ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

Trabajo Fin de Grado

MONITORIZACIÓN DE LA INFORMACIÓN DEL ELECTROENCEFALOGRAMA EN EL PROCESO ANESTÉSICO

Titulación: Grado en Ingeniería Electrónica Industrial y Automática

Alumno: José Manuel González Cava
Tutor: Juan Albino Méndez Pérez

Julio, 2014

Índice de contenidos

Índice de ilustraciones

Índice de tablas

1.Introducción

1.1. Motivación

El presente trabajo fin de grado se enmarca en una aplicación de la automática al campo de la ingeniería biomédica. Dentro de esta disciplina, la ingeniería de control adquiere especial relevancia, estableciéndose como una herramienta puntera en multitud de campos en bioingeniería donde se incorporan técnicas de control en diversos ámbitos: implementación de cirugía robótica; sistemas electrofisiológicos (marcapasos y desfibriladores); soporte vital (ventiladores y corazones artificiales); terapia y cirugía guiada por imagen, entre otros.

Uno de los objetivos principales del control en bioingeniería reside en mantener variables fisiológicas en rangos preestablecidos (niveles de glucemia, marcapasos, respiradores, etc.). Este hecho ha propiciado la incorporación de técnicas de control en la Anestesiología. En concreto, el acto anestésico está encaminado a proteger al paciente de la agresión que supone una intervención quirúrgica y se basa en la aplicación de los principios de Farmacología Clínica para conseguir esta meta. El objetivo final del control persigue desarrollar e implementar aplicaciones que permitan la dosificación del fármaco de forma automática de acuerdo con las necesidades del paciente, obteniendo como resultado unos niveles de eficiencia similar o superior a los obtenidos con infusión manual.

No ajenos a este hecho, desde la Universidad de La Laguna se trabaja en un proyecto de automatización de anestesia desarrollado por el Grupo de Ingeniería Control en colaboración con el equipo de anestesistas del Hospital Universitario de Canarias. El objetivo del proyecto es el desarrollo de estrategias de control para la regulación del estado hipnótico en pacientes sometidos a anestesia intravenosa. El sistema de control diseñado está basado en la regulación del nivel de inconsciencia con rechazo perturbaciones y optimización del proceso. Asimismo, se trabaja para el desarrollo de una herramienta que sirva de interfaz única y permita la monitorización del paciente, la comunicación con el sistema de infusión y la gestión de toda la información generada en el proceso anestésico, librando al anestesista de tareas rutinarias y permitiendo su concentración en otros puntos críticos que pudieran suponer una amenaza para la seguridad del paciente durante la intervención.

En el marco de este trabajo fin de grado, y en vistas de servir de complemento al trabajo ya realizado, se busca obtener, registrar y representar una de las variables fundamentales para el control del proceso anestésico, el electroencefalograma (EEG). De esta forma, se pretende complementar el diseño de la interfaz realizada hasta el momento y facilitar la posibilidad de

procesar los datos con posterioridad si fuera necesario para la adquisición de nueva información relativa al estado del paciente.

1.2. Antecedentes

A través de la historia de la medicina, los especialistas se han inquietado y preocupado por mitigar o, al menos, controlar el dolor físico producido a causa de una intervención quirúrgica, de cuidados intensivos o del ámbito de la emergencia. En todos ellos, la observación y supervisión de las funciones del cuerpo resulta fundamental en vistas de evitar consecuencias negativas para la salud del paciente.

El proceso anestésico queda definido por tres variables básicas: la relajación muscular, la analgesia y la hipnosis. El esquema de trabajo habitual en Anestesiología consiste en administrar un fármaco, observar su efecto en el paciente, ver la interacción fármaco-paciente y adaptar los requerimientos farmacológicos a las características del paciente y a las propiedades del fármaco administrado. Como alternativa, en los últimos años se ha propuesto métodos de control automático para la anestesia basados en la realimentación de alguna o algunas variables fundamentales en el proceso. El empleo de estos controladores en lazo cerrado ofrece ciertas ventajas, entre las que destacan la prevención de eventuales sobredosis o la disminución del costo del proceso anestésico.

Pese a la automatización del procedimiento, es necesario que el especialista pueda supervisar las distintas variables que forman parte del proceso. Dentro del proyecto de la Universidad de la Laguna, se trabaja en el desarrollo de una interfaz gráfica única que permita recoger toda la información, aunando los datos obtenidos de todos los dispositivos en un único medio.

En lo referente al registro de la hipnosis, actualmente se trabaja en el registro del índice biespectral (BIS) a partir del monitor BIS VISTA, así como en el almacenamiento de una serie de parámetros que hacen alusión a la validez de dicho parámetro. Sin embargo, el grupo de investigación está iniciando estudios encaminados a comprender los mecanismos que definen el estado analgésico del paciente. Se supone que el dolor que sufre el paciente debe reflejarse en su actividad cerebral. Por ello, surge la necesidad de procesar toda esta actividad con la intención de buscar patrones o índices que permitan conocer el estado analgésico del paciente para, en un futuro, diseñar un sistema automático para administrar analgésico. En

este sentido se pretende hacer uso del monitor BIS, ya que además del índice BIS permite visualizar el EEG, así como registrar sus valores.

Por todo ello, este trabajo fin de grado pretende dotar de una herramienta al equipo investigador que le permita capturar y procesar las señales del EEG para analizar la actividad cerebral del paciente, paliando las carencias técnicas actuales y mejorando la labor de especialistas en el ámbito de la medicina quirúrgica.

1.3. El proceso anestésico

Aunque no existe una única definición científica, la anestesia clínica se puede concebir como un estado de inconsciencia inducida por fármacos, donde el paciente no percibe ni recuerda los estímulos nocivos. De esta forma, el proceso anestésico adecuado puede definirse como un estado farmacológico donde la relajación muscular del paciente, la analgesia y la hipnosis están garantizadas.

1.3.1. Relajación muscular

La relajación muscular es inducida para facilitar el acceso a los órganos internos y para eliminar movimientos como respuesta a estímulos quirúrgicos. El uso clínico de los relajantes musculares se plantea siempre que se requiere intubación endotraqueal, debido a que los tejidos de esta zona son muy reflexógenos y siempre que la cirugía que se va a realizar requiera de la relajación de los tejidos musculares. El grado de relajación muscular puede ser estimado mediante técnicas que han mostrado cierto grado de éxito.

1.3.2. Hipnosis

La hipnosis es un término que indica pérdida de consciencia y ausencia de recuerdos postoperación de los hechos ocurridos durante la intervención. El grado de hipnosis puede obtenerse a través de diversos métodos, siendo el más eficiente el procesado de las señales del electroencefalograma. La hipnosis se consigue mediante el uso de anestésicos endovenosos o inhalatorios, empleando de manera general la inducción endovenosa dada la mayor confortabilidad para el paciente.

1.3.2.1. Anestesia inhalatoria

La anestesia inhalatoria es la técnica que utiliza como agente principal para el mantenimiento de la anestesia un gas anestésico, que puede ser utilizado incluso como agente

inductor. Los anestésicos inhalatorios son sustancias que se introducen en el cuerpo a través de los pulmones y se distribuyen con la sangre en los diferentes tejidos. El objetivo principal de este tipo de anestésicos es el cerebro. Los más empleados son el óxido nitroso y los anestésicos halogenados, como el isoflurano. Como ventajas de este tipo de sustancias, destaca la regulación de concentración de gas inspirado de forma continua así como la monitorización continua de la concentración en el gas inspirado/espirado.

1.3.2.2. Anestesia total intravenosa

La anestesia total intravenosa se define como una técnica de anestesia general usando una combinación de drogas administrada únicamente por vía intravenosa. Permite la inducción y mantenimiento de la anestesia general con una infusión controlada de hipnóticos, opioides y relajantes musculares. Entre los anestésicos endovenosos más empleados destaca el uso de propofol, un fármaco con acción sedante e hipnótica corta, cuyo efecto presenta una duración breve y una recuperación muy rápida, suave y con confusión postoperatoria mínima.

1.3.3. Analgesia

La analgesia está basada en la eliminación de la sensación del dolor. La Asociación para el Estudio del Dolor (IASP) define el dolor como "una experiencia sensorial y emocional desagradable asociada con daño tisular real o potencial, o descrito en términos de tales daños". Esta definición pone de manifiesto el carácter subjetivo de la sensación de dolor, lo que dificulta la búsqueda de métodos efectivos que permitan cuantificarlo de forma absoluta.

En los procesos quirúrgicos con anestesia se utilizan analgésicos de gran potencia, como son los opiáceos mayores. Uno de los más empleados tanto en procesos cortos como en procesos largos con perfusión continua es el remifentanilo.

1.3.4. Fases del proceso anestésico

1.3.4.1. Inducción

Es el primer paso en el proceso anestésico y concibe la transición del estado de vigilia al de hipnosis o sueño. En general, en esta primera fase se debe cumplir con los objetivos de hipnosis, analgesia y relajación muscular.

1.3.4.2. Mantenimiento

La situación anestésica conseguida tras la inducción debe mantenerse tanto tiempo como dure la situación que lo ha requerido. Esto se conseguirá con los mismos fármacos empleados en la fase anterior suministrándose a través de sistemas de perfusión o bolos de fármacos según se requiera.

1.3.4.3. Recuperación

Al cesar la administración del hipnótico se producirá una vuelta al estado vigil. Es importante que en este momento el paciente tenga una buena analgesia para evitar la aparición repentina del dolor. Asimismo, al retirar los fármacos hipnóticos se debe asegurar que no exista relajación muscular para evitar situaciones angustiosas para el paciente.

1.4. Objetivos

El objetivo general de este trabajo fin de grado se centra en complementar la interfaz gráfica desarrollada en el marco del proyecto de automatización de anestesia desarrollado en la ULL incluyendo nueva información a través de una variable hasta ahora no registrada, el electroencefalograma. Todo ello permitirá dotar al experto de una interfaz única para la monitorización de la información del paciente y la gestión de la información generada en el proceso anestésico. Además, el registro del EEG pretende usarse para su posterior análisis en la búsqueda de un índice que se correlacione con el dolor que sufre el paciente. Se parte de la hipótesis de que los estímulos dolorosos provocan cambios notables en la actividad cerebral.

Por su parte, los objetivos específicos a desarrollar en este trabajo son:

- Establecer una comunicación a través de un puerto serial entre el ordenador y el monitor BIS, generando y extrayendo la información en el formato y protocolo adecuados.
- Registro de los valores del EEG durante el proceso anestésico, de forma que los datos puedan ser analizados con posterioridad
- Representación del EEG en función del tiempo, en vistas de simular la información disponible en el monitor BIS.
- Registro de las variables procesadas que suministra el monitor BIS. En particular se registrarán las variables: índice biespectral (BIS), electromiograma (EMG), tasa de supresión e índice de calidad de la señal.

- Integración del código elaborado en una interfaz de usuario.
- Comprobación y validación de los resultados obtenidos al realizar pruebas reales.
- Incorporación de la nueva información en la interfaz principal y comprobación del conjunto.

1.5. Estructura del documento

Este documento se estructura en un total de seis capítulos en los que se abordarán los distintos aspectos contemplados para la elaboración del presente trabajo fin de grado.

En el capítulo 2 se exponen los principales métodos para el control y monitorización de la anestesia. En concreto, se hará especial hincapié en los monitores de profundidad anestésica basados en los registros de la actividad electroencefalográfica. Asimismo, se hará una revisión de las principales características del monitor BIS VISTA® empleado en la práctica.

El capítulo 3 recoge las especificaciones técnicas del puerto serial que permite la comunicación entre el monitor y el ordenador para la obtención de datos. Se describen, además, las características físicas de la interfaz empleada y los formatos de los datos recibidos y transmitidos a través del puerto serial.

Una vez explicados los principales conceptos relativos a la comunicación entre monitor y ordenador, en el capítulo 4 se procede al desarrollo de la interfaz para la captura del EEG y el resto de variables procesadas. En él se explicará el código empleado para el procesamiento de los datos, así como su integración en una interfaz de usuario.

En el capítulo 5 se realizará una verificación de los resultados obtenidos de la interfaz comparándolos con los registrados en el monitor BIS VISTA. Finalmente, el capítulo 6 recogerá las principales conclusiones derivadas de este trabajo fin de grado, así como las posibles líneas abiertas de investigación que se pudieran desarrollar en un futuro.

2.Monitorización y control de anestesia: el monitor BIS

2.1. Métodos de detección del estado de hipnosis

En las primeras épocas de la práctica anestésica, la profundidad de la hipnosis se evaluaba por parámetros clínicos, entre los cuales el movimiento era uno de los más importantes. La introducción de los relajantes musculares aumentó la dificultad de la valoración de la profundidad anestésica. Otros datos que pueden indicar el estado de hipnosis del paciente son los derivados de la monitorización habitual. El creciente interés por disminuir la incidencia de despertar intraoperatorio ha propiciado la aparición de nuevos aparatos que, a través del análisis de la actividad eléctrica cerebral, pueden volcar mejores resultados en lo referente al estado de hipnosis de los pacientes.

Con todo ello, la observación clínica, la monitorización convencional y los monitores de función cerebral pueden mostrar patrones que se corresponden con variaciones en la profundidad anestésica, dando información acerca del estado hipnótico del paciente.

2.1.1. Evaluación clínica

Entre los parámetros clínicos usados para determinar el nivel de consciencia intraoperatoria se encuentran la presencia de movimientos, la respuesta a órdenes, la apertura de ojos, el reflejo corneal, el tamaño y reactividad pupilar, la sudoración y el lagrimeo. Estas medidas pueden ayudar a evaluar la profundidad anestésica, aunque no hay estudios que demuestren hasta qué grado son útiles para el control del estado hipnótico. Su valor puede verse afectado por distintos fármacos.

2.1.2. Monitorización tradicional

Incluye los monitores habituales (electrocardiograma, tensión arterial, frecuencia cardiaca, pulsioximetría, volumen tidal, capnografía) y el análisis de aspiración de gases anestésicos. Los datos derivados de estos monitores pueden ayudar a determinar la profundidad anestésica, informando de la aparición de cambios hemodinámicos o respiratorios. Sin embargo, durante la anestesia, pueden producirse variaciones de parámetros influenciadas por los fármacos de uso habitual en el periodo intraoperatorio, y no por causas relacionadas con el nivel de hipnosis. Este hecho resta validez a los resultados obtenidos mediante este método.

2.1.3. Monitores de actividad electroencefalográfica

Son aparatos que recogen y procesan la actividad eléctrica cerebral y convierten esta señal eléctrica, a través de algoritmos matemáticos, en un índice reconocible (habitualmente una escala numérica entre 0 y 100). La señal captada puede ser la actividad eléctrica cortical espontánea (EEG) o la actividad evocada por estímulos (potenciales evocados). En algunos casos, se opta por recoger también la actividad electromiográfica de los músculos de la calota. Un monitor fiable de anestesia debería mostrar una buena correlación entre el valor medido y la respuesta fisiológica durante la intervención, independientemente del anestésico administrado, y debería tener poca variabilidad interpersonal.

2.2. Fundamentos de la monitorización electroencefalográfica en anestesia

El electroencefalograma es el registro de la actividad eléctrica de las neuronas piramidales del córtex. Esta actividad eléctrica atraviesa los tejidos hasta la piel y es recogida por los electrodos. Tras un proceso de filtrado para eliminar los artefactos y de amplificación, la señal se representa de forma gráfica en forma de ondas.

Las ondas del registro se caracterizan por su frecuencia (número de ondas por segundo [Hz]), por su amplitud (altura de onda medida en microvoltios [μV]) y por su fase (desajuste de inicio de cada tren de ondas respecto al punto de ángulo cero). Como norma general, las ondas se clasifican atendiendo a su frecuencia:

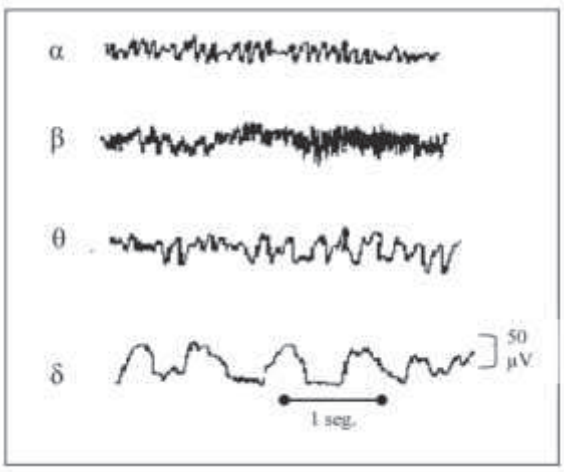

Figura 1. Ondas del EEG. Fuente: Aspect Medical System.

- Ondas β: 13-45 Hz: ondas de pequeño voltaje que aparecen con el paciente en estado vigil, con los ojos abiertos.
- Ondas α: 8-13 Hz: aparecen en pacientes despiertos con los ojos cerrados.
- Ondas Θ: 4-7 Hz: se presentan con el paciente somnoliento o sedado.
- Ondas δ: 0,5-4Hz: sueño profundo (fisiológico o inducido por fármacos).

De la figura 1 se aprecia que, desde el punto de vista encefalográfico, el estado vigil se caracteriza por un registro en el que predominan ondas rápidas (de alta frecuencia) y de pequeño voltaje (ondas α y β). El paso a un estado de hipnosis profunda va transformando el EEG en ondas cada vez más lentas y de mayor amplitud.

Para la obtención de información útil a partir del EEG es necesario filtrarlo, computarizarlo y simplificarlo mediante el uso de algoritmos matemáticos, basados en el análisis en el dominio temporal y de la frecuencia.

2.2.1. Análisis en el dominio temporal

Valora los cambios que se van sucediendo de forma cronológica en el registro del electroencefalograma. Uno de los datos derivados de este análisis es la tasa de supresión, parámetro indicador de la relación existente entre los periodos con presencia de señal electroencefalográfica de gran amplitud y los periodos en que aparece silencio eléctrico (EEG plano).

2.2.2. Análisis en el dominio de la frecuencia

Descompone los trenes de ondas en sus componentes más simples. Este tipo de análisis incluye:

a) Análisis espectral: Consiste en analizar pequeños fragmentos del EEG y descomponerlos en trenes de ondas con frecuencia y amplitud determinados. Basándose en estos datos se puede determinar:
   i. Potencia espectral: Es el cuadrado de la amplitud de cada una de las frecuencias que componen el fragmento del EEG. Con este cálculo, se convierte el trazado electroencefalográfico en un histograma de amplitudes en función de la frecuencia, permitiendo una computación más eficiente de los datos.

ii. Potencia delta relativa: Describe el porcentaje de la potencia del EEG en el rango de las frecuencias delta en relación con la potencia espectral de todas las frecuencias.
iii. Límite espectral 95% (LE95%): Es el valor de frecuencia por debajo de la cual está contenido el 95% de total de la potencia del espectro.

b) Análisis biespectral: Analiza el grado de coherencia entre las fases de las ondas, incrementando la información sobre los cambios que aparecen en el EEG.

2.3. Monitores de profundidad anestésica

Los monitores de profundidad anestésica son aparatos que recogen la actividad eléctrica cerebral espontánea o evocada por estímulos. Tras amplificar la señal, eliminar interferencias y convertir los datos analógicos en digitales, se aplican diferentes algoritmos matemáticos a los datos obtenidos para generar un índice simple. Éste índice representa la progresión de los estados clínicos de consciencia, pasando por la sedación y grados crecientes de profundidad anestésica.

Actualmente se dispone de una amplia gama de monitores de uso clínico:

- BIS®: Este dispositivo convierte un canal único del EEG frontal en un dígito, el índice biespectral, con valores entre 0 y 100. El algoritmo matemático para su obtención basado en el análisis en el dominio temporal y de frecuencia no ha sido publicado en su totalidad.
- Entropía: El módulo de entropía describe la irregularidad, complejidad o predecibilidad de la señal EEG. El algoritmo se encarga de calcular la entropía de estado y la entropía de respuesta, generando finalmente un índice que oscila entre 0 y 100.
- Narcotrend: Este monitor resulta de un sistema desarrollado para la clasificación visual de patrones electroencefalográficos asociados a diferentes fases del sueño. Del resultado de la clasificación se obtiene un índice numérico entre 0 y 100 para proporcionar una escala cuantitativa similar al BIS.
- PSA: El monitor PSA genera una escala adimensional de 0 a 100, denominada PSI, un índice empírico derivado del análisis de un EEG de cuatro canales. El algoritmo que calcula el PSI está basado en un análisis multivariante de variables electroencefalográficas derivadas de tres bases de datos.

- SNAP Index: Derivado de un canal electroencefalográfico, este monitor se basa en el análisis espectral de la actividad del EEG en los rangos de frecuencia 0-18 Hz y 80-42 Hz y en la tasa de supresión. El rango del índice SNAP oscila entre 0 y 100.
- Cerebral State Index: Dispositivo que centra su análisis en un canal del EEG, dando como resultado un índice comprendido entre 0 y 100. Además, indica la tasa de supresión y la actividad miográfica.
- Potenciales evocados auditivos: A diferencia de todos los monitores anteriores, no representa la actividad electroencefalográfica, sino que estudia las respuestas eléctricas cerebrales inducidas por estímulos sonoros que se transmiten a través de auriculares. Las técnicas más avanzadas combinan las características del EEG y potenciales evocados auditivos para producir el AAI, un índice con un rango de 0 a 100 indicador de la profundidad hipnótica del paciente.

Pese a los diferentes métodos de detección del estado de hipnosis, las recomendaciones de la American Society of Anesthesiologists aconsejan la utilización de monitores de profundidad anestésica, en vistas a disminuir las incidencias de despertar intraoperatorio. Dentro del amplio abanico de posibilidades, el único monitor que ha que demostrado reducir la incidencia de despertar intraoperatorio en un 80% es el BIS. Este resultado va acompañado de un menor consumo de anestésicos, reducción del malestar postoperatorio y disminución de tiempos de extubación, despertar y recuperación.

Todo ello convierte al BIS en el monitor de uso más extendido en el ámbito de anestesia quirúrgica. Este hecho ha propiciado su utilización dentro del marco del proyecto de investigación realizado por la Universidad de La Laguna como indicador del nivel de hipnosis del paciente, sirviendo de base el índice espectral para la realimentación del sistema de control.

2.4. El monitor BIS VISTA®

Desde su introducción en 1996, el dispositivo BIS ha experimentado una evolución notable, tanto en tecnología destinada a la disminución de interferencias, como en las posibilidades de personalizar la pantalla. En concreto, el modelo de monitor empleado en este trabajo fin de grado es el monitor BIS VISTA®. A continuación se procede a realizar un análisis de su funcionamiento así como del aspecto que presenta.

