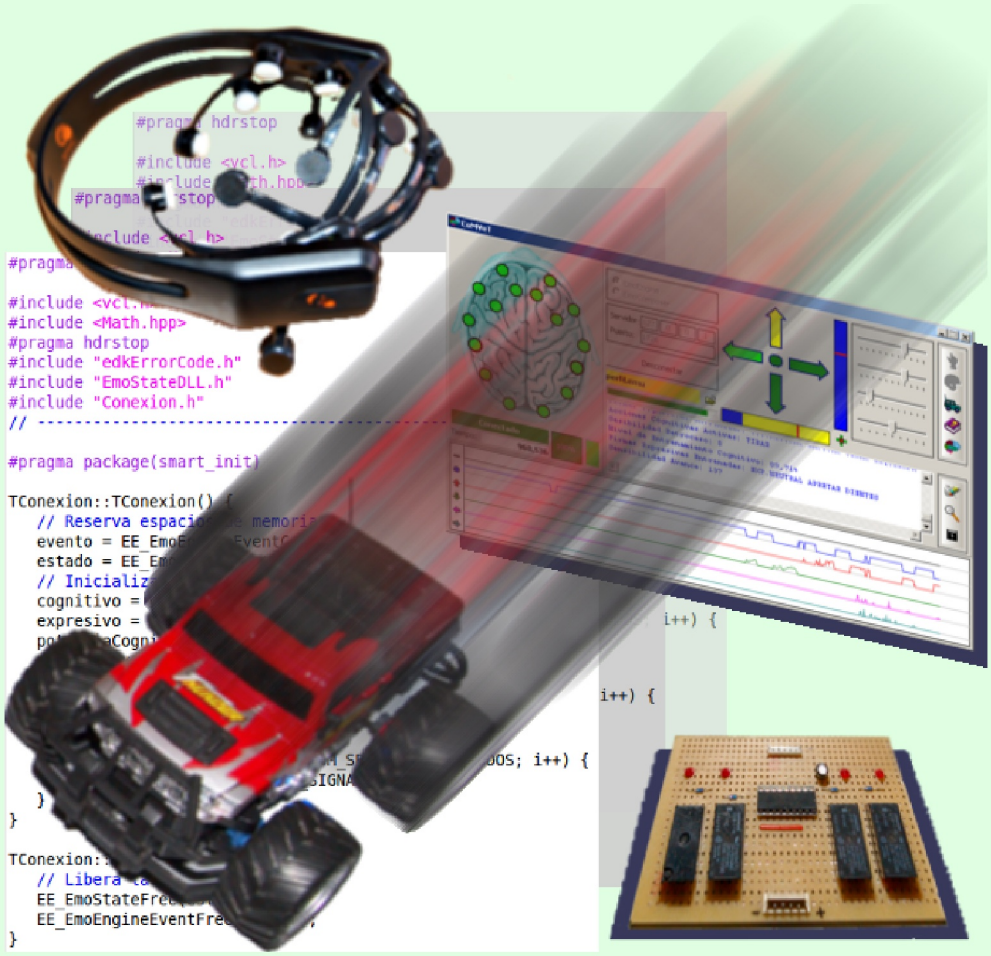
CoMVeT

Control Mental de Vehículos Teledirigidos



Daniel Héctor Stolfi Rosso

Sergio Gálvez Rojas

UNIVERSIDAD DE MÁLAGA

CC BY NC ND

CoMVeT

Control Mental de Vehículos Teledirigidos

Daniel Héctor Stolfi Rosso

Ingeniero en Informática

Sergio Gálvez Rojas

Doctor Ingeniero en Informática

Dpto. de Lenguajes y Ciencias de la Computación

E.T.S. de Ingeniería Informática

Universidad de Málaga

UNIVERSIDAD
DE MÁLAGA

CoMVeT – Control Mental de Vehículos Teledirigidos

Basado en el proyecto de fin de carrera de Daniel Héctor Stolfi Rosso, dirigido por el Dr. Sergio Gálvez Rojas

(cc) Reconocimiento-NoComercial-CompartirIgual, 2010

Los autores no poseen ninguna relación con Emotiv Systems, Inc.

AUTORES: Daniel Héctor Stolfi Rosso
Sergio Gálvez Rojas
PORTADA: Daniel Héctor Stolfi Rosso

DEPÓSITO LEGAL: MA-897-2011
ISBN: 978-84-694-3343-0

Índice de Contenido

Capítulo 1

Introducción

En este libro se aborda el estudio del ***Emotiv Beta SDK*** (Kit Beta de Desarrollo Software de Emotiv[1]) y de la **API**[2] contenida en el mismo para dialogar con el casco **Emotiv Epoc**, y como aplicación de este estudio se propone el movimiento mediante control mental de un vehículo teledirigido.

Para ello se desarrollará una aplicación capaz de utilizar dicha **API** y mediante la interpretación de las señales obtenidas controlar el puerto paralelo del PC en el cual se encontrará conectada una placa de circuito impreso, desarrollada también dentro del ámbito de este proyecto, que actuará sobre el mando de control remoto por radio frecuencia del vehículo teledirigido (Figura 1).

Figura 1: Esquema de funcionamiento del desarrollo realizado

Como se ha mencionado, primero se abordará la fase de estudio de la **API** para familiarizarse con las funciones disponibles, las posibilidades de desarrollo que ofrece y experimentar con los programas de ejemplo suministrados observando como interactúan con el casco **Emotiv Epoc**.

Posteriormente se abordará el desarrollo software, escogiendo el entorno de desarrollo y lenguaje de programación que mejor se ajusten al diseño realizado. Una vez que la aplicación se encuentre desarrollada, se procederá al diseño y montaje del circuito que se encargará de operar el mando del vehículo radio controlado a partir de las señales volcadas en el puerto paralelo.

Por último se realizará la integración del casco, API, software, hardware, mando y vehículo,

1 Todos los productos Emotiv pertenecen a Emotiv Systems, Inc.
2 *Application Programming Interface* - Interfaz de Programación de Aplicaciones.

comprobando el funcionamiento y realizando los ajustes y modificaciones necesarias para alcanzar el funcionamiento deseado, así como plantear posibles ampliaciones y mejoras.

El método de construcción consistirá en primer lugar en la toma de contacto con el software suministrado con el **Kit de Desarrollo**, realizando la instalación del mismo, poniendo en funcionamiento el casco y realizando las primeras pruebas de control mental.

Posteriormente se pasará al estudio de la **API** conocer las funciones suministradas, los tipos de datos y enumeraciones que se utilizan y las posibilidades de desarrollo que se proporcionan. Asimismo se realizarán modificaciones al código de los programas de ejemplo, para afianzar los conocimientos adquiridos.

En este punto se procederá a la elección del entorno de desarrollo así como el lenguaje de programación el cuál estará limitado a **C**, **C++** ó **C#** por tratarse de las opciones que ofrecen las bibliotecas Emotiv, indispensables para el desarrollo de este proyecto.

La aplicación constará de un modo de operación manual de forma de poder comprobar el funcionamiento de la interfaz y el vehículo radiocontrolado de forma independiente a las detecciones provenientes del casco. Además se podrá conectar desde la aplicación tanto con el casco a través del **Motor Emotiv** (**EmoEngine**) como con el programa emulador **EmoComposer**, el cual será de especial utilidad para realizar las tareas de depuración permitiendo simular los distintos eventos que produce normalmente el casco sin la necesidad de estar utilizándolo en todo momento.

Para presentar la información de monitorización del estado de la batería, contactos de los electrodos, etc, se utilizarán imágenes y elementos gráficos dentro de la interfaz de usuario así como los indicadores de avance, retroceso, giro a la izquierda, giro a la derecha y neutral.

El movimiento de avance tendrá asociado una acción del modo de expresivo, mientras que la acción de retroceso utilizará una del modo cognitivo. De esta manera se minimiza la interacción entre detecciones evitando los falsos positivos.

En cuanto a los movimientos de giro hacia la izquierda y derecha se utilizará la información provista por el acelerómetro del giróscopo que indica el movimiento de la cabeza en ambos sentidos.

Para visualizar la información de las detecciones a lo largo del tiempo se empleará una gráfica de líneas paralelas que junto con cada evento registrará la intensidad de las señales para las cuatro acciones antes descritas más un canal para el estado neutral y otro para el caso en que no

se realice ninguna detección.

Las acciones de avance y retroceso tendrán un control deslizable para ajustar el umbral a partir del cual se activará cada una atendiendo al valor de intensidad del evento. Un control similar estará destinado para el giro de la cabeza que determinará la activación de los giros, pero en este caso el parámetro a tener en cuenta será la brusquedad (aceleración) del movimiento realizado.

La carga de perfiles de usuario se realizará a través de un botón que presentará un diálogo para seleccionar el fichero a cargar y existirán dos controles deslizables para modificar las sensibilidades guardadas dentro del perfil en el momento de su entrenamiento. Estos entrenamientos se realizarán mediante las herramientas suministradas con el **Kit de Desarrollo Emotiv**.

Por último para presentar la información referente a la conexión, optimizaciones, recepción de eventos, etc, se dispondrá de un área de texto para el registro, el cual podrá volcarse a un fichero en disco para su posterior consulta.

Para controlar el puerto paralelo, será necesario utilizar una biblioteca *DLL*[3] externa ya que los núcleos basados en **Microsoft Windows NT**[4], como **Microsoft Windows XP**[5] y posteriores, no permiten el acceso al puerto paralelo a los programas en modo usuario.

Para el diseño del esquema y la placa de circuito impreso se utilizará el programa **KiCad**[6], el cual es *open source*, multiplataforma y compatible con las bibliotecas OrCAD[7], SPICE[8], etc. En primer lugar se dibujará el circuito definiendo una biblioteca propia en la que se añadirán los componentes que se emplearán para evitar que su desaparición en versiones posteriores afecte al proyecto.

De cada componente se apuntará en una tabla su referencia, fabricante, el código utilizado por el fabricante y el código del artículo en la conocida web de venta online **Farnell** [19]. Además se tomará nota del precio orientativo del componente a la fecha de la redacción de este ejemplar.

Luego se asociará cada componente con su huella en el circuito impreso para comenzar con el

3 *Dinamic-Link Library* – Biblioteca de enlace dinámico. Biblioteca de funciones que se carga sólo una vez en memoria pudiendo ser compartida por varios procesos.
4 Microsoft Windows NT es un producto de Microsoft Corporation.
5 Microsoft Windows XP es un producto de Microsoft Corporation.
6 KiCad es un software diseñado y escrito por Jean-Pierre Charras bajo la GNU GPL v:2.
7 OrCAD es un producto de Cadence Design Systems, Inc.
8 SPICE es un desarrollo de Laurence Nagel para el Laboratorio de Investigación Electrónica de la Universidad de California, Berkeley.

diseño de la placa. Primero se modificará la rejilla para que la distancia entre componentes se ajuste al modelo de placa perforada a utilizar, luego se definirá el contorno del PCB[9] para ajustarlo al espacio disponible dentro de la caja escogida y finalmente se ubicarán las huellas de los componentes uniéndolas con pistas según el diseño previo.

Por último se generarán los ficheros *Gerber*[10] y el fichero de perforaciones (*.drl*), útiles en caso de que la construcción de las placas corra a cargo de un tercero.

Una vez montada la placa se la situará en su caja y se soldarán los conectores. Luego se construirán los cables para la interconexión con el puerto paralelo y con el mando según el diseño realizado, para proceder a continuación a realizar las pruebas de integración entre el casco, el software, el hardware y el mando del vehículo teledirigido, validando así el desarrollo.

La planificación inicial del proyecto comienza con los primeros ensayos llevados a cabo dentro del estudio de viabilidad, continuando con la fase de análisis y obteniéndose como resultado la escritura del anteproyecto (Figura 2).

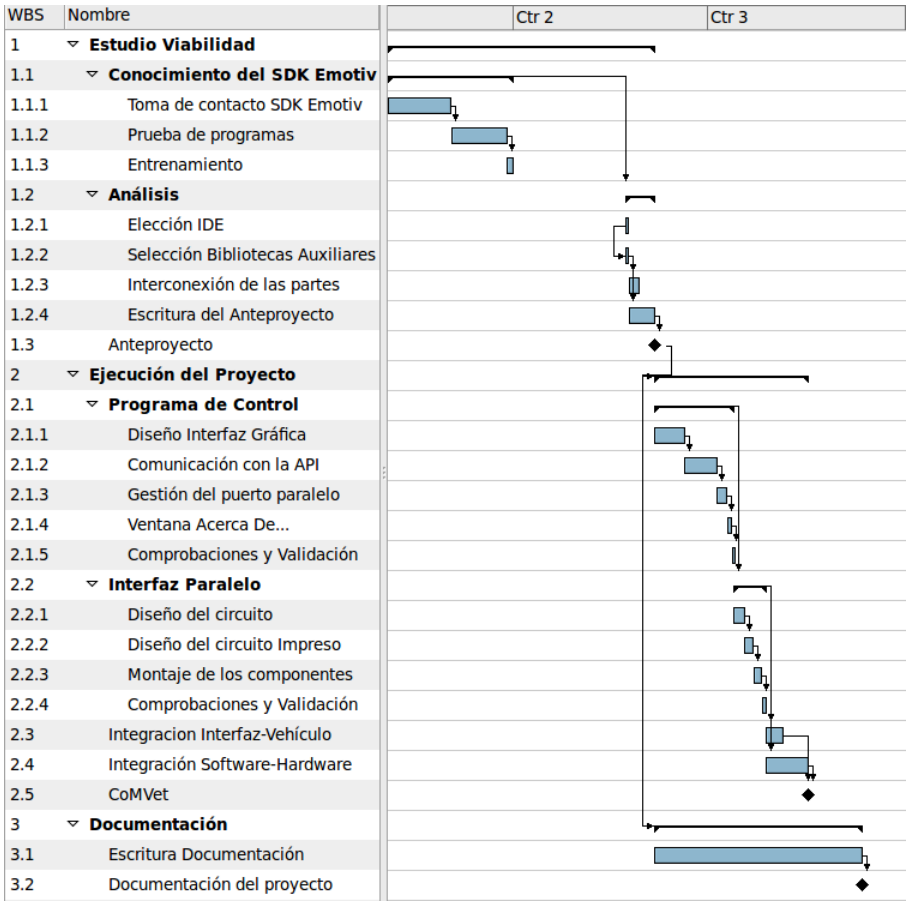


Figura 2: Planificación del proyecto

9 *Printed Circuit Board* – Circuito Impreso

10 Un fichero *Gerber* contiene la información necesaria para la fabricación de la placa de circuito impreso. El estándar más común hoy en día es el RS-274-X, aunque existen otros menos frecuentes. [11]

A continuación, habiéndose aprobado el anteproyecto, se comienza la ejecución del proyecto mediante el desarrollo del programa de control, la construcción de la interfaz hardware y la integración de las partes, obteniéndose como resultado el producto a construir.

Simultáneamente con la fase anterior se irá escribiendo la documentación que formará la base sobre la cual se ha confeccionado este documento. Véase en la Figura 79 en el Apéndice A para consultar la planificación con más detalle.

La estructura de este libro comienza con esta Introducción en donde se ha descrito el desarrollo que se va a realizar y la planificación a seguir, a continuación, en el epígrafe siguiente, se dará una noción sobre las técnicas de electroencefalografía relacionándolas con las características del casco **Emotiv Epoc**. Posteriormente se procederá a describir la instalación del Kit de Desarrollo Emotiv, los componentes y programas que incluye así como las funciones disponibles dentro de la API suministrada.

Luego, para comenzar con el trabajo se abordará el Desarrollo Software en donde se describirá el diseño de la aplicación mediante una serie de diagramas para continuar con el Desarrollo Hardware en donde se expondrá el diseño del circuito para la interfaz y la construcción y montaje de la placa de circuito impreso que lo implementará.

A continuación se comentará la integración de los componentes del desarrollo y las pruebas realizadas para su validación, sin olvidar las conclusiones y trabajo futuro.

Dentro de los apéndices se incluirán algunos diagramas con detalles extras los que, debido a su tamaño, no se han incluido dentro de los epígrafes anteriores. También se podrán encontrar aquí los listados con el contenido de los ficheros de cabeceras que representan los atributos y métodos pertenecientes a cada clase definida dentro del desarrollo software.

Por último se encuentran disponibles las referencias bibliográficas empleadas así como los enlaces a las respectivas páginas web en los casos en que se traten de recursos provenientes de Internet.

Capítulo 2

EEG y el casco Emotiv Epoc

El electroencefalograma es el registro de la actividad eléctrica generada por las neuronas en el interior del cerebro la cual se obtiene a través del cráneo por medio de electrodos situados en la superficie del cuero cabelludo.

La actividad eléctrica, caracterizada por salvas de ondas lentas sobre las que se superponen ritmos rápidos, tiene su origen en la actividad sináptica de las neuronas corticales las que se encuentran dispuestas en regiones determinadas de la superficie cortical[11]. Cada una de estas regiones capaz de producir actividad eléctrica se los denomina **generador**. Dado que en un registro normal se recoge actividad de muchos miles de neuronas, para poder conseguir una actividad mínima global es preciso que las neuronas vecinas se encuentren sincronizadas. Cuando esto ocurre, se pueden observar ondas tanto mayores y tanto mas lentas, cuanta mayor sea la sincronía de los generadores.

La captación de las señales bioeléctricas puede realizarse sobre el cuero cabelludo utilizando **electrodos superficiales**, en la base del cráneo a través de **electrodos basales**, en el cerebro expuesto o en localizaciones cerebrales profundas utilizando **electrodos quirúrgicos**. Dependiendo de la forma de captación el registro de señales toma el nombre de **electroencefalograma** (EEG) cuando se utilizan electrodos de superficie o basales, **electrocorticograma** (EcoG) para el caso que se utilicen electrodos quirúrgicos de superficie y **estéreo electroencefalograma** (E-EEG) cuando se trate de electrodos quirúrgicos de aplicación profunda.

En cuanto a los tipos de electrodos superficiales se distinguen los **adheridos** que consisten en pequeños discos metálicos que se fijan con pasta conductora dando resistencias de contacto muy bajas; **de contacto**, compuestos de pequeños tubos de plata clorurada roscados a soportes de plástico, que contienen en su extremo una almohadilla humedecida con una solución conductora las que se sujetan al cráneo con unas bandas elásticas y que se conectan mediante pinzas de cocodrilo; **en casco de malla**, similar al anterior tienen la ventaja de estar los electrodos sujetos a

11 La corteza cerebral es el manto de tejido nervioso que cubre la superficie de los hemisferios cerebrales, alcanzando su máximo desarrollo en los primates. [5]

un casco elástico de modo que resultan más cómodos para el paciente y presentan una gran precisión de colocación. Nótese que la amplitud de la señal de electroencefalograma típica de un adulto es de 10-20mV cuando se la mide con electrodos basales y de 10-100µV al realizar la medición sobre el cuero cabelludo.

Los electrodos del casco **Emotiv Epoc** se encuentran sujetos a unos brazos de plástico que garantizan la correcta ubicación, estando también dotados de una almohadilla humedecida en una solución salina para favorecer la conducción. Un factor extra de comodidad viene dado por el enlace inalámbrico con el PC de modo que el usuario sólo se encuentra limitado en su movilidad por el alcance de la señal de radio.

El sistema internacional de disposición de los electrodos 10-20 es el más utilizado actualmente (Figura 3).

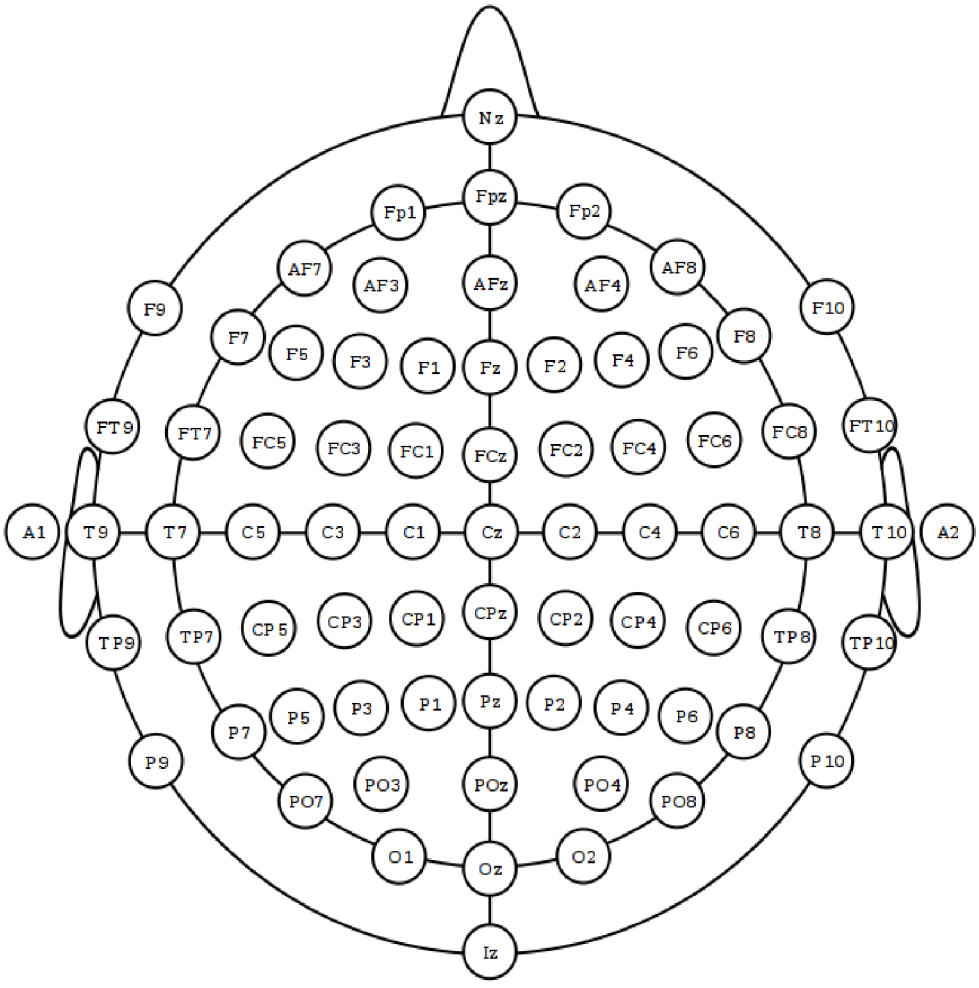


Figura 3: Sistema 10-20 - Ilustración de Marius 't Hart (cc)

Este método se desarrolló con el fin de garantizar una estandarización y repetibilidad de

manera que los estudios realizados a un individuo puedan ser comparados en el transcurso del tiempo y también para comparar los resultados obtenidos entre distintos individuos.

El sistema se basa en la relación entre la localización de un electrodo y el área del *cortex* cerebral subyacente. El "10" y "20" se refieren a la distancia entre dos electrodos adyacentes que es del 10% de la longitud del cráneo desde el nasión[12] hasta el inión[13] ó del 20% de la distancia entre ambos puntos preauriculares[14] pasando por el vertex[15].

Nótese que la letra empleada para el nombre de los puntos de contacto identifica al lóbulo y el número, a la ubicación dentro del hemisferio. Siendo las letras F, T, C, P y O la iniciales de frontal, temporal, central, parietal y occipital respectivamente (La letra C se utiliza para identificar la línea horizontal central y no hace referencia a ningún lóbulo). Los números pares se corresponden con electrodos en el hemisferio derecho y los impares con los del izquierdo. Los subíndices z (*zero* - cero) se utilizan para identificar la línea vertical central de electrodos.

Como para poder obtener un registro es necesario tener de una señal y una referencia, existen varios métodos de conexión de los electrodos dependiendo de los canales disponibles y del propósito del registro a realizar. Un método es el **registro monopolar** o **referencial** en el cual cada electrodo provee una señal independiente, todas ellas con una referencia común (teóricamente a un potencial cero) que se puede conseguir situando electrodos en el lóbulo de la oreja, mentón o en el mastoides[16]. Existe también el método de **registro bipolar** en donde se toman parejas de electrodos, dos a dos, registrándose las diferencias de potencial entre cada par de puntos. En esta configuración es posible realizar un número enorme de combinaciones llamadas **montajes**, los cuales también se encuentran clasificados por la Federación Internacional de EEG en **montajes longitudinales** y **transversales** dependiendo del sentido en que se registre la información sobre el cráneo.

La disposición de los electrodos del casco **Emotiv Epoc**, se ajusta al sistema 10-20, pero sólo se utilizan catorce posiciones de contacto (Figura 4) más un par de referencia (CMS y DRL) a cada lado, detrás de la oreja o encima de ella. Aunque no se ofrece información sobre el tipo de registro utilizado, todo hace suponer que se trata de un registro monopolar al existir sendos

12 Intersección del frontal y los dos huesos nasales del cráneo humano. Se diferencia claramente en la cara como un área deprimida entre los ojos, justo encima del puente de la nariz. [16]

13 Es el punto más prominente del hueso occipital en la parte posterior del cráneo. [13]

14 Situados por delante del pabellón auditivo.

15 Superficie superior de la cabeza. En los seres humanos, el *vertex* craneal está formado por cuatro huesos del cráneo: el hueso frontal, los dos huesos parietales y el hueso occipital. [28]

16 La apófisis mastoides es una prominente proyección redondeada del hueso temporal localizado detrás del conducto auditivo externo y constituye un importante punto de inserción de músculos, incluyendo el esternocleidomastoideo. [4]

electrodos de referencia, indispensables para que la obtención de las señales.

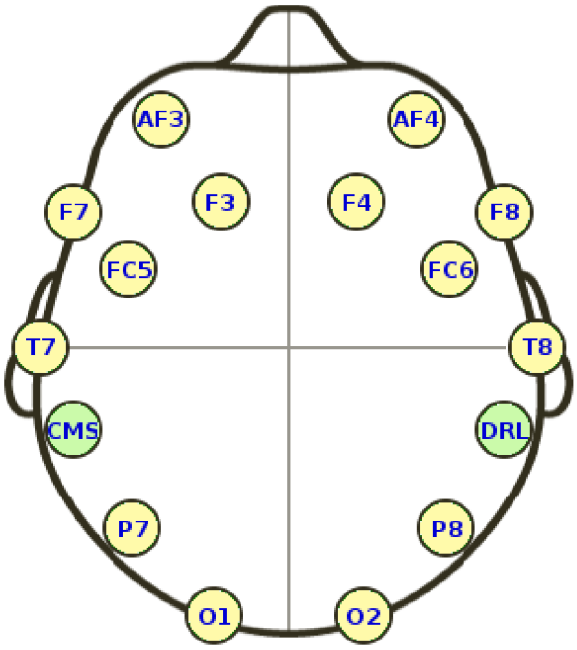


Figura 4: Electrodos del Casco Emotiv Epoc

Además de los electrodos, el casco **Emotiv Epoc** (Figura 5) contiene un giróscopo, compuesto por dos acelerómetros que facilitan información sobre los movimientos que el usuario realiza con su cabeza, un transmisor inalámbrico mediante el cual se mantiene el enlace con el receptor USB conectado al PC, todo esto alimentado por una batería recargable vía cable USB.

Figura 5: Casco Emotiv Epoc

Tanto desde la **API** como desde el software incluido en el **Kit de Desarrollo** se puede monitorizar el nivel de contacto de los electrodos, el movimiento del giróscopo, la intensidad de la señal inalámbrica y la carga de la batería.

Capítulo 3

Kit de Desarrollo Emotiv

3.1 Instalación

El **Kit de Desarrollo Emotiv** consiste en el conjunto de bibliotecas que permiten la comunicación con el casco **Emotiv Epoc**, la **API** para desarrolladores, el **El Motor Emotiv**, **El Panel de Control**, **El Programa EmoComposer**, **El Programa EmoKey**, el manual de usuario y código fuente con ejemplos.

La instalación se divide en seis pasos en los que se solicita la conformidad con la licencia de uso, a continuación se debe validar la copia introduciendo el número pedido y de serie, dando lugar a la petición de la ruta de destino de los ficheros a instalar.

Posteriormente y antes de dar por finalizado el proceso de instalación, se comprobará si se encuentra instalado en el sistema el *Microsoft .Net Framework 3.5*, procediéndose a su descarga e instalación si así fuera necesario.

El proceso descrito se puede seguir en las Figuras 6, 7, 8, 9, 10, 11, 12 y 13.

Figura 6: Instalación Emotiv SDK - Paso 1

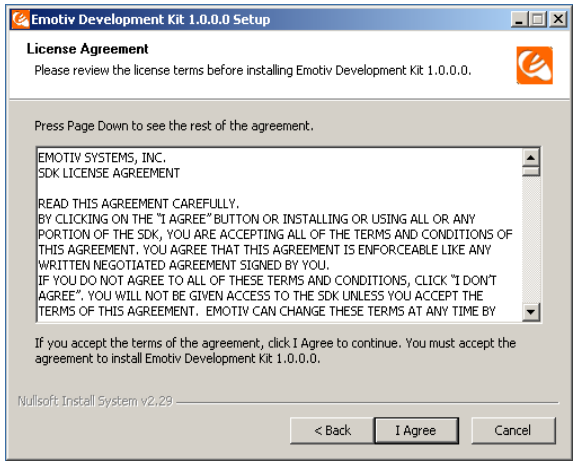


Figura 7: Instalación Emotiv SDK - Paso 2

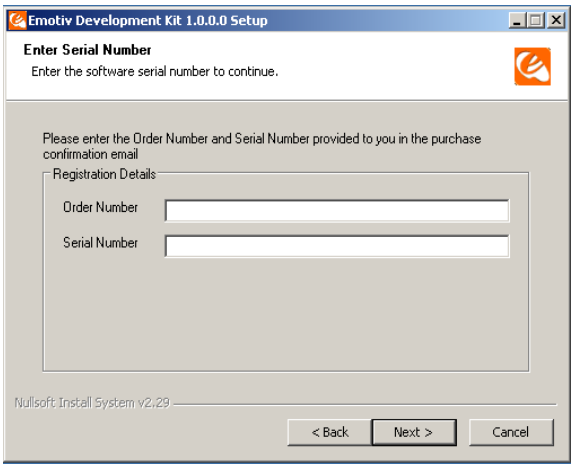


Figura 8: Instalación Emotiv SDK - Paso 3

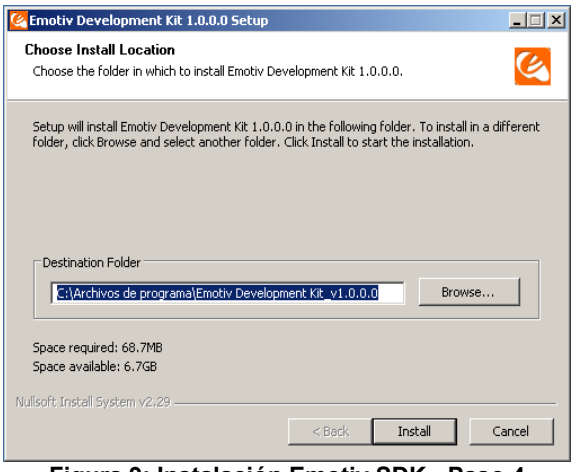


Figura 9: Instalación Emotiv SDK - Paso 4

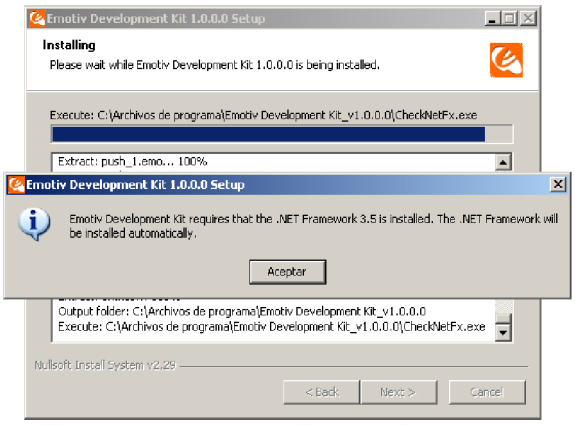


Figura 10: Instalación Emotiv SDK - Paso 5

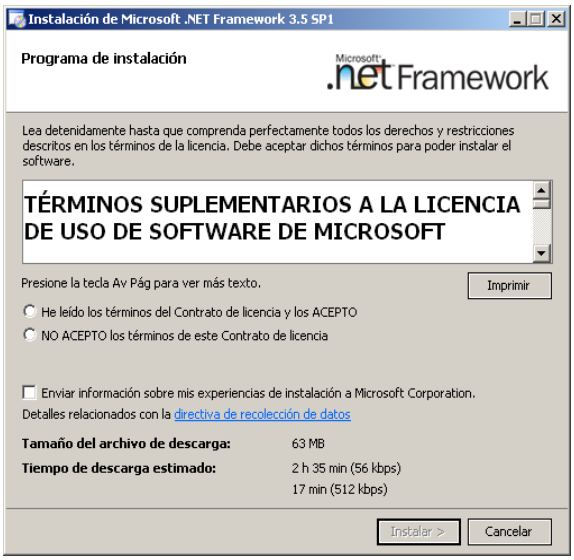


Figura 11: Instalación .net Framework - Paso 1

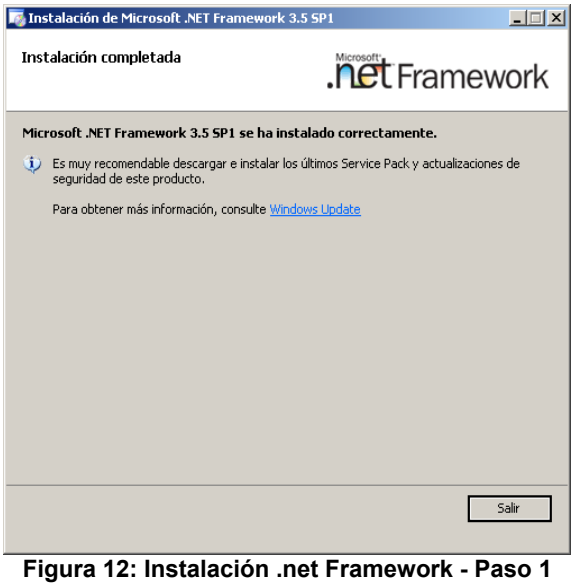


Figura 12: Instalación .net Framework - Paso 1

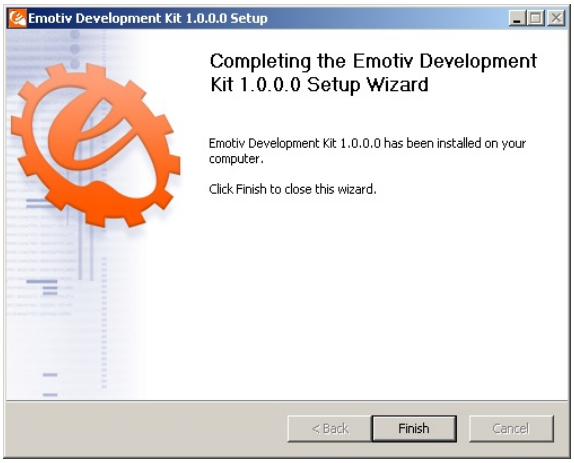


Figura 13: Instalación Emotiv SDK - Paso 6

A continuación se analizarán los componentes software incluidos en el **Kit de Desarrollo Emotiv**.

3.2 El Motor Emotiv

El **Motor Emotiv** es el componente base para la detección e interpretación de las señales provenientes de los electrodos que se encuentran situados en el casco y que capturan la información del electroencefalograma. Además se encarga de la monitorización del estado de la batería, la intensidad de la señal inalámbrica, el registro del tiempo de conexión así como de entrenar los algoritmos de reconocimiento para los modos expresivo y cognitivo, aplicando posteriormente optimizaciones a cada uno de ellos.

Se puede interactuar con él a través del **Panel de Control**, capturando eventos con el programa **EmoKey** o mediante el **API** suministrado para el desarrollo de programas. En todos los casos, salvo para el entrenamiento de los algoritmos de reconocimiento, se puede utilizar el programa **EmoComposer** para emular los eventos que produce el casco y de esta forma facilitar la depuración del código de una aplicación en desarrollo o bien comprobar las funciones de los programas suministrados con el **Kit de Desarrollo**.

3.3 El Panel de Control

El **Kit de Desarrollo Emotiv** provee un **Panel de Control** (Figura 14) que permite el acceso a las distintas funciones del motor.

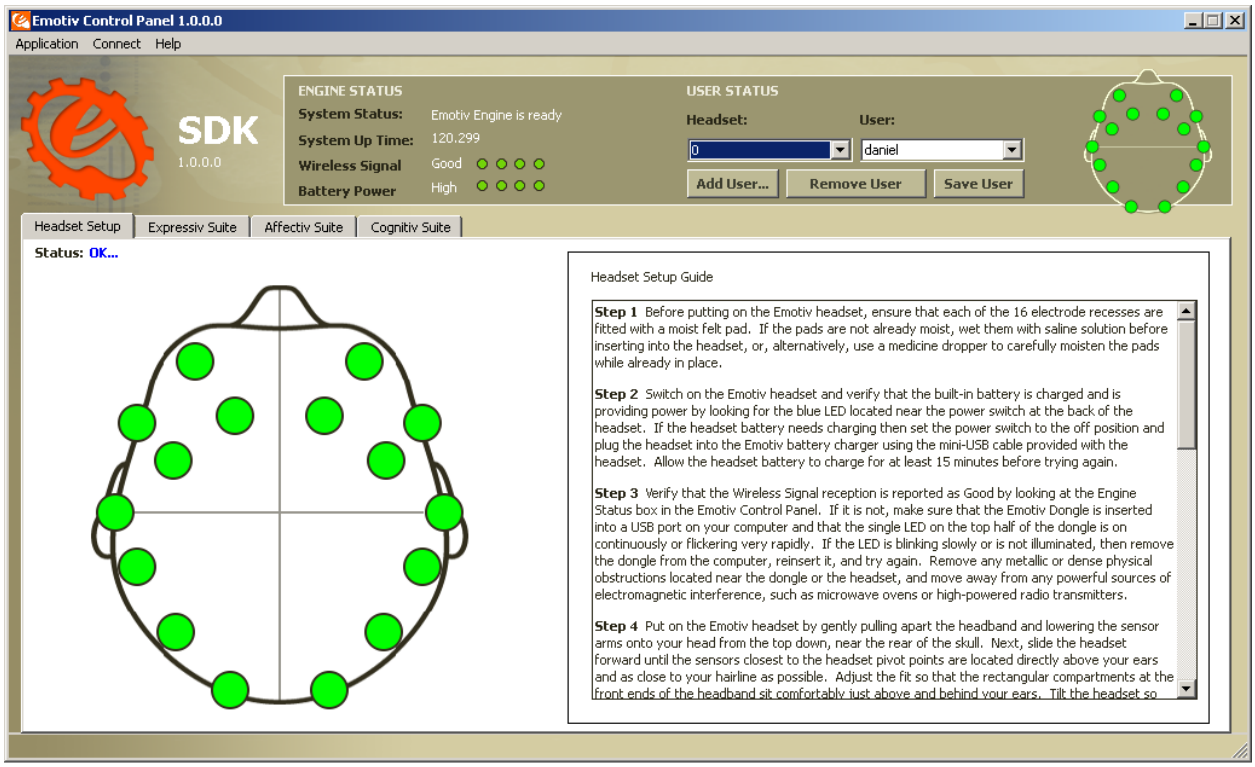


Figura 14: Panel de Control Emotiv

El mismo consta de una región superior en dónde se puede visualizar el estado del sistema, el tiempo de conexión, la intensidad de la señal inalámbrica, el nivel de carga de la batería y la calidad de contacto de los electrodos. Además se puede seleccionar el casco sobre el cual operar (se admiten hasta dos cascos conectados simultáneamente) y cargar/gestionar los perfiles de entrenamiento.

