UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

PROYECTO FIN DE CARRERA

Desarrollo de una interfaz gráfica de usuario en entorno MATLAB para el manejo de software de segmentación de imágenes RMN

Realizado por:
Javier Cancio del Busto

Dirigido por:
José Vicente Manjón Herrera

Valencia, Septiembre de 2010

"Investigar es ver lo que
todo el mundo ha visto,
y pensar lo que
nadie más ha pensado."

Albert Szent-Györgyi
(Filósofo húngaro, 1893-1986)

Índice de Contenidos

1. Introducción

1.1 Prólogo

El inicio del siglo XIX vino marcado por el gran avance que supuso la "Revolución Industrial". De forma análoga, durante el siglo XX se produce otro proceso revolucionario, gracias al avance de la ciencia, y concretamente de la "Ciencia de la computación", a la que aún le queda un largo camino. Hoy en día, nos resulta prácticamente imposible pensar en la vida sin la ayuda que nos proporciona la informática en casi todos los ámbitos: económicos, sociales y tecnológicos.

Uno de los campos afectados por este avance es el campo de la salud. Gracias al progreso tecnológico ha sido posible juntar disciplinas aparentemente divergentes como el de la informática y la medicina, y gracias a esta convergencia ha sido posible realizar diagnósticos con una velocidad mucho mayor, explorar las causas de un gran número de enfermedades y aumentar la eficacia de los tratamientos.

El presente texto se centra en la integración de diferentes métodos tecnológicos, desarrollados para poder tratar con una mayor eficiencia las enfermedades que derivan en un cambio en el volumen de ciertos tejidos cerebrales, como es el caso del Alzheimer. El Alzheimer es una enfermedad neurodegenerativa que se muestra como un deterioro cognitivo y trastornos conductuales. Se caracteriza típicamente por una pérdida progresiva de memoria y otras capacidades mentales, a medida que las neuronas mueren y se atrofian diferentes zonas del cerebro. Es precisamente la muerte de las células nerviosas lo que causa pérdida de tejido, especialmente en la región del hipocampo, hasta en un 32%[1].

Es por estas enfermedades el interés de crear una herramienta gráfica que simplifique el uso de los métodos de segmentación en imágenes de resonancia magnética nuclear, junto con métodos previos, como el filtrado de imágenes y también poder procesar simultáneamente múltiples casos.

[1] Olivier Colliot, Gaël Chételat, Marie Chupin. "Discrimination between Alzheimer Disease, Mild cognitive Impairment, and Normal Aging by Using Automated Segmentation of the Hippocampus"

1.2 Situación y objetivos

El estudio actual se enmarca dentro del área de investigación del grupo ITACA (Instituto de Aplicaciones de las Tecnologías de la Información y de las Comunicaciones Avanzadas). El grupo ITACA cuenta con un total de cinco secciones de I+D+i entre las que se encuentra el grupo de informática biomédica (IBIME). Dentro de esta área existe un departamento que se ocupa del tratamiento informatizado de las imágenes médicas digitales facilitando las tareas de análisis, procesado, reconstrucción, segmentación, etc.

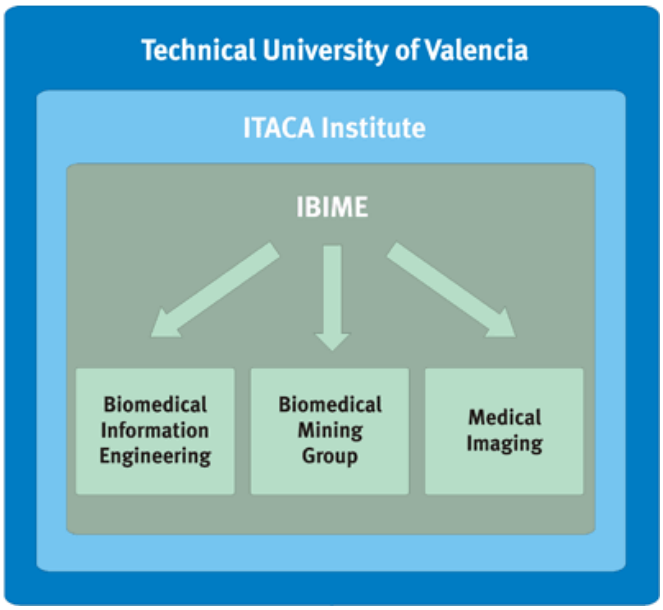


Figura 1.1. Jerarquía del grupo de investigación IBIME.