2.4.1. Principio de funcionamiento

El índice biespectral del EEG es un sistema de monitorización no invasiva que permite una cuantificación del nivel de hipnosis del paciente. El sistema BIS emplea un sensor colocado en la frente para transmitir las señales del EEG desde el paciente hasta un convertidor de señal digital, el cual digitaliza la señal y la envía a un monitor para su procesamiento y análisis. En el monitor, los artefactos son identificados y rechazados y un complejo análisis informático procesa la información del EEG de forma continua, calculando un número entre 0 y 100, donde 0 indica la ausencia de actividad eléctrica cerebral (EEG plano) y 100 indica que el paciente está despierto. Por tanto, el BIS proporciona una medición directa del nivel de consciencia del paciente y de su respuesta a la sedación.

Dentro del rango de valores, cifras entre 40 y 60 se consideran adecuadas para la anestesia quirúrgica y pueden evitar episodios de despertar intraoperatorio y recuerdo en pacientes. Valores por debajo de 40 implican una sobresedación, no aportando mejorías al proceso. En la figura 2 se muestra una guía de intervalos del BIS.

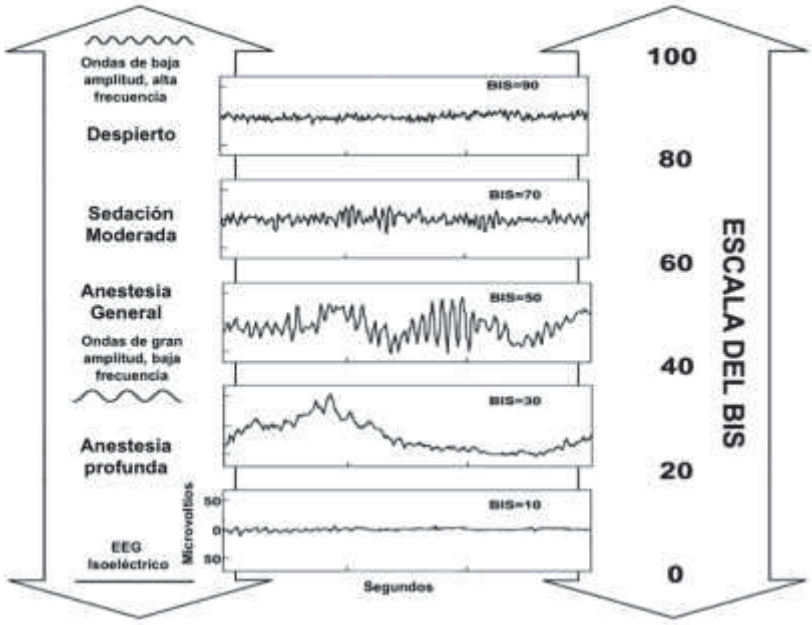

Figura 2. Rango del índice BIS. Fuente: Aspect Medical System.

El monitor BIS, además del valor numérico y de su tendencia, aporta información sobre la calidad de detección de la señal del EEG (SQI), de la tasa de supresión (SR) y aporta información visual de la morfología de las ondas del EEG y de la posible interferencia del electromiograma (EMG) en los valores mostrados. El sistema realiza autocomprobaciones

para garantizar que el monitor y sus componentes funcionan correctamente y que los niveles de impedancia de los sensores del paciente se encuentran en unos límites aceptables.

2.4.2. Componentes del sistema BIS

La figura 3 recoge los elementos que componen el sistema BIS VISTA®.

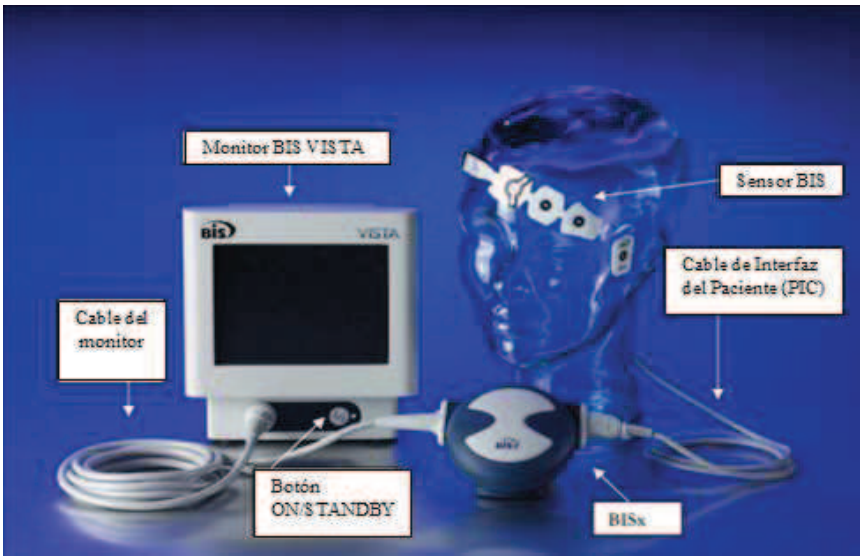

Figura 3. Componentes del sistema BIS. Fuente: Aspect Medical System.

- Sensor BIS: El sensor BIS es un sistema complejo formado por cuatro electrodos numerados, compuestos de plata/cloruro de plata, que permite captar la señal electroencefalográfica del paciente. El sensor es de un único uso y debe reemplazarse antes de cada nueva monitorización.
- Cable de Interfaz del Paciente(PIC): La señal electroencefalográfica sin procesar se transmite a través de este cable desde el sensor hasta el módulo BISx
- BISx: El BISx recibe, filtra y procesa las señales del EEG del paciente. Se coloca cerca de la cabeza del paciente, lugar en el que la señal del EEG está expuesta a menos interferencias de otros equipos médicos.
- Cable del monitor: El cable largo y flexible de interfaz del monitor transmite la señal ya procesada desde el módulo BISx hasta el monitor.
- Monitor BIS VISTA: El monitor está conformado por un panel frontal compuesto por una pantalla táctil desde donde se visualizan y ajustan los parámetros medidos, el puerto de conexión del BISx y el botón de ON/STANDBY. En la parte trasera, se incluyen dos puertos USB, un puerto RS-232, el botón de reinicio, la batería y el receptáculo del cable eléctrico.

2.4.3. Visualización de datos capturados

Una vez la comprobación del sensor ha resultado satisfactoria, se inicia el sistema y aparece la información correspondiente en la pantalla principal.

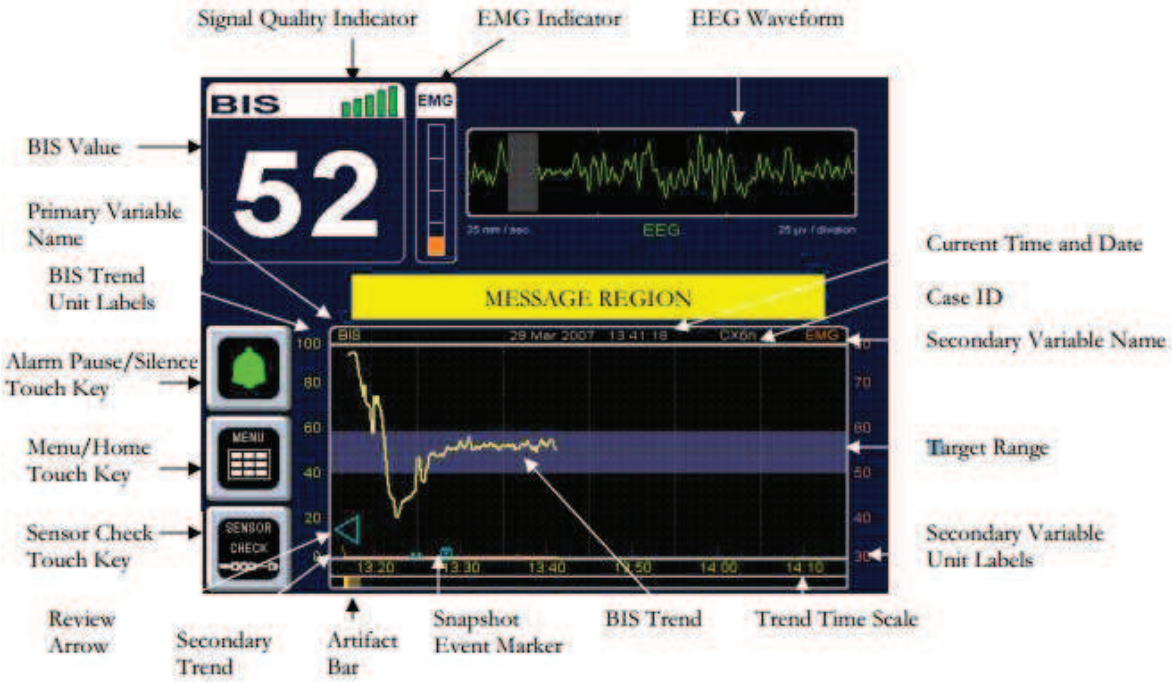

Figura 4. Pantalla del monitor BIS VISTA. Fuente: Aspect Medical System.

La figura 4 muestra la información recogida en la pantalla del dispositivo. Los parámetros principales a tener en cuenta son:

- Valor del índice biespectral (BIS): Este índice se muestra en la parte superior izquierda de la pantalla y se actualiza siempre que la calidad de la señal sea adecuada.
- Indicador de calidad de señal (SQI): Parámetro indicador de la calidad de la señal registrada, calculada a partir de la impedancia, artefactos y otras variables influyentes en el proceso. La gráfica de barras que indica el SQI se muestra a la derecha de la etiqueta BIS. La calidad de la señal es óptima cuando las cinco barras se muestran en verde. En el caso de que este parámetro no cumpla con los mínimos, el BIS no se representará en la pantalla.
- Indicador de EMG: La barra vertical a la derecha del índice espectral representa la potencia, en decibelios, en la frecuencia de 70-110 Hz. Este rango de frecuencia recoge información acerca de la actividad muscular. Las condiciones óptimas para el cálculo del BIS son aquellas en las que esta barra está vacía.

- Representación de EEG: En la parte superior derecha de la pantalla se representa el EEG filtrado, con un barrido de 25mm por segundo y una escala de 25 μV (1 canal) o 50 μV (2 canales).
- Región de mensajes: En este espacio pueden mostrarse mensajes sobre estado y error. El color en que aparece el mensaje indica su prioridad: naranja (alta prioridad), amarillo (prioridad media), azul claro (baja prioridad) o azul oscuro (mensaje informativo)
- Gráfica de tendencias: Gráfica que ocupa la parte central de la pantalla y representa los valores del BIS a lo largo del tiempo, así como información adicional acerca de la hora o un rango objetivo prefijado para la variable. Los valores se indican en el eje izquierdo. Además, esta gráfica permite la visualización de variables secundarias cuyos valores estarán asociados al eje derecho del gráfico.

En general, se pueden visualizar nuevas variables, así como modificar el aspecto general del monitor desde las opciones habilitadas en el menú principal del dispositivo adaptándolo a las necesidades.

3.Captura del EEG

3.1. Interfaz de comunicación

El monitor BIS VISTA® está dotado en su parte trasera de una serie de puertos que permiten obtener la información capturada por el dispositivo. En concreto, el monitor cuenta con un puerto serial RS-232, a través del cual se puede extraer la información relativa tanto al EEG como a las variables procesadas en tiempo real para su almacenamiento, representación o impresión. Este hecho, propicia el uso del puerto serial para cumplir con los objetivos propuestos.

3.1.1. Puerto serial RS-232

El puerto serial RS-232 se trata de un puerto asíncrono dotado de un conector DB-9 hembra. Sus pines vienen definidos según la figura 5.

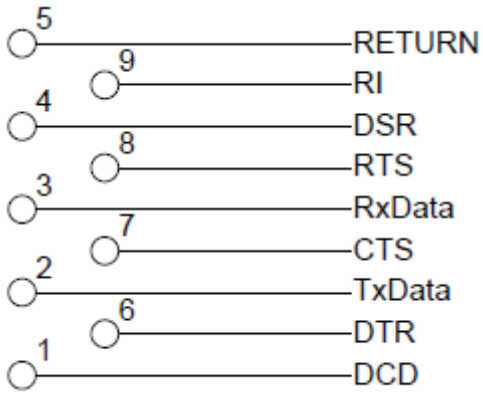

Figura 5. Especificación del conector RS-232 DB-9 hembra.

La configuración del puerto serie tanto para la transmisión como para la recepción viene descrita en la tabla 1.

	Protocolo ASCII	**Protocolo Binario**
Velocidad de transmisión	9600	57600
Bits de datos	8	8
Bits de parada	1	1
Paridad	Sin paridad	Sin paridad
Control de flujo	No	No

Tabla 1. Configuración del puerto serie.

3.1.2. Adaptador FTDI

Con el paso del tiempo, el puerto serie RS-232 ha entrado en desuso para muchas aplicaciones informáticas. Este hecho lleva a que muchos ordenadores no incluyan un puerto serie de comunicación. Para prevenir posibles problemas en posteriores implantaciones de nuestra aplicación, se opta por emular un puerto serie RS-232 a partir de un puerto USB.

Los adaptadores USB a serie RS-232 del fabricante FTDI permiten crear un puerto serie virtual a través del puerto USB mediante un software. Como ventaja adicional de este tipo de dispositivos, cuenta con un API abierta que permite realizar la comunicación con los dispositivos directamente, prescindiendo de las interfaces de puerto serie de MATLAB. Esto permite una comunicación más sencilla, robusta y eficiente.

Otra de las ventajas incluidas por el FTDI es la detección de problemas y estados inconscientes del controlador de forma temprana, permitiendo enviar comandos que provoquen la recuperación del chip FTDI en caso de fallo. La implementación de las funciones del FTDI se analizará en el capítulo 4.

3.1.3. Diagrama de conexiones

Este apartado se centra en especificar las conexiones físicas realizadas entre los distintos dispositivos. En la figura 6 se muestra la conexión entre el monitor BIS VISTA® y el ordenador.

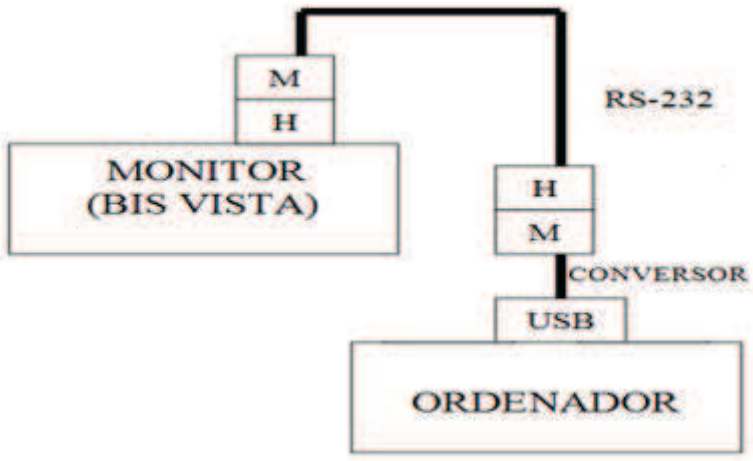

Figura 6. Conexión del monitor con el ordenador.

Por su parte, la figura 7 especifica la conexión de pines entre el monitor y el conversor RS-232/USB.

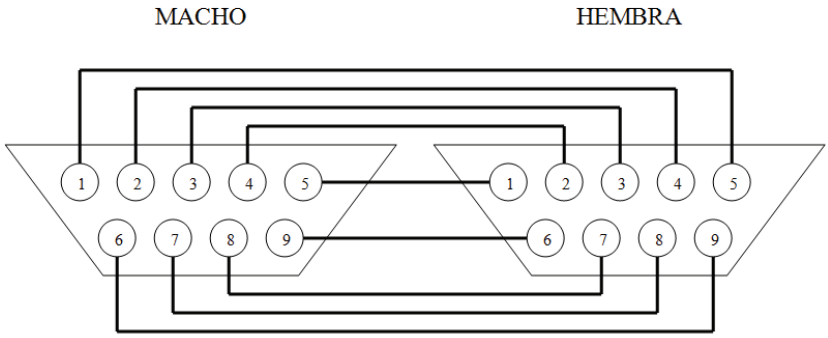

Figura 7. Conexión RS-232 del BIS al conversor macho RS-232/USB.

3.2. Protocolo de comunicación: El protocolo Binario

El puerto serial del monitor puede operar con dos tipos de protocolos: ASCII o Binario. El protocolo ASCII es un protocolo sencillo que permite la adquisición de las variables procesadas. Por su parte, el protocolo Binario permite, además, obtener la información relativa al EEG. Dado que nuestro objetivo principal es la obtención del EEG, el protocolo de comunicación a emplear es el protocolo Binario.

En los siguientes apartados se analizarán las características principales del protocolo Binario así como la estructura que adquieren los paquetes de comunicación.

3.2.1. Características del protocolo Binario

El protocolo Binario es un protocolo basado en el intercambio de información a través de paquetes. Cuenta con tres tipos básicos de paquetes:

- Paquetes de datos: Incluye la información a transmitir
- Paquetes ACK (acknowledgement): Confirma la validez del paquete de datos recibido.
- Paquetes NACK (negative acknowledgement): Informa de un error en la recepción del paquete de datos.

En el caso de que se reciba un paquete NACK, el último paquete de datos debe ser reenviado hasta que se reciba un mensaje de confirmación. El elevado número de datos transmitidos por el monitor BIS VISTA® impide el proceso de reenvío, por ello, recibir un

paquete NACK o ACK es equivalente, procediéndose a transmitir el siguiente paquete de datos. Como consecuencia, no será necesario enviar paquetes ACK o NACK al monitor.

Otra de las características que se debe tener en cuenta, es que el monitor de BIS es un dispositivo Little-endian. Ello implica que los valores de más de un byte transmitirán primero el byte menos significativo y, por último, el byte más significativo. Para ajustar la configuración al protocolo binario, se seleccionará desde el Menú Principal del monitor la opción "VISTA Binary" dentro del apartado de Mantenimiento.

3.2.2. Estructura de los paquetes de datos

Los paquetes binarios presentan un formato multicapa, de forma que cada mensaje está conformado por tres capas. A continuación, se procede a estudiar y analizar la estructura de cada mensaje.

3.2.2.1. Capa 1: Capa física

La capa 1, ubicada en la parte baja de la jerarquía, contiene la información relativa a la transmisión de datos a través de la conexión serial. El formato de la capa 1 responde a la estructura recogida en la tabla 2.

Campo de paquete	Número de bits	Descripción
ID. de comienzo de paquete (SPI)	16	Cada paquete comienza con la secuencia 0xabba
ID. de la secuencia de paquete (PSID)	16	Rango:0x0000-0xFFFF
Número de Bytes de Datos Opcionales (OD)	16	Número de Bytes en las capas 2 y 3. Rango: 0-0x800
Directiva de capa 1 (LD)	16	Existen 3 tipos diferentes en función del tipo de paquete: - 1: Paquete de datos - 2: Paquete ACK - 3: Paquete NACK
Datos Opcionales	Variable	
Checksum (CS)	16	Suma de los bytes de: - Identificación de la secuencia de paquete - Número de Bytes de Datos Opcionales. - Directiva de capa 1 - Datos Opcionales

Tabla 2. Formato de la capa física.

El número de secuencia se inicializa a cero. Cada paquete de datos enviado a través de la interfaz aumentará en uno el número de secuencia, permitiendo llevar una cuenta del número de paquetes de datos transmitidos. En el caso de los Datos Opcionales para paquetes ACK y NACK, el número de bits está fijado en cero.

3.2.2.2. Capa 2: Capa de enrutamiento

La capa 2 proporciona información para el enrutamiento del mensaje en un sistema con múltiples procesadores o procesos. La información de la capa 2 se recoge en la tabla 3.

Campo de paquete	Número de bits	Descripción
ID de enrutamiento (RID)	32	Es empleado para identificar el destino al cual el comando o el resultado del comando va a ser dirigido
Datos de Aplicación	Variable	

Tabla 3. Formato de la capa de enrutamiento.

3.2.2.3. Capa 3: Capa de aplicación

La capa de aplicación, ubicada en la parte alta de la jerarquía, está directamente relacionada con el intercambio de datos específicos. Esta capa se divide en los campos contemplados en la tabla 4.

Campo de paquete	Número de bits	Descripción
ID de mensaje (MID)	32	Indica el tipo de datos de aplicación en el paquete
Número de secuencia (SN)	16	Indica el número de mensajes enviados correspondiente a cada identificador de mensaje. Rango: 0-0xFFFF
Longitud (L)	16	Número de bytes dependiente de los datos del mensaje
Datos dependientes del mensaje (MDD)	Variable	

Tabla 4. Formato de la capa de aplicación.

Inicialmente, el número de secuencia se fija en cero. Cada identificador de mensaje tiene asociado un número de secuencia, de forma que, al enviar un mensaje a través de la interfaz, el número de secuencia correspondiente al MID del mensaje aumenta en uno.

El esquema final que adquiere un paquete completo de datos aparece representado en la figura 8.

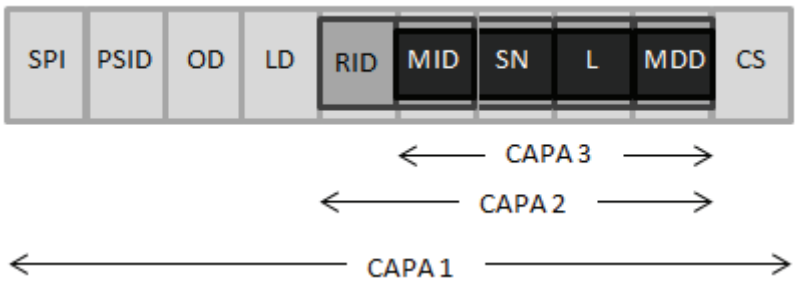

Figura 8. Estructura de paquete de datos.

3.2.3. Mensajes del ordenador al monitor

Todos los comandos enviados desde el ordenador al monitor tienen un único identificador de mensaje. El campo MDD de la capa 3 tiene parámetros específicos del comando, así como una longitud de datos asociada. En el caso de que el mensaje enviado al monitor sea erróneo, el monitor devolverá un paquete NACK.

En la tabla 5 se muestra la lista con los principales comandos válidos que pueden ser enviados al monitor BIS VISTA®.