La región central del **Panel de Control** consiste en cuatro pestañas. En la primera (*Headset Setup* – Configuración del casco, Figura 14) se visualiza la disposición de los electrodos y la calidad de la señal recibida desde cada uno de ellos junto a una guía con los pasos a seguir para el ajuste y puesta en marcha del casco de forma de obtener en todos los electrodos un nivel óptimo de señal. Cada representación de un electrodo indicará mediante un código de colores el nivel de la señal que se corresponderá de forma directa con la calidad del contacto con el cuero cabelludo del usuario.

El código de colores junto a una breve descripción de cada uno se puede consultar a continuación en la Tabla 1.

Color	Descripción
Negro	Sin señal
Rojo	Señal muy pobre
Naranja	Señal pobre
Amarillo	Señal aceptable
Verde	Señal buena

Tabla 1: Código de colores para los electrodos

Existen dos sensores especiales que son utilizados como referencia, para los cuales puede escogerse su ubicación encima o detrás de cada oreja. El par de contactos que no se destinen al sensor se protegerán con una cobertura plástica en vez del fieltro humedecido (Figura 15) para evitar que entren en contacto con la piel.

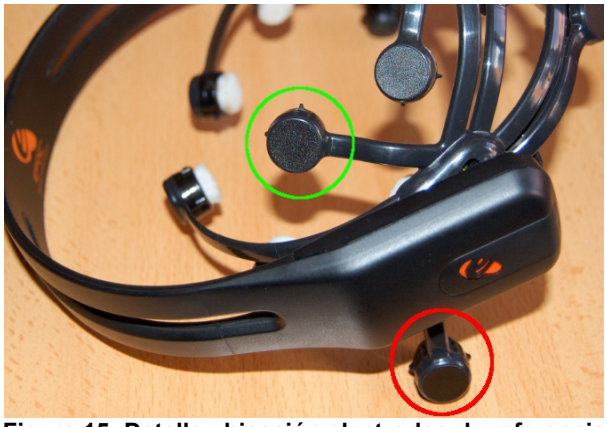


Figura 15: Detalle ubicación electrodos de referencia

Si la señal proveniente de los sensores de referencia no es de buena calidad, el resto no reportará señal alguna, coloreándose en negro al situarse todos ellos en el estado "sin señal".

3.3.1 Modo Expresivo

El modo expresivo, utiliza la información obtenida desde el electroencefalograma para determinar las expresiones faciales que el usuario está realizando.

La segunda pestaña (*Expressiv*[17] *Suite* – Modo Expresivo, Figura 16) situada en la zona central del **Panel de Control** se encuentra dedicada al modo expresivo. En la misma se pueden comprobar las detecciones que se producen en este modo así como realizar el entrenamiento del mismo. Las expresiones faciales que se detectan son las correspondientes a los siguientes gestos:

- Pestañear (*Blink*)
- Guiñar el ojo derecho (*Right Wink*)
- Guiñar el ojo izquierdo (*Left Wink*)
- Mirar hacia la derecha / izquierda (*Look Right/Left*)
- Levantar las cejas (*Raise Brow*)
- Fruncir el entrecejo (*Furrow Brow*)
- Sonreír (*Smile*)
- Apretar los dientes (*Clench*)
- Levantar la mejilla derecha (*Right Smirk*)
- Levantar la mejilla izquierda (*Left Smirk*)
- Reírse (*Laugh*)

17 *Expressiv* en una marca registrada de Emotiv Systems, Inc.

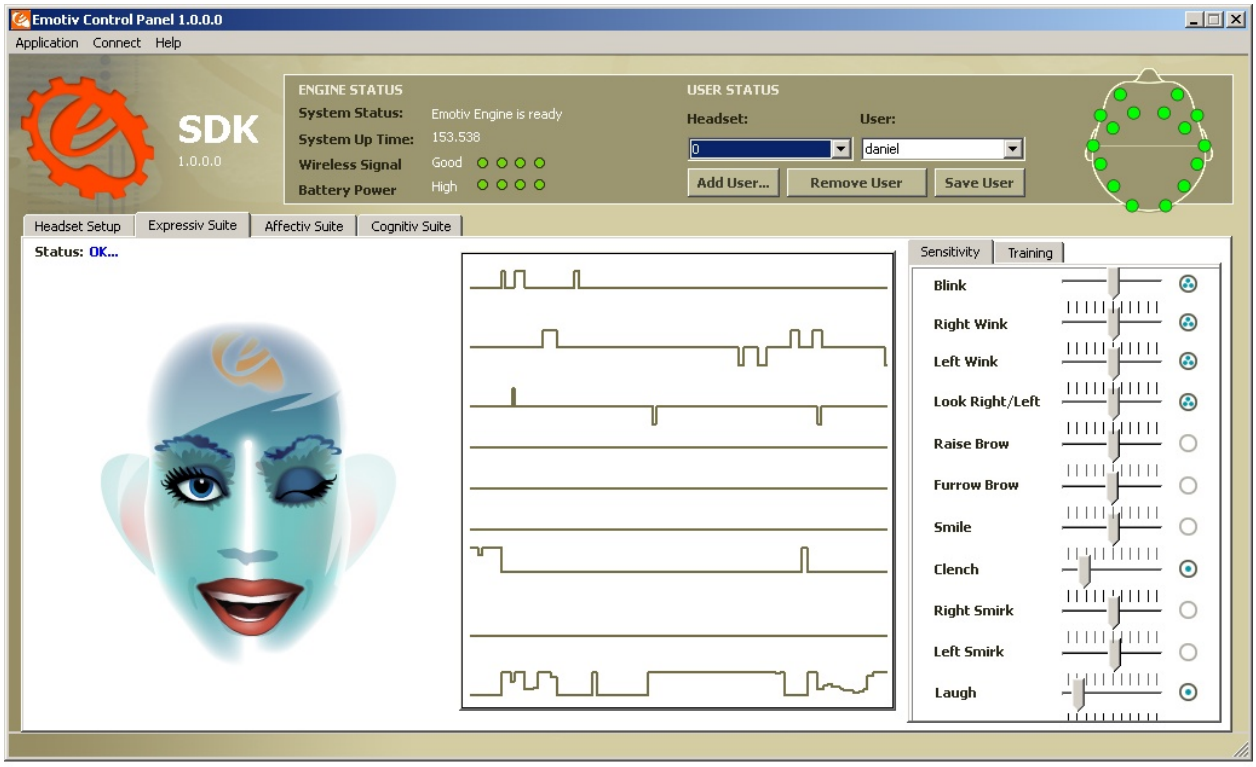


Figura 16: Panel de Control Emotiv - Modo Expresivo

La monitorización se realiza mediante un avatar azul que representa los gestos que se detectan en el usuario mediante el casco y una gráfica con una serie de señales que muestran la evolución de los gestos en el transcurso de los últimos 10 segundos. En el panel de la derecha se puede ajustar la sensibilidad de la detección de cada gesto y accediendo a la segunda pestaña realizar el entrenamiento de los mismos (Figura 17).

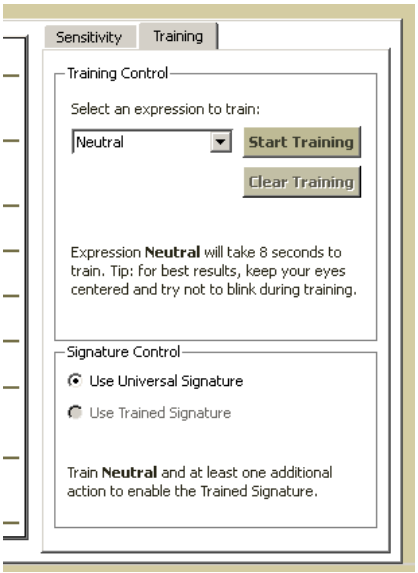


Figura 17: Entrenamiento del Modo Expresivo

Por defecto cada gesto del modo expresivo contiene una "firma" o entrenamiento universal

que es el apropiado para la mayoría de las personas, sin embargo existe la posibilidad de realizar un entrenamiento particular para cada usuario, ganando en precisión en las detecciones. Por el contrario los gestos relacionados con los ojos (pestañear, guiñar el ojo derecho, guiñar el ojo izquierdo, mirar hacia la derecha / izquierda) no pueden entrenarse estando sólo disponible su firma universal.

Un gesto para el cual haya sido entrenada su firma se representará con el icono de un círculo con un punto central celeste, para los gestos con firmas de detección universales se utilizará un icono circular con tres puntos celestes y para los que se encuentren pendientes de entrenamiento, un círculo vacío (Figura 18).

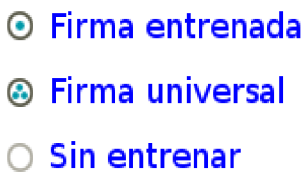


Figura 18: Iconos entrenamiento expresivo

3.3.2 Modo Afectivo

La tercera pestaña (*Affectiv*[18] *Suite* – Modo Afectivo, Figura 19) del **Panel de Control** se corresponde con el modo afectivo en donde se indica la evolución en tiempo real de los cambios emocionales experimentados por el usuario.

Se dispone de dos gráficas en las que se podrán seleccionar las señales correspondientes a los niveles de:

- Compromiso/Aburrimiento (*Engagement/Boredom*)
- Frustración (*Frustation*)
- Meditación (*Meditation*)
- Emoción instantánea (*Instantaneous Excitement*)
- Emoción a largo plazo (*Long-Term Excitement*)

18 *Affectiv* en una marca registrada de Emotiv Systems, Inc.

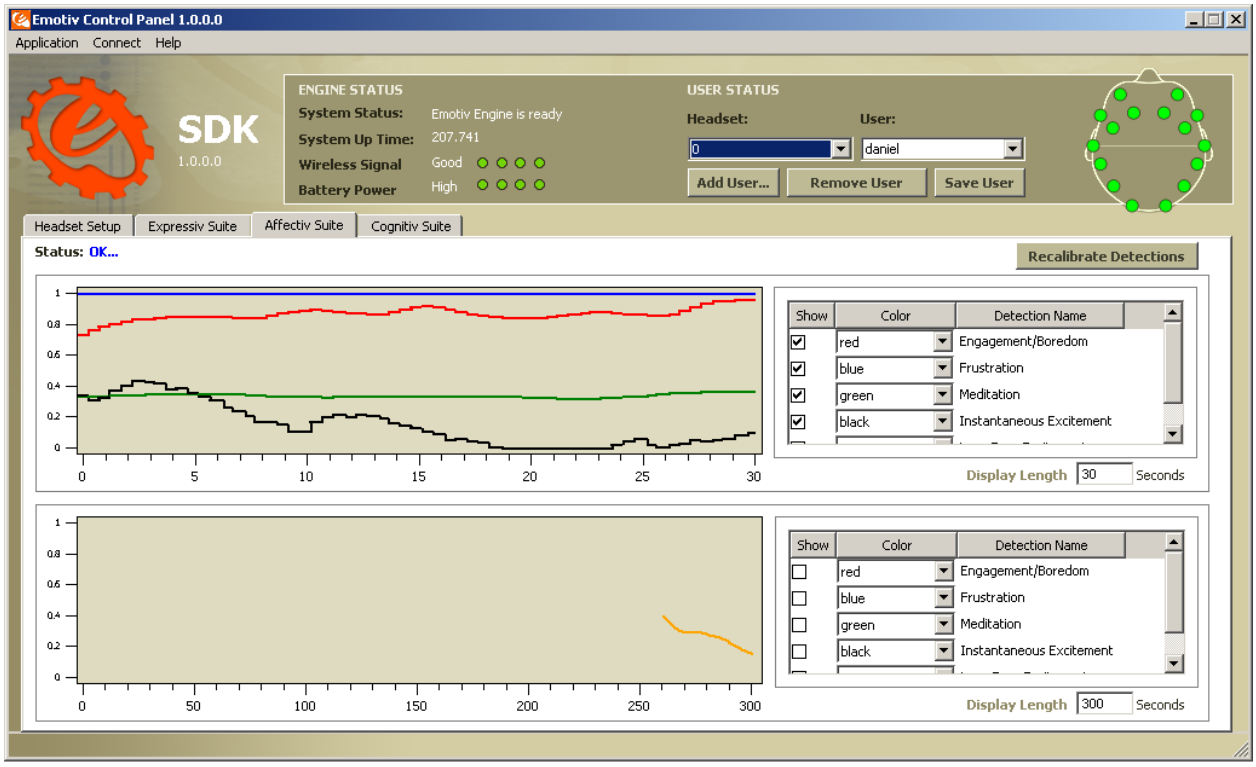


Figura 19: Panel de Control Emotiv - Modo Afectivo

Por defecto en la gráfica superior visualiza la evolución de las señales en un lapso de 30 segundos, mientras que la inferior se visualizan los últimos 5 minutos lo que la hace propicia para representar la señal correspondiente a la emoción a largo plazo. De todas formas las escalas temporales pueden modificarse en el panel de la derecha así como escoger qué señales representar, incluso asignándoles un color en particular.

La detección en el modo afectivo no necesita entrenamiento alguno ya que se basa en ondas cerebrales características y comunes a todos los individuos, sin embargo la información recolectada es guardada en el perfil del usuario para ser utilizada como ajuste de los extremos mínimos y máximos, mejorando así el rango de detección y la precisión.

3.3.3 Modo Cognitivo

La cuarta y última pestaña del **Panel de Control** (*Cognitiv*[19] *Suit* – Modo Cognitivo, Figura 20) gestiona la detección y entrenamiento del modo cognitivo. En este modo se detectan hasta 13 diferentes acciones mediante un escaneo en tiempo real de las ondas cerebrales cuando el usuario intenta interactuar con un objeto real o virtual.

19 *Cognitiv* en una marca registrada de Emotiv Systems, Inc.

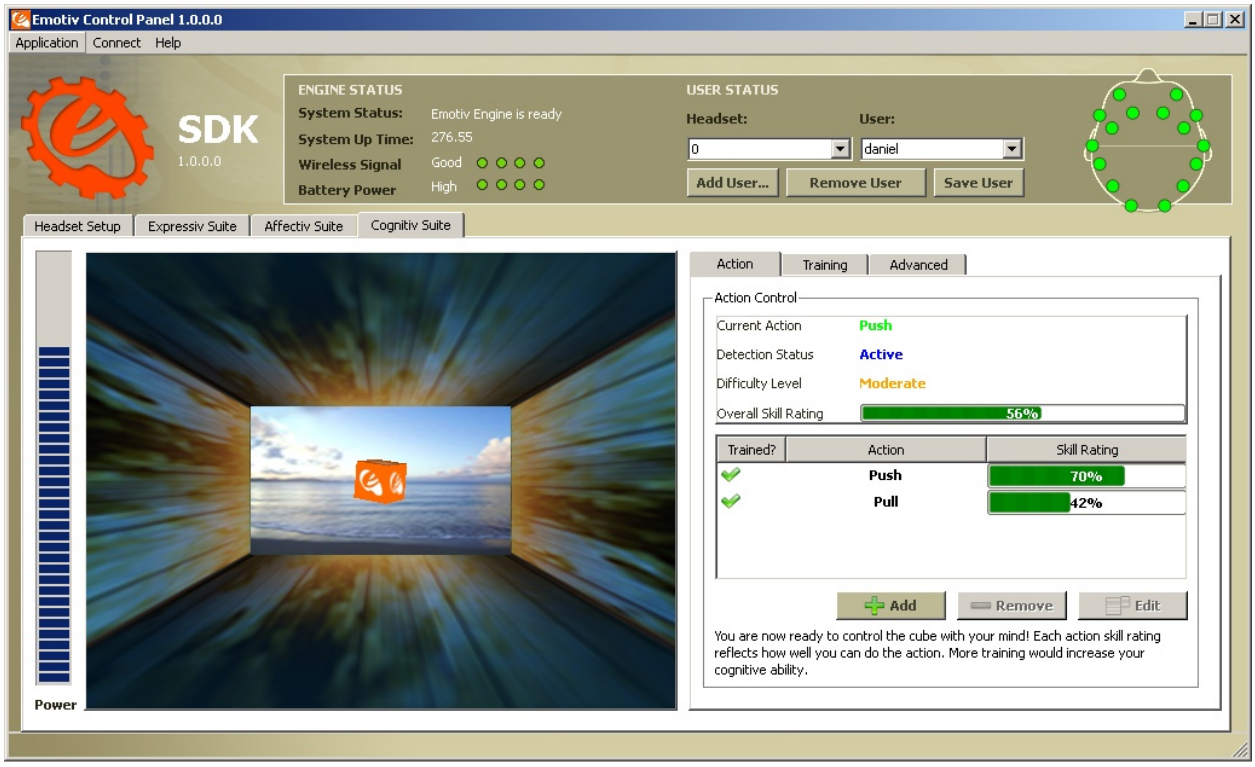


Figura 20: Panel de Control Emotiv - Modo Cognitivo

Las acciones se dividen en direccionales (Figura 21):

- Empujar (*Push*)
- Tirar (*Pull*)
- Desplazar hacia la Izquierda (*Left*)
- Desplazar hacia la Derecha (*Right*)
- Desplazar hacia Arriba (*Up*)
- Desplazar hacia Abajo (*Down*)

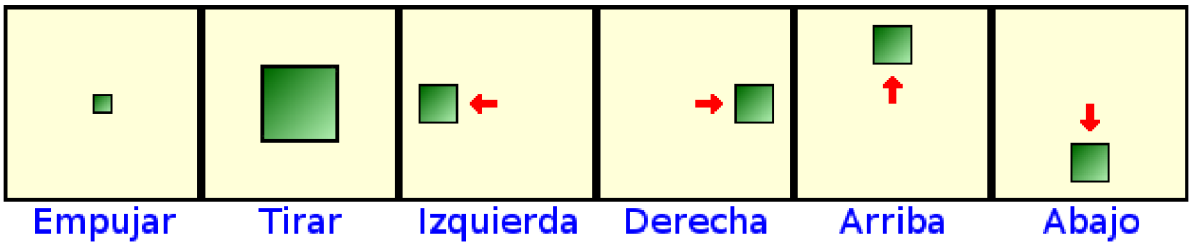


Figura 21: Acciones cognitivas direccionales

Rotacionales (Figura 22):

- Rotar en sentido Horario (*Rotate Clockwise*)
- Rotar en sentido Antihorario (*Rotate Counter-clockwise*)
- Rotar hacia la Izquierda (*Rotate Left*)
- Rotar hacia la Derecha (*Rotate Right*)
- Rotar hacia Adelante (*Rotate Forward*)
- Rotar hacia Atrás (*Rotate Backward*)

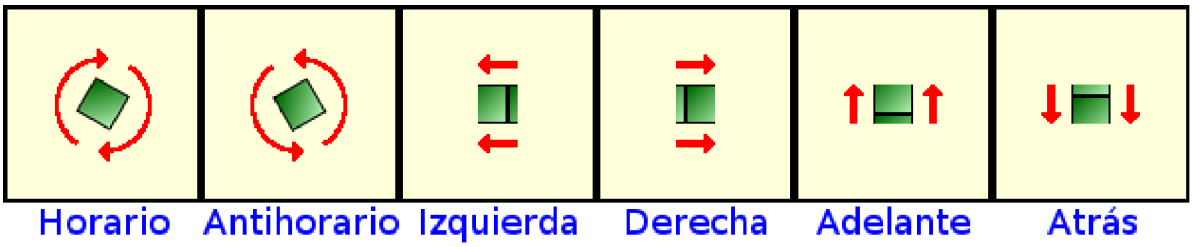


Figura 22: Acciones cognitivas rotativas

Y por último una acción que parte de la imaginación del usuario: Hacer Desaparecer (*Dissapear*) (Figura 23).

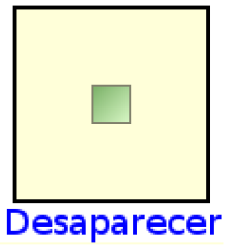


Figura 23: Acción cognitiva "Hacer desaparecer"

Sólo se pueden escoger cuatro acciones para ser entrenadas a la vez, más una acción especial que se corresponde con el estado neutral o de reposo. A medida que se entrenan más acciones, se hace más difícil el ser preciso en la activación de las mismas requiriendo un mayor grado de experiencia para evitar sobre todo falsos positivos.

La activación de una acción implica un nivel de potencia que aumenta con la concentración del usuario, esto se verá reflejado con un aumento del valor indicado en la barra de la izquierda del panel situada junto a la animación correspondiente del cubo. Este valor se hará cada vez mayor a medida que la potencia se eleve. Por ejemplo al activarse la acción de Empujar, a más nivel de concentración, mayor potencia en la barra de la izquierda y más lejos será “empujado” el cubo en la animación.

En la Figura 24 puede verse la pestaña de entrenamiento del modo cognitivo.

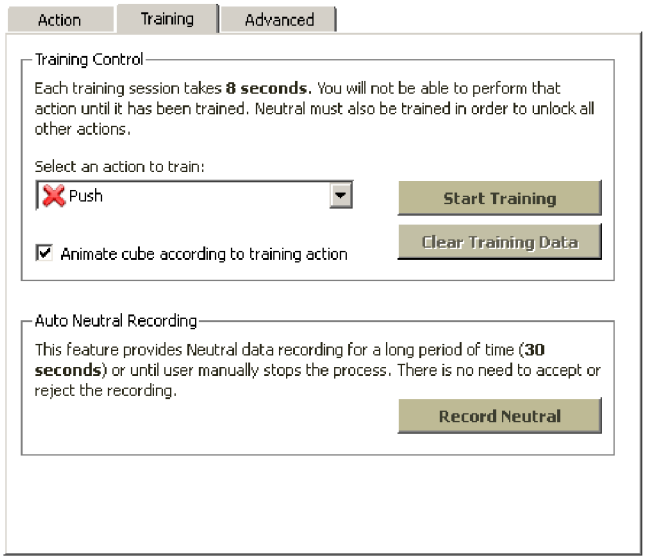


Figura 24: Entrenamiento del Modo Cognitivo

A diferencia del modo expresivo y en aras de una mejor precisión, será necesario realizar varias sesiones de entrenamiento para cada acción (cada sesión tiene una duración de 8 segundos) lo que se traducirá en un aumento del porcentaje de entrenamiento (*Skill Rating*) completado para cada una de ellas, incrementándose también el porcentaje general de entrenamiento (*Overall Skill Rating*). Véase en la Figura 25 el detalle de los porcentajes de entrenamiento para las dos acciones (70% y 42%) y el porcentaje general (56%).

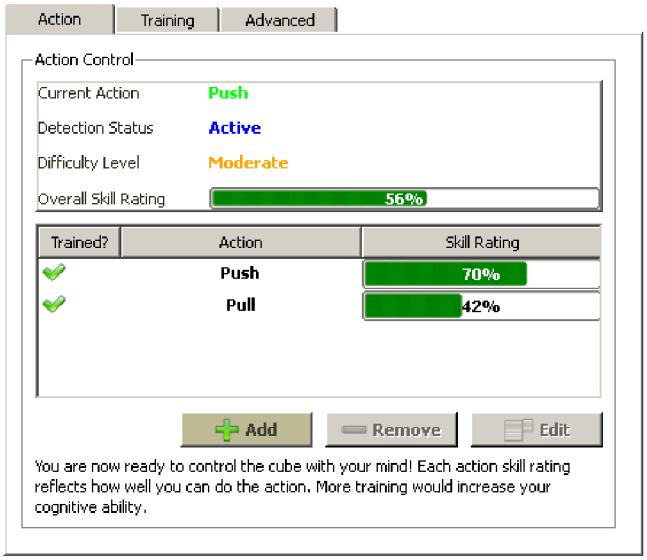


Figura 25: Detalle de los porcentajes de entrenamiento

Nótese que si el entrenamiento es fallido, es decir no sigue el mismo patrón cerebral que el de las sesiones de entrenamiento realizadas previamente para esa acción, el porcentaje de

entrenamiento lejos de aumentar, disminuirá. Para evitar esta situación desfavorable, el programa permite aceptar o rechazar cada sesión de entrenamiento una vez finalizada la misma o incluso interrumpirla cuando el usuario así lo considere apropiado (por ejemplo si se ha distraído o ha perdido la concentración). Una sesión de entrenamiento se interrumpirá automáticamente si durante la misma se pierde la señal de enlace con el casco o disminuye la calidad de contacto de alguno de los electrodos.

Antes de proceder al entrenamiento de las acciones, deberá entrenarse el modo neutral. Además con el objeto de mejorar aún más la detección evitando falsos positivos, existe un entrenamiento especial que registra un estado de inactividad durante 30 segundos durante el cual el usuario deberá mantener un estado mental pasivo o de relajación. También puede ser de utilidad para entrenar el modo neutral pensar en una situación diferente al resto de las acciones entrenadas.

Por último existe también en el panel lateral una pestaña con opciones avanzadas (Figura 26) para ajustar el comportamiento de la detección cognitiva. No se recomienda la modificación de estos parámetros ya que vienen configurados de la forma más apropiada para la gran mayoría de los seres humanos.

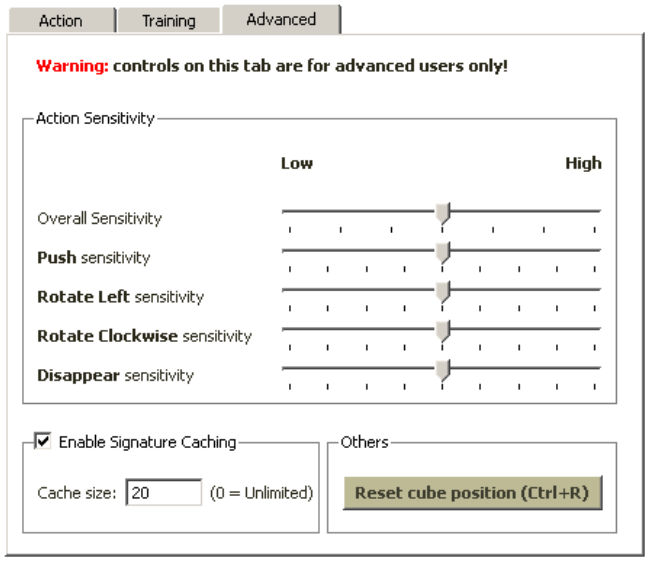


Figura 26: Opciones avanzadas del modo cognitivo

3.4 El Programa EmoComposer

El programa **EmoComposer** se utiliza para emular el funcionamiento del casco **Emotiv Epoc** pudiendo generar actualizaciones de estados con información de la carga de la batería, nivel de señal inalámbrica, calidad de contacto de los electrodos y por supuesto, acciones de los tres

modos: expresivo, afectivo y cognitivo.

Tanto el **Panel de Control**, como el programa **EmoKey** pueden conectarse al **EmoComposer**, incluso la **API** provista dispone de una función para ello, lo cual es especialmente útil para la fase de desarrollo ya que permite realizar las comprobaciones del programa que se está desarrollando en un entorno controlado, sin la necesidad de estar utilizando el casco en todo momento.

El programa **EmoComposer** está dividido en dos pestañas, *EmoScript* – Guiones (Figura 27) e *Interactiv* – Interactivo, (Figura 28) dedicadas cada una a un modo de funcionamiento diferente.

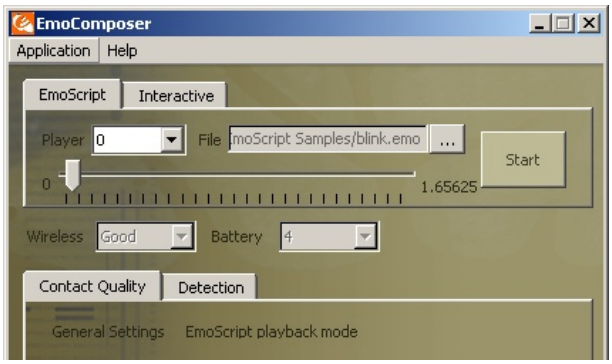


Figura 27: EmoComposer - Modo Guiones

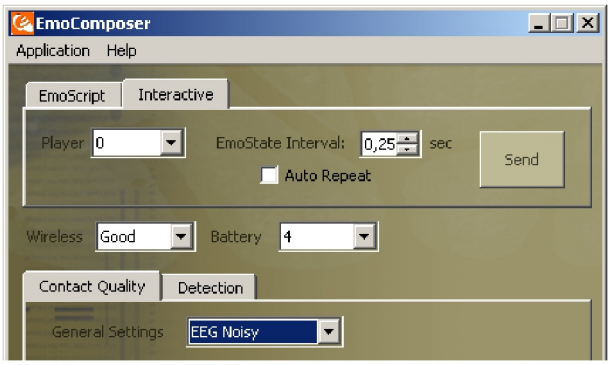


Figura 28: EmoComposer - Modo Interactivo

La pestaña para trabajar con **Guiones** (*scripts*) permite cargar un fichero con eventos el cual se codifica utilizando el lenguaje EML[20]. Mediante el control deslizable se podrá avanzar y retroceder por la línea temporal de los eventos definidos en el fichero, observando los cambios que se producen en el resto de los paneles del emulador según las instrucciones definidas en el guión, mientras que con el botón de inicio (*Start*) se dará inicio a la ejecución del fichero de eventos. En el Listado 1 se puede observar un ejemplo de un guión de eventos escrito en lenguaje EML.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE EML>
<EML version="1.0" language="en_US">
<sequence>
   <time value ="0s23t">
      <signal_quality value="1,1,1,1,1,1,1,0.909091,1,1,1,1,1,1,1,1,1,1" />
   </time>
   <time value ="0s25t">
      <expressiv_eye event="blink" value="1" />
      <signal_quality value="1,1,1,1,1,1,1,0.909091,1,1,1,1,1,1,1,1,1,1" />
   </time>
   <time value ="0s28t">
      <expressiv_eye event="blink" value="1" />
       <signal_quality value="1,1,1,1,1,1,1,0.909091,1,1,1,1,1,1,1,1,1,1" />
```

20 EML: EmoComposer Markup Languaje – Lenguaje de Marcado del EmoComposer.