Las imágenes que se han utilizado en este proyecto corresponden al campo de la resonancia magnética nuclear (en adelante RM). Las continuas mejoras que se han producido en este tipo de imágenes durante los últimos años, han propiciado la ejecución de numerosos ensayos debido a la mayor resolución y nivel de precisión que aportan a los equipos. Dentro del abanico de posibilidades que ofrece la tecnología actual, la herramienta desarrollada trabajará con imágenes RM pertenecientes a una zona anatómica muy concreta: el encéfalo. El encéfalo se encuentra situado en el interior del cráneo y está constituido por el cerebro, el cerebelo y el tronco cerebral.

El objetivo del presente proyecto es el desarrollo de una interfaz gráfica de usuario (GUI) que sirva de herramienta a los operarios de las imágenes RM para poder realizar las tareas de segmentación de forma más fácil y cómoda. La herramienta no sólo tiene como finalidad la segmentación de las

imágenes, sino también poder aplicar otros tratamientos a la imagen, como filtrado de ruido, extracción del cerebro, corrección de homogeneidad y poder editar la máscara de cada caso para corregir errores. Así mismo, tendrá como finalidad la posibilidad de procesar múltiples casos de manera secuencial, con lo que el operario podrá dejar los casos en ejecución y obtener en un único fichero los datos derivados de la segmentación de cada uno de ellos.

Gracias a los métodos de segmentación desarrollados por el grupo de imagen médica, podemos obtener como resultado el volumen (en mililitros) de partes del encéfalo: materia gris, materia blanca y líquido cefalorraquídeo. Esto permitirá el diagnóstico de ciertas enfermedades, como el Alzheimer antes citado.

1.3 Estructura del documento

Esta memoria está estructurada de forma que los conceptos aparecen de forma progresiva, de manera que en un determinado punto se describen nociones o ideas que subyacen al contenido de la sección siguiente. Por este motivo, se ha intentado respetar en todo momento la jerarquía de los contenidos, tratando aspectos más generales al principio para profundizar después en matices más específicos.

Con todo esto, la redacción del proyecto se estructura en dos partes bien diferenciadas. La primera parte, describe los principios y fundamentos más elementales de la resonancia magnética nuclear. Este punto responde a una mera recopilación de información que resultará útil para entender las bases físicas y metodológicas de la imagen por resonancia magnética. La inclusión de este apartado permite la adquisición de los conocimientos necesarios sin necesidad de consultar demasiadas fuentes externas de información.

Por otro lado, en la segunda parte del documento se introduce de lleno en la descripción de los materiales utilizados para poder desarrollar la herramienta.

Los siguientes puntos del texto se centran en exponer y realizar una interpretación de los resultados obtenidos, realizando una visión generalizada sobre el funcionamiento y la utilidad real de la herramienta.

Por último, en el anexo se incluye el código fuente de la herramienta desarrollada, incluyendo únicamente lo referente a la interfaz, dejando de lado el código fuente de los métodos desarrollados por el grupo de imagen médica.

2. Conceptos básicos de Resonancia Magnética Nuclear

2.1 Origen y aplicaciones

La resonancia magnética nuclear (en adelante RMN) fue descrita y medida en rayos moleculares por Isidor Rabi en 1938, por lo que obtuvo el Premio Nobel de física en 1944. Posteriormente, en 1946, Félix Bloch y Edward Mills Purcell refinan la técnica para ser utilizada en líquidos y sólidos, por lo que compartieron el Premio Nobel de física en 1952.

La RM hace uso de las propiedades de resonancia aplicando radiofrecuencias a los átomos o dipolos entre los campos alineados de la muestra, y permite estudiar la información estructural o química de una muestra. La RMN se utiliza en el campo de investigación de ordenadores cuánticos[2], encontrándose sus aplicaciones más frecuentes en los campos de medicina, bioquímica y química orgánica. Es el campo de la medicina, y concretamente el de imágenes médicas en el que se centrará este proyecto.