Comandos	ID	Descripción
Envío EEG bruto	111	Solicita EEG bruto. Se transmiten 8 paquetes por segundo, con dos canales intercalados de datos de EEG.
Parar EEG bruto	112	Para el envío de EEG bruto
Envío de variables procesadas	115	Solicita variables procesadas
Parar variables procesadas	116	Para el envío de variables procesadas
Activar mensaje de impedancia	1000	Solicita el envío de datos de impedancia registrados por el sensor
Desactivar mensaje de impedancia	1001	Para el envío de datos de impedancia
Activar mensaje de error	1002	Solicita el envío de mensajes de error
Desactivar mensaje de error	1003	Para el envío de mensajes de error
Envío de información relativa al monitor	1004	Solicita el envío de información relativa al monitor
Envío de etiquetas de variables procesadas	133	Solicita el envío de las etiquetas de las variables procesadas

Tabla 5. Mensajes del ordenador al monitor.

En los siguientes apartados, se procede a analizar en profundidad las características de cada comando para la formación de los mensajes requeridos de acuerdo a nuestras necesidades. Asimismo, se ilustrará cada caso con un ejemplo.

3.2.3.1. Envío de EEG bruto

El paquete de datos asociado a la petición de envío de EEG bruto deberá contener los parámetros recogidos en la tabla 6.

Campo	Posibles valores
RID	4
MID	111
Longitud	2
MDD	128

Tabla 6. Valores de mensaje para el envío de EEG bruto.

De acuerdo al formato del paquete, y adaptando la información anterior, los campos del mensaje serán los siguientes:

Campo de paquete	Valor	Valor en formato de paquete (Bytes)
SPI	0xabba	186, 171
PSID	0	0, 0
OD	14	14,0
LD	1	1,0
RID	4	4,0,0,0
MID	111	111, 0,0,0
SN	0	0,0
Longitud	2	2,0
MDD	128	128,0
Checksum	259	4, 1

Tabla 7. Valores de los campos de paquete para solicitud del EEG bruto.

Los valores de PSID y SN los fijaremos a cero para la creación de mensajes hacia el monitor dado que, a priori, no serán datos que vayamos a tener en cuenta.

El aspecto final del mensaje es el que recoge la figura 9.

186	171	0	0	14	0	1	0	4	0	0	0	111	0	0	0	0	0	2	0	128	0	4	1

Figura 9. Paquete de mensaje de petición del EEG bruto.

3.2.3.2. Parada de envío de EEG bruto

Una vez se realiza la solicitud de envío de EEG bruto, se comienza la recepción de paquetes de forma ininterrumpida cada 125ms. Para parar el proceso es necesario generar un mensaje de parada de acuerdo a los datos siguientes.

Campo	Posibles valores
RID	4
MID	112
Longitud	0

Tabla 8. Valores de mensaje para la parada de EEG bruto.

El resultado de los campos del paquete se recoge en la tabla 9.

Campo de paquete	Valor	Valor en formato de paquete (Bytes)
SPI	0xabba	186, 171
PSID	0	0, 0
OD	12	12,0
LD	1	1,0
RID	4	4,0,0,0
MID	112	112, 0,0,0
SN	0	0,0
Longitud	0	0,0
Checksum	129	129,0

Tabla 9. Valores de los campos de paquete para la parada de EEG bruto.

El mensaje final tomará la siguiente forma:

186	171	0	0	12	0	1	0	4	0	0	0	112	0	0	0	0	0	0	0	129	0

Figura 10. Paquete de mensaje de parada del EEG bruto.

3.2.3.3. Envío de variables procesadas

En un primer momento, las variables procesadas necesarias tanto para el proceso de control como para la representación en la interfaz original se obtenían a través del protocolo ASCII. Con miras a poder incorporar la representación del EEG, es necesario proporcionar esta información en protocolo Binario. El mensaje deberá contener la siguiente información:

Campo	Posibles valores
RID	4
MID	115
Longitud	1
MDD	0 - Envío de variables procesadas sin espectro 1 – Envío de variables procesadas con espectro

Tabla 10. Valores de mensaje para el envío de variables procesadas.

En nuestro caso, dada la versión de protocolo con la que cuenta el monitor, se enviará siempre la información relativa al espectro, independientemente del valor del MDD. Los campos del mensaje serán, por tanto, los recogidos en la tabla 11.

Campo de paquete	Valor	Valor en formato de paquete (Bytes)
SPI	0xabba	186, 171
PSID	0	0, 0
OD	13	13,0
LD	1	1,0
RID	4	4,0,0,0
MID	115	115, 0,0,0
SN	0	0,0
Longitud	1	1,0
MDD	0	0
Checksum	134	134,0

Tabla 11. Valores de los campos de paquete para el envío de las variables procesadas.

El mensaje resultante será el siguiente:

186	171	0	0	13	0	1	0	4	0	0	0	115	0	0	0	0	0	1	0	0	134	0

Figura 11. Paquete de mensaje de petición de variables procesadas.

3.2.3.4. Parada de envío de variables procesadas

Como ocurría con el EEG, los paquetes de datos de las variables procesadas se recibirán de forma ininterrumpida, en este caso, un paquete por segundo, hasta que se solicite su parada. El mensaje se formará con los siguientes campos:

Campo	Posibles valores
RID	4
MID	116
Longitud	0

Tabla 12. Valores de mensaje para la parada de variables procesadas.

La composición total de los campos será la descrita en la tabla 13.

Campo de paquete	Valor	Valor en formato de paquete (Bytes)
SPI	0xabba	186, 171
PSID	0	0, 0
OD	12	12,0
LD	1	1,0
RID	4	4,0,0,0
MID	112	112, 0,0,0
SN	0	0,0
Longitud	0	0,0
Checksum	129	129,0

Tabla 13. Valores de los campos de paquete para la parada de EEG bruto.

El mensaje a enviar adquiere la siguiente forma

186	171	0	0	12	0	1	0	4	0	0	0	116	0	0	0	0	0	0	0	133	0

Figura 12. Paquete de mensaje de parada de variables procesadas.

3.2.4. Mensajes del monitor al ordenador

De forma análoga a lo descrito en el apartado anterior, todos los mensajes enviados desde el monitor al ordenador tienen un único identificador. En la tabla 14 se muestran las principales respuestas generadas por el monitor, así como una breve descripción.

Respuestas	ID	Descripción
Mensaje EEG bruto	50	EEG bruto, enviando ocho paquetes por segundo.
Mensaje variables procesadas	52	Variables procesadas.
Mensaje variables procesadas y espectro	53	Variables procesadas y el espectro
Mensaje impedancias	1100	Valores de impedancia del sensor
Mensaje de error	1101	Especifica los mensajes de error
Mensaje de información relativa al monitor	1102	Información sobre el Software
Mensaje de etiquetas de var. procesadas	63	Etiquetas de las variables procesadas

Tabla 14. Mensajes del monitor al ordenador.

3.2.4.1. Mensaje de EEG bruto

Los datos del EEG bruto vienen codificados en dos bytes en complemento a dos con signo. Al contar con un sensor de dos canales, los datos se transmitirán intercalados (el primer dato corresponderá al primer valor de EEG del canal uno, el segundo al primer valor de EEG del canal dos, y así sucesivamente). Los valores de cada campo se recogen en la tabla 15.

Campo	Valores
MID	50
Longitud	4+((Velocidad/8)*número de canales*2) = 68 - Velocidad : 128 muestras/s - Número de canales: 2
MDD - 2 Bytes: Nº de canales - 2 Bytes: Velocidad - Resto de Bytes	 - 2 canales - 128 muestras/s - Datos EEG en 2 bytes en complemento a 2

Tabla 15. Valores de los campos de mensaje de EEG bruto.

La estructura del paquete será la siguiente:

Campo de paquete	Valor	Valor en formato de paquete (Bytes)
SPI	0xabba	186, 171
PSID	Según cuenta	-, -
OD	80	80,0
LD	1	1,0
RID	0	0,0,0,0
MID	50	50, 0,0,0
SN	Según cuenta	-,-
Longitud	68	68,0
MDD - Canales - Velocidad - EEG	 2 128 Variable	 2,0 128,0 Variable (64 Bytes)
Checksum	Variable	Variable (2 Bytes)

Tabla 16. Estructura de mensaje de EEG bruto.

El mensaje recibido presentará una apariencia similar a la expuesta en el siguiente ejemplo.

186	171	109	6	80	9	1	0	0	0	0	0	50	0	0	0	101
6	68	0	2	0	128	0	101	245	246	244	213	244	152	245	207	244
60	245	106	245	26	245	135	244	136	245	81	245	246	244	252	244	140
245	176	244	80	245	127	245	244	244	166	244	135	245	64	245	242	244
42	245	86	245	156	244	93	245	128	245	241	244	172	244	144	245	18
245	5	245	254	49												

Figura 13. Paquete de mensaje con datos de EEG bruto.

Los campos sombreados en gris se corresponden a los datos del EEG del canal uno y los sombreados en azul representan los datos del EEG del canal dos. En el capítulo siguiente, se explicará el procesamiento que siguen los datos para obtener el valor real del EEG.

3.2.4.2. Mensaje de variables procesadas

Las variables procesadas son enviadas al ordenador cada segundo. Por defecto, se envían dos conjuntos de datos del espectro, siendo válido únicamente el primero de ellos.

Los campos del paquete están compuestos de los valores mostrados en la tabla 17.

Campo	Valores
MID	53
Longitud	364
MDD	Variable (364 Bytes)

Tabla 17. Valores de los campos de mensaje de variables procesadas.

Para analizar los datos de MDD, se recurre a la información que se adjunta en el Anexo1, referente a las estructuras y definiciones de datos. La estructura completa del mensaje se recoge en la tabla 18.

Campo de paquete	Valor	Valor en formato de paquete (Bytes)
SPI	0xabba	186, 171
PSID	Según cuenta	-, -
OD	376	120,1
LD	1	1,0
RID	0	0,0,0,0
MID	53	53, 0,0,0
SN	Según cuenta	-,-
Longitud	364	108,1
MDD - Información - Impedancia - Filtro - Suavizado - Máscara - Variables Procesadas - Espectro	Variable(24Bytes) Variable (8 Bytes) Variable (4 Bytes) Variable (4 Bytes) Variable (8 Bytes) Variable (72 Bytes) 0 (244 Bytes)	
Checksum	Variable	Variable (2 Bytes)

Tabla 18. Estructura de mensaje de variables procesadas.

Dentro del rango de variables procesadas, se encuentra información relativa a tres canales: canal uno, canal dos y canal doce (combinación de ambos canales). La información mostrada por el monitor es la recogida en el canal doce, siendo de éste del que extraeremos la información.

186	171	255	35	120	1	1	0	0	0	0	0	53	0	0	0	85
0	108	1	10	1	27	1	2	0	0	0	32	161	7	0	128	150
152	0	192	78	185	236	160	134	1	0	129	0	0	0	103	0	0
0	0	0	0	0	2	0	3	0	255	255	255	255	255	255	255	255
0	0	201	2	193	32	205	3	0	128	0	128	177	28	243	19	187
1	0	0	128	2	0	0	0	0	24	0	193	32	205	3	0	128
0	128	177	28	243	19	187	3	0	0	128	0	0	0	0	0	201
2	193	32	205	3	0	128	0	128	177	28	243	19	187	1	0	0
128	2	0	0	0	…		229	36								

Figura 14. Paquete de mensaje con variables procesadas.

En la figura 14 recoge un ejemplo de paquete donde se han sombreados las variables procesadas correspondientes al canal 12:

- Tasa de supresión (SR): Color verde
- Índice Biespectral (BIS): Color naranja
- Electromiograma (EMG): Color gris
- Calidad de señal (SQI): Color azul

Las variables procesadas son transmitidas como enteros. Para obtener su valor final deben dividirse por un factor contenido en la información del Anexo 1.

3.2.5. Diagrama de transmisión de mensajes

A modo de resumen, se incluye un diagrama con el flujo de mensajes que se implementarán entre el ordenador y el monitor.

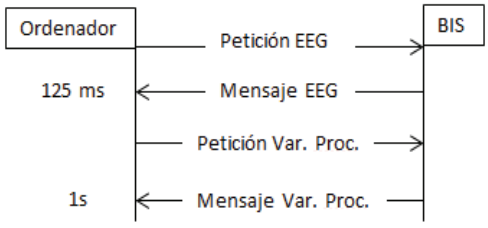

Figura 15. Diagrama de flujo de datos.

4.Desarrollo de la interfaz para la captura del EEG

4.1. Programación Orientada a Objetos

Con vistas a integrar el resultado de este trabajo fin de grado en el software general y respetando los requisitos de organización y modularidad de un proyecto de éstas características, se ha considerado conveniente aplicar los principios de programación orientada a objetos.

La herramienta que servirá de base para el desarrollo de la interfaz será Matlab. En sus últimas versiones, Matlab incluye herramientas propias de la programación orientada a objetos (classdef). No obstante, con objeto de facilitar la comprensión del código de futuros usuarios no familiarizados con este tipo de programación, y siguiendo la filosofía del proyecto original, se opta por el modelo orientado a objetos clásico de Matlab.

En términos generales, es posible encapsular o aislar los diferentes componentes que forman parte de la programación, evitando la proliferación de variables globales y sustituyéndolas por atributos de los objetos. Ello implica mayor seguridad, no siendo posible la manipulación directa de las variables. Toda clase en Matlab está formada por dos tipos de elementos:

- Atributos: Son las variables que almacenan el estado de un objeto particular de esa clase. Únicamente se puede acceder a un atributo desde un método de la clase del objeto.
- Métodos: Son las únicas funciones que están autorizadas a acceder directamente a los atributos de un objeto. Se construyen como funciones comunes de Matlab, siendo el primer parámetro a recibir un objeto de la clase.

Para la implementación de cada clase, se creará un directorio cuyo nombre coincida con el de la clase, anteponiendo el carácter "@" como prefijo. Dentro de éste, se colocarán todos los ficheros M con los métodos. Adicionalmente, se definirá el constructor como una función, con el mismo nombre de la clase, y en el que se especificarán los campos que conformarán los atributos del objeto. Finalmente, con ayuda de la función "class", se devuelve el objeto creado como valor de retorno del constructor.

Una opción que permite el Matlab es generar métodos privados. Estos se ubicarán en una carpeta dentro del directorio, bajo el nombre "private". La principal característica es que

los ficheros colocados dentro de "private" sólo serán accesibles desde otros métodos de la clase.

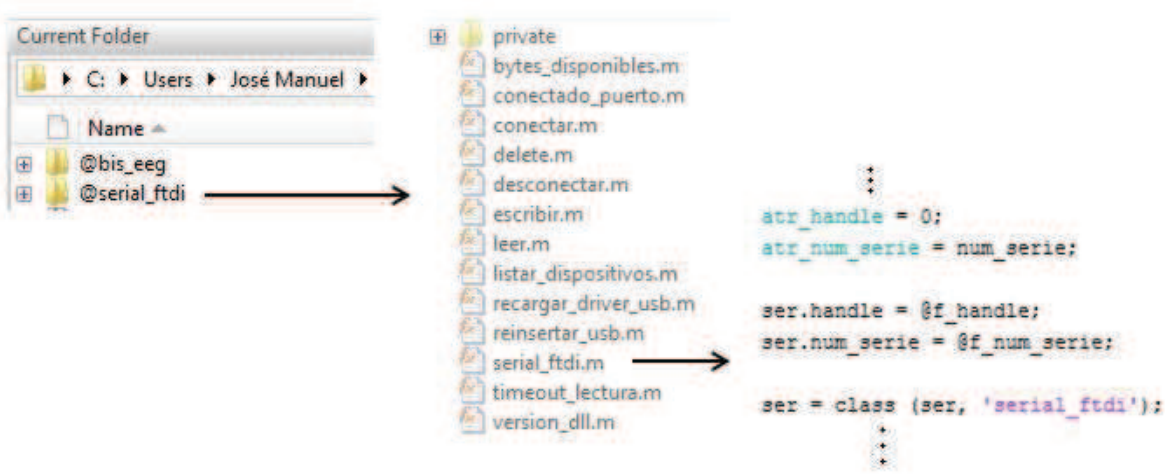

Figura 16. Esquema con los principales elementos de la programación orientada a objetos en Matlab.

Para ilustrar todos estos conceptos, véase la figura 16. En ella se observan los métodos de la clase serial_ftdi, incluyendo la carpeta "private" y, dentro del constructor, la implementación de la función class para la obtención del objeto, así como la definición de los atributos.

A pesar de las innumerables ventajas que ofrece este software matemático, presenta una importante limitación que perjudica al diseño de la aplicación en general: la inmutabilidad. En las siguientes secciones se explicará este problema y la metodología implementada para subsanarlo.

4.1.1. Problema de la inmutabilidad

La inmutabilidad consiste en la imposibilidad de modificar los parámetros de una función. Si se intenta llevar a cabo la modificación, los cambios se realizan sólo de manera local, no permaneciendo tras la ejecución de la función. La manera de aplicar los cambios en los parámetros es devolver como resultado de la función las variables modificadas.

En nuestro caso, si quisiéramos modificar un atributo de un objeto, la única manera sería devolver el objeto modificado como resultado del método. Este hecho introduce importantes problemas en el diseño y acceso de datos. Como solución, se opta por el uso de la técnica de "closure". Ésta se explica con detenimiento en el siguiente apartado.

4.1.2. Técnica de closure

La técnica del closure está basada en el uso de funciones anidadas dentro de otras, capaces de acceder a una variable local de la función contenedora. De esta manera, cuando la función externa se ejecuta, se forma una clausura, consistiendo en el código de la función interna y las referencias a todas las variables de la función externa que son requeridas por la clausura. Para aplicar el concepto anterior a nuestro programa, se asume el siguiente convenio:

- Los atributos cuyos valores puedan ser modificados no se almacenan en el campo de la estructura, sino en variables locales del constructor.

```
atr_handle = 0;
atr_num_serie = num_serie;
```

Figura 17. Atributos declarados como variables locales en la clase serial_ftdi.

- Los atributos del objeto pasan a ser handlers a funciones accesoras.

```
ser.handle = @f_handle;
ser.num_serie = @f_num_serie;
```

Figura 18. Atributos declarados como handlers a funciones accesoras.

- Para poder acceder a los atributos una vez creado el objeto, se definen funciones accesoras anidadas dentro del constructor. Estas funciones permitirán a los métodos del objeto leer el valor de un atributo o modificarlo.

```
function h = f_handle (h)
    switch nargin
        case 0
            h = atr_handle;
        case 1
            atr_handle = h;
    end
end
```

Figura 19. Funciones accesoras para modificar o leer el valor de un atributo.

En el caso de que se acceda a la función sin parámetros, se leerá el valor almacenado en la variable local. Si, por el contrario, la función incluye un parámetro, el atributo del objeto verá modificado su valor.

De esta forma, un objeto no se ve afectado por la inmutabilidad. Los atributos del objeto propiamente dichos son meros handlers a funciones accesoras. Los verdaderos atributos quedan escondidos en las closures de cada función accesora.

4.2. Programación de la interfaz

Basándonos en los conceptos introducidos en el apartado anterior, se emplearán dos clases que nos permitirán obtener el EEG bruto del monitor y las variables procesadas:

- @serial_ftdi: Clase encargada de definir y establecer las comunicaciones con el objeto serial para la comunicación entre el ordenador y el monitor BIS.
- @bis_eeg: Permite gestionar el manejo del puerto serie (objeto de clase serial_ftdi), recepción de datos, envío de comandos, clasificación de la información y representación del EEG bruto.

Además de las clases anteriores, se crean una serie de funciones genéricas que permiten adaptar el formato de los paquetes de datos recibidos al formato requerido según las necesidades. En los próximos epígrafes se procede al análisis de la implementación.

4.2.1. Clase @serial_ftdi

Tal y como se explicó en el capítulo 3, para la comunicación serial se va a utilizar el API de programación de FTDI. A través de los distintos métodos se procederá a la apertura, cierre y configuración de la comunicación, así como a la lectura y escritura de datos. Ésta clase ha sido tomada de la interfaz original, por lo que haremos una breve revisión de sus principales funciones sin entrar en detalles en el código.

4.2.1.1. Constructor

El constructor permite la asignación de valores que caracterizan la comunicación a través de argumentos de la función: baudios, número de bits de datos, bits de parada y paridad. Se comprueba que los datos introducidos son correctos y se almacenan en los atributos. En caso de no especificarse, se inicializan a los valores por defecto vistos en el capítulo 3.

4.2.1.2. Métodos

Los principales métodos implementados en la clase son:

- Listar_dispositivos.m: El primer paso para la comunicación es detectar e identificar los adaptadores conectados. A continuación, se almacena en una estructura el número de serie de cada dispositivo (identificación) y el parámetro de flags (permite saber si un dispositivo está ocupado o abierto por otro proceso).
- Conectar.m: Introduciendo el número de serie del dispositivo como argumento, realiza la conexión entre el monitor y el ordenador. En este punto, se configura la comunicación con los atributos de la clase.
- Desconectar.m: Cuando se termina de usar el adaptador USB, esta función permite cerrar el puerto, liberando los recursos ocupados.
- Leer.m: Método que permite la lectura de datos almacenados en el buffer interno del FTDI. Se puede especificar como argumento el número de bytes a leer, o leer el número total de bytes que componen el buffer omitiendo el argumento.
- Escribir.m: Si está conectado, se envían los datos a través del puerto serial, bien hasta finalice el envío de datos, o bien hasta que concluya el tiempo de escritura.
- Conectado_puerto: Método que permite comprobar si el dispositivo está conectado.

Todas estas funciones se complementan con métodos que permiten asegurar el funcionamiento y la robustez de la comunicación minimizando el número de incidentes.

4.2.2. Clase @bis_eeg

Esta clase aúna todas aquellas acciones que han de realizarse para la conexión, envío, recepción y adaptación de datos recibidos del monitor BIS, así como su visualización. Asimismo, se integrará un objeto de la clase serial_ftdi que permita la utilización de los métodos antes mencionados.

En primer lugar, se ha optado por desarrollar de manera genérica una serie de funciones cuyo cometido general se centra en la adaptación de los distintos formatos de datos a los requerimientos para cada caso. En concreto, se desarrollan tres funciones:

- Formato.m: Esta función permite adaptar los valores enteros al formato del paquete, es decir, formato binario y Little Endian.

```
function m = formato (valor, bits)
 %________FUNCIÓN PARA ADAPTACIÓN DE FORMATOS_______
% Función que permite adaptar los valores al formato requerido para el
% mensaje. Se introducen los numeros en ASCII.

v = valor;
r = [];
for n = 1:(bits/8)
    r(end+1) = mod (v, 256);
    v = floor (v / 256);
end
m = r;
end
```

Figura 20. Código de la función formato.m.