```
   </time>
</sequence>
</EML>
```

Listado 1: Guión EML de ejemplo

La pestaña para el **Modo Interactivo** (*Interactive*) permite configurar manualmente un estado **Emotiv**, pudiéndose indicar el estado de la carga de la batería y el nivel de la señal del enlace inalámbrico. Además se podrá programar el envío repetitivo de eventos fijando el intervalo deseado y el usuario que los origina. Esta pestaña está a su vez dividida en *Contact Quality* – Calidad de los Contactos (Figura 29) y *Detection* – Detección (Figura 30).

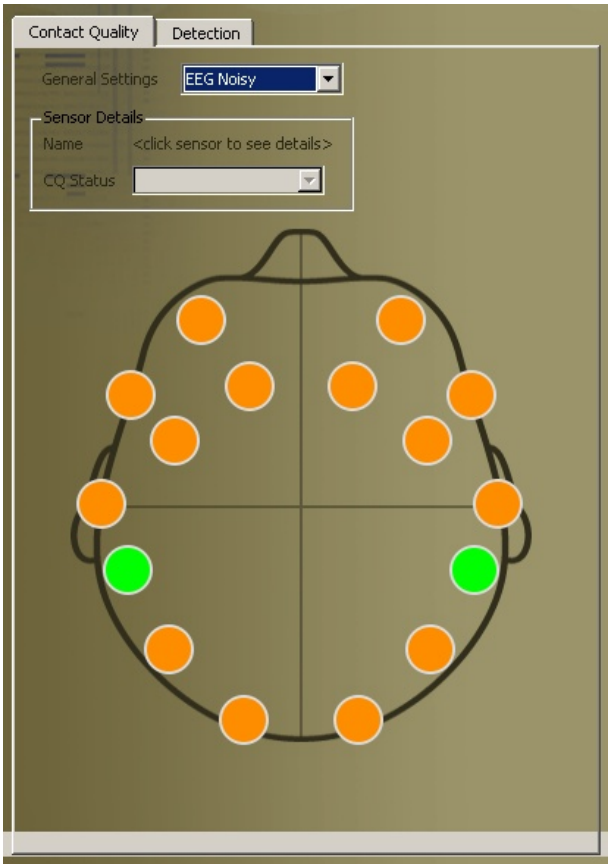


Figura 29: EmoComposer: Calidad de los contactos

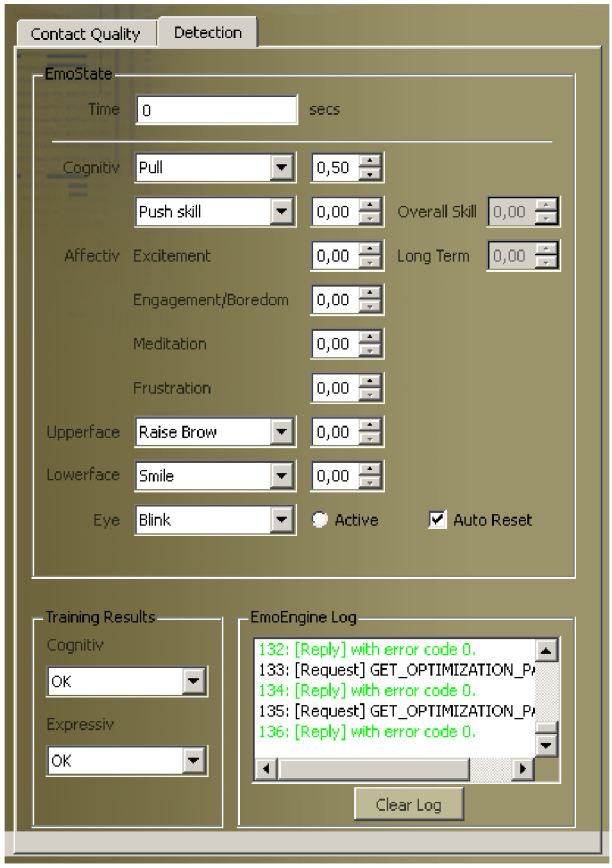


Figura 30: EmoComposer: Detección

Modificando la **Calidad de los Contactos** se pueden simular situaciones en las que el casco no se encuentra correctamente colocado o algún electrodo cuyo fieltro no se ha humedecido apropiadamente, pudiéndose modificar todos los electrodos a la vez o cada uno individualmente.

Por otro lado desde la pestaña de **Detección** se tiene disponible toda la gama de acciones correspondientes a los tres modos de funcionamiento, junto con la marca temporal del evento a generar. Además se podrán programar los resultados de las acciones de entrenamiento para los dos modos que lo admiten (expresivo y cognitivo). Por último esta pestaña dispone de un área de

texto en donde se visualiza el registro de la interacción del emulador y el programa cliente[21] conectado al mismo. Desde la pestaña **Detección** el valor de la calidad de contacto de los electrodos que se envía dentro del estado generado siempre será "Buena".

3.5 El Programa EmoKey

El programa **EmoKey** asigna a los eventos detectados por el **Motor Emotiv** o bien emulados por el programa **EmoComposer** una secuencia predefinida de pulsaciones de teclas (llamada *EmoKey Mapping* – Mapeo EmoKey) que se pueden enviar a cualquier programa de Microsoft Windows en ejecución. Nótese que con el fin de poder seleccionar el usuario y configurar los parámetros de detección, la conexión entre el programa **EmoKey** y el **Motor Emotiv** se realiza vía el **Panel de Control**.

En la Figura 31 se puede observar la ventana del Programa **EmoKey** en donde se ha configurado el envío de la tecla "A" a una aplicación de Microsoft Windows (por ejemplo el Bloc de Notas) cuando se detecte la acción del Modo Cognitivo, empujar (*pull*) con una potencia mayor que el 80%.

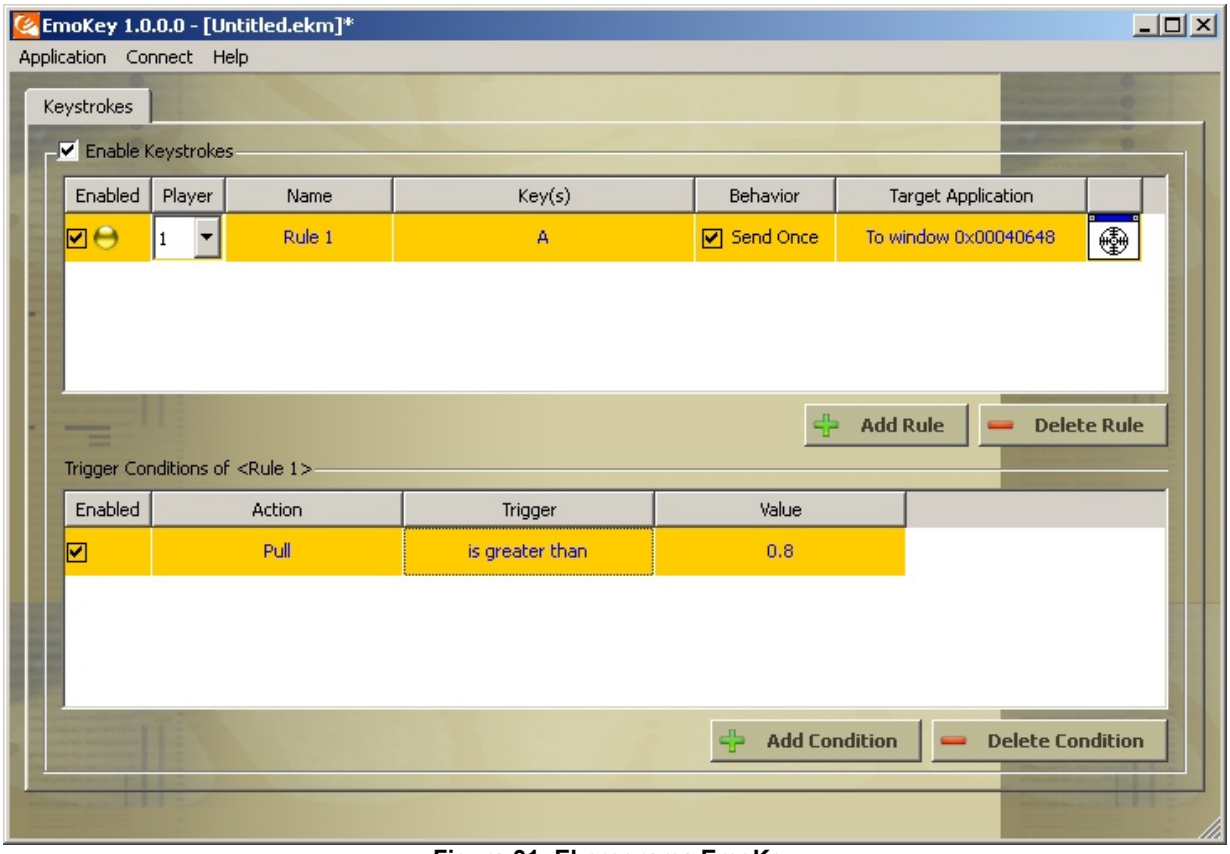


Figura 31: El programa EmoKey

21 Nótese que el programa **EmoComposer** se comporta como un servidor de **Eventos Emotiv** escuchando en una dirección IP y puerto determinados, a la espera de peticiones de conexión de los clientes.

Es posible definir cuantas veces se enviará la secuencia a la aplicación destino y también se podrá especificar más de una condición de disparo debiéndose cumplir todas ellas para que se active el envío.

La selección de la aplicación de destino se realiza haciendo doble clic sobre la celda *Target Application (*Aplicación Destino) de cada línea de secuencia, esto visualizará el diálogo de la Figura 32 en donde se podrá seleccionar como destino la aplicación que tiene el foco, o bien, una en particular arrastrando y soltando el icono de la diana sobre una ventana del escritorio.

Una vez creada la relación, y cuando se satisfagan todas condiciones para una secuencia, esta se disparará enviando las pulsaciones definidas a la aplicación de destino.

Por último podrán guardarse las configuraciones realizadas en un fichero en disco para poder ser posteriormente recuperadas.

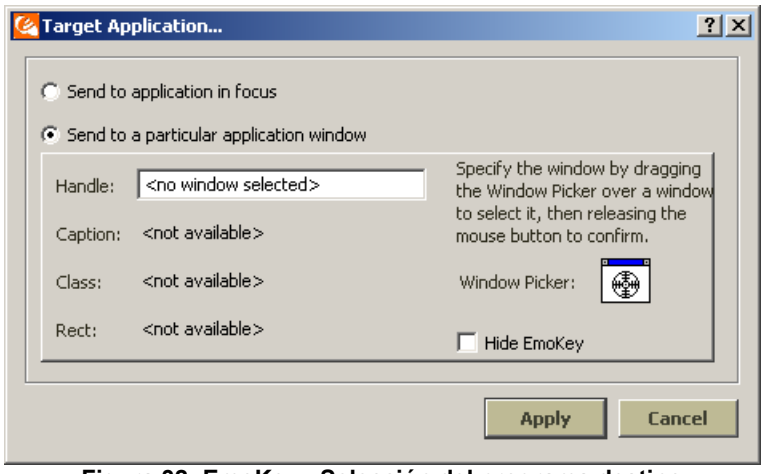


Figura 32: EmoKey - Selección del programa destino

3.6 API

El **API Emotiv** consiste en tres ficheros de cabecera (**edk.h**, **EmoStateDLL.h** y **edkErrorCode.h**) y dos *DLL's* para la plataforma Microsoft Windows (**edk.dll** y **edk_utils.dll**).

El **Motor Emotiv**, cuya funcionalidad la proporciona la biblioteca **edk.dll**, provee las siguientes funcionalidades:

- Comunicación con el **Casco Emotiv Epoc**.
- Recepción y procesamiento de la señal de electroencefalograma y del giróscopo.
- Configuración del funcionamiento según el usuario y las características de la aplicación.
- Procesamiento y volcado de los resultados de la detección en un **Estado Emotiv**

(EmoState).

Un **Estado Emotiv** es una estructura de datos que contiene información sobre una detección que comprende el estado del casco (batería, señal inalámbrica, calidad de los contactos, etc.), la expresión facial, el estado emocional y el estado cognitivo del usuario.

Para acceder a las actualizaciones de estado primero será necesario conectar con el **Motor Emotiv** mediante la función **EE_EngineConnect** o bien al programa **EmoComposer** utilizando la función **EE_EngineRemoteConnect**.

Una vez que la aplicación se ha conectado, las actualizaciones se obtienen mediante eventos que pueden ser recuperados utilizando la función **EE_EngineGetNextEvent**. Estas peticiones suelen realizarse a través del bucle principal de la aplicación, siendo recomendable realizar de 10 a 15 por segundo para obtener una respuesta en tiempo real.

Por último, antes de finalizar la aplicación, se deberá cerrar la conexión mediante el uso de la función **EE_EngineDisconnect**.

En la Figura 33 se presenta un diagrama con el bucle típico de recuperación de eventos.

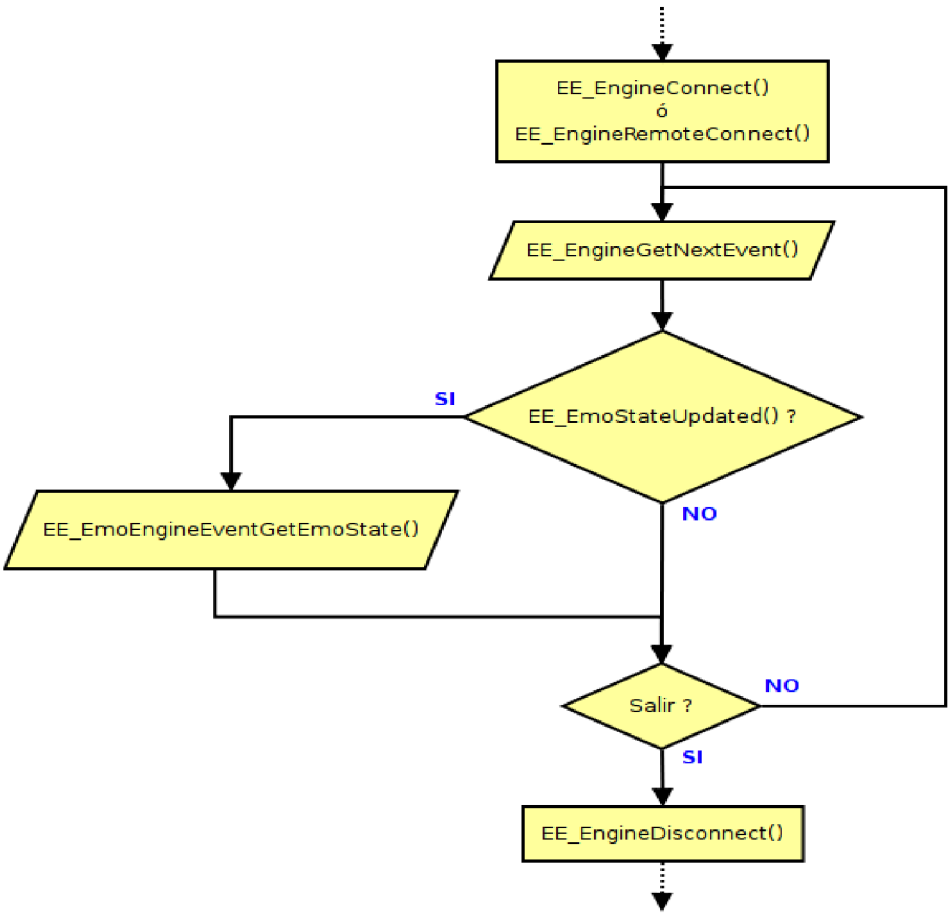


Figura 33: Bucle de recuperación de eventos

Los eventos principales que pueden recuperarse mediante las peticiones son:

- Eventos relativos al hardware, por ejemplo conexión y desconexión de los dispositivos.
- Eventos de estado (EmoState), cambios en los estados expresivo, cognitivo o afectivo.
- Eventos específicos relacionados con el entrenamiento y la detección de un nuevo estado.

Un modo de funcionamiento distinto consiste en el entrenamiento de una acción o gesto que pasará a formar parte del perfil del usuario. La secuencia a seguir difiere levemente si se trata de entrenar un gesto perteneciente al modo expresivo o una acción del modo cognitivo.

En la Figura 34 se puede visualizar la secuencia a seguir para el entrenamiento de una acción en **Modo Expresivo**.

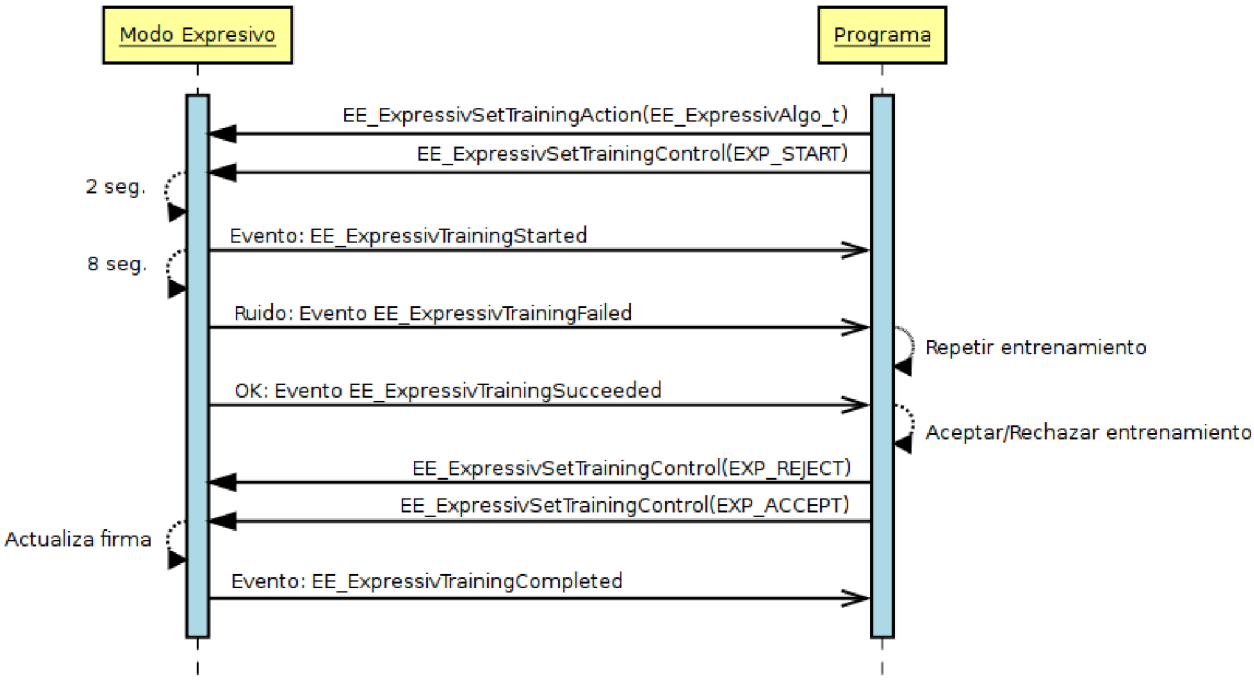


Figura 34: Entrenamiento del Modo Expresivo

En primer lugar se llama a la función **EE_ExpressivSetTrainingAction** indicando la acción expresiva (gesto) a entrenar utilizando una de las constantes pertenecientes a la enumeración **EE_ExpressivAlgo_enum** (**EE_ExpressivAlgo_t**) para luego realizar la solicitud de comienzo de entrenamiento utilizando la función **EE_ExpressivSetTrainingControl** con la señal **EXP_START** perteneciente a la enumeración **EE_ExpressivTrainingControl_enum**.

Transcurridos 2 segundos que tienen como objeto facilitar que el usuario se prepare para la sesión, el **Motor Emotiv** generará el evento **EE_ExpressivTrainingStarted** perteneciente a la enumeración **EE_ExpressivEvent_enum** para indicar el comienzo del entrenamiento el cual se

extenderá por 8 segundos. Una vez transcurrido este lapso, si la señal del electroencefalograma no sufre interferencias ni una disminución de la calidad de los contactos, el **Motor Emotiv** generará el evento **EE_ExpressivTrainingSucceeded** indicando un entrenamiento exitoso (Figura 35).

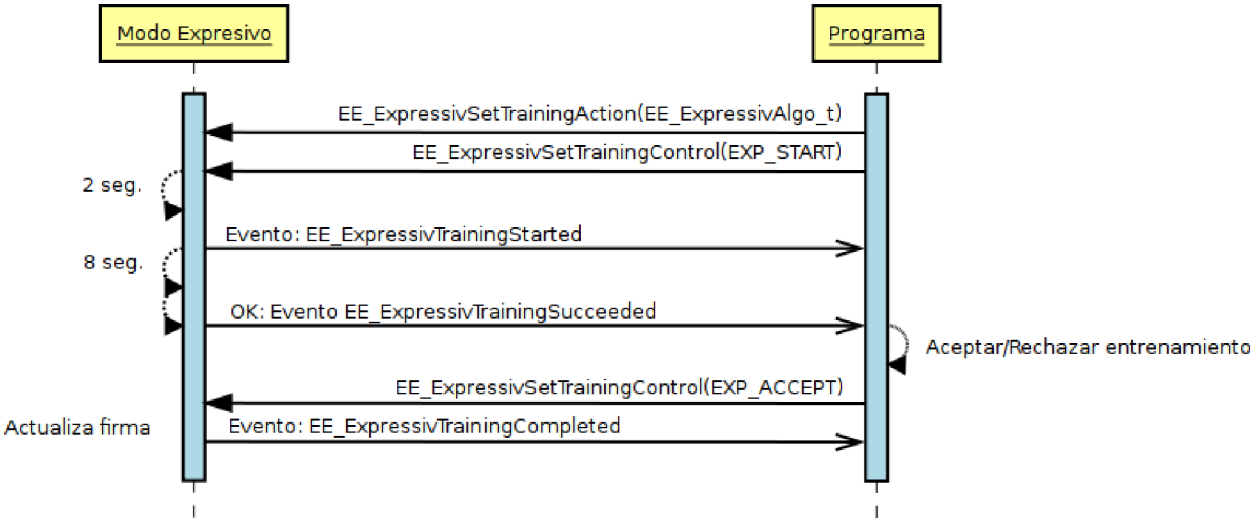


Figura 35: Entrenamiento del Modo Expresivo aceptado por el usuario

A continuación el programa podrá proceder a su registro dentro del perfil de usuario utilizando la función **EE_ExpressivSetTrainingControl** con el parámetro **EXP_ACCEPT** de la enumeración **EE_ExpressivTrainingControl_enum**, en cuyo caso el **Motor Emotiv** responderá con el evento **EE_ExpressivTrainingCompleted** perteneciente a la enumeración **EE_ExpressivEvent_enum** (Figura 35).

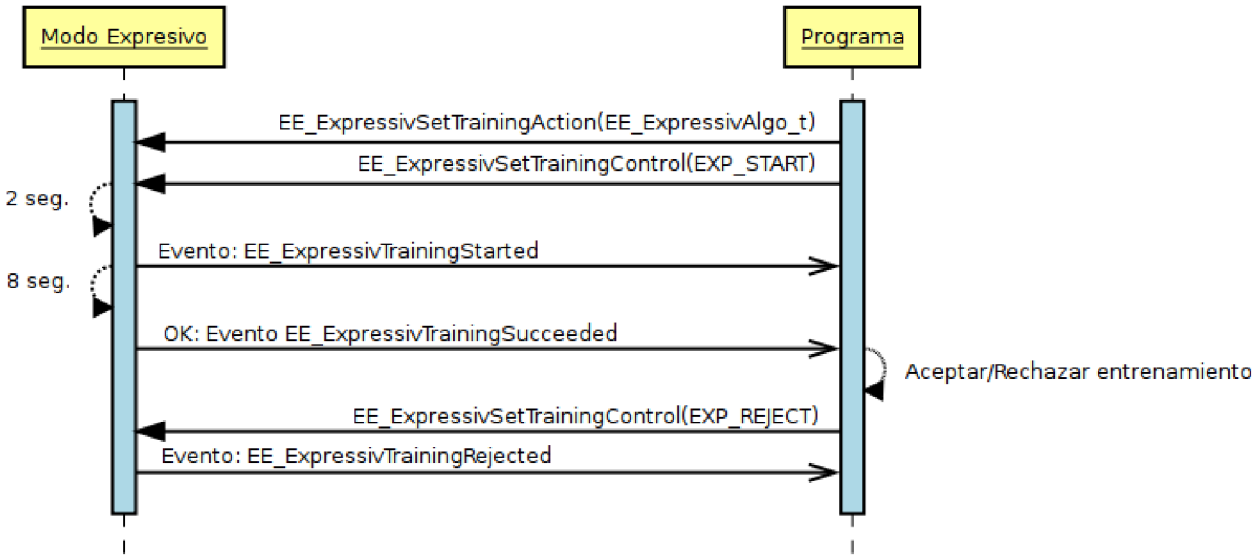


Figura 36: Entrenamiento del Modo Expresivo rechazado por el usuario

Por el contrario, para rechazar el entrenamiento se empleará el parámetro **EXP_REJECT**. En

este caso el **Motor Emotiv** responderá con un evento **EE_ExpressivTrainingRejected** (Figura 36).

Por último, existe la posibilidad de que la sesión de entrenamiento se haya visto afectada por una señal de mala calidad proveniente del casco o por algún contacto deficiente de los electrodos con el cuero cabelludo, en este caso la respuesta del **Motor Emotiv** será mediante el evento **EE_ExpressivTrainingFailed** (Figura 37).

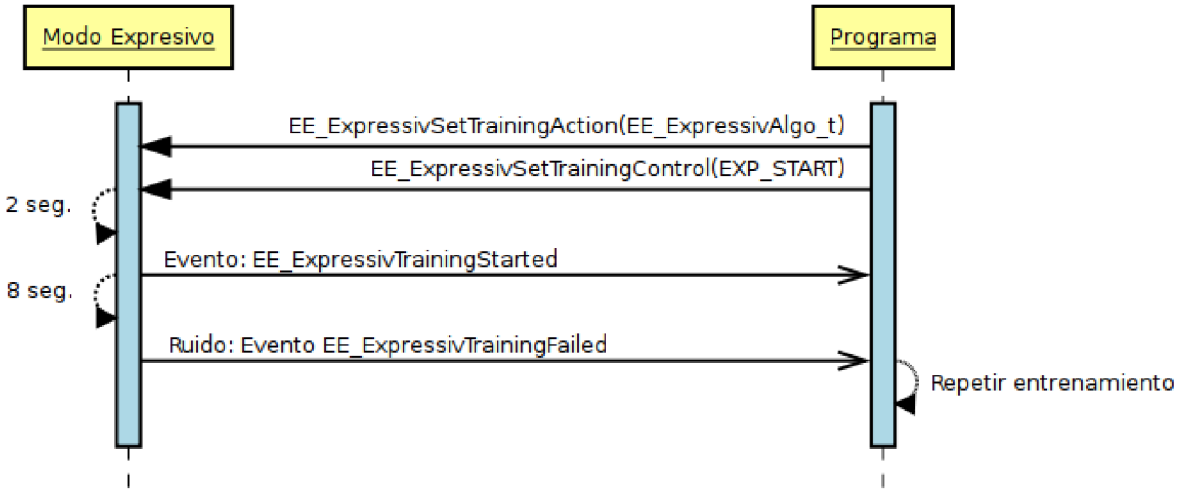


Figura 37: Entrenamiento fallido del Modo Expresivo

El otro modo que contempla entrenamiento es el modo cognitivo. Como se aprecia en la Figura 38, la secuencia de entrenamiento es similar a la ya vista para el modo expresivo con la salvedad que en este caso se utilizan las funciones del modo cognitivo **EE_CognitivGetTrainingAction** y **EE_CognitivSetTrainingControl** mientras que las enumeraciones empleadas para los parámetros y eventos son **EE_CognitivAction_enum** (**EE_CognitivAction_t**), **EE_CognitivEvent_enum** y **EE_CognitivTrainingControl_enum**.

Nótese además que en este caso no existe el retardo de 2 segundos antes del comienzo de la sesión de entrenamiento.

Como se ha mencionado anteriormente, sólo podrá configurarse la detección de cuatro acciones cognitivas simultáneamente junto con la acción neutral.

A continuación se estudiarán cada una de las funciones contenidas dentro de la **API**, así como las estructuras, tipos y enumeraciones. Nótese que la mayoría de las funciones devuelven un valor entero (*int*) correspondiente al tipo **EDK_ERROR** indicando si la función se ha ejecutado con éxito o si ha ocurrido algún error (véase en el punto 3.6.6 la definición de los códigos de error).

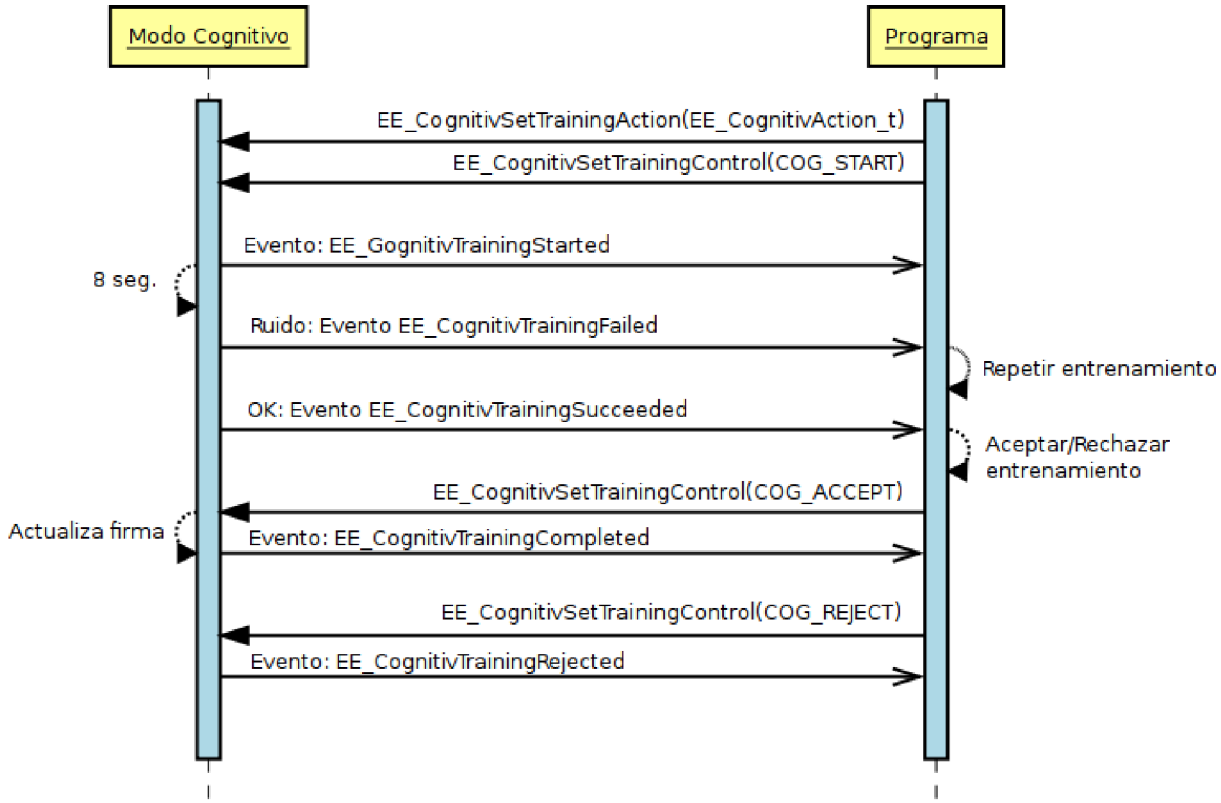


Figura 38: Entrenamiento del Modo Cognitivo

3.6.1 Funciones del Kit de Desarrollo Emotiv (EDK_API)

Estas funciones forman parte de la interfaz principal que permite las interacciones entre los programas externos y el motor de detección **Emotiv**.

Ninguna de estas funciones son *thread-safe*.

- EE_CognitivEventGetType

```
EE_CognitivEvent_t EE_CognitivEventGetType ( EmoEngineEventHandle hEvent )
```

Devuelve el tipo del evento del modo cognitivo que ha sido previamente obtenido utilizando la función **EE_EngineGetNextEvent**.

Toma como parámetro el puntero al evento del modo cognitivo y devuelve el tipo de evento al que pertenece.

- EE_CognitivGetActionSensitivity

```
int EE_CognitivGetActionSensitivity ( unsigned int userId,
                                      int *pAction1SensitivityOut,
                                      int *pAction2SensitivityOut,
                                      int *pAction3SensitivityOut,
                                      int *pAction4SensitivityOut )
```

Realiza una consulta sobre la sensibilidad de las acciones del modo cognitivo definidas.

Los parámetros de corresponden con el identificador de usuario y las cuatro sensibilidades que se devuelven por referencia. Además la función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivGetActionSkillRating**

```
int EE_CognitivGetActionSkillRating ( unsigned int userId,
                                      EE_CognitivAction_t action,
                                      float *pActionSkillRatingOut )
```

Devuelve el nivel de entrenamiento para una acción del modo cognitivo específica. La llamada a esta función bloquea el flujo de ejecución.

Como parámetro se pasa el identificador de usuario y la acción del modo cognitivo a consultar, mientras que en el tercer parámetro se devuelve por referencia el nivel de entrenamiento en un rango de valores entre 0.0 y 1.0.

La función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivGetActivationLevel**

```
int EE_CognitivGetActivationLevel ( unsigned int userId,
                                    int *pLevelOut )
```

Proporciona la sensibilidad conjunta de todas las acciones del modo cognitivo.

Recibe como parámetro el identificador de usuario y devuelve por referencia el nivel de sensibilidad en un rango de valores entre 1 y 10.

La función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivGetActiveActions**

```
int EE_CognitivGetActiveActions ( unsigned int userId,
                                  unsigned long *pActiveActionsOut )
```

Proporciona los tipos de acciones del modo cognitivo que se encuentran activas.

Recibe como parámetro el identificador de usuario y devuelve por referencia un vector compuesto por las constantes definidas en el tipo enumerado **EE_CognitivAction_t** (**EE_CognitivAction_enum**) como un mapa de bits.

La función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- EE_CognitivGetOverallSkillRating

```
int EE_CognitivGetOverallSkillRating ( unsigned int userId,
                                       float *pOverallSkillRatingOut )
```

Proporciona el nivel de entrenamiento en modo cognitivo conjunto del usuario. La llamada a esta función bloquea el flujo de ejecución.

Se le pasa como parámetro el identificador de usuario y devuelve por referencia el valor de entrenamiento conjunto en un rango entre 0.0 y 1.0.

La función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- EE_CognitivGetSignatureCacheSize

```
int EE_CognitivGetSignatureCacheSize ( unsigned int userId,
                                       unsigned int *pSizeOut )
```

Esta función obtiene el tamaño actual de la caché para la captura de firmas de detección en modo cognitivo.

Requiere como parámetro el identificador de usuario y devuelve por referencia el número de firmas de detección que serán mantenidas en la caché. Si este valor es 0 indicará que el espacio para las firmas es ilimitado.

La función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- EE_CognitivGetSignatureCaching

```
int EE_CognitivGetSignatureCaching ( unsigned int userId,
                                     unsigned int *pEnabledOut )
```

Consulta el estado de la captura de firmas de detección en modo cognitivo que podrá estar habilitado o inhabilitado.

Esta función toma como parámetros el identificador de usuario y devuelve por referencia el estado de la captura de firmas de detección indicando con un valor igual a 1 que se encuentra habilitado y con 0 que se encuentra inhabilitado.

La función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivGetTrainedSignatureActions**

```
int EE_CognitivGetTrainedSignatureActions ( unsigned int  userId,
                                            unsigned long *pTrainedActionsOut )
```

Obtiene la lista de las acciones del modo cognitivo que el usuario ha entrenado. La llamada a esta función bloquea el flujo de ejecución.

El parámetro de esta función se corresponde con el identificador de usuario y como salida se obtiene un vector compuesto por las constantes definidas en el tipo enumerado **EE_CognitivAction_t** (**EE_CognitivAction_enum**) como un mapa de bits.

La función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivGetTrainingAction**

```
int EE_CognitivGetTrainingAction ( unsigned int  userId,
                                   EE_CognitivAction_t *pActionOut )
```

Obtiene la acción cognitiva seleccionada para realizar el entrenamiento.

La función toma como parámetro el identificador de usuario y devuelve por referencia la acción como una de las constantes definidas en el tipo enumerado **EE_CognitivAction_t** (**EE_CognitivAction_enum**).

La función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivGetTrainingTime**

```
int EE_CognitivGetTrainingTime ( unsigned int userId,
                                 unsigned int *pTrainingTimeOut )
```

Esta función se utiliza para devolver la duración de la sesión de entrenamiento en modo cognitivo.

Recibe como parámetros el identificador de usuario y devuelve por referencia el tiempo de entrenamiento en milisegundos.

La función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivSetActionSensitivity**

```
int EE_CognitivSetActionSensitivity ( unsigned int userId,
                                      int action1Sensitivity,
                                      int action2Sensitivity,
                                      int action3Sensitivity,
                                      int action4Sensitivity )
```

Cambia la sensibilidad de las acciones del modo cognitivo.

Los parámetros para esta función son el identificador de usuario y las sensibilidades para las cuatro acciones del modo cognitivo seleccionadas en un rango entre 1 y 10.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivSetActivationLevel**

```
int EE_CognitivSetActivationLevel ( unsigned int userId,
                                    int level )
```

Cambia la sensibilidad conjunta de todas las acciones del modo cognitivo.

Toma como parámetros el identificador de usuario y el nivel de sensibilidad para todas las acciones, siendo 1 el menor y 7 el mayor.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivSetActiveActions**

```
int EE_CognitivSetActiveActions  ( unsigned int userId,
                                   unsigned long activeActions )
```

Cambia los tipos de acciones del modo cognitivo actualmente seleccionadas.

Esta función utiliza como parámetros el identificador de usuario y un vector de constantes del tipo enumerado **EE_CognitivAction_t** (**EE_CognitivAction_enum**) como un mapa de bits.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivSetSignatureCacheSize**

```
int EE_CognitivSetSignatureCacheSize ( unsigned int userId,
                                       unsigned int size )
```

Cambia el tamaño de la caché de captura de firmas de detección en modo cognitivo.

Los parámetros son el identificador de usuario y el nuevo tamaño para la caché, utilizándose el valor 0 para el tamaño ilimitado.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivSetSignatureCaching**

```
int EE_CognitivSetSignatureCaching ( unsigned int userId,
                                     unsigned int enabled )
```

Habilita/inhabilita la captura de firmas de detección en modo cognitivo.

Toma como parámetros el identificador de usuario y un 1 para habilitar la captura o un 0 para inhabilitarla.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivSetTrainingAction**

```
int EE_CognitivSetTrainingAction ( unsigned int userId,
                                   EE_CognitivAction_t action )
```

Selecciona el tipo de acción cognitiva que va a ser entrenada.

Esta función toma como parámetro el identificador de usuario y la acción perteneciente al tipo **EE_CognitivAction_t**.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivSetTrainingControl**

```
int EE_CognitivSetTrainingControl ( unsigned int userId,
                                    EE_CognitivTrainingControl_t control )
```

Asigna un valor de control para entrenamiento cognitivo.

Los parámetros requeridos son el identificador de usuario y el valor de control.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivStartSamplingNeutral**

```
int EE_CognitivStartSamplingNeutral ( unsigned int userId )
```

Comienza el muestreo del estado neuronal en el modo cognitivo.

Toma como parámetro el identificador de usuario

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_CognitivStopSamplingNeutral**

```
int EE_CognitivStopSamplingNeutral ( unsigned int userId )
```

Detiene el muestreo del estado neuronal en el modo cognitivo.

Toma como parámetro el identificador de usuario

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_EmoEngineEventCreate**

```
EmoEngineEventHandle EE_EmoEngineEventCreate (  )
```

Esta función devuelve un puntero a un espacio de memoria para contener un evento del **Motor Emotiv**. Este puntero podrá ser reutilizado para obtener los eventos que se producen durante la conexión.

- **EE_EmoEngineEventFree**

```
void EE_EmoEngineEventFree ( EmoEngineEventHandle hEvent )
```

Libera la memoria referenciada por el puntero que se le pasa como parámetro. Este puntero debe haber sido creado con anterioridad por las funciones **EE_EmoEngineEventCreate** o **EE_ProfileEventCreate** .

- **EE_EmoEngineEventGetEmoState**

```
int EE_EmoEngineEventGetEmoState ( EmoEngineEventHandle hEvent,
                                   EmoStateHandle hEmoState )
```

Copia un **Estado Emotiv** generado cuando se produce un evento **EE_EmoStateUpdate** y suministrado como primer parámetro en la memoria referenciada por el puntero pasado como segundo parámetro, que a su vez debe haber sido inicializado por una llamada a la función

EE_EmoStateCreate.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_EmoEngineEventGetType**

```
EE_Event_t EE_EmoEngineEventGetType ( EmoEngineEventHandle hEvent )
```

Devuelve el tipo del evento que ha sido obtenido utilizando la función **EE_EngineGetNextEvent**.

Como parámetro se le pasa un puntero de eventos previamente inicializado utilizando la función **EE_EmoEngineEventCreate**.

- **EE_EmoEngineEventGetUserId**

```
int EE_EmoEngineEventGetUserId ( EmoEngineEventHandle hEvent,
                                 unsigned int *pUserIdOut )
```

Obtiene el identificador de usuario desde los eventos **EE_UserAdded** y **EE_UserRemoved**.

Recibe como parámetros un puntero inicializado por la función **EE_EmoEngineEventCreate** y devuelve como referencia el identificador de usuario asociado al evento.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_EmoStateCreate**

```
EmoStateHandle EE_EmoStateCreate (  )
```

Devuelve un puntero a memoria para un **Estado Emotiv**. Este puntero puede ser reutilizado en sucesivas llamadas para obtener estados sucesivos.

- **EE_EmoStateFree**

```
void EE_EmoStateFree ( EmoStateHandle hState )
```

Libera la memoria referenciada por el puntero a **Estado Emotiv**.

Toma como parámetro el puntero inicializado por la función **EE_EmoStateCreate**.

- **EE_EnableDiagnostics**

```
int EE_EnableDiagnostics ( const char *szFilename,
                           int fEnable,
                           int nReserved )
```

Habilita/Inhabilita la salida de información de depuración. Por defecto se encuentra inactiva y sólo debería emplearse para fines de diagnóstico y soporte.

El primer parámetro es la ruta al fichero de registro y el segundo toma el valor 0 para inhabilitar el registro y un valor distinto de 0 para habilitarlo. El tercer parámetro no tiene función asignada por el momento.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_EngineClearEventQueue**

```
int EE_EngineClearEventQueue ( int eventTypes )
```

Elimina un determinado tipo de evento de la cola de eventos o bien vacía toda la cola. Los *flags* de eventos pueden ser combinados conjuntamente dentro del parámetro exceptuando a los valores **EE_UnknownEvent** y **EE_EmulatorError**.

El parámetro requerido es del tipo de evento perteneciente a **EE_Event_t (EE_Event_enum**), en donde podrán combinarse más de uno utilizando la función lógica **OR** a nivel de bits, como por ejemplo **EE_UserAdded** | **EE_UserRemoved**.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si los eventos han sido eliminados de la cola o **EDK_INVALID_PARAMETER** si el parámetro suministrado no es válido.

- **EE_EngineConnect**

```
int EE_EngineConnect (  )
```

Inicia la conexión con el **Motor Emotiv**.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha establecido la conexión.

- **EE_EngineDisconnect**

```
int EE_EngineDisconnect (  )
```

Finaliza la conexión con el **Motor Emotiv**.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha realizado la desconexión.

- **EE_EngineGetNextEvent**

```
int EE_EngineGetNextEvent ( EmoEngineEventHandle hEvent )
```

Obtiene el siguiente evento del **Motor Emotiv**. Esta llamada no bloquea el flujo de ejecución.

Como parámetro la función recibe un puntero inicializado previamente por la función **EE_EmoEngineEventCreate**.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha obtenido un nuevo evento o **EDK_NO_EVENT** si el **Motor Emotiv** no ha generado ninguno nuevo.

- **EE_EngineGetNumUser**

```
int EE_EngineGetNumUser ( unsigned int *pNumUserOut )
```

Obtiene el número de usuarios activos conectados al **Motor Emotiv**.

Esta función devuelve en su parámetro por referencia el número de usuarios.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_EngineRemoteConnect**

```
int EE_EngineRemoteConnect ( const char *szHost,
                             unsigned short port )
```

Inicia la conexión con una instancia remota el **Motor Emotiv**. La llamada a esta función bloquea el flujo de ejecución.

Los parámetros son el nombre o la dirección IP del servidor remoto y el puerto de escucha que para el **Panel de Control Emotiv** es el puerto 3008 y para el **EmoComposer** es el 1726.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha establecido la conexión.

- **EE_ExpressivEventGetType**

```
EE_ExpressivEvent_t EE_ExpressivEventGetType ( EmoEngineEventHandle hEvent )
```

Devuelve el tipo de un evento expresivo ya obtenido utilizando la función **EE_EngineGetNextEvent**.

Como parámetro se le pasa un puntero de eventos previamente inicializado utilizando la función **EE_EmoEngineEventCreate**.

- **EE_ExpressivGetSignatureType**

```
int EE_ExpressivGetSignatureType ( unsigned int userId,
                                   EE_ExpressivSignature_t *pSigTypeOut )
```

Indica si el modo expresivo está utilizando las firmas de detección universales o las que han sido entrenadas por el usuario. La llamada a esta función bloquea el flujo de ejecución.

Esta función toma como parámetros el identificador de usuario y devuelve por referencia en el segundo parámetro el tipo de firma en uso actualmente.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si la consulta ha tenido éxito.

- **EE_ExpressivGetThreshold**

```
int EE_ExpressivGetThreshold ( unsigned int userId,
                               EE_ExpressivAlgo_t algoName,
                               EE_ExpressivThreshold_t thresholdName,
                               int *pValueOut )
```

Obtiene el umbral de disparo para los algoritmos en modo expresivo.

Toma como parámetros el identificador de usuario, el tipo de algoritmo, el tipo de umbral de disparo para el modo expresivo y por último devuelve por referencia el valor del umbral.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- **EE_ExpressivGetTrainedSignatureActions**

```
int EE_ExpressivGetTrainedSignatureActions ( unsigned int userId,
                                             unsigned long *pTrainedActionsOut )
```

Obtiene un listado con las acciones que han sido entrenadas por el usuario. La llamada a esta función bloquea el flujo de ejecución.

Esta función toma como parámetros el identificador de usuario y devuelve como referencia un

vector compuesto por los bits que se corresponden con las constantes del tipo **EE_ExpressivAlgo_t** (**EE_ExpressivAlgo_enum**).

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si la consulta ha tenido éxito.

- **EE_ExpressivGetTrainedSignatureAvailable**

```
int EE_ExpressivGetTrainedSignatureAvailable ( unsigned int userId,
                                               int *pfAvailableOut )
```

Obtiene un indicador que informa si el usuario ha realizado suficientes acciones de entrenamiento para que se activen las firmas de detección. La llamada a esta función bloquea el flujo de ejecución.

Toma como parámetros el identificador de usuario y devuelve por referencia un valor que será distinto de 0 si el usuario ha entrenado la acción **EXP_NEUTRAL** y al menos otra acción del modo expresivo, en caso contrario el segundo parámetro tomará el valor 0.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si la consulta ha tenido éxito.

- **EE_ExpressivGetTrainingAction**