En la década de los años 70 se obtuvieron las primeras imágenes de RMN, concretamente de un dedo, obteniéndose en 1977 la primera imagen cerebral, y en 1978 la del abdomen. Actualmente, la RMN constituye unas de las herramientas más importantes para el diagnóstico médico por imagen. Alguna de las ventajas que supone este método como técnica de imagen médica son:

- Puede penetrar huesos y estructuras huecas con atenuación despreciable.
- Es una técnica no ionizante, por lo que es segura.
- Permite obtener buenos contrastes en las imágenes, por lo que se puede diferenciar correctamente los tejidos.
- Facilita la obtención de imágenes en cualquier plano del espacio.

[2] http://planetphysics.org/encyclopedia/QuantumComputers.html

2.2 Principios físicos de la RMN

La materia está constituida por átomos, y estos a su vez tienen ciertas partículas elementales que componen su núcleo atómico, como el caso de los protones (carga positiva) y los neutrones (carga neutra), dejando los electrones (carga negativa) en la corteza. Estas partículas tienen la propiedad mecánico-cuántica del espín, que se modelizar como un giro de la partícula alrededor del propio eje.

El espín, como toda carga en movimiento o corrientes eléctricas, presentan propiedades magnéticas. El electrón, el protón y el neutrón tienen un momento magnético asociado $\vec{\mu}$ representado por un vector con dirección la del eje de giro y sentido en función del sentido de giro de la partícula. La orientación del vector $\vec{\mu}$ es de vital importancia en todo el proceso que conlleva el fenómeno de la RMN.

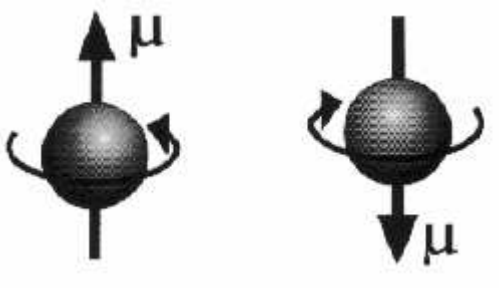


Figura 2.1. Espín de una partícula y momento magnético asociado $\vec{\mu}$.

El espín de un núcleo está determinado por el número cuántico del espín S. Si el número combinado de protones y neutrones en un isótopo dado es par, los espines tienen sentidos opuestos y sus efectos magnéticos se anulan. En este caso tenemos que S=0. Por tanto, la RMN sólo es útil en núcleos atómicos que tengan un número impar de neutrones o protones, en cuyo caso tenemos que S está asociado a un momento magnético distinto de cero ($\vec{\mu}$) que viene dado por:

$$\vec{\mu} = \gamma S$$

Ecuación 2.1. Ecuación del momento magnético asociado.

Donde γ es la constante de proporción giromagnética e indica la intensidad de la señal de cada isótopo utilizado en la RMN.

El núcleo del átomo de hidrógeno (H^+) tiene sólo un protón y es un elemento muy abundante en el cuerpo humano, por lo que es idóneo para el uso de la RMN en la obtención de imágenes médicas.

En ausencia del campo magnético, los espines de los protones están orientados al azar y el momento magnético resultante de sumar todos los $\vec{\mu}$ en un elemento de volumen es cero.

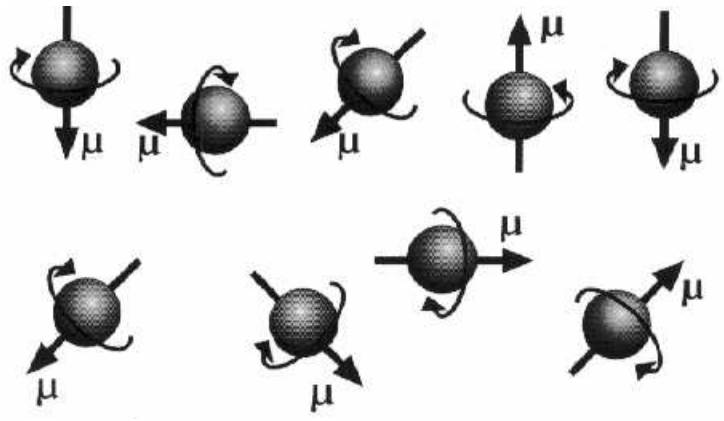


Figura 2.2. Momentos magnéticos de los núcleos de H^+ orientados al azar en ausencia de campo magnético.