La función recibe como parámetros de entrada el valor a convertir y el número de bits que tendrá el valor a la salida. La función contiene un bucle que se ejecutará tantas veces como bytes se precisen a la salida. Se divide el valor entre 256, donde el resto de las sucesivas divisiones conformarán los bytes, almacenándose en las primeras posiciones del vector los valores menos significativos. De esta forma se consigue, además, tener el valor almacenado en formato Little Endian.

- Formato_RAW_EEG: Esta función permite obtener, a partir de los valores contenidos en el paquete del EEG bruto en complemento a dos, el valor real que adquiere el EEG, así como almacenar los resultados en un fichero de texto. El resultado de la función se devuelve en tres variables, referidas respectivamente al canal 1, canal 2 y tiempo.

```
function [ chan1, chan2, t ] = formato_RAW_EEG( dato, fichero, tiempo)
%   Función que permite obtener los valores reales del RAW_EEG contenidos
%   en el paquete de datos en formato complemento a dos con signo y
%   almacenarlos en un fichero de texto
    dato = dato(2:2:end)*256 + dato(1:2:end);
    m = dato>2^15;
    dato(m)=dato(m)-2^16;
    chan1 = dato(1:2:end);
    chan2 = dato(2:2:end);
    for i = 1: length(chan1)
        fprintf(fichero,'EEG: %i    Tiempo: %f \n',chan1(i), tiempo);
        tiempo=tiempo+(0.125/16);
    end
    t = tiempo;
end
```

Figura 21. Código de la función formato_RAW_EEG.m.

La función recibe como parámetro de entrada un vector que contendrá los valores del EEG bruto en dos bytes alternados para cada canal, un parámetro relativo al tiempo en el que se toma cada muestra y el fichero en el que se va a almacenar la trama del EEG. En primer lugar, multiplica el byte más significativo (byte pares) por 256 y se le suma el impar, de forma que se obtiene el valor real del EEG (dato). Como está en complemento a 2, aquellos datos que superen el valor 2^15 implica que el bit más significativo está a uno, siendo por tanto un número negativo. Para conseguir el valor final se le resta el peso del valor más significativo (2^16). Finalmente, los datos son almacenados en un fichero de texto.

- Formato_processed_vars: Esta función permite pasar el formato de los datos contenidos en el mensaje de las variables procesadas a valores enteros.

```
1   function [a] = formato_processed_vars( valor )
2   %Función que permite pasar el formato de los mensajes (Little-Endian)en bytes
3   %a entero
4       indice = 1;
5       total = 0;
6       for i = 1: length(valor)
7           total = total + valor(i)*indice;
8           indice = indice*256;
9       end
10      a = total;
11
12
13  end
```

Figura 22. Código de la función de Formato_processed_vars.

La función recibe como parámetro de entrada un vector en formato Little Endian. Desde el bucle se va multiplicando los elementos del vector por su peso correspondiente, devolviéndose la suma de todos los elementos a la salida.

Una vez explicadas estas funciones básicas para los cambios de formato, se procede a analizar en profundidad la clase bis_eeg.

4.2.2.1. Constructor

El constructor de la clase recibe como parámetro de entrada el fichero en el que se decide almacenar la trama del EEG. Internamente, el constructor de la clase se va a dividir en tres partes. En primer lugar, se procede a definir los atributos de la clase incluyendo los campos de cada paquete, la creación del objeto serial, los referidos al procesamiento de datos y el timer para controlar los intervalos de procesamiento. En una segunda parte, se hace

referencia al uso de la técnica del closure para la asignación de los atributos y finalmente, se implementan las funciones accesoras parasu modificación. Para analizar con detenimiento el constructor, se puede consultar el Anexo2 en el que se incluye el código íntegro de la clase bis_eeg.

4.2.2.1.1. Timer_serial

Uno de los elementos fundamentales del programa y que se declara en el constructor es el uso del timer. Una vez se envía el comando para la recepción de datos, es necesario procesarlos y emitir la representación en vistas a obtener un resultado en tiempo real. Cada paquete de datos de EEG es recibido cada 125 ms mientras que el de variables procesadas se obtiene cada segundo. Una primera opción que se podría estudiar es la implementación de un timer de alta velocidad que leyera cada paquete de datos de EEG, lo procesara y representara. No obstante, los timer de Matlab no se caracterizan por una precisión rigurosa, por lo que su uso podría comprometer el funcionamiento general del programa. Además, en vistas a poder integrar posteriormente nuestro programa en el software original, podría acarrear problemas con otros timer, impidiendo ejecuciones simultáneas.

Como alternativa, se propone la creación de un timer más lento, de forma que el procesamiento de datos incluya más de un paquete. El tiempo establecido inicialmente para el timer es de 0,5 segundos, con un retardo inicial de 0,1 segundos. Además, el timer es del tipo "fixed-spacing", indicado por la mayor seguridad y robustez que proporciona. La declaración del timer se incluye a continuación.

```
44        %Creación del timer para la lectura y procesamiento de datos
45 -      timer_serial = timer ('Name', 'timer_serial','StartDelay',...
46           0.1, 'Period', 0.5, 'ExecutionMode', 'fixedSpacing','BusyMode',...
47           'drop');
48
49 -      BIS.timer_serial = timer_serial;
50
```

Figura 23. Código de declaración del timer_serial.

Finalmente, se le asocia al timer la función ProcesaDatos, cuya aplicación se describirá posteriormente.

```
79 -      set (timer_serial, 'TimerFcn', @(o, e) ProcesaDatos (BIS));
```

Figura 24. Asociación del timer a la función ProcesaDatos.

4.2.2.2. Métodos

El objetivo de este apartado es explicar el procedimiento seguido para la captura del EEG a través de los métodos de la clase bis_eeg. A continuación, se hace una revisión de los métodos empleados explicando las partes fundamentales.

4.2.2.2.1. Métodos relativos a la conexión serial

Estos métodos hacen referencia a todas aquellas acciones necesarias para establecer la comunicación entre el monitor BIS y el ordenador

- Listar_dispositivos.m: El objetivo de este método es obtener el número de serie del monitor BIS, haciendo uso de los métodos del objeto serial declarado en el constructor. El resultado lo almacena en el atributo "puerto" de la clase.
- Conectar.m: Método que permite conectar a través del número de serie el monitor con el ordenador empleando los métodos del objeto serial. Se le puede asignar un puerto como argumento, o bien tomar el obtenido a través de la función Listar_dispositivos.
- Conectado_puerto.m: Método que devuelve un uno en el caso de que el puerto esté conectado y un cero en caso contrario.
- Desconectar.m: A través de este método es posible parar el timer_serial una vez se haya finalizado el proceso, así como desconectar el puerto y reiniciar los atributos del objeto creado a los valores por defecto.

4.2.2.2.2. Métodos relativos al procesamiento y representación de datos

En este apartado, se recoge la descripción de aquellos métodos empleados para el procesamiento y representación tanto del EEG como de las variables procesadas.

- Comienzo.m: Este método permite el inicio del proceso una vez se ha establecido la conexión. A su vez llama a las funciones que confeccionan el mensaje para la petición del EEG bruto (Selección_EEG.m) y las variables procesadas (Selección_pvars.m).
- Selección_EEG: Método encargado de modificar el valor de los atributos asociados a los campos del mensaje, estableciendo los valores adecuados para la petición del EEG bruto. Finalmente, manda el mensaje a través del serial con la función Send_M.m e inicia el timer_serial.

```
function  seleccion_EEG(BIS)
%Función que genera el mensaje para recepción de EEG
    MID = 111;
    SN = 0;
    Longitud = 2;
    RID = 4;
    PSID = 0;
    MDD = 128;
    BMDD = 16;
%Se actualizan los datos de los atributos
    BIS.MID(MID);
    BIS.Longitud(Longitud);
    BIS.RID(RID);
    BIS.MDD(MDD);
    BIS.BMDD(BMDD);
%Se envía el mensaje y comienza el timer
    Send_M(BIS);
    start (BIS.timer_serial);


end
```

Figura 25. Código del método selección_EEG.

- Selección_pvars: Método similar al anterior que permite modificar el valor de los atributos adecuándolos al mensaje de petición de las variables procesadas.

- Send_M: Método cuya finalidad es la confección del paquete del mensaje a enviar. En primer lugar, adapta los valores de los campos del mensaje registrados con las dos funciones anteriores al formato del paquete, a través de la función formato.m.

```
function Send_M (BIS)
% Función encargada de confeccionar los mensajes con los valores de los
% campos almacenados en los atributos
    MID = formato(BIS.MID(),32);
    SN = formato(BIS.SN, 16);
    Longitud = formato (BIS.Longitud(), 16);
    RID = formato (BIS.RID(), 32);
    PSID = formato (BIS.PSID(), 16);
```

Figura 26. Adaptación del valor de los campos del mensaje al formato de paquete en función Send_M.

La siguiente parte del código se centra en la confección de las tres capas del mensaje según la estructura analizada en el capítulo 3.

```
% Generación de las tres capas del mensaje.
    %Creación de la capa 3
    if BIS.MDD()~=0
        MDD = formato (BIS.MDD(), BIS.BMDD());
        Layer3 = [MID, SN, Longitud,MDD];
    else
        Layer3 = [MID, SN, Longitud];
    end
    % Creación de la capa 2
    Layer2 = [RID, Layer3];

    % Creación de la capa 3
    OD= formato (length(Layer2),16);
    LD = formato (1, 16);

    Layer1 = [PSID, OD, LD, Layer2];
    CheckSum = formato((sum(Layer1)),16);
    Layer1 = [BIS.SPI, Layer1, CheckSum];
```

Figura 27. Creación de las diferentes capas de mensaje en función Send_M.

Finalmente, con el método Escribir.m del objeto serial se envían los datos a través del puerto hacia el monitor.

Una vez se ha habilitado el proceso de recepción de paquetes mediante la función Comienzo.m, se activa timer_serial cada 0,5s y, por tanto, la función asociada a él: ProcesaDatos.m. El objetivo de esta función es la lectura de datos en el buffer serial, el procesamiento de esos valores y la preparación para su posterior representación gráfica. Para ello, este método llama a su vez a tres métodos de la clase bis_eeg: Leer.m Paquete.m y Representación.m. A continuación, se procede a explicar estos métodos y a su vez, los métodos auxiliares que precisan.

- Leer.m: Método que lee y almacena los datos contenidos en el buffer actualmente en el atributo DatosLeidos.

- Paquete.m: Uno de los principales inconvenientes de establecer el timer en 0,5 segundos es que el buffer serial contiene más de un paquete de datos para cada consulta. Asimismo, al combinar diferentes tipos de paquete (EEG bruto y variables procesadas), no todos se pueden clasificar bajo un patrón unívoco. Para ello, surge

el método Paquete.m que, además, clasifica el tipo de paquete y extrae los datos según proceda.

```
13 -    while i<=length(datos_leidos)
14 -        if (datos_leidos(i-1)==186)&&(datos_leidos(i)==171)&&...
15                 (sum(formato(sum(m(3:length(m)-2)),16)==[m(length(m)-1),m(end)])==2)
16 -            Valor_EEG(BIS,m);
17 -            m=[datos_leidos(i-1),1,1];
18 -            k=2;
19 -            i =i+1;
20
21
22 -        else
23 -            m(k)=datos_leidos(i-1);
24 -            i=i+1;
25 -            k=k+1;
26
27
28 -        end
29
30 -    end
31      % Para el caso del último valor leído del buffer serial
32 -    m(k)=datos_leidos(end);
33 -    if(sum(formato(sum(m(3:length(m)-2)),16)==[m(length(m)-1),m(end)])==2)
34 -         Valor_EEG(BIS, m);
35
36
37 -    end
```

Figura 28. Extracto de código del método Paquete.m para la clasificación de los tipos de paquete recibido.

Para la separación en paquetes, se comienza a recorrer el vector de datos leídos almacenando cada valor en un vector auxiliar "m". En el caso de que el valor actual de análisis sea igual a 186 y el siguiente 171, podría ser indicativo de que comienza un nuevo paquete de datos. No obstante, este hecho no es excluyente con que de forma aleatoria surja una combinación de ambos valores sin ser necesariamente el final del paquete. Para ello, además, se debe verificar que la suma de los datos coincida con el checksum de "m". En caso que se cumplan ambas condiciones, se procede a la clasificación del tipo de paquete con la función Valor_EEG.m.

- Valor_EEG.m: A través de este método se clasifica el tipo de paquete en función del formato.

```
6       Comprobamos que el formato se corresponde con tipo de mensaje RAW_EEG
7 -     if (length(paquete)==90)
8 -         if (paquete(1)==186 && paquete(2)==171)
9 -             if (sum(paquete(13:16)==formato(50,32))==4)
10 -                dato = paquete(25:length(paquete)-2);
11 -                [a,b,t] = formato_RAW_EEG(dato,BIS.fichero,BIS.tiempo());
12 -                BIS.tiempo(t);
13 -                chan1 = [chan1,a];
14 -                chan2 = [chan2,b];
15 -                BIS.chan1(chan1);
16 -                BIS.chan2(chan2);
17
```

Figura 29. Extracto de código del método Valor_EEG.m para la clasificación de paquetes de EEG bruto.

En la figura 26 se observa que, si el paquete está formado por 90 bytes y, además, el campo de identificación de mensaje se corresponde con el estipulado en el capítulo 3 para el EEG bruto, se extrae la información del paquete a través de la función formato_RAW_EEG, almacenándose los resultados en los atributos "chan1" y "chan2".

```
25      % Compromabos que el formato se corresponde con el tipo de mensaje
26      % variables procesadas
27 -    else if (length(paquete)==386)
28 -            if (paquete(1)==186 && paquete(2)==171)
29 -                if (sum(paquete(13:16)==formato(53,32))==4)
30 -                    BIS.SQI_Temp(round((formato_processed_vars(paquete(133:136)))/10));
31 -                    BIS.EMG_Temp(round((formato_processed_vars(paquete(131:132)))/100));
32 -                    BIS.SR_Temp(round((formato_processed_vars(paquete(117:118)))/10));
33 -                    if ((BIS.SQI_Temp()) > 15)
34 -                        BIS.IndBis_Temp(round((formato_processed_vars(paquete(123:124)))/10));
35 -                    end
```

Figura 30. Extracto de código del método Valor_EEG.m para la clasificación de paquetes de variables procesadas.

Por su parte, se sigue el proceso análogo para la clasificación de paquetes con información de variables procesadas, extrayéndose los recursos necesarios. Como aspecto importante cabe resaltar que, en el caso de que el índice de calidad de señal tenga un valor por debajo a quince (valor mínimo aceptable), prevalecerá el valor anterior de BIS. De esta forma, queda almacenada la información relevante en los atributos de la clase.

- Representación.m: Para la representación gráfica de las variables procesadas se ha optado por tomar una escala de tiempo que emule el monitor BIS, esto es, de 4 segundos. El objetivo de esta función es almacenar los datos extraídos de los paquetes de EEG bruto en un vector de tal forma que se actualice la trama de datos

a representar cada vez que se ejecute la función ProcesaDatos asociada al timer, simulando la actualización de datos en tiempo real.

```
11 -        for indice =1:16:(length(valor1))
12 -            representacion1(inicio:inicio+15) = valor1(indice:indice+15);
13 -            BIS.RepChan1(representacion1);
14 -            representacion2(inicio:inicio+15) = valor2(indice:indice+15);
15 -            BIS.RepChan2(representacion2);
16 -            inicio = inicio +16;
17 -            if inicio >=512
18 -                inicio = 1;
19 -            end
```

Figura 31. Extracto de código del método Representacion,m para la creación del vector de datos a representar.

A través del bucle for, se actualiza el vector que contiene los datos a representar ("representación"), actualizándose el atributo de la clase. En el caso de que el índice del valor exceda el tamaño del vector para la representación, se inicializa a uno, sobrescribiendo los datos iniciales.

Finalmente, una vez ha concluido el proceso, se procede al envío de los mensajes referentes a la parada del EEG bruto y de las variables procesadas a través del método Parada.m. Éste a su vez, recurre a las funciones parar_EEG.m y parar_pvars cuyo funcionamiento es similar a los métodos de selección explicados previamente.

De forma adicional, se incluyen dos métodos que serán empleados por la interfaz de usuario para la representación del EEG y las variables procesadas.

- Rep.m. Este método recibe como parámetro de entrada, además del objeto de la clase eeg_bis, el handler de la gráfica en el que se representarán los resultados del EEG, modificando sus valores a través de la función "set". En este punto, cabe resaltar que, aunque se registran los valores de ambos canales, sólo se procederá a la representación del canal uno, dada la mayor representatividad de los datos que contiene en el ámbito que aquí se plantea

```
1    function Rep( BIS,h )
2    %Función encargada de la representación final
3 -    t = (0:0.125/16:4-0.125/16);
4 -    set(h,'YData',BIS.RepChan1());
5 -    set(h,'XData',t);
6
7
8 -  end
```

Figura 32. Método Rep.m para la representación del EEG bruto.

- Actualizacion.m: Método que recibe como parámetro de entrada los handlers de los objetos en la interfaz gráfica cuya propiedad "string" se debe modificar de acuerdo a las variables procesadas. Este método, además, permite almacenar los valores de BIS, Índice de Calidad de Señal, Electromiograma y Tasa de Supresión en el fichero pasado como último argumento de la función.

```
1  function  Actualizacion( BIS, handle1,handle2,handle3,handle4,fid )
2  % Función que permite actualizar los valores de las variables procesadas y
3  % volcarlos en el fichero fid.
4
5
6  s = ActualizaDatos(BIS);
7  set(handle1,'string',s(1));
8  set(handle2,'string',s(2));
9  set(handle3,'string',s(3));
10 set(handle4,'string',s(4));
11 fprintf(fid,'BIS: %i   SR: %i   EMG: %i   SQI: %i    Hora: %s \n',...
12     s(1), s(2),s(3),s(4), datestr(now));
13 end
```

Figura 33. Método para la actualización y registro de las variables procesadas.

Dentro de este método se recurre a ActualizaDatos, con el que se actualiza la información relativa a los parámetros de las variables procesadas con los valores actuales que contiene el paquete de datos.

```
1  function s = ActualizaDatos( BIS )
2  %Función encargada de actualizar lso datos a motrar por pantalla cada 5
3  %segundos
4
5
6      s(1) = BIS.IndBis(BIS.IndBis_Temp());
7      s(2) = BIS.SR(BIS.SR_Temp());
8      s(3) = BIS.EMG(BIS.EMG_Temp());
9      s(4) = BIS.SQI(BIS.SQI_Temp());
10
11
12 end
```

Figura 34. Código del método ActualizaDatos.m.

Para un analizar con más detenimiento del código, se adjunta como Anexo2 la declaración e implementación íntegra de todos los métodos con comentarios aclaratorios.

4.3. Diseño de la interfaz de usuario

Con vistas a dotar al usuario final de una forma de acceso a la funcionalidad implementada en el apartado anterior, se opta por el diseño de una interfaz gráfica. La base de diseño de la misma estará orientada a generar una interfaz sencilla que facilite su uso de manera intuitiva. Para el diseño se empleará la herramienta Guide que proporciona el Matlab que generará la función final con el nombre Aplicacion.m.

El diseño final se dividirá en 3 zonas:

- Zona de datos del paciente: En esta zona se recogerá la información relativa a los datos del paciente a monitorizar (nombre, edad, peso y altura).
- Zona destinada a la comunicación con el monitor: En este espacio, el usuario podrá establecer la conexión con el dispositivo y dar la orden de comienzo de proceso
- Zona de visualización de variables: Zona reservada para la representación del EEG bruto y la visualización de las variables procesadas.

El aspecto final de la interfaz es el recogido en la figura 32.

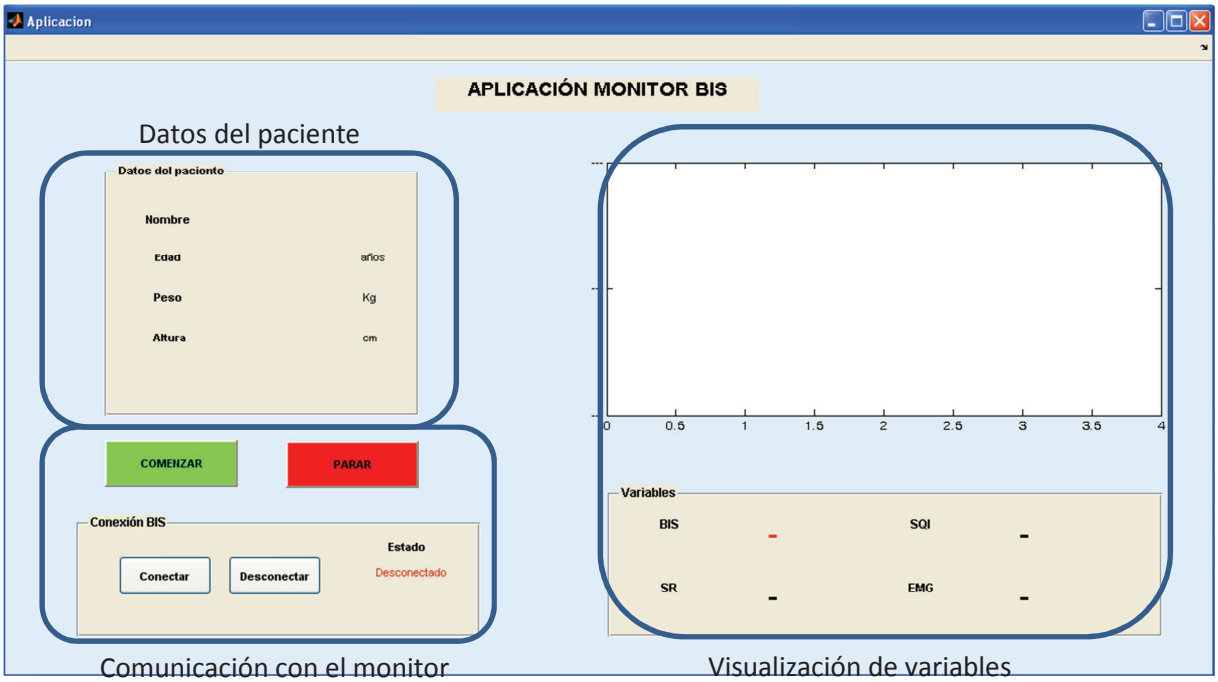

Figura 35. Estructura de la interfaz gráfica.