```
int EE_ExpressivGetTrainingAction ( unsigned int userId,
                                    EE_ExpressivAlgo_t *pActionOut )
```

Obtiene la expresión facial seleccionada actualmente para ser entrenada. La llamada a esta función bloquea el flujo de ejecución.

Esta función toma como parámetros el identificador de usuario y devuelve por referencia la expresión facial a entrenar.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si la consulta ha tenido éxito.

- **EE_ExpressivGetTrainingTime**

```
int EE_ExpressivGetTrainingTime ( unsigned int userId,
                                  unsigned int *pTrainingTimeOut )
```

Obtiene la duración de la sesión de entrenamiento en modo expresivo.

Recibe como parámetros el identificador de usuario y devuelve por referencia el tiempo de entrenamiento en milisegundos.

La función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- EE_ExpressivSetSignatureType

```
int EE_ExpressivSetSignatureType ( unsigned int userId,
                                   EE_ExpressivSignature_t sigType )
```

Modifica la configuración del modo expresivo para que utilice la firma universal de detección o bien una obtenida a través del entrenamiento realizado por el usuario. Por defecto la suite expresiva utiliza la firma universal. La llamada a esta función fallará si el valor devuelto por referencia por la función **EE_ExpressivGetTrainedSignatureAvailable** es 0. La llamada a esta función bloquea el flujo de ejecución.

Esta función recibe como parámetros el identificador de usuario y el tipo de firma a utilizar y devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si se ha ejecutado con éxito.

- EE_ExpressivSetThreshold

```
int EE_ExpressivSetThreshold ( unsigned int userId,
                               EE_ExpressivAlgo_t algoName,
                               EE_ExpressivThreshold_t thresholdName,
                               int value )
```

Modifica el umbral de disparo para los algoritmos del modo expresivo.

Recibe como parámetros el identificador de usuario, el nombre del algoritmo del modo expresivo, el tipo de umbral de disparo y el nuevo valor para el umbral en un rango entre 0 y 1000.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- EE_ExpressivSetTrainingAction

```
int EE_ExpressivSetTrainingAction ( unsigned int userId,
                                    EE_ExpressivAlgo_t action )
```

Modifica el tipo de expresión facial para el entrenamiento en modo expresivo. La llamada a esta función bloquea el flujo de ejecución.

Los parámetros que se pasan a esta función son el identificador de usuario y el tipo de expresión facial a entrenar.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK**

si el comando se ha ejecutado con éxito.

- **EE_ExpressivSetTrainingControl**

```
int EE_ExpressivSetTrainingControl ( unsigned int userId,
                                     EE_ExpressivTrainingControl_t control )
```

Activa el indicador de entrenamiento en modo expresivo. La llamada a esta función bloquea el flujo de ejecución.

Esta función toma como parámetros el identificador de usuario y la instrucción de control del tipo **EE_ExpressivTrainingControl_t** (**EE_ExpressivTrainingControl_enum**).

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_GetBaseProfile**

```
int EE_GetBaseProfile ( EmoEngineEventHandle hEvent )
```

Devuelve mediante una llamada síncrona el perfil del usuario por defecto en forma *serializada,* completando la referencia pasada como parámetro con un evento del tipo **EE_ProfileEvent** que contiene la información del perfil del usuario por defecto.

El parámetro que toma esta función es el puntero de eventos que inicializa la función **EE_EmoEngineEventCreate**.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_GetUserProfile**

```
int EE_GetUserProfile ( unsigned int userId,
                        EmoEngineEventHandle hEvent )
```

Devuelve mediante una llamada síncrona el perfil del usuario, completando la referencia pasada como parámetro con los datos del perfil del usuario que se especifica como primer parámetro.

El primer parámetro es el identificador de usuario y el segundo es el puntero a eventos que inicializa la función **EE_EmoEngineEventCreate**.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- EE_GetUserProfileBytes

```
int EE_GetUserProfileBytes ( EmoEngineEventHandle hEvt,
                             unsigned char destBuffer[],
                             unsigned int length )
```

Copia la versión *serializada* del perfil del usuario indicado en el *buffer* de destino.

Esta función toma como parámetros un puntero a eventos, un puntero al *buffer* destino y el tamaño del mismo en bytes.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- EE_GetUserProfileSize

```
int EE_GetUserProfileSize ( EmoEngineEventHandle hEvt,
                            unsigned int *pProfileSizeOut )
```

Obtiene el número de bytes requeridos para guardar la versión *serializada* del perfil de usuario.

Toma como parámetro un puntero a eventos devolviendo como referencia el tamaño del perfil en bytes.

Esta función devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- EE_HardwareGetVersion

```
int EE_HardwareGetVersion ( unsigned int userId,
                            unsigned long *pHwVersionOut )
```

Obtiene la versión del hardware del casco y receptor USB para un usuario determinado.

Como parámetros esta función recibe el identificador de usuario a consultar y devuelve por referencia las versiones hardware, encontrándose en la palabra (*word*) más significativa la versión del casco y en la menos significativa la del dispositivo receptor USB.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_HeadsetGetGyroDelta**

```
int EE_HeadsetGetGyroDelta ( unsigned int userId,
                             int *pXOut,
                             int *pYOut )
```

Obtiene el delta del movimiento del giróscopo desde la llamada anterior para un usuario específico.

Esta función toma como parámetro el identificador de usuario y devuelve por referencia el desplazamiento horizontal y el vertical.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si la consulta ha tenido éxito.

- **EE_HeadsetGetSensorDetails**

```
int EE_HeadsetGetSensorDetails ( EE_InputChannels_t channelId,
                                 InputSensorDescriptor_t *pDescriptorOut )
```

Obtiene una estructura que contiene los detalles sobre un canal de EEG específico del casco.

Esta función toma como parámetro el identificador del canal a consultar y devuelve por referencia la localización detallada del sensor junto a otra información extra.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si la consulta ha tenido éxito.

- **EE_HeadsetGyroRezero**

```
int EE_HeadsetGyroRezero ( unsigned int userId )
```

Pone a cero el giróscopo para el usuario especificado.

Esta función toma como parámetro el identificador de usuario.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_LoadUserProfile**

```
int EE_LoadUserProfile ( unsigned int userID,
                         const char *szInputFilename )
```

Esta función carga un perfil desde el disco y lo asigna al usuario especificado.

Toma como parámetro el identificador de usuario y el fichero en donde se encuentra almacenado el perfil.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el perfil se ha cargado con éxito.

- EE_OptimizationDisable

```
int EE_OptimizationDisable (  )
```

Inhabilita la optimización.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- EE_OptimizationEnable

```
int EE_OptimizationEnable ( OptimizationParamHandle hParam )
```

Habilita la optimización. El **Motor Emotiv** intentará optimizar su rendimiento de forma acorde a la información pasada en el parámetro de optimización, garantizando la exactitud de los resultados de los algoritmos esenciales. Para el resto de algoritmos los resultados no se encuentran definidos.

Esta función toma como parámetro el puntero inicializado por la función **EE_OptimizationParamCreate** y devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- EE_OptimizationGetParam

```
int EE_OptimizationGetParam ( OptimizationParamHandle hParam )
```

Obtiene el parámetro de optimización. Si la optimización no se encuentra habilitada (lo que se puede comprobar utilizando la función **EE_OptimizationIsEnabled**) entonces los resultados devueltos en el parámetro no se encuentran definidos.

Esta función recibe como parámetro el puntero inicializado por la función **EE_OptimizationParamCreate** y devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si la consulta se ha ejecutado con éxito.

- EE_OptimizationGetVitalAlgorithm

```
int EE_OptimizationGetVitalAlgorithm ( OptimizationParamHandle hParam,
                                       EE_EmotivSuite_t suite,
                                       unsigned int *pVitalAlgorithmBitVectorOut )
```

Obtiene una lista de los algoritmos esenciales para un modo de funcionamiento específico desde el parámetro de optimización.

A esta función se le pasa como parámetro un puntero inicializado por la función **EE_OptimizationParamCreate** y el modo a consultar, y devuelve por referencia el algoritmo vital compuesto por los bits de **EE_ExpressivAlgo_t (EE_ExpressivAlgo_enum)**, **EE_AffectivAlgo_t (EE_AffectivAlgo_enum)** o **EE_CognitivAction_t (EE_CognitivAction_enum)** según el modo de funcionamiento seleccionado.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si la consulta ha tenido éxito.

- EE_OptimizationIsEnabled

```
int EE_OptimizationIsEnabled ( bool *pEnabledOut )
```

Determina si la optimización se encuentra habilitada.

Esta función devuelve por referencia un valor booleano indicando el estado de la optimización y devuelve como resultado un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si la consulta ha tenido éxito.

- EE_OptimizationParamCreate

```
OptimizationParamHandle EE_OptimizationParamCreate (  )
```

Devuelve un puntero a una zona de memoria que puede contener un parámetro el cual será utilizado para configurar el comportamiento de las optimizaciones.

- EE_OptimizationParamFree

```
void EE_OptimizationParamFree ( OptimizationParamHandle hParam )
```

Libera la memoria referenciada por un puntero a parámetro de optimización.

Esta función recibe como parámetro el puntero devuelto por la función **EE_OptimizationParamCreate**.

- EE_OptimizationSetVitalAlgorithm

```
int EE_OptimizationSetVitalAlgorithm ( OptimizationParamHandle hParam,
                                       EE_EmotivSuite_t suite,
                                       unsigned int vitalAlgorithmBitVector )
```

Modifica la lista de algoritmos esenciales de un modo de funcionamiento específico para ser utilizado como parámetro de optimización.

Esta función recibe como parámetro el puntero devuelto por la función

EE_OptimizationParamCreate, el modo de funcionamiento y el algoritmo vital compuesto por los bits de los tipos **EE_ExpressivAlgo_t** (**EE_ExpressivAlgo_enum**), **EE_AffectivAlgo_t** (**EE_AffectivAlgo_enum**) o **EE_CognitivAction_t** (**EE_CognitivAction_enum**) dependiendo del modo de funcionamiento seleccionado.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_ProfileEventCreate**

```
EmoEngineEventHandle EE_ProfileEventCreate    (  )
```

Devuelve un puntero a una zona de memoria capaz de contener una cadena de bytes de un perfil. Este puntero podrá ser reutilizado posteriormente para obtener otras cadenas de perfil.

- **EE_ResetDetection**

```
int EE_ResetDetection ( unsigned int userId,
                        EE_EmotivSuite_t suite,
                        unsigned int detectionBitVector )
```

Inicializa todos los ajustes y los datos de perfil específicos del usuario para el modo de funcionamiento indicado.

Esta función toma como parámetros el identificador de usuario, el modo de funcionamiento (expresivo, afectivo o cognitivo) y un vector de bits que identifica una detección en particular. Un valor de 0 en este parámetro se corresponderá con todas las detecciones.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_SaveUserProfile**

```
int EE_SaveUserProfile ( unsigned int userID,
                         const char *szOutputFilename )
```

Guarda en disco el perfil para el usuario especificado.

Esta función recibe como parámetro un identificador de usuario y el fichero destino.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_SetHardwarePlayerDisplay**

```
int EE_SetHardwarePlayerDisplay ( unsigned int userId,
                                  unsigned int playerNum )
```

Modifica el número de "jugador" visualizado en el receptor USB que corresponde al identificador de usuario especificado como parámetro.

Esta función toma como parámetros el identificador de usuario del **Motor Emotiv** y el número de jugador que se visualiza en el dispositivo receptor USB asignado a la aplicación, el cual debe estar en el rango de 1 a 4.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_SetUserProfile**

```
int EE_SetUserProfile ( unsigned int userId,
                        const unsigned char profileBuffer[],
                        unsigned int length )
```

Carga un perfil del **Motor Emotiv** para el usuario especificado.

Esta función toma como parámetros el identificador de usuario, un puntero al *buffer* que contiene el perfil *serializado* del usuario previamente devuelto por el **Motor Emotiv** y el tamaño del *buffer* en bytes.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si el comando se ha ejecutado con éxito.

- **EE_SoftwareGetVersion**

```
int EE_SoftwareGetVersion ( char *pszVersionOut,
                            unsigned int nVersionChars,
                            unsigned long *pBuildNumOut )
```

Obtiene la versión actual del **Kit de Desarrollo Emotiv**.

Esta función devuelve en el *buffer* apuntado por el primer parámetro la versión del software en el formato X.X.X.X, teniendo las versiones actuales en fase beta como primer valor un 0. Como segundo parámetro se indica la longitud del *buffer* de caracteres apuntados por el primer parámetro y en el tercer parámetro se devuelve por referencia el número de la compilación el cual es único para cada versión.

Devuelve un valor de tipo **EDK_ERROR_CODE** que toma el valor **EDK_OK** si la consulta ha tenido éxito.

3.6.2 Funciones de estados Emotiv (EMOSTATE_DLL_API)

Los **Estados Emotiv** son generados por el **Motor de Detección Emotiv** y representan el estado emocional del usuario en un momento dado. Ninguna de estas funciones son *thread-safe*.

- ES_AffectivEqual

```
int ES_AffectivEqual ( EmoStateHandle a,
                       EmoStateHandle b )
```

Comprueba si dos **Estados Emotiv** son idénticos para el modo afectivo.

Esta función toma como parámetros dos punteros a **Estados Emotiv** y devuelve 1 si son iguales y 0 si son diferentes para el modo afectivo.

- ES_AffectivGetEngagementBoredomScore

```
float ES_AffectivGetEngagementBoredomScore ( EmoStateHandle state )
```

Obtiene el nivel de atención/aburrimiento del usuario.

Recibe como parámetro un puntero a **Estado Emotiv** y devuelve el nivel de atención/aburrimiento en un rango entre 0.0 a 1.0.

- ES_AffectivGetExcitementLongTermScore

```
float ES_AffectivGetExcitementLongTermScore ( EmoStateHandle state )
```

Obtiene el nivel de emoción a largo plazo del usuario.

Recibe como parámetro un puntero a **Estado Emotiv** y devuelve el nivel de emoción en un rango entre 0.0 y 1.0.

- ES_AffectivGetExcitementShortTermScore

```
float ES_AffectivGetExcitementShortTermScore ( EmoStateHandle state )
```

Obtiene el nivel de emoción a corto plazo del usuario.

Recibe como parámetro un puntero a **Estado Emotiv** y devuelve el nivel de emoción en un rango entre 0.0 y 1.0.

- ES_AffectivGetFrustrationScore

```
float ES_AffectivGetFrustrationScore ( EmoStateHandle state )
```

Obtiene el nivel de frustración del usuario.

Recibe como parámetro un puntero a **Estado Emotiv** y devuelve el nivel de frustración en un rango entre 0.0 y 1.0.

- ES_AffectivGetMeditationScore

```
float ES_AffectivGetMeditationScore ( EmoStateHandle state )
```

Obtiene el nivel de meditación del usuario.

Recibe como parámetro un puntero a **Estado Emotiv** y devuelve el nivel de meditación en un rango entre 0.0 y 1.0.

- ES_AffectivIsActive

```
int ES_AffectivIsActive ( EmoStateHandle state,
                          EE_AffectivAlgo_t type )
```

Comprueba que la señal no sea muy ruidosa de forma que impida la detección en modo afectivo.

Recibe como parámetros un puntero a **Estado Emotiv** y el tipo de detección en modo afectivo a comprobar.

Devuelve el estado de la detección, indicando un valor con 0 que no está activa y con un valor 1 que si lo está.

- ES_CognitivEqual

```
int ES_CognitivEqual ( EmoStateHandle a,
                       EmoStateHandle b )
```

Comprueba si dos **Estados Emotiv** son idénticos para el modo cognitivo.

Esta función toma como parámetros dos punteros a **Estados Emotiv** y devuelve 1 si son iguales y 0 si son diferentes para el modo cognitivo.

- ES_CognitivGetCurrentAction

```
EE_CognitivAction_t ES_CognitivGetCurrentAction ( EmoStateHandle state )
```

Obtiene la acción detectada en modo cognitivo del usuario.

Esta función toma como parámetro un puntero a **Estado Emotiv** y devuelve el tipo de acción en modo cognitivo.

- ES_CognitivGetCurrentActionPower

```
float ES_CognitivGetCurrentActionPower ( EmoStateHandle state )
```

Obtiene el nivel de potencia de la acción en modo cognitivo detectada del usuario.

Esta función toma como parámetro un puntero a **Estado Emotiv** y devuelve el nivel de potencia de la acción en modo cognitivo.

- ES_CognitivIsActive

```
int ES_CognitivIsActive ( EmoStateHandle state )
```

Comprueba que la señal no sea muy ruidosa de forma que impida la detección en modo cognitivo.

Recibe como parámetro un puntero a **Estado Emotiv** y devuelve el estado de la detección, indicando con un valor 0 que no está activa y con un valor 1 que si lo está.

- ES_Copy

```
void ES_Copy ( EmoStateHandle a,
               EmoStateHandle b )
```

Duplica un **Estado Emotiv**. El primer parámetro es el puntero al **Estado Emotiv** destino y el segundo el puntero al **Estado Emotiv** origen.

- ES_Create

```
EmoStateHandle ES_Create (  )
```

Devuelve un puntero a memoria para contener un **Estado Emotiv**.

Nota: Esta función es obsoleta. Se debe utilizar **EE_EmoStateCreate** en su lugar.

Luego de la creación del puntero a **Estado Emotiv** se realiza una llamada a la función **ES_Init** de forma automática. La función **ES_Free** debe llamarse para liberar recursos antes de la destrucción de este puntero.

- ES_EmoEngineEqual

```
int ES_EmoEngineEqual ( EmoStateHandle a,
                        EmoStateHandle b )
```

Comprueba si dos **Estados Emotiv** son idénticos para el modo emotivo. Esta función compara el tiempo transcurrido desde que el **Motor Emotiv** comenzó a ejecutarse, el nivel de la señal inalámbrica y la calidad de contacto de cada uno de los canales.

- ES_Equal

```
int ES_Equal ( EmoStateHandle a,
               EmoStateHandle b )
```

Comprueba si dos punteros a **Estado Emotiv** son idénticos.

Esta función toma como parámetros dos punteros a **Estados Emotiv** y devuelve 1 si son iguales y 0 si son diferentes.

- ES_ExpressivEqual

```
int ES_ExpressivEqual ( EmoStateHandle a,
                        EmoStateHandle b )
```

Comprueba si dos **Estados Emotiv** son idénticos para el modo expresivo, por ejemplo si representan a la misma expresión facial.

Esta función toma como parámetros dos punteros a **Estados Emotiv** y devuelve 1 si son iguales y 0 si son diferentes para el modo expresivo.

- ES_ExpressivGetClenchExtent

```
float ES_ExpressivGetClenchExtent ( EmoStateHandle state )
```

Obtiene el grado en que el usuario aprieta los dientes (función obsoleta).

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve el grado de la expresión facial en un rango entre 0.0 y 1.0.

- ES_ExpressivGetEyebrowExtent

```
float ES_ExpressivGetEyebrowExtent ( EmoStateHandle state )
```

Obtiene el grado en que el usuario mueve la frente (función obsoleta).

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve el grado de la expresión facial en un rango entre 0.0 y 1.0.

- ES_ExpressivGetEyelidState

```
void ES_ExpressivGetEyelidState ( EmoStateHandle state,
                                  float *leftEye,
                                  float *rightEye )
```

Consulta el estado de los párpados del usuario. El estado de los párpados izquierdo y derecho se devuelven en el correspondiente parámetro con valores entre 0.0 y 1.0. Un valor 0.0 indica que el párpado se encuentra totalmente abierto, mientras que un valor de 1.0 representa un

párpado totalmente cerrado.

Esta función recibe como parámetro un puntero a un **Estado Emotiv** y devuelve por referencia el estado del párpado izquierdo y del derecho en un rango de valores entre 0.0 y 1.0.

- ES_ExpressivGetEyeLocation

```
void ES_ExpressivGetEyeLocation ( EmoStateHandle state,
                                  float *x,
                                  float *y )
```

Consulta la posición de los ojos del usuario.

La posición horizontal y vertical de los ojos del usuario se almacenan en el parámetro x e y respectivamente como valores en un rango entre -1.0 y 1.0. Esta función asume que ambos ojos tienen la misma posición horizontal y vertical.

Una posición horizontal de -1.0 indica que el usuario se encuentra mirando hacia la izquierda mientras que una posición horizontal de 1.0 indica que se encuentra mirando hacia la derecha.

Una posición vertical de -1.0 indica que el usuario se encuentra mirando hacia abajo mientras que una posición vertical de 1.0 indica que se encuentra mirando hacia arriba.

Esta función recibe como parámetro un puntero a un **Estado Emotiv** y devuelve por referencia la posición horizontal y vertical de los ojos.

- ES_ExpressivGetLowerFaceAction

```
EE_ExpressivAlgo_t ES_ExpressivGetLowerFaceAction ( EmoStateHandle state )
```

Obtiene la acción facial en modo expresivo detectada correspondiente a la región inferior del rostro del usuario.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un tipo de acción facial entre los predefinidos.

- ES_ExpressivGetLowerFaceActionPower

```
float ES_ExpressivGetLowerFaceActionPower ( EmoStateHandle state )
```

Obtiene el nivel de potencia de la acción facial en modo expresivo de la región inferior del rostro del usuario.

Esta función toma como parámetro un puntero a un **Estado Emotiv** y devuelve el nivel de potencia de la acción facial en modo expresivo en un rango entre 0.0 y 1.0.

- ES_ExpressivGetSmileExtent

```
float ES_ExpressivGetSmileExtent ( EmoStateHandle state )
```

Devuelve el grado en que el usuario sonríe (función obsoleta).

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve el grado de la expresión facial en un rango entre 0.0 y 1.0.

- ES_ExpressivGetUpperFaceAction

```
EE_ExpressivAlgo_t ES_ExpressivGetUpperFaceAction ( EmoStateHandle state )
```

Obtiene la acción facial en modo expresivo detectada correspondiente a la región superior del rostro del usuario.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un tipo de acción facial entre los predefinidos.

- ES_ExpressivGetUpperFaceActionPower

```
float ES_ExpressivGetUpperFaceActionPower ( EmoStateHandle state )
```

Obtiene el nivel de potencia de la acción facial en modo expresivo de la región superior del rostro del usuario.

Esta función toma como parámetro un puntero a un **Estado Emotiv** y devuelve el nivel de potencia de la acción facial en modo expresivo en un rango entre 0.0 y 1.0.

- ES_ExpressivIsActive

```
int ES_ExpressivIsActive ( EmoStateHandle state,
                           EE_ExpressivAlgo_t type )
```

Comprueba que la señal no sea muy ruidosa de forma que impida la detección en modo expresivo.

Recibe como parámetros un puntero a un **Estado Emotiv** y el tipo de detección en modo expresivo a comprobar.

Devuelve el estado de la detección, indicando un valor con 0 que no está activa y con un valor 1 que si lo está.

- **ES_ExpressivIsBlink**

```
int ES_ExpressivIsBlink ( EmoStateHandle state )
```

Consulta si el usuario se encontraba parpadeando en el momento en que el **Estado Emotiv** fue capturado.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un 1 en caso de parpadeo y un 0 en caso contrario.

- **ES_ExpressivIsEyesOpen**

```
int ES_ExpressivIsEyesOpen ( EmoStateHandle state )
```

Consulta si los ojos del usuario se encontraban abiertos en el momento en que el **Estado Emotiv** fue capturado.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un 1 si los ojos se encontraban abiertos y un 0 en caso contrario.

- **ES_ExpressivIsLeftWink**

```
int ES_ExpressivIsLeftWink ( EmoStateHandle state )
```

Consulta si el usuario se encontraba guiñando el ojo izquierdo en el momento en que el **Estado Emotiv** fue capturado.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un 1 si el ojo izquierdo se encontraba guiñado y un 0 en caso contrario.

- **ES_ExpressivIsLookingDown**

```
int ES_ExpressivIsLookingDown ( EmoStateHandle state )
```

Consulta si el usuario se encontraba mirando hacia abajo en el momento en que el **Estado Emotiv** fue capturado.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un 1 si miraba hacia abajo y un 0 en caso contrario.

- **ES_ExpressivIsLookingLeft**

```
int ES_ExpressivIsLookingLeft ( EmoStateHandle state )
```

Consulta si el usuario se encontraba mirando hacia la izquierda en el momento en que el **Estado Emotiv** fue capturado.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un 1 si miraba hacia la izquierda y un 0 en caso contrario.

- **ES_ExpressivIsLookingRight**

```
int ES_ExpressivIsLookingRight ( EmoStateHandle state )
```

Consulta si el usuario se encontraba mirando hacia la derecha en el momento en que el **Estado Emotiv** fue capturado.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un 1 si miraba hacia la derecha y un 0 en caso contrario.

- **ES_ExpressivIsLookingUp**

```
int ES_ExpressivIsLookingUp ( EmoStateHandle state )
```

Consulta si el usuario se encontraba mirando hacia arriba en el momento en que el **Estado Emotiv** fue capturado.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un 1 si miraba hacia arriba y un 0 en caso contrario.

- **ES_ExpressivIsRightWink**

```
int ES_ExpressivIsRightWink ( EmoStateHandle state )
```

Consulta si el usuario se encontraba guiñando el ojo derecho en el momento en que el **Estado Emotiv** fue capturado.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un 1 si el ojo derecho se encontraba guiñado y un 0 en caso contrario.

- **ES_Free**

```
void ES_Free ( EmoStateHandle state )
```

Libera la memoria apuntada por el puntero de un **Estado Emotiv**.

Nota: Esta función es obsoleta. Se debe utilizar **EE_EmoStateFree** en su lugar.

Recibe como parámetro un puntero a un **Estado Emotiv** previamente inicializado por la función **ES_Create**.

- **ES_GetBatteryChargeLevel**

```
void ES_GetBatteryChargeLevel ( EmoStateHandle state,
                                int *chargeLevel,
                                int *maxChargeLevel )
```

Obtiene el nivel de carga de la batería del casco.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve por referencia el nivel actual de carga de la batería y el nivel máximo que esta puede alcanzar.

- **ES_GetContactQuality**

```
EE_EEG_ContactQuality_t ES_GetContactQuality ( EmoStateHandle state,
                                               int electroIdx )
```

Consulta la calidad del contacto de un electrodo de EEG específico.

Recibe como parámetros un puntero a un **Estado Emotiv**, el índice del electrodo a consultar y devuelve el valor enumerado que caracteriza la calidad del contacto del electrodo de EEG para el canal especificado.

- **ES_GetContactQualityFromAllChannels**

```
int ES_GetContactQualityFromAllChannels ( EmoStateHandle state,
                                          EE_EEG_ContactQuality_t *contactQuality,
                                          size_t numChannels )
```

Consulta la calidad de contacto para todos los electrodos en una sola llamada.

La calidad de los contactos se guardará en el vector pasado a la función. El valor devuelto en el primer elemento es idéntico al devuelto por la llamada a la función **ES_GetContactQuality(state, 0)**, el valor devuelto en el segundo elemento es idéntico al devuelto por la llamada a la función **ES_GetContactQuality(state, 1)**, etc. El orden del vector es consistente con el de los canales lógicos de entrada en la enumeración **EE_InputChannels_enum**.

Recibe como parámetros un puntero a un **Estado Emotiv**, un puntero a un vector de tamaño igual al número de canales y el tamaño de éste en cantidad de números en coma flotante (*floats*) y devuelve el número de valores de calidad de contactos copiados dentro del vector.

- **ES_GetHeadsetOn**

```
int ES_GetHeadsetOn ( EmoStateHandle state )
```

Comprueba que el casco se encuentre colocado correctamente.

Si el casco no se encuentra colocado correctamente en la cabeza del usuario no se obtendrá ningún valor de calidad de señal para cada uno de los electrodos.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve un 1 para indicar si el casco se encuentra bien colocado y un 0 en caso contrario.

- ES_GetNumContactQualityChannels

```
int ES_GetNumContactQualityChannels ( EmoStateHandle state )
```

Comprueba el número de canales con datos de calidad en los contactos de los sensores disponible.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve el número de canales con disponibilidad de información de la calidad de contacto.

- ES_GetTimeFromStart

```
float ES_GetTimeFromStart ( EmoStateHandle state )
```

Obtiene el tiempo transcurrido desde el inicio de la conexión entre **Motor Emotiv** y el casco.

Si se produce la desconexión entre el casco y el **Motor Emotiv** debido a un nivel bajo de batería o a la debilidad de la señal inalámbrica, el tiempo de conexión se inicializará a cero.

Recibe como parámetro un puntero a un **Estado Emotiv** y devuelve el tiempo en segundos.

- ES_GetWirelessSignalStatus

```
EE_SignalStrength_t ES_GetWirelessSignalStatus ( EmoStateHandle state )
```

Obtiene la fuerza de la señal inalámbrica.

Recibe como parámetro un puntero a un **Estado Emotiv** y la fuerza de la señal inalámbrica como uno de los valores de la enumeración **EE_SignalStrength_t** (**EE_SignalStrength_enum**).

- ES_Init

```
void ES_Init ( EmoStateHandle state )
```

Inicializa el **Estado Emotiv** asignándole un estado neutral.

Recibe como parámetro un puntero a un **Estado Emotiv**.

3.6.3 Estructuras del Kit de Desarrollo Emotiv

- InputSensorDescriptor_struct

```
typedef struct InputSensorDescriptor_struct {
      EE_InputChannels_t channelId; // logical channel id
      int                fExists;   // does this sensor exist on this headset model
      const char*        pszLabel;  // text label identifying this sensor
      double             xLoc;      // x coordinate from center of head towards nose
      double             yLoc;      // y coordinate from center of head towards ears
      double             zLoc;      // z coordinate from center of head toward top of
                                    // skull
} InputSensorDescriptor_t;
```

Estructura que contiene la información de cada sensor o electrodo.

3.6.4 Enumeraciones del Kit de Desarrollo Emotiv

- EE_CognitivEvent_enum

```
typedef enum EE_CognitivEvent_enum {
      EE_CognitivNoEvent = 0,
      EE_CognitivTrainingStarted,
      EE_CognitivTrainingSucceeded,
      EE_CognitivTrainingFailed,
      EE_CognitivTrainingCompleted,
      EE_CognitivTrainingDataErased,
      EE_CognitivTrainingRejected,
      EE_CognitivTrainingReset,
      EE_CognitivAutoSamplingNeutralCompleted,
      EE_CognitivSignatureUpdated
} EE_CognitivEvent_t;
```

Tipos de eventos específicos del modo cognitivo.

- EE_CognitivTrainingControl_enum

```
typedef enum EE_CognitivTrainingControl_enum {
      COG_NONE = 0,
      COG_START,
      COG_ACCEPT,
      COG_REJECT,
      COG_ERASE,
      COG_RESET
} EE_CognitivTrainingControl_t;
```

Tipos de control para el entrenamiento en modo cognitivo.

- **EE_Event_enum**

```
typedef enum EE_Event_enum {
       EE_UnknownEvent         = 0x0000,
       EE_EmulatorError        = 0x0001,
       EE_ReservedEvent        = 0x0002,
       EE_UserAdded            = 0x0010,
       EE_UserRemoved          = 0x0020,
       EE_EmoStateUpdated      = 0x0040,
       EE_ProfileEvent         = 0x0080,
       EE_CognitivEvent        = 0x0100,
       EE_ExpressivEvent       = 0x0200,
       EE_InternalStateChanged = 0x0400,
       EE_AllEvent             = EE_UserAdded | EE_UserRemoved | EE_EmoStateUpdated |
                                 EE_ProfileEvent | EE_CognitivEvent |
                                 EE_ExpressivEvent | EE_InternalStateChanged
} EE_Event_t;
```

Tipos de eventos del **Motor Emotiv**.

- **EE_ExpressivEvent_enum**

```
typedef enum EE_ExpressivEvent_enum {
       EE_ExpressivNoEvent = 0,
       EE_ExpressivTrainingStarted,
       EE_ExpressivTrainingSucceeded,
       EE_ExpressivTrainingFailed,
       EE_ExpressivTrainingCompleted,
       EE_ExpressivTrainingDataErased,
       EE_ExpressivTrainingRejected,
       EE_ExpressivTrainingReset
} EE_ExpressivEvent_t;
```

Tipos de eventos específicos del modo expresivo.

- **EE_ExpressivSignature_enum**

```
typedef enum EE_ExpressivSignature_enum {
       EXP_SIG_UNIVERSAL = 0,
       EXP_SIG_TRAINED
} EE_ExpressivSignature_t;
```

Tipos de firmas para el modo de funcionamiento expresivo.

- **EE_ExpressivThreshold_enum**

```
typedef enum EE_ExpressivThreshold_enum {
       EXP_SENSITIVITY
} EE_ExpressivThreshold_t;
```

Tipos de umbrales de disparo para el modo de funcionamiento expresivo.

- **EE_ExpressivTrainingControl_enum**

```
typedef enum EE_ExpressivTrainingControl_enum {
	EXP_NONE = 0,
	EXP_START,
	EXP_ACCEPT,
	EXP_REJECT,
	EXP_ERASE,
	EXP_RESET
} EE_ExpressivTrainingControl_t;
```

Tipos de controles de entrenamiento para el modo de funcionamiento expresivo.

3.6.5 Enumeraciones de la gestión de estados Emotiv

- **EE_AffectivAlgo_enum**

```
typedef enum EE_AffectivAlgo_enum {
	AFF_EXCITEMENT         = 0x0001,
	AFF_MEDITATION         = 0x0002,
	AFF_FRUSTRATION        = 0x0004,
	AFF_ENGAGEMENT_BOREDOM = 0x0008
} EE_AffectivAlgo_t;
```

Tipos de emociones para el modo afectivo.

- **EE_CognitivAction_enum**

```
typedef enum EE_CognitivAction_enum {
	COG_NEUTRAL                  = 0x0001,
	COG_PUSH                     = 0x0002,
	COG_PULL                     = 0x0004,
	COG_LIFT                     = 0x0008,
	COG_DROP                     = 0x0010,
	COG_LEFT                     = 0x0020,
	COG_RIGHT                    = 0x0040,
	COG_ROTATE_LEFT              = 0x0080,
	COG_ROTATE_RIGHT             = 0x0100,
	COG_ROTATE_CLOCKWISE         = 0x0200,
	COG_ROTATE_COUNTER_CLOCKWISE = 0x0400,
	COG_ROTATE_FORWARDS          = 0x0800,
	COG_ROTATE_REVERSE           = 0x1000,
	COG_DISAPPEAR                = 0x2000
} EE_CognitivAction_t;
```

Tipos de acciones para el modo cognitivo

- **EE_EEG_ContactQuality_enum**

```
typedef enum EE_EEG_ContactQuality_enum {
	EEG_CQ_NO_SIGNAL,
	EEG_CQ_VERY_BAD,
	EEG_CQ_POOR,
	EEG_CQ_FAIR,
	EEG_CQ_GOOD
} EE_EEG_ContactQuality_t;
```

Tipos para la calidad de contacto de los electrodos de EEG.

Caracteriza la calidad de recepción del EEG para un sensor del casco. No confundir con la potencia de la señal inalámbrica la cual hace referencia al enlace radial entre el transmisor en el casco y el receptor del dispositivo USB.

- **EE_EmotivSuite_enum**

```
typedef enum EE_EmotivSuite_enum {
	EE_EXPRESSIV = 0,
	EE_AFFECTIV,
	EE_COGNITIV
} EE_EmotivSuite_t;
```

Tipos de modos de detección.

- **EE_ExpressivAlgo_enum**

```
typedef enum EE_ExpressivAlgo_enum {
	EXP_NEUTRAL     = 0x0001,
	EXP_BLINK       = 0x0002,
	EXP_WINK_LEFT   = 0x0004,
	EXP_WINK_RIGHT  = 0x0008,
	EXP_HORIEYE     = 0x0010,
	EXP_EYEBROW     = 0x0020,
	EXP_FURROW      = 0x0040,
	EXP_SMILE       = 0x0080,
	EXP_CLENCH      = 0x0100,
	EXP_LAUGH       = 0x0200,
	EXP_SMIRK_LEFT  = 0x0400,
	EXP_SMIRK_RIGHT = 0x0800
} EE_ExpressivAlgo_t;
```

Tipos de expresiones faciales para el modo expresivo.

- **EE_InputChannels_enum**

```
typedef enum EE_InputChannels_enum {
	EE_CHAN_CMS = 0,
	EE_CHAN_DRL,
	EE_CHAN_FP1,
	EE_CHAN_AF3,
	EE_CHAN_F7,
	EE_CHAN_F3,
	EE_CHAN_FC5,
	EE_CHAN_T7,
	EE_CHAN_P7,
	EE_CHAN_O1,
	EE_CHAN_O2,
	EE_CHAN_P8,
	EE_CHAN_T8,
	EE_CHAN_FC6,
	EE_CHAN_F4,
	EE_CHAN_F8,
	EE_CHAN_AF4,
	EE_CHAN_FP2
} EE_InputChannels_t;
```

Identificadores para cada canal de entrada lógico

Nota: El número de canales no es necesariamente igual al número de electrodos en el casco. Además la potencia de señal y datos de entrada para algunos sensores son idénticos: CMS = DRL, FP1 = AF3 y FP2 = AF4

- EE_SignalStrength_enum