Cuando sobre los protones actúa un campo magnético $\vec{B}_0$ éstos tienden a orientarse de tal manera que su $\vec{\mu}$ sea paralelo a $\vec{B}_0$. Dado que llevan incorporado un movimiento de rotación, este se combina con el movimiento producido por la fuerza magnética que ejerce $\vec{B}_0$ sobre el protón dando como resultado un *movimiento giroscópico* consistente en que al mismo tiempo que el protón gira alrededor de su eje, el eje gira alrededor de la dirección del $\vec{B}_0$.

Cuando los protones se alinean en la misma dirección que $\vec{B}_0$, pueden adoptar dos orientaciones diferentes:

- Orientación paralela o *up*. Corresponde a un estado energético bajo y es en el cual $\vec{\mu}$ apunta en el mismo sentido que $\vec{B}_0$.

- Orientación antiparalela o *down*. Corresponde a un estado energético alto y es en el cual $\vec{\mu}$ apunta en sentido opuesto a $\vec{B}_0$.

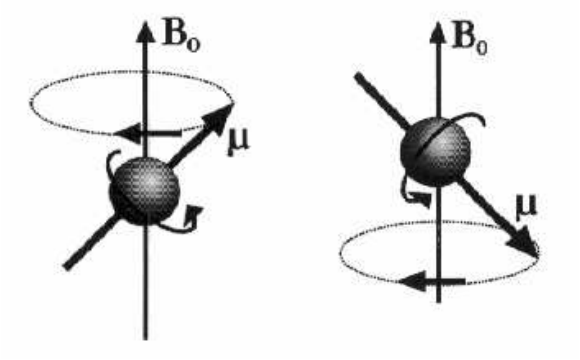


Figura 2.3. Posibles orientaciones: paralela (izquierda) o antiparalela (derecha).

2.3 Imágenes médicas por RMN

Como hemos comentado antes, la RM es una técnica de diagnóstico por imagen. Esta definición la engloba dentro de un conjunto de técnicas médicas cuyo objetivo es facilitar un diagnóstico mediante una "visualización" del interior del cuerpo. Dentro de estas técnicas se encuentran la ecografía, la tomografía axial computerizada (TAC, popularmente conocida como escáner) y los archifamosos rayos X (RX), y todas ellas forman imágenes del interior de nuestro cuerpo.

Aunque las imágenes finales puedan resultar muy similares, en realidad son muy distintas tanto en lo que muestran como en el método seguido para obtenerlas. Las que presentan mayores similitudes son los RX y TAC que se basan en el coeficiente de atenuación de los rayos X. Al bombardear un cuerpo con fotones de alta intensidad, estos son capaces de atravesar la materia "blanda" o poco densa (pulmones, intestinos, músculos…) quedando retenidos por la materia "dura" o densa (fundamentalmente los huesos). Los fotones que consiguen atravesar nuestro cuerpo impresionan una película fotográfica, creando una imagen que en realidad no es más que una sombra ya que donde se «ve» un hueso es en realidad una zona que no pudo ser alcanzada por los fotones.

El fundamento del TAC es similar pero mucho más avanzado ya que potentes ordenadores son capaces de procesar las imágenes para reconstruirlas formando cortes axiales. La superposición de múltiples cortes consecutivos puede formar imágenes en 3D.

Los RX y su evolución el TAC tienen gran importancia en el diagnóstico médico. Sin embargo presentan dos grandes inconvenientes:

- La aplicación continuada de RX es peligrosa para la salud. De ahí las medidas de seguridad que adoptan los profesionales.

- Con esta técnica las partes blandas no forman imágenes claras. Quedaban así sin poder verse órganos como tendones, ligamentos o el cerebro por citar tres ejemplos.

Es por eso tan importante el uso de la RMN, ya que es un método por imagen no radiológico (no necesita proyectar ningún tipo de radiación contra el objeto de estudio) por lo que resulta inocua.