En los siguientes apartados se procede a analizar la implementación de cada una de las zonas o funcionalidades que ofrece la aplicación.

4.3.1. Datos del paciente

Para el registro de los datos del paciente, con la apertura de la aplicación se genera un diálogo emergente que solicita los datos al usuario. Estos datos son asociados a los valores de campos de los correspondientes elementos de la interfaz gráfica.

```
55 -    x = inputdlg({'Nombre','Edad (años)','Peso (Kg)','Altura (cm)'},...
56                  'Introduzca los datos del paciente', [1 50; 1 15; 1 15;1 15]);
57 -    handles.x = x;
58 -    set(handles.text6,'string',handles.x(1));
59 -    set(handles.text7,'string',handles.x(2));
60 -    set(handles.text8,'string',handles.x(3));
61 -    set(handles.text9,'string',handles.x(4));
```

Figura 36. Extracto de código de la función Aplicación.m para la petición de los datos del paciente.

Además, con la apertura de la aplicación se generarán dos ficheros de texto destinados a almacenar los datos recogidos en el proceso de monitorización. La cabecera de cada fichero incluirá los datos del paciente.

```
62      %Creamos el fichero de texto
63 -        handles.fid=fopen('resultado.txt','w');
64 -        fprintf(handles.fid, 'Nombre: %s \n',char(handles.x(1)));
65 -        fprintf(handles.fid, 'Edad: %s \n',char(handles.x(2)));
66 -        fprintf(handles.fid, 'Peso: %s\n',char(handles.x(3)));
67 -        fprintf(handles.fid, 'Altura: %s\n ',char(handles.x(4)));
68
69      %Creamos el fichero de texto para el EEG
70 -        handles.fid2=fopen('EEG.txt','w');
71 -        fprintf(handles.fid2, 'Nombre: %s \n',char(handles.x(1)));
72 -        fprintf(handles.fid2, 'Edad: %s \n',char(handles.x(2)));
73 -        fprintf(handles.fid2, 'Peso: %s\n',char(handles.x(3)));
74 -        fprintf(handles.fid2, 'Altura: %s\n ',char(handles.x(4)));
75
```

Figura 37. Extracto de código de la función Aplicación.m para la creación de los ficheros de texto.

Finalmente, se genera el objeto BIS de la clase bis_eeg pasándole como argumento el fichero de texto del EEG.

4.3.2. Comunicación con el monitor

Antes del comienzo del proceso, es necesario que el usuario conecte el dispositivo con el ordenador. En caso de no llevarse a cabo, aparecerá un mensaje de error que alertará de esta situación.

```
160 -   if ~conectado_puerto (handles.BIS)
161 -       errordlg('Debe conectar primero el monitor','ERROR')
```

Figura 38. Extracto de código de la función Aplicación.m para creación de mensaje de error en caso de falta de conexión.

Para la conexión del dispositivo, se debe pulsar el botón "Conectar", asociado a las funciones Listar_dispositivos.m y Conectar.m. Cuando la conexión se haya establecido, la etiqueta "Estado" pasará a estar "Conectado".

```
130 -   listar_dispositivos(handles.BIS);
131 -   if ~conectado_puerto(handles.BIS)
132 -       conectar(handles.BIS);
133 -   end
134 -   if conectado_puerto(handles.BIS)
135 -       set(handles.text14,'string','Conectado');
```

Figura 39. Extracto de código de la función Aplicación.m para conexión de dispositivo

Para solicitar la desconexión, se procede a pulsar el botón "Desconectar". De forma análoga al caso anterior, el botón se encuentra asociado a la función Desconectar.m, modificándose la etiqueta "Estado" a "Desconectado".

Una vez se asegura que el monitor está conectado, se podrá iniciar el proceso de monitorización. Para ello se debe pulsar el botón "COMENZAR". Ello lleva a la llamada del método Comienzo.m. De forma simultánea, se iniciarán dos timers cuya justificación se verá en el siguiente apartado.

```
162 -   elseif (strcmp(get(handles.timer_Rep,'Running'),'off'))
163 -       Comienzo(handles.BIS);
164 -       start(handles.timer_Rep);
165 -       start(handles.timer_repvars);
```

Figura 40. Extracto de código de la función Aplicación.m para el comienzo de la monitorización.

En caso de que el usuario quiera parar el proceso, bastará con pulsar el botón "PARAR", asociado al método Parada.m. La parada llevará asociada también la parada de los dos timers y el cierre de los ficheros de texto.

```
181 -       Parada(handles.BIS);
182 -       stop(handles.timer_repvars);
183 -       stop(handles.timer_Rep);
184 -       fclose(handles.fid)
185 -       fclose(handles.fid2)
```

Figura 41. Extracto de código de la función Aplicación.m para la parada de la monitorización.

4.3.3. Visualización de variables

Una vez se ha habilitado el comienzo del proceso, se debe visualizar la representación del EEG bruto, así como el valor de las variables procesadas. Para ello, se deben tener en cuenta las siguientes consideraciones:

- El valor del timer para la lectura del buffer serial se estableció en 0,5 segundos. En este tiempo se obtienen en media un total de 64 datos de EEG bruto. En un compromiso para evitar cambios muy bruscos de la gráfica y, a su vez, no poner en riesgo el resto del programa por valores bajos del timer, se opta por actualizar la representación cada 0,5 segundos.
- En la práctica, tanto para la aplicación de procesos de control como para visualización, es suficiente con mostrar las variables procesadas actualizadas cada cinco segundos, por lo que se generará un timer con este periodo.

4.3.3.1. Gráfica del EEG bruto

El método de representación de la clase (Rep) será llamado a través del timer timer_Rep, definido como viene recogido en la figura 39.

```
88 -    timer_Rep = timer ('Name', 'timer_Rep','StartDelay',...
89          0.1, 'Period', 0.5, 'ExecutionMode', 'fixedSpacing','BusyMode',...
90          'drop');
91 -    handles.timer_Rep = timer_Rep;
92 -    set (timer_Rep, 'TimerFcn', @(o, e) Rep (handles.BIS,handles.h));
93
```

Figura 42. Extracto de código de la función Aplicación.m para la declaración del timer timer_Rep.

El argumento handles.h corresponde al handle de un plot creado inicialmente y modificado para la representación de los valores de EEG bruto.

```
81 -    handles.h =plot((0:0.125/16:4-0.125/16),zeros(1,512));
82 -    set(handles.axes1,'YLim',[-4500,-1500]);
83 -    set(handles.axes1,'YLimMode','manual');
84 -    set(handles.axes1,'YTick',(-4500:1500:-1500))
85 -    set(handles.axes1,'YTickLabel',['-' '-' '-']);
86 -    set (handles.axes1,'XLim',[0,4]);
```

Figura 43. Extracto de código de la función Aplicación para confeccionar la gráfica de EEG bruto.

Un aspecto importante a tener en cuenta para la representación es que los valores obtenidos no se corresponden con los valores representados en el monitor BIS VISTA. Como alternativa, los ejes se fijan de forma que la información que aparece en la representación de la aplicación cubra el mismo rango que la visualización en el monitor, sin otorgarles valores por el momento.

4.3.3.2. Visualización de las variables procesadas

El timer de cinco segundos (timer_repvars) será el encargado de llamar al método Actualizacion.m

```
95      %Creamos el Timer para la representación de las variables procesadas
96 -        timer_repvars = timer ('Name', 'timer_repvars','StartDelay',...
97             0.1, 'Period', 5, 'ExecutionMode', 'fixedSpacing','BusyMode',...
98             'drop');
99 -            handles.timer_repvars = timer_repvars;
100 -           set (timer_repvars, 'TimerFcn', @(o, e) Actualizacion (handles.BIS,...
101                 handles.text20,handles.text21,handles.text22,handles.text23,handles.fid));
```

Figura 44. Extracto de código de la función Aplicación.m para la declaración del timer timer_repvars.

Como argumentos de la función se pasan, además del objeto de la clase, los handles de los textos que recogen los valores a modificar, así como el fichero en el que se almacenarán los datos recogidos.

El documento íntegro con la implementación del código relativo a la interfaz gráfica se adjunta como Anexo3 al final de este trabajo. De esta forma, queda analizada la interfaz realizada para la captura del EEG.

5.Validación de resultados

5.1. Validación de la interfaz de usuario

Una vez realizado el código, se procede a comprobar si los resultados obtenidos cumplen con los requisitos establecidos inicialmente. Se comienza por validar el resultado final de la interfaz de usuario. Al ejecutar la función Aplicación.m a través de línea de comandos de Matlab aparece el siguiente mensaje:

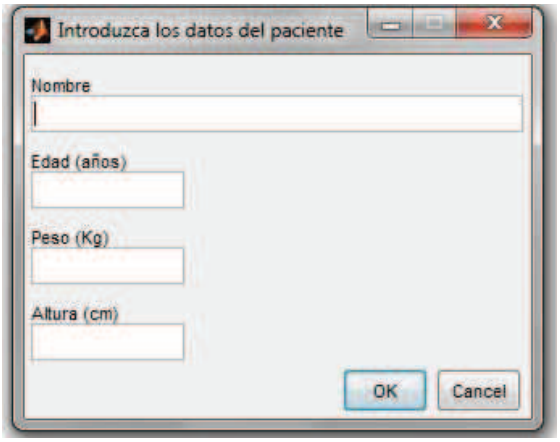

Figura 45. Cuadro de diálogo para introducir los datos del paciente.

Tras rellenar los campos, aparece la pantalla principal de nuestra interfaz, con los valores relativos al paciente ya almacenados.

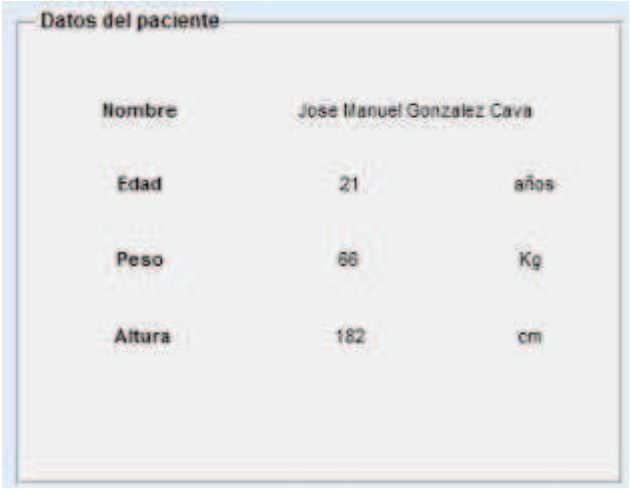

Figura 46. Cuadro con los datos del paciente.

Para comenzar con el proceso, es preciso realizar la conexión con el dispositivo. En caso de que no esté conectado, aparecerá el siguiente cuadro con un mensaje de error:

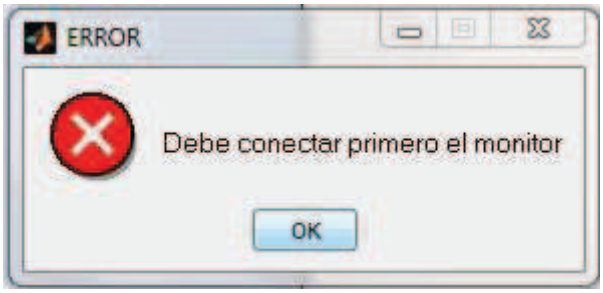

Figura 47. Mensaje de error en caso de que el monitor esté desconectado.

Una vez conectado el dispositivo, se procede al comienzo de la monitorización, apareciendo el EEG bruto en la gráfica y los valores relativos a las variables procesadas. El aspecto general que adquiere la aplicación en funcionamiento es el recogido en la figura 48.

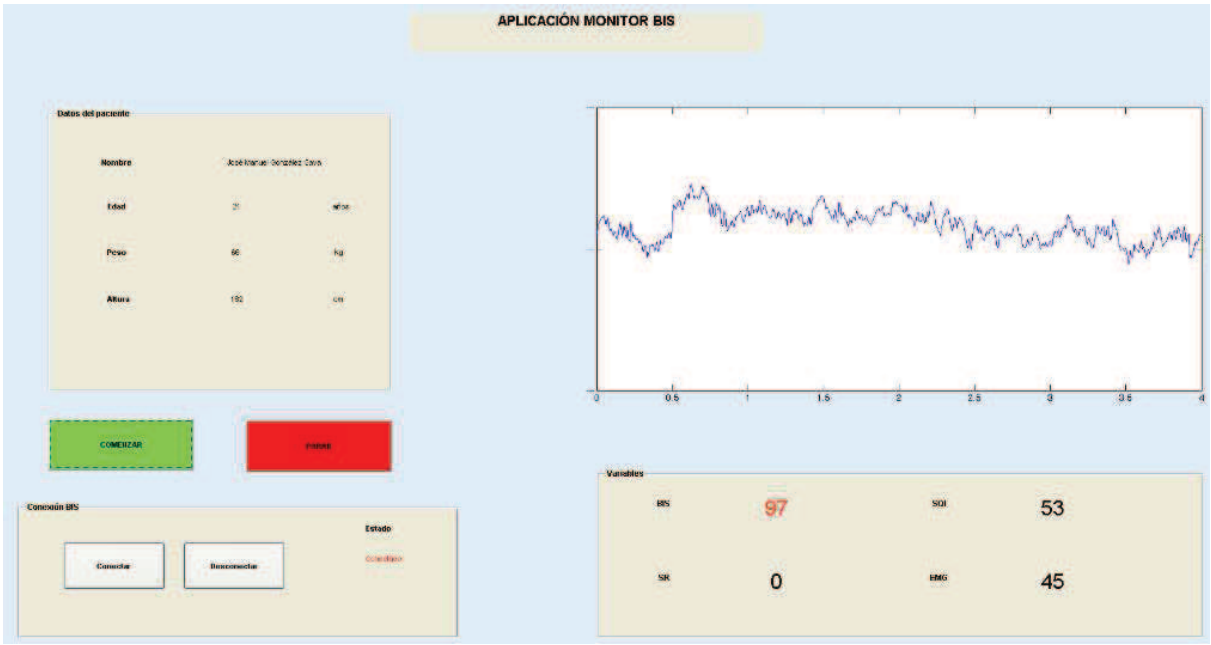

Figura 48. Aspecto general de la aplicación de usuario.

Una vez finalizada la monitorización, se procede al parado y a la desconexión del dispositivo con los botones habilitados para tal fin. El resultado se desarrolla sin incidentes, por lo que se da por válida la interfaz de usuario.

5.2. Validación de la gráfica de EEG bruto

Como se comentó en el capítulo anterior, una problemática encontrada en el desarrollo del proyecto es la falta de información acerca del formato final de los datos procesados con respecto a la información mostrada en el monitor. Si bien la escala en el monitor BIS está dividida en cuatro regiones donde cada una representa una diferencia de potencial de 25 μV, los valores encontrados tras el procesamiento oscilan en torno a los -2500 (sin unidades).

Como medida provisional hasta encontrar un parámetro que relacione ambos resultados se ha optado por establecer el límite de los ejes a un valor tal que la información visualizada por el monitor coincida con la información mostrada por la aplicación.

A continuación, se procede a analizar y comparar las formas del EEG bruto mostrados por el monitor BIS y por la aplicación.

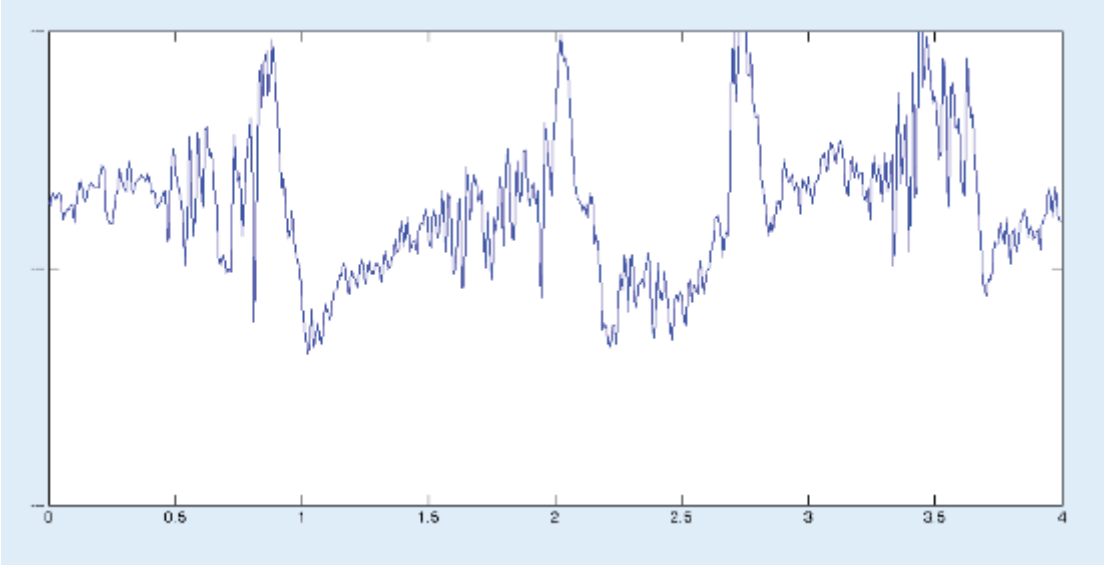

Figura 49. Gráfica del EEG obtenida a través de la aplicación.

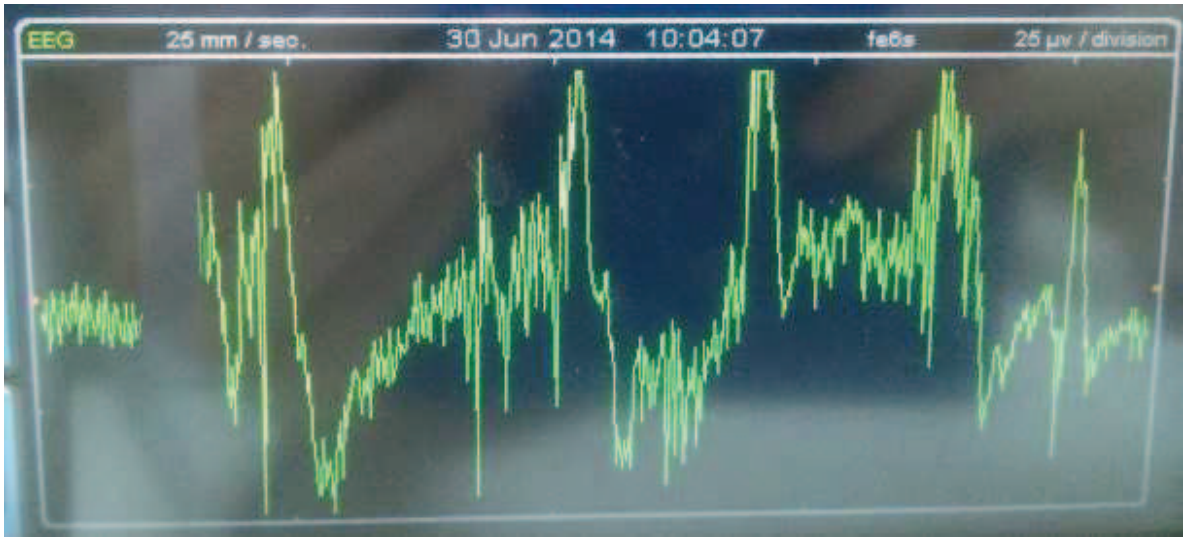

Figura 50. Gráfica del EEG obtenida a través del monitor BIS VISTA.

En términos generales se observa que las formas en ambas representaciones se respetan, por lo que, a pesar de no tener los valores numéricos, la información más relevante relativa al EEG se puede obtener.

De la misma forma, procedemos a analizar intervalos de tiempo, en vistas de garantizar que se respeta también el formato original de la señal

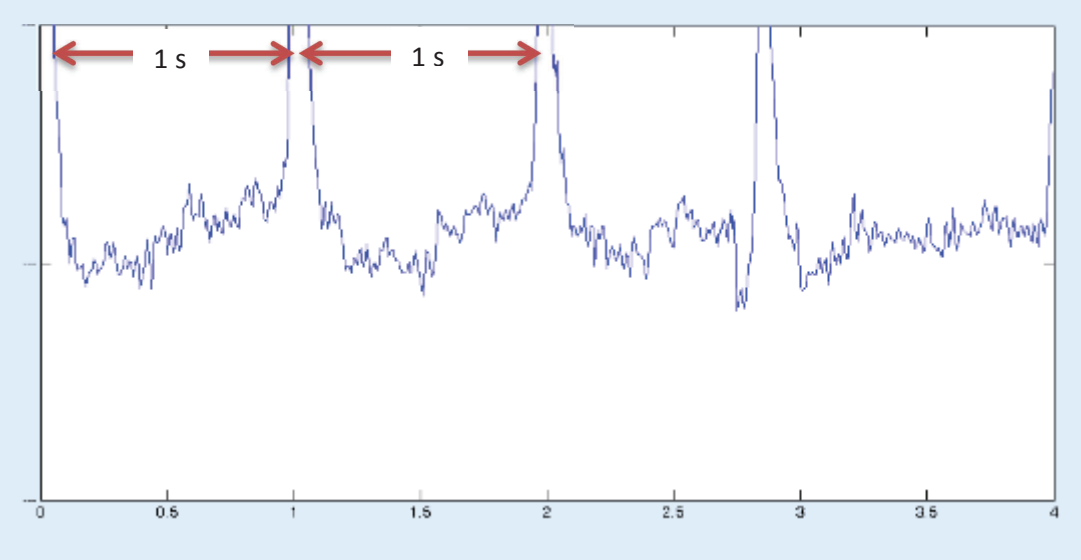

Figura 51. Gráfica del EEG obtenida a través de la aplicación para comprobación de tiempo.