```
typedef enum EE_SignalStrength_enum {
      NO_SIGNAL = 0,
      BAD_SIGNAL,
      GOOD_SIGNAL
} EE_SignalStrength_t;
```

Niveles de señal inalámbrica.

3.6.6 Códigos de Error

EDK_BUFFER_TOO_SMALL

El *buffer* provisto a la función no tiene el tamaño suficiente.

EDK_CANNOT_ACQUIRE_DATA

El **Motor Emotiv** no es capaz de obtener datos de EEG para ser procesados.

EDK_COG_EXCESS_MAX_ACTIONS

El vector de bits de acción utilizado contiene más tipos de acciones que las permitidas.

EDK_COG_INVALID_ACTIVE_ACTION

Uno de los campos en el vector de bits de acción utilizado no es válido.

EDK_COG_INVALID_TRAINING_ACTION

La acción de entrenamiento utilizada no se encuentra entre las acciones de entrenamiento esperadas.

EDK_COG_INVALID_TRAINING_CONTROL

El control de entrenamiento utilizado no se encuentra en la lista de los controles de entrenamiento esperados.

EDK_EMOENGINE_DISCONNECTED

Se ha perdido la conexión con una instancia remota del **Motor Emotiv** realizada mediante la función **EE_EngineRemoteConnect**.

EDK_EMOENGINE_PROXY_ERROR

No se ha podido establecer la conexión con la instancia remota del **Motor Emotiv**.

EDK_EMOENGINE_UNINITIALIZED

El **Motor Emotiv** necesita ser inicializado utilizando la función **EE_EngineConnect** o **EE_EngineRemoteConnect**.

EDK_EXP_NO_SIG_AVAILABLE

La firma de detección no se encuentra disponible para ser utilizada. Puede ser necesaria la adición de acciones incluyendo la neutral.

EDK_FILESYSTEM_ERROR

Ha ocurrido un error en el sistema de ficheros que impide que la función finalice su ejecución con éxito.

EDK_GYRO_NOT_CALIBRATED

El giróscopo no se encuentra calibrado. Es necesario que el usuario se mantenga quieto al menos por medio segundo.

EDK_INVALID_PARAMETER

Uno de los parámetros suministrados a la función no es válido.

EDK_INVALID_PROFILE_ARCHIVE

Los contenidos del *buffer* suministrado a la función **EE_SetUserProfile** no son válidos como perfil *serializado* del **Motor Emotiv**.

EDK_INVALID_USER_ID

El identificador de usuario suministrado a la función no es válido.

EDK_NO_EVENT

En este momento no hay nuevos eventos del **Motor Emotiv**.

EDK_NO_USER_FOR_BASEPROFILE

La función **EE_EmoEngineEventGetUserId** devuelve este valor si el evento suministrado contiene un perfil básico que no se encuentra asociado a un usuario en particular.

EDK_OK

Valor por defecto que indica una ejecución exitosa.

EDK_OPTIMIZATION_IS_ON

La operación ha fallado debido a la optimización.

EDK_OUT_OF_RANGE

Uno de los parámetros suministrados a la función se encuentra fuera de rango.

EDK_PARAMETER_LOCKED

El valor del parámetro se encuentra bloqueado por un proceso de detección y no puede ser modificado en este momento.

EDK_RESERVED1

Valor de retorno reservado.

EDK_UNKNOWN_ERROR

Ha ocurrido un error interno.

Capítulo 4

Desarrollo Software

El objetivo del programa a desarrollar es el de ser utilizado como vínculo entre la **API** contenida dentro del **Kit de Desarrollo Emotiv** y la placa de interfaz que irá conectada al puerto paralelo (Figura 39).

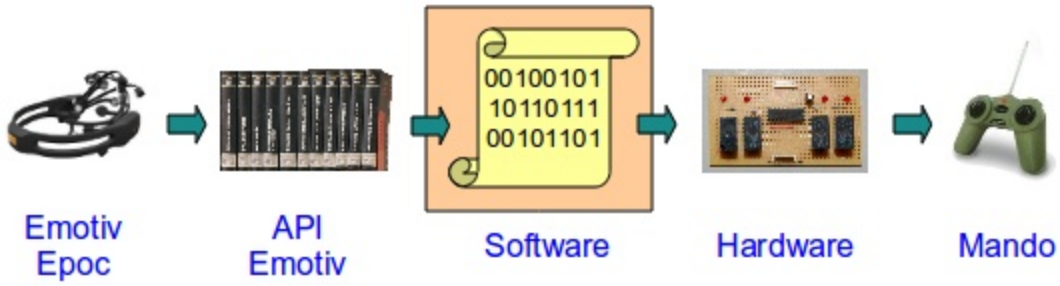


Figura 39: Desarrollo Software

Además deberá permitir operar el vehículo manualmente, conectar tanto con el **Motor Emotiv** como con el programa **EmoComposer**, monitorizar el estado del casco además de procesar las acciones detectadas y registrar los eventos recibidos pudiéndose guardar dicho registro en un fichero en el disco.

Adicionalmente será necesario poder realizar ajustes en los umbrales de activación de las acciones así como en la sensibilidad de las mismas, calibrar el acelerómetro del giróscopo, cargar un perfil de entrenamiento para el usuario activo y visualizar el manual de ayuda así como la ventana "Acerca de..." con información de la aplicación.

Las bibliotecas provistas dentro de la **API** se encuentran disponible tanto en C, como en C#. Se han escogido por afinidad las primeras y como entorno de desarrollo, C++Builder[22], este último especialmente por las facilidades que ofrece para desarrollar aplicaciones *win32* utilizando las bibliotecas VCL[23].

Para controlar el puerto paralelo fue necesario emplear la biblioteca específica **inpout32** la cual es gratuita para uso no comercial, y que permite a la aplicación acceder al mismo ya que a

22 C++Builder es un producto de Embarcadero Technologies, Inc.
23 *Visual Component Library* es un marco de trabajo desarrollado por Borland, orientado a objetos, basado en componentes visuales para desarrollar aplicaciones de Microsoft Windows. [29]

partir de Microsoft Windows NT no es posible modificar los registros del puerto paralelo directamente desde los programas que se ejecutan en modo usuario[24]. Esta biblioteca se encuentra disponible en el sitio web http://logix4u.net/ distribuyéndose compilada en el fichero *inpout32.dll* junto con el código fuente y ficheros de ejemplo.

A continuación se describirá la funcionalidad del programa mediante los Diagramas de Casos de Uso, se abordará el diseño arquitectónico utilizando los Diagramas de Clases y los Diagramas de Estados, se explicará como interactúan los objetos mediante los Diagramas de Secuencia y Colaboración y por último se ahondará en el funcionamiento del bucle principal por medio de los Diagramas de Actividad.

4.1 Diagramas de Casos de Uso

En la Figura 40 se observa el caso de uso que describe las interacciones del usuario con el **Motor Emotiv** y con la interfaz hardware a través del puerto paralelo. Las operaciones disponibles para el usuario desde la interfaz gráfica del programa son:

- **Ajustar los umbrales de disparo** para la detección de las acciones: avanzar, retroceder, girar a la izquierda y girar a la derecha.
- **Conectar con el Motor Emotiv o con el programa emulador EmoComposer**, en este último caso será necesario especificar la dirección IP y el puerto configurados en el emulador.
- **Calibrar el acelerómetro** mejorando la precisión en la detección de los giros de la cabeza del usuario. Éste deberá permanecer inmóvil durante medio segundo mientras se realiza la calibración.
- **Cargar el perfil de usuario** desde un fichero previamente guardado en disco. Los perfiles se obtendrán realizando los entrenamientos apropiados utilizando el **Motor Emotiv**.
- **Ajustar las sensibilidades** de detección de las acciones entrenadas dentro del perfil cargado.
- **Operación manual** del vehículo utilizando el ratón para ejecutar los movimientos de avance y retroceso así como para fijar la posición de las ruedas para realizar un giro hacia la izquierda o hacia la derecha.

Utilizando el casco **Emotiv Epoc**, se podrá **operar mentalmente** el vehículo realizando las

24 De allí la necesidad de utilizar esta biblioteca que se ejecuta en modo *kernel* como si de un *driver* se tratase.

cuatro acciones antes enumeradas.

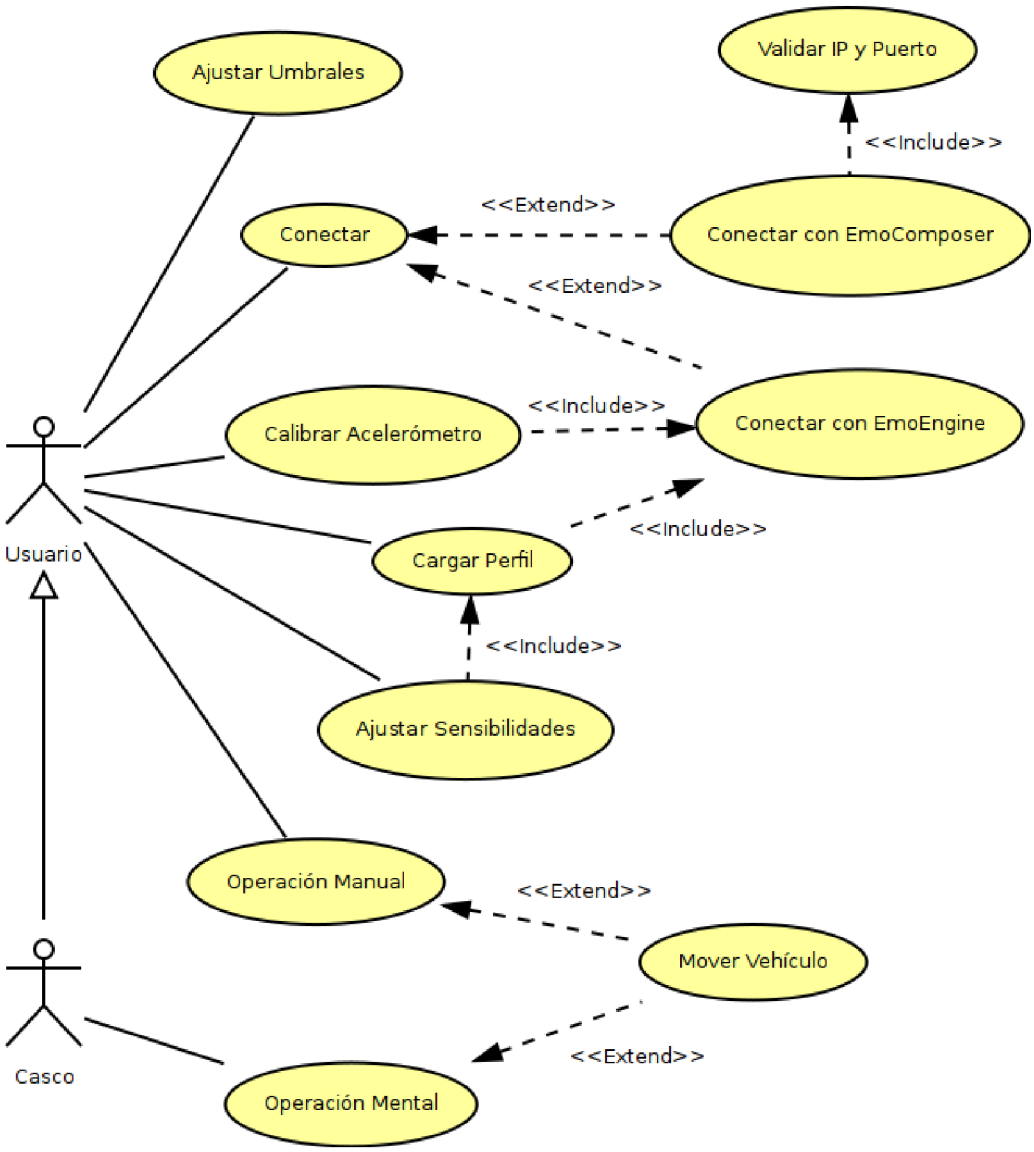


Figura 40: Caso de Uso: Interacciones con el Motor Emotiv y la interfaz

En la Figura 41 se representa el caso de uso correspondiente a las operaciones posibles sobre el registro del programa. El registro consiste en un área de texto en el cual se vuelcan los mensajes del programa, los errores y el contenido de cada estado recibido desde el casco. De este modo el usuario podrá:

- **Borrar el registro**, vaciando el espacio de texto de modo que se descarten los mensajes antiguos.
- **Guardar el registro** en disco para preservar los estados y eventos recibidos ya que al cerrar la aplicación el contenido del espacio de texto se pierde.

A través de los eventos recibidos desde el casco **Emotiv Epoc** se producirá el **volcado del estado** contenido en cada uno de ellos en el registro de la aplicación.

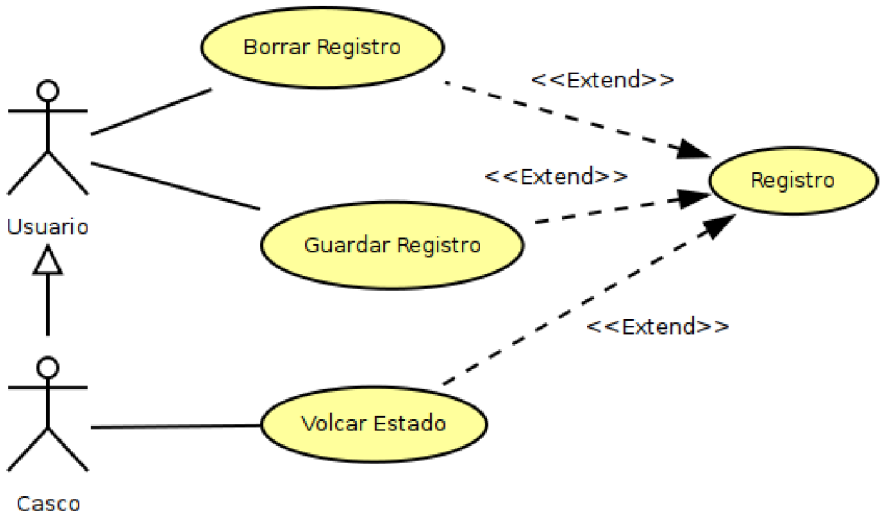


Figura 41: Caso de uso: Operaciones sobre el registro

Por último en la Figura 42 se presenta el caso de uso que se corresponde con las opciones de **visualizar el manual de ayuda** o la **ventana "Acerca de..."**.

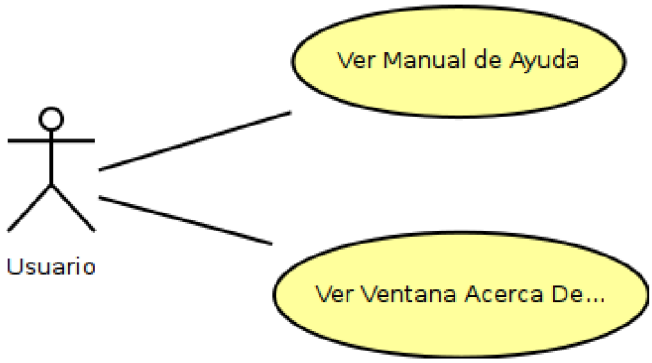


Figura 42: Caso de uso: Visualización de Ayuda y ventana Acerca de...

4.2 Diagramas de Clases

Habiendo definido en el punto anterior las acciones que se pueden realizar sobre el programa, ahora se procederá a describir el diseño arquitectónico del mismo.

En la Figura 43 se visualiza el diagrama de clases del diseño en el que se pueden observar las extensiones y agregaciones entre las distintas clases que componen el programa (véase en la Figura 80 dentro del Apéndice A, el diagrama completo con los atributos y métodos de cada clase).

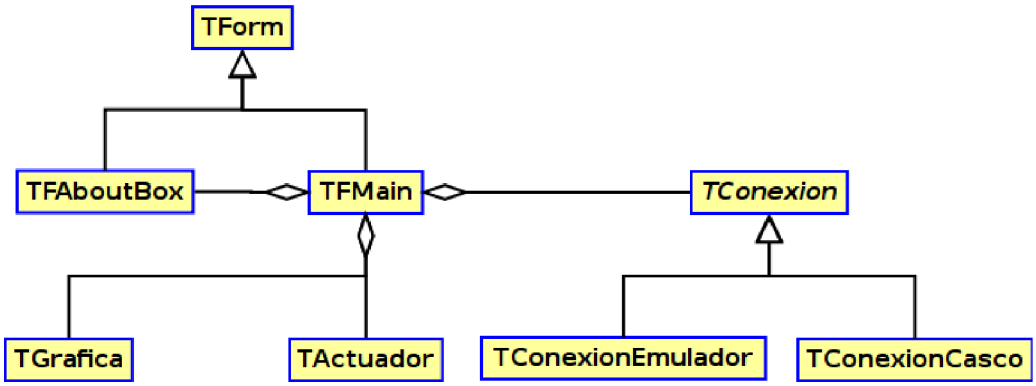


Figura 43: Diagrama de clases

La aplicación comprende dos formularios que extienden la clase **TForm** que forma parte de las bibliotecas VCL. El primero, **TFMain**, representa la ventana principal de la aplicación en donde se gestionan todos los eventos de la interfaz gráfica además de contener el bucle principal de recuperación de eventos. El otro formulario, **TFAboutBox**, se utiliza para presentar la ventana "Acerca de..." en donde se presenta la información de la versión, autoría, etc. Nótese que como la visualización de este formulario se realiza desde la instancia de la clase **TFMain**, ambas clases están relacionadas mediante una agregación.

La clase **TGrafica** también es una agregación de la clase **TFMain** ya que se encarga de gestionar, mediante seis gráficas que varían en el tiempo, la visualización de las señales: ninguna, neutral, avanzar, retroceder, girar a la izquierda y girar a la derecha que se reciben desde el casco.

La clase **TActuador** gestiona las operaciones sobre el puerto paralelo haciendo uso de la biblioteca *impout32*. La instancia de esta clase se crea desde la clase **TFMain** por lo que también es una agregación de la misma.

Para gestionar las operaciones con la **API Emotiv** se ha dispuesto la clase **TConexion** la cual es una clase abstracta ya que no implementa el método de conexión. Este método es implementado por las clases que la extienden, **TConexionEmulador** y **TConexionCasco**, cuya función es particularizar el comportamiento en el caso de que la conexión se vaya a realizar con el emulador **EmoComposer** o con el **Motor Emotiv** respectivamente.

Nótese que la clase **TFMain** también define una referencia a la clase **TConexion** para comunicarse con la **API Emotiv** lo que se denota con una relación de agregación, aunque como esta es abstracta, en realidad se define una instancia de una de sus clases hijas haciendo uso de polimorfismo (véase el Listado 2).

```
private:
   TConexion *con; // Puntero a la instancia de conexión

(...)
// Conecta según la selección
if (RadioGroupMotor->ItemIndex == 0) {
   // Conexión con el motor
   con = new TConexionCasco();
   (...)
} else {
   // Conexión con el emulador
   con = new TConexionEmulador(AnsiString(cadenaIp).c_str(),
                               StrToInt(EditPuerto->Text));
   (...)
}
```

Listado 2: Uso del polimorfismo para la conexión

4.3 Diagramas de Estados

La aplicación durante su tiempo de vida pasa por diversos estados, los cuales se pueden visualizar en la Figura 44.

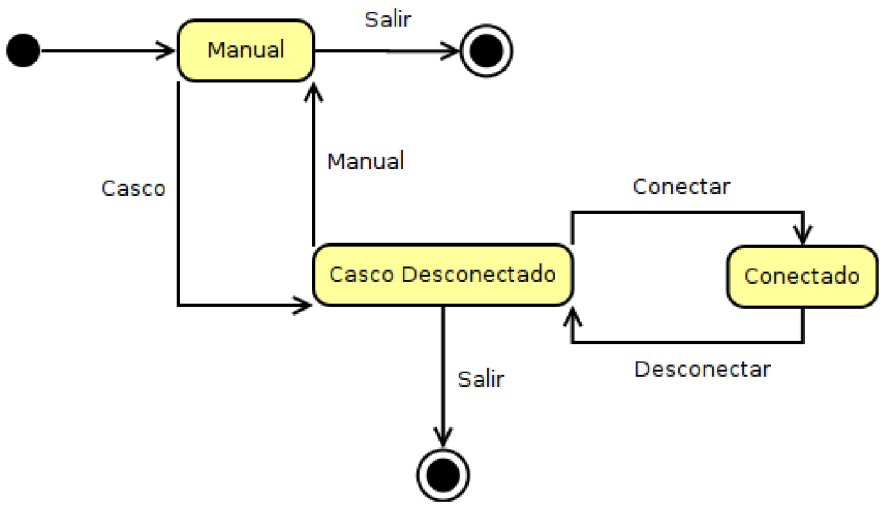


Figura 44: Diagrama de estados de la aplicación

Inicialmente el sistema se encuentra en modo manual (estado **Manual**) por lo que el vehículo se podrá controlar mediante el ratón, desde aquí es posible finalizar la ejecución de la aplicación, o bien pasar al modo casco (estado **Casco Desconectado**) desde el cual se puede emprender la conexión con el **Motor Emotiv** o con el emulador **EmoComposer**, existiendo también la posibilidad de finalizar la ejecución del programa. Por último en el estado **Conectado** sólo es posible desconectar y volver al estado anterior, no pudiéndose abandonar la aplicación en ningún caso.

Dentro del estado **Conectado**, existe una subdivisión de estados que se corresponden al estado de la conexión del **Motor Emotiv** con el casco (la aplicación se encuentra conectada al

Motor Emotiv pero éste no necesariamente existe un enlace con el casco).

Siguiendo el diagrama de estados de la Figura 45, se observa que si el dispositivo receptor USB se encuentra desconectado, el enlace con el casco se encontrará en el estado **Desconectado**, al conectar el dispositivo, se activa el enlace pasando al estado **Conectado**, mientras que si aún no se ha cargado el perfil del usuario se pasará al estado **Sin Perfil**. Si al menos alguna de las señales recogidas desde los electrodos no proviene de un buen contacto el estado destino será el de **Señal Ruidosa**, mientras que si el enlace inalámbrico no es de calidad el estado al que se pasará será el llamado **Sin Señal**. Para más detalles sobre el bucle principal véanse los Diagramas de Actividad en el punto 4.5.

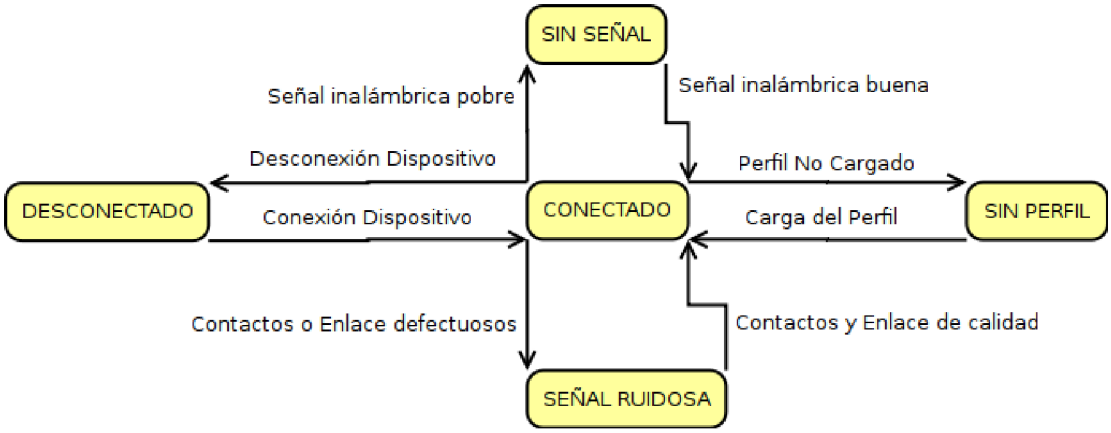


Figura 45: Diagrama de estados del bucle principal

4.4 Diagramas de Secuencia y Colaboración

La función de esta serie de diagramas es la de proporcionar información sobre la forma en que dialogan los objetos de la aplicación. Por una parte se describen la secuencia de señales que se envía y reciben durante el transcurso del tiempo (diagramas de secuencia) y por otra se identifican qué objetos intervienen en cada paso de señales (diagramas de colaboración).

En la Figura 46 se visualiza el diagrama de secuencia que representa la creación de instancias y la inicialización de las mismas que se produce al inicio de la aplicación. Nótese que luego de crear la gráfica, se la inhabilita y se vacía toda la información que pudiera tener almacenada. En cuanto al actuador, luego de su creación se procede a su inicialización (poniendo en estado bajo las salidas del puerto paralelo) para posteriormente inhabilitarlo de modo que no se produzcan cambios hasta que sea expresamente habilitado desde la interfaz de usuario.

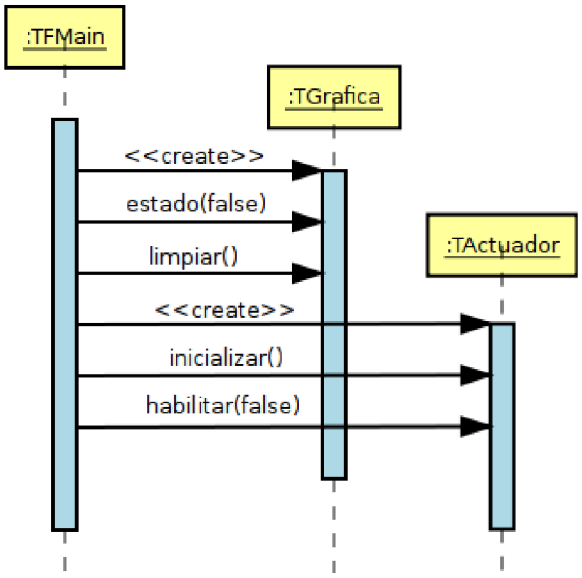


Figura 46: Secuencia de inicio de la aplicación

En la Figura 47 se puede visualizar este diálogo entre instancias en donde se destaca que el objeto de la clase **TFMain** dialoga con la instancia de la clase **TGrafica** y con la de la clase **TActuador**.

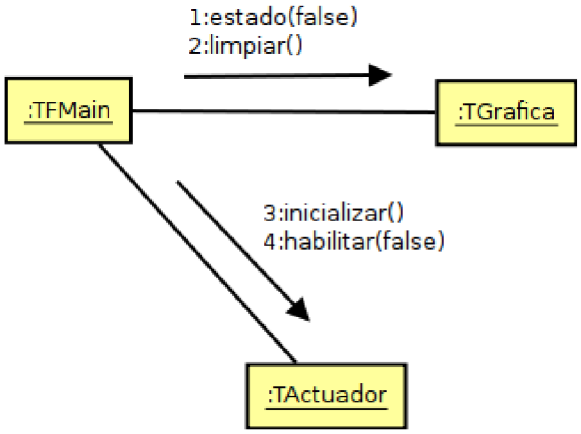


Figura 47: Colaboración de inicio de la aplicación

En el Listado 3 se observan las porciones de código en dónde se crean e inicializan las instancias de las clases **TGrafica** y **TActuador**.

```
// Inicialización de los atributos
grafica = new TGrafica(PaintBoxGrafica,ImageGrafica);
(...)
// Inicialización de los componentes del formulario
// Configura el formulario
(...)
grafica->estado(false);
grafica->limpiar();
(...)
// Inicialización del actuador
```

```
actuador = new TActuador();
if (actuador->estadoError()) {
   registro(RG_Error,actuador->textoError());
} else {
   actuador->inicializar();
   actuador->habilitar(ActionActuador->Checked);
}
```

Listado 3: Creación de instancias al inicio de la aplicación

A continuación, en la Figura 48 se visualiza el diagrama de secuencia correspondiente al intercambio de mensajes que se produce al conectar y al desconectar con el **Motor Emotiv** (o con el programa **EmoComposer**).

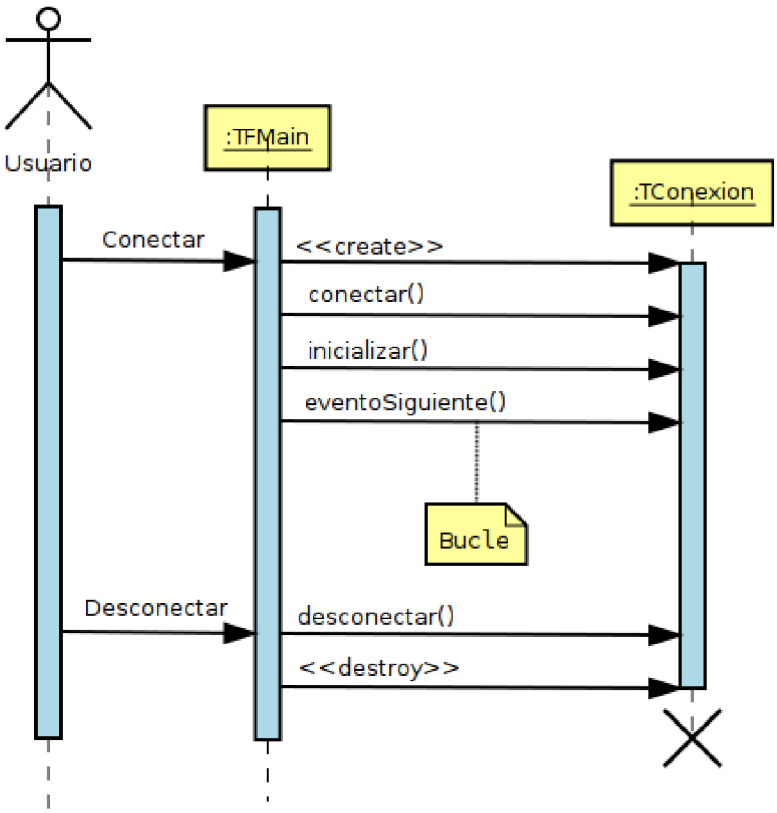


Figura 48: Secuencia de conexión y desconexión

Cuando el **Usuario** da la orden de conexión, la instancia de la clase **TFMain** crea una nueva instancia de la clase **TConexion** y a continuación le envía el mensaje para comenzar la conexión seguido de la solicitud de inicialización en dónde se activan las optimizaciones de los algoritmos de detección. Luego comienza a ejecutarse el bucle principal de recuperación de eventos que se extiende hasta que el usuario da la orden de desconexión, momento en el cual la instancia de **TFMain** envía el mensaje desconectar a la instancia de **TConexion** antes creada, para finalizar procediendo a su destrucción.

En la Figura 49 se presenta el diagrama de colaboración para la conexión y desconexión en

donde se puede apreciar la unidireccionalidad de los mensajes y como la instancia de la clase **TFMain** gestiona las peticiones del usuario, creando, dialogando y posteriormente destruyendo la instancia de la clase **TConexion**.

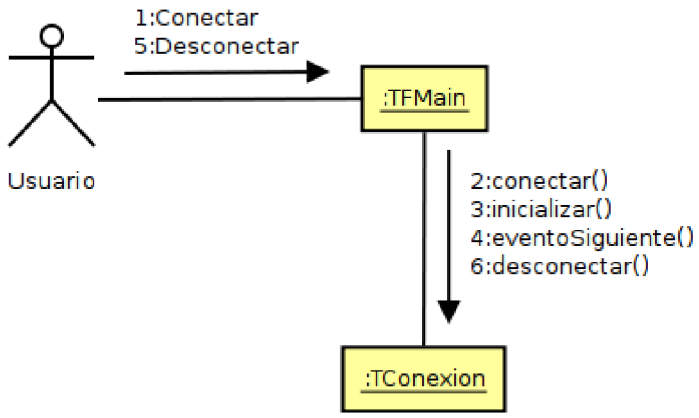


Figura 49: Colaboración de conexión y desconexión

En el Listado 4 se presenta la porción de código en donde se crea la instancia de la clase **TConexion** utilizando polimorfismo, se establece la conexión y se inicializa la misma. Posteriormente, luego de la ejecución del bucle principal (analizado a continuación) se procede a desconectar y a liberar la memoria ocupada por la instancia antes creada.

```
private:
   TConexion *con; // Puntero a la instancia de conexión
(...)
// Conecta según la selección
if (RadioGroupMotor->ItemIndex == 0) {
   // Conexión con el motor
   con = new TConexionCasco();
   (...)
} else {
   // Conexión con el emulador
   con = new TconexionEmulador(AnsiString(cadenaIp).c_str(),
                               StrToInt(EditPuerto->Text));
   (...)
}
(...)
if (con->conectar()) {
   (...)
   conectado = true;
   // Optimización
   if (con->inicializar()) {
      (...)
   }
}
(...)
if (conectado) {
  // Desconectar
   if (con->desconectar()) registro(RG_Mensaje, "Desconectado");
   (...)
}
delete con;
```

Listado 4: Conexión, optimización y posterior desconexión

En la Figura 50 se presenta el diagrama de secuencia correspondiente al bucle principal dentro del cual se obtienen los nuevos estados desde el casco. En primer lugar se consulta a la instancia de la clase **TConexion** por el siguiente evento y cuando éste se produce se obtiene el tiempo de conexión desde la **API Emotiv** presentando este valor en la interfaz gráfica, luego si el evento se corresponde con una actualización de estado se procede a obtener la información de la carga de la batería, el nivel de la señal inalámbrica y la calidad de contacto de los electrodos, presentando los datos obtenidos en forma de indicadores visuales en la interfaz gráfica.

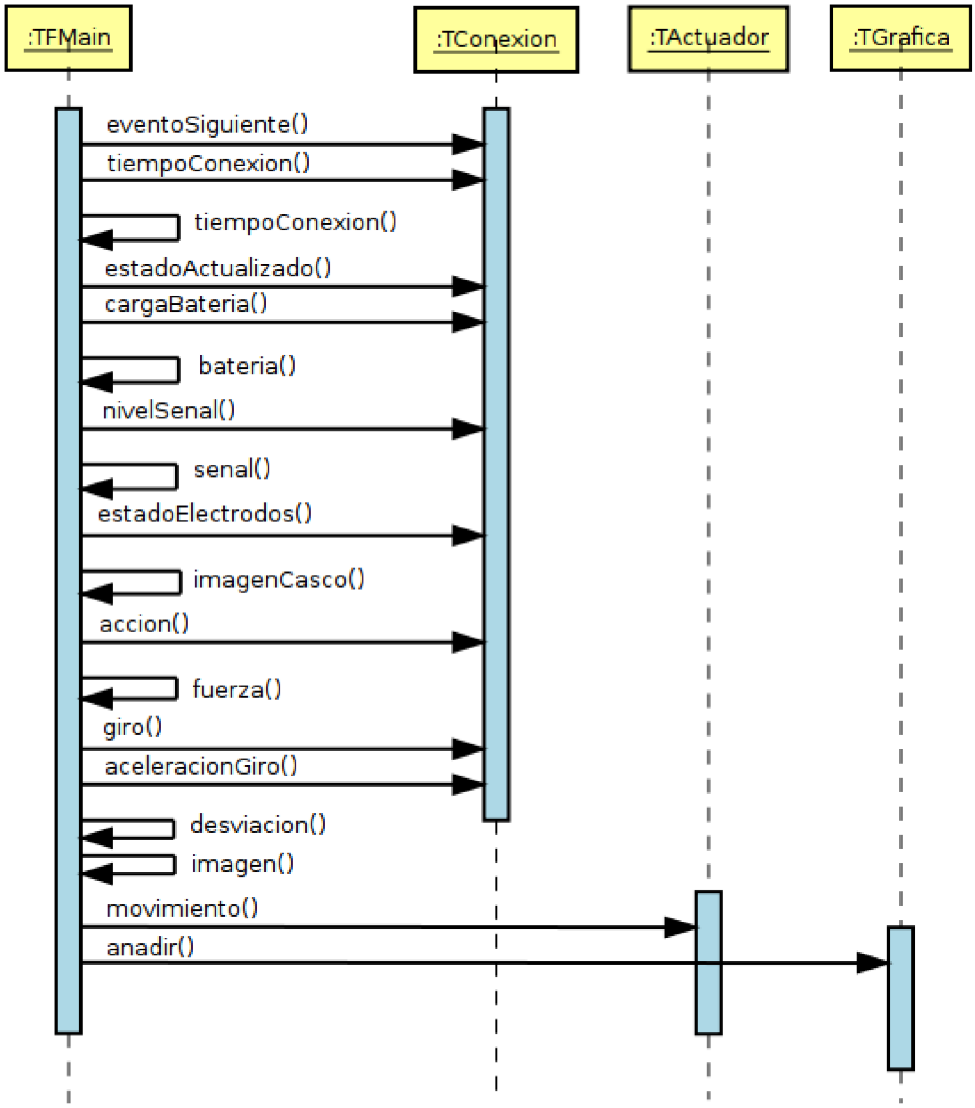


Figura 50: Secuencia que se ejecuta dentro del bucle principal

A continuación se obtiene la acción detectada y la intensidad de la misma así como el movimiento de la cabeza detectado a través del giróscopo y con qué aceleración fue realizado. Esta información también se visualiza en la interfaz gráfica y con ella se calcula el movimiento a realizar, enviando el resultado mediante un mensaje a la instancia de la clase **TActuador**.

Por último se vuelcan las detecciones obtenidas en la gráfica a través del diálogo con la instancia de la clase **TGrafica**.

En la Figura 51 se presenta la colaboración entre las instancias de las clases que intervienen en el bucle principal de obtención de eventos. Nótese que cada actualización de la información que se visualiza en la interfaz gráfica del formulario principal (instancia de la clase **TFMain**) implica un mensaje que la instancia se envía a si misma.

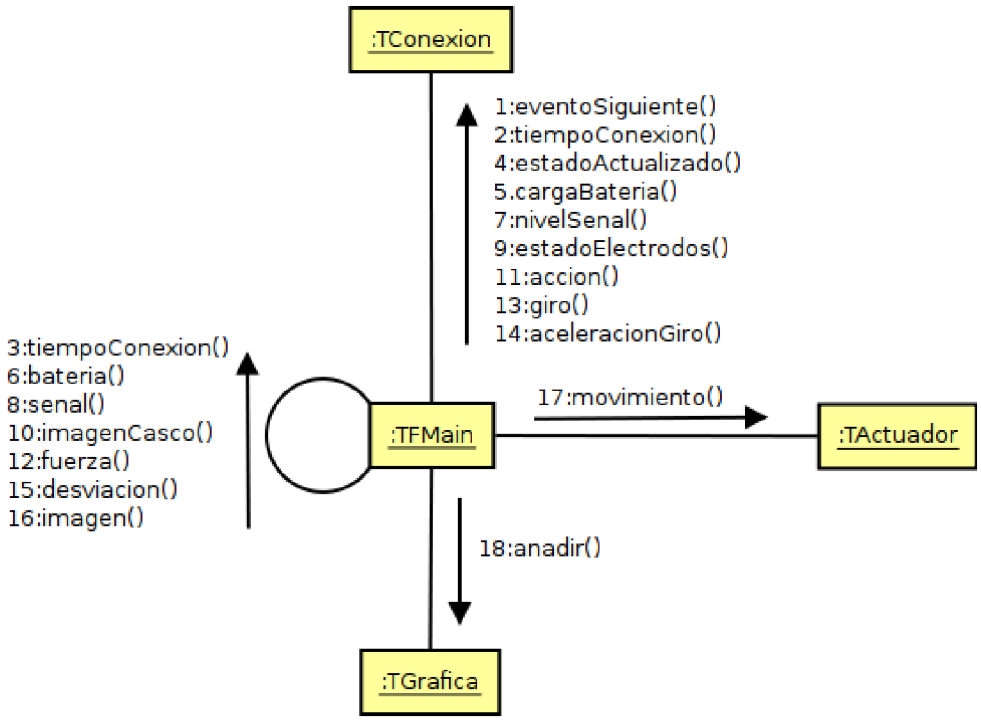


Figura 51: Colaboración correspondiente al bucle principal

Para más información sobre el bucle principal de obtención de eventos, véanse los Diagramas de Actividad en el punto 4.5.

En la Figura 52 se presenta la secuencia que sigue la aplicación para calibrar el acelerómetro del giróscopo. Todo comienza con la solicitud que realiza el **Usuario** a la instancia de la clase **TFMain**, ésta a su vez, luego de recuperar el siguiente evento, envía la petición de inicialización del giróscopo a la instancia de la clase **TConexion**.

Nótese que la llamada al método **inicializaGiro()** sólo debe realizarse luego de que se haya recuperado un evento.

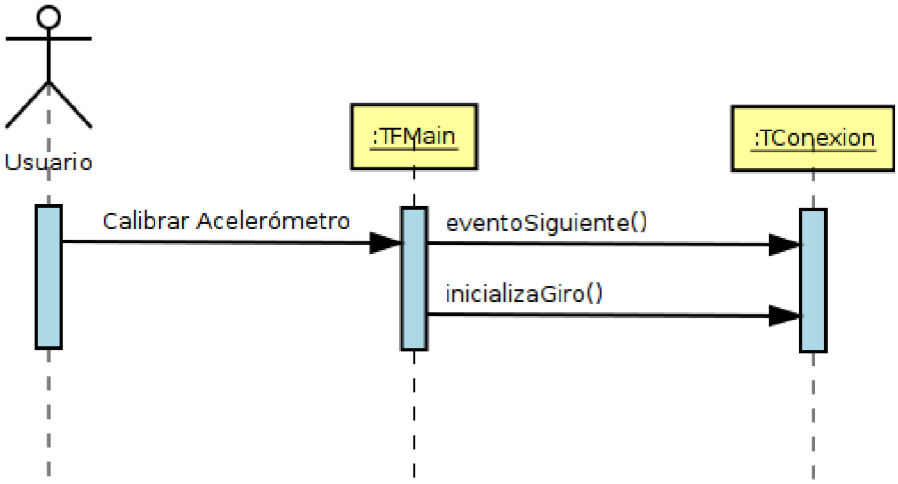


Figura 52: Secuencia de calibración del acelerómetro

En la Figura 53 se puede visualizar la colaboración que ocurre entre el usuario y las instancias de las clases involucradas cuando se solicita la calibración del acelerómetro y en el Listado 5, un extracto del código que realiza dicha operación.

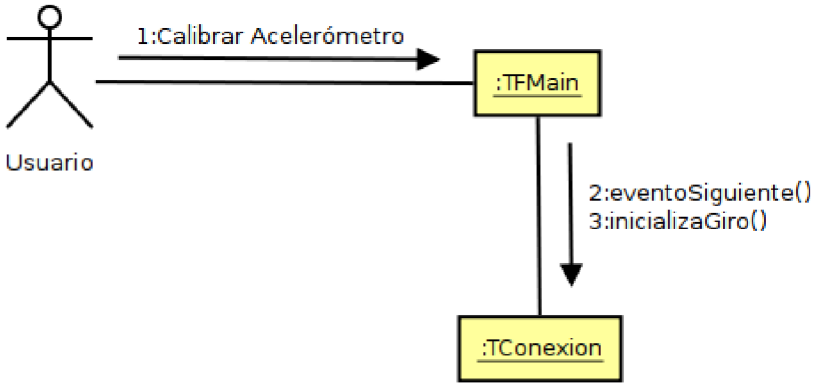


Figura 53: Colaboración de calibración del acelerómetro