La imagen por resonancia magnética (en adelante IRM) tiene ventajas significativas sobre el resto de métodos empleados en la actualidad para la obtención de imágenes médicas:

- No utiliza radiaciones ionizantes. La imagen se obtiene mediante campos magnéticos y radiofrecuencia, con lo que se evitan los pequeños riesgos que acompañan a las dosis bajas de radiación administradas en TAC y RX convencionales.

- Mejor resolución de bajo contraste. La IRM se basa en tres parámetros independientes (T1, T2 y SD). Estos varían considerablemente (20-40%) entre tejidos diferentes, mientras que μ sólo varía un 1%. Estas diferencias son las responsables de su excelente resolución de bajo contraste que constituye su principal ventaja.

- Imagen multiplanar. Se pueden obtener planos trasversales, coronarios, sagitales y oblicuos. Se pueden obtener imágenes volumétricas.

- Las medidas de flujo son directas. Se puede visualizar y cuantificar directamente el flujo de sangre.

- No resulta invasiva. Dado su excelente resolución de bajo contraste no es necesario utilizar medios de contraste.

Pasos para obtener la IMR

Para obtener la imagen básicamente cuatro pasos que se resumen de manera elemental como:

- Se coloca al paciente dentro de un campo magnético.

- Se le envía una onda de radio.

- Se interrumpe la onda de radio.

- El paciente emite una señal que es recibida y utilizada para reconstruir la imagen

¿Por qué se coloca al paciente dentro de un campo magnético?

Las partículas subatómicas, como hemos comentario en el punto anterior, tienen propiedades inherentes. Protones y electrones tienen carga mientras que los neutrones no. Del mismo modo que la carga de un átomo es la suma de todas sus cargas, el espín total será la suma de los espines de todas sus partículas subatómicas.

Imaginemos el átomo más sencillo: un protón. La carga del protón realiza un movimiento de espín con él ¿y qué es una carga eléctrica en movimiento? Es una corriente eléctrica. Y las cargas eléctricas producen campos magnéticos. Así pues, ese protón (y como él cualquier núcleo) cargado y girando produce a su alrededor un campo magnético.

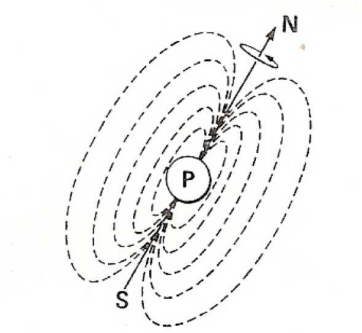


Figura 2.4. Campo magnético creado por un protón.

En la mayor parte de los materiales, los momentos magnéticos están orientados al azar (figura 2.2), anulándose unos a otros y dando como resultado un momento magnético total de cero. Pero, ¿qué ocurre cuando colocamos los átomos bajo un intenso campo magnético externo? Los momentos magnéticos se alinean en la dirección del campo magnético como pequeñas brújulas.

Cierto que se alinean en la misma dirección pero pueden estar en sentidos contrarios. Aunque, en realidad, hay más protones que se alinean en el sentido de las líneas de campo magnético externo porque este estado tiene menos energía (de la misma manera que nosotros podemos alinearnos con las líneas del campo gravitatorio andando sobre nuestras manos o nuestros pies, este último estado es más estable energéticamente). Queda así un remanente de protones que no anulan sus momentos magnéticos con sus opuestos. En estas condiciones el paciente se transforma en un imán.

Pero los protones no giran con un movimiento limpio alrededor de un eje perfectamente alineado con el campo magnético externo, sino que al igual que una peonza sometida a un campo gravitatorio describen un giro de forma cónica. Los protones sometidos a un campo magnético describen el mismo giro, llamado *precesión*. La rapidez con la que «precesa» un núcleo se llama frecuencia de precesión y depende exclusivamente de la fuerza del campo magnético externo y del tipo de núcleo implicado. Esta relación queda definida por la ecuación de *Larmor*:

$$W = \gamma \vec{B}_0$$

Ecuación 2.2. Ecuación de Larmor.

En esta ecuación, W representa la frecuencia de *precesión* (en Hz), $\vec{B}_0$ la fuerza del campo magnético (en teslas) y γ la constante giromagnética (utilizada en la ecuación 2.1).