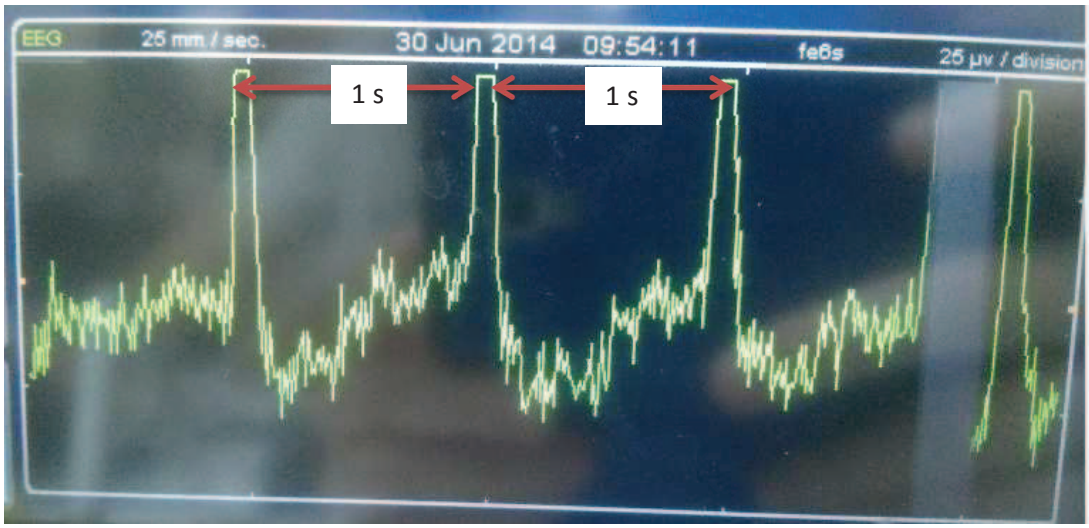

Figura 52. Gráfica del EEG obtenida a través de monitor BIS VISTA para comprobación de tiempo.

Pese al desplazamiento en el eje temporal (principalmente debido a que la trama en la aplicación no tiene por qué comenzar con el barrido temporal del BIS), se observa que los tiempos entre picos se mantienen.

Las dos evidencias anteriores nos permiten llegar a la conclusión de que la representación realizada sobre el EEG se realiza de manera correcta.

5.3. Validación de las variables procesadas

En este nuevo apartado se analiza la validez de los datos mostrados por la aplicación de las variables procesadas registradas en comparación con los valores mostrados por el monitor. Para ello, analizamos los valores en un instante de tiempo y comparamos los dos métodos de monitorización.

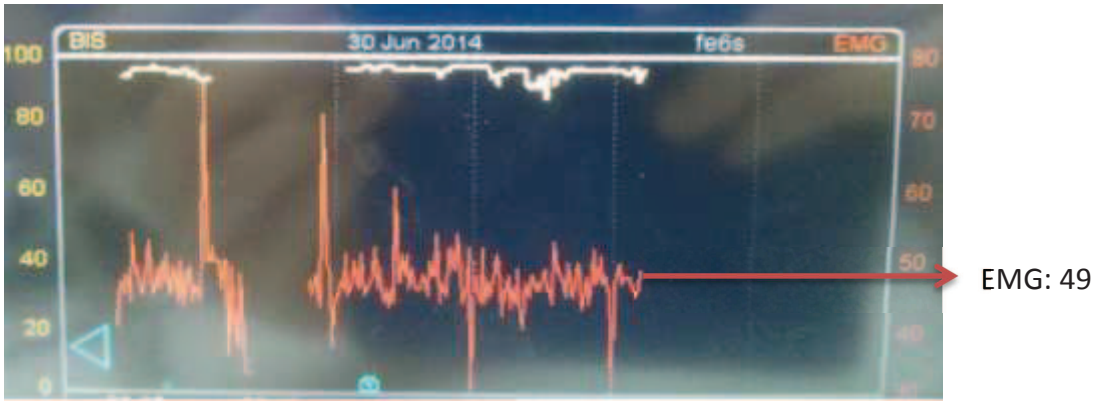

Figura 53. Gráfica del EMG obtenida con el monitor BIS VISTA.

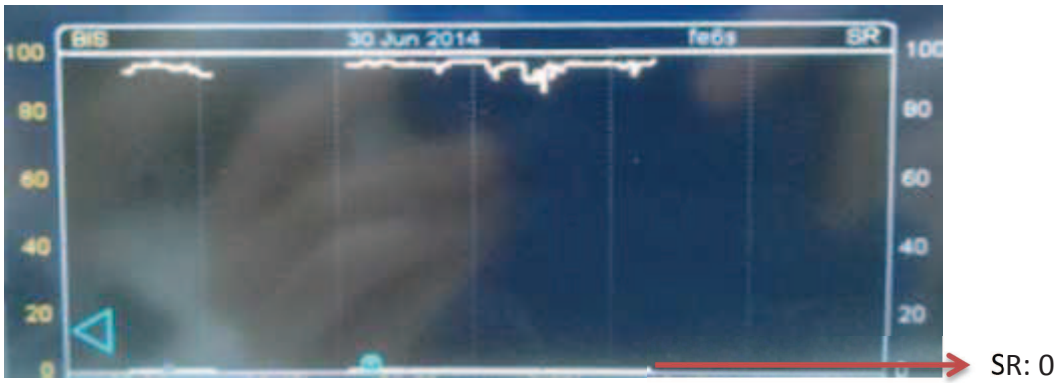

Figura 54. Gráfica del SR obtenida a través del monitor BIS VISTA.

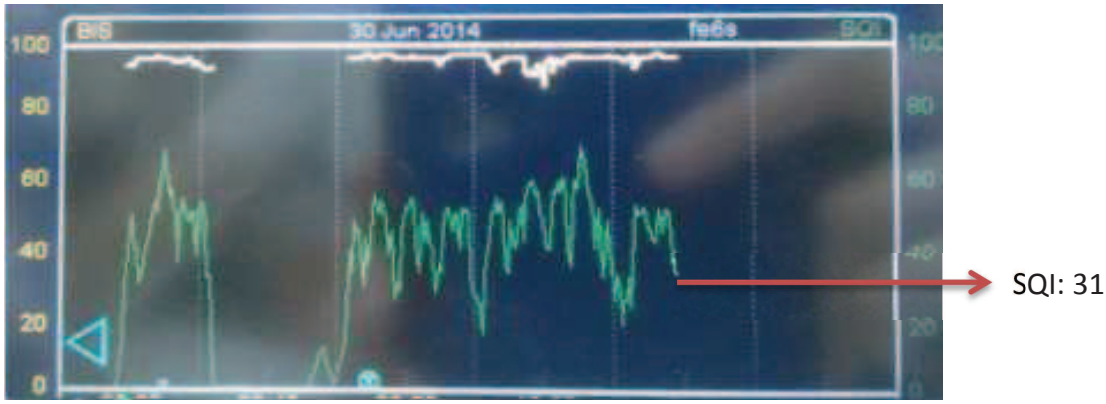

Figura 55. Gráfica del SQI obtenida a través del monitor BIS VISTA.

Figura 56. Valor de BIS por el monitor BIS VISTA.

En la figura 57 se añaden los datos mostrados por la interfaz gráfica.

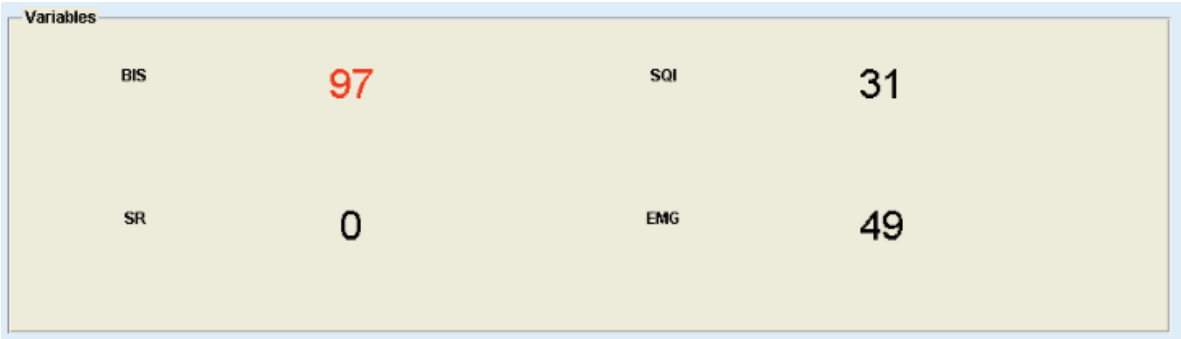

Figura 57. Valores de las variables procesadas mostrados por la interfaz gráfica.

Se comprueba que los resultados mostrados a través de la interfaz gráfica se corresponden con los registrados por el monitor BIS VISTA. Todo ello nos lleva a validar el funcionamiento del programa en lo referente a las variables procesadas.

5.4. Ficheros generados

Una vez se detiene la simulación, se generarán dos ficheros: un primer fichero que contendrá los valores de las variables procesadas almacenadas cada 5 segundos; un segundo fichero que contendrá la información relativa al EEG. Ambos aparecerán identificados por los datos del paciente. Como muestra de los datos registrados, se añaden las figuras 58 y 59.

```
Nombre: José Manuel González Cava
Edad: 21
Peso: 66
Altura: 182
BIS: 0   SR: 0   EMG: 0   SQI: 0    Hora: 30-Jun-2014 17:27:07
BIS: 97   SR: 0   EMG: 52   SQI: 23    Hora: 30-Jun-2014 17:27:12
BIS: 97   SR: 0   EMG: 51   SQI: 26    Hora: 30-Jun-2014 17:27:17
BIS: 97   SR: 0   EMG: 45   SQI: 33    Hora: 30-Jun-2014 17:27:22
BIS: 97   SR: 0   EMG: 43   SQI: 39    Hora: 30-Jun-2014 17:27:27
BIS: 97   SR: 0   EMG: 44   SQI: 47    Hora: 30-Jun-2014 17:27:32
BIS: 96   SR: 0   EMG: 48   SQI: 53    Hora: 30-Jun-2014 17:27:37
```

Figura 58. Fragmento de fichero Resultado.txt generado tras la monitorización.

```
1  Nombre:  Jose Manuel González Cava
2  Edad:  21
3  Peso:  66
4  Altura: 182
5   EEG: -4299    Tiempo: 0.000000
6  EEG: -831    Tiempo: 0.007813
7  EEG: -4447    Tiempo: 0.015625
8  EEG: -1871    Tiempo: 0.023438
9  EEG: -1836    Tiempo: 0.031250
10 EEG: -4822    Tiempo: 0.039063
11 EEG: -748    Tiempo: 0.046875
12 EEG: -3865    Tiempo: 0.054688
13 EEG: -2526    Tiempo: 0.062500
14 EEG: -1345    Tiempo: 0.070313
15 EEG: -5018    Tiempo: 0.078125
16 EEG: -781    Tiempo: 0.085938
17 EEG: -3310    Tiempo: 0.093750
```

Figura 59. Fragmento de fichero EEG.txt generado tras la monitorización.

En definitiva, del resultado de la comparación entre los datos mostrados por el monitor y los datos adquiridos a través de nuestra aplicación, podemos determinar la validez del desarrollo de este trabajo fin de grado.

6.Conclusions and future research

6.1. Conclusions

The main objective of this project is related to the development of a software application to monitor the electroencephalographic activity during the anesthetic process. Capturing this information will allow the research team to perform offline studies of brain activity.

As a first objective, we have been able to acquire processed results from BIS VISTA monitor through serial port, according to binary protocol given by the technical specifications. To do this, we studied the different layers that data packets are formed by. Then, we were able to design and send messages to start receiving information from BIS monitor about variables and data we were interested in.

After processing data packets, it was possible to get information about EEG and some variables related to brain activity. Finally, through the development of this software, we have obtained a user interface that is capable to show the EEG waveforms in a real time process based on BIS VISTA monitor data.

As a second objective we capture some processed variables such as bispectral index (BIS), suppression ratio (SR), signal quality index (SQI) or electromyogram (EEM). After comparing these results with monitor display values, we can conclude that our software is a reliable tool that can be helpful for anesthesic monitoring process. Besides, designing a tool that joins the main functions of BIS monitor in a single screen simplifies the anesthetists tasks and allows to focus on the main process variables.

To sum up, a software that will be useful to study the influence of pain in electroencephalogram has been created. Saving data of EEG during surgical interventions will let the researches of ULL analyze if there is a relation between pain suffered by patients and EEG waveforms.

6.2. Future research

This project has some aspects that can be improved or even used as starting point of other projects related to anesthetic process control.

- First task that would be interesting is the integration of this software in the main software program developed by the Control Engineering Group of university of La Laguna.
- Absolute values of raw EEG data obtained through serial port are not the same as data used by the monitor to make the representation. It could be possible to study the relation among both values to look for equivalence, improving the graph in the user interface.
- An interesting research line that can be proposed is the analysis of the EEG to find alternative to BIS index to measure the hypnosis level of patient.
- Studying values of raw EEG to obtain a relation between pain and electroencephalographic activity. The result of this study could be useful to implement a mechanism of control to supply analgesic during surgery process. Finally, the whole information could be added to the original software.

Anexo 1: Especificaciones técnicas del puerto serial

Appendix A Structures and Definitions

```
#define L1_DATA_PACKET   1 /* Layer 1 directive for data/command packets */
#define L1_ACK_PACKET    2 /* Layer 1 directive for ACK packets */
#define L1_NAK_PACKET    3 /* Layer 1 directive for NAK packets */

#define SIZE_OF_HOST_POWER_SPECTRUM 60 /* 60 values for 0.5 hz - 30.0 hz */
#define SPECTRA_NUMOFCHAN    2    /* Number of spectra per message */
#define SIZE_M_SPECTRA ((SIZE_OF_HOST_POWER_SPECTRUM + 1)*4)
#define SIZE_M_PROCESSED_VARS    120
#define SIZE_M_PROCESSED_VARS_AND_SPECTRA  (SIZE_M_PROCESSED_VARS + SIZE_M_SPECTRA)
#define SIZE_M_PROCESSED_VARS_WITH_EXTRA_VARS  156
#define SIZE_M_PROCESSED_VARS_AND_SPECTRA_WITH_EXTRA_VARS
       (SIZE_M_PROCESSED_VARS_WITH_EXTRA_VARS + SIZE_M_SPECTRA)
#define SIZE_M_PROCESSED_VARS_LABELS    120
#define SIZE_M_PROCESSED_EXTRA_VARS_LABELS     192
#define PERCENT_DIVISOR 10
#define DB_POWER_DIVISOR 100
#define FREQUENCY_DIVISOR 100
#define IMPED_MSG_DATA_LEN 49     /* Lengths of message-dependent data in bytes */
#define ERROR_MSG_DATA_LEN 62
#define REVISION_INFO_DATA_LEN 28
#define EEG_SNIPPET_HEADER_DATA_LEN 28
#define EEG_SNIPPET_PROCESSED_DATA_LEN 40
#define EEG_SNIPPET_PROCESSED_DATA_WITH_EXTRA_VARS_LEN 64
#define BIS_HISTORY_HEADER_DATA_LEN 24
#define MAX_BIS_HISTORY_DATA_LEN 1020
#define MAX_BIS_HISTORY_DATA_RECORDS 85 /* Max records per big History message */
#define EVENT_MSG_DATA_LEN 31

#define START_CASE_RECORD 1
#define DATA_RECORD 2
#define EVENT_RECORD 3
#define CLOCK_ADJUST_RECORD 5
#define TIME_RECORD 6



struct processed_vars_and_spectra_msg
{
  struct processed_vars_msg processed_vars;
  struct spectra_msg spectra;
};


struct processed_vars_and_spectra_with_extra_vars_msg
{
  struct processed_vars_with_extra_vars_msg processed_vars_extra;
  struct spectra_msg spectra;
};


struct processed_vars_msg
{
   struct dsc_info_struct proc_dsc_info;  /* Info. on dsc, pic and selftest */

   /* Impedance info for 2 channels. */
   struct impedance_info_struct impedance_info[2];

   /* Host filter settings:
      Byte 0 (LSByte) - High pass filter setting
                        0 - 0.25 Hz
                        1 - 1.00 Hz
                        2 - 2.00 Hz
```

```
                        3 - 2.5 Hz
      Byte 1 - Low pass filter setting
                        0 - none
                        1 - 30 Hz
                        2 - 50 Hz
                        3 - 70 Hz
      Byte 2 - Notch filter setting
                        0 - none
                        1 - 50 Hz
                        2 - 60 Hz
                        3 - 50 & 60 Hz */
   unsigned long host_filt_setting;

   /* Spectral and Bispectral smoothing rates:
      Byte 0 (LSByte) - Spectral smoothing rate
                        0 - 0 seconds
                        1 - 5 seconds
                        2 - 10 seconds (default)
                        3 - 30 seconds
                        4 - 60 seconds
      Byte 1          - Bispectral smoothing rate
                        0 - 15 seconds (default)
                        1 - 30 seconds
                        2 - 60 seconds
                        3 - 10 seconds */
   unsigned long host_smoothing_setting;

   /* Spectral and Bispectral artifact detection masks. */
   unsigned long host_spectral_art_mask;
   unsigned long host_bispectral_art_mask;

   /*-------------------------------------------------------------+
   |  trend variables for 3 channels (ch1, ch2 and ch12 combined)  |
   +-------------------------------------------------------------*/

   struct be_trend_variables_info trend_variables[3];
};


struct processed_vars_with_extra_vars_msg
{
   struct dsc_info_struct proc_dsc_info;  /* Info. on dsc, pic and selftest */

   /* Impedance info for 2 channels. */
   struct impedance_info_struct impedance_info[2];

   /* Host filter settings:
      Byte 0 (LSByte) - High pass filter setting
                        0 - 0.25 Hz
                        1 - 1.00 Hz
                        2 - 2.00 Hz
                        3 - 2.5 Hz
      Byte 1 - Low pass filter setting
                        0 - none
                        1 - 30 Hz
                        2 - 50 Hz
                        3 - 70 Hz
      Byte 2 - Notch filter setting
                        0 - none
                        1 - 50 Hz
                        2 - 60 Hz
                        3 - 50 & 60 Hz */
   unsigned long host_filt_setting;

   /* Spectral and Bispectral smoothing rates:
      Byte 0 (LSByte) - Spectral smoothing rate
```

```
                        0 - 0 seconds
                        1 - 5 seconds
                        2 - 10 seconds (default)
                        3 - 30 seconds
                        4 - 60 seconds
      Byte 1          - Bispectral smoothing rate
                        0 - 15 seconds (default)
                        1 - 30 seconds
                        2 - 60 seconds
                        3 - 10 seconds */
   unsigned long host_smoothing_setting;

   /* Spectral and Bispectral artifact detection masks. */
   unsigned long host_spectral_art_mask;
   unsigned long host_bispectral_art_mask;

   /*-------------------------------------------------------------+
   |  trend variables for 3 channels (ch1, ch2 and ch12 combined)  |
   |  with extra variables                                         |
   +-------------------------------------------------------------*/

   struct be_trend_variables_extra_info trend_variables_extra[3];
};


struct dsc_info_struct
{
  unsigned char dsc_id;          /* DSC ID from status nibble 1 */
  unsigned char dsc_id_legal;    /* Flag when non-zero indicates that a legal
                                    dsc is connected */
  unsigned char pic_id;          /* (Sensor Type * 10)+ PIC ID */

#define SINGLE_CHANNEL_OR_SENSOR_TYPE                1      /* Sensor Plus */
#define DUAL_CHANNEL_OR_SENSOR_TYPE                  2      /* Quatro */
#define PEDIATRIC_OR_SENSOR_TYPE                     3      /* Pediatric */
#define DUAL_CHANNEL_ICU_SENSOR_TYPE                 4      /* Extend */
#define DEMO_SENSOR_TYPE                             5      /* Demo (Plus) */
#define PEDIATRIC_XP_SENSOR_TYPE                     7      /* Pediatric XP */
#define SINGLE_CHANNEL_SENSOR_SIMULATOR              8      /* Plus Simulator */
#define DUAL_CHANNEL_SENSOR_SIMULATOR                9      /* Quatro Sim. */
#define SEMI_REUSABLE_SENSOR_TYPE                    12     /* SRS  Type 1*/
#define EXTEND_DEMO_SENSOR_TYPE                      13     /* Demo (Extend) */
#define BILATERAL_SENSOR_TYPE                        14     /* Bilateral */
#define BILATERAL_SENSOR_TYPE_1                      16     /* new Bilateral */
#define FOUR_CHANNEL_SENSOR_SIMULATOR                17     /* 4 channel Sim. */

  unsigned char pic_id_legal;    /* if non-zero, a legal pic is connected. */
  unsigned short dsc_numofchan;  /* Number of channels on the DSC connected.
                                    Valid only when dsc_id is legal. */
  unsigned short quick_test_result; /* If zero, test passed.
                                       If non-zero, the bit fields in the
                                         result indicate which test(s) failed. */

#define QUICK_SELFTEST_PASS   0
#define QUICK_GAIN_TEST_BIT   0x1   /* Set when avg noise test fails. */
#define QUICK_NOISE_TEST_BIT  0x2   /* Set when blocked gain test fails. */
#define QUICK_TEST_FAIL_BIT   0x4   /* Set for timeout, DSC disconnect and
                                       other failures during the test .*/
#define QUICK_TEST_RESULT_VALID_BIT 0x8  /* Set when the quick test has not
                                       been run at all. Cleared when the test
                                       is done at least once. */


  /* DSC gain (in uV/ADC) = dsc_gain_num/dsc_gain_divisor */
  signed long dsc_gain_num;
  signed long dsc_gain_divisor;
```

```
  /* DSC offset (in uV/ADC) = dsc_offset_num/dsc_offset_divisor */
  signed long dsc_offset_num;
  signed long dsc_offset_divisor;
};


struct impedance_info_struct
{
   /*-----------------------------------------------------------+
   |  Impedance in units of 100 Ohms (or 1000 Ohms for ground  |
   |  impedances).                                             |
   +-----------------------------------------------------------*/
#define GND_SCALE       1000
#define NON_GND_SCALE   100

/* Limits for the impedance values */
#define MAX_GND_VALUE   100.0   /* in Kohms */
#define MAX_COM_VALUE   150.0   /* in 100 ohms */
#define MAX_POS_VALUE   75.0   /* in 100 ohms */

   unsigned short impedance_value; /* This is the value of the
                                        impedance for the channel */
   unsigned short imped_test_result; /* Result of the impedance test.
                                     If zero, test passed.
                                     If non-zero, the bit fields in the
                                      result indicate which test(s) failed. */

#define  IMPED_TEST_PASS         0    /* Test passed */
#define  IMPED_TEST_FAIL         0x1  /* High impedance  */
#define  IMPED_TEST_FAIL_CLIP    0x2  /* Noise */
#define  IMPED_TEST_FAIL_LEADOFF   0x4  /* Lead off */
};


struct be_trend_variables_info
{
   signed short burst_suppress_ratio; /* index variable giving percent of
                                         suppressed seconds in last 63 sec.
                                         for selected channel. range from
                                         0 - 1000 in .1% steps */
   signed short spectral_edge_95; /* in HZ ranged from 0-30.0 Hz in
                                    units of 0.01 Hz */
   signed short bis_bits; /* BIS field debug data */
   signed short bispectral_index;  /* Ranges from 0 - 100 */
   signed short bispectral_alternate_index;  /* same as above */
   signed short bispectral_alternate2_index;  /* same as above */
   signed short total_power; /* in dB with respect to .01 uV rms. ranged
                              from 0 to 100 dB in 0.01 units */
   signed short emg_low; /* in dB with respect to .01 uV rms. ranged
                          from 0 to 100 dB in 0.01 units */
   signed long bis_signal_quality; /* index variable giving the signal
                                      quality of the bisIndex which is
                                      combined with BSR 0 - 1000 in .1%
                                      steps */
   unsigned long second_artifact; /* bit field indicating TYPE OF ARTIFACT
                                     for the last second. */
};


struct be_trend_variables_extra_info
{
   signed short burst_suppress_ratio; /* index variable giving percent of
                                         suppressed seconds in last 63 sec.
                                         for selected channel. range from
```

```
                                        0 - 1000 in .1% steps */
   signed short spectral_edge_95; /* in HZ ranged from 0-30.0 Hz in
                                     units of 0.01 Hz */
   signed short bis_bits; /* BIS field debug data */
   signed short bispectral_index;  /* Ranges from 0 - 100 */
   signed short bispectral_alternate_index;  /* same as above */
   signed short bispectral_alternate2_index;  /* same as above */
   signed short total_power; /* in dB with respect to .01 uV rms. ranged
                                from 0 to 100 dB in 0.01 units */
   signed short emg_low; /* in dB with respect to .01 uV rms. ranged
                            from 0 to 100 dB in 0.01 units */
   signed long bis_signal_quality; /* index variable giving the signal
                                      quality of the bisIndex which is
                                      combined with BSR 0 - 1000 in .1%
                                      steps */
   unsigned long second_artifact; /* bit field indicating TYPE OF ARTIFACT
                                     for the last second. */
   signed short burst_per_min;    /* ranges from 0 to 30 */
   signed short rfu1;             /* reserved for future use */
   signed short rfu2;
   signed short rfu3;
   signed short rfu4;
   signed short rfu5;
};



struct spectra_msg
{
   unsigned short spect_numofchan;
   unsigned short spect_size;
   signed short power_spectrum[SPECTRA_NUMOFCHAN][SIZE_OF_HOST_POWER_SPECTRUM];
};


struct revision_info_msg
{
   char system_revision[4];       /* Binary bytes: first is major rev. number, */
   char host_revision[4];         /* second is minor rev. number, third is */
   char bis_eng_revision[4];      /* not defined, fourth is always 0. */
   char protocol_revision[4];      /* Displayed as major.minor */
   char boot_revision[4];
   char hardware_revision[4];
   char serial_number[4];         /* First 3 bytes make up a 24-bit unsigned         */
};                                /* integer (LSB first), last byte is an alphabetic */
                                  /* char (typ. 'A').                                */
                                   /* Displayed as char followed by integer.         */

struct be_trend_variables_labels
{
   char burst_suppress_ratio_label[12];        /* ASCII strings */
   char spectral_edge_95_label[12];
   char bis_bits_label[12];
   char bispectral_index_label[12];
   char bispectral_alternate_index_label[12];
   char bispectral_alternate2_index_label[12];
   char total_power_label[12];
   char emg_low_label[12];
   char bis_signal_quality_label[12];
   char second_artifact_label[12];
};


struct be_trend_variables_extra_labels
{
    char burst_suppress_ratio_label[12];       /* ASCII strings */
```

```
    char spectral_edge_95_label[12];
    char bis_bits_label[12];
    char bispectral_index_label[12];
    char bispectral_alternate_index_label[12];
    char bispectral_alternate2_index_label[12];
    char total_power_label[12];
    char emg_low_label[12];
    char bis_signal_quality_label[12];
    char second_artifact_label[12];
    char burst_per_min_label[12];
    char rfu1_label[12];
    char rfu2_label[12];
    char rfu3_label[12];
    char rfu4_label[12];
    char rfu5_label[12];
};


struct time_date
{
   short second;    /* second, 0 - 59 */
   short minute;    /* minute, 0 - 59 */
   short hour;      /* hour, 0 - 23 */
   short day;       /* day, 1 - 31 */
   short month;     /* month, 1 - 12 */
   short year;      /* 4-digit year (with century) */
};


struct snippet_info_msg
{
   struct time_date event_date_time;  /* Date/Time */
   char system_revision[4];   /* System revision number */
   char serial_number[4];     /* Monitor serial number */
   short num_raw_records;     /* Number of raw data records */
   short num_proc_records;    /* Number of processed data records */
   unsigned char dsc_id;      /* DSC ID */
   unsigned char pic_id;      /* PIC ID */
   short padding;             /* fills out last word */
};


struct snippet_processed_vars_msg
{
   struct snippet_trend_variables_info snippet_vars[2];
};


struct snippet_trend_variables_info
{
   short impedance_value;
   short burst_suppress_ratio;
   short bis_bits;
   short bispectral_index;
   short bispectral_alternate_index;
   short bispectral_alternate2_index;
   short emg_low;
   short bis_signal_quality;
   unsigned long second_artifact;
};


struct snippet_processed_extra_vars_msg
{
   struct snippet_trend_extra_variables_info snippet_vars[2];
};
```

```
struct snippet_trend_extra_variables_info
{
   short impedance_value;
   short burst_suppress_ratio;
   short bis_bits;
   short bispectral_index;
   short bispectral_alternate_index;
   short bispectral_alternate2_index;
   short emg_low;
   short bis_signal_quality;
   unsigned long second_artifact;
   short burst_per_min;
   short rfu_1;
   short rfu_2;
   short rfu_3;
   short rfu_4;
   short rfu_5;
};


struct history_info_msg
{
   struct time_date date_time;    /* Date/Time */
   char system_revision[4];       /* System revision number */
   char serial_number[4];         /* Monitor serial number */
   long num_records;              /* Number of data records */
};


struct history_data_msg
{
   unsigned long num_records;
   struct bh_record_struct data[MAX_BIS_HISTORY_DATA_RECORDS];
};

struct bh_record_struct
{
   union bh_record_body {
      struct bh_startcase_or_clockadjust start;
      struct bh_data data;
   } body;
};


struct bh_startcase_or_clockadjust
{
   unsigned char type;     /* Record type (start case = 1, clock adjust = 5) */
   unsigned char pad1;     /* padding */
   short pad2;             /* padding */
   long time;              /* Time of start case or new clock time, as utime */
   long case_id;           /* Case ID */
   long pad3;              /* padding */
};


struct bh_data
{
   unsigned char type;           /* Record type (data = 2) */
   unsigned char avg_bis;        /* Average BIS for 1 minute */
   unsigned char min_bis;        /* Minimum BIS for 1 minute */
   unsigned char max_bis;        /* Maximum BIS for 1 minute */
   unsigned char avg_bisalt;     /* Average BISALT for 1 minute */
   unsigned char avg_bisalt2;    /* Average BISALT2 for 1 minute */
   unsigned char avg_sqi;        /* Average SQI for 1 minute */
```

```
   unsigned char avg_emg;          /* Average EMG for 1 minute */
   unsigned char avg_sr;           /* Average SR for 1 minute */
   unsigned char avg_imped[2];     /* Average combined impedances for 1 minute */
   short bis_bits;                 /* BIS Bits from last update in minute */
   byte avg_burst_per_min;         /* Unused if extra vars have not been requested */
   byte reserved;                  /* Reserved for future use */
};
```

