**](#_Toc454465729)

[**1.1. Objetivos y alcance del proyecto 8**](#_Toc454465730)

[**2. ANATOMÍA DEL OJO HUMANO 9**](#_Toc454465731)

[**2.1. El ojo en el sistema visual 9**](#_Toc454465732)

[**2.2. Anatomía y funcionamiento del ojo 10**](#_Toc454465733)

[**2.3. Clasificación de los movimientos oculares 13**](#_Toc454465734)

[**2.4. Modelo eléctrico del ojo 15**](#_Toc454465735)

[**3. LA ELECTROOCULOGRAFÍA 17**](#_Toc454465736)

[**3.1. Tecnologías de seguimiento y medida del movimiento ocular 17**](#_Toc454465738)

[**3.1.1. Seguimiento mediante sensorizado invasivo (Bobina de búsqueda) 17**](#_Toc454465739)

[**3.1.2. Seguimiento mediante sensorizado no invasivo (Videooculograma) 17**](#_Toc454465740)

[**3.1.3. Seguimiento mediante potenciales eléctricos (Electrooculograma) 18**](#_Toc454465741)

[**3.2. Fundamentos de la electrooculografía 19**](#_Toc454465742)

[**3.3. Aplicaciones 20**](#_Toc454465743)

[**3.4. Equipos comerciales 21**](#_Toc454465744)

[**4. PLATAFORMA HARDWARE 24**](#_Toc454465745)

[**4.1. Adquisición de señales 24**](#_Toc454465746)

[**4.1.1. Disposición de los electrodos 25**](#_Toc454465747)

[**4.1.2. Disposición alternativa de los electrodos 26**](#_Toc454465748)

[**4.2. Etapa analógica 26**](#_Toc454465749)

[**4.3. Etapa digital 28**](#_Toc454465750)

[**4.4. Comunicación inalámbrica 29**](#_Toc454465751)

[**5. PLATAFORMA SOFTWARE 31**](#_Toc454465753)

[**5.1. Lenguaje y bibliotecas utilizadas 31**](#_Toc454465754)

[**5.2. Procesamiento digital de la señal 33**](#_Toc454465755)

[**5.2.1. Lectura de datos de la plataforma hardware 33**](#_Toc454465756)

[**5.2.2. Mapeado de los datos 33**](#_Toc454465757)

[**5.2.3. Tratamiento de los datos 34**](#_Toc454465758)

[**5.2.4. Toma de decisiones 35**](#_Toc454465759)

[**5.2.5. Modularidad y concurrencia 36**](#_Toc454465760)

[**5.2.6. Comunicación entre módulos 37**](#_Toc454465761)

[**5.3. Interfaz de usuario 38**](#_Toc454465762)

[**5.3.1 Sistema de escritura 39**](#_Toc454465763)

[**5.3.2 Funcionalidades adicionales de la aplicación 43**](#_Toc454465764)

[**5.3.3 Configuración 45**](#_Toc454465765)

[**5.4 Puesta en marcha: instalación y funcionamiento de la aplicación 46**](#_Toc454465766)

[**6. PRUEBAS 48**](#_Toc454465767)

[**6.1. Pruebas con parámetros de configuración 48**](#_Toc454465768)

[**6.2. Pruebas de velocidad de escritura 49**](#_Toc454465769)

[**6.3. Pruebas de interoperabilidad 50**](#_Toc454465770)

[**7. CONCLUSIONES 51**](#_Toc454465771)

[**8. POSIBLES MEJORAS DEL SISTEMA 53**](#_Toc454465776)

[**8.1. Uso de la transformada wavelet para la detección del parpadeo 53**](#_Toc454465777)

[**8.2. Aprendizaje automático (Machine learning) 53**](#_Toc454465778)

[**8.3. Mejoras y nuevas funcionalidades de la aplicación 54**](#_Toc454465779)

[**9. REFERENCIAS 55**](#_Toc454465780)

[**A. ANEXO I: CÓDIGO FUENTE DE LA APLICACIÓN 57**](#_Toc454465781)

[**A.1. Clase principal (mainwindow) 57**](#_Toc454465782)

[**A.2. Lectura de datos (btreaderthread) 67**](#_Toc454465783)

[**A.3. Mapeado de datos (dataparserthread) 70**](#_Toc454465784)

[**A.4. Tratamiento de datos (datatreatmentthread) 72**](#_Toc454465785)

[**A.5. Toma de decisiones (decisionthread) 74**](#_Toc454465786)

[**A.6. Escritura de letras (writting) 78**](#_Toc454465787)

[**A.7. Escritura de palabras (words) 86**](#_Toc454465788)

[**A.8. Menú de configuración (btsettingsdialog) 94**](#_Toc454465789)

[**A.9. Recursos de la aplicación (application.qrc) 98**](#_Toc454465790)

[**A.10. Ficheros de configuración (config.ini, mapper.ini) 99**](#_Toc454465791)

[**B. ANEXO II: PLANIFICACIÓN DEL PROYECTO 100**](#_Toc454465792)

# ÍNDICE DE FIGURAS

**Figura 2.1. El sistema visual 9**

**Figura 2.2. Órganos anexos del sistema visual 9**

**Figura 2.3. Sección del ojo humano 10**

**Figura 2.4. Músculos extraoculares 11**

**Figura 2.5. Ejemplo de formación de la imagen en la retina 12**

**Figura 2.6. Dipolo eléctrico córnea-retina 16**

**Figura 3.1. Bobina de búsqueda 17**

**Figura 3.2. Videooculograma 18**

**Figura 3.3. Máquina EOG 18**

**Figura 3.4. Gorro con sensores 18**

**Figura 3.5. Relación entre la posición angular del ojo y el potencial reflejado en el electrooculograma 19**

**Figura 3.6. Equipo de BioMuse 21**

**Figura 3.7. Equipo de CyberLink 22**

**Figura 3.8. Equipo Irisbond 22**

**Figura 3.9. Equipo BlueGain EOG 23**

**Figura 4.1. Plataforma hardware utilizada en el desarrollo del trabajo 24**

**Figura 4.2. Electrodos foam 24**

**Figura 4.3. Disposición de los electrodos 25**

**Figura 4.4. Señal EOG obtenida mediante la plataforma hardware 25**

**Figura 4.5. Disposición alternativa de los electrodos 26**

**Figura 4.6. Diagrama de bloques de la etapa analógica 26**

**Figura 4.7. Señales obtenidas en el canal horizontal a la salida de la etapa analógica 27**

**Figura 4.8. Señales obtenidas en el canal vertical a la salida de la etapa analógica 27**

**Figura 4.9. Diagrama de bloques de la plataforma hardware 28**

**Figura 4.10. Módulo Bluetooth WT12 de BlueGiga 29**

**Figura 5.1. Entorno de desarrollo multiplataforma Qt Creator 3.5.1 32**

**Figura 5.2. Configuración de depuración y compilación 32**

**Figura 5.3. Diagrama de flujo del módulo de lectura 33**

**Figura 5.4. Diagrama de flujo del módulo de mapeado 34**

**Figura 5.5. Diagrama de flujo del módulo de tratamiento 35**

**Figura 5.6. Diagrama de flujo del módulo de decisión 36**

**Figura 5.7. Mecanismo de señales y slots 37**

**Figura 5.8. Interfaz gráfico de usuario de la aplicación. Menú principal 38**

**Figura 5.9. Botones de escritura y lectura 39**

**Figura 5.10. Sistema de crucetas, etapa 1 40**

**Figura 5.11. Sistema de crucetas, etapa 2 40**

**Figura 5.12. Sistema de crucetas, etapa 3 41**

**Figura 5.13. Diagrama de flujo del módulo de escritura 42**

**Figura 5.14. Menú introducir palabra 43**

**Figura 5.15. Funcionalidades adicionales 43**

**Figura 5.16. Mensaje copiar a portapapeles 44**

**Figura 5.17. Ficheros de texto generados por la aplicación 44**

**Figura 5.18. Mensaje guardar texto 45**

**Figura 5.19. Diálogo de configuración 45**

**Figura 5.20. Fichero de configuración config.ini 45**

**Figura 5.21. Fichero de configuración mapper.ini 45**

**Figura 5.22. Arborescencia de la aplicación 46**

**Figura B.1. Diagrama de Gantt 100**

**Figura B.2. Detalle de tiempos diagrama de Gantt 100**

# ÍNDICE DE TABLAS

**Tabla 2.1. Relación entre los músculos extraoculares y su movimiento asociado 12**

**Tabla 4.1. Resultados digitalizados del canal vertical 30**

**Tabla 6.1. Prueba de escritura 1 49**

**Tabla 6.2. Prueba de escritura 2 49**

# **0. PRÓLOGO**

Según un informe mundial sobre la discapacidad [1], realizado por la Organización Mundial de la Salud (OMS), más de mil millones de personas viven en todo el mundo con alguna forma de discapacidad; de ellas, casi 200 millones (casi el 3% de la población mundial) experimentan dificultades considerables en su funcionamiento diario.

Según se recoge en el estudio, en los años futuros, la discapacidad será un motivo de preocupación aún mayor, pues su prevalencia está aumentando debido a que la población mundial está cada vez más envejecida (consecuencia directa del aumento de la esperanza de vida experimentado entre los siglos XX y XXI), siendo el riesgo de discapacidad superior entre los adultos mayores, unido todo ello al aumento mundial de enfermedades crónicas.

En el caso concreto de las personas con discapacidades motrices graves (como distrofia muscular, parálisis cerebral, lesiones en la espina dorsal, esclerosis...), presentan una problemática comunicacional que las aísla de su medio. El hecho de que no dominen la motricidad fina conduce a que no puedan tomar un lápiz para escribir ni presionar las pequeñas teclas de un teclado estándar de computador, impidiendo, entre otras cosas, la telecomunicación de estas personas con otras. En ocasiones estas personas, también presentan dificultades en el habla, motivos todos ellos que complican su acceso a la educación, al ocio y dificultan la comunicación con su entorno.

El objetivo de este informe es proporcionar datos destinados a la formulación de políticas y programas innovadores que mejoren la vida de las personas con discapacidades y faciliten la aplicación de la Convención de Naciones Unidas sobre los Derechos de las Personas con Discapacidad, que entró en vigor en mayo de 2008, y requiere, entre otros, que:

* **Artículo 9**. Los países deben identificar y eliminar los obstáculos y las barreras y asegurar que las personas con discapacidad puedan tener acceso a su entorno, al transporte, las instalaciones y los servicios públicos, y tecnologías de la información y las comunicaciones.
* **Artículo 19**. Las personas con discapacidad deben tener la opción de vivir de forma independiente, ser incluidas en la comunidad, elegir dónde y con quién vivir y tener acceso a servicios de apoyo en el hogar, en residencias y en la comunidad.
* **Artículo 20**. Debe promoverse la movilidad personal y la independencia, facilitando la movilidad personal asequible, la capacitación al respecto y el acceso a ayudas para la movilidad, aparatos, tecnologías de asistencia y asistencia personal.
* **Artículo 21**. Los países deben promover el acceso a la información, proporcionando la información prevista para el público en general en formatos y tecnologías accesibles, facilitando el uso del Braille, el lenguaje por señas y otras formas de comunicación.

# **INTRODUCCIÓN**

Este proyecto nace del interés de proporcionar a las personas con discapacidades motrices graves (miopatías y neuropatías como parálisis cerebral (PC), síndrome de enclaustramiento (LIS) o esclerosis lateral amiotrófica (ELA), entre otras) una herramienta de asistencia que facilite la comunicación mediante el uso de tecnologías de la información y las comunicaciones, otorgando así al usuario de un mayor grado de independencia y aumentando su calidad de vida.

Por lo general, el movimiento ocular es algo que rara vez se pierde en cualquier tipo de discapacidad, hecho por el cual muchas de las investigaciones y desarrollos llevados a cabo persiguiendo este interés, se han centrado en el estudio de los ojos con el fin de establecer un interfaz o mecanismo de comunicación y control consciente por parte de un usuario con una discapacidad motriz grave, que no está en disposición de poseerlo de otra manera debido a este hecho.

A día de hoy, existe el suficiente conocimiento del ojo humano como para hacer del mismo un mecanismo fiable para la comunicación, utilizando para ello, entre otras técnicas, la electrooculografía (EOG), que permite registrar y medir las señales bioeléctricas generadas por los movimientos oculares. Dichas señales pueden ser utilizadas con el fin de proporcionar un interfaz de señales conscientes con las que controlar diversos dispositivos software y hardware.

## **1.1. Objetivos y alcance del proyecto**

El objetivo principal de este proyecto es el diseño e implementación de un sistema de comunicación para personas con discapacidades motrices graves basado en señales electrooculográficas. Para ello, se deberán llevar a cabo los siguientes pasos:

* Estudio sobre la anatomía ocular y las señales bioeléctricas generadas por el movimiento de los ojos.
* Estudio y acondicionamiento de la tarjeta de adquisición de señales bioeléctricas.
* Desarrollo de un software multiplataforma a tiempo real que sirva de interfaz de comunicación entre usuario y computador. Estudio del lenguaje de programación y bibliotecas a utilizar.
* Desarrollo de un método de escritura eficiente y funcional que reduzca los tiempos de escritura respecto del teclado virtual convencional.
* Realización de pruebas para determinar la usabilidad del sistema de escritura y software.

No se encuentra entre los objetivos de este proyecto el desarrollar un conjunto hardware y software totalmente funcional y comerciable de cara al usuario final (debido principalmente a la limitación de recursos existente), sino tan solo un prototipo para su estudio, que será la base para una posible implementación real y comercialización.

# **ANATOMÍA DEL OJO HUMANO**

Siendo el objetivo del proyecto el desarrollo de un sistema de comunicación controlado mediante el movimiento de los ojos, es primordial comprender la anatomía y el funcionamiento de los mismos, especialmente en lo que concierne al plano óptico, muscular y eléctrico, pues de estos elementos tomaremos los parámetros necesarios para el diseño e implementación del sistema propuesto.

## **2.1. El ojo en el sistema visual**

El ojo es un órgano que, junto al cerebro, forma parte del sistema visual. Su función es la de captar las ondas electromagnéticas que conforman el espectro visual, y convertirlas en señales eléctricas para transmitir al cerebro a través del nervio óptico. El cerebro, por su parte, se encarga del procesamiento de esta información para interpretarla como las formas y colores que percibimos en última instancia (mediante procesos que a priori se escapan al propósito de este proyecto).

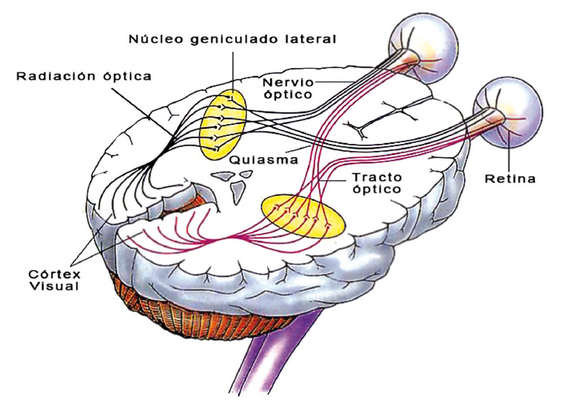


Figura 2.1. El sistema visual [2]

Además de estos órganos principales, el sistema visual cuenta con un conjunto de órganos anexos, entre cuyas funciones se encuentran la de proteger el ojo, mantener la hidratación y los electrolitos necesarios para su correcto funcionamiento o fijar la dirección de la mirada, como son los párpados y pestañas, las cejas, las glándulas lagrimales, el conducto nasolagrimal y los músculos externos.



Figura 2.2. Órganos anexos del sistema visual [3]

## **2.2. Anatomía y funcionamiento del ojo**

El ojo, o globo ocular, es una estructura esférica compleja de aproximadamente 2,5 centímetros de diámetro, con un marcado abombamiento sobre su superficie anterior. En su parte más exterior, está recubierto por tres capas esféricas concéntricas:

* Capa externa o **esclerótica**, cuya función es la de proteger y dar consistencia al globo ocular. Cubre aproximadamente cinco sextos de la superficie del ojo (exceptuando la parte anterior).
* Capa media o **úvea**, formada por tres regiones diferenciadas: la coroides, que reviste tres quintas partes de la superficie, el iris en la parte frontal, en el que se encuentra la pupila (que permite el paso de la luz) y el cuerpociliar, que une al iris con la coroides y produce el humor acuoso (un líquido fundamental para nutrir y oxigenar las estructuras oculares que carecen de aporte sanguíneo).
* Capa interna, donde se encuentra la **retina**, encargada de generar los impulsos nerviosos que serán transmitidos al cerebro para su procesado, en respuesta a los estímulos luminosos que recibe.

A su vez, el globo ocular se divide en tres partes diferenciadas por su composición y funciones dentro del ojo, la zona frontal o anterior, la zona interior y la zona posterior.

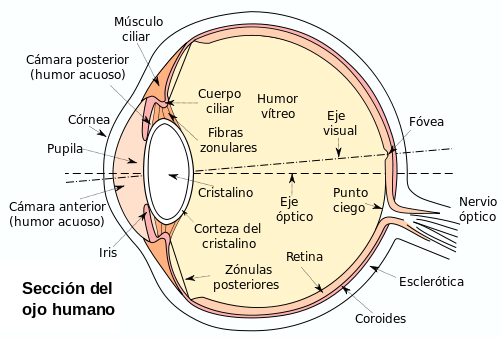


Figura 2.3. Sección del ojo humano [4]

En la parte frontal del ojo se encuentra la **córnea**, una membrana resistente compuesta por cinco capas, a través de la cual penetra la luz en el interior del ojo. Detrás de la córnea se sitúa la **cámara anterior**, que está llena de un fluido claro y húmedo llamado **humor acuoso**, que separa la córnea del cristalino y a su vez los nutre y oxigena, ya que estas estructuras carecen de aporte sanguíneo. El **cristalino** es una lente biconvexa constituida por un gran número de fibras transparentes dispuestas en capas, cuya función principal es la de enfocar objetos a diferentes distancias. Para ello se sirve del **músculo ciliar**, que tiene forma de anillo y rodea la estructura mediante unos ligamentos. También se encuentra en esta zona anterior del ojo el **iris,** una estructura circular pigmentada suspendida entre la córnea y el cristalino, que posee una abertura central de tamaño variable, llamada **pupila**. La variación del tamaño de la pupila depende de dos músculos que rodean sus bordes, el músculo esfínter del iris y el músculo dilatador del iris, los cuales disminuyen y aumentan, respectivamente, dicho tamaño, con el objeto de controlar la cantidad de luz que penetra en el ojo.

La parte interior del ojo está compuesta fundamentalmente por el **humor vítreo,**  un líquido gelatinoso y transparente, compuesto en un 99% por agua y el resto por pequeñas cantidades de cloro, sodio, glucosa, potasio, colágeno, ácido hialurónico y proteínas. Rellena el espacio comprendido entre la superficie interna de la retina y la cara posterior del cristalino, lo que constituye cuatro quintas partes del volumen total del ojo. Es más denso que el humor acuoso y permite al globo ocular mantener su forma esférica debido a su consistencia. Al igual que el humor acuoso, tampoco está irrigado por ningún vaso sanguíneo. Dentro del humor vítreo se pueden distinguir tres partes; el **conducto hialoideo**, vestigio de la etapa de desarrollo embrionario, el **cortex**, que corresponde a la porción periférica más densa, y el **vítreo central**, que posee una menor densidad. Cabe mencionar que, al contrario que el humor acuoso, el humor vítreo no se renueva ya que éste se forma durante la etapa de desarrollo embrionario, sino que contiene células fagocíticas que contribuyen a eliminar los deshechos celulares que pueden acumularse en su interior y mantener así su renovación.

En la parte posterior del ojo se encuentra la **retina**, una capa compleja compuesta principalmente por células nerviosas sensibles a la luz. Estas células tienen forma de **bastones** y **conos**, las primeras permiten distinguir el blanco, el negro y la escala de grises, mientras que las segundas hacen posible la visión de los colores. El área de la retina donde se enfocan los rayos luminosos se conoce como **fóvea**, y se encuentra especialmente capacitada para la visión del color, puesto que se compone de células con forma de conos, mientras que según nos alejamos del área sensible, van apareciendo las células con forma de bastones. El **nervio óptico** entra en el globo ocular por debajo y algo inclinado hacia el lado interno de la fóvea central, originando en la retina una pequeña mancha redondeada llamada **disco óptico**.

Ya en la parte externa al globo ocular, es necesario hablar de los **músculos extraoculares**, seis músculos responsables de los movimientos oculares: recto externo, recto interno, recto superior, recto inferior, oblicuo superior y oblicuo inferior.