```
inicializarGiro = true;
(...)
if (inicializarGiro) { // Si hay una petición para inicializar el acelerómetro
   inicializarGiro = false;
   (...)
   if (!con->inicializaGiro()) registro(RG_Error, con->cadenaError());
   (...)
}
```

Listado 5: Solicitud y procesamiento de la calibración del acelerómetro

En la Figura 54 se presenta la secuencia para la carga del perfil de usuario. Luego de que el **Usuario** realiza la solicitud a la instancia de la clase **TFMain**, esta efectúa la llamada a la instancia de la clase **TConexión** para luego obtener los niveles de entrenamiento y la sensibilidades de avance y retroceso almacenadas dentro del perfil.

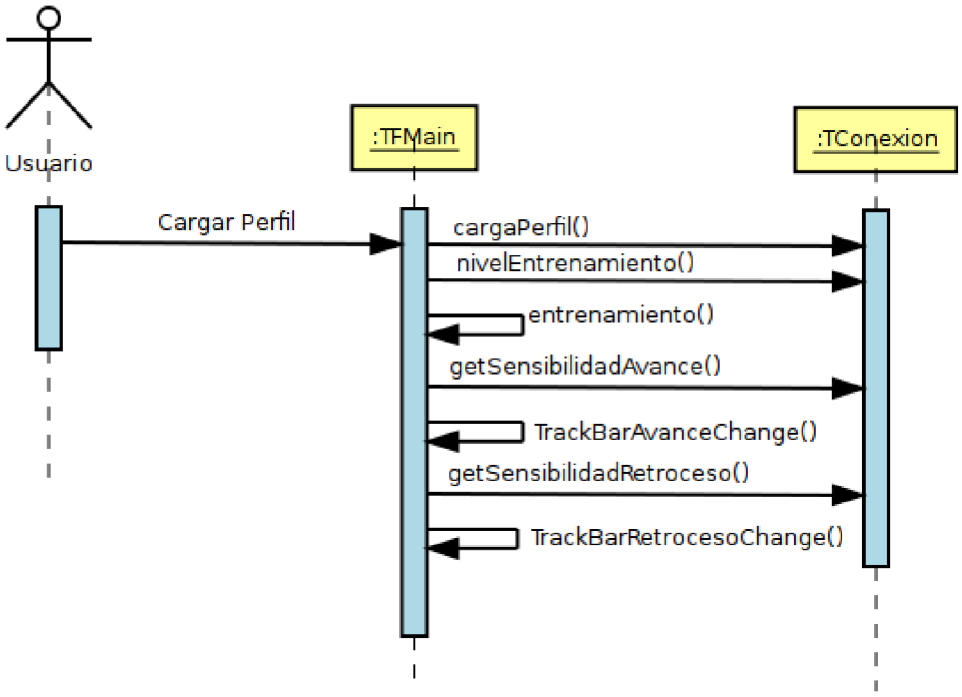


Figura 54: Secuencia de carga del perfil de usuario

Nótese que cada dato obtenido provoca una actualización de la interfaz gráfica provista por la instancia de la clase **TFMain**.

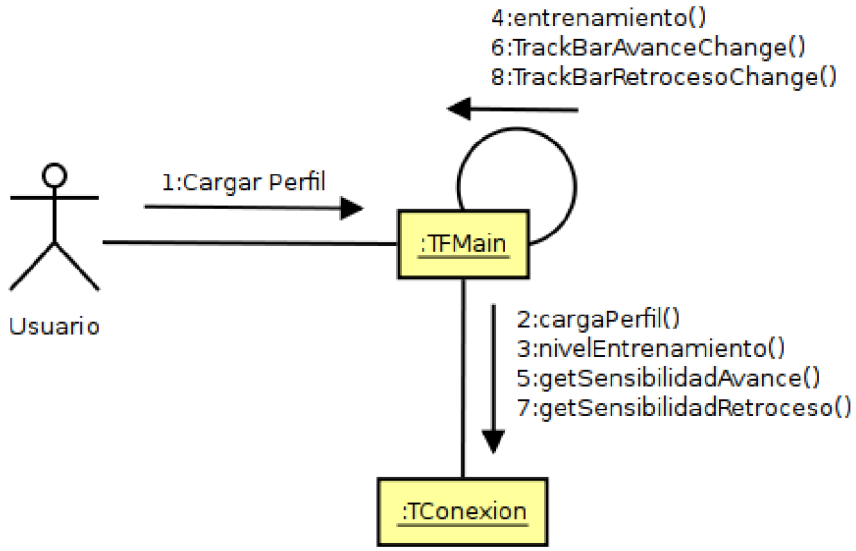


Figura 55: Colaboración para la carga del perfil de usuario

En la Figura 55 se puede visualizar la colaboración que se produce entre instancias cuando el **Usuario** solicita la carga de un perfil desde el disco y en el Listado 6 un extracto del código que realiza esta operación.

```
if (OpenDialog->Execute()) {
   cargarPerfil = true; // Petición de carga de perfil
   ficheroPerfil = OpenDialog->FileName;
   (...)
}

(...)

if (cargarPerfil) { // Petición de carga de perfil
   cargarPerfil = false;
   if (con->cargaPerfil(AnsiString(ficheroPerfil).c_str())) {
      perfilCargado = true;
      (...)
      entrenamiento(OpenDialog->FileName,con->nivelEntrenamiento());
      TrackBarAvance->Position = con->getSensibilidadAvance();
      TrackBarAvanceChange(TrackBarAvance);
      TrackBarRetroceso->Position = con->getSensibilidadRetroceso();
      TrackBarRetrocesoChange(TrackBarRetroceso);
   }
   (...)
}
```

Listado 6: Solicitud y procesamiento de la carga del perfil de usuario

En la Figura 56 se presenta el diagrama de secuencia en que se puede apreciar el intercambio de mensajes que se produce cuando el **Usuario** solicita visualizar la ventana "Acerca de...".

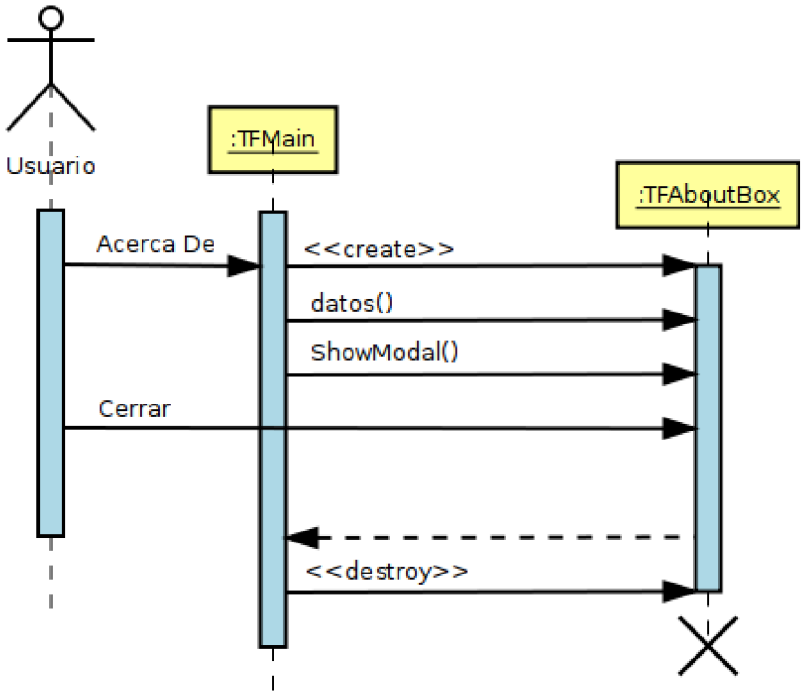


Figura 56: Secuencia de visualización de la ventana "Acerca de..."

Una vez recibida la petición a través de la interfaz de usuario, la instancia de la clase **TFMain** crea una instancia de la clase **TFAboutBox** para luego cargar los datos de la versión y visualizarla en modo *modal*[25]. Cuando el usuario cierre la ventana, la instancia de la clase

25 Modo de visualización de un formulario de manera que la ejecución de la aplicación permanece suspendida hasta

TFMain procederá a destruirla liberando la memoria que ésta ocupaba.

En la colaboración asociada (Figura 57) se visualiza el diálogo entre las clases mediante el paso de mensajes. Nótese que en este caso cuando se cierra la ventana que corresponde a una instancia de la clase **TFAboutBox** envía una notificación a la instancia de la clase **TFMain** de modo que ésta la destruya sólo cuando ya no se encuentra visible.

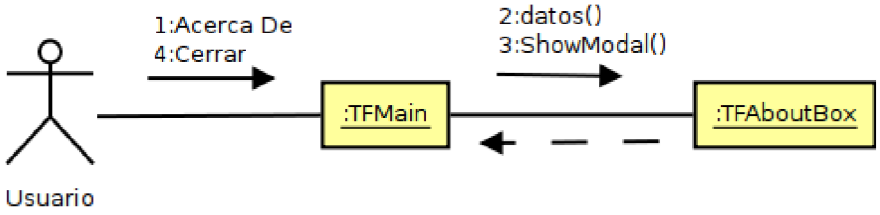


Figura 57: Colaboración para la visualización de la ventana "Acerca de..."

En el Listado 7 se puede consultar la porción de código que se ejecuta cuando el **Usuario** solicita ver la ventana “Acerca de...”.

```
// Visualiza la ventana de Acerca de...
Application->CreateForm(__classid(TFAboutBox), &FAboutBox);
FaboutBox->datos(NOMBRE, PROYECTO, FECHA_VERSION + " - " + AUTOR,
                 "Versión: " + VERSION + " (" + BUILD + ")");
FAboutBox->ShowModal();
FAboutBox->Release();
```

Listado 7: Creación, visualización y destrucción de la ventana “Acerca de...”

4.5 Diagramas de Actividad

En el siguiente diagrama de actividad (Figura 58) se visualiza el flujo de ejecución que sigue el programa en cada iteración del bucle principal de obtención de eventos desde el **Motor Emotiv** o, alternativamente, desde el emulador **EmoComposer**.

La primera decisión que se toma es referente a si se ha recibido un nuevo evento, si no es el caso se pasa a comprobar si hay una petición pendiente de carga de perfil de usuario. Si por el contrario se ha recibido un nuevo evento, se extrae desde el mismo la marca de tiempo y se la visualiza en la interfaz gráfica pasándose a tomar una nueva decisión teniendo en cuenta en este caso el tipo de evento recibido.

que el usuario procede a cerrarlo, generalmente desde el botón “Aceptar”.

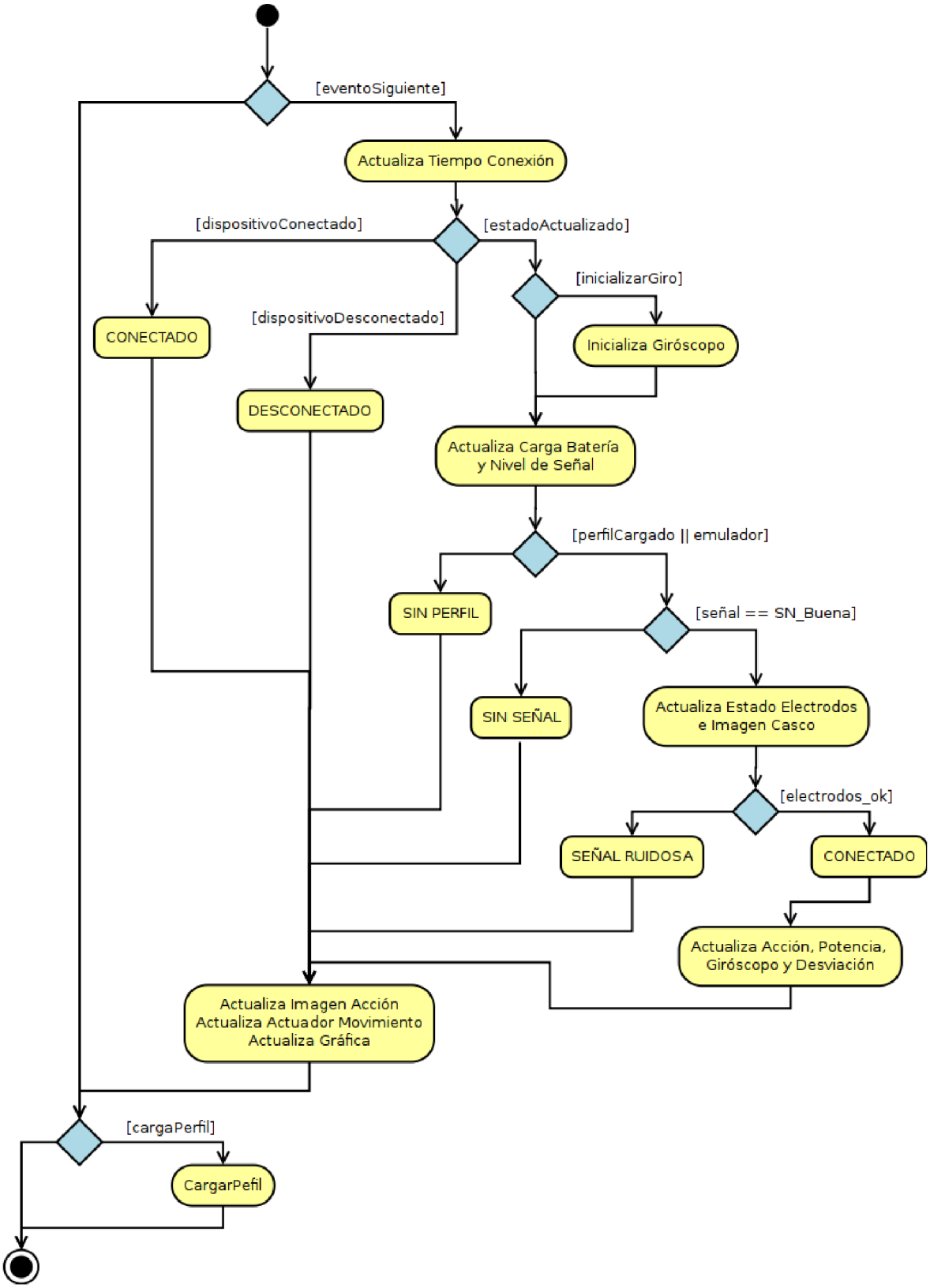


Figura 58: Diagrama de actividad para el bucle principal de la aplicación

Si se trata del evento correspondiente a la conexión del receptor USB, el proceso pasará al estado CONECTADO procediendo a actualizar la imagen de la acción que se está ejecutando (en

este caso la imagen que se corresponde con ninguna acción), se actualiza el estado del actuador (todas las salidas inactivas) y se registra en las gráficas de acciones la señal correspondiente a la inactividad.

En el caso de que el tipo de evento sea el correspondiente a la desconexión del dispositivo receptor USB, el proceso pasará al estado DESCONECTADO y las actualizaciones de la interfaz gráfica serán similares al caso anterior así como la inactividad del actuador.

Por último si el tipo de evento denota una actualización del **Estado Emotiv**, se procederá a comprobar si existe una solicitud pendiente de calibración del giróscopo (recuérdese que sólo puede realizarse cuando se recibe un nuevo estado) procediendo a la misma en caso afirmativo.

Siguiendo con el flujo de ejecución el paso siguiente será obtener la información de la carga de la batería y de la intensidad de la señal inalámbrica contenidas en el nuevo estado y presentarla en la interfaz gráfica.

Ahora para detectar una acción a partir del nuevo estado, es imprescindible que se haya cargado antes un perfil de usuario. Si este no es el caso, se presentará esta situación adoptándose el estado SIN PERFIL y pasando a actualizar la información de la interfaz gráfica como ya se ha hecho con anterioridad. Por otro lado, en el caso de existir un perfil cargado (o cuando se está utilizando el emulador) se comprobará el nivel de señal del enlace inalámbrico ya que si no es suficientemente buena se pasará al estado SIN SEÑAL.

Si la señal inalámbrica es buena, se actualizará la información referente a la calidad de contactos de los electrodos y si esta no es aceptable se pasará al estado SEÑAL RUIDOSA, mientras que en caso contrario se mantendrá el estado CONECTADO procesando la acción recibida así como la aceleración obtenida desde el giróscopo.

Cuando desde la acción se obtenga un movimiento, se visualizará en la interfaz gráfica indicándose también en la señal de la gráficas temporales un aumento de intensidad proporcional a la potencia recibida. Por último si se superan los umbrales de activación, también se volcará en el actuador las operaciones correspondientes para realizar el movimiento del vehículo.

En la Figura 59 se presenta el diagrama de actividad que indica el flujo de ejecución que sigue el proceso para detectar la dirección de giro y para realizar la activación de la señal correspondiente en el actuador.

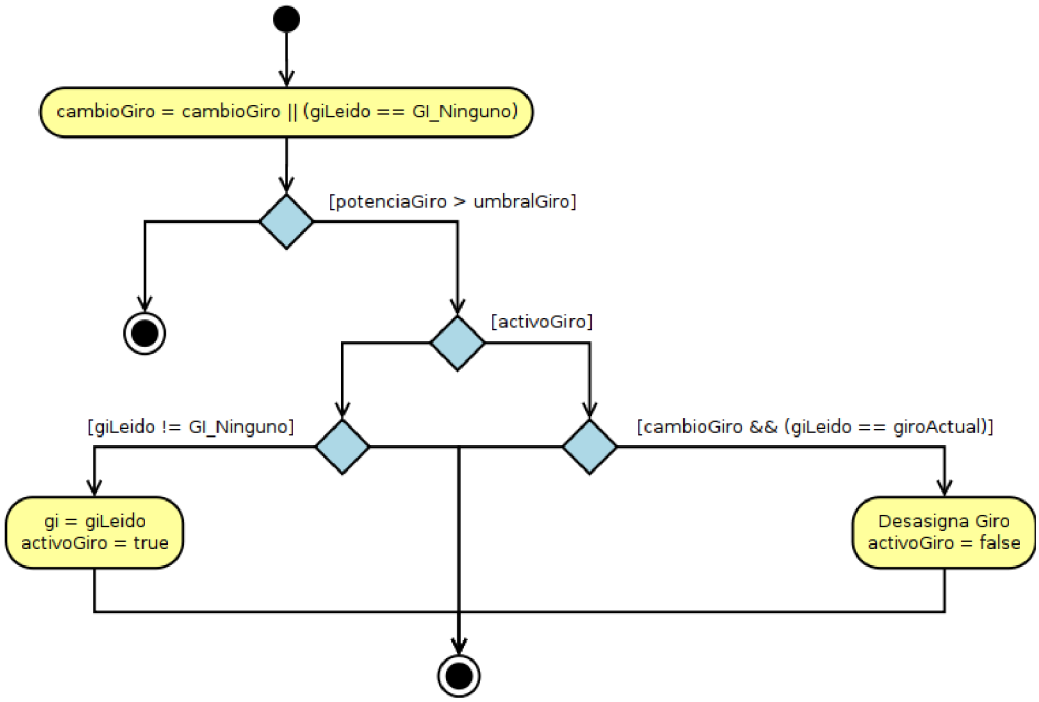


Figura 59: Diagrama de actividad para la detección de giros

En primer lugar se actualiza el valor de **cambioGiro** que depende de si se encontraba activo con anterioridad o de que no se haya detectado ningún movimiento de la cabeza del usuario. **CambioGiro** cumple la función de indicar que la cabeza del usuario a vuelto a su posición central y que en las próximas detecciones será posible desactivar el giro actualmente activo (las acciones de giro se comportan como interruptores de forma que cuando se activa el giro hacia la izquierda, éste se mantendrá activo hasta que se produzca una nueva detección de giro a la izquierda, provocando su desactivación, esto sólo será posible cuando la cabeza vuelva a situarse en la posición central. Lo mismo se aplica para el giro hacia la derecha).

Si la aceleración (**potenciaGiro**) detectada supera el umbral configurado desde la interfaz gráfica se procede a comprobar si hay un giro activo (**activoGiro**) en cuyo caso si se está en condiciones de cambiar el estado (**cambioGiro == true**) y se ha vuelto a leer el mismo giro que el que se encontraba activo (se vuelve a girar la cabeza para el mismo lado) se procederá con la desactivación de la acción de giro.

En el caso de que no hubiera ningún giro activo y que se lea un movimiento hacia la izquierda o hacia la derecha de la cabeza, se procederá a su activación de la acción correspondiente,

En el Listado 8 se puede consultar las principales instrucciones del código para el bucle principal de detección de eventos.

```
if (con->eventoSiguiente()) { // Si hay un evento desde el casco
   tiempoConexion(con->tiempoConexion());

   if (con->estadoActualizado()) { // Nuevo estado Emotivo

      if (inicializarGiro) { // Si hay una petición para inicializar
                             // el acelerómetro
         inicializarGiro = false;
         (...)
         if (!con->inicializaGiro()) registro(RG_Error, con->cadenaError());
         (...)
      }

      // indicadores
      con->cargaBateria(&carga_bat, &carga_max);
      bateria(carga_bat, carga_max);
      senal(con->nivelSenal());

      if (perfilCargado || emulador) { // debe estar cargado el perfil
                                       // o bien estar conectado al emulador

         // Imágenes
         con->estadoElectrodos(electrodos, &electrodos_ok);
         imagenCasco(electrodos_ok,electrodos);

         if (electrodos_ok) { // Si la detección es fiable

            // conectado y recibiendo
            estado(ES_Conectado);

            // Acción
            con->accion(&ac, &potencia);
            activoAccion = potencia >= (unsigned int) TrackBarUmbralAvRet->Position;
            fuerza(potencia);

            // Giróscopo
            con->giro(&giLeido); // acción de giro
            potenciaGiro = con->aceleracionGiro(); // nivel de aceleración de giro
            desviacion(potenciaGiro);

            retardoGiro++;

            if (retardoGiro > 10) {

               cambioGiro = cambioGiro ||
                            (giLeido == GI_Ninguno);
               // Si deja de detectar el mismo sentido ya puede ser desactivado

               if (potenciaGiro >= (unsigned int) TrackBarUmbralGiro->Position) {
                  // supera umbral

                  if (activoGiro) { // si estaba girando
                     if (cambioGiro) { // si hubo cambio
                        if (giLeido == gi) { // si el leído es igual al actual
                           gi = GI_Ninguno;  // se desasigna
                           activoGiro = false;  // se desactiva
                           retardoGiro = 0;
                        }
                     }
                  } else { // no estaba girando
                     if giLeido != GI_Ninguno) {
                        // si se ha leido una dirección de giro
                        gi = giLeido;  // se asigna
                        activoGiro = true;  // y se activa
                        retardoGiro = 0;
```

```
                    }
                 }
              }
           }

           // Gestiona la imagen a mostrar
           if (gi == GI_Ninguno) giImg = giLeido;
           else giImg = gi;

           // Registra el estado leido
           if (ActionEstados->Checked) registro(RG_Estado, con->cadenaEstado());

        } else {
           ac = AC_Ninguna; // Detección no fiable
           estado(ES_Ruidoso);
        }

     } else {
        ac = AC_Ninguna; // Perfil no cargado
        estado(ES_Perfil);
     }

  } else if (con->dispositivoConectado()) { // conexión dispositivo
     tiempoConexion(0.0);
     estado(ES_Conectado);
     registro(RG_Mensaje, "Dispositivo detectado");
  } else if (con->dispositivoDesconectado()) { // desconexión dispositivo
     tiempoConexion(0.0);
     estado(ES_NoDetectado);
     imagenCasco(false,NULL);
     registro(RG_Mensaje, "Dispositivo no detectado");
  } else registro(RG_Estado, con->tipoEstado());  // otro estado

  // Imagen acción
  imagen(ac, giImg, activoAccion, activoGiro);

  // Actuador
  if (activoAccion) ac_mov = ac;
  else ac_mov = AC_Ninguna;
  if (activoGiro) gi_mov = gi;
  else gi_mov = GI_Ninguno;
  actuador->movimiento(ac_mov, gi_mov);

  // Gráfica
  for (unsigned int i = 0; i < NUM_GRAFICAS; i++) {
     gr[i] = 0;
  }
  switch (ac) {
     case AC_Ninguna:  gr[SG_Ninguna] = 10; break;
     case AC_Centro:   gr[SG_Centro] = 10; break;
     case AC_Adelante: gr[SG_Adelante] = potencia; break;
     case AC_Atras:    gr[SG_Atras] = potencia; break;
  default:
     ;
  }
  switch (giLeido) {
     case GI_Izquierda: gr[SG_Izquierda] = potenciaGiro; break;
     case GI_Derecha:   gr[SG_Derecha] = potenciaGiro; break;
  default:
     ;
  }
  grafica->anadir(gr);

} // No hay eventos nuevos
```

```
if (cargarPerfil) { // Petición de carga de perfil
  cargarPerfil = false;
  if (con->cargaPerfil(AnsiString(ficheroPerfil).c_str())) {
    perfilCargado = true;
    (...)
    entrenamiento(OpenDialog->FileName,con->nivelEntrenamiento());
    TrackBarAvance->Position = con->getSensibilidadAvance();
    TrackBarAvanceChange(TrackBarAvance);
    TrackBarRetroceso->Position = con->getSensibilidadRetroceso();
    TrackBarRetrocesoChange(TrackBarRetroceso);
  } else {
    perfilCargado = false;
    (...)
    registro(RG_Error,"Error cargando el perfil desde " + OpenDialog->FileName);
    registro(RG_Error,con->cadenaError());
    entrenamiento("",0);
  }
  labelsFicheroPerfil(conectado);
  SpeedButtonPerfil->Enabled = true;
}