/* VISTA Binary Mode Structures */

```
#define SIZE_M_PROCESSED_VARS_4B    288
#define SIZE_M_PROCESSED_VARS_AND_SPECTRA_4B  (SIZE_M_PROCESSED_VARS_4B + SIZE_M_SPECTRA)


struct revision_info_msg
{                                  /* Binary bytes: first is major rev. number, */
                                   /* second is minor rev. number
                                   /* third and fourth are always 0.            */
                                   /* Displayed as major.minor                  */
   char system_revision[4];        /* VISTA Master Software Revision            */
   char host_revision[4];          /* VISTA Application Revision                */
   char bis_eng_revision[4];       /* BISx Software Revision                    */
   char protocol_revision[4];      /* Serial Protocol Revision                  */
   char boot_revision[4];          /* BISx Serial Number - see serial_number below
                                      BISx   last digit is 'B'
                                      BISx4  last digit is '4'                   */
   char hardware_revision[4];      // VISTA Hardware Revision
   char serial_number[4];          /* First 3 bytes make up a 24-bit unsigned */
                                   /* integer (LSB first), last byte is first alphabetic */
                                   /* char.  Displayed as char followed by integer. */
                                   /* e.g. VT01234 -> 0xD2 0x04 0x00 0x56 */
                                   /* VISTA: last byte is 'V'           */
};


struct be_bilateral_trend_variables_labels
{
  char burst_suppress_ratio_label[12];
  char spectral_edge_95_label[12];
  char spectral_edge_50_label[12];
  char bis_bits_label[12];
  char bispectral_index_label[12];
  char bispectral_alternate_index_label[12];
  char bispectral_alternate2_index_label[12];
  char total_power_label[12];
  char emg_low_label[12];
  char bis_signal_quality_label[12];
  char second_artifact_label[12];
  char burst_per_min_label[6];
  char reserved_label[6];
  char asym_label[12];
  char std_bis_label[12];
  char std_emg_label[12];
  char reserved_label_0[12];
  char reserved_label_1[12];
  char reserved_label_2[12];
 };


struct processed_vars_and_spectra_msg_4b
{
  struct processed_vars_msg_4b processed_vars;
  struct spectra_msg           spectra;
};


struct processed_vars_msg_4b
{
   struct dsc_bilateral_info_struct proc_dsc_info;  /* Info. on dsc, pic and selftest */

   /* Impedance info for up to 4 channels. */
   struct impedance_info_struct impedance_info[4];
```

```
   /* Host filter settings:
      Byte 0 (LSByte) - High pass filter setting
                        0 - 0.25 Hz
                        1 - 1.00 Hz
                        2 - 2.00 Hz
                        3 - 2.5 Hz
      Byte 1 - Low pass filter setting
                        0 - none
                        1 - 30 Hz
                        2 - 50 Hz
                        3 - 70 Hz
      Byte 2 - Notch filter setting
                        0 - none
                        1 - 50 Hz
                        2 - 60 Hz
                        3 - 50 & 60 Hz
      Byte 3 - Operating environment: OPERATING_ENV_OR or OPERATING_ENV_ICU
   */
   unsigned long host_filt_setting;

   /* Spectral and Bispectral smoothing rates:
      Byte 0 (LSByte) - Spectral smoothing rate
      Byte 1          - Bispectral smoothing rate
   */
   unsigned long host_smoothing_setting;
   /*
       Spectral smoothing rates    Bispectral smoothing rates
       Option   Rate (in sec)      Option   Rate (in sec)
       0        0                  0        15
       1        5                  1        30
       2        10                 2        60
       3        30                 3        10
       4        60
       */

# define MIN_SPECT_SMOOTHING_RATE  0
# define MAX_SPECT_SMOOTHING_RATE  4
# define MIN_BISPECT_SMOOTHING_RATE  0
# define MAX_BISPECT_SMOOTHING_RATE  3

   /* Spectral and Bispectral artifact detection masks.
      These are provided for validation purposes only and should be ignored
      during normal monitoring. */

   unsigned long host_spectral_art_mask;
   unsigned long host_bispectral_art_mask;

   /*-------------------------------------------------------------------+
   |  trend variables for up to 4 channels (ch1, ch2, ch3, ch4)
   +-------------------------------------------------------------------*/

   struct be_bilateral_trend_variables_info trend_variables[4];

   unsigned short sqi_left_index;  /* Index into trend_variables for selecting
                                      the 'bis_signal_quality' variable to use for
                                      sqi threshold comparisons.
                                      Possible values: 0 - 4 */
   unsigned short sqi_right_index; /* Index into trend_variables for selecting
                                      the 'bis_signal_quality' variable to use for
                                      sqi threshold comparisons.
                                      Possible values: 0 - 4 */
   unsigned short bis_left_index;   /* Index into trend_variables for selecting
                                      the 'bispectral_index' variable to display as BIS.
                                      Possible values: 0 - 4 */
   unsigned short bis_right_index; /* Index into trend_variables for selecting
```

```
                                    the 'bispectral_index' variable to display as BIS.
                                    Possible values: 0 - 4 */
 };


struct dsc_bilateral_info_struct
{
  unsigned char dsc_id;    /* DSC ID from status nibble 1 */

#define DSC_NONE_ID  0           /* No DSC connected */
#define DSC_BISX4_ID 13

       /* Only DSC IDs of DSC_NONE_ID or DSC_BISX4_ID are considered legal;
             For all other values, dsc_id_legal is set to zero to indicate
             illegal DSC ID. */

  unsigned char dsc_id_legal; /* Flag when non-zero indicates that a legal
                                 dsc is connected */
  unsigned char pic_id;       /* Sensor/pigtail ID */

#define PIC_NONE_ID     0
#define PIC_SS_ID       7

  /* Legal PIC IDs are:
     PIC_NONE_ID
     PIC_SS_ID + (Sensor Type * 10)
     For all other IDs, pic_id_legal is set to zero. */

  unsigned char pic_id_legal;    /* if non-zero, a legal pic is connected. */
  unsigned short dsc_numofchan;  /* Number of channels on the DSC connected.
                                    Valid only when dsc_id is legal. */
  unsigned short quick_test_result; /* If zero, test passed.
                                       If non-zero, the bit fields in the
                                       result indicate which test(s) failed. */

#define QUICK_SELFTEST_PASS   0
#define QUICK_GAIN_TEST_BIT   0x1   /* Set when avg noise test fails. */
#define QUICK_NOISE_TEST_BIT  0x2   /* Set when blocked gain test fails. */
#define QUICK_TEST_FAIL_BIT   0x4   /* Set for timeout, DSC disconnect and
                                       other failures during the test .*/
#define QUICK_TEST_RESULT_VALID_BIT 0x8  /* Set when the quick test has not
                                    been run at all. Cleared when the test
                                    is done atleast once. */

  unsigned short dsc_update_status;      /* Bit field indicating DSC update status. */
#define  DSC_UPDATE_DONE                      0
#define  DSC_UPDATE_IN_PROGRESS               0x1
#define  DSC_UPDATE_FAILED                    0x2
  unsigned short pad1;     // for alignment

  /* DSC gain (in uV/ADC) = dsc_gain_num/dsc_gain_divisor */
  signed long dsc_gain_num;
  signed long dsc_gain_divisor;

  /* DSC offset (in uV/ADC) = dsc_offset_num/dsc_offset_divisor */
  signed long dsc_offset_num;
  signed long dsc_offset_divisor;

  /****************/
  /* Sensor Info  */
  /****************/

  unsigned short sensor_status; /* high nibble is status nibble 5 */

#define SENSOR_VALID             0
#define SENSOR_INVALID                 1
```

```
#define TOO_MANY_USES            2
#define SENSOR_EXPIRED                3

        /* Nibble-5 status              Bit # */
#define SENSOR_POWERED_OFF       12
#define SENSOR_OVERCURRENT_BIT   13
#define SENSOR_POS_GND_FAULT_BIT 14
#define SENSOR_NEG_GND_FAULT_BIT 15

#define SENSOR_UNSUPPORTED           0x20     /* Bit 5: Unsupported sensor */
#define SRS_ELECTRODES_NOT_CONNECTED 0x40     /* Bit 6: 0 if electrodes
                                                        detected, 1 if not
                                                        detected. */
#define NEW_VALIDITY_UNKNOWN         0x3f
#define EXTEND_TYPE_SENSOR           0x200    /* Bit 9: Sensor activates
                                                  Burst Count features */
#define SENSOR_SIMULATOR             0x400    /* Bit 10: BIS values are blanked */
#define DEMO_DEVICE                  0x800    /* Bit 11: Must display Demo
                                                  Device message */


#define NEW_SS_BITS_MASK   (0xffff080 | SRS_ELECTRODES_NOT_CONNECTED |\
                            SRS_TYPE_SENSOR |\
                            EXTEND_TYPE_SENSOR |\
                            SENSOR_SIMULATOR |\
                            DEMO_DEVICE)
   struct bilateral_sensor_description_struct sensor_desc;
   unsigned short pad2;    // for alignment
   unsigned long lot_code;
   unsigned short shelf_life;
   unsigned short serial_number;
   unsigned long usage_count;
};


struct bilateral_sensor_description_struct
{
  unsigned char sensor_name[12];
  /* 12 character alpabetic string uniquely identifying the sensor. */

  unsigned char sensor_type;     /* Unique sensor identifier. */

#define BILATERAL_SENSOR_TYPE                   14  /* Bilateral */

  unsigned char sensor_graphic_type;
  /* Number of channels to display during impedance checking.
        If sensor_graphic_type is 6, impedance values should be checked for
     channel 1,channel 2, channel 3 and channel 4.
  */

  unsigned char sensor_eeg_channels;
  /* Number of channels of filtered EEG to be displayed. */

  unsigned char eeg_left_display_index;
  unsigned char eeg_right_display_index;
  /* This index indicates the channel to be displayed. */

  unsigned char sensor_sqi_channels;
  /* Number of channels of SQI values to be displayed. */

  unsigned char sqi_left_display_index;
  unsigned char sqi_right_display_index;
  /* Use this as an index into the
     channel data structure with sqi value. */

  unsigned char sensor_emg_channels;
```

```
  /* Number of channels of EMG values to be displayed. */

  unsigned char emg_left_display_index;
  unsigned char emg_right_display_index;

  unsigned char pad1;
  /* Use this as an index into the
     channel data structure with emg value. */

  unsigned char case_id[4];
  /* Alphanumeric characters representing case ID formed by combining
     sensor serial number and BIS Engine serial number.
     Display format: xxxx
  */
};


struct be_bilateral_trend_variables_info
{
  signed short burst_suppress_ratio; /* index variable giving percent of
                                        suppressed seconds in last 63 sec.
                                        for selected channel. range from
                                        0 - 1000 in .1% steps */
  signed short spectral_edge_95; /* in HZ ranged from 0-30.0 Hz in
                                    units of 0.01 Hz */
  signed short spectral_edge_50; /* in HZ ranged from 0-30.0 Hz in
                                    units of 0.01 Hz */
  signed short bis_bits; /* BIS field debug data */
  signed short bispectral_index;  /* Ranges from 0 - 100 */
  signed short bispectral_alternate_index;  /* same as above */
  signed short bispectral_alternate2_index;  /* same as above */
  signed short total_power; /* in dB with respect to .01 uv rms. ranged
                               from 0 to 100 dB in 0.01 units */
  signed short emg_low; /* in dB with respect to .01 uv rms. ranged
                           from 0 to 100 dB in 0.01 units */
  unsigned short pad1;     /* for alignment */
  signed long bis_signal_quality; /* index variable giving the signal
                                     quality of the bisIndex which is
                                     combined with BSR 0 - 1000 in .1%
                                     steps */
  unsigned long second_artifact; /* bit field indicating TYPE OF ARTIFACT
                                    for the last second. */

  unsigned char burst_count;      /* bursts/minute 0 - 30 */
#define BURST_COUNT_MASK   0x3f   /* mask for burst count value */
#define BURST_COUNT_BLANK  0x40   /* burst count blanked if this bit is set */
#define BURST_COUNT_ENABLE 0x80   /* burst count enabled if this bit is set */
  unsigned char reserved_byte_var;      /* reserved*/
  signed short asym_index;
  signed short std_bis;                 /* standard deviation of BIS */
  signed short std_emg;                 /* standard deviation of EMG */
  signed short reserved_short_var_0;
  signed short reserved_short_var_1;
  signed short reserved_short_var_2;
  unsigned short pad2;                  /* for alignment */
};
```

Appendix B Description of Processed EEG Variables

Variable Name	Description	Range	Divisor
Suppression Ratio (SR)	The percentage of epochs in the past 63 seconds in which the EEG signal is considered suppressed.	0.0 - 100.0 %	10
Burst Count (BURST)[4]	An alternate means of measuring suppression: number of EEG bursts in the last 60 seconds. Enabled only by the Extend (A-2000, VISTA Only) and Bilateral (VISTA) sensors.	0 – 30 bursts/minute	1
Spectral Edge Frequency (SEF)	The frequency at which 95% of the total power lies below it and 5% lies above it.	0.50 - 30.00 Hz	100
BIS bits (BISBIT)	These bits indicate the state of the Bispectral Index algorithm. (Aspect use only.)		n.a.
Bispectral Index (BIS) (displayed on screen as BIS; transmitted as B34… or DB1…)	This variable gives a bispectral index.	0.0 - 100.0	10
Bispectral Alternate Index (BISALT)	This variable gives an alternate bispectral index. (Aspect use only – not in ASCII data records.)	-3276.7 (0.0 - 100.0 when enabled)	10
Bispectral Alternate 2 Index (BISAL2)	This variable gives another alternate bispectral index. (Aspect use only – not in ASCII data records.)	-3276.7 (0.0 - 100.0 when enabled)	10
Total Power (TOTPOW)	A measure of the absolute total power in the frequency range from 0.5 to 30.0 Hz. Value is in dB with respect to 0.0001 μV^2.	40.00 - 100.00 dB	100
EMG Low (EMGLOW)	The absolute power in the 70 - 110 Hz range. Value is in dB with respect to 0.0001 μV^2.	30.00 - 80.00 dB	100
Signal Quality Index (SQI)	The percentage of good epochs and suppressed epochs in the last 120 (61.5 seconds) that could be used in the Bispectral Index calculation.	0.0 - 100.0 %	10

[4] BURST in the bilateral M_PROCESSED_VARS_4B and M_PROCESSED_VARS_4B_AND_SPECTRA messages includes status bits which must be checked. See struct be_bilateral_trend_variables_info in Appendix A for details.