Estas propiedades del campo magnético pueden utilizarse para analizar muestras. Supongamos que colocamos una muestra de material desconocido en un campo magnético de valor conocido: Si obtenemos un valor de la frecuencia de precesión podemos calcular la constante giromagnética, dado que es específica para cada material, sabremos de qué material se trata. La fuerza de la señal nos indicará la cantidad de material de ese tipo que hay en la muestra.

Sin embargo, los núcleos precesan al azar, es decir, en un instante dado, existen núcleos orientados en cualquier dirección dentro de su giro de precesión. Fuerzas magnéticas orientadas en direcciones opuestas se cancelan unas a otras. Esto es así para todas las direcciones excepto para el eje Z (a lo largo del campo magnético externo). En la dirección Z los vectores suman sus fuerzas magnéticas dando como resultado final un vector magnético en la misma dirección que el campo externo.

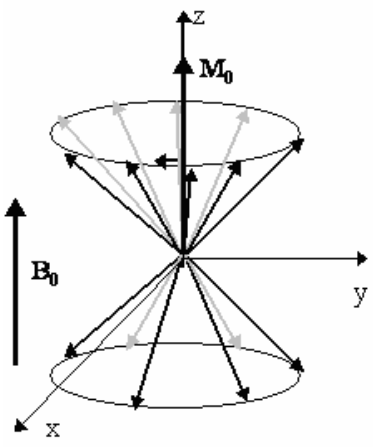


Figura 2.5. Momento magnético resultante de todos los $\vec{\mu}$.

En estas condiciones el paciente se transforma en un imán y este vector que emite se llama *magnetización longitudinal*. Es este nuevo vector el que puede ser utilizado para obtener la señal, el problema es que mientras esté paralelo a las líneas del campo magnético externo no podemos cuantificarlo. Parece que entonces no hemos conseguido nada, pero continuemos con el segundo paso: ondas de radio.

Para campos magnéticos de intensidad razonablemente elevada, la frecuencia de precesión de casi todos los núcleos de interés queda dentro de la banda de radiofrecuencia (RF). Si mandamos a un paciente «imantado» un pulso de radiofrecuencia (PRF) igual a la frecuencia de precesión del hidrógeno bajo ese campo magnético se producirá un fenómeno conocido como resonancia. Se puede ilustrar con el uso de diapasones; si usted entra en una habitación con diapasones afinados en do, re, mi y toca un diapasón de frecuencia do, hará vibrar a todos los diapasones que estén en do y sólo a esos. Pues bien, los H^+ absorberán energía de esa radiofrecuencia y se pasarán a su estado de alta energía, es decir se colocarán en sentido opuesto al campo magnético externo, reduciéndose la magnetización longitudinal (recordamos que los momentos en sentidos opuestos se anulan). Pero, no sólo ocurre eso, sino que además los H^+ comenzarán a precesar en fase, por lo tanto ahora sus momentos magnéticos en las direcciones X e Y se suman en vez de anularse. Aparece entonces un vector de magnetización transversal.

Este nuevo vector de magnetización transversal, al ser la suma de los momentos de cada protón, precesa de la misma forma. Si miramos desde fuera se acerca nosotros para luego alejarse y después volver a acercarse. Este vector magnético en movimiento induce una corriente eléctrica que puede ser registrada por una antena. Cuanto mayor sea el vector, mejor será la imagen de RM obtenida. El valor del mismo depende de *y,* $\vec{B}_0$ y la densidad de espines.

Bien, ya tenemos una señal, pero ¿de qué parte del cuerpo procede?. El truco para acotar el vector de magnetización transversal a un plano estrecho del cuerpo consiste en no someter al paciente a un campo magnético homogéneo, ya que en ese caso todos los H^+ precesarían a la misma frecuencia y en consecuencia todos serían alterados por la misma RF. Si, en cambio, el paciente se somete a un gradiente de campo, cada tramo de su cuerpo tendrá los H^+ precesando a una frecuencia característica, dada por la ecuación de Larmor. Así, variando la RF obtenemos señales de distintos planos del cuerpo, tanto más finos cuanta mayor sea la graduación del $\vec{B}_0$.

El siguiente paso es cortar la radiofrecuencia, recordamos que mandamos pulsos de radiofrecuencia (PRF). Ahora los H^+ no tienen ninguna energía que les haga girar en fase y poco a poco vuelven a desfasarse, además de recuperar su estado de menor energía. Comienza a crecer la

magnetización longitudinal a medida que decrece la transversal. La velocidad con la que se producen estos fenómenos (T1 y T2) es característica para tejido. El ordenador es capaz de dar un código de grises para cada intensidad de señal, marcando así cada uno de los tejidos.

2.4 El concepto de vóxel

El vóxel (Volumetric Pixel o más correctamente Volumetric Picture Element) es una unidad cúbica que compone un objeto tridimensional. Constituye la unidad mínima procesable de una matriz tridimensional y es, por tanto, el equivalente al píxel en un objeto 2D.

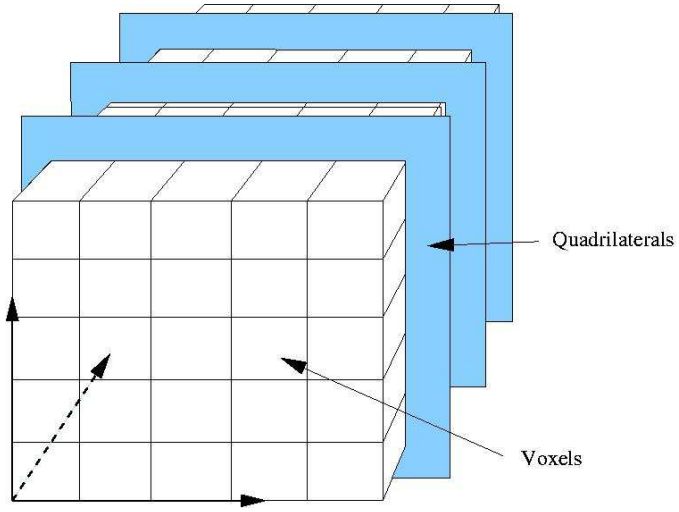


Figura 2.6. Representación de vóxeles en una matriz tridimensional.

Para crear una imagen en tres dimensiones, los vóxeles tienen que sufrir una transformación de opacidad. Esta información da diferentes valores de opacidad a cada vóxel. Esto es importante cuando se han de mostrar detalles interiores de una imagen que quedaría tapada por la capa exterior más opaca de los vóxeles.

Las imágenes con vóxeles se usan generalmente en el campo de la medicina y se aplican, por ejemplo, en la tomografía axial computarizada o para las resonancias magnéticas. De este modo, los profesionales pueden obtener un modelo preciso en tres dimensiones del cuerpo humano.

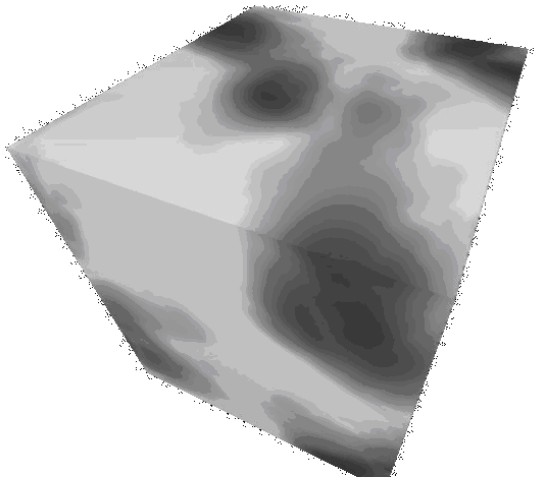


Figura 2.7. Vóxel en el que se pueden diferenciar distintos tejidos.

Actualmente, su uso ya se ha extendido en multitud de campos como la medicina, ingeniería, cine, videojuegos...

Al igual que los píxeles, los vóxeles no contienen su posición (x,y,z) en el espacio 3D, sino que esta se deduce por la posición del vóxel dentro del archivo de datos, y tienen un tamaño concreto.

3. Material y métodos utilizados

3.1 Imágenes médicas

El desarrollo de la aplicación está enfocada al análisis y manipulación de imágenes médicas correspondientes al encéfalo. Para la prueba del software se han utilizado imágenes anónimas de pacientes de varias edades, con orientación mayormente axial.

A la hora de tomar las imágenes, no existe un convenio sobre qué orientación deben tener, por ello el software desarrollado permite reorientar las imágenes a una orientación axial una vez cargadas, previa selección del usuario sobre qué orientación tenía inicialmente.

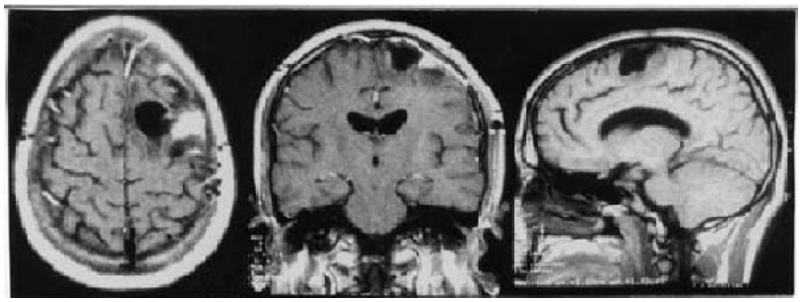


Figura 3.1. Corte axial (izquierda). Corte coronal (centro). Corte sagital (derecha).

Es necesario realizar una reorientación en el caso de que no sean axiales para poder aplicar los métodos desarrollados por el grupo de imagen médica. Las imágenes utilizadas tendrán formato *.img* y *.nii*.

3.2 Matlab

MATLAB (MATrix LABoratory) es un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (scripts y ficheros .mex escritos en lenguaje C).

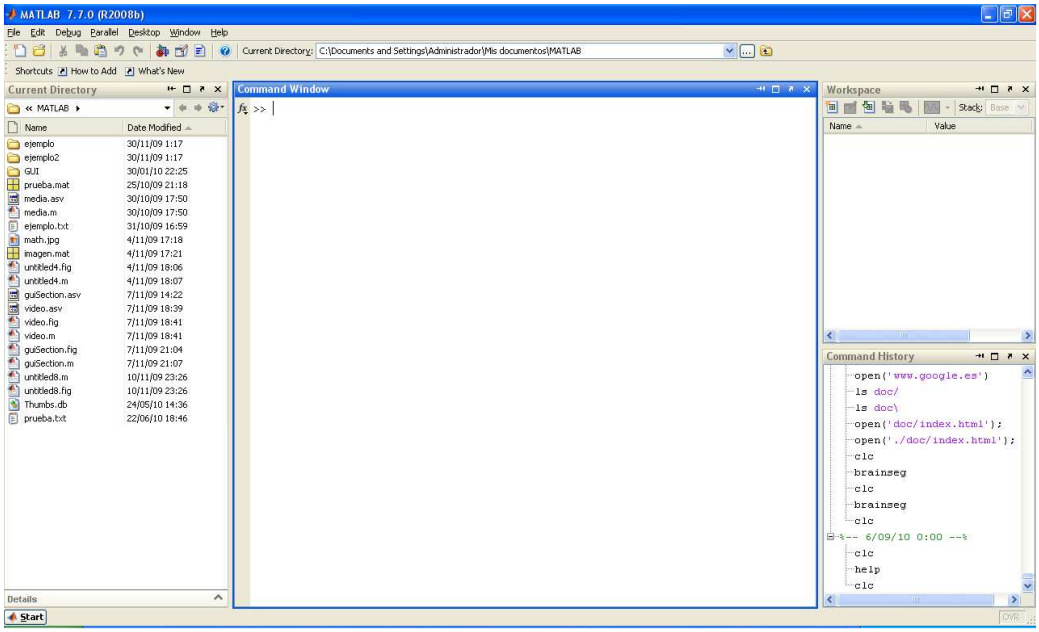


Figura 3.2. Pantalla principal de MATLAB.

Entre sus prestaciones básicas se hallan:

- Manipulación de matrices.
- Representación de datos y funciones.
- Implementación de algoritmos
- Creación de interfaces de usuario
- Comunicación con programas en otros lenguajes y con otros dispositivos hardware.

El paquete MATLAB dispone de una herramienta adicional que expande sus prestaciones, y que nos va a resultar de gran ayuda a la hora de crear interfaces de usuario: GUIDE.