// Refresco cambios en la GUI
Application->ProcessMessages();
```

Listado 8: Bucle principal para la obtención de eventos.

Una vez desarrollada la aplicación, se comprobó su funcionamiento conectando con el emulador **EmoComposer** generando distintos tipos de eventos para comprobar que las respuestas y visualizaciones eran las correctas (Figura 60).

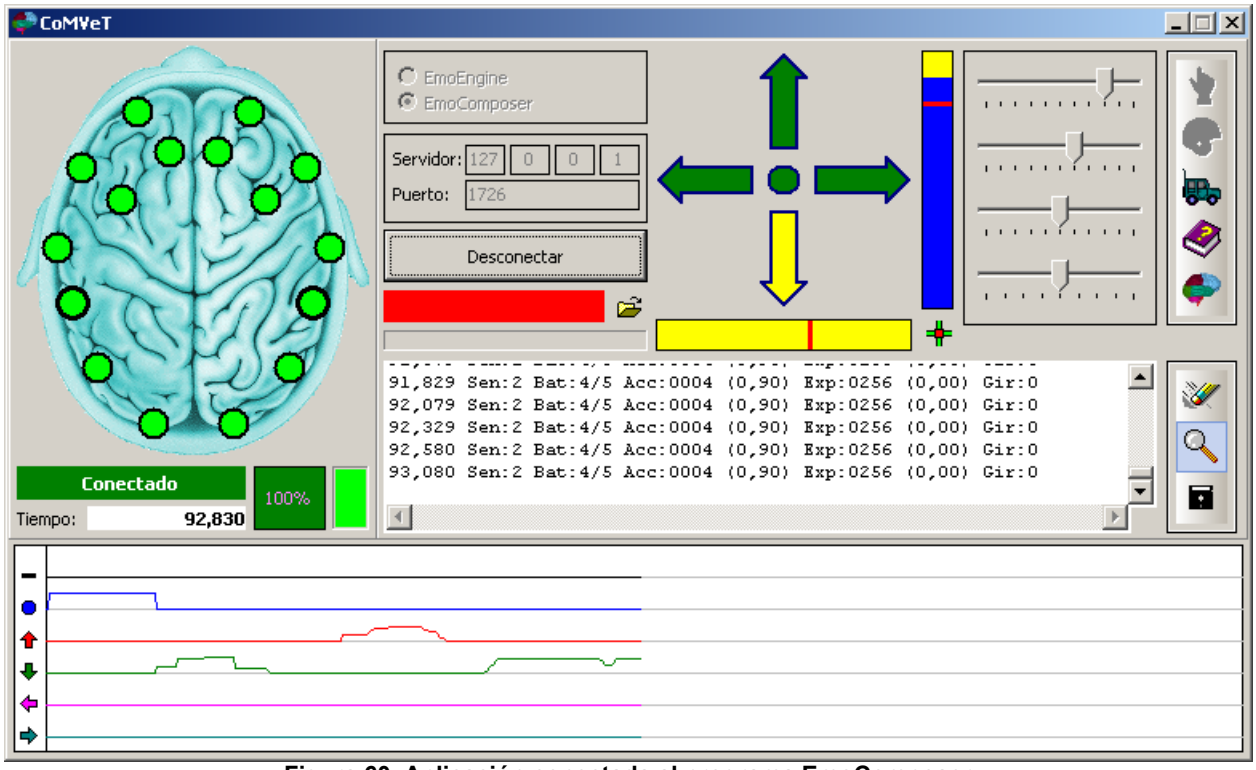


Figura 60: Aplicación conectada al programa EmoComposer

Nótese que desde el emulador no se pueden generar eventos para comprobar el funcionamiento de las acciones provenientes del giróscopo y que tampoco es posible utilizar un

perfil de entrenamiento. Estas comprobaciones se realizarán en la etapa Integración y pruebas en el Capítulo 6.

Por último, también se comprobó la correcta visualización del fichero de ayuda, el cual fue construido utilizando el programa Microsoft Help WorkShop, y la ventana Acerca de... con información del autor y de la versión (Figura 61).

Figura 61: Ventana Acerca de...

Capítulo 5

Desarrollo Hardware

Para que las detecciones obtenidas por el software se vean reflejadas en los movimientos del vehículo teledirigido se ha desarrollado la **Interfaz Hardware** (Figura 62).

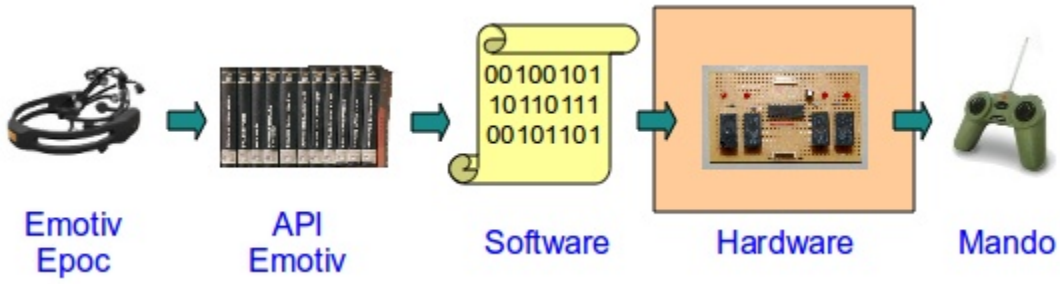


Figura 62: Desarrollo Hardware

Se escogió el puerto paralelo del PC debido a su sencillez electrónica ya que sus salidas de datos se encuentran directamente expuestas en el conector, aunque como se ha visto en el apartado anterior (Desarrollo Software) se hizo necesario utilizar una biblioteca externa (*inpout32.dll*) para poder gestionarlas desde Microsoft Windows.

El circuito para la interfaz debía cumplir la función de simular las acciones del usuario sobre el mando del vehículo teledirigido y a la vez ofrecer algún tipo de información visual de la información que se estaba volcando por el puerto paralelo. Además la idea era que la interfaz pudiera adaptarse a una amplia gama de mandos, en vez de sólo servir para el modelo particular escogido para este proyecto.

Con estas premisas se decidió utilizar un relé para activar cada operación disponible en el mando, ya que de esta forma se simulaba de la forma más fiable la activación de los interruptores de las palancas originales y se dotaba de cierta separación electrónica entre el puerto del PC y el mando de radiocontrol.

Para la monitorización de las señales se emplearon diodos LED[26] y como la corriente de salida de cada línea del puerto no era suficiente para alimentar los devanados de los relés se intercaló un transistor *darlington*[27] en cada línea para dotarla de la potencia necesaria (Figura 63).

26 *Light-Emitting Diode* – Diodo Emisor de Luz. [7]
27 Dispositivo semiconductor que combina dos transistores bipolares en un tándem en un único dispositivo. [27]

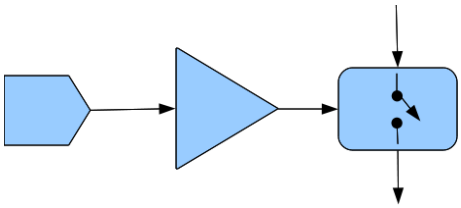


Figura 63: Diagrama en bloques de una línea de la Interfaz

Al emplear 4 líneas del puerto paralelo para cada una de las acciones (avanzar, retroceder, girar a la izquierda y girar a la derecha) se ha utilizado en lugar de 4 transistores independientes, el circuito integrado **ULN2803A** que contiene 8 transistores *darlington* (Figura 64) y que está indicado especialmente para aumentar la potencia de señales digitales, además de por su bajo coste y alta disponibilidad en el mercado local.

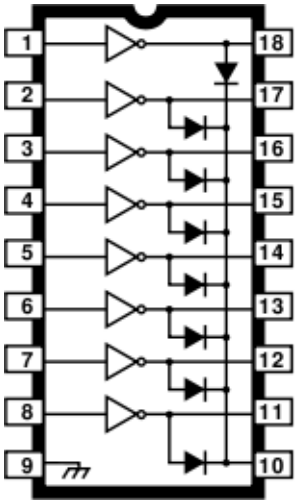


Figura 64: Circuito integrado ULN2803A

Se ha prestado especial atención a la corriente que se va a drenar del puerto paralelo, observándose en la hoja de datos que nunca será mayor a los 2.5mA que es capaz de suministrar el puerto sin que su tensión caiga por debajo de los 2.4V. Por otro lado la máxima corriente de salida de cada transistor *darlington* integrado supera los 150mA (4 transistores conduciendo simultáneamente) lo cual excede ampliamente los requisitos de cada relé escogido que ronda los 25mA para una alimentación de 9V. Otra característica interesante de este circuito integrado es que incorpora para cada transistor un diodo de descarga, muy útil para los casos como el que nos ocupa en los que se conmuta cargas inductivas[28].

En la Figura 65 se pueden observar las gráficas de polarización y los valores máximos de corriente para el circuito integrado **ULN2803A**.

28 La conmutación de cargas inductivas produce como efecto secundario la generación de una fuerza contra-electromotriz o tensión inversa que puede ocasionar la destrucción de los semiconductores.

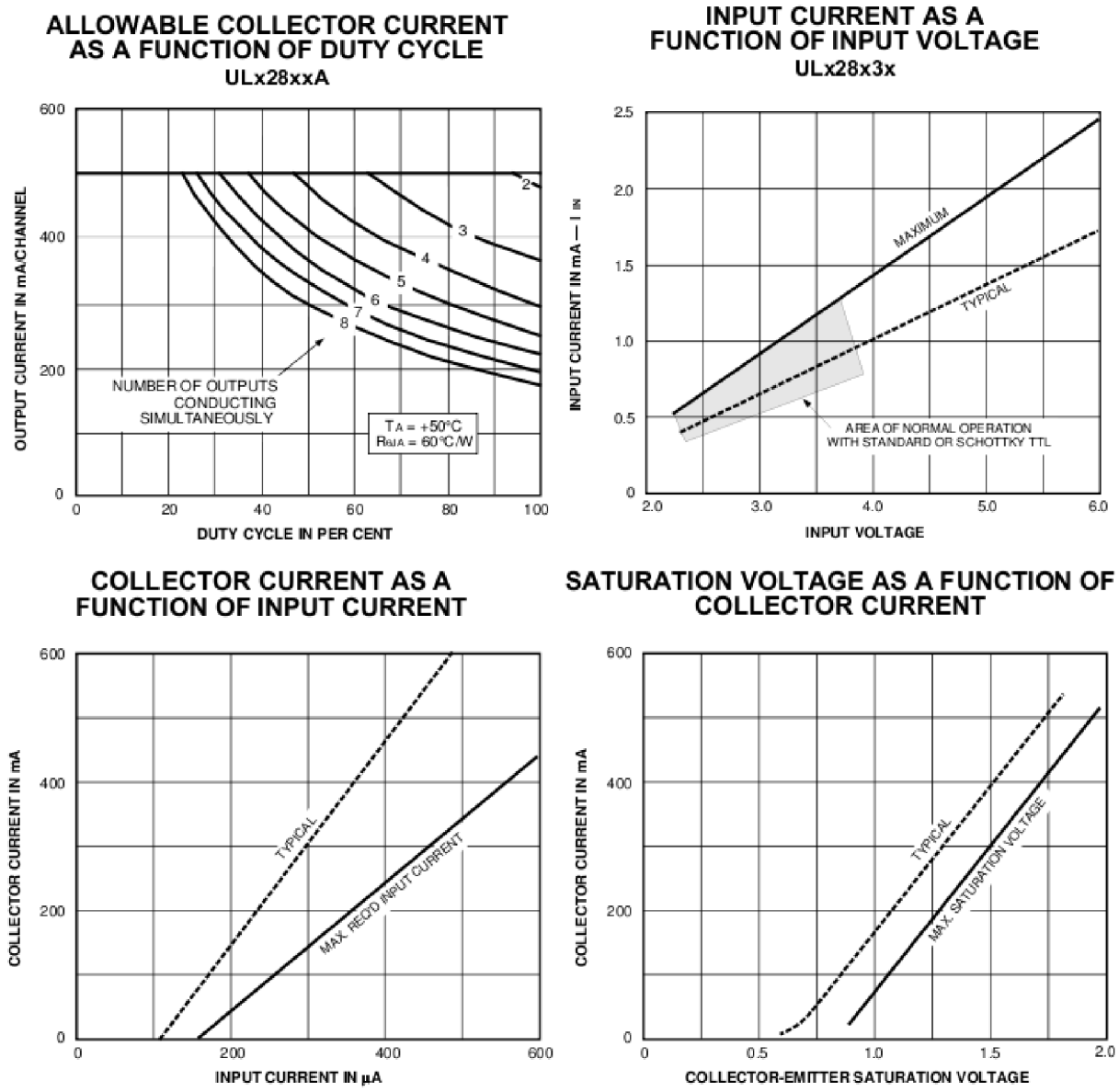


Figura 65: Gráficas del circuito integrado ULN2803A

El modelo de relé a utilizar será el **G58-1-H** de OMRON, entre cuyas características destacan la antes mencionada corriente por el devanado (25mA), la máxima corriente entre contactos (3A) y la máxima tensión de conmutación (30V cc).

Por último se dispondrán 4 diodos **LED** para señalizar la activación de las líneas del puerto paralelo y el correspondiente relé. Para ello se ha realizado el cálculo de la resistencia a emplear para cada uno (Fórmula 1) atendiendo a la tensión entre ánodo y cátodo típica (2V) pero escogiendo la mitad de la corriente máxima (15mA) sacrificando apenas la luminosidad pero ganando en autonomía ya que este circuito será alimentado por la misma batería que el mando. El valor comercial más cercano al obtenido es de **1KΩ**.

$$R = \frac{V_{CC} - V_{DL}}{I_{DL}} = \frac{9\text{V} - 2\text{V}}{7.5\text{mA}} = 933.3\,\Omega \approx 1\text{K}\,\Omega$$

Fórmula 1: Cálculo de la resistencia para el LED

El diseño se completa con un condensador de filtro de alimentación de **10µF/25V**, el conector de entrada para recibir la información desde el puerto paralelo y el conector de salida que comunicará la interfaz con el mando del vehículo teledirigido. Para el primero se ha empleado un conector **DB9**, mientras que para el segundo, uno del tipo **DIN-5**.

Utilizando el programa KiCad se procedió a dibujar el esquema de forma que luego fuera más sencillo el diseño del circuito impreso. En la Figura 66 se puede observar el esquema realizado para el circuito de la interfaz, en la Figura 67 el de los cables de interconexión y en la Tabla 2 el listado de los componentes principales.

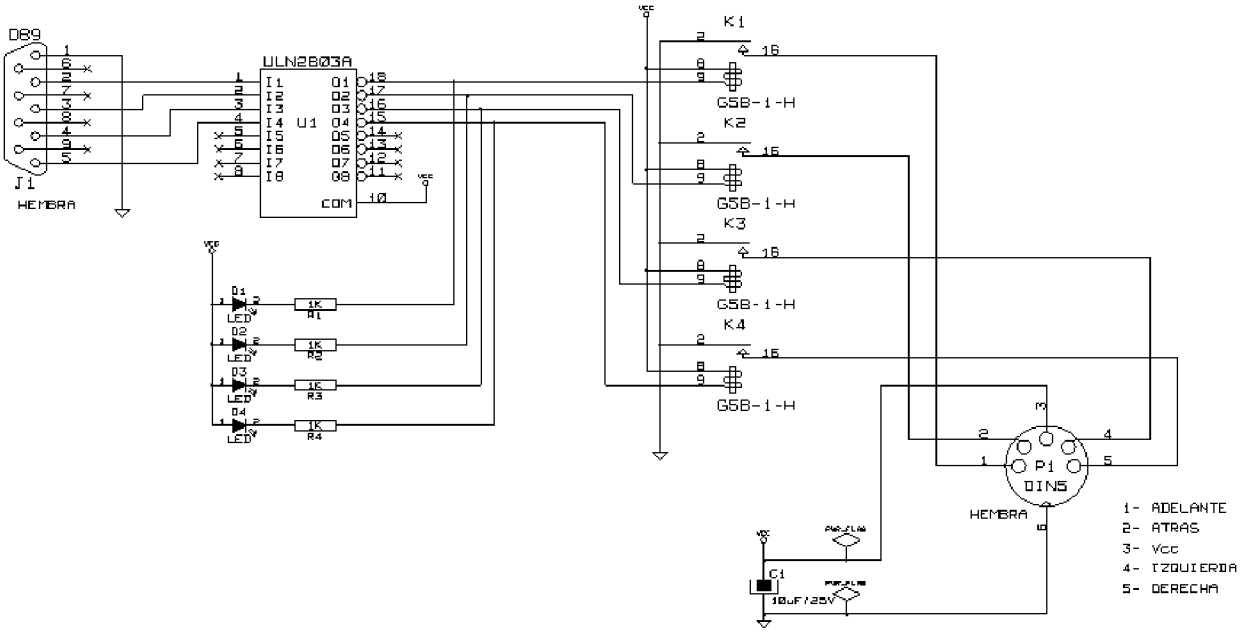


Figura 66: Esquema del circuito de la Interfaz

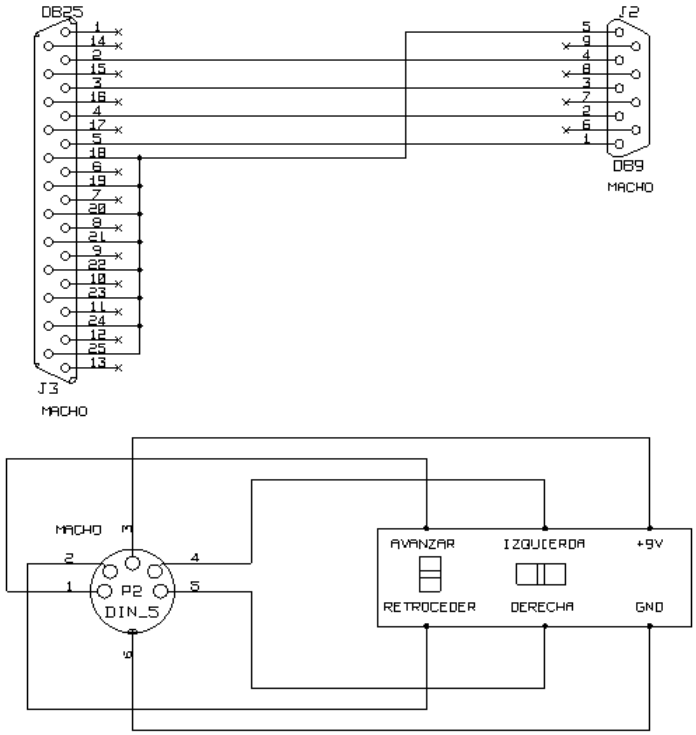


Figura 67: Esquema cables de interconexión

Cant.	Referencia	Componente	Fabricante	Código Fabricante	Código Farnell	Precio Unit. (Ago-2010)
4	D1,D2,D3,D4	LED 3mm	VISHAY SEMICOND.	TLHR4400	1612433	0,132 €
4	K1,K2,K3,K4	G5B-1-H	OMRON	No disponible	No disponible	2,550 €
4	R1,R2,R3,R4	1KΩ	MULTICOMP	MOR0S2J0102A50	1357860	0,059 €
1	C1	10µF / 25V	MULTICOMP	MR25V106M4X77	3017357	0,014 €
1	U1	ULN2803A	TOSHIBA	ULN2803APG	1047761	0,760 €
1	P1	DIN5 Hembra	MULTICOMP	SJ217-5P	1715943	0,990 €
1	P2	DIN5 Macho	NEUTRIK	NYS322G	4632515	1,210 €
1	J1	DB9 Hembra	ITT CANNON	ZDE9S	1348016	0,730 €
1	J2	DB9 Macho	ITT CANNON	ZDE9P	1348011	0,700 €
1	J3	DB25 Macho	ITT CANNON	ZDB25P	1348014	1,000 €

Tabla 2: Listado de los componentes principales del circuito de la interfaz

Una vez realizado el esquema se procedió a comprobar el funcionamiento integrado con el puerto paralelo en una *protoboard*[29] y una vez validado se pasó a diseñar de la placa de circuito impreso, generando la *netlist*[30] asignando a cada componente su huella en la placa[31].

29 Placa de ensayo en la cual los componentes se "pinchan" evitando realizar soldaduras, permitiendo comprobar fácilmente el funcionamiento de un circuito electrónico y realizar modificaciones.

30 Asociación entre los terminales de los componentes electrónicos y su "huella" en la placa de circuito impreso.

31 Para este desarrollo se ha creado una biblioteca personalizada tanto de componentes como de huellas, de modo que no se vea afectado por cambios futuros en las versiones del programa y de sus bibliotecas incluidas de serie.

Por simplicidad y debido a la baja densidad de componentes se optó por utilizar como base una placa perforada de modo que las pistas se formarán uniendo con soldadura las isletas predefinidas.

En las figuras 68, 69, 70, 71, 72 y 73 se pueden observar las distintas capas del diseño del circuito impreso realizado. Nótese que también se han generado los ficheros *Gerber* correspondientes así como el fichero de perforaciones.

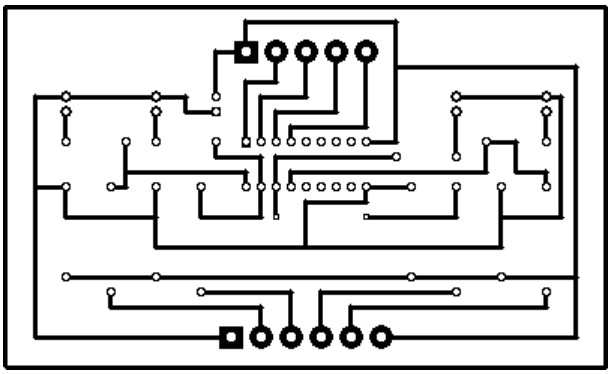


Figura 68: Pistas lado cobre

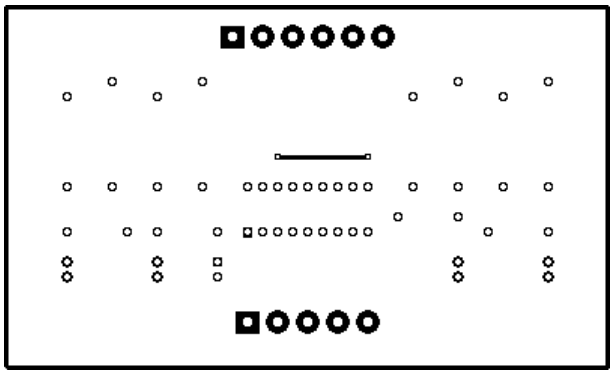


Figura 69: Pistas lado componentes

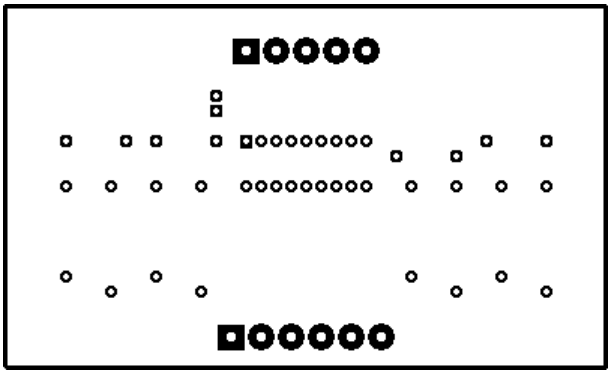


Figura 70: Máscara lado cobre

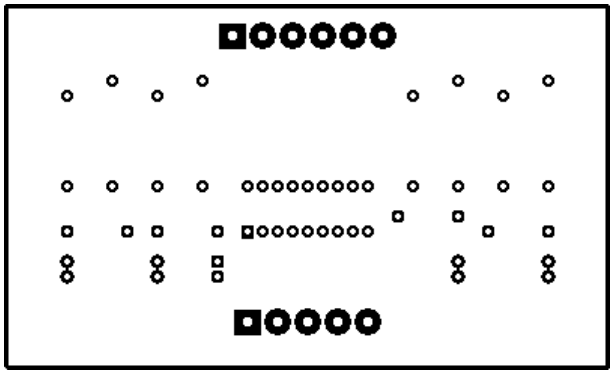


Figura 71: Máscara lado componentes

Figura 72: Contorno placa

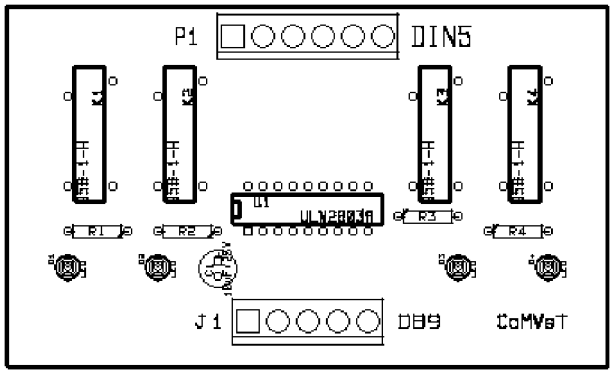


Figura 73: Serigrafía lado componentes

En las figuras 74 y 75 se puede observar la placa ya montada, tanto del lado de cobre como del lado de los componentes.

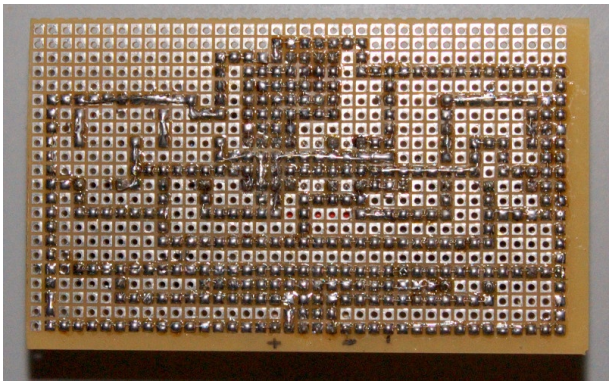


Figura 74: Placa lado cobre

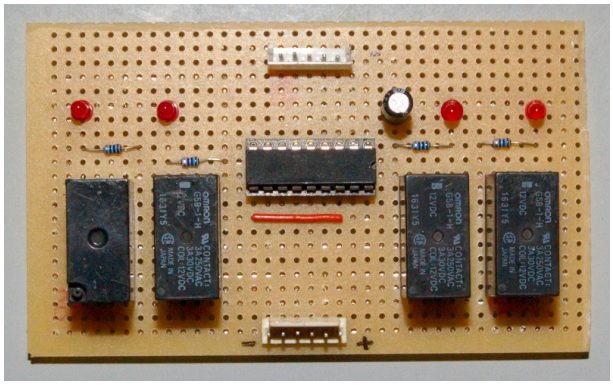


Figura 75: Placa lado componentes

Capítulo 6

Integración y pruebas

En esta fase final del desarrollo se ha procedido a conectar al puerto paralelo la placa de circuito impreso ya montada y ésta, a su vez, al mando del vehículo teledirigido empleando los cables específicamente construidos (Figura 76). A continuación se colocaron las baterías tanto al mando como al vehículo y se procedió a cargar la aplicación desarrollada en el PC.

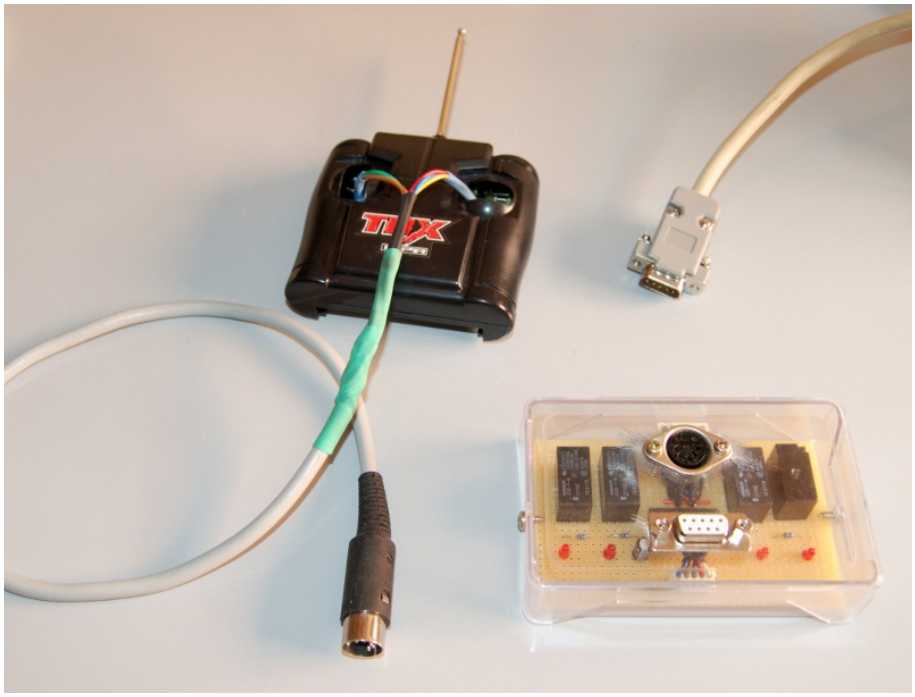


Figura 76: Integración PC-interfaz-mando

La primera prueba consistió en ejecutar las acciones manualmente utilizando para ello las flechas de dirección de la interfaz de usuario, comprobando que se activaban los relés correspondientes mediante la visualización de los LED's y que el vehículo también respondía ejecutando los movimientos adecuados. De esta forma se validó la correcta gestión del puerto paralelo, el cableado puerto-interfaz, el montaje de la placa (el diseño ya había sido validado en el apartado Desarrollo Hardware) y el cableado interfaz-mando.

Para la próxima prueba, fue necesario conectar el casco (la validación con el emulador fue realizada en el apartado Desarrollo Software), realizando la conexión con el **Motor Emotiv** para comprobar que la sucesión de eventos era la correcta y que se activaban las optimizaciones para

los algoritmos que se iban a utilizar.

Luego se cargó un perfil de usuario ya entrenado observando que la información visualizada en el cuadro de registro era correcta. Luego, una vez activado el casco, se comprobó que la calidad de contacto de los electrodos, así como la carga de la batería e intensidad de la señal inalámbrica eran detectadas y visualizadas correctamente (Figura 77).

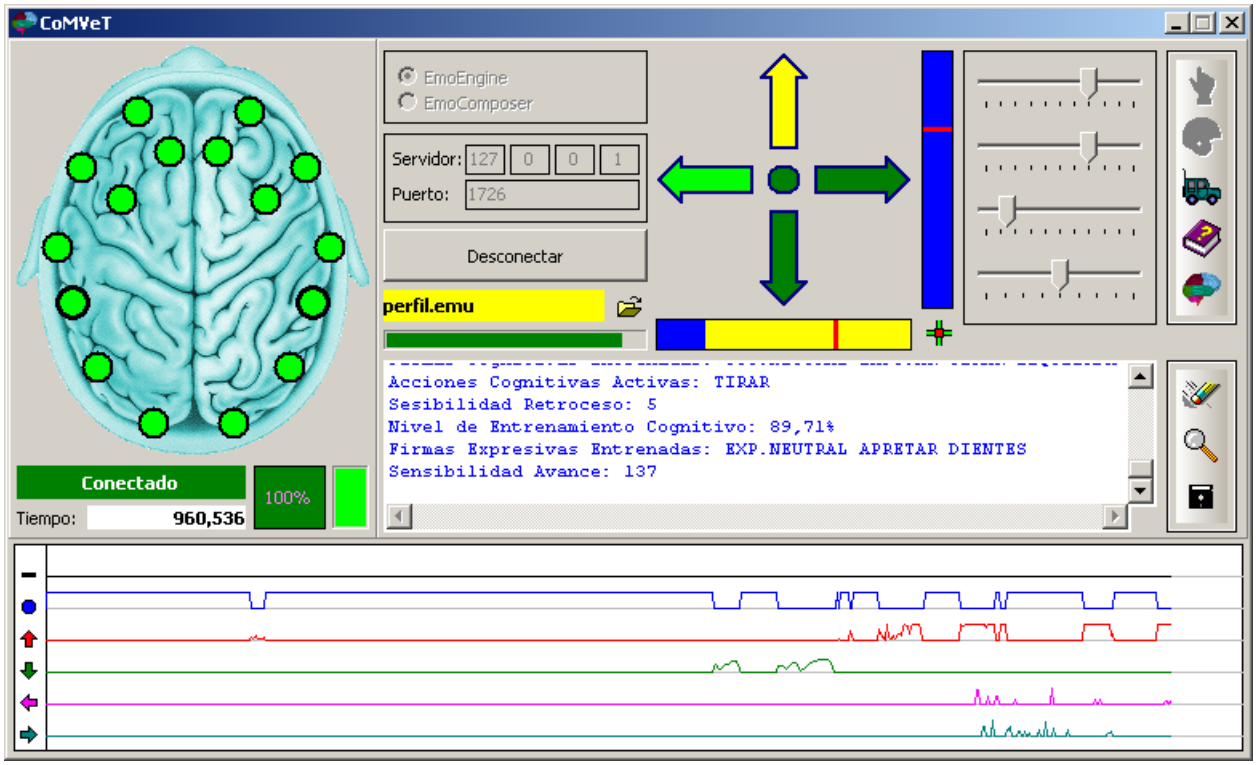


Figura 77: Aplicación conectada al Motor Emotiv

Por último se procedió a ejecutar las distintas acciones entrenadas comprobando su correcta detección y ajustando los niveles de los umbrales para obtener los mejores resultados, evitando falsas activaciones.

Para ello se realizó el gesto de "apretar dientes" provocando el avance del vehículo activándose el modo expresivo, posteriormente se pensó en la acción "tirar hacia atrás" para que el vehículo retrocediese mediante la activación en este caso del modo cognitivo. Finalmente se realizaron los giros hacia la izquierda y derecha de la cabeza para comprobar que las ruedas doblaban hacia sendas direcciones mediante la información que suministraba el acelerómetro.

De esta forma quedó validado el desarrollo realizado y la integración de las distintas partes, dando por finalizada esta fase del proyecto.

Capítulo 7

Conclusiones y trabajo futuro

Durante el proyecto expuesto en este libro se ha tomado contacto con el **Kit de Desarrollo Emotiv** instalando el software suministrado y poniendo en funcionamiento el casco **Emotiv Epoc** adquiriendo destreza en su uso. Además se ha abordado el estudio de la biblioteca de funciones provistas con la **API** para comprender mejor el funcionamiento del **Motor Emotiv** y tener una mejor perspectiva sobre las posibilidades de desarrollo disponibles.

Luego se ha afrontado el desarrollo de una aplicación para que sirviese como vínculo entre el **Motor Emotiv** y la interfaz para controlar el mando del vehículo teledirigido. Además debía permitir cargar perfiles de usuario, ajustar umbrales y sensibilidades, monitorizar el estado del casco y guardar en un registro los eventos recibidos.

A continuación se ha diseñado y construido la interfaz hardware para tomar las señales del puerto paralelo y aplicarlas al mando del vehículo teledirigido, sirviendo al mismo tiempo para monitorizar el funcionamiento gracias a una serie de LED's indicadores.

Por último se ha integrado el conjunto casco **Emotiv Epoc**, **Motor Emotiv**, **API**, aplicación desarrollada, cable puerto-interfaz, interfaz construida, cable interfaz-mando, mando y vehículo teledirigido (Figura 78), y se han realizado las comprobaciones de funcionamiento, validando el diseño y observando que se cumpliesen los objetivos planteados inicialmente.

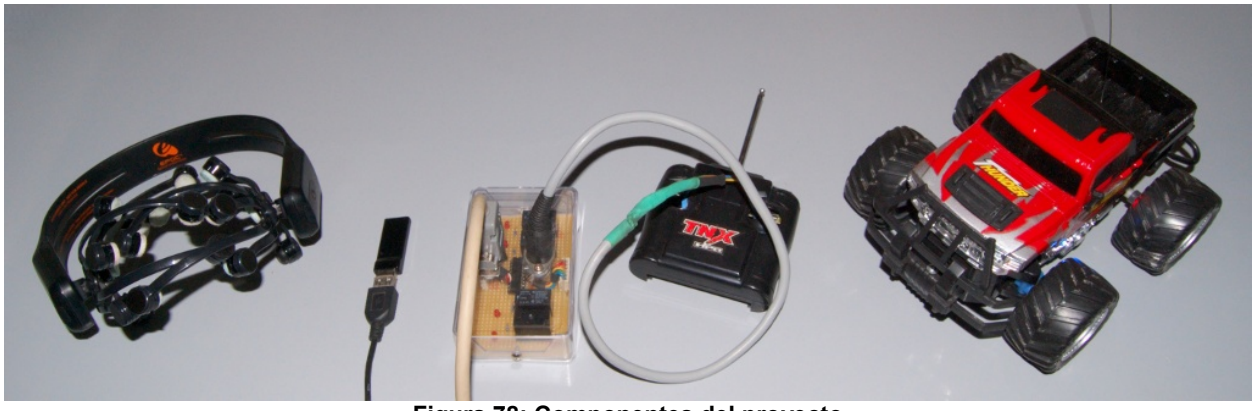


Figura 78: Componentes del proyecto

Los principales escollos que hubo que sortear fueron fundamentalmente los relacionados con la utilización del casco ya que cada vez que se lo iba a emplear era necesario humedecer y

montar los 16 electrodos. Luego, el entrenamiento del modo cognitivo ha requerido un gran esfuerzo mental para obtener un nivel elevado cercano al 100% y en cuanto al software, la necesidad de utilizar una biblioteca externa para acceder al puerto paralelo ante la imposibilidad de hacerlo en modo usuario en sistemas operativos Windows basados en el núcleo NT.

En cuanto a las mejoras y trabajo futuro sobre este desarrollo se destaca principalmente la utilización de esta tecnología en aplicaciones de uso cotidiano. Si bien, es verdad que el casco se puede considerar demasiado incómodo y aparatoso, también lo es que la tecnología avanza inexorablemente y que su principal baza en estos tiempos que corren siempre ha sido la miniaturización. Como ejemplo, ¿Quién no ha visto por la calle a personas hablando por el móvil utilizando auriculares BlueTooth? Algo que hace 10 años sonaba a ciencia ficción, hoy es de uso cotidiano, entonces no es de extrañar que en poco tiempo nos encontremos con mini-electrodos inalámbricos que se adhieran al cuero cabelludo y que transmitan su información utilizando tecnologías similares al RFID[32].

Asimismo un aumento en el número de sensores, así como de los algoritmos empleados, deberían mejorar la precisión de las detecciones, minimizando los falsos positivos, lo que en conjunto traería aparejado el surgimiento de nuevas aplicaciones especialmente útiles para personas con discapacidades motoras.

Conducir una silla de ruedas, escribir en un PC o controlar los sistemas domóticos de una vivienda, podrían ser actividades cotidianas para personas parapléjicas mejorando así su calidad de vida.

Otras aplicaciones posibles son la detección de somnolencia en un conductor desde el mismo vehículo, el cual lo obligaría a descansar antes de seguir conduciendo, la adecuación del entorno según el estado de ánimo del usuario y por supuesto toda una gama de aplicaciones lúdicas que ya se comienzan a vislumbrar en Internet. Esto se ve reforzado en que la tendencia de los fabricantes de consolas de videojuegos hoy en día, es más en centrarse en la interfaz usuario-máquina que en las capacidades gráficas del hardware que desarrollan, lo cual, hay que reconocerlo, ha tenido una respuesta excelente por parte del mercado.

Sin embargo, existen especulaciones [3] sobre los posibles efectos colaterales que puedan derivarse del uso de la interfaces cerebro-ordenador similares al síndrome de túnel carpiano[33]

32 *Radio Frequency Identification* - Identificación por Radiofrecuencia. Sistema de almacenamiento y recuperación de datos remotos que usa dispositivos denominados etiquetas, tarjetas, transpondedores o tags RFID [24]

33 Neuropatía periférica que ocurre cuando el nervio mediano, que abarca desde el antebrazo hasta la mano, se presiona o se atrapa dentro del túnel carpiano, a nivel de la muñeca. [26]

ocasionado por el uso del ratón o "el efecto brazo de gorila"[34] relacionado con el uso de pantallas táctiles dispuestas verticalmente o lápices ópticos, aunque en el caso que nos ocupa, van más encaminadas a una modificación de la forma de pensar del usuario debido a que los niveles de concentración y patrones cerebrales necesarios para interactuar con estas interfaces mentales difieren del comportamiento habitual que desempeñamos a diario.

34 Efecto de pesadez y cansancio en los brazos debido a que los seres humanos no se encuentran "diseñados" para realizar pequeños movimientos manteniendo sus brazos extendidos hacia delante. [12]

Apéndice A

Diagramas

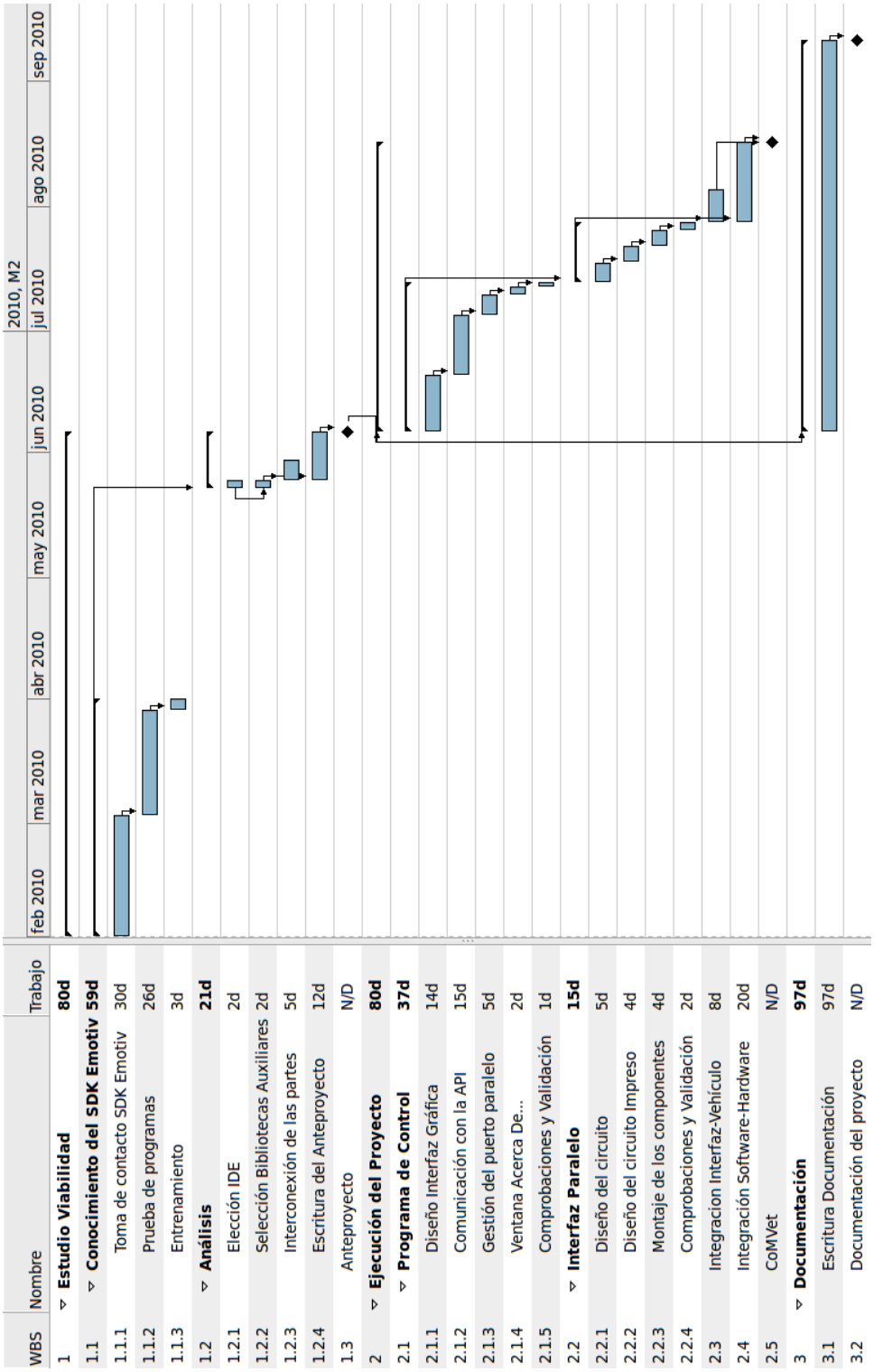
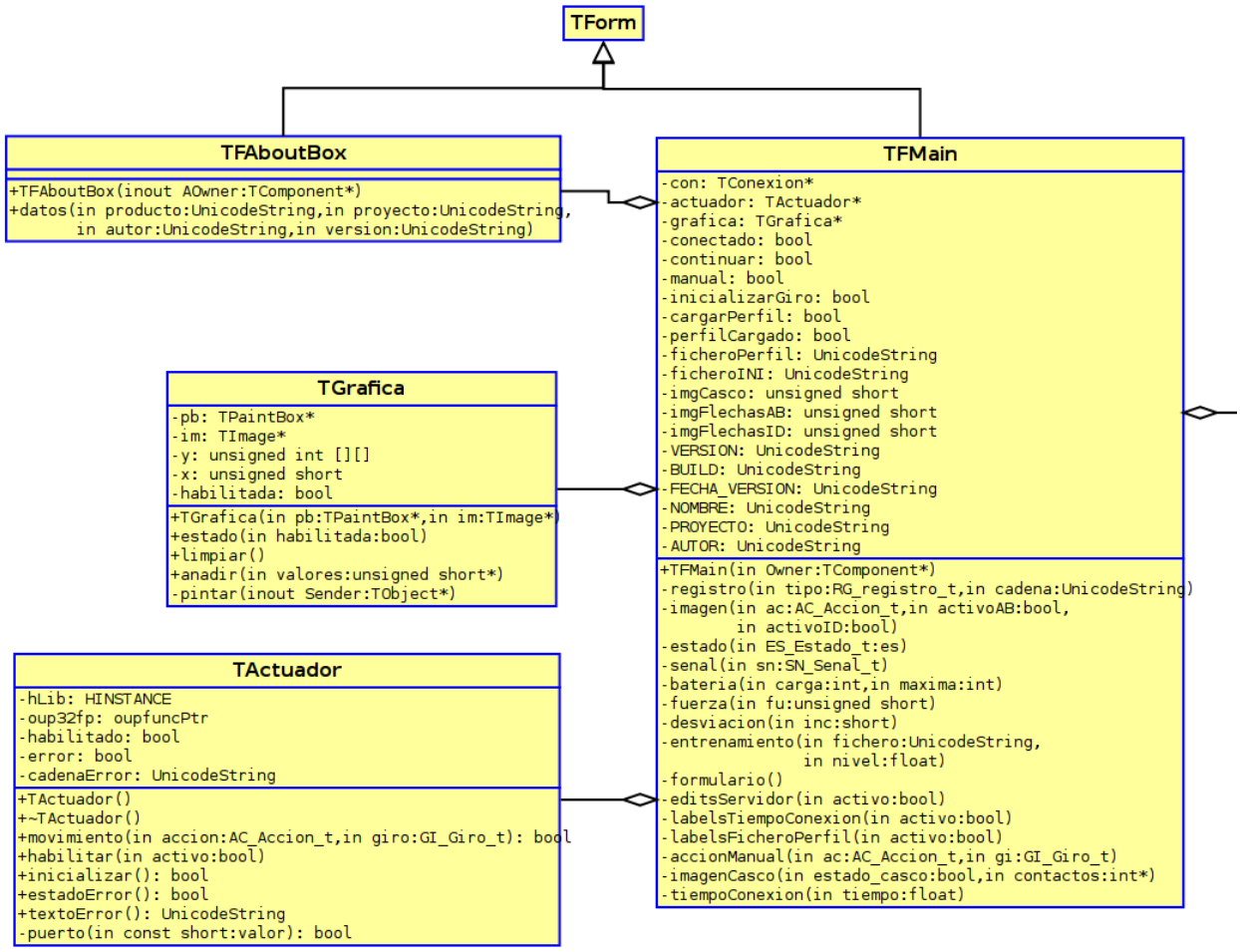


Figura 79: Planificación Completa


TForm
TFAboutBox
+TFAboutBox(inout AOwner:TComponent*)
+datos(in producto:UnicodeString,in proyecto:UnicodeString,
in autor:UnicodeString,in version:UnicodeString)
TFMain
-con: TConexion*
-actuador: TActuador*
-grafica: TGrafica*
-conectado: bool
-continuar: bool
-manual: bool
-inicializarGiro: bool
-cargarPerfil: bool
-perfilCargado: bool
-ficheroPerfil: UnicodeString
-ficheroINI: UnicodeString
-imgCasco: unsigned short
-imgFlechasAB: unsigned short
-imgFlechasID: unsigned short
-VERSION: UnicodeString
-BUILD: UnicodeString
-FECHA_VERSION: UnicodeString
-NOMBRE: UnicodeString
-PROYECTO: UnicodeString
-AUTOR: UnicodeString
+TFMain(in Owner:TComponent*)
-registro(in tipo:RG_registro_t,in cadena:UnicodeString)
-imagen(in ac:AC_Accion_t,in activoAB:bool,
in activoID:bool)
-estado(in ES_Estado_t:es)
-senal(in sn:SN_Senal_t)
-bateria(in carga:int,in maxima:int)
-fuerza(in fu:unsigned short)
-desviacion(in inc:short)
-entrenamiento(in fichero:UnicodeString,
in nivel:float)
-formulario()
-editsServidor(in activo:bool)
-labelsTiempoConexion(in activo:bool)
-labelsFicheroPerfil(in activo:bool)
-accionManual(in ac:AC_Accion_t,in gi:GI_Giro_t)
-imagenCasco(in estado_casco:bool,in contactos:int*)
-tiempoConexion(in tiempo:float)
TGrafica
-pb: TPaintBox*
-im: TImage*
-y: unsigned int [][]
-x: unsigned short
-habilitada: bool
+TGrafica(in pb:TPaintBox*,in im:TImage*)
+estado(in habilitada:bool)
+limpiar()
+anadir(in valores:unsigned short*)
-pintar(inout Sender:TObject*)
TActuador
-hLib: HINSTANCE
-oup32fp: oupfuncPtr
-habilitado: bool
-error: bool
-cadenaError: UnicodeString
+TActuador()
+~TActuador()
+movimiento(in accion:AC_Accion_t,in giro:GI_Giro_t): bool
+habilitar(in activo:bool)
+inicializar(): bool
+estadoError(): bool
+textoError(): UnicodeString
-puerto(in const short:valor): bool

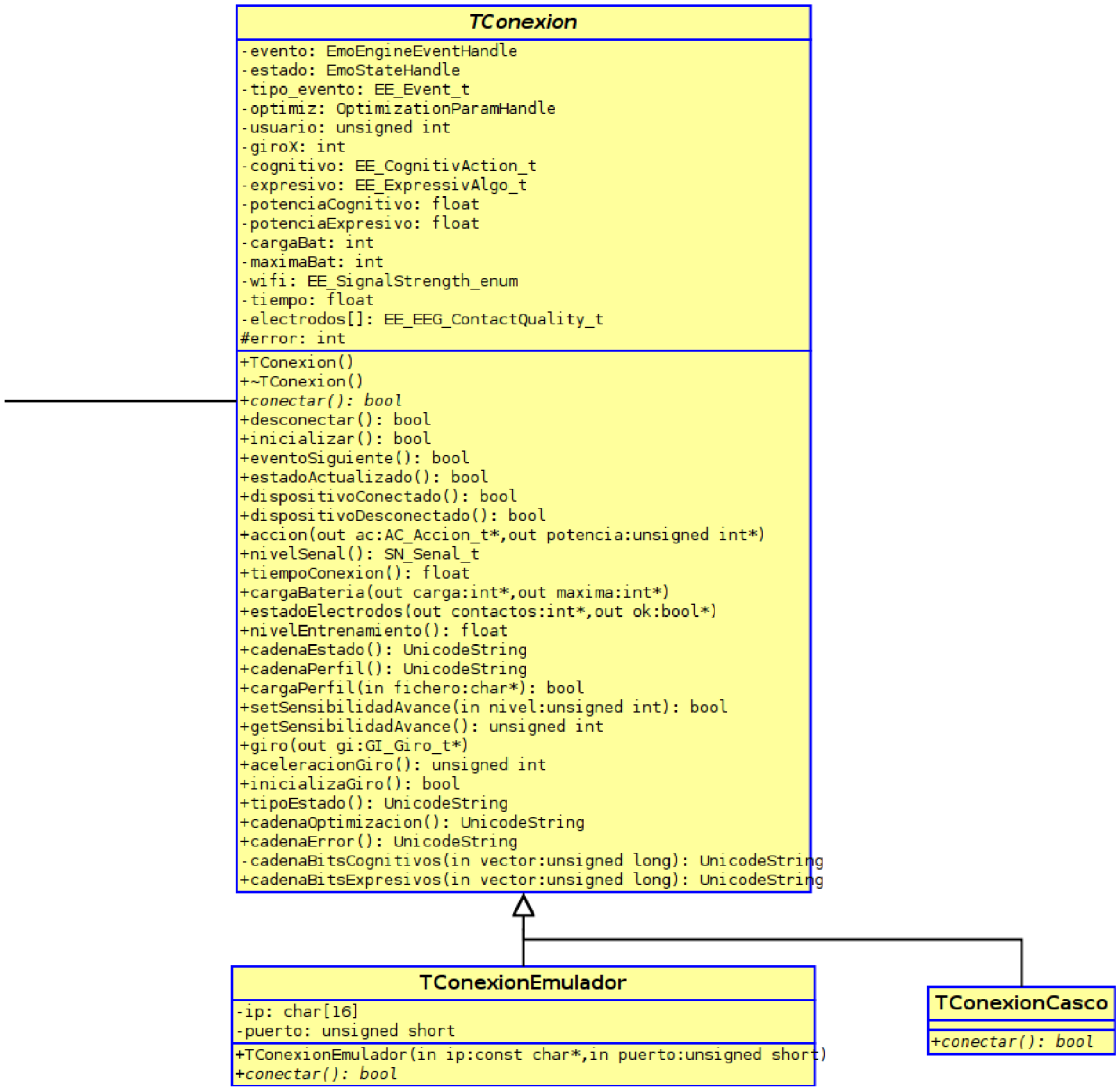


Figura 80: Diagrama de clases completo

Apéndice B

Cabeceras

I. About.h