Impedance (IMPEDNCE)	The combined impedance (sum of signal and reference electrode impedances) for the last second.	0 - 1000 Kohms	1
Artifact Flags (ARTF2)	See below.	See below.	n.a.
sBIS (VISTA Bilateral Only)	A BIS variability index calculated for each side of the head. The index represents the standard deviation of the BIS variable over the last minute.	0.0 to 10.0 BIS units	10
sEMG (VISTA Bilateral Only)	An EMG variability index calculated for each side of the head. The index represents the standard deviation of the EMG values over the last minute.	0.0 to 10.0 dB with respect to 0.001 μV^2	10
RESVAR0	Reserved data field	0.0	10
ASYM (VISTA Bilateral Only)	The percentage of EEG power present in left or right hemispheres. Calculated by BISx4 as: (Total Power Left / Total Power Left + Right) *1000 Displayed by VISTA as: ASYM > 500: xxL where xx = $((ASYM_{BISx4} / 10) - 50) * 2$ ASYM < 500: xxR where xx = $(50 – (ASYM_{BISx4} / 10)) * 2$	ASCII Protocol: 0.0 to 100.0, where 0.0 is 100% right, 100.0 is 100% left, 50.0 is equal power. Binary Protocol: 0 to 1000, where 0 is 100% right, 1000 is 100% left, 500 is equal power.	10
BILBITS (VISTA Bilateral Only)	Supplemental Bilateral Bits Bit 0: 1 indicates Bilateral Sensor Bit 1: 0 indicates Left Hemisphere, 1 indicates Right Hemisphere	Hex Display	n.a.

The Processed EEG variables are transmitted as integers, and must be divided by the specified divisor before being displayed.

The Artifact Flags indicate the types of artifacts (if any) detected within the previous update period. These flags are bit-mapped as follows:

Anexo 2: Código de la clase bis_eeg

```
function  Actualizacion( BIS, handle1,handle2,handle3,handle4,fid )
% Función que permite actualizar los valores de las variables procesadas y
% volcarlos en el fichero fid.


s = ActualizaDatos(BIS);
set(handle1,'string',s(1));
set(handle2,'string',s(2));
set(handle3,'string',s(3));
set(handle4,'string',s(4));
fprintf(fid,'BIS: %i   SR: %i   EMG: %i   SQI: %i    Hora: %s \n',...
    s(1), s(2),s(3),s(4), datestr(now));
end
```

```
function s = ActualizaDatos( BIS )
%Función encargada de actualizar lso datos a motrar por pantalla cada 5
%segundos


    s(1) = BIS.IndBis(BIS.IndBis_Temp());
    s(2) = BIS.SR(BIS.SR_Temp());
    s(3) = BIS.EMG(BIS.EMG_Temp());
    s(4) = BIS.SQI(BIS.SQI_Temp());


end
```

```
function BIS = bis_eeg (fichero)
    %   Constructor de la clase bis_eeg

    %Cargamos la libería headerftd2xx
    loadlibrary ('ftd2xx', @headerftd2xx);

    %____DEFINICIÓN DE ATRIBUTOS____
    % Atributos referidos a los campos del paquete
    % MID: Message ID: Tipo de aplicación del paquete. 32 bits
    % SN: Sequence Number (0-65535). 16 bits
    % Longitud: N° de identificación de paquete de capa 3. 16 bits
    % MDD: Message Dependent Data. Variable
    % BMDD: Bytes del Message Dependent Data
    % RID: Routing ID. 32 bits
    % PSID: Packet Sequence ID. 16 bits
    % SPI: Start Packet Identifier. 16 bits
    atr_MID = 0;
    BIS.SN = 0;
    atr_Longitud = 0;
    atr_RID = 0;
    BIS.PSID = 0;
    atr_MDD = 0;
    atr_BMDD = 0;
    BIS.SPI = [hex2dec('ba'),hex2dec('ab')];
    % Atributos referidos a la creación del objeto serial
    BIS.serial = serial_ftdi;
    atr_puerto = 0;
    % Atributos referidos al procesamiento de datos
    atr_chan1 = [];
    atr_chan2 = [];
    atr_DatosLeidos = [];
    atr_numPaquete = 1;
    atr_RepChan1 = zeros(1,512);
    atr_RepChan2 = zeros(1,512);
    atr_IndBis_Temp = 0;
    atr_SR_Temp = 0;
    atr_EMG_Temp = 0;
    atr_SQI_Temp = 0;
    atr_IndBis = 0;
    atr_SR = 0;
    atr_EMG = 0;
    atr_SQI = 0;
    BIS.fichero = fichero;
    atr_tiempo = 0;

     %Creación del timer para la lectura y procesamiento de datos
    timer_serial = timer ('Name', 'timer_serial','StartDelay',...
       0.1, 'Period', 0.5, 'ExecutionMode', 'fixedSpacing','BusyMode',...
       'drop');

    BIS.timer_serial = timer_serial;


    %___ACCESORES A LOS ATRIBUTOS___
    BIS.MID = @f_MID;
```

```
BIS.Longitud = @f_Longitud;
BIS.RID = @f_RID;
BIS.MDD = @f_MDD;
BIS.BMDD = @f_BMDD;
BIS.puerto = @f_puerto
BIS.chan1 = @f_chan1;
BIS.chan2 = @f_chan2;
BIS.DatosLeidos = @f_DatosLeidos;
BIS.numPaquete = @f_numPaquete;
BIS.RepChan1 = @f_RepChan1;
BIS.RepChan2 = @f_RepChan2;
BIS.IndBis_Temp = @f_IndBis_Temp;
BIS.SR_Temp = @f_SR_Temp;
BIS.EMG_Temp = @f_EMG_Temp;
BIS.SQI_Temp = @f_SQI_Temp;
BIS.IndBis = @f_IndBis;
BIS.SR= @f_SR;
BIS.EMG = @f_EMG;
BIS.SQI = @f_SQI;
BIS.tiempo = @f_tiempo;

% Creación del objeto
BIS = class(BIS,'bis_eeg');

set (timer_serial, 'TimerFcn', @(o, e) ProcesaDatos (BIS));

%____DEFINCIOÓN DE FUNCIONES ACCESORAS___
function m = f_MID(m)
    switch nargin
        case 0
            m = atr_MID;
        case 1
            atr_MID = m;
    end
end

function l = f_Longitud(l)
    switch nargin
        case 0
            l = atr_Longitud;
        case 1
            atr_Longitud = l;
    end
end

function r = f_RID(r)
    switch nargin
        case 0
            r = atr_RID;
        case 1
            atr_RID = r;
    end
end

function m = f_MDD(m)
```

```
    switch nargin
        case 0
            m = atr_MDD;
        case 1
            atr_MDD = m;
    end
end

function b = f_BMDD(b)
    switch nargin
        case 0
            b = atr_BMDD;
        case 1
            atr_BMDD = b;
    end
end

function p = f_puerto(p)
    switch nargin
        case 0
            p = atr_puerto;
        case 1
            atr_puerto = p;
    end
end

function c = f_chan1(c)
    switch nargin
        case 0
            c = atr_chan1;
        case 1
            atr_chan1 = c;
    end
end

function c = f_chan2(c)
    switch nargin
        case 0
            c = atr_chan2;
        case 1
            atr_chan2 = c;
    end
end


function d = f_DatosLeidos(d)
    switch nargin
        case 0
            d = atr_DatosLeidos;
        case 1
            atr_DatosLeidos = d;
    end
end

function n = f_numPaquete(n)
```

```
    switch nargin
        case 0
            n = atr_numPaquete;
        case 1
            atr_numPaquete = n;
    end
end

function r = f_RepChan1(r)
    switch nargin
        case 0
            r = atr_RepChan1;
        case 1
            atr_RepChan1 = r;
    end
end

function r = f_RepChan2(r)
    switch nargin
        case 0
            r = atr_RepChan2;
        case 1
            atr_RepChan2 = r;
    end
end

function i = f_IndBis_Temp(i)
    switch nargin
        case 0
            i = atr_IndBis_Temp;
        case 1
            atr_IndBis_Temp = i;
    end
end

function i = f_IndBis(i)
    switch nargin
        case 0
            i = atr_IndBis;
        case 1
            atr_IndBis = i;
    end
end

function s = f_SR_Temp(s)
    switch nargin
        case 0
            s = atr_SR_Temp;
        case 1
            atr_SR_Temp = s;
    end
end

function s = f_SR(s)
    switch nargin
```

```
            case 0
                s = atr_SR;
            case 1
                atr_SR = s;
        end
    end

    function e = f_EMG_Temp(e)
        switch nargin
            case 0
                e = atr_EMG_Temp;
            case 1
                atr_EMG_Temp = e;
        end
    end

    function e = f_EMG(e)
        switch nargin
            case 0
                e = atr_EMG;
            case 1
                atr_EMG = e;
        end
    end


    function s = f_SQI_Temp(s)
        switch nargin
            case 0
                s = atr_SQI_Temp;
            case 1
                atr_SQI_Temp = s;
        end
    end

    function s = f_SQI(s)
        switch nargin
            case 0
                s = atr_SQI;
            case 1
                atr_SQI= s;
        end
    end
    function t = f_tiempo(t)
        switch nargin
            case 0
                t = atr_tiempo;
            case 1
                atr_tiempo= t;
        end
    end

end
```

```
function  Comienzo( BIS )
% Función que envía los mensajes para la recepción de datos
    seleccion_EEG(BIS);
    seleccion_pvars(BIS);

end
```

```
function r = conectado_puerto (BIS)
% Función que comprueba si un puerto en concreto ya está conectado
        r = conectado_puerto (BIS.serial());

end
```

```
function conectar (BIS,puerto)
 % Función que permite conectar, a través del número de serie, el monitor
 % con el ordenador con ayuda de los métodos del objeto serial.
    if nargin < 2
        puerto = BIS.puerto()
    end

    BIS.puerto(puerto);

    % Primero fuerza la desconexión
    if conectado_puerto (BIS)
        desconectar (BIS);
    end


    %Conectamos el dispositivo
    conectar (BIS.serial, BIS.puerto());



end
```

```
function desconectar (BIS)
% Función que permite la desconexión del puerto serial y su liberalización.
% Asimismo, reinicia todos los atributos de la clase que hayan podido
% ser modificados durante el proceso para una posterior conexión. Para el
% timer_serial

    stop (BIS.timer_serial);
    disp ('Desconectando BIS');

    desconectar(BIS.serial());

    %Inicializamos los parámetros
    Leer(BIS);
    BIS.DatosLeidos([]);
    BIS.chan1([]);
    BIS.chan2([]);
    BIS.numPaquete(1);
    BIS.RepChan1(zeros(1,512));
    BIS.RepChan2(zeros(1,512));
    BIS.IndBis_Temp(0);
    BIS.SR_Temp (0);
    BIS.EMG_Temp(0);
    BIS.SQI_Temp(0);
    BIS.IndBis (0);
    BIS.SR= (0);
    BIS.EMG (0);
    BIS.SQI (0);


end
```

```
function Leer(BIS)
% Función que permite la lectura del buffer serial
    datos=leer(BIS.serial);
    BIS.DatosLeidos(datos);

end
```

```
function listar_dispositivos( BIS )
%Función que permite obtener el Handle y numero de serie del monitor BIS
%
    lista = listar_dispositivos(BIS.serial);
    BIS.puerto(lista(1).num_serie)

end
```

```
function Paquete( BIS )
    %Función que separa los datos leídos en paquete y obtiene el valor del
    %RAW_EEG y las variables procesadas
    chan1 = BIS.chan1([]);
    chan2 = BIS.chan2([]);
    datos_leidos = BIS.DatosLeidos();
    i = 2;
    k=1;
    m=[1,1,1];
    % Si se cumple las condiciones que aseguran que se ha almacenado un
    % paquete completo, se procesa. En caso contrario, se almacena el
    % siguiente valor en el array auxiliar
    while i<=length(datos_leidos)
        if (datos_leidos(i-1)==186)&&(datos_leidos(i)==171)&&...
                (sum(formato(sum(m(3:length(m)-2)),16)==[m(length(m)-1),m(end)])==2)
            Valor_EEG(BIS,m);
            m=[datos_leidos(i-1),1,1];
            k=2;
            i =i+1;


        else
            m(k)=datos_leidos(i-1);
            i=i+1;
            k=k+1;


        end

    end
   % Para el caso del último valor leído del buffer serial
    m(k)=datos_leidos(end);
    if(sum(formato(sum(m(3:length(m)-2)),16)==[m(length(m)-1),m(end)])==2)
         Valor_EEG(BIS, m);


    end


end
```

```
function  Parada( BIS )
%Función que interrumpe la recepción de datos
    parar_EEG(BIS);
    parar_pvars(BIS);
    reinicio(BIS);

end
```

```
function parar_EEG(BIS)
%Mensaje para parar el envío del EEG
    MID = 112;
    SN = 0;
    Longitud = 0;
    RID = 4;
    PSID = 0;
    MDD = 0;
    BMDD = 0;
% Se actualizan los datos de los atributos
    BIS.MID(MID);
    BIS.Longitud(Longitud);
    BIS.RID(RID);
    BIS.MDD(MDD);
    BIS.BMDD(BMDD);
%Se envía el mensaje y se para el timer
    Send_M(BIS);
    stop (BIS.timer_serial);


end
```

```
function parar_pvars( BIS )
%Mensaje para parar el envío de var. procesadas
    MID = 116;
    SN = 0;
    Longitud = 0;
    RID = 4;
    PSID = 0;
    MDD = 0;
    BMDD = 0;
% Se actualizan los datos de los atributos
    BIS.MID(MID);
    BIS.Longitud(Longitud);
    BIS.RID(RID);
    BIS.MDD(MDD);
    BIS.BMDD(BMDD);
%Se envía el mensaje
    Send_M(BIS);

end
```

```
function ProcesaDatos( BIS )
    %Función encargada de leer los datos, procesarlos y preparar los
    %paquetes para la representación.
      Leer (BIS);
      Paquete(BIS);
      Representacion(BIS);



end
```

```
function reinicio(BIS)
% Función que permite reinicializar los valores de los atributos
    Leer(BIS);
    BIS.DatosLeidos([]);
    BIS.chan1([]);
    BIS.chan2([]);
    BIS.numPaquete(1);
    BIS.RepChan1(zeros(1,512));
    BIS.RepChan2(zeros(1,512));
    BIS.IndBis_Temp(0);
    BIS.SR_Temp (0);
    BIS.EMG_Temp(0);
    BIS.SQI_Temp(0);
    BIS.IndBis (0);
    BIS.SR= (0);
    BIS.EMG (0);
    BIS.SQI (0);

end
```

```
function Rep( BIS,h )
%Función encargada de la representación final
  t = (0:0.125/16:4-0.125/16);
  set(h,'YData',BIS.RepChan1());
  set(h,'XData',t);


end
```

```
function Representacion(BIS)
    %Función encargada de almacenar los vectores para la representación del
    %EEG cada 4 s.
%
     inicio =BIS.numPaquete();
     valor1 = BIS.chan1();
     representacion1 = BIS.RepChan1();
     valor2 = BIS.chan2();
     representacion2 = BIS.RepChan2();

     for indice =1:16:(length(valor1))
         representacion1(inicio:inicio+15) = valor1(indice:indice+15);
         BIS.RepChan1(representacion1);
         representacion2(inicio:inicio+15) = valor2(indice:indice+15);
         BIS.RepChan2(representacion2);
         inicio = inicio +16;
         if inicio >=512
             inicio = 1;
         end
     end
     BIS.numPaquete(inicio);


end
```

```
function  seleccion_EEG(BIS)
%Función que genera el mensaje para recepción de EEG
    MID = 111;
    SN = 0;
    Longitud = 2;
    RID = 4;
    PSID = 0;
    MDD = 128;
    BMDD = 16;
%Se actualizan los datos de los atributos
    BIS.MID(MID);
    BIS.Longitud(Longitud);
    BIS.RID(RID);
    BIS.MDD(MDD);
    BIS.BMDD(BMDD);
%Se envía el mensaje y comienza el timer
    Send_M(BIS);
    start (BIS.timer_serial);


end
```

```
function  seleccion_pvars( BIS )
%Función que genera el mensaje para la recepción de las variables
%procesadas
    MID = 115;
    SN = 0;
    Longitud = 1;
    RID = 4;
    PSID = 0;
    MDD = 0;
    BMDD = 8;
% Se actualizan los datos de los atributos
    BIS.MID(MID);
    BIS.Longitud(Longitud);
    BIS.RID(RID);
    BIS.MDD(MDD);
    BIS.BMDD(BMDD);
%Se envía el mensaje y comienza el timer
    Send_M(BIS);


end
```

```
function Send_M (BIS)
% Función encargada de confeccionar los mensajes con los valores de los
% campos almacenados en los atributos
    MID = formato(BIS.MID(),32);
    SN = formato(BIS.SN, 16);
    Longitud = formato (BIS.Longitud(), 16);
    RID = formato (BIS.RID(), 32);
    PSID = formato (BIS.PSID(), 16);

% Generación de las tres capas del mensaje.
    %Creación de la capa 3
    if BIS.MDD()~=0
        MDD = formato (BIS.MDD(), BIS.BMDD());
        Layer3 = [MID, SN, Longitud,MDD];
    else
        Layer3 = [MID, SN, Longitud];
    end
    % Creación de la capa 2
    Layer2 = [RID, Layer3];

    % Creación de la capa 3
    OD= formato (length(Layer2),16);
    LD = formato (1, 16);

    Layer1 = [PSID, OD, LD, Layer2];
    CheckSum = formato((sum(Layer1)),16);
    Layer1 = [BIS.SPI, Layer1, CheckSum];
% Envío del mensaje
    escribir (BIS.serial, Layer1);



end
```

Anexo 3: Código de la Interfaz de Usuario

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

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

% Last Modified by GUIDE v2.5 30-Jun-2014 10:25:35

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

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


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

x = inputdlg({'Nombre','Edad (años)','Peso (Kg)','Altura (cm)'},...
```

```
            'Introduzca los datos del paciente', [1 50; 1 15; 1 15;1 15]);
handles.x = x;
set(handles.text6,'string',handles.x(1));
set(handles.text7,'string',handles.x(2));
set(handles.text8,'string',handles.x(3));
set(handles.text9,'string',handles.x(4));
%Creamos el fichero de texto
    handles.fid=fopen('resultado.txt','w');
    fprintf(handles.fid, 'Nombre: %s \n',char(handles.x(1)));
    fprintf(handles.fid, 'Edad: %s \n',char(handles.x(2)));
    fprintf(handles.fid, 'Peso: %s\n',char(handles.x(3)));
    fprintf(handles.fid, 'Altura: %s\n ',char(handles.x(4)));

%Creamos el fichero de texto para el EEG
    handles.fid2=fopen('EEG.txt','w');
    fprintf(handles.fid2, 'Nombre: %s \n',char(handles.x(1)));
    fprintf(handles.fid2, 'Edad: %s \n',char(handles.x(2)));
    fprintf(handles.fid2, 'Peso: %s\n',char(handles.x(3)));
    fprintf(handles.fid2, 'Altura: %s\n ',char(handles.x(4)));

    BIS = bis_eeg (handles.fid2);
    handles.BIS = BIS;
%Creamos el Timer para la representación del EEG y modificamos los valores
%del gráfico

    handles.h =plot((0:0.125/16:4-0.125/16),zeros(1,512));
    set(handles.axes1,'YLim',[-4500,-1500]);
    set(handles.axes1,'YLimMode','manual');
    set(handles.axes1,'YTick',(-4500:1500:-1500))
    set(handles.axes1,'YTickLabel',['-' '-' '-']);
    set (handles.axes1,'XLim',[0,4]);

    timer_Rep = timer ('Name', 'timer_Rep','StartDelay',...
       0.1, 'Period', 0.5, 'ExecutionMode', 'fixedSpacing','BusyMode',...
       'drop');
    handles.timer_Rep = timer_Rep;
    set (timer_Rep, 'TimerFcn', @(o, e) Rep (handles.BIS,handles.h));


%Creamos el Timer para la representación de las variables procesadas
    timer_repvars = timer ('Name', 'timer_repvars','StartDelay',...
       0.1, 'Period', 5, 'ExecutionMode', 'fixedSpacing','BusyMode',...
       'drop');
       handles.timer_repvars = timer_repvars;
       set (timer_repvars, 'TimerFcn', @(o, e) Actualizacion (handles.BIS,...
           handles.text20,handles.text21,handles.text22,handles.text23,handles.fid));


% Choose default command line output for Aplicacion
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Aplicacion wait for user response (see UIRESUME)
```

```
% uiwait(handles.figure1);


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

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
listar_dispositivos(handles.BIS);
if ~conectado_puerto(handles.BIS)
    conectar(handles.BIS);
end
if conectado_puerto(handles.BIS)
    set(handles.text14,'string','Conectado');
end
guidata(hObject, handles);


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
desconectar(handles.BIS);
stop(handles.timer_repvars);
stop(handles.timer_Rep);
if ~conectado_puerto(handles.BIS)
    set(handles.text14,'string','Desconectado');
end

guidata(hObject, handles);


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if ~conectado_puerto (handles.BIS)
    errordlg('Debe conectar primero el monitor','ERROR')
elseif (strcmp(get(handles.timer_Rep,'Running'),'off'))
    Comienzo(handles.BIS);
    start(handles.timer_Rep);
    start(handles.timer_repvars);
```

```
else
    errordlg('Ya ha comenzado la monitorización','ATENCIÓN')
end
guidata(hObject, handles);



% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
  if ~conectado_puerto (handles.BIS)
    errordlg('Debe conectar primero el monitor','ERROR')
else
    Parada(handles.BIS);
    stop(handles.timer_repvars);
    stop(handles.timer_Rep);
    fclose(handles.fid)
    fclose(handles.fid2)
  end
```

Bibliografía

- Aspect Medical Systems. BIS VISTA Monitoring System. Operating Manual. 2008
- Aspect Medical Systems. BIS VISTA. Serial Port Technical Specification. 2008
- Félix Buisán, Nuria Ruiz; Grupo de Trabajo de la Sociedad Castellano-Leonesa de Anestesiología, Reanimación y Terapéutica del Dolor (SOCLARTD). Índice biespectral (BIS) para monitorización de la consciencia en anestesia y cuidados críticos: guía de práctica clínica. Valladolid: SOCLARTD; 2008
- Reboso Morales, Héctor Javier. Tesis Doctoral: Modelado y control del proceso anestésico mediante infusión de propofol y realimentación del índice biespectral. Universidad de La Laguna. 2012.
- Saboya Sánchez, S; Martín Vivas, Silvia Obregón, J.A.; Romera Ortega, M.Á.; Chamorro Jambrina, C; La Torre Marco, I.; Camarero Jorge, E. Monitorización de la sedación profunda. El monitor BIS. Enfermería Intensiva. Vol.20.Núm.04. Octubre-Diciembre 2009.
- Soler E, Faus M, Burguera R, Fernandez J, Mula. Anestesiología. Tomo II. Capitulo 2 (2002).
- Tutoriales de Matlab: http://www.mathworks.es