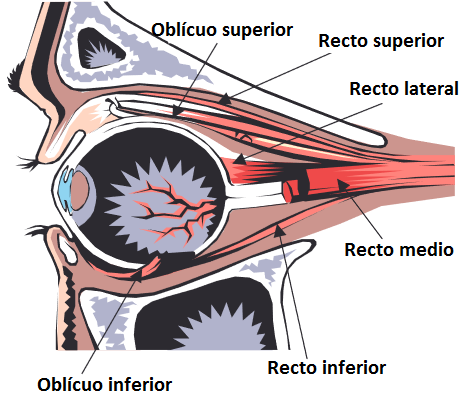


Figura 2.4. Músculos extraoculares [5].

Estos músculos, a su vez, son inervados por los siguientes **nervios craneanos**:

* Nervio oculomotor o motor ocular común (III par craneal): músculos recto interno, recto superior, recto inferior y oblicuo inferior.
* Nervio troclear o patético (IV par craneal): músculo oblicuo superior.
* Nervio abducente o motor ocular externo (VI par craneal): músculo recto externo.

Los movimientos de los ojos son entonces, consecuencia directa de los músculos externos del ojo, movidos mediante los impulsos recibidos a través de estos nervios, como se recoge en la siguiente tabla.

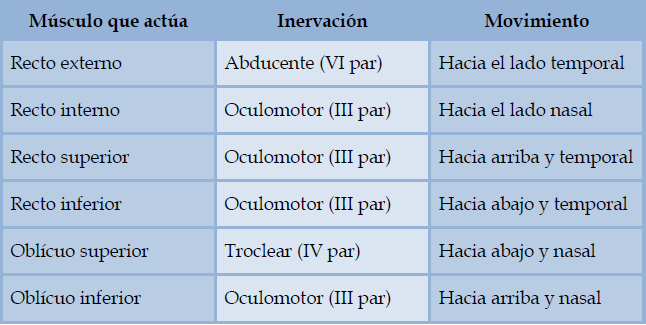


Tabla 2.1. Relación entre los músculos extraoculares y su movimiento asociado

En cuanto a su funcionamiento, el ojo recoge las perturbaciones electromagnéticas que constituyen los estímulos luminosos procedentes del entorno. La luz atraviesa los medios transparentes de la parte anterior hasta llegar al cristalino, que forma en la retina una imagen invertida de los objetos que enfoca, como se observa en la figura 2.5.

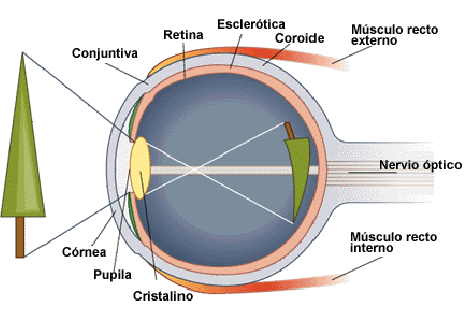


Figura 2.5. Ejemplo de formación de la imagen en la retina [6]

Gracias al proceso conocido como **acomodación**, el cristalino se aplana o redondea con el objetivo de enfocar el objeto. En condiciones de relajación el cristalino está plano y el ojo es capaz de percibir los objetos lejanos, mientras que para enfocar los más cercanos es necesaria la contracción del músculo ciliar, redondeándose la lente en consecuencia.

Debido a la estructura nerviosa de la retina, las células con forma de conos están conectadas de forma individual con otras fibras nerviosas, de modo que los estímulos que llegan a cada una de ellas se reproducen, permitiendo así distinguir los pequeños detalles. Por otro lado, las células con forma de bastones se conectan en grupo y responden a los estímulos que alcanzan un área más general (es decir, los estímulos luminosos), pero no tienen capacidad para separar los pequeños detalles de la imagen visual. La diferente localización y estructura de estas células conducen a la división del campo visual del ojo en una pequeña región central de gran agudeza y en las zonas que la rodean, de menor agudeza y con una gran sensibilidad a la luz. Así, durante la noche, los objetos confusos se pueden ver por la parte periférica de la retina, cuando son invisibles para la fóvea central. Por último, los impulsos eléctricos generados por estas células con forma de conos y bastones, son enviados al cerebro a través del nervio óptico para su procesamiento e interpretación.

Los músculos oculares externos, por su parte, son los encargados de fijar la dirección de la mirada hacia cualquier punto del campo visual con gran precisión, campo visual que se estima en alrededor de cien mil puntos distintos en un ángulo de visión de 60 grados en cada eje (30 en cada dirección). Los músculos de los dos ojos funcionan de manera simultánea, pudiendo apuntar a un mismo punto lejano con la misma posición angular, o converger para enfocar un objeto más cercano. Estos movimientos oculares contribuyen también a la estimación visual del tamaño y la distancia de los objetos, así como a la estabilización de la imagen en la retina, como se pasa a detallar en el apartado 2.3.

## **2.3. Clasificación de los movimientos oculares**

Los movimientos oculares son una combinación de rotaciones en torno a un eje horizontal, rotaciones en torno a un eje vertical y torsiones en torno al eje de la mirada. Las distintas combinaciones de estos movimientos básicos dan lugar a los movimientos característicos de los ojos. Cualquier movimiento característico de los ojos puede expresarse en función de uno o una combinación de los siguientes:

* **Movimientos compensatorios**

Son un tipo de movimiento inconsciente muy rápido (unos 16 milisegundos), generado mediante el reflejo vestíbulo-ocular (VOR) para compensar los movimientos realizados por la cabeza y asegurar la estabilidad de la imagen reflejada en la retina. De esta forma, cuando se realiza un movimiento con la cabeza en un sentido, el VOR ejecuta un movimiento ocular de sentido opuesto y misma desviación asegurando el enfoque de una imagen estable.

* **Movimientos sacádicos**

Este tipo de movimientos constituyen uno los más característicos del ojo humano, siendo fundamentalmente voluntarios (aunque también los hay involuntarios). Los movimientos sacádicos voluntarios permiten recorrer diversas zonas del campo visual sin mover la cabeza para recoger toda la información disponible en el campo visual, es decir, permite dirigir la mirada intencionadamente a diferentes puntos del campo visual. Los involuntarios por su parte, son aquellos característicos de la fase REM del sueño.

El objetivo principal de este tipo de movimientos es el de enfocar la imagen visual en la fóvea, que es la región de la retina que presenta una mayor agudeza visual (como se describe en el apartado 2.2). Durante esta transición entre objetivos, ocurre un fenómeno momentáneo de pérdida de información denominado supresión sacádica. Debido a este fenómeno es por lo que no es posible percibir una imagen dinámica de los ojos al intentar observarla en una superficie reflectante.

A lo largo de un día, son ejecutados alrededor de 230.000 movimientos sacádicos de manera inconsciente. Los ojos sólo permanecen relativamente quietos para enfocar una zona concreta de una escena durante periodos de tiempo muy breves, de 200 a 350 milisegundos de duración.

* **Micro-movimientos de fijación**

Se trata de unos minúsculos movimientos de naturaleza involuntaria y amplitud inferior a 1 grado de ángulo visual, que procuran ubicar la imagen reflejada en la fóvea de la forma más precisa posible. Estos movimientos se dividen en “drifts” y “flicks”. Los primeros son movimientos lentos de apenas 0,1 grados por segundo. Los segundos pueden ser considerados como movimientos micro-sacádicos muy rápidos, que pueden alcanzar 1 grado de amplitud en intervalos de entre 20 y 30 milisegundos.

* **Movimientos de seguimiento o persecución lenta**

Este tipo de movimiento se produce de forma involuntaria y coordinada entre ambos ojos y tiene por finalidad seguir estímulos visuales que se desplazan lentamente. Su velocidad oscila entre 1 y 30 grados por segundo. Su finalidad no es otra que la de estabilizar la imagen visual en movimiento sobre la retina; sin embargo, es posible ejercer control sobre los mismos mediante entrenamiento.

* **Movimientos de vergencia**

Alternativamente a los movimientos de persecución, los de vergencia implican el movimiento de los ojos en direcciones opuestas, y su finalidad es proyectar la imagen sobre la retina de los dos ojos y obtener una única imagen fusionada, combinando las imágenes generadas en ambas retinas, y permitiendo por tanto mantener una percepción del espacio en tres dimensiones. Los movimientos de vergencia pueden ser de:

* Convergencia (hacia la nariz): en los casos en que el objeto a visualizar se acerca al sujeto.
* Divergencia (hacia el exterior): en los casos en los que el objeto a visualizar se aleja del sujeto.

En ambos casos, las velocidades alcanzadas pueden llegar a ser de 10grados por segundo con una amplitud que alcanza los 15grados de ángulo visual.

* **Nistagmo ocular**

Este movimiento sigue un patrón coordinado y se caracteriza por una oscilación rítmica o alternante de los ojos con el objetivo de percibir cualquier objeto en movimiento del campo visual que contenga patrones repetidos. Consta de dos fases, una de ida o fase lenta y otra de retorno o fase rápida. En la fase lenta los ojos se mueven para focalizar un objeto en la retina (de manera similar al observado en los movimientos de seguimiento o persecución lenta) mientras que la fase rápida se caracteriza por un movimiento de vuelta o retorno, similar a un salto sacádico.

## **2.4. Modelo eléctrico del ojo**

En el año 1848 el fisiólogo alemán Emil du Bois-Reymond, observó tras diversos estudios [7] que el ojo actúa como un **dipolo eléctrico** en el que la córnea se encuentra cargada positivamente, mientras que la retina está cargada negativamente. Los movimientos oculares se corresponden con las variaciones de potencial de este dipolo, fundamento sobre el cual se sostiene la electrooculografía.

Cabe notar que este potencial córnea-retina es en parte generado debido a la respuesta eléctrica de los conos y bastones situados en la retina (apartado 2.2), por lo que la condiciones ambientales de luminosidad influirán en las señales bioeléctricas, variando los potenciales medidos en las mismas condiciones de movimiento.

* **Potencial córnea-retina. El dipolo ocular**

Existe una diferencia de potencial de hasta 1 mV entre la córnea y la retina del ojo, que varía en función de la amplitud de movimiento de los ojos. Este potencial tiene importantes variaciones diurnas y con el nivel de adaptación a la luz, pudiéndose necesitar entre 30 y 60 minutos de adaptación del individuo a las condiciones lumínicas.

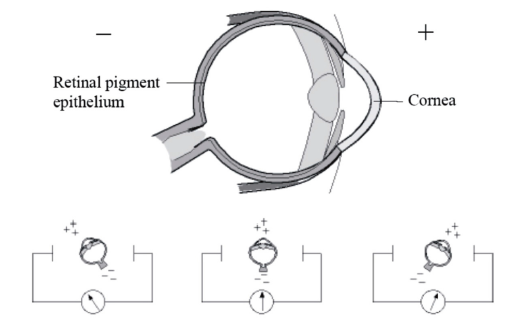


Figura 2.6. Dipolo eléctrico córnea-retina [6].

La señal bioeléctrica resultante de la variación de esta diferencia de potencial posee, por tanto, una **Amplitud y Frecuencia** que depende de cada persona y de las condiciones ambientales, por lo que a priori, no es posible establecer unos parámetros generales que definan estas señales, aunque varios autores han sugerido algunos:

* John Enderle sugiere que la Amplitud se encuentra en el rango de 50 a 3500 μV y la Frecuencia varía desde 1 a 50 Hz [8].
* A. S. Cohen asegura, por su parte, que la Frecuencia puede alcanzar los 100 Hz y la Amplitud presenta un mayor rango de variación, desde los 10 μV hasta los 5 mV [9].

Por norma general, la Amplitud de la señal es directamente proporcional al ángulo de rotación de los ojos, siendo la sensibilidad del orden de 20 μV por grado de movimiento.

La **impedancia** medida entre electrodos colocados en los cantos exteriores de los dos ojos varía con la posición de los ojos. La variación de la resistencia se asocia tanto a la naturaleza anisotrópica de las características eléctricas de los tejidos oculares como a la forma no perfectamente esférica del ojo, que supone un cambio en el camino resistivo entre los dos ojos cuando se produce una variación de posición en los mismos.

# **LA ELECTROOCULOGRAFÍA**

## La electrooculografía (EOG) es la técnica que se encarga de la medición y el estudio de los parámetros bioeléctricos descritos en el apartado 2.4, aunque no es el único método para obtener los parámetros del movimiento ocular.

## **3.1. Tecnologías de seguimiento y medida del movimiento ocular**

Existen varios métodos a la hora de detectar el movimiento ocular, basados en diferentes tipos de tecnologías, pudiéndose clasificar en tres tipos de seguimiento:

### **3.1.1. Seguimiento mediante sensorizado invasivo (Bobina de búsqueda)**

Se trata de una técnica que mide las variaciones del flujo magnético producidas con el movimiento de los ojos. Para ello, se coloca un detector de bucle de inducción en la córnea (puede colocarse solo en un ojo o en los dos), mientras que los imanes colocados alrededor del ojo crean campos magnéticos alternos que generan corrientes eléctricas en las bobinas (sensores) por inducción electromagnética. La polaridad y la amplitud de estas corrientes varían con la dirección y el desplazamiento angular del ojo, pudiéndose determinar así la posición del mismo en cada momento. Esta técnica es la más invasiva de todas, y es muy incómoda para el usuario.



Figura 3.1. Bobina de búsqueda [10]

### **3.1.2. Seguimiento mediante sensorizado no invasivo (Videooculograma)**

El videooculograma (VOG), o “*eye tracking”* como se conoce en inglés, consiste en hacer un seguimiento del movimiento de los ojos en relación con la cabeza, con el fin de evaluar la dirección donde se fija la mirada. Para ello se utiliza una cámara de vídeo que apunta al ojo y, mediante un algoritmo computacional, se calcula la posición del iris y la pupila para determinar la dirección de la mirada, así como el número de parpadeos.

Esta técnica tiene la ventaja de ser la menos invasiva, ya que no necesita de la conexión de ningún sensor para la captación del movimiento ocular. Por otra parte, la principal desventaja frente a las demás, es el alto coste de implementación, ya que se necesitan aparatos de alta precisión que encarecen el precio final de este sistema.



Figura 3.2. Videooculograma [11]

### **3.1.3. Seguimiento mediante potenciales eléctricos (Electrooculograma)**

La electrooculografía consiste, como se ha descrito en la introducción, en la medición de la diferencia de potencial que generan los movimientos oculares entre el dipolo córnea-retina para determinar la dirección de la mirada. Esta técnica resulta mucho menos invasiva que la bobina de búsqueda, puesto que la última necesita de la introducción de una bobina metálica en el interior del ojo, con la incomodidad implícita para el usuario, mientras que los costes de implementación serían similares para ambas. Respecto al videooculograma, por contra, el electrooculograma es mucho más barato de implementar, puesto que solo es necesario un módulo de captación de señales y cinco sensores, frente a los sofisticados dispositivos de captación de imágenes del VOG. La técnica del electrooculograma es poco más invasiva que la del videooculograma, aunque no supone ninguna dificultad ni incomodidad añadida para el usuario, máxime cuando existen ya en el mercado dispositivos de captación de las señales incorporados a gafas, antifaces o gorros.

Figura 3.3. Máquina EOG [12] Figura 3.4. Gorro con sensores [13]

## **3.2. Fundamentos de la electrooculografía**

La electrooculografía (EOG) es una técnica que permite registrar las señales bioeléctricas generadas en el ojo debido a la diferencia de potencial existente entre la córnea y la retina del mismo, las cuales conforman un dipolo que varía de potencial con los movimientos de los músculos oculares (apartado 2.4). La medición de esta diferencia de potencial se realiza mediante la colocación de unos electrodos en las cercanías de los ojos, siendo las señales continuas registradas por estos electrodos lo que se conoce como electrooculograma.

Una correcta colocación de los electrodos, permite generar un electrooculograma del cual es posible deducir los movimientos oculares en los ejes horizontal y vertical, gracias a la relación existente entre la posición angular del ojo y el potencial reflejado en el electrooculograma, como se muestra en la figura 3.5.

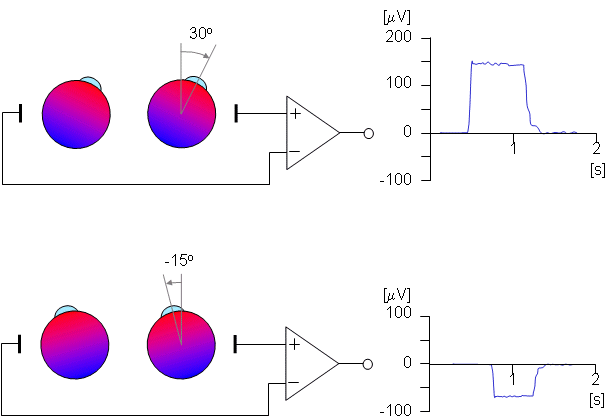


Figura 3.5. Relación entre la posición angular del ojo

y el potencial reflejado en el electrooculograma [6]

Debido a la fiabilidad de esta relación entre la posición angular del ojo y el potencial reflejado en el electrooculograma, unido al hecho de que la mayoría de comportamientos que pudiesen interferir en una correcta detección del movimiento ocular, se pueden parametrizar y eliminar, la electrooculografía es un método fiable para la creación de comandos de control de interfaces usuario-máquina.

La electrooculografía es utilizada como parte fundamental en la polisomnografía (estudio del sueño), junto con la electroencefalografía (EEG) y la electromiografía (EMG), para el estudio y diagnóstico de diversas enfermedades y patologías relacionadas con el sueño. Es una técnica fundamental en oftalmología, junto con el electrorretinograma (ERG). También se utiliza para el diagnóstico de la ataxia, tanto cuando se presenta en forma de síntoma, como cuando se trata de la afección principal, como en el caso de la ataxia cerebelosa o la ataxia de Friedreich.

## **3.3. Aplicaciones**

La electrooculografía tiene aplicaciones en varios campos; desde la medicina como método de estudio y diagnóstico (complementado en muchos casos mediante el electrorretinograma), hasta la ingeniería, en la que es utilizada para el control de diversos dispositivos:

* Oftalmología: la EOG es, junto a la ERG, un elemento muy importante en oftalmología. Mediante esta técnica es posible detectar defectos en la musculatura que rodea el ojo, así como anomalías en los impulsos nerviosos que permiten un correcto funcionamiento de la vista, gracias a lo cual se pueden diagnosticar enfermedades como la retinosis pigmentaria o la ataxia ocular. También es posible detectar una toxicidad medicamentosa mediante el electrooculograma, ya que éste puede presentar anomalías en estadios preclínicos de intoxicación, siendo así decisivo para la interrupción o reducción de un tratamiento farmacológico.
* Neurología: existen enfermedades neurológicas que pueden ser detectadas a través del estudio de los movimientos oculares, como son la ataxia cerebelosa o la ataxia de Friedreich. Junto con el electrorretinograma, es especialmente valioso en el diagnóstico precoz del daño glaucomatoso, así como neuropatías ópticas isquémicas, entre otras afecciones.
* Polisomnografía: la electrooculografía es una herramienta fundamental en el estudio de las fases del sueño. Con ella se pueden detectar, entre otras cosas, anomalías en los movimientos oculares que caracterizan la fase REM (Rapid Eye Movement) del sueño, y diagnosticar diversas enfermedades relacionadas con el mismo, como el síndrome de apnea obstructiva del sueño, que afecta a alrededor del 4% de la población mundial.
* Ingeniería: la principal aplicación de la electrooculografía en la ingeniería es el uso de las señales generadas debido a la dirección de la mirada para el control de dispositivos hardware, como una silla de ruedas, componentes robóticos, dispositivos teledirigidos, ordenadores, etc. En todos los casos, se debe desarrollar un software que traduzca las señales EOG en instrucciones que los dispositivos hardware puedan interpretar y responder en consecuencia. En el caso de los ordenadores se deberá, además, desarrollar una interfaz de usuario simple que minimice la dificultad de manejo de un sistema que normalmente requiere unos periféricos más complejos (como ratón y teclado).

## **3.4. Equipos comerciales**

Actualmente existen varios equipos comerciales desarrollados con la finalidad de detectar la dirección de la mirada, basándose en las técnicas anteriormente descritas (apartado 3.3). Estos equipos comerciales disponibles en el mercado alcanzan gran precisión y son transparentes a los programas en el entorno Windows, pero a pesar de sus virtudes, sus precios los hacen poco accesibles para el usuario medio, y es aquí donde la técnica propuesta se hace atractiva.

A continuación se enumeran algunos de los sistemas disponibles actualmente en el mercado, junto con una breve descripción:

* **BioControl Systems**

BioControl Systems es una empresa con casi treinta años de experiencia en el desarrollo de interfaces hombre-máquina mediante señales bioeléctricas. Fue fundada en 1987 por el Dr. R. Benjamin Knapp y el Dr. Hugh Lusted.

BioCcontrol Systems no desarrolla sistemas de control de interfaces en sí mismos, su producto consiste en la tecnología necesaria para desarrollarlos. Lo más próximo al prototipo que se ha utilizado como plataforma hardware es su BioMuse (figura 3.6), que es un sistema de adquisición de 8 canales que sirve como base para el desarrollo de sistemas de control.

****

Figura 3.6. Equipo de BioMuse [14]

* **CyberLink de BrainActuated Technologies**

Utilizando solo tres electrodos en una banda elástica, colocada alrededor de la cabeza tapando la frente, permite manejar un computador mediante las señales EOG, EMG y EEG.

Las señales, una vez amplificadas y digitalizadas, son separadas en el computador en múltiples bandas de frecuencia denominadas Brainfingers. De esta manera, cada una de ellas puede ser utilizada para controlar distintos aspectos del computador. El mayor inconveniente de este sistema es su elevado precio (alrededor de 2.000€).

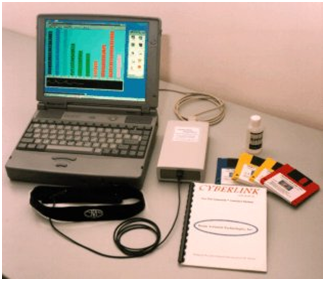
****

Figura 3.7. Equipo de CyberLink [15]

* **Irisbond**

La empresa Irisbond, asentada en Guipúzcoa, ha desarrollado un sistema de control del ordenador mediante la mirada que es fiable, intuitivo, sencillo de usar y preciso. Esta empresa de reciente creación (2013) tiene como objetivo el desarrollo de sistemas basados en tecnologías de comunicación asistida, como el *eye tracking* (apartado 3.1.2), especialmente orientados a personas con gran discapacidad que no pueden comunicarse a través de medios convencionales.

Su dispositivo Irisbond Primma (figura 3.8) permite sustituir el teclado y el ratón como medio para interaccionar con el computador. Está compuesto, básicamente, por una videocámara y unos leds sincronizados que emiten luz infrarroja inocua. Este dispositivo está unido mediante un puerto USB al ordenador donde reside la aplicación que incorpora el procesado de imagen y los algoritmos de control. Esta información es gestionada en el computador a través de una aplicación propia, previamente instalada, de tal manera que los movimientos de los ojos del usuario son transformados en coordenadas cartesianas de posición del ratón en pantalla. Su precio oscila los 2.200€.



Figura 3.8. Equipo Irisbond [16]

* **Bionomadix 2CH EOG Amplifier - BN-EOG2**

El par Dual Wireless EOG BioNomadix consta de un módulo transmisor y un receptor compatible. Está diseñado específicamente para medir dos canales de datos EOG y emular una conexión "por cable" del computador al terminal, con todos los beneficios de un sistema de grabación totalmente inalámbrico. La batería del transmisor puede alcanzar hasta 72 horas de funcionamiento continuo.

El par se utiliza para amplificar los potenciales que aparecen en las superficies de la piel, que rodea los ojos, proporcionales al grado de movimiento de los ojos. La alta relación señal-ruido (SNR) y la alta resolución de muestreo permite a la pareja ser utilizada para la investigación oculomotora de precisión, la atención visual y estudios de respuesta ante estímulos visuales.

* **BlueGain EOG Biosignal Amplifier**

La empresa Cambridge Research Systems ha desarrollado una gama de amplificadores biomédicos conectados de forma inalámbrica para la medición directa de biopotenciales de pacientes. En particular el BlueGain EOG (figura 3.9) se ha configurado para grabar electrooculogramas y comunicar estas señales a través de Bluetooth a un ordenador central, realizando la cantidad mínima de procesamiento. Un sensor de infrarrojos permite que los datos se sincronicen con eventos externos. Tiene un precio aproximado de 1.600€.



Figura 3.9. Equipo BlueGain EOG [17]

# **PLATAFORMA HARDWARE**

La adquisición de las señales bioeléctricas para el control de la interfaz software se realizará utilizado la plataforma hardware desarrollada en la Tesis Fin de Máster de Alberto López Martínez [18]. La plataforma consta de una fase de adquisición de las señales mediante el uso de sensores, una etapa analógica que persigue amplificar y filtrar las señales obtenidas, y una etapa de conversión de analógico a digital para la transmisión de la información a través de un módulo Bluetooth.



Figura 4.1. Plataforma hardware utilizada en el desarrollo del trabajo [18]

## **4.1. Adquisición de señales**

La adquisición de las señales bioeléctricas se consigue mediante la colocación de unos electrodos especiales para medir biopotenciales. Se utilizarán electrodos foam de un solo uso para monitorización a corto plazo, que cuentan con un conector de corchete metálico de Ag/AgCl rodeado de hidrogel de alta calidad. Alrededor cuentan con una capa de pegamento para la sujeción a la piel.



Figura 4.2. Electrodos foam

### **4.1.1. Disposición de los electrodos**

El potencial generado entre la córnea y la retina de los glóbulos oculares puede ser detectado colocando estos electrodos en las zonas adecuadas. Existen varias disposiciones a la hora de colocar los electrodos alrededor de los ojos. Para este proyecto se utilizará la siguiente distribución, tal como se muestra en la figura 4.3.

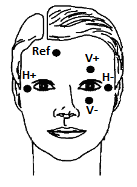


Figura 4.3. Disposición de los electrodos [19]

Los electrodos H+ y H- se colocan en las sienes, a la altura de los ojos, y recogerán los movimientos oculares en el eje horizontal. Los electrodos V+ y V- se colocan por encima y por debajo de uno de los ojos, para recoger los movimientos en el eje vertical y los parpadeos. Por último, se coloca un electrodo de referencia en la frente.

De esta forma, las señales obtenidas mediante la plataforma hardware tendrán el aspecto de la figura 4.4 (señal final una vez procesada).



Figura 4.4. Señal EOG obtenida mediante la plataforma hardware

### **4.1.2. Disposición alternativa de los electrodos**

Existe otra forma alternativa a la utilizada de colocar los electrodos, consistente en cruzar los electrodos de los canales vertical y horizontal, para posteriormente obtener los mismos resultados que con el anterior método al realizar operaciones de suma y resta de los valores obtenidos en ambos canales.



Figura 4.5. Disposición alternativa de los electrodos [19]

La principal ventaja de esta disposición alternativa de los electrodos es que la medición se realiza por duplicado y es posible efectuar con posterioridad operaciones de redundancia, lo que puede ayudar a eliminar disparidades en las mediciones de los dos canales, y teóricamente evitar acoplamiento entre ambos. Además cabe destacar que, mediante esta disposición, los electrodos que normalmente corresponderían al canal horizontal ya no se encuentran directamente situados encima del músculo temporal, hecho que ayuda a evitar las posibles interferencias debidas a la tensión muscular. Sin embargo la plataforma hardware utilizada está diseñada para la adquisición mediante la técnica directa, por lo que no es posible realizar pruebas con esta disposición de los electrodos.

## **4.2. Etapa analógica**

El tratamiento analógico de las señales obtenidas persigue amplificar, filtrar y acondicionar las mismas, para posteriormente enviar el resultado a la etapa digital. La entrada de esta plataforma estará compuesta por cinco señales, dos para cada eje de rotación de los ojos (canal horizontal y canal vertical) y uno de referencia, como se describe en el apartado 4.1.1.



Figura 4.6. Diagrama de bloques de la etapa analógica

El esquema seguido es el mismo para cada canal (vertical y horizontal) y similar al de la mayoría de bioamplificadores.

La etapa de preamplificación es una de las más importantes de la tarjeta de adquisición ya que está en contacto directo con el usuario y de su correcto diseño dependen en gran medida las prestaciones del sistema. Consta de tres elementos: el amplificador de instrumentación, que por sus características es ideal en la captación de biopotenciales, el circuito de referencia, cuya función principal es la de atenuar la interferencia de modo común mediante realimentación activa (electrodo de referencia), y el circuito de compensación de deriva, que contrarresta los desplazamientos de la línea de base en la señal bioeléctrica. Se persigue con ello mejorar la relación señal-ruido (SNR) de las señales adquiridas.

A continuación se realiza un filtrado para eliminar en la medida de lo posible el ruido que acompaña a la señal. Las frecuencias de interés (donde está contenida la información EOG) no superan los 50 Hz, sin embargo, lo ideal sería elegir un rango de frecuencias entre 0,1 y 40 Hz para así eliminar el ruido introducido por el sistema de alimentación, sin perder apenas información útil. Para realizar este corte de frecuencias, se implementan un filtro paso-alto con frecuencia de corte 0,05 Hz (que elimina los efectos del ruido en continua) y un filtro paso-bajo de frecuencia de corte 40 Hz. El resultado es una señal sin apenas componente en continua y sin armónicos de alimentación del sistema.

Finalmente, una vez eliminados los componentes no deseados de la señal, se implementa una fase de acondicionamiento de la señal para su posterior digitalización. Para ello la señal se amplifica y se le suma una tensión continua que evite que adquiera valores negativos para adaptarla al rango deseado de 0 a 3,3 V (para lo que se utiliza un amplificador inversor). A la salida, la señal estará en condiciones de ser digitalizada para adaptar los niveles de señal a los permitidos por las entradas del microcontrolador de la etapa digital.

Mirada a la izquierda Mirada a la derecha

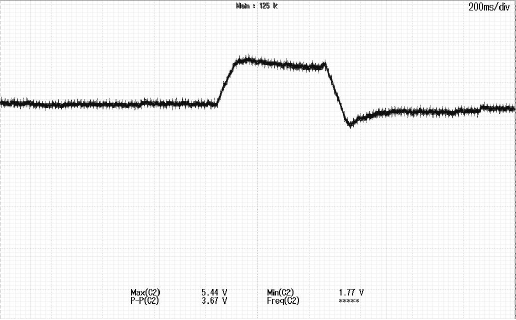
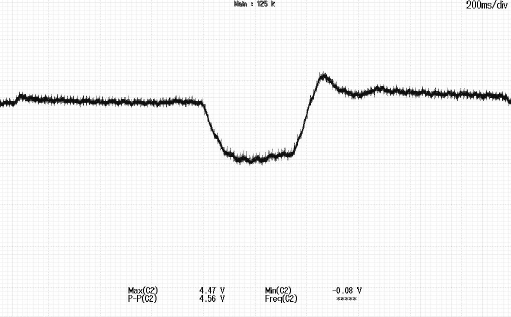
 

Figura 4.7. Señales obtenidas en el canal horizontal a la salida de la etapa analógica

Mirada abajo Mirada arriba

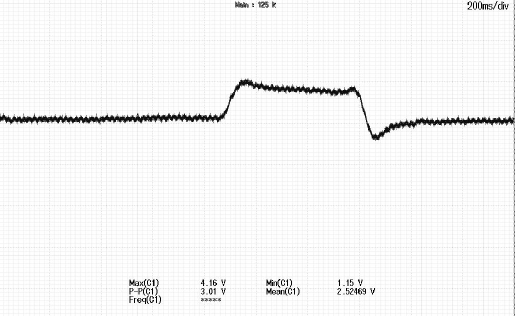
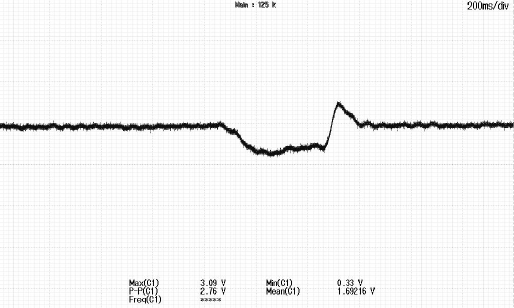
 

Figura 4.8. Señales obtenidas en el canal vertical a la salida de la etapa analógica

## **4.3. Etapa digital**

La finalidad de esta etapa es la de la digitalización de la señal tratada en la etapa analógica y su posterior envío a un computador para su procesamiento. Para ello esta etapa se compone por un microcontrolador (de la serie AT90USB de ATMEL) que se encarga de la conversión A/D y el filtrado digital, conectado a un módulo Bluetooth (BlueGiga) que servirá para la comunicación inalámbrica de datos con el computador.

El convertidor A/D del microcontrolador dispone de una resolución de 10 bits, lo que genera una señal digital cuyo valor oscilará entre 0 y 1023 (correspondientes a los 0 a 3,3 V de la señal analógica acondicionada). La frecuencia de muestreo es uno de los factores clave del rendimiento del sistema propuesto ya que, a mayor frecuencia de muestreo, mayor será la precisión y menor el tiempo de respuesta del sistema. Para elegirla, se debe tener en cuenta el tiempo que transcurre durante la ejecución de los movimientos oculares mencionados en el apartado 2.3. Para poder detectar la mayoría de los movimientos oculares, en especial los sacádicos, se estima necesaria una frecuencia de muestreo superior a 100 muestras por segundo.

Los datos ya digitalizados pasan a ser procesados por el microcontrolador, que se encarga de gestionar las conexiones y el envío de datos al computador a través del módulo Bluetooth, como se detalla en el apartado 4.4. El lenguaje de programación utilizado para programar el microcontrolador es C, utilizando el entorno de desarrollo AVR Studio 5.1 de la marca ATMEL.



Figura 4.9. Diagrama de bloques de la plataforma hardware

## **4.4. Comunicación inalámbrica**

Las muestras obtenidas en la plataforma hardware se enviarán utilizando un módulo Bluetooth de bajo consumo (versión 4.0), el módulo WT12 de BlueGiga. Esta comunicación inalámbrica se basa en el estándar IEEE 802.15.1, que permite un tráfico de hasta 3 Mbps.

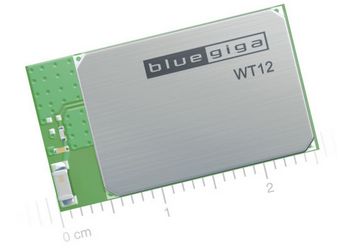


Figura 4.10. Módulo Bluetooth WT12 de BlueGiga [20]

Este módulo utiliza el protocolo de descubrimiento para comprobar periódicamente la existencia de un dispositivo Bluetooth que pueda recibir la señal EOG. Una vez lo localiza comprueba que el computador se llama EOG y, si es así, realiza una petición de conexión junto con una contraseña predeterminada (1111). Una vez enlazados ambos dispositivos, la plataforma hardware comienza a enviar los resultados de la conversión A/D.

Esta conexión Bluetooth será reconocida por ambos dispositivos como una conexión de tipo serie que se gestionará mediante el protocolo RFCOMM de la pila Bluetooth, y un perfil de puerto en serie (puerto COM). Además, se configurará para que presente una velocidad en baudios de 115.200 símbolos por segundo como máximo, aunque esta velocidad será configurable desde el interfaz de la aplicación (apartado 5.3.2)

Los valores para cada canal oscilan entre 0 y 1023 (como se puede observar en la figura 4.4), por lo que se asumen 4 símbolos por ser la condición más restrictiva. Por lo tanto, teniendo en cuenta que cada muestra son 11 símbolos, el máximo teórico en frecuencia será 10,472 kHz.

Es importante destacar que este límite no tiene en cuenta que la tecnología Bluetooth es altamente susceptible a interferencias y que requiere una estrecha cercanía entre los dispositivos enlazados (a pesar de que el estándar indique un alcance de más de 10 metros). Estos factores pueden provocar pérdidas de información en el enlace, y por lo tanto la frecuencia máxima de muestreo deducida podría no alcanzarse.

La tabla 4.1 muestra una captura de la señal digitalizada, correspondiente a los valores del canal vertical al mirar de arriba (1023) a abajo (0) dos veces seguidas. La muestra se ha dividido en columnas, dada la longitud de esta (corresponde a una secuencia de aproximadamente 5 segundos). Se puede observar cómo el potencial aumenta y disminuye de manera directamente proporcional al ángulo de la mirada, saturando el canal al mirar hacia arriba y marcando valores de tensión nulos al mirar hacia abajo (debido al acondicionamiento de la señal descrito en los apartados 4.2 y 4.3).

## 

Tabla 4.1. Resultados digitalizados del canal vertical

# **PLATAFORMA SOFTWARE**

El desarrollo e implementación de la aplicación de escritura controlada mediante señales electrooculográficas pasa por el procesamiento de la señal procedente de la plataforma hardware para su uso como mecanismo de control de la interfaz de usuario que permite dicha escritura. Este mecanismo consistirá en el control del cursor del computador, que se moverá acorde a los movimientos oculares, para desplazarse por el menú de la aplicación y ejecutar las funcionalidades de que dispondrá la aplicación.

Además, la aplicación debe ejecutar las órdenes que recibe a través del Bluetooth a tiempo real, con lo que debe ser modular y concurrente en el procesamiento de la información de control y la ejecución de las órdenes correspondientes, especialmente en la fase de procesamiento digital de la señal.

## **5.1. Lenguaje y bibliotecas utilizadas**

Cada vez existen más opciones a la hora de elegir el lenguaje de programación más apropiado para el desarrollo de una aplicación; C, C++, JavaScript, Python, Ruby... El uso de un lenguaje u otro, por lo general, varía en función de las necesidades y preferencias del desarrollador, por lo que es necesario conocer las diferencias y singularidades de cada uno, así como sus ventajas e inconvenientes.

Partiendo de los objetivos del proyecto (apartado 1.1), se ha priorizado la elección de un entorno de desarrollo para aplicaciones multiplataforma que permita a su vez el desarrollo de una interfaz gráfica de usuario sencilla para su manejo. Por ello se ha optado por el uso de Qt, una biblioteca de desarrollo multiplataforma muy extendida para la programación de aplicaciones con interfaz gráfica de usuario (GUI) [21]. Qt provee de un conjunto de bibliotecas multiplataforma propias que permiten la abstracción del Sistema Operativo. Además, cuenta con un mecanismo de señales y slots que es idóneo para los propósitos de este proyecto (apartado 5.2.6).

Qt utiliza el lenguaje de programación C++ de forma nativa, por lo que también es el lenguaje por defecto a usar en este entorno de desarrollo, aunque es posible utilizar otros lenguajes como Python o Ruby mediante el uso de bindings (una adaptación de las bibliotecas para ser usadas en un lenguaje de programación distinto de aquel en el que ha sido escrita).

C++ es un lenguaje de programación multiparadigma y orientado a objetos (OOP) desarrollado a partir del lenguaje de programación C, con el objetivo de proporcionar a este lenguaje los mecanismos que permitieran la manipulación de objetos. Posee un conjunto completo de instrucciones de control (complementado además por Qt) y permite su agrupación, permite la separación por módulos y su compilación independiente, así como la programación a bajo nivel y el uso de punteros. C++ es un lenguaje muy potente y robusto en lo que se refiere a la creación de sistemas complejos, e idóneo para el propósito del software.

Por todo ello, se utilizará el entorno de desarrollo de Qt Creator 5.3.1 junto con el lenguaje de programación C++ para el desarrollo de esta aplicación.

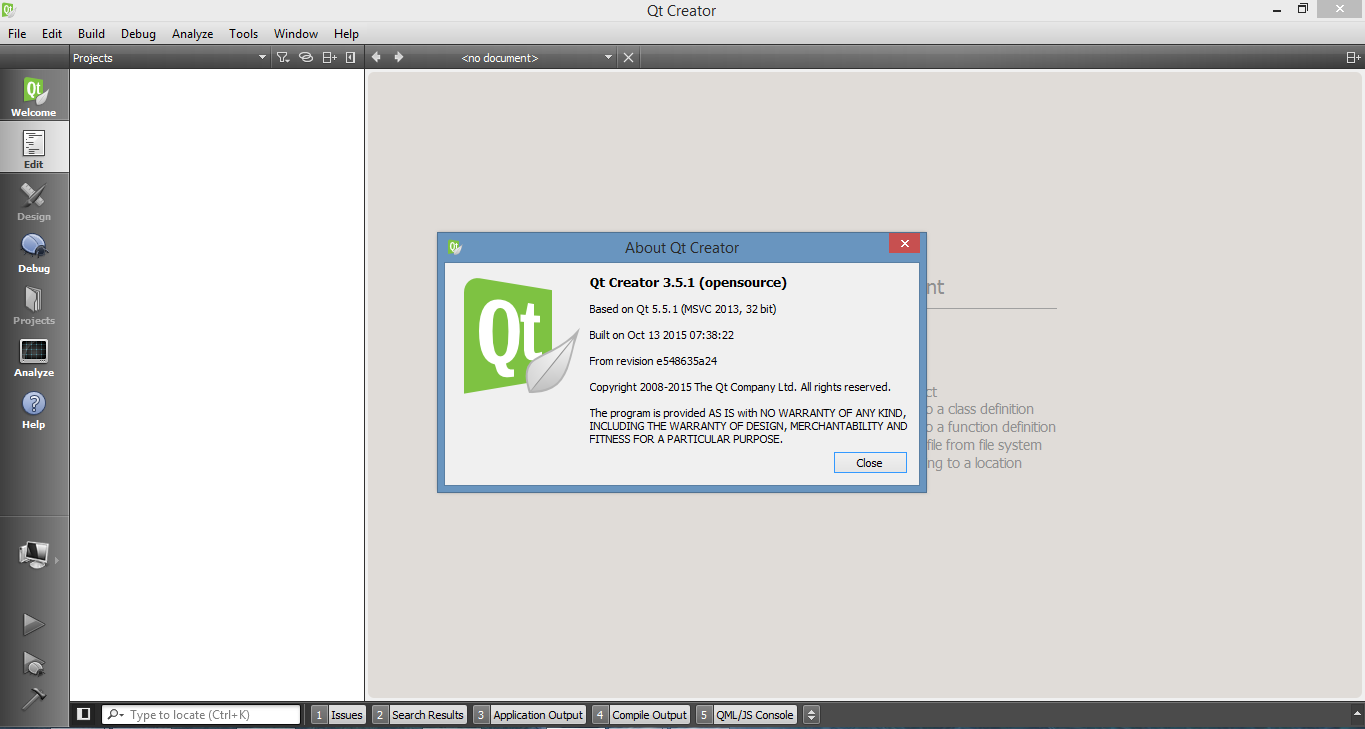


Figura 5.1. Entorno de desarrollo multiplataforma Qt Creator 3.5.1

Para compilar la aplicación, se ha utilizado el compilador MSVC2013 64 bits (Microsoft Visual C++) con el depurador WDK 8.1 (Windows Driver Kit), a través del entorno de desarrollo Qt Creator 5.3.1.

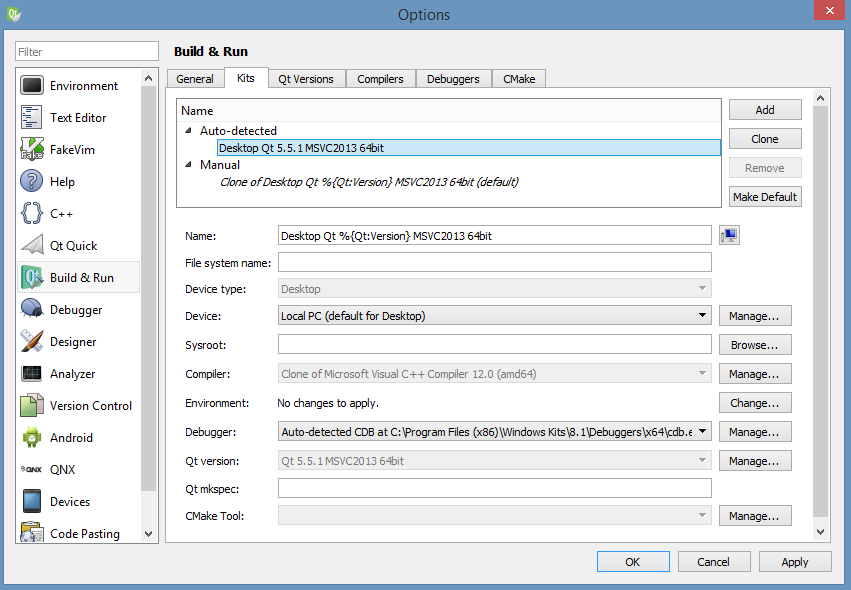


Figura 5.2. Configuración de depuración y compilación

Además de las bibliotecas propias de Qt, también ha sido necesario el uso de la biblioteca MS Speech API (SAPI 5.4) de Microsoft [22] para añadir la funcionalidad de lectura (sintetizador de voz) que se describe en el apartado 5.3.1. Esta biblioteca permite la lectura de un texto mediante el uso de la tecnología Text To Speech (TTS), a través de las bibliotecas de voz que provee Microsoft. En este caso se han utilizado las del español.

## **5.2. Procesamiento digital de la señal**

La fase de procesamiento digital de la señal consta de cuatro etapas: lectura de datos, mapeado de datos, tratamiento de datos y toma de decisiones.

### **5.2.1. Lectura de datos de la plataforma hardware**

El módulo de lectura de datos (btreaderthread, apartado A.2 del Anexo I) se encarga de iniciar y gestionar la comunicación Bluetooth con la plataforma hardware. Para ello, crea un objeto QSerialPort de Qt, con los parámetros de conexión Bluetooth (apartado 5.3.3) y envía una señal a la clase principal para indicar que está listo para la recepción de datos.

Una vez establecida la conexión, el módulo lee los datos (que llegan en formato de cadenas de bytes) de manera recursiva mientras el puerto permanezca abierto, que irá enviando en el mismo orden y formato de llegada al módulo de mapeado mediante el uso de señales. Al terminar el envío de datos, el objeto cierra el puerto y se liberan los recursos.

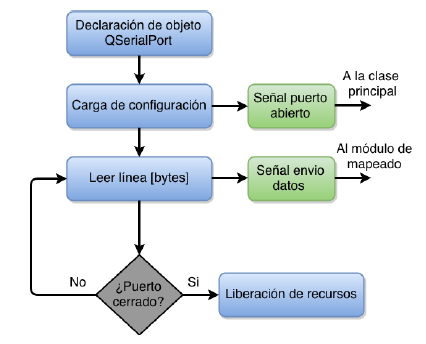


Figura 5.3. Diagrama de flujo del módulo de lectura

### **5.2.2. Mapeado de los datos**

La función del módulo de mapeado de datos (dataparserthread, apartado A.3 del Anexo I) es traducir las cadenas de bytes de datos que recibe del módulo de lectura a un formato más legible para su procesamiento, separando las muestras por canales (horizontal y vertical) y transformando las muestras en una cadena de texto con el valor numérico (escalado de 0000 a 1023) correspondiente para cada canal, separado por un espacio, y con un retorno de carro y salto de línea para delimitar cada muestra.

Donde “X” e “Y” serán los canales

“XXXX[espacio]YYYY\r\n””

Horizontal y Vertical respectivamente.

Finalmente, se envían los datos al módulo de tratamiento para procesar los valores obtenidos y determinar la dirección de la mirada.

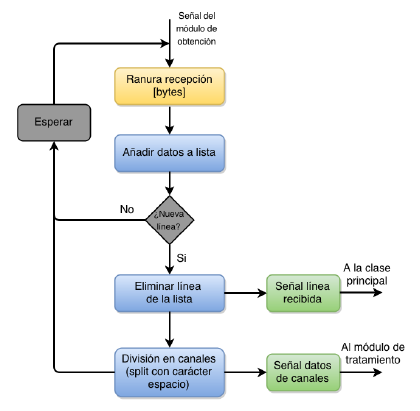


Figura 5.4. Diagrama de flujo del módulo de mapeado

### **5.2.3. Tratamiento de los datos**

El módulo de tratamiento de datos (datatreatmentthread, apartado A.4 del Anexo I), es el núcleo de la fase de procesamiento digital de la señal. En este módulo, se procesan las muestras recibidas desde el módulo de mapeado (ya formateadas en cadenas con los canales separados), para determinar qué movimiento se ha ejecutado con los ojos, de entre la gama de cinco movimientos disponibles (para los que se ha diseñado la aplicación); mirada arriba o abajo, izquierda o derecha, o parpadeo.

El algoritmo emplea umbrales, que pueden ser definidos en el archivo de configuración config.ini (apartado A.10 del Anexo I), para determinar qué movimiento se ha efectuado y enviar esta información al módulo de decisión, mediante su correspondiente señal, para que se ejecute la instrucción apropiada, que dependerá de la funcionalidad de la aplicación que se esté ejecutando. Cada canal se procesará de manera independiente, tal y como muestra la figura 5.5, pudiendo efectuarse más de un movimiento (canal horizontal y canal vertical) al mismo tiempo, resultando en ese caso un movimiento oblicuo, suma de las componente horizontal y vertical.

Los umbrales, tras la realización de varias pruebas (apartado 6.1) de manejabilidad y rendimiento, han sido fijados en 1000 por arriba y 100 por abajo, dando como resultado una correcta interpretación de los movimientos oculares por parte del algoritmo de procesamiento.

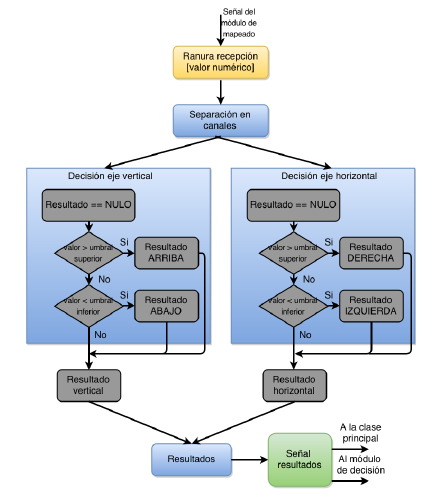


Figura 5.5. Diagrama de flujo del módulo de tratamiento

En este módulo es donde se realizarían los posibles trabajos para la mejora del rendimiento de la aplicación, si éste fuera susceptible de mejoras.

### **5.2.4. Toma de decisiones**

El módulo de toma de decisiones (decisionthread, apartado A.5 del Anexo I) recibe entonces los datos ya definidos como direcciones de la mirada, aunque aún es necesario añadir una lógica que determine si el movimiento se produjo en un solo eje o en los dos, o si hubo parpadeo, antes de tomar la decisión de movimiento de cursor. Una vez procesada la información por este módulo, se tiene el resultado final de la instrucción emitida mediante el movimiento ocular, que será enviada mediante una señal para el movimiento de cursor, que ejecuta la actualización de coordenadas de cursor (como se muestra en la figura 5.6), y otra señal distinta para el parpadeo, que ejecuta la rutina “clic” del cursor. Ambas señales son recibidas por funciones definidas en la clase principal (mainwindow, apartado A.1 del Anexo I), la primera por la función encargada del movimiento del cursor, mediante la actualización de las coordenadas del mismo, y la segunda por la función encargada de invocar la rutina de “clic” del cursor. Ambas funciones utilizan la clase QCursor, proporcionada por las librerías Qt, que se comunica con el Sistema Operativo para que sea éste quien ejecute las órdenes correspondientes.

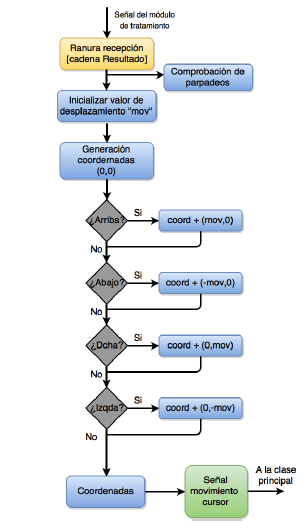


Figura 5.6. Diagrama de flujo del módulo de decisión

### **5.2.5. Modularidad y concurrencia**

El paradigma de la modularidad, y más concretamente en la programación orientada a objetos (OOP), es la propiedad que permite subdividir una aplicación en partes más pequeñas, llamadas módulos, cada una de las cuales con la propiedad de ser tan independiente como sea posible de la aplicación en sí y de las restantes partes, pudiendo compilarse cada una por separado, aun teniendo comunicaciones con los otros módulos.

La aplicación ha sido diseñada teniendo en cuenta esta característica, siendo cada uno de los módulos descritos en este apartado 5.2, así como los mismos descritos en el apartado 5.3, independientes entre sí, pero estando comunicados al mismo tiempo. De esta forma, cada módulo se ejecuta sin la necesidad de conocer el origen de los datos que procesa, ni lo que hará el siguiente módulo con los datos que le transmite.

Además, el diseño modulable permite el desarrollo independiente de cada uno de los módulos sin tener que modificar el resto, proporcionando una base sólida para cualquier mejora de la aplicación (por ejemplo en el procesamiento digital de la señal) y el desarrollo de funcionalidades adicionales en el futuro (como las descritas en el apartado 8.3).

Unido al factor de la modularidad, se encuentra el diseño de la aplicación basado en una arquitectura en cadena de montaje, que permite la concurrencia de operaciones en la aplicación, es decir, la aplicación es capaz de procesar en varias fases secuenciales un conjunto de muestras recibidas previamente (siendo la entrada de cada una la salida de la anterior), mientras procesa las nuevas que está recibiendo. Cada módulo se ejecuta independiente y simultáneamente separados en hilos (ejecución multi-hilo), pudiendo interactuar entre ellos, como se describe en el apartado 5.2.6.

Esta característica supone un aumento en el rendimiento de la aplicación que permite el funcionamiento a tiempo real de la misma, frente a la opción de procesar cada muestra recibida antes de procesar la siguiente, que supondría introducir limitaciones tanto en el número de muestras generadas por la plataforma hardware, como en el tratamiento digital de la señal por parte de la plataforma software.

### **5.2.6. Comunicación entre módulos**

Para la comunicación entre módulos u objetos se han usado las funcionalidades de “*signals y slots*” (señales y ranuras) de Qt. El mecanismo de signal-slot es una de las características más importantes de la librería Qt, probablemente la que le diferencia de otros framework para el desarrollo de interfaces de usuario. Las señales parten de un objeto emisor, pudiendo contener datos en las mismas, y llegan a un objeto receptor que decide si ejecuta un slot o emite una nueva señal para seguir con la comunicación. Los slots, son pequeñas funciones que se ejecutan cuando se emite la señal correspondiente que la llama. Para que las señales y slots se comuniquen, es necesario definir las conexiones siguiendo la siguiente sintaxis:

connect(emisor, SIGNAL(señal\_emitida()), receptor, SLOT(slot\_accion()));

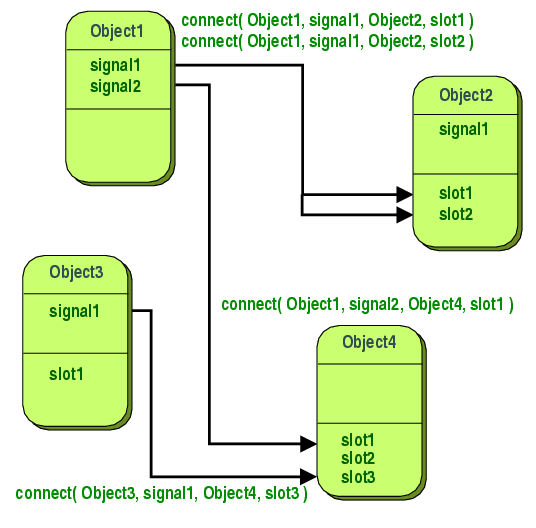


Figura 5.7. Mecanismo de señales y slots [22]

Las conexiones entre objetos crean colas de señales (en un buffer) que se ejecutan según la disciplina de colas FIFO (First In, First Out) y se van vaciando a medida que los slots correspondientes se van ejecutando, de manera secuencial. Mientras los objetos no sean complejos, los tiempos de retardo introducidos por la comunicación entre módulos son despreciables, de ahí que se haya optado por la división en cuatro etapas del procesamiento digital de la señal.

Este sistema sigue el esquema de diseño de publicación/suscripción, en el que las clases que emiten las señales (signals) estarían publicando datos y las clases que los reciben (slots) se suscribirían para recibirlos.

## **5.3. Interfaz de usuario**

El interfaz de usuario ha sido diseñado de la forma más sencilla y eficiente posible, teniendo en cuenta las necesidades del usuario final (apartado 1.1).

Se ha creado un conjunto de nueve botones, que servirán para ejecutar las funcionalidades de las que dispone la aplicación (de escritura, lectura, configuración y funcionalidades adicionales), junto con un cuadro de edición de texto donde irá apareciendo el texto introducido por el usuario.

El usuario se irá desplazando por el menú principal de la aplicación, controlando el cursor mediante los movimientos oculares, hasta llegar al botón correspondiente a la función que se desea ejecutar, efectuando un parpadeo para seleccionar dicha función.



Figura 5.8. Interfaz gráfico de usuario de la aplicación. Menú principal

El interfaz gráfico de usuario (GUI) ha sido diseñado utilizando el “modo diseño” de Qt Creator, que permite añadir los widgets de forma manual, así como su presentación y los slots que se ejecutan al emitirse la señal que indica que han sido pulsados (on\_button\_clicked).

A continuación se detallan las funcionalidades disponibles en la aplicación.

### **5.3.1 Sistema de escritura**

La funcionalidad principal de la aplicación es, como se recoge entre los objetivos del proyecto (apartado 1.1), el sistema de escritura controlado mediante señales electrooculográficas. La aplicación cuenta con varios botones en su ventana principal destinados a tal fin:

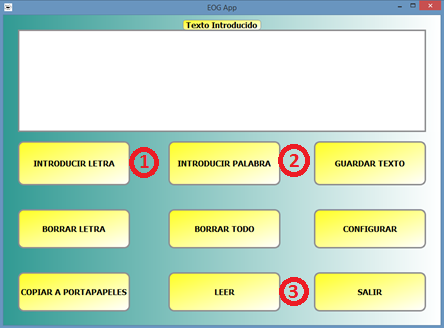


Figura 5.9. Botones de escritura y lectura

El botón “INTRODUCIR LETRA” (1) es el núcleo principal del método de escritura, y sirve para seleccionar los caracteres alfanuméricos disponibles mediante el algoritmo de escritura, el botón “INTRODUCIR PALABRA” (2) sirve para seleccionar una o más palabras de entre un conjunto de palabras preestablecido y por último el botón “LEER” (3) transforma el texto introducido en voz. A continuación se pasa a detallar cada una de estas funcionalidades.

* **1. INTRODUCIR LETRA**

Actualmente existe una gran gama de teclados virtuales de todos los tamaños y estilos imaginables, con lo que no sería necesario el desarrollo de un teclado virtual en cualquier caso, sino su integración con la aplicación. No obstante, el método de escritura diseñado y desarrollado para la aplicación (módulo writting, apartado A.6 del Anexo I) no es un teclado virtual al uso, sino un método de escritura ideado teniendo en cuenta los objetivos de este proyecto. Se trata de un método de escritura más eficiente que el teclado virtual clásico, ya que no hace falta desplazarse por el mismo para seleccionar una letra sino que, en modo escritura, las señales se utilizan para navegar por un sistema de crucetas basado en un algoritmo escalable en el que, con tres movimientos de los ojos (en ambas direcciones de los ejes vertical y horizontal) se asegura alcanzar cualquier carácter alfanumérico de los 64 disponibles en este prototipo.

Nº de caracteres = 4n

Donde 4 es el número de direcciones disponibles en cada cruceta (arriba, abajo, izquierda y derecha) y “n” el número de iteraciones del algoritmo que se ejecuten. Por ejemplo, con n=4 dispondríamos del número necesario para utilizar ASCII (256 caracteres).

En el caso de este prototipo, se han implementado tres iteraciones del algoritmo, con lo que resultan un total de 64 caracteres disponibles. En la primera iteración del algoritmo (etapa 1), se mostrarán estos 64 caracteres agrupados en 4 grupos de 16 (en las cuatro direcciones de la cruceta), en la segunda (etapa 2) se mostrarán los 16 caracteres seleccionados en la primera iteración, agrupados en 4 grupos de 4, y por último en la tercera iteración (etapa 3), se mostrará cada una de las 4 letras seleccionadas del grupo de la segunda iteración, en cada una de las 4 direcciones disponibles en la cruceta, para seleccionar con el tercer movimiento la letra deseada.

Por ejemplo, para introducir la letra “H”, se deberán efectuar tres movimientos en tres etapas (las tres iteraciones del algoritmo):

* Etapa 1: dado que la “H” se encuentra en el grupo de letras de la izquierda, la primera dirección de la mirada se debe realizar a la izquierda.

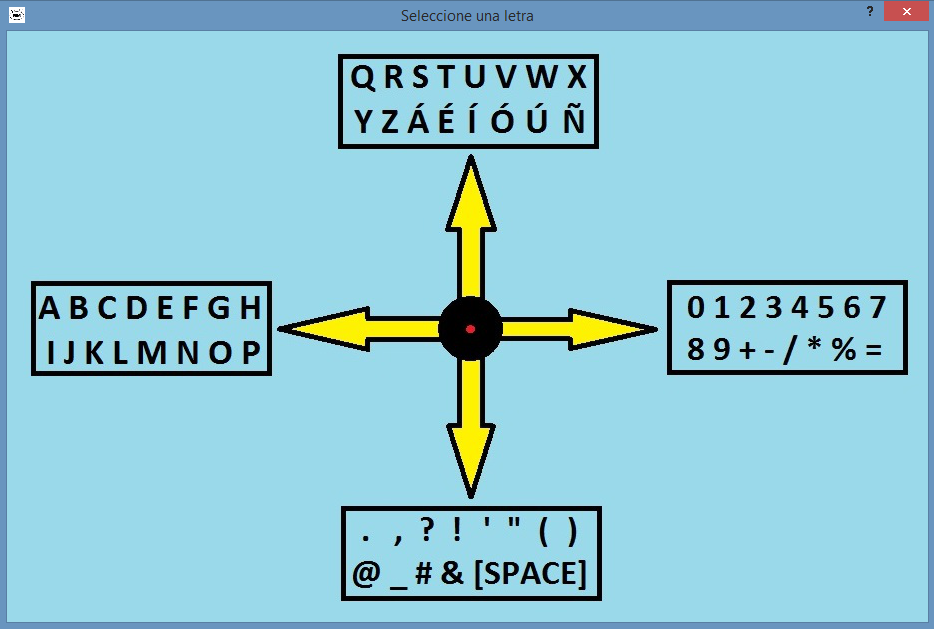


Figura 5.10. Sistema de crucetas, etapa 1

* Etapa 2: el grupo de letras seleccionado en la etapa 1 se despliega en grupos de cuatro, en las cuatro direcciones de la cruceta. En esta ocasión, se deberá dirigir la mirada hacia arriba para seleccionar el grupo de letras que contiene la letra “H”.

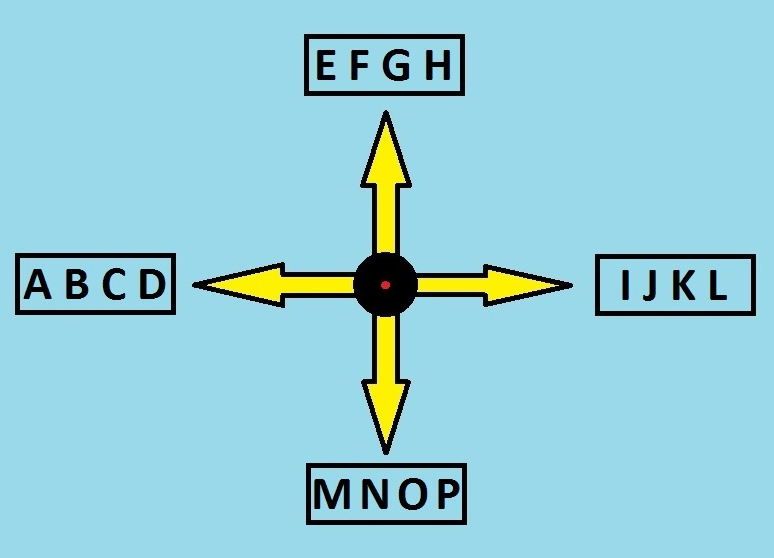


Figura 5.11. Sistema de crucetas, etapa 2

* Etapa 3: por último, tras desplegarse las cuatro letras seleccionadas en la etapa 2 en cada una de las cuatro direcciones de la cruceta, con una mirada hacia abajo la letra “H” será seleccionada y aparecerá en el cuadro de texto de la ventana principal de la aplicación, quedando ésta inmediatamente lista para seleccionar otra letra o ejecutar cualquier otra funcionalidad de la misma.

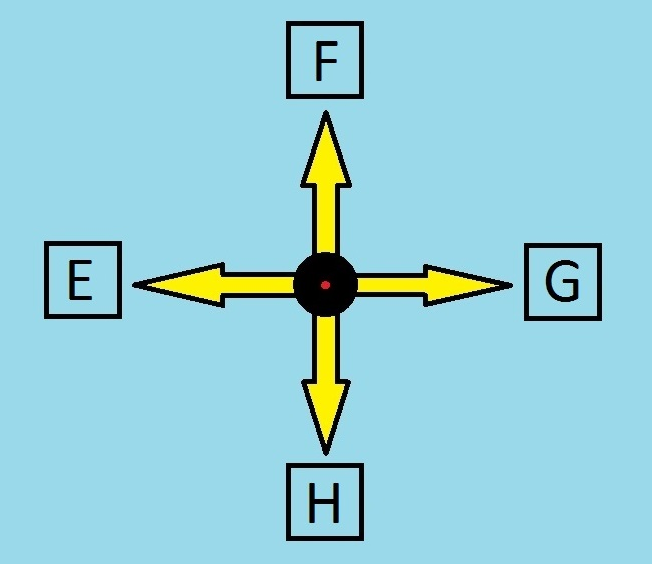


Figura 5.12. Sistema de crucetas, etapa 3

El sistema se implementa mediante el uso de varias funciones auxiliares que sirven al algoritmo principal. En primer lugar, se desconectan las señales de decisión de las funciones de control del cursor, y se crean nuevas conexiones que servirán para desplazarse por el menú de crucetas mediante los movimientos oculares (las conexiones serán restauradas al finalizar la introducción del carácter). La función auxiliar “ShowImage()” muestra la cruceta correspondiente a la etapa de iteración, teniendo en cuenta las selecciones efectuadas previamente si las hubiera. En el mismo momento en que se muestra la imagen (cargada del archivo de recursos de la aplicación application.qrc, apartado A.9 del Anexo I), un buffer auxiliar comienza a almacenar las muestras recibidas de los hilos de procesamiento digital de la señal (apartado 5.2), hasta que se llena el vector de muestras de cualquiera de las cuatro direcciones de la cruceta disponibles (arriba, abajo, izquierda y derecha). Entonces, se envía a la función principal de selección de letra la dirección de la mirada, para desplazarse por el menú de etapas, donde en función de esta dirección se irán mostrando las crucetas correspondientes con los grupos de letras que se van seleccionando, como se ha descrito previamente en este apartado.

El uso del buffer viene dado por el hecho de que la velocidad de muestreo genera un gran número de muestras por movimiento y, por cuestiones de funcionalidad, fue necesario ajustar este número de muestras, ya que durante la ejecución de las pruebas de rendimiento se observó que con una sola mirada se pasaba por todas las etapas de selección, llegando a la letra final con este único movimiento ocular. Además, dada la posibilidad de tener varias direcciones de la mirada a la vez en cada muestra, mediante el uso de este buffer se evitan duplicidades en las instrucciones de selección de la dirección de la cruceta.

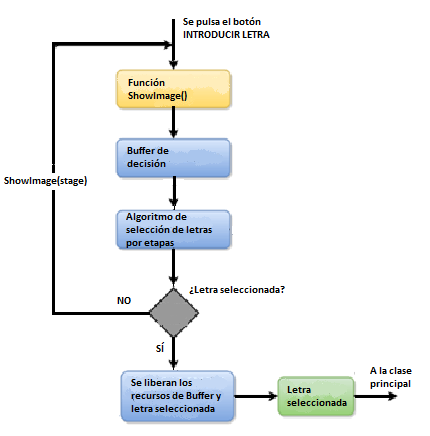


Figura 5.13. Diagrama de flujo del módulo de escritura

* **2. INTRODUCIR PALABRA**

Adicionalmente al sistema de escritura mediante introducción de letras, se ha añadido una funcionalidad para introducir palabras ya construidas, de una lista pre introducida de palabras. El módulo de escritura por palabras (words, apartado A.7 del Anexo I) abre una nueva ventana con todas las palabras disponibles, que irán apareciendo en el cuadro auxiliar de edición de texto. Cada vez que una palabra es seleccionada se emite una señal que contiene la misma y es recogida por la clase principal para su impresión en el cuadro de edición de texto. Una vez se ha terminado de introducir palabras, se hará clic sobre el botón <TERMINADO> para volver a la ventana principal de la aplicación, donde inmediatamente aparecerán las palabras introducidas en la ventana de edición de texto principal (a continuación del texto ya introducido, si lo hubiera) y se podrá seguir navegando por la aplicación para utilizar cualquier otra funcionalidad.

Para que su uso resulte más práctico y visible, se han agrupado por colores las palabras relacionadas (figura 5.14), de manera que los pronombres están rodeados por un ribete verde, los verbos por un ribete azul, comidas con ribete amarillo, lugares con ribete magenta, etc. Los botones especiales (<ESPACIO> y <TERMINADO>), con ribete rojo, se han colocado en el centro del menú de botones por cuestiones prácticas.

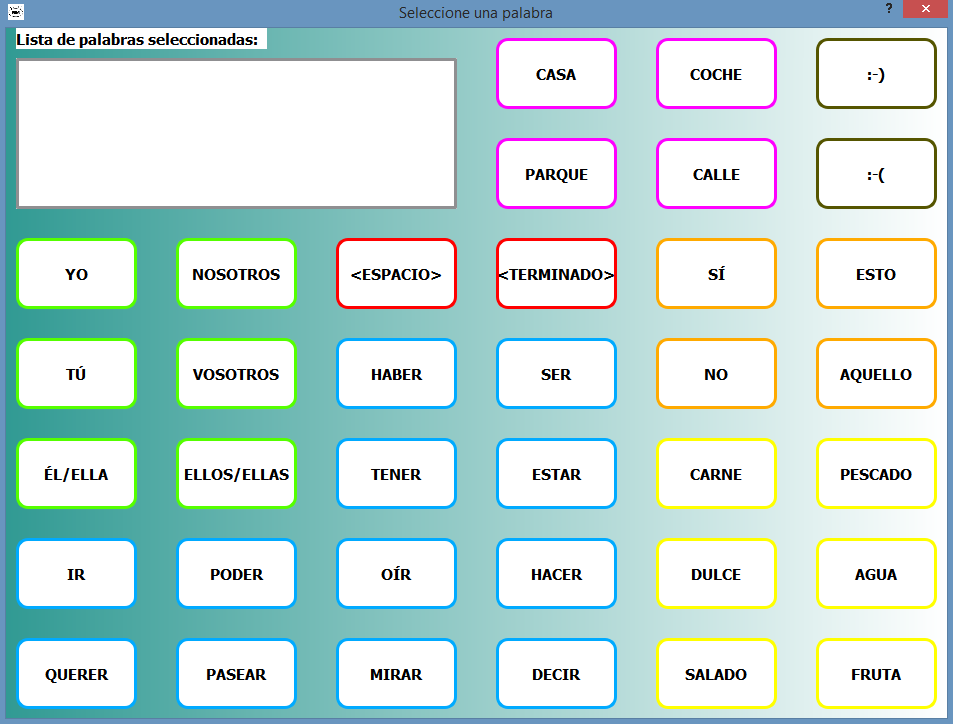


Figura 5.14. Menú introducir palabra

* **3. LEER**

Se ha añadido un botón para complementar la escritura, que permite leer en voz alta el texto introducido (mainwindow, apartado A.1 del Anexo I) mediante un sintetizador de voz, con el fin de permitir la comunicación no solo por escritura, sino también por voz.

Para ello se han utilizado las bibliotecas MS Speech API (SAPI 5.4) [22] de Microsoft, que permite la lectura de un texto mediante el uso de la tecnología Text To Speech (TTS), utilizando las bibliotecas de voz que provee Microsoft (en este caso se han utilizado las correspondientes al español).

### **5.3.2 Funcionalidades adicionales de la aplicación**

Además de las funcionalidades de escritura descritas en el apartado 5.3.1, la aplicación cuenta con varias funcionalidades más que se pasa a detallar.

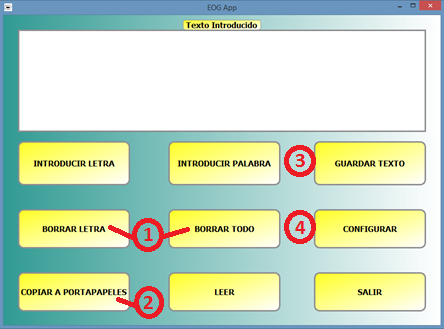


Figura 5.15. Funcionalidades adicionales

* **1. BORRAR LETRA / BORRAR TODO**

La aplicación cuenta con un botón (mainwindow, apartado A.1 del Anexo I) para borrar la última letra introducida por el usuario en caso de error. También es útil para borrar la última palabra introducida en el caso de introducir una palabra errónea utilizando el modo de escritura por palabras, en este caso borrando letra por letra.

Otra opción disponible es la de borrar todo el texto introducido (mainwindow, apartado A.1 del Anexo I), mediante el botón “BORRAR TODO”, en caso de necesitar borrar todo el texto introducido hasta el momento y empezar otro escrito desde cero.

* **2. COPIAR A PORTAPAPELES**

Se ha añadido un botón (mainwindow, apartado A.1 del Anexo I) con la funcionalidad de copiar el texto al portapapeles del Sistema Operativo con el fin de permitir la interoperabilidad entre aplicaciones. Una vez en el portapapeles del SO, el texto puede ser utilizado para introducirlo en otra aplicación de texto o un cuadro de búsqueda en Internet, por ejemplo.

Una vez copiado el texto al portapapeles, aparecerá un cuadro de diálogo donde se da la posibilidad de borrar el texto introducido o seguir editándolo.

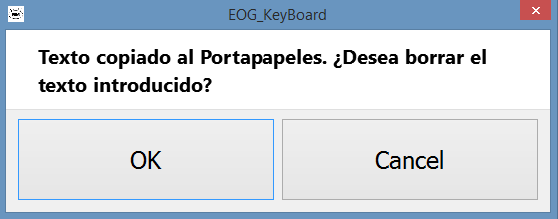


Figura 5.16. Mensaje copiar a portapapeles

* **3. GUARDAR TEXTO**

La aplicación también cuenta con la opción de guardar el texto introducido por el usuario en cualquier momento (mainwindow, apartado A.1 del Anexo I). Los textos se almacenan en una carpeta llamada “texts” que se encuentra en la carpeta de la aplicación (figura 5.22), al mismo nivel que el ejecutable. Al pulsar el botón, se genera automáticamente un fichero de texto plano “.txt”, con el nombre correspondiente a la fecha y hora del sistema y el texto introducido mediante los mecanismos de escritura.



Figura 5.17. Ficheros de texto generados por la aplicación

Una vez guardado el texto en el archivo de texto, aparecerá un cuadro de diálogo donde se da la posibilidad de borrar el texto introducido o seguir editándolo.

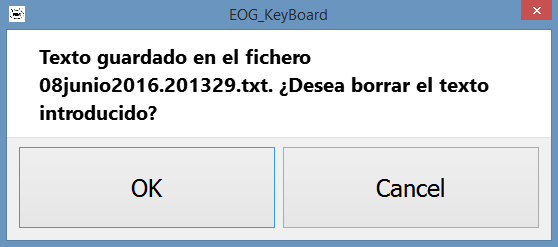


Figura 5.18. Mensaje guardar texto

### **5.3.3 Configuración**

Debido a las diversas configuraciones disponibles para establecer la conexión Bluetooth con los diferentes computadores en los que es posible ejecutar la aplicación, se ha habilitado un diálogo de configuración (btsettingsdialog, apartado A.8 del Anexo I) con el fin de permitir establecer los parámetros de comunicación adecuados para cada usuario.

A través de este diálogo es posible configurar el puerto COM por el que se establece la comunicación, así como la velocidad en Baudios, bits de datos y de parada, paridad y control de flujo. Será necesario reiniciar la aplicación para que los cambios surtan efecto, ya que estos parámetros se cargan en el arranque para establecer la conexión Bluetooth.



Figura 5.19. Diálogo de configuración

Los parámetros se configuran por defecto mediante la carga de los ficheros de configuración config.ini y mapper.ini (apartado A.10 del Anexo I)

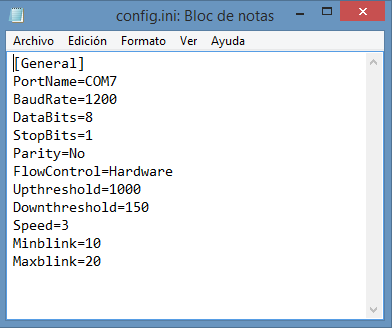
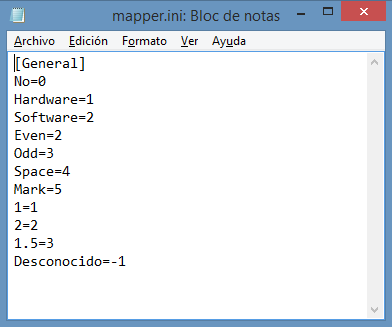
 

Figura 5.20. Fichero de configuración config.ini Figura 5.21. Fichero de configuración mapper.ini

## **5.4 Puesta en marcha: instalación y funcionamiento de la aplicación**

Para poner en marcha el sistema es necesario realizar los siguientes pasos:

* **Instalación de la aplicación**

1. Se debe realizar una copia del contenido del DVD de la aplicación en la carpeta donde se desee instalar el software (por ejemplo, una carpeta en el escritorio). De esta forma, la arborescencia debe ser la siguiente:

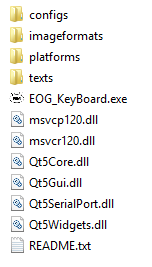


Figura 5.22. Arborescencia de la aplicación

1. Opcionalmente, se puede crear un acceso directo desde el escritorio al ejecutable de la aplicación (botón derecho sobre EOG\_KeyBoard.exe, enviar a escritorio).
2. Es posible que en algunos equipos sea necesario indicar al antivirus que se confía en el editor de la aplicación y permitir su ejecución.

1. Será suficiente con un doble clic sobre el icono del ejecutable de la aplicación (EOG\_KeyBoard.exe) para su inicialización.

* **Funcionamiento**

1. Colocar los sensores foam sobre la superficie de la cara y conectarlos a los cables de la plataforma hardware, como se indica en el apartado 4.1.1.
2. Comprobar que el nombre del computador donde está instalada la aplicación tiene como nombre “EOG” (requisito de la plataforma hardware). Si tiene cualquier otro nombre, debe cambiarse por el de “EOG” para que el sistema pueda arrancar.
3. Conectar la plataforma hardware a un puerto USB disponible para su alimentación y puesta en funcionamiento.
4. Cuando recibe la alimentación, la tarjeta comienza a buscar los dispositivos Bluetooth disponibles dentro de su rango de señal para la conexión de manera automática, hasta detectar uno con el nombre “EOG”.
5. Es entonces cuando se deben sincronizar los dispositivos de forma manual desde el computador. El módulo Bluetooth pedirá la contraseña (1111) para establecer la conexión de manera segura.
6. Una vez sincronizados los dispositivos, el Bluetooth espera por la apertura del puerto de comunicación de los datos, que se produce al ejecutar la aplicación. Estos pasos en la conexión Bluetooth están asignados a un color diferente en un sistema de LEDs añadido a la plataforma hardware para la visualización del estado de las conexiones. Los colores y sus fases de conexión correspondientes son:
   * Verde = Fase 1. La tarjeta recibe corriente.
   * Naranja = Fase 2. Búsqueda de dispositivos con el nombre “EOG”.
   * Azul = Fase 3. Dispositivo encontrado, se establece la comunicación.
   * Verde parpadeante = Fase 4. Transmisión en curso.
7. Una vez establecida la comunicación, es conveniente esperar a que la plataforma hardware alcance su punto óptimo de rendimiento y los parámetros biométricos se adapten a condiciones (debido a limitaciones de la EOG).
   * Nota: si una vez llegado a este punto no se ha establecido la conexión, se deberán revisar los parámetros de configuración Bluetooth (apartado 5.3.2) y adaptarlos a las necesidades del sistema de comunicación.
8. Por último, el usuario interactuará con la aplicación mediante las siguientes instrucciones:
   * Miradas en cualquier dirección: el ratón se moverá en la misma dirección que la mirada.
   * Parpadeos: se traducirán por “clics” del ratón.
   * Sistema de crucetas: descrito en el apartado 5.3.1.

El usuario podrá desplazarse a través del menú de la aplicación y ejecutar cualquiera de las funcionalidades descritas en el apartado 5.3.

Toda esta información está contenida en el fichero README.txt que acompaña al ejecutable de la aplicación.

# **PRUEBAS**

Con el objetivo de medir el rendimiento del software desarrollado, así como su usabilidad y manejabilidad, para que resulte apto para su uso, se han diseñado y efectuado varias pruebas. Es necesario, determinar el grado de fiabilidad con el que la aplicación es capaz de funcionar (generar los comandos correspondientes a las órdenes dadas de manera fiable) y parametrizar los tiempos de respuesta y retardo de la aplicación para determinar los parámetros de configuración de la conexión más adecuados.

Con tal fin, en primer lugar se realizaron pruebas relativas a la usabilidad del sistema, probando varios valores de configuración y midiendo la respuesta del sistema. Una vez ajustados los parámetros de configuración, se efectuaron varias pruebas de escritura midiendo los tiempos empleados. También se ha probado la aplicación en varias plataformas para verificar su interoperabilidad. Las conclusiones de todas estas pruebas se recogen en el apartado 7.3.

## **6.1. Pruebas con parámetros de configuración**

Por las propias necesidades de usabilidad de la aplicación, al efectuar las primeras pruebas una vez generado el ejecutable de la aplicación, fue necesario ajustar varios parámetros para que el rendimiento de la aplicación fuera óptimo. Para ello, se programaron una serie de parámetros de configuración que se cargan mediante ficheros de configuración, con lo que no es necesario volver a compilar la aplicación cada vez que se desee ajustar el funcionamiento de la aplicación.

Se han probado varias configuraciones de los siguientes parámetros:

* **Umbrales de miradas:** definidos en 50 por abajo y 1000 por arriba (para ambos canales), según los valores digitales a la salida de la plataforma hardware. Si se superan estos valores umbral (definidos en tensión), el algoritmo de procesamiento de la mirada determina la dirección en la que se ha producido la misma.
* **Umbrales de parpadeo:** dado que el parpadeo se detecta en el canal vertical y tiene como resultado la saturación de este por arriba, al igual que una mirada hacia arriba, se definieron umbrales (en el tiempo) para distinguir ambas instrucciones. Si la señal que genera la saturación del canal vertical se produce en un margen de entre 10 y 20 milisegundos, se traducirá por un parpadeo y no por una mirada hacia arriba.
* **Salto de cursor:** mediante este parámetro se puede configurar la velocidad a la que se mueve el cursor. El algoritmo de actualización de coordenadas recibe este parámetro para actualizar la posición del cursor a tiempo real, con lo que la actualización se producirá moviendo en el eje correspondiente, el número de posiciones indicado en este parámetro. El valor se ha fijado en 3 tras probar en un rango de 1 a 5, dado que ha resultado más manejable con este número de saltos por actualización de posición.
* **Parámetros de configuración de la conexión Bluetooth:** se ha jugado con la velocidad de transmisión en Baudios para ajustar el número de muestras por segundo que recibe la aplicación. Se ha fijado una tasa de 1200 Baudios como la más óptima para el manejo de la aplicación.

## **6.2. Pruebas de velocidad de escritura**

Con el objeto de medir los tiempos de respuesta y retardos de la aplicación en el modo escritura, se han diseñado y efectuado unas sencillas pruebas de escritura consistentes en escribir una palabra o frase y pulsar el botón leer para que se reproduzca por voz el texto introducido. Los resultados se muestran en las siguientes tablas:

* **Prueba de escritura 1:** escribir “HOLA” y pulsar leer (tiempos en segundos)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nº Fallos  (borrar letra) | Nº Fallos  (botones) | Tiempo de  Escritura | Tiempo Total |
| Primer intento | 1 | 0 | 46 | 55 |
| Segundo intento | 1 | 0 | 41 | 48 |
| Tercer intento | 0 | 0 | 33 | 41 |
| Cuarto intento | 1 | 0 | 40 | 48 |
| Quinto intento | 0 | 0 | 34 | 41 |

Tabla 6.1. Prueba de escritura 1

El tiempo total menos el tiempo de escritura, es el tiempo empleado en desplazarse por el menú de botones y hacer los clics.

* **Prueba de escritura 2:** escribir “HOLA MUNDO” y pulsar leer (tiempos en segundos)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nº Fallos  (borrar letra) | Nº Fallos  (botones) | Tiempo de  escritura | Tiempo Total |
| Primer intento | 1 | 0 | 86 | 94 |
| Segundo intento | 0 | 0 | 83 | 90 |
| Tercer intento | 1 | 1 | 95 | 103 |
| Cuarto intento | 0 | 0 | 76 | 84 |
| Quinto intento | 1 | 0 | 81 | 87 |

Tabla 6.2. Prueba de escritura 2

El tiempo total menos el tiempo de escritura, es el tiempo empleado en desplazarse por el menú de botones y hacer los clics.

No se han observado tiempos significativos de retardo en las señales, después de ejecutar todas estas pruebas de manera secuencial y sin reiniciar la plataforma hardware (aproximadamente 45 minutos).

## **6.3. Pruebas de interoperabilidad**

Dado que la aplicación ha sido diseñada para cumplir el requisito de ser multiplataforma, ha sido probada sobre varios Sistemas Operativos:

* Windows 8 64bits (PC de desarrollo)
* Windows 7 32 bits (PC del laboratorio)
* Ubuntu 64bits (máquina virtual)

La aplicación ha funcionado según lo esperado en todos estos Sistemas Operativos, siguiendo los pasos descritos en el apartado 5.4.

# **CONCLUSIONES**

La consecución del objetivo de desarrollar un sistema de escritura funcional para personas con discapacidad grave que, en muchas ocasiones, es posible que sea la herramienta que les permita comunicarse por primera vez en mucho tiempo o incluso en la vida con sus seres queridos, ha resultado muy satisfactoria para el proyectante.

Un sistema de estas características podría mejorar de manera notable la calidad de vida de muchas personas que sufren discapacidades graves.

La realización de un proyecto de estas características implica la adquisición de nuevos y muy variados conocimientos. Desde los conocimientos adquiridos en el campo de la Biología a través del estudio de la anatomía del ojo, pasando por los conocimientos en Electrónica y Telecomunicaciones necesarios para el trabajo con las señales bioeléctricas y su adquisición mediante la plataforma hardware, así como para su procesamiento digital, y terminando con los conocimientos en Informática necesarios para el desarrollo del software, tanto en programación como en arquitectura de computadores.

Además, se ha ideado e implementado un nuevo método de escritura a través de las señales oculares que no existía hasta el momento, y que permite la escritura de manera más eficiente y cómoda, sin tener que desplazarse por el teclado virtual para seleccionar una letra, lo que implica varios movimientos y correcciones de posición. Mediante este método, se asegura alcanzar cualquier carácter alfanumérico de los disponibles con solo tres movimientos de los ojos.

Los equipos comerciales para electrooculografía disponibles en el mercado son en su mayoría muy caros (por encima de los 2.000€), como se observa en el apartado 2.4. El prototipo diseñado e implementado en este proyecto, permitiría disponer prácticamente de las mismas funcionalidades con las que cuentan los sistemas disponibles en el mercado, a costes mucho más reducidos (aproximadamente 200€, el coste de la fabricación de la plataforma hardware).

Ante una posible comercialización de un sistema basado en este prototipo, los costes de adquisición se verían reducidos en un 90% respecto de los equipos disponibles actualmente en el mercado, con lo que favorecería el acceso a este tipo de sistemas a prácticamente cualquier persona (al menos, eliminaría el factor económico de entre las causas por las que no resulta posible la adquisición de un sistema de estas características).

En lo que respecta a las pruebas efectuadas (capítulo 6), el sistema planteado ha demostrado ser lo suficientemente fiable para constituir una base sólida para el desarrollo de un sistema comercial. El ajuste de los parámetros de configuración descritos en el apartado 6.1, ha dado como resultado la correcta manejabilidad y eficiencia del sistema.

Las pruebas de escritura realizadas han resultado satisfactorias: los tiempos de escritura pueden ser notablemente inferiores respecto al teclado virtual tradicional. Además, estos tiempos también pueden ser reducidos con el uso sistemático de la aplicación, pudiendo un usuario llegar a alcanzar la capacidad de recordar los tres movimientos necesarios para alcanzar cualquier carácter alfanumérico de los disponibles en el sistema de escritura. Debe también notarse que la introducción de algún fallo al insertar una letra no supone una pérdida de tiempo grande.

La interfaz desarrollada cumple los objetivos de manejabilidad fijados en los objetivos del proyecto (apartado 1.1), ajustando sus funcionalidades a las capacidades disponibles de manejo de cursor y generación de clics de ratón. Además la aplicación puede ser utilizada en una gran variedad de Sistemas Operativos.

Es importante destacar también que, durante el transcurso de las investigaciones relativas a la adquisición de los conocimientos sobre bioseñales, ha sido imposible no tropezar en varias ocasiones en el uso de las técnicas de electroencefalografía (EEG) disponibles a día de hoy, y no percibir que es este campo el que está en auge en estos momentos. Dado el extendido conocimiento sobre el cerebro existente en la actualidad, la investigación de las posibilidades que ofrece la electroencefalografía para la implementación de este tipo de sistemas, al ser potencialmente mucho más versátil para el control de dispositivos, resulta muy atractiva. Todos los conocimientos adquiridos durante el desarrollo de este proyecto resultarían muy útiles para la exploración de este nuevo campo con el fin de desarrollar nuevos sistemas asistenciales para personas con discapacidades motoras severas, pero que de otra manera, cuentan con un cerebro perfectamente funcional.

# **POSIBLES MEJORAS DEL SISTEMA**

## **8.1. Uso de la transformada wavelet para la detección del parpadeo**

Durante el transcurso de las pruebas se ha observado que la señal producida por el parpadeo del ojo es muy parecida a la que genera el mismo al dirigir la mirada hacia arriba, saturando el canal vertical al igual que la mirada arriba, solo que al ser un movimiento rápido involuntario, la duración en el tiempo es menor.

Dadas estas características, se ha planteado la posibilidad de introducir un mecanismo de transformación de la señal en Tiempo o Frecuencia, con el fin de distinguir claramente el parpadeo de la mirada rápida hacia arriba, ya que ambas generan señales muy similares y esto podría derivar en una incorrecta interpretación de las señales por parte de la aplicación.

La transformada de Fourier (en Frecuencia) o la transformada wavelet (en Tiempo-Frecuencia) son dos opciones que podrían conseguir este objetivo, quedando el estudio e implementación de estos mecanismos para posibles ampliaciones futuras de este proyecto.

## **8.2. Aprendizaje automático (Machine learning)**

El aprendizaje automático es una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las máquinas aprender, lo que quiere decir que la máquina sea capaz de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos [24].

Esta mejora consistiría por tanto, en introducir mecanismos de aprendizaje para que la aplicación sepa interpretar de manera más eficiente y precisa las instrucciones de movimiento provenientes de la plataforma hardware. Para ello sería necesario implementar este mecanismo y efectuar numerosas pruebas con el objeto de que los algoritmos de aprendizaje obtuvieran una definición media de las señales que genera cada tipo de mirada. Ha de destacarse que este aprendizaje debe realizarse para cada usuario de manera particular (calibración), ya que, como se ha comentado en el apartado 2.4, cada sesión de uso del sistema puede variar con el individuo y su fisiología, así como con las condiciones ambientales.

Otra de las ventajas de utilizar esta tecnología sería la eliminación de las componentes externas no deseadas, ya que al tener aprendidas las señales generadas por cada movimiento ocular, la determinación de su tipo se produce por aproximación a estas señales, con lo que la señal final quedaría ya libre de interferencias y ruidos (tanto internos como externos).

## **8.3. Mejoras y nuevas funcionalidades de la aplicación**

Gracias al diseño modular de la aplicación, es posible añadir de manera fácil nuevas funcionalidades a la aplicación. Algunas de las posibles mejoras o nuevas funcionalidades que podrían añadirse a la aplicación son las siguientes:

* Desarrollo de un módulo que permita interactuar con el resto del Sistema Operativo, y no sólo dentro del ámbito de la aplicación. En este caso la aplicación funcionaría como método de escritura intermediario entre el usuario y la aplicación final a utilizar (por ejemplo, explorador de Internet, editor de textos, juegos...).
* Sería posible incorporar un sistema de predicción de texto que intuya las palabras que se van a introducir en el modo escritura por letras, en función de las primeras letras de la palabra introducidas.
* Un mecanismo de edición de las palabras disponibles en el módulo words (apartado 5.1.1) que permita configurar las palabras disponibles para su uso que aparecen en la ventana auxiliar, así como su número, de manera que se adapte a las necesidades de cada usuario.
* Cualquier ampliación que se proponga resultaría fácil de implementar siempre que se respete la estructura global de la aplicación y se tengan en cuenta las limitaciones de rendimiento

# **REFERENCIAS**

[1] Informe mundial sobre la discapacidad, Organización Mundial de la Salud (OMS)

http://www.who.int/disabilities/world\_report/2011/accessible\_es.pdf?ua=1

[2] Imagen: http://www.lucescei.com/estudios-y-eficiencia/

extractos-libro-blanco-de-iluminacion/el-sistema-visual-humano/

[3] Imagen: http://es.slideshare.net/guest5aa7da5/receptores-i

[4] Imagen: https://es.wikipedia.org/wiki/Ojo\_humano#/media/File:Eyesection-es.svg

[5] http://www.ece.mcmaster.ca/faculty/debruin/Engineering%20Education%204BD4/

Teacher%20Edition%20Lab%20Chapters/Electro-Oculography\_I\_Teacher.pdf

[6] Anatomía del ojo humano. National Eye Institute.

https://www.nei.nih.gov/health/espanol/saludable/saludable

[7] Imagen: http://www.facmed.unam.mx/Libro-NeuroFisio/Personas/Reymond/

Reymond.html

[8] Models of Horizontal Eye Movements: Early models of saccades and smooth pursuit, John D. Enderele (2010)

[9] Informationsaufnahme beim Befahren von Kurven, Psychologic für die Praxis, Cohen A. S. (1983)

[10] Imagen: www.chronos-vision.de

[11] Imagen: https://uxmag.com/topics/eye-tracking

[12] Imagen: http://www.inframedica.com/clinica\_del\_sueno.html

[13] Imagen: http://biofeedbackspain.es/productos/nuevos-productos/nexus-eeg-cap/

[14] Bio Muse, BioControl Systems. www.biocontrol.com

[15] Cyberlink Brain Actuated Technologies. www.brainfingers.com/cyberlink.htm

[16] Irisbond Tech. www.irisbond.com

[17] Cambridge Research Systems. http://www.crsltd.com/tools-for-vision-science

/eye-tracking/bluegain-eog-biosignal-amplifier/

[18] Alberto López Martínez. Julio 2012. Tesis Fin de Master en Tecnologías de la Información y comunicaciones en redes móviles. Sistema asistencial basado en computador empleando señales electrooculográficas orientado a personas discapacitadas.

[19] Cohen, A.S. Biomedical signal processing vol.I (1986)

[20] Bluetooth BlueGiga. https://www.bluegiga.com/en-US/products/wt12-bluetooth--class-2-module/

[21] Documentación Qt. https://www.qt.io/, http://doc.qt.io/qt-5/

[22] Biblioteca Microsoft Speech API (SAPI) 5.4. https://msdn.microsoft.com/en-us/

library/ee125663(v=vs.85).aspx2

[23] Documentación Qt sobre señales y slots. <http://doc.qt.io/qt-4.8/signalsandslots.html>

[24] https://es.wikipedia.org/wiki/Aprendizaje\_autom%C3%A1tico

# **A. ANEXO I: CÓDIGO FUENTE DE LA APLICACIÓN**

## **A.1. Clase principal (mainwindow)**

Mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QtWidgets>

/\* Bibliotecas Text to Speech \*/

#define \_ATL\_APARTMENT\_THREADED

#include <atlbase.h>

extern CComModule \_Module;

#include <atlcom.h>

#include <sapi.h>

/\* Bibliotecas propias \*/

#include "btreaderthread.h"

#include "dataparserthread.h"

#include "datatreatmentthread.h"

#include "decisionthread.h"

#include "writting.h"

#include "btsettingsdialog.h"

#include "words.h"

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

signals:

void openSignal();

void closeSignal();

private slots:

void newMovement(QList<int> coord);

void newBlink(bool);

void getLetter(QString letter);

void writtingClosed();

void wordsClosed();

//QButtons

void on\_ModoEscritura\_clicked();

void on\_Salir\_clicked();

void on\_BorrarTodo\_clicked();

void on\_BorrarLetra\_clicked();

void on\_CopiarTexto\_clicked();

void on\_Guardar\_clicked();

void on\_Configurar\_clicked();

void on\_Palabra\_clicked();

void on\_Leer\_clicked();

private:

Ui::MainWindow \*ui;

void createDefaultMapper();

void createDefaultConfig();

QList<int> processedTimeStamps;

QList<int> receivedTimeStamps;

QElapsedTimer timer;

QCursor \*cur;

BTReaderThread \*readerThread;

DataParserThread \*parserThread;

DataTreatmentThread \*treatmentThread;

DecisionThread \*decisionThread;

Writting \*writting;

BTSettingsDialog \*btDialog;

Words \*words;

int stage;

bool wordsRunning;

};

#endif // MAINWINDOW\_H

Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

/\* ---------------------- MAIN WINDOW --------------------- \*/

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

// Preparación de la interfaz gráfica de usuario.

ui->setupUi(this);

ui->centralWidget->setAutoFillBackground(true);

// Comprobación de la existencia de los archivos de

configuración mapper.ini y config.ini

// Si no existen llama a las funciones para crearlos con

los parámetros por defecto.

QFileInfo checkFileMap("configs/mapper.ini");

if (!(checkFileMap.exists() && checkFileMap.isFile())) {

createDefaultMapper();

}

QFileInfo checkFileCfg("configs/config.ini");

if (!(checkFileCfg.exists() && checkFileCfg.isFile())) {

createDefaultConfig();

}

// Definición de los hilos de lectura, parseo, tratamiento

y decision.

readerThread = new BTReaderThread(this);

parserThread = new DataParserThread(this);

treatmentThread = new DataTreatmentThread(this);

decisionThread = new DecisionThread(this);

// Conexiones entre las señales y los slots encargados del

procesamiento de los datos.

connect(readerThread,SIGNAL(DataBytesSignal(QByteArray)),

parserThread,SLOT(onDataBytes(QByteArray)));

connect(parserThread,SIGNAL(ChannelsDataSignal(QStringList

)),treatmentThread,SLOT(onChannelsData(QStringList)));

connect(treatmentThread,SIGNAL(ShowResultsSignal

(QStringList)),decisionThread,SLOT(onChannelResults

(QStringList)));

connect(decisionThread,SIGNAL(MovementSignal(QList

<int>)),this,SLOT(newMovement(QList<int>)));

connect(decisionThread,SIGNAL(BlinkSignal(bool)),

this,SLOT(newBlink(bool)));

//Se inician los hilos y se comienzan a procesar los datos

recibidos desde el BlueTooth.

readerThread->start();

decisionThread->start();

treatmentThread->start();

parserThread->start();

// Definición de la clase words.

wordsRunning = true;

words = new Words(this);

}

// ------------------------ SLOTS ------------------------ //

// Slot de actualización de coordenadas de cursor (posicion en ambos ejes), recibidas a través de la señal MovementSignal(QList<int>), de la clase decisionthread.

void MainWindow::newMovement(QList<int> coord)

{

QPoint p = cur->pos();

p.setY(p.y()+coord[1]);

p.setX(p.x()+coord[0]);

cur->setPos(p);

if (timer.isValid())

{ processedTimeStamps << timer.nsecsElapsed();}

else

{ processedTimeStamps << -1;

timer.start();}

}

// Slot para invocar el "click" de ratón mediante el parpadeo, invocado cuando decisionthread determina que ha habido un parpadeo y emite la señal MovementSignal(QList<int>).

void MainWindow::newBlink(bool performed)

{

if (performed){

QWidget \*p = qApp->widgetAt(cur->pos());

if (p){

QMetaObject::invokeMethod(p, "clicked");

}

}

}

// Slot para obtener la letra seleccionada mediante la clase Writting. Se emite una señal cada vez que una letra es seleccionada, la cual recibe el slot como argumento, para poder así mostrarla en la ventana principal de la aplicación (TextEdit).

void MainWindow::getLetter(QString letter)

{

QString text = ui->textEdit->toPlainText();

ui->textEdit->setText(text+letter);

}

// Slot para desconectar las señales de movimiento de la clase Writting al seleccionar una letra, y volver a conectarlas a los movimientos de cursor.

void MainWindow::writtingClosed()

{

disconnect(writting,SIGNAL(letterSelected(QString))

,this,SLOT(on\_ModoEscritura\_clicked()));

disconnect(treatmentThread,SIGNAL(ShowResultsSignal

(QStringList)),writting,SLOT(buffer(QStringList)));

connect(decisionThread,SIGNAL(MovementSignal(QList

<int>)),this,SLOT(newMovement(QList<int>)));

connect(decisionThread,SIGNAL(BlinkSignal(bool)),this

,SLOT(newBlink(bool)));

writting->close();

}

// Slot para desconectar la clase Words.

void MainWindow::wordsClosed()

{

wordsRunning = false;

disconnect(words,SIGNAL(wordSelected(QString)),this,SLOT

(on\_Palabra\_clicked()));

}

// -------------------- SLOTS DE QBUTTONS ---------------------- //

// Botón "Introducir Letra": se inicia la clase writting, donde está el algoritmo para seleccionar la letra mediante las crucetas. Se realiza la desconexión de los hilos de tratamiento a los de decisión y se conectan los de la clase Writting: los movimientos de los ojos pasan a usarse para seleccionar una letra mediante las crucetas, y no para el control del cursor. Al terminar se reestablecen las conexiones (writtingClosed()).

// Nota: La función se llama a sí misma (recursividad) hasta llegar a la letra (3 iteraciones).

void MainWindow::on\_ModoEscritura\_clicked()

{

disconnect(decisionThread,SIGNAL(MovementSignal(QList

<int>)),this,SLOT(newMovement(QList<int>)));

disconnect(decisionThread,SIGNAL(BlinkSignal(bool)),this

,SLOT(newBlink(bool)));

writting = new Writting(this);

connect(treatmentThread,SIGNAL(ShowResultsSignal

(QStringList)),writting,SLOT(buffer(QStringList)));

writting->show();

}

// Botón "Salir": la aplicación se cierra.

void MainWindow::on\_Salir\_clicked()

{

this->close();

}

// Botón "Borrar Todo": elimina todo el texto introducido en la ventana principal (TextEdit).

void MainWindow::on\_BorrarTodo\_clicked()

{

ui->textEdit->clear();

}

// Botón "Borrar Letra": elimina el último caracter introducido en la ventana principal (TextEdit).

void MainWindow::on\_BorrarLetra\_clicked()

{

QString text;

text = ui->textEdit->toPlainText();

QString texto;

texto = text.left(text.length()-1);

ui->textEdit->setText(texto);

}

// Botón "Copiar a Portapapeles": copia el texto introducido en la ventana principal (TextEdit) al portapapeles del Sistema Operativo. Después da la opción de eliminar el texto ya copiado.

void MainWindow::on\_CopiarTexto\_clicked()

{

QString text;

text = ui->textEdit->toPlainText();

QClipboard \*clipboard = QApplication::clipboard();

clipboard->setText(text);

// Diálogo para borrar o no el texto copiado

QMessageBox msg;

msg.setText("<html><head/><body><p><span style=\" font-

size:15pt; font-weight:600;\">Texto copiado al

Portapapeles. ¿Desea borrar el texto introducido?");

msg.setStandardButtons(QMessageBox::Ok |

QMessageBox::Cancel);

msg.setStyleSheet("QLabel{height: 200px;}

QPushButton{width:250px; height:75px; font-size: 24px;}");

int ret = msg.*exec*();

switch (ret) {

case QMessageBox::Ok:

ui->textEdit->clear();

break;

case QMessageBox::Cancel:

break;}

}

// Botón "Guardar Texto": crea un fichero con nombre igual a la fecha y hora del sistema y guarda el texto introducido en la ventana principal (TextEdit) en un archivo de texto en la carpeta "texts", en la misma localización que el ejecutable. Después da la opción de eliminar el texto ya guardado.

void MainWindow::on\_Guardar\_clicked()

{

QString dir = "texts/";

QDate dateq = QDate::currentDate();

QString date = dateq.toString("ddMMMMyyyy");

QTime hourq = QTime::currentTime();

QString hour = hourq.toString("hhmmss");

QString ext = ".txt";

QString fichero = dir+date+"."+hour+ext;

QString fichero\_display = date+"."+hour+ext;

QFile file(fichero);

file.*open*(QIODevice::WriteOnly | QIODevice::Text);

QTextStream texto(&file);

texto << ui->textEdit->toPlainText();

// Diálogo para borrar o no el texto copiado

QMessageBox msg;

msg.setText("<html><head/><body><p><span style=\" font-

size:15pt; font-weight:600;\">Texto guardado en el fichero

"+fichero\_display+".\n¿Desea borrar el texto

introducido?");

msg.setStandardButtons(QMessageBox::Ok |

QMessageBox::Cancel);

msg.setStyleSheet("QLabel{height: 200px;}

QPushButton{width:250px; height:75px; font-size: 24px;}");

int ret = msg.*exec*();

switch (ret) {

case QMessageBox::Ok:

ui->textEdit->clear();

break;

case QMessageBox::Cancel:

break;}

}

// Botón "Configurar": abre un diálogo para la configuración de los parámetros de conexión BlueTooth.

void MainWindow::on\_Configurar\_clicked()

{

btDialog = new BTSettingsDialog();

btDialog->setModal(true);

btDialog->show();

}

// Botón "Introducir Palabra": llama a la clase Words, que despliega un menú con palabras preintroducidas para seleccionar y que esta aparezca en la ventana principal (TextEdit). Se emplea una lógica de estados simple para la conexión entre las dos ventanas y evitar diálogos duplicados, ya que las funcionalidades de movimiento del cursor siguen activas.

void MainWindow::on\_Palabra\_clicked()

{

if (wordsRunning)

{

words->show();

QString selWord = words->word;

QString text = ui->textEdit->toPlainText();

QString displayText = text+selWord;

ui->textEdit->setText(displayText);

words->word.clear();

}

else

{

words = new Words(this);

words->show();

wordsRunning = true;

QString selWord = words->word;

QString text = ui->textEdit->toPlainText();

QString displayText = text+selWord;

ui->textEdit->setText(displayText);

words->word.clear();

}

}

// Botón "Leer": se utilizan las bibliotecas "Speech API"(SAPI) de Microsoft(R) para reproducir mediante voz el texto introducido en la ventana principal (TextEdit).

void MainWindow::on\_Leer\_clicked()

{

ISpVoice \* pVoice = NULL;

if (FAILED(::CoInitialize(NULL)))

QMessageBox::critical(this, "Error", "CoInitialize

failed!");

HRESULT hr = CoCreateInstance(CLSID\_SpVoice, NULL,

CLSCTX\_ALL,IID\_ISpVoice, (void \*\*)&pVoice);

if( SUCCEEDED( hr ) )

{

hr = pVoice->*Speak*(ui->textEdit->

toPlainText().toStdWString().c\_str(),

SPF\_IS\_XML, NULL);

pVoice->*Release*();

pVoice = NULL;

}

::CoUninitialize();

}

// ---------------- FUNCIONES AUXILIARES ------------------ //

// Función para crear el fichero mapper.ini con los valores por defecto

void MainWindow::createDefaultMapper()

{

QSettings mapper(QString("configs/mapper.ini"),

QSettings::IniFormat);

mapper.setValue("No",0);

mapper.setValue("Hardware",1);

mapper.setValue("Software",2);

mapper.setValue("Even",2);

mapper.setValue("Odd",3);

mapper.setValue("Space",4);

mapper.setValue("Mark",5);

mapper.setValue("1",1);

mapper.setValue("2",2);

mapper.setValue("1.5",3);

mapper.setValue("Desconocido",-1);

}

// Función para crear el fichero config.ini con los valores por defecto

void MainWindow::createDefaultConfig()

{

QSettings mapper(QString("configs/config.ini"),

QSettings::IniFormat);

mapper.setValue("PortName","COM7");

mapper.setValue("BaudRate",1200);

mapper.setValue("DataBits",8);

mapper.setValue("StopBits",1);

mapper.setValue("Parity","No");

mapper.setValue("FlowControl","Hardware");

mapper.setValue("Upthreshold",1000);

mapper.setValue("Downthreshold",150);

mapper.setValue("Speed",3);

mapper.setValue("Minblink",10);

mapper.setValue("Maxblink",20);

}

## **A.2. Lectura de datos (btreaderthread)**

btreaderthread.h

#ifndef BTREADERTHREAD\_H

#define BTREADERTHREAD\_H

#include <QtCore>

#include <QtSerialPort/QSerialPort>

class BTReaderThread : public QThread

{

Q\_OBJECT

public:

explicit BTReaderThread(QObject \*parent = 0);

void *run*();

private slots:

void readData();

void handleError(QSerialPort::SerialPortError error);

void openSerialPort();

void closeSerialPort();

signals:

void ClosedSignal();

void OpenedSignal();

void DataBytesSignal(QByteArray);

void ShowErrorSignal(QString);

void SampleReadSignal();

void FinishSignal();

private:

QSerialPort \*serial;

int sleeptime;

QStringList list;

};

#endif // BTREADERTHREAD\_H

btreaderthread.cpp

#include "btreaderthread.h"

/\* ------------- HILO DE LECTURA POR BLUETOOTH -------------\*/

BTReaderThread::BTReaderThread(QObject \*parent) : QThread (parent)

{

serial = new QSerialPort(this);

openSerialPort();

}

void BTReaderThread::*run*()

{

// Conexión entre el objeto QSerialPort (BlueTooth) y los

slots que manejarán los datos.

if (serial!=NULL){

connect(serial, SIGNAL(error(QSerialPort::

SerialPortError)) ,this,SLOT(handleError

(QSerialPort::SerialPortError)));

connect(serial, SIGNAL(readyRead()), this,

SLOT(readData()));

}

else {

emit ShowErrorSignal("Error al iniciar el puerto

serie");

}

}

void BTReaderThread::openSerialPort()

{

// Lectura, parseado y asignación de los parámetros de

configuración del BlueTooth desde los ficheros de

configuración de la aplicación.

QSettings settings(QString("configs/config.ini"),

QSettings::IniFormat);

QSettings map(QString("configs/mapper.ini"),

QSettings::IniFormat);

serial->setPortName

(settings.value("PortName").toString());

serial->setBaudRate(settings.value("BaudRate").toInt());

serial->setDataBits(QSerialPort::DataBits(map.value

(settings.value("DataBits").toString()).toInt()));

serial->setStopBits

(QSerialPort::StopBits(map.value(settings

.value("StopBits").toString()).toInt()));

serial->setParity(QSerialPort::Parity(map.value(settings

.value("Parity").toString()).toInt()));

serial->setFlowControl(QSerialPort::FlowControl(map.value

(settings.value("FlowControl").toString()).toInt()));

// Comprobación para saber si el dispositivo está listo

para transmitir.

if (serial->*open*(QIODevice::ReadOnly)) {

emit OpenedSignal();

} else {

emit ShowErrorSignal(serial->errorString());

}

}

// Slot para cerrar la conexión con el dispositivo BlueTooth.

void BTReaderThread::closeSerialPort()

{

serial->*close*();

emit ClosedSignal();

}

// Slot para leer los datos recibidos desde el BlueTooth y emitir una señal con un array de bytes que contiene dichos datos. Esta señal será recogida por el hilo de parseo de datos.

void BTReaderThread::readData()

{

QByteArray data = serial->readLine();

emit DataBytesSignal(data);

}

// Slot para mostrar un mensaje de error en caso de haber algún problema en el establecimiento de la conexión BlueTooth.

void BTReaderThread::handleError(QSerialPort::SerialPortError error)

{

if (error == QSerialPort::ResourceError) {

closeSerialPort();

emit ShowErrorSignal(serial->errorString());

}

}

## **A.3. Mapeado de datos (dataparserthread)**

dataparserthread.h

#ifndef DATAPARSERTHREAD\_H

#define DATAPARSERTHREAD\_H

#include <QtCore>

class DataParserThread : public QThread

{

Q\_OBJECT

public:

explicit DataParserThread(QObject \*parent = 0);

signals:

void ChannelsDataSignal(QStringList);

void SampleReadSignal();

private slots:

void onDataBytes(QByteArray data);

private:

QByteArray queue;

};

#endif // DATAPARSERTHREAD\_H

dataparserthread.cpp

#include "dataparserthread.h"

/\* -------------- HILO DE PREPARACIóN DE DATOS ------------ \*/

DataParserThread::DataParserThread(QObject \*parent) : QThread (parent)

{

}

// Slot que se lanza cada vez que se emite la señal DataBytesSignal(QByteArray) de la clase btreaderthread (conexiones en mainwindow). Almacena los bytes recibidos en un buffer, hasta que llegue un salto de línea. Extrae los datos de los dos canales (vertical y horizontal) y emite una señal enviando estos datos, convertidos de bytes a strings, al hilo de tratamiento.

void DataParserThread::onDataBytes(QByteArray data){

queue+=data;

int index = queue.indexOf("\n\r", 0);

if (index >= 0){

QString line = QString(queue.mid(0, index));

queue.remove(0, index + 1);

QStringList channels = line.split(" ");

emit ChannelsDataSignal(channels);

}

}

## **A.4. Tratamiento de datos (datatreatmentthread)**

datatreatmentthread.h

#ifndef DATATREATMENTTHREAD\_H

#define DATATREATMENTTHREAD\_H

#include <QtCore>

class DataTreatmentThread : public QThread

{

Q\_OBJECT

public:

explicit DataTreatmentThread(QObject \*parent = 0);

private slots:

void onChannelsData(QStringList channels);

signals:

void ShowResultsSignal(QStringList);

private:

double lowerThreshold;

double upperThreshold;

};

#endif // DATATREATMENTTHREAD\_H

datatreatmentthread.cpp

#include "datatreatmentthread.h"

/\* -------------- HILO DE TRATAMIENTO DE DATOS ------------ \*/

DataTreatmentThread::DataTreatmentThread(QObject \*parent) : QThread (parent)

{

// Carga de valores umbrales que determinan la dirección

de la mirada en los ejes vertical y horizontal, desde fichero

de configuración. Si falla asigna valores por defecto.

QSettings settings(QString("configs/config.ini"),

QSettings::IniFormat);

if (settings.value("Upthreshold").toString().toInt()>0){

upperThreshold = settings.value("Upthreshold")

.toString().toInt();}

else upperThreshold = 1000;

if (settings.value("Downthreshold").toString().toInt()>0){

lowerThreshold = settings.value("Downthreshold")

.toString().toInt();}

else lowerThreshold = 150;

}

// Slot de tratamiento de datos. Recibe datos a través de la señal ChannelsDataSignal(QStringList) de la clase dataparserthread. Se separan los canales en horizontal y vertical, y se decide hacia dónde se dirige la mirada en función de los umbrales definidos. Los resultados se envían al slot onChannelResults(QStringList) del hilo de tratamiento, a través de la señal ShowResultsSignal(results) separados por canales y con el resultado final de la dirección de la mirada.

void DataTreatmentThread::onChannelsData(QStringList channels)

{

int vertical = channels[1].toDouble();

int horizontal = channels[0].toDouble();

QString VResult="-";

QString HResult="-";

if (vertical>upperThreshold) {

VResult = "DOWN";}

if (vertical<lowerThreshold) {

VResult = "UP";}

if (horizontal>upperThreshold){

HResult = "LEFT";}

if (horizontal<lowerThreshold){

HResult = "RIGHT";}

QStringList results;

results << HResult << VResult;

emit ShowResultsSignal(results);

}

## **A.5. Toma de decisiones (decisionthread)**

decisionthread.h

#ifndef DECISIONTHREAD\_H

#define DECISIONTHREAD\_H

#include <QtCore>

class DecisionThread : public QThread

{

Q\_OBJECT

public:

explicit DecisionThread(QObject \*parent = 0);

private slots:

void onChannelResults(QStringList results);

signals:

void MovementSignal(QList<int>);

void BlinkSignal(bool);

void sendSamples(QList<QStringList> samples);

void newDecision(QStringList sample);

private:

void checkBlink();

void clearBuffer();

QList<QStringList> samples;

int units;

int blinkCounter;

int lowerBlink;

int upperBlink;

};

#endif // DECISIONTHREAD\_H

decisionthread.cpp

#include "decisionthread.h"

/\* ------------------ HILO DE TOMA DE DECISIÓN ---------------- \*/

DecisionThread::DecisionThread(QObject \*parent) : QThread (parent)

{

blinkCounter = 0; // Se inicializa a 0 contador de parpadeos

// Carga por fichero de los parámetros de configuración de

salto de cursor y umbrales de parpadeo.

QSettings settings(QString("configs/config.ini"),

QSettings::IniFormat);

if (settings.value("Speed").toString().toInt()>0){

units = settings.value("Speed").toString().toInt();}

else units = 3;

if (settings.value("Minblink").toString().toInt()>0){

lowerBlink = settings.value("Minblink").toString()

.toInt();}

else lowerBlink = 10;

if (settings.value("Maxblink").toString().toInt()>0){

upperBlink = settings.value("Maxblink").toString()

.toInt();}

else upperBlink = 20;

}

// Slot que recibe los datos de la dirección de la mirada de la clase datatreatmentthread y genera un vector de coordenadas que se envia a través de la señal MovementSignal(coord), conectada con el slot newMovement(QList<int>) de mainwindow, que moverá el cursor del ratón en función de la dirección a la que apunta la mirada. Al procesar los dos canales simultáneamente es capaz de emitir movimientos en diagonal.

void DecisionThread::onChannelResults(QStringList results)

{

samples.push\_back(results);

if (samples.length()>=30)

{

checkBlink();

QStringList sample = samples[0];

emit newDecision(sample);

samples.pop\_front();

int hMov=0;

int vMov=0;

if(sample[1]=="UP"){

vMov = -units;}

if(sample[1]=="DOWN"){

vMov = units;}

if(sample[0]=="RIGHT"){

hMov = units;}

if(sample[0]=="LEFT"){

hMov = -units;}

QList<int> coord;

coord << hMov << vMov;

emit MovementSignal(coord);

}

}

// Función para comprobar si la señal recibida corresponde a un parpadeo. Si corresponde, se emite la señal BlinkSignal(true), conectada con el slot newBlink(bool) de mainwindow, que invoca a la función "click" del ratón en lugar de mover el cursor.

void DecisionThread::checkBlink()

{

emit sendSamples(samples);

for (int i=0;i<samples.length();i++)

{

if ((samples[i])[1]=="UP")

{

blinkCounter++;

emit BlinkSignal(false);

}

else

{

if ((blinkCounter > lowerBlink)&&(blinkCounter <

upperBlink)&&(i>lowerBlink)&&(i<25))

{

clearBuffer();

emit BlinkSignal(true);

}

else

{

emit BlinkSignal(false);

}

blinkCounter = 0;

}

}

blinkCounter = 0;

}

// Función para liberar el buffer de datos.

void DecisionThread::clearBuffer()

{

for (int i=0; i<samples.length();i++)

{

(samples[i])[1] = "-";

}

emit sendSamples(samples);

}

## **A.6. Escritura de letras (writting)**

writting.h

#ifndef WRITTING\_H

#define WRITTING\_H

#include <QImage>

#include <QPixmap>

#include <QtCore>

#include <QtWidgets>

#include "datatreatmentthread.h"

#include "btreaderthread.h"

#include "dataparserthread.h"

namespace Ui {

class Writting;

}

class Writting : public QDialog

{

Q\_OBJECT

public:

explicit Writting(QWidget \*parent = 0);

void closingWritting();

QString letra;

signals:

void letterSelected(QString);

void closeWritting();

private slots:

void buffer(QStringList);

private:

Ui::Writting \*ui;

void showImage(int);

void selectLetter(QString);

int stage;

int countUP;

int countDOWN;

int countRIGHT;

int countLEFT;

QString select;

};

#endif // WRITTING\_H

writting.cpp

#include "writting.h"

#include "ui\_writting.h"

/\* --------------------- ESCRITURA DE LETRAS ------------------- \*/

Writting::Writting(QWidget \*parent) :

QDialog(parent),

ui(new Ui::Writting)

{

// Preparación de la interfaz gráfica de usuario.

ui->setupUi(this);

setWindowTitle("Seleccione una letra");

// Inicialización de variables

stage = 1;

countUP = 0;

countDOWN = 0;

countRIGHT = 0;

countLEFT = 0;

// Llamada a la función auxiliar que muestra las imágenes de

las crucetas

Writting::showImage(stage);

// Se establecen las conexiones entre señales y slots

encargadas de tomar el control de los movimientos hasta

llegar a la letra a seleccionar.

connect(this,SIGNAL(letterSelected(QString)),this->

parent(),

SLOT(getLetter(QString)));

connect(this,SIGNAL(closeWritting()),this->parent(),

SLOT(writtingClosed()));

}

// Cierra el diálogo de las crucetas y emite la señal closeWritting() para terminar las conexiones a Writting y devolver el control a MainWindow.

void Writting::closingWritting()

{

delete ui;

emit closeWritting();

}

// Función auxiliar de buffering utilizada para que la velocidad de recepción de los movimientos se ajuste a la manejabilidad del sistema de crucetas por parte del usuario. Una vez decidida la dirección llama a la función selectLetter(QString) pasándole ésta como argumento.

void Writting::buffer(QStringList results)

{

if (results[1] == "UP"){

countUP++;

if (countUP>30)

{

countUP = 0;

countDOWN = 0;

countRIGHT = 0;

countLEFT = 0;

Writting::selectLetter("UP");

}

}if (results[1] == "DOWN"){

countDOWN++;

if (countDOWN>30)

{

countUP = 0;

countDOWN = 0;

countRIGHT = 0;

countLEFT = 0;

Writting::selectLetter("DOWN");

}

}if (results[0] == "RIGHT"){

countRIGHT++;

if (countRIGHT>30)

{

countUP = 0;

countDOWN = 0;

countRIGHT = 0;

countLEFT = 0;

Writting::selectLetter("RIGHT");

}

}if (results[0] == "LEFT"){

countLEFT++;

if (countLEFT>30)

{

countUP = 0;

countDOWN = 0;

countRIGHT = 0;

countLEFT = 0;

Writting::selectLetter("LEFT");

}

}

}

// Algoritmo de selección de letra. Recibe por parámetro el resultado del movimiento y asigna un nuevo valor de etapa (stage) para moverse a través del menú de crucetas hasta llegar a la letra a seleccionar. Una vez seleccionada, emite una señal que será recogida por el stot getLetter(QString letter) de MainWindow para imprimir en la ventana principal (TextEdit).

void Writting::selectLetter(QString result)

{

select = result;

switch(stage){

case 1:

{if(select=="LEFT"){

stage = 2;

} if(select=="UP"){

stage = 3;

} if(select=="RIGHT"){

stage = 4;

} if(select=="DOWN"){

stage = 5;}

select.clear();

Writting::showImage(stage);

break;}

case 2:

{if(select=="LEFT"){

stage = 6;

} if(select=="UP"){

stage = 7;

} if(select=="RIGHT"){

stage = 8;

} if(select=="DOWN"){

stage = 9;}

select.clear();

Writting::showImage(stage);

break;}

case 3:

{if(select=="LEFT"){

stage = 10;

} if(select=="UP"){

stage = 11;

} if(select=="RIGHT"){

stage = 12;

} if(select=="DOWN"){

stage = 13;}

select.clear();

Writting::showImage(stage);

break;}

case 4:

{if(select=="LEFT"){

stage = 14;

} if(select=="UP"){

stage = 15;

} if(select=="RIGHT"){

stage = 16;

} if(select=="DOWN"){

stage = 17;}

select.clear();

Writting::showImage(stage);

break;}

case 5:

{if(select=="LEFT"){

stage = 18;

} if(select=="UP"){

stage = 19;

} if(select=="RIGHT"){

stage = 20;

} if(select=="DOWN"){

letra = " ";

break;}

select.clear();

Writting::showImage(stage);

break;}

case 6:

{if(select=="LEFT"){

letra = "A";

} if(select=="UP"){

letra = "B";

} if(select=="RIGHT"){

letra = "C";

} if(select=="DOWN"){

letra = "D";}

break;}

case 7:

{if(select=="LEFT"){

letra = "E";

} if(select=="UP"){

letra = "F";

} if(select=="RIGHT"){

letra = "G";

} if(select=="DOWN"){

letra = "H";}

break;}

case 8:

{if(select=="LEFT"){

letra = "I";

} if(select=="UP"){

letra = "J";

} if(select=="RIGHT"){

letra = "K";

} if(select=="DOWN"){

letra = "L";}

break;}

case 9:

{if(select=="LEFT"){

letra = "M";

} if(select=="UP"){

letra = "N";

} if(select=="RIGHT"){

letra = "O";

} if(select=="DOWN"){

letra = "P";}

break;}

case 10:

{if(select=="LEFT"){

letra = "Q";

} if(select=="UP"){

letra = "R";

} if(select=="RIGHT"){

letra = "S";

} if(select=="DOWN"){

letra = "T";}

break;}

case 11:

{if(select=="LEFT"){

letra = "U";

} if(select=="UP"){

letra = "V";

} if(select=="RIGHT"){

letra = "W";

} if(select=="DOWN"){

letra = "X";}

break;}

case 12:

{if(select=="LEFT"){

letra = "Y";

} if(select=="UP"){

letra = "Z";

} if(select=="RIGHT"){

letra = "Á";

} if(select=="DOWN"){

letra = "É";}

break;}

case 13:

{if(select=="LEFT"){

letra = "Í";

} if(select=="UP"){

letra = "Ó";

} if(select=="RIGHT"){

letra = "Ú";

} if(select=="DOWN"){

letra = "Ñ";}

break;}

case 14:

{if(select=="LEFT"){

letra = "0";

} if(select=="UP"){

letra = "1";

} if(select=="RIGHT"){

letra = "2";

} if(select=="DOWN"){

letra = "3";}

break;}

case 15:

{if(select=="LEFT"){

letra = "4";

} if(select=="UP"){

letra = "5";

} if(select=="RIGHT"){

letra = "6";

} if(select=="DOWN"){

letra = "7";}

break;}

case 16:

{if(select=="LEFT"){

letra = "8";

} if(select=="UP"){

letra = "9";

} if(select=="RIGHT"){

letra = "+";

} if(select=="DOWN"){

letra = "-";}

break;}

case 17:

{if(select=="LEFT"){

letra = "/";

} if(select=="UP"){

letra = "\*";

} if(select=="RIGHT"){

letra = "%";

} if(select=="DOWN"){

letra = "=";}

break;}

case 18:

{if(select=="LEFT"){

letra = ".";

} if(select=="UP"){

letra = ",";

} if(select=="RIGHT"){

letra = "?";

} if(select=="DOWN"){

letra = "!";}

break;}

case 19:

{if(select=="LEFT"){

letra = "\'";

} if(select=="UP"){

letra = "\"";

} if(select=="RIGHT"){

letra = "(";

} if(select=="DOWN"){

letra = ")";}

break;}

case 20:

{if(select=="LEFT"){

letra = "@";

} if(select=="UP"){

letra = "\_";

} if(select=="RIGHT"){

letra = "#";

} if(select=="DOWN"){

letra = "&";}

break;}

}

// Comprobación de letra seleccionada

if (!letra.isEmpty())

{

emit letterSelected(letra);

letra.clear();

select.clear();

stage = 1;

Writting::closeWritting();

}

}

// Función auxiliar para mostrar de forma dinámica las imágenes de las crucetas, en función de la etapa en que se encuentre el usuario. Cada vez que se selecciona un conjunto de letras de las crucetas y se pasa a la siguiente etapa, esta función va mostrando las imágenes correspondientes a cada una de las etapas hasta llegar a la letra seleccionada.

void Writting::showImage(int stage)

{

QDir directory(":/images/");

QString path = directory.filePath("Cruceta%1.jpg")

.arg(stage);

QPixmap pixmap(path);

ui->label->setPixmap(pixmap);

}

## **A.7. Escritura de palabras (words)**

words.h

#ifndef WORDS\_H

#define WORDS\_H

#include <QDialog>

#include <QtWidgets>

namespace Ui {

class Words;

}

class Words : public QDialog

{

Q\_OBJECT

public:

explicit Words(QWidget \*parent = 0);

~Words();

QString word;

signals:

void wordSelected(QString word);

void closeWords();

private slots: //QButtons

void on\_yo\_clicked();

void on\_tu\_clicked();

void on\_el\_ella\_clicked();

void on\_nosotros\_clicked();

void on\_vosotros\_clicked();

void on\_ellos\_ellas\_clicked();

void on\_ser\_clicked();

void on\_ir\_clicked();

void on\_querer\_clicked();

void on\_poder\_clicked();

void on\_pasear\_clicked();

void on\_decir\_clicked();

void on\_casa\_clicked();

void on\_calle\_clicked();

void on\_parque\_clicked();

void on\_tener\_clicked();

void on\_estar\_clicked();

void on\_coche\_clicked();

void on\_agua\_clicked();

void on\_fruta\_clicked();

void on\_carne\_clicked();

void on\_pescado\_clicked();

void on\_dulce\_clicked();

void on\_salado\_clicked();

void on\_em\_alegre\_clicked();

void on\_em\_triste\_clicked();

void on\_si\_clicked();

void on\_no\_clicked();

void on\_espacio\_clicked();

void on\_terminado\_clicked();

void on\_haber\_clicked();

void on\_mirar\_clicked();

void on\_hacer\_clicked();

void on\_oir\_clicked();

void on\_esto\_clicked();

void on\_aquello\_clicked();

private:

Ui::Words \*ui;

};

#endif // WORDS\_H

words.cpp

#include "words.h"

#include "ui\_words.h"

/\* ----------------- ESCRITURA DE PALABRAS --------------- \*/

Words::Words(QWidget \*parent) :

QDialog(parent),

ui(new Ui::Words)

{

// Preparación de la interfaz gráfica de usuario.

ui->setupUi(this);

setWindowTitle("Seleccione una palabra");

ui->textEditWords->setAutoFillBackground(true);

ui->label->setAutoFillBackground(true);

// Se establecen las conexiones para imprimir las palabras

seleccionadas en la ventana principal (TextEdit) hasta que

se cierre el diálogo.

connect(this,SIGNAL(closeWords()),this->parent(),

SLOT(wordsClosed()));

connect(this,SIGNAL(wordSelected(QString)),this->parent(),

SLOT(on\_Palabra\_clicked()));

}

// Liberación memoria de la interfaz gráfica de usuario al salir.

Words::~*Words*()

{

delete ui;

}

/\* ----------------------- QBUTTONS ----------------------- \*/

// Cuando un botón es pulsado, se asigna su correspondiente palabra a la variable global 'word' y se emite una señal que la contiene. Esta será usada por el slot 'on\_Palabra\_clicked()' de MainWindow para imprimirla en la ventana principal. Pueden ser seleccionadas una o varias palabras al mismo tiempo.

void Words::on\_yo\_clicked()

{

word = "YO";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_tu\_clicked()

{

word = "TÚ";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_el\_ella\_clicked()

{

word = "ÉL/ELLA";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_nosotros\_clicked()

{

word = "NOSOTROS";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_vosotros\_clicked()

{

word = "VOSOTROS";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_ellos\_ellas\_clicked()

{

word = "ELLOS/ELLAS";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_ser\_clicked()

{

word = "SER";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_ir\_clicked()

{

word = "IR";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_querer\_clicked()

{

word = "QUERER";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_poder\_clicked()

{

word = "PODER";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_pasear\_clicked()

{

word = "PASEAR";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_decir\_clicked()

{

word = "DECIR";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_casa\_clicked()

{

word = "CASA";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_calle\_clicked()

{

word = "CALLE";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_parque\_clicked()

{

word = "PARQUE";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_tener\_clicked()

{

word = "TENER";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_estar\_clicked()

{

word = "ESTAR";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_coche\_clicked()

{

word = "COCHE";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_agua\_clicked()

{

word = "AGUA";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_fruta\_clicked()

{

word = "FRUTA";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_carne\_clicked()

{

word = "CARNE";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_pescado\_clicked()

{

word = "PESCADO";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_dulce\_clicked()

{

word = "DULCE";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_salado\_clicked()

{

word = "SALADO";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_em\_alegre\_clicked()

{

word = ":-)";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_em\_triste\_clicked()

{

word = ":-(";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_si\_clicked()

{

word = "SÍ";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_no\_clicked()

{

word = "NO";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_espacio\_clicked()

{

word = " ";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_haber\_clicked()

{

word = "HABER";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_mirar\_clicked()

{

word = "MIRAR";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_hacer\_clicked()

{

word = "HACER";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_oir\_clicked()

{

word = "OÍR";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_esto\_clicked()

{

word = "ESTO";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

void Words::on\_aquello\_clicked()

{

word = "AQUELLO";

QString text = ui->textEditWords->toPlainText();

ui->textEditWords->setText(text+word);

emit wordSelected(word);

}

// Cierra el diálogo Words

void Words::on\_terminado\_clicked()

{

emit closeWords();

this->~Words();

}

## **A.8. Menú de configuración (btsettingsdialog)**

btsettingsdialog.h

#ifndef BTSETTINGSDIALOG\_H

#define BTSETTINGSDIALOG\_H

#include <QDialog>

#include <QSerialPortInfo>

#include <QSettings>

namespace Ui {

class BTSettingsDialog;

}

class BTSettingsDialog : public QDialog

{

Q\_OBJECT

public:

explicit BTSettingsDialog(QWidget \*parent = 0);

~*BTSettingsDialog*();

private slots:

void on\_nameComboBox\_currentIndexChanged(int index);

void on\_buttonBox\_accepted();

private:

Ui::BTSettingsDialog \*ui;

QList<QSerialPortInfo> infoPorts;

};

#endif // BTSETTINGSDIALOG\_H

btsettingsdialog.cpp

#include "btsettingsdialog.h"

#include "ui\_btsettingsdialog.h"

/\* ---------- DIÁLOGO DE CONFIGURACIÓN BLUETOOTH ---------- \*/

BTSettingsDialog::BTSettingsDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::BTSettingsDialog)

{

// Se lanza el interfaz gráfico para seleccionar la

configuración del BlueTooth.

ui->setupUi(this);

// Enumeración de la opciones de configuración disponibles.

QSettings settings(QString("configs/config.ini"),

QSettings::IniFormat);

// Puertos COM.

infoPorts = QSerialPortInfo::availablePorts();

for (int i = 0; i<infoPorts.size(); i++){

ui->nameComboBox->addItem(infoPorts.at(i).portName());}

// Bits de datos.

QStringList dataBits;

dataBits << "5" << "6" << "7" << "8" << "Desconocido";

ui->dataBitsComboBox->addItems(dataBits);

ui->dataBitsComboBox->setCurrentIndex(ui->

dataBitsComboBox->findText(settings.value

("DataBits").toString()));

// Control de flujo.

QStringList flowControl;

flowControl << "No" << "Hardware" << "Software" <<

"Desconocido";

ui->flowComboBox->addItems(flowControl);

ui->flowComboBox->setCurrentIndex(ui->flowComboBox->

findText(settings.value("FlowControl").toString()));

// Paridad.

QStringList parity;

parity << "No" << "Even Parity" << "Odd Parity" << "Space

Parity" << "Mark Parity" << "Desconocido";

ui->parityComboBox->addItems(parity);

ui->parityComboBox->setCurrentIndex(ui->parityComboBox->

findText(settings.value("Parity").toString()));

// Bits de parada.

QStringList stopBits;

stopBits << "1" << "2" << "1.5" << "Desconocido";

ui->stopBitsComboBox->addItems(stopBits);

ui->stopBitsComboBox->setCurrentIndex(ui->

stopBitsComboBox->findText(settings.value

("StopBits").toString()));

}

// Liberación memoria de la interfaz gráfica de usuario al salir.

BTSettingsDialog::~*BTSettingsDialog*()

{

delete ui;

}

// Slot para mostrar las nuevas tasas de Baudios disponibles al cambiar de puerto COM.

void BTSettingsDialog::on\_nameComboBox\_currentIndexChanged(int index)

{

QList<qint32> baudRates = infoPorts.at(index)

.standardBaudRates();

for (int i = 0; i<baudRates.size(); i++)

{

ui->baudComboBox->addItem(QString::number

(baudRates.at(i)));

}

QSettings settings(QString("configs/config.ini"),

QSettings::IniFormat);

int indexBR = ui->baudComboBox->findText(settings

.value("BaudRate").toString());

ui->baudComboBox->setCurrentIndex(indexBR);

}

// Botón aceptar.

void BTSettingsDialog::on\_buttonBox\_accepted()

{

// Una vez se acepta, los parámetros se guardan en el archivo

de configuración, siendo necesario reiniciar la aplicación

para que se establezca la nueva conexión.

QSettings settings(QString("configs/config.ini"),

QSettings::IniFormat);

if (ui->nameComboBox->count()>0){

settings.setValue("PortName",ui->nameComboBox->

currentText());}

if (ui->baudComboBox->count()>0){

settings.setValue("BaudRate",ui->baudComboBox->

currentText());}

settings.setValue("DataBits",ui->dataBitsComboBox->

currentText());

settings.setValue("FlowControl",ui->flowComboBox->

currentText());

settings.setValue("Parity",ui->parityComboBox->

currentText());

settings.setValue("StopBits",ui->stopBitsComboBox->

currentText());

}

## **A.9. Recursos de la aplicación (application.qrc)**

application.qrc

<RCC>

<qresource prefix="/">

<file>images/Cruceta1.jpg</file>

<file>images/Cruceta2.jpg</file>

<file>images/Cruceta3.jpg</file>

<file>images/Cruceta4.jpg</file>

<file>images/Cruceta5.jpg</file>

<file>images/Cruceta6.jpg</file>

<file>images/Cruceta7.jpg</file>

<file>images/Cruceta8.jpg</file>

<file>images/Cruceta9.jpg</file>

<file>images/Cruceta10.jpg</file>

<file>images/Cruceta11.jpg</file>

<file>images/Cruceta12.jpg</file>

<file>images/Cruceta13.jpg</file>

<file>images/Cruceta14.jpg</file>

<file>images/Cruceta15.jpg</file>

<file>images/Cruceta16.jpg</file>

<file>images/Cruceta17.jpg</file>

<file>images/Cruceta18.jpg</file>

<file>images/Cruceta19.jpg</file>

<file>images/Cruceta20.jpg</file>

<file>images/EOG.ico</file>

</qresource>

</RCC>

## **A.10. Ficheros de configuración (config.ini, mapper.ini)**

config.ini

[General]

PortName=COM7

BaudRate=1200

DataBits=8

StopBits=1

Parity=No

FlowControl=Hardware

Upthreshold=1000

Downthreshold=100

Speed=3

Minblink=10

Maxblink=20

mapper.ini

[General]

No=0

Hardware=1

Software=2

Even=2

Odd=3

Space=4

Mark=5

1=1

2=2

1.5=3

Desconocido=-1

# **B. ANEXO II: PLANIFICACIÓN DEL PROYECTO**

## **Diagramas de Gantt**

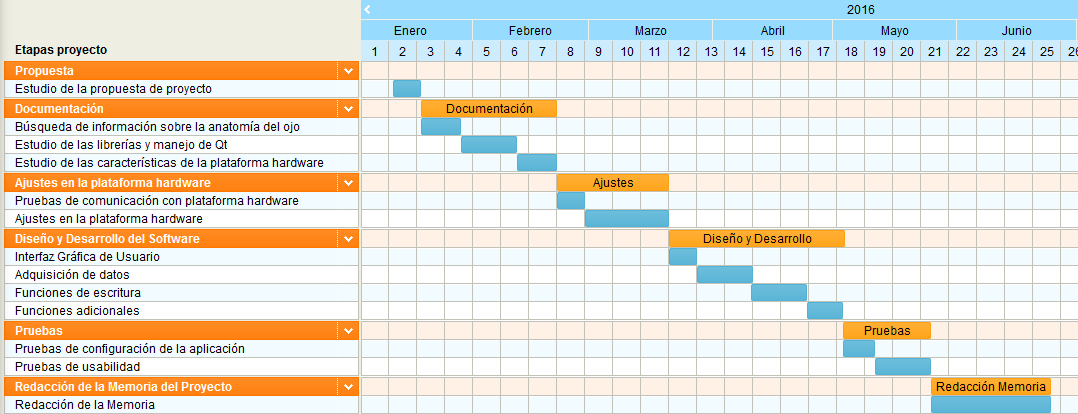


Figura B.1. Diagrama de Gantt

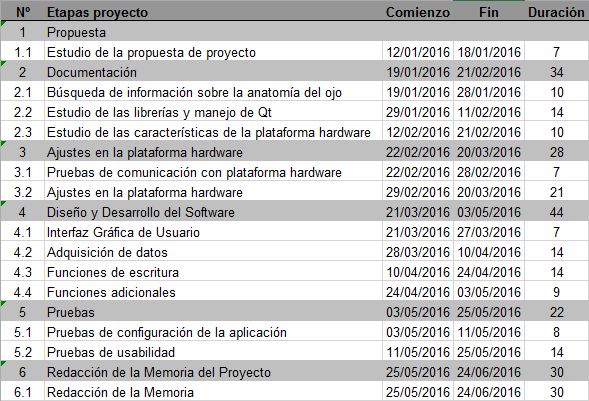


Figura B.2. Detalle de tiempos diagrama de Gantt