```
/************************************************************
 * CoMVeT – Control Mental de Vehículos Teledirigidos       *
 * Daniel Héctor Stolfi Rosso                               *
 * 2010                                                     *
 *                                                          *
 * About.h                                                  *
 * Ventana con la información de acerca de... (cabecera)    *
 *                                                          *
 ************************************************************/
//---------------------------------------------------------------------------
#ifndef AboutH
#define AboutH
#include <Classes.hpp>
#include <Controls.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
#include <StdCtrls.hpp>
//---------------------------------------------------------------------------
class TFAboutBox : public TForm
{
__published:
   TPanel *PanelAbout;
   TImage *ImageIconoPrograma;
   TLabel *LabelProducto;
   TLabel *LabelVersion;
   TLabel *LabelAutor;
   TLabel *LabelComentarios;
   TButton *ButtonAceptar;
   TLabel *LabelProyecto;
private:

public:
   /* Constructor */
   virtual __fastcall TFAboutBox(TComponent* AOwner);

   /* datos(UnicodeString producto, UnicodeString proyecto, UnicodeString autor,
            UnicodeString version)
      Asigna los valores a ser mostrados en la ventana */
   void datos(UnicodeString producto, UnicodeString proyecto, UnicodeString autor,
              UnicodeString version);
};
//---------------------------------------------------------------------------
extern PACKAGE TFAboutBox *FAboutBox;
//---------------------------------------------------------------------------
#endif
```

Listado 9: About.h

II. Actuador.h

```
/************************************************************
 * CoMVeT – Control Mental de Vehículos Teledirigidos       *
```

```
 * Daniel Héctor Stolfi Rosso                                    *
 * 2010                                                          *
 *                                                               *
 * Actuador.h                                                    *
 * Gestiona la salida de las señales de control por el puerto    *
 * paralelo (cabecera)                                           *
 *                                                               *
 ****************************************************************/
//---------------------------------------------------------------------------
#ifndef ActuadorH
#define ActuadorH
#include "Tipos.h"
#include <system.hpp>
//---------------------------------------------------------------------------
/* Valores a volcar en el puerto paralelo
   para cada una de las acciones posibles */
const char PUERTO_NINGUNA   = 0;
const char PUERTO_CENTRO    = 0;
const char PUERTO_ARRIBA    = 1;
const char PUERTO_ABAJO     = 2;
const char PUERTO_IZQUIERDA = 4;
const char PUERTO_DERECHA   = 8;
/* Puerto de salida */
const short PUERTO_PARALELO = 0x378;
/* función de salida en dll */
typedef void (_stdcall *oupfuncPtr)(short portaddr, short datum);
//---------------------------------------------------------------------------
class TActuador
{
private:
   HINSTANCE hLib;            // Instancia para la dll que accede al puerto
   oupfuncPtr oup32fp;        // Puntero a la función de salida
   bool habilitado;           // Almacena el estado del actuador
   bool error;                // Estado de inicialización erróneo
   UnicodeString cadenaError; // Cadena con el último error producido

public:
   /* Constructor */
   TActuador();

   /* Destructor */
   ~TActuador();

   /* movimiento(AC_Accion_t accion, GI_Giro_t giro)
      Compone la acción y el giro pasados como parámetro y los vuelca en el puerto
      paralelo . Devuelve true si no se han producido errores */
   bool movimiento(AC_Accion_t accion, GI_Giro_t giro);

   /* habilitar(bool activo)
      habilita / inhabilita el volcado de los datos en el puerto paralelo
      devuelve el nuevo estado del actuador */
   bool habilitar(bool activo);

   /* inicializar()
      Inicializa el puerto volcando en las líneas el valor PUERTO_NINGUNA */
   bool inicializar();

   /* estadoError()
      Devuelve true si se ha producido un error */
   bool estadoError();

   /* textoError()
      Devuelve la descripción del último error ocurrido */
   UnicodeString textoError();
```

```
private:
   /* puerto(const short valor)
      Vuelca en el puerto paralelo el valor pasado como parámetro
      Devuelve true si no se han producido errores */
   bool puerto(const short valor);

};
#endif
```

Listado 10: Actuador.h

III. Conexion.h

```
/*************************************************************
 * CoMVeT – Control Mental de Vehículos Teledirigidos        *
 * Daniel Héctor Stolfi Rosso                                *
 * 2010                                                      *
 *                                                           *
 * Conexion.h                                                *
 * Clase abstracta que gestiona las conexiones (cabecera)    *
 * Se creará una instancia de una de sus clases hijas.       *
 *                                                           *
 *************************************************************/
// ---------------------------------------------------------------------------
#ifndef ConexionH
#define ConexionH
#include <_stddef.h>
#include <system.hpp>
#include "edk.h"
#include "Tipos.h"
// ---------------------------------------------------------------------------
/* Sensibilidad mínima giro */
const int GIRO_UMBRAL = 50;
/* Máximo giro para ponderar el valor de salida */
const int GIRO_MAXIMO = 300;
/* Número de señales de electrodos desde el API */
const unsigned int NUM_SENALES_ELECTRODOS = 18;
/* Acción expresiva a utilizar para avanzar */
const EE_ExpressivAlgo_t AVANZAR = EXP_CLENCH;
/* Acción cognitiva a utilizar para retroceder */
const EE_CognitivAction_t RETROCEDER = COG_PULL;

class TConexion {

private:
   EmoEngineEventHandle evento;     // Evento obtenido para operar con el estado
   EmoStateHandle estado;           // Estado actual
   EE_Event_t tipo_evento;          // Tipo de evento
   OptimizationParamHandle optimiz; // Optimizador de eventos cognitivos
   unsigned int usuario;            // Identificador del usuario
   int giroX;                       // Posición X giróscopo
   EE_CognitivAction_t cognitivo;   // Accion cognitiva detectada en el último evento
   EE_ExpressivAlgo_t expresivo;    // Accion expresiva detectada en el último evento
   float potenciaCognitivo;         // Nivel de potencia de la accion cognitiva
detectada
   float potenciaExpresivo;         // Nivel de potencia de la accion expresiva
detectada
   int cargaBat, maximaBat;         // Nivel de carga y carga máxima de la batería
   EE_SignalStrength_enum wifi;     // Nivel de señal del enlace inalámbrico
   float tiempo;                    // Marca temporal del último evento recibido
   EE_EEG_ContactQuality_t electrodos[NUM_SENALES_ELECTRODOS]; // Vector con el
                                                               // estado de los
                                                               // electrodos
```

```
protected:
   int error;                        // Código de error devuelto por la última llamada
                                     // a función

public:
   /* Constructor */
   TConexion();

   /* Destructor */
   ~TConexion();

   /* conectar()
      Inicia la conexión. Método abstracto que se implementa en las clases hijas */
   virtual bool conectar() = 0;

   /* desconectar()
      Desconecta del casco o del emulador
      Devuelve true si no se han producido errores */
   bool desconectar();

   /* inicializar()
      Inicializa la conexión activando el optimizador
      Devuelve true si no se han producido errores */
   bool inicializar();

   /* eventoSiguiente()
      Recupera el evento siguiente y devuelve true si ha tenido éxito */
   bool eventoSiguiente();

   /* estadoActualizado()
      Devuelve true si se ha actualizado el estado */
   bool estadoActualizado();

   /* dispositivoConectado()
      Devuelve true si se ha conectado el dispositivo */
   bool dispositivoConectado();

   /* dispositivoDesconectado()
      Devuelve true si se ha desconectado el dispositivo */
   bool dispositivoDesconectado();

   /* accion(AC_Accion_t* ac, unsigned int* potencia)
      Devuelve por referencia el tipo de acción del estado actual
      y el nivel de potencia detectado (0 - 10) */
   void accion(AC_Accion_t* ac, unsigned int* potencia);

   /* nivelSenal()
      Devuelve el nivel de señal del enlace wifi */
   SN_Senal_t nivelSenal();

   /* tiempoConexion()
      Devuelve los segundos transcurridos desde la conexión */
   float tiempoConexion();

   /* cargaBateria(int *carga, int *maxima)
      Devuelve por referencia la carga de la bateria y el nivel máximo de carga */
   void cargaBateria(int* carga, int* maxima);

   /* estadoElectrodos(int* contactos, bool* ok)
      Devuelve por referencia el array de electrodos con la calidad de contacto de
      cada uno y si el estado es suficiente para una detección de calidad */
   void estadoElectrodos(int* contactos, bool* ok);

   /* nivelEntrenamiento()
      Devuelve el nivel de entrenamiento del perfil de usuario activo (0.0 - 1.0) */
```

```
   float nivelEntrenamiento();

   /* cadenaEstado()
      Devuelve en una cadena el contenido del último estado detectado */
   UnicodeString cadenaEstado();

   /* cadenaPerfil()
      Devuelve una cadena descriptiva del perfil de usuario activo */
   UnicodeString cadenaPerfil();

   /* cargaPerfil(char* fichero)
      Carga desde el fichero el perfil de usuario
      Devuelve true si no se han producido errores */
   bool cargaPerfil(char* fichero);

   /* setSensibilidadAvance(unsigned int nivel)
      Cambia la sensibilidad para la acción de avance (0 - 1000) en el perfil de
      usuario activo . Devuelve true si no se han producido errores */
   bool setSensibilidadAvance(unsigned int nivel);

   /* getSensibilidadAvance()
      Devuelve la sensibilidad para la acción de avance (0 - 1000) del perfil de
      usuario activo */
   unsigned int getSensibilidadAvance();

   /* setSensibilidadRetroceso(unsigned int nivel)
      Cambia la sensibilidad para la acción de retroceso (1 - 9) en el perfil de
      usuario activo . Devuelve true si no se han producido errores */
   bool setSensibilidadRetroceso(unsigned int nivel);

   /* getSensibilidadRetroceso()
      Devuelve la sensibilidad para la acción de retroceso (1 - 9) del perfil de
      usuario activo */
   unsigned int getSensibilidadRetroceso();

   /* giro(GI_Giro_t* gi)
      Devuelve la acción detectada desde el acelerómetro */
   void giro(GI_Giro_t* gi);

   /* aceleracionGiro()
      Devuelve el valor diferencial del movimiento detectado */
   unsigned int aceleracionGiro();

   /* inicializaGiro()
      Inicializa el acelerómetro en la posición actual del casco */
   bool inicializaGiro();

   /* tipoEstado()
      Devuelve una cadena descriptiva del último estado detectado */
   UnicodeString tipoEstado();

   /* cadenaOptimizacion()
      Devuelve una cadena con la información de las optimizaciones activas */
   UnicodeString cadenaOptimizacion();

   /* cadenaError()
      Devuelve una cadena con la descripción del último error acontecido */
   UnicodeString cadenaError();

private:
   /* cadenaBitsCognitivos(unsigned long vector)
      Convierte los bits cognitivos utilizados por el programa que se obtienen
      desde el vector pasado como parámetro en una cadena de caracteres */
   UnicodeString cadenaBitsCognitivos(unsigned long vector);
```

```
   /* cadenaBitsExpresivos(unsigned long vector)
      Convierte los bits expresivos utilizados por el programa que se obtienen
      desde el vector pasado como parámetro en una cadena de caracteres */
   UnicodeString cadenaBitsExpresivos(unsigned long vector);

};
#endif
```

Listado 11: Conexion.h

IV. ConexionCasco.h

```
/*************************************************************
 * CoMVeT – Control Mental de Vehículos Teledirigidos        *
 * Daniel Héctor Stolfi Rosso                                *
 * 2010                                                      *
 *                                                           *
 * ConexionCasco.h                                           *
 * Clase hija de Conexion que particulariza la conexión con el *
 * casco emotiv (cabecera)                                   *
 *                                                           *
 *************************************************************/
//---------------------------------------------------------------------------
#ifndef ConexionCascoH
#define ConexionCascoH
#include "Conexion.h"
//---------------------------------------------------------------------------
class TConexionCasco : public TConexion
{
private:

public:
   /* conectar()
      Conecta con el casco Emotiv */
   virtual bool conectar();
};
#endif
```

Listado 12: ConexionCasco.h

V. ConexionEmulador.h

```
/*************************************************************
 * CoMVeT – Control Mental de Vehículos Teledirigidos        *
 * Daniel Héctor Stolfi Rosso                                *
 * 2010                                                      *
 *                                                           *
 * ConexionEmulador.h                                        *
 * Clase hija de Conexion que particulariza la conexión con el *
 * emulador emoComposer (cabecera)                           *
 *                                                           *
 *************************************************************/
//---------------------------------------------------------------------------
#ifndef ConexionEmuladorH
#define ConexionEmuladorH
#include "Conexion.h"
#include <_str.h>
//---------------------------------------------------------------------------
class TConexionEmulador : public TConexion
{
private:
   char ip[16];
   unsigned short puerto;
public:
```

```
   /* Constructor
      Se especifica la dirección ip y el puerto del emulador */
   TConexionEmulador(const char* ip, unsigned short puerto);

   /* conectar()
      Conecta con el emulador
      Devuelve true si no se han producido errores */
   virtual bool conectar();
};
#endif
```

Listado 13: ConexionEmulador.h

VI. Grafica.h

```
/*************************************************************
 * CoMVeT – Control Mental de Vehículos Teledirigidos        *
 * Daniel Héctor Stolfi Rosso                                *
 * 2010                                                      *
 *                                                           *
 * Grafica.h                                                 *
 * Clase que gestiona la gráfica de señales (cabecera)       *
 *                                                           *
 *************************************************************/
//---------------------------------------------------------------------------
#ifndef GraficaH
#define GraficaH
#include <Graphics.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
/* Señales gráfica */
typedef enum SenalesGrafica {
   SG_Ninguna = 0, SG_Centro, SG_Adelante, SG_Atras, SG_Izquierda, SG_Derecha
} SG_SenalesGrafica_t;
/* Número de gráficas */
static const unsigned short NUM_GRAFICAS=6;
/* Número de puntos */
static const unsigned short NUM_PUNTOS=748;
/* Desplazamiento de avance */
static const unsigned short DESPLAZAMIENTO=50;
/* Posiciones gráficas */
static const unsigned short y0[NUM_GRAFICAS] = {20,40,60,80,100,120};
/* Colores gráficas */
static const TColor color[NUM_GRAFICAS] = {clBlack, clBlue, clRed, clGreen,
clFuchsia, clTeal};

class TGrafica {
private:
   TPaintBox *pb;                              // PaintBox sobre la que se dibujará
   TImage *im;                                 // Image con las referencias
   unsigned int y[NUM_GRAFICAS][NUM_PUNTOS];   // Matriz con los valores a dibujar
   unsigned short x;                           // Coordenada x del último valor
   bool habilitada;                            // Almacena el estado de la gráfica

public:
   /* Constructor
      Recibe el puntero al PaintBox que se utilizará como gráfica y
      a la imagen con las referencias */
   TGrafica(TPaintBox *pb, TImage *im);

   /* estado(bool habilitada)
      Habilita / inhabilita la gráfica */
   void estado(bool habilitada);
```

```
   /* limpiar()
      Vacía el contenido de la gráfica */
   void limpiar();

   /* anadir(unsigned short* valores)
      Añade una nueva lista de valores pasados en el parámetro */
   void anadir(unsigned short* valores);

private:
   /* Dibuja sobre el PaintBox */
   void __fastcall pintar(TObject *Sender);

};
#endif
```

Listado 14: Grafica.h

VII. Main.h

```
/*************************************************************
 * CoMVeT – Control Mental de Vehículos Teledirigidos        *
 * Daniel Héctor Stolfi Rosso                                *
 * 2010                                                      *
 *                                                           *
 * Main.h                                                    *
 * Interfaz gráfica y punto de entrada del programa (cabecera) *
 *                                                           *
 *************************************************************/
//---------------------------------------------------------------------------
#ifndef MainH
#define MainH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ComCtrls.hpp>
#include <ExtCtrls.hpp>
#include "cgauges.h"
#include <ToolWin.hpp>
#include <Graphics.hpp>
#include <ImgList.hpp>
#include <ActnList.hpp>
#include <ActnMan.hpp>
#include <PlatformDefaultStyleActnCtrls.hpp>
#include <StdActns.hpp>
#include "Tipos.h"
#include "Conexion.h"
#include "Actuador.h"
#include "Grafica.h"
#include <Dialogs.hpp>
//---------------------------------------------------------------------------
/* Número de electrodos en el casco */
const unsigned short NUM_ELECTRODOS = 16;
/* Colores texto registro según tipo */
const TColor COLORES_REGISTRO[] = {clWindowText, clBlue, clRed};
/* Numero de estados posibles para los electrodos del casco */
const unsigned short NUM_ESTADOS = 5;
/* Colores para cada estado posible de los electrodos del casco */
const TColor ESTADOS[NUM_ESTADOS] = {clBlack, clRed, TColor(33023), clYellow,
clLime};
/* Puerto por defecto del emulador */
const UnicodeString PUERTO_EMULADOR = "1726";
```

```
//---------------------------------------------------------------------------
class TFMain : public TForm
{
__published: // IDE-managed Components
    TRichEdit *RichEditRegistro;
    TPanel *PanelSuperior;
    TPanel *PanelConectar;
    TRadioGroup *RadioGroupMotor;
    TGroupBox *GroupBoxServidor;
    TEdit *EditIp1;
    TEdit *EditPuerto;
    TBitBtn *BitBtnConectar;
    TBitBtn *BitBtnDesconectar;
    TLabel *LabelServidor;
    TLabel *LabelPuerto;
    TCGauge *CGaugeFuerza;
    TImage *ImageFlechasArribaAbajo;
    TToolBar *ToolBarPrincipal;
    TToolButton *ToolButtonManual;
    TToolButton *ToolButtonCasco;
    TToolButton *ToolButtonActuador;
    TImageList *ImageListFlechasArribaAbajo;
    TActionManager *ActionManager;
    TAction *ActionCasco;
    TAction *ActionManual;
    TPanel *PanelContenedor;
    TPanel *PanelSensores;
    TPanel *PanelGrafica;
    TImage *ImageCasco;
    TPanel *PanelIndicadores;
    TLabel *LabelEstado;
    TCGauge *CGaugeBateria;
    TAction *ActionActuador;
    TProgressBar *ProgressBarSenal;
    TAction *ActionConectar;
    TAction *ActionDesconectar;
    TPaintBox *PaintBoxGrafica;
    TEdit *EditIp2;
    TEdit *EditIp3;
    TEdit *EditIp4;
    TShape *ShapeElectodo01;
    TShape *ShapeElectodo02;
    TShape *ShapeElectodo03;
    TShape *ShapeElectodo04;
    TShape *ShapeElectodo05;
    TShape *ShapeElectodo06;
    TShape *ShapeElectodo07;
    TShape *ShapeElectodo08;
    TShape *ShapeElectodo09;
    TShape *ShapeElectodo10;
    TShape *ShapeElectodo11;
    TShape *ShapeElectodo12;
    TShape *ShapeElectodo13;
    TShape *ShapeElectodo14;
    TShape *ShapeElectodo15;
    TShape *ShapeElectodo16;
    TImageList *ImageListCasco;
    TAction *ActionCargaPerfil;
    TOpenDialog *OpenDialog;
    TImage *ImageGrafica;
    TLabel *LabelTiempoConexionEtq;
    TLabel *LabelTiempoConexion;
    TPanel *PanelRegistro;
    TToolBar *ToolBarRegistro;
```

```
    TToolButton *ToolButtonLimpiar;
    TToolButton *ToolButtonGuardar;
    TToolButton *ToolButtonEstados;
    TToolButton *ToolButtonAyuda;
    TToolButton *ToolButtonAcercaDe;
    TAction *ActionLimpiar;
    TAction *ActionGuardar;
    TAction *ActionEstados;
    TAction *ActionAyuda;
    TAction *ActionAcercaDe;
    TSaveDialog *SaveDialog;
    TImageList *ImageListToolBars;
    TImage *ImageFlechasIzquierdaDerecha;
    TImageList *ImageListFlechasIzquierdaDerecha;
    TCGauge *CGaugeGiro;
    TProgressBar *ProgressBarEntrenamiento;
    TGroupBox *GroupBoxTrackBar;
    TTrackBar *TrackBarUmbralAvRet;
    TTrackBar *TrackBarAvance;
    TTrackBar *TrackBarUmbralGiro;
    TTrackBar *TrackBarRetroceso;
    TSpeedButton *SpeedButtonCentrarCasco;
    TAction *ActionCentrarCasco;
    TPanel *PanelPerfil;
    TLabel *LabelFicheroPerfil;
    TSpeedButton *SpeedButtonPerfil;
    TShape *ShapeIndicadorGiro;
    TShape *ShapeIndicadorFuerza;
    TPanel *PanelCalibrandoAcelerometros;
    TLabel *LabelCalibrando;
    TLabel *LabelPermanezca;
    void __fastcall RadioGroupMotorClick(TObject *Sender);
    void __fastcall ActionManualExecute(TObject *Sender);
    void __fastcall ActionCascoExecute(TObject *Sender);
    void __fastcall ActionConectarExecute(TObject *Sender);
    void __fastcall ActionDesconectarExecute(TObject *Sender);
    void __fastcall ImageFlechasArribaAbajoMouseDown(TObject *Sender,
                                                     TMouseButton Button,
                                                     TShiftState Shift,
                                                     int X, int Y);
    void __fastcall ImageFlechasIzquierdaDerechaMouseDown(TObject *Sender,
                                                          TMouseButton Button,
                                                          TShiftState Shift,
                                                          int X, int Y);
    void __fastcall ImageFlechasArribaAbajoMouseUp(TObject *Sender,
                                                   TMouseButton Button,
                                                   TShiftState Shift,
                                                   int X, int Y);
    void __fastcall ActionActuadorExecute(TObject *Sender);
    void __fastcall FormDestroy(TObject *Sender);
    void __fastcall FormCloseQuery(TObject *Sender, bool &CanClose);
    void __fastcall EditPuertoExit(TObject *Sender);
    void __fastcall EditIp1Exit(TObject *Sender);
    void __fastcall ActionCargaPerfilExecute(TObject *Sender);
    void __fastcall TrackBarUmbralAvRetChange(TObject *Sender);
    void __fastcall TrackBarUmbralGiroChange(TObject *Sender);
    void __fastcall TrackBarAvanceChange(TObject *Sender);
    void __fastcall TrackBarRetrocesoChange(TObject *Sender);
    void __fastcall ActionLimpiarExecute(TObject *Sender);
    void __fastcall ActionEstadosExecute(TObject *Sender);
    void __fastcall ActionGuardarExecute(TObject *Sender);
    void __fastcall ActionAyudaExecute(TObject *Sender);
    void __fastcall ActionAcercaDeExecute(TObject *Sender);
    void __fastcall ActionCentrarCascoExecute(TObject *Sender);
```

```
private:     // User declarations

   TConexion *con;              // Puntero a la instancia de conexión
   TActuador *actuador;         // Puntero a la instancia de actuador
   TGrafica *grafica;           // Puntero a la instancia de gráfica
   bool conectado;              // Indica si se encuentra conectado
   bool continuar;              // Control del bucle de lectura de eventos
   bool manual;                 // Indica que se encuentra en el modo manual
   bool inicializarGiro;        // Indica la petición pendiente de puesta a cero del
                                // acelerómetro
   bool cargarPerfil;           // Indica la petición pendiente de carga de perfil
   bool perfilCargado;          // Indica si se ha cargado un perfil
   UnicodeString ficheroPerfil; // Cadena con la ruta al fichero de perfil
   UnicodeString ficheroINI;    // Cadena con la ruta al fichero INI
   GI_Giro_t giroManual;        // Indica dirección de giro
   unsigned short imgCasco;     // Índice de la imagen del casco actual
   unsigned short imgFlechasAB; // Índice de la imagen de dirección actual
                                // (Arriba-Abajo)
   unsigned short imgFlechasID; // Índice de la imagen de dirección actual
                                // (Izquierda-Derecha)
   UnicodeString VERSION;       // Versión del programa
   UnicodeString BUILD;         // Versión del build
   UnicodeString FECHA_VERSION; // Fecha del programa
   UnicodeString NOMBRE;        // Nombre del programa
   UnicodeString PROYECTO;      // Nombre del proyecto
   UnicodeString AUTOR;         // Nombre del autor

   /* registro(RG_Registro_t tipo, UnicodeString cadena)
      Registra el mensaje que se recibe en el parámetro cadena
      del tipo indicado en el parámetro tipo */
   void registro(RG_Registro_t tipo, UnicodeString cadena);

   /* imagen(AC_Accion_t ac, GI_Giro_t gi, bool activoAB, bool activoID)
      Cambia la imagen según la acción, giro y los estados de actividad recibidos
      como parámetro */
   void imagen(AC_Accion_t ac, GI_Giro_t gi, bool activoAB, bool activoID);

   /* estado(ES_Estado_t es)
      Representa en la interfaz gráfica el estado que se recibe como parámetro */
   void estado(ES_Estado_t es);

   /* senal(SN_Senal_t sn)
      Representa en la interfaz gráfica el valor de la señal de enlace con el casco
      que se recibe por parámetro a través del parámetro */
   void senal(SN_Senal_t sn);

   /* bateria(int carga, int maxima)
      Representa en la interfaz gráfica el nivel de carga de batería.
      Recibe como parámetro el valor de la carga y la carga máxima posible */
   void bateria(int carga, int maxima);

   /* fuerza(unsigned short fu)
      Representa en la interfaz gráfica el nivel de fuerza del evento detectado
      que se proporciona como parámetro (0 - 10) */
   void fuerza(unsigned short fu);

   /* desviacion(short inc)
      Representa en la interfaz gráfica el nivel de fuerza del evento detectado
      que se proporciona como parámetro (0 - 10) */
   void desviacion(short inc);

   /* entrenamiento(UnicodeString fichero, float nivel)
      Representa en la interfaz gráfica el fichero del perfil de usuario y
      el porcentual del nivel de entrenamiento (0.0 - 100.0) */
   void entrenamiento(UnicodeString fichero, float nivel);
```

```
   /* formulario()
      Configura el formulario según el estado de la conexión
      y si se encuentra en el modo manual */
   void formulario();

   /* editsServidor(bool activo)
      Activa/desactiva los edits con los parámetros del servidor */
   void editsServidor(bool activo);

   /* labelsTiempoConexion(bool activo)
      Activa/desactiva las etiquetas de tiempo de conexión */
   void labelsTiempoConexion(bool activo);

   /* labelsFicheroPerfil(bool activo)
      Activa/desactiva las etiquetas del fichero del perfil */
   void labelsFicheroPerfil(bool activo);

   /* accionManual(AC_Accion_t ac, GI_Giro_t gi)
      Ejecuta de forma manual la acción compuesta por los parámetros pasados */
   void accionManual(AC_Accion_t ac, GI_Giro_t gi);

   /* imagenCasco(bool estado_casco, int *contactos)
      Representa en la interfaz gráfica los estados de los contactos
      recibidos en el array contactos o bien los inhabilita a todos mediante el
      valor del parámetro estado_casco */
   void imagenCasco(bool estado_casco, int *contactos);

   /* tiempoConexion(float tiempo)
      Actualiza la etiqueta con el tiempo de conexión */
   void tiempoConexion(float tiempo);

public:
   /* Constructor */
       __fastcall TFMain(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TFMain *FMain;
//---------------------------------------------------------------------------
#endif
```

Listado 15: Main.h

VIII. Tipos.h

```
/*************************************************************
 * CoMVeT – Control Mental de Vehículos Teledirigidos        *
 * Daniel Héctor Stolfi Rosso                                *
 * 2010                                                      *
 *                                                           *
 * Tipos.h                                                   *
 * Define los tipos y las enumaraciones a utilizar dentro del  *
 * programa.                                                 *
 *                                                           *
 *************************************************************/
// ---------------------------------------------------------------------------

#ifndef TiposH
#define TiposH
// ---------------------------------------------------------------------------

/* Acciones */
typedef enum Accion {
   AC_Ninguna = 0, AC_Centro, AC_Adelante, AC_Atras
```

```
} AC_Accion_t;

/* Dirección de giro */
typedef enum Giro {
   GI_Ninguno = 0, GI_Izquierda, GI_Derecha
} GI_Giro_t;

/* Niveles de señal */
typedef enum Senal {
   SN_Ninguna = 0, SN_No, SN_Mala, SN_Buena
} SN_Senal_t;

/* Estados GUI */
typedef enum Estado {
   ES_Desconectado = 0, ES_NoDetectado, ES_Error, ES_Conectado, ES_Ruidoso,
ES_Perfil, ES_NoSenal
} ES_Estado_t;

/* Tipo de Registro */
typedef enum Registro {
   RG_Estado = 0, RG_Mensaje, RG_Error
} RG_Registro_t;

#endif
```

Listado 16: Tipos.h

Apéndice C

Índices

Índice de Figuras

Índice de Listados

Bibliografía

[1] 10-20 system – Wikipedia (Consultado en agosto de 2010):
http://en.wikipedia.org/wiki/10-20_system_(EEG)

[2] Allegro MicroSystems, Inc. “2803 thru 2804 high-voltage high-current darlington arrays”

[3] Alvy – Cooking ideas (Consultado en agosto de 2010):
http://www.cookingideas.es/brazos-de-gorila-nuevas-interfaces-cerebro-ordenador-20100720.html

[4] Apófisis Mastoides – Wikipedia (Consultado en agosto de 2010):
http://es.wikipedia.org/wiki/Apófisis_mastoides

[5] Corteza Cerebral – Wikipedia (Consultado en agosto de 2010):
http://es.wikipedia.org/wiki/Corteza_cerebral

[6] David Jahshan (2006) “KiCad Step by Step Tutorial ”

[7] Diodo Emisor de Luz – Wikipedia (Consultado en agosto de 2010):
http://es.wikipedia.org/wiki/Diodo_emisor_de_luz

[8] Electroencephalography – Wikipedia (Consultado en agosto de 2010):
http://en.wikipedia.org/wiki/Eeg

[9] Embarcadero Technologies, Inc. (2009) “Embarcadero RAD Studio Documentation”

[10] Emotiv Software Development Kit “User Manual for Beta Release 1.0.x.“

[11] Gerber – Wikipedia (Consultado en agosto de 2010):
http://es.wikipedia.org/wiki/Gerber_(formato_de_fichero)

[12] Gorilla arm – The Jargon File (Consultado en agosto de 2010):
http://catb.org/jargon/html/G/gorilla-arm.html

[13] Inion – Wikipedia (Consultado en agosto de 2010):
http://en.wikipedia.org/wiki/Inion

[14] ITW McMurdo Connectors “D range connectors”

[15] Manuel Ujaldón Martínez. - Editorial Ciencia-3, S.L. (2003)

"Arquitectura del PC – Volumen II: La información Memorias y buses"

[16] Nasion – Wikipedia (Consultado en agosto de 2010):

http://en.wikipedia.org/wiki/Nasion

[17] Neutrik AG "DIN Connectors"

[18] OMRON Electronic Components LLC "Miniature Power Relay G5B"

[19] Premier Farnell Group (Consultado en agosto de 2010): http://es.farnell.com/

[20] Premier Farnell Group (2004) "Multicomp - Ultra-Miniaturized Radial Capacitors "

[21] Premier Farnell Group (2007) "Multicomp - Metal Oxide Film Resistors "

[22] Premier Farnell Group (2009) "Multicomp - DIN Socket - 5 Way "

[23] Rafel Barea Navarro – Universidad de Alcalá

"Instrumentación Biomédica. Tema 5. Electroencefalografía"

[24] RFID – Wikipedia (Consultado en agosto de 2010):

http://es.wikipedia.org/wiki/RFID

[25] SGS-Thomson Microelectronics (1997) "ULN2801 A Eight darlington arrays"

[26] Síndrome del túnel carpiano – Wikipedia (Consultado en agosto de 2010):

http://es.wikipedia.org/wiki/Síndrome_del_túnel_carpiano

[27] Transistor Darlington – Wikipedia (Consultado en agosto de 2010):

http://es.wikipedia.org/wiki/Transistor_Darlington

[28] Vértice – Wikipedia (Consultado en agosto de 2010):

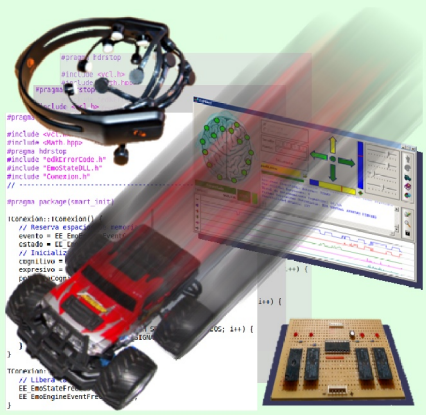
http://es.wikipedia.org/wiki/Vértice_(anatomía)

[29] VCL – Wikipedia (Consultado en agosto de 2010):

http://en.wikipedia.org/wiki/Visual_Component_Library

[30] Vishay Intertechnology, Inc "High Efficiency LED in Ø 3 mm Tinted Diffused Package "

CoMVeT
Control Mental de Vehículos Teledirigidos



Daniel Héctor Stolfi Rosso
Sergio Gálvez Rojas

La vertiginosa evolución en la forma con que los usuarios interaccionan con los ordenadores actuales está aproximando al presente lo que, hasta hace muy poco, eran ideas futuristas de ciencia ficción. Los monitores tradicionales han sido sustituidos en poco tiempo por pantallas TFT y de LED que, en breve, desaparecerán para dar paso a otras 3D en las que no será necesario el uso de gafas especiales. Los dispositivos de entrada tradicionales como el teclado, el ratón o incluso la tableta gráfica también se encuentran en fase de retroceso a favor de teclados virtuales en pantallas táctiles o el reconocimiento del lenguaje natural para expresar las órdenes que debe ejecutar la máquina.

Sin embargo, el súmmum del control sobre el ordenador viene dado de la mano de los dispositivos EEG. Un dispositivo ElectroEncefaloGráfico suele estar formado por un casco que tiene distribuidos estratégicamente en su interior diversos sensores para detectar la actividad mental existente en diferentes zonas del cerebro. Pensar en algo agradable o desagradable, emocionarse, aburrirse, tener miedo o estar contento son estados mentales que activan diferentes zonas del cerebro y que, por tanto, pueden diferenciarse a través de la actividad recogida por estos sensores.

El presente volumen permite al lector conocer y sacar partido de uno de los primeros dispositivos EEG comerciales, el casco Emotiv EPOC. Este casco se suministra con una API que permite convertir los pensamientos, emociones y expresiones faciales en cualesquiera órdenes para el ordenador tales como pulsaciones de teclas. Simultáneamente, presentamos un proyecto de diseño electrónico de fácil ejecución con el que controlar un vehículo teledirigido a través del teclado del ordenador usando para ello la interfaz paralela. Con la conexión de la mente al PC a través del casco Emotiv EPOC, y del PC al vehículo inalámbrico previo procesamiento por parte del software desarrollado, conseguimos superar los límites que la Naturaleza ha impuesto al Hombre, siendo capaces de inducir movimiento en un vehículo a través de la mente. Esto es Telekinesia.

ISBN: 978-84-694-3343-0

UNIVERSIDAD DE MÁLAGA

BY NC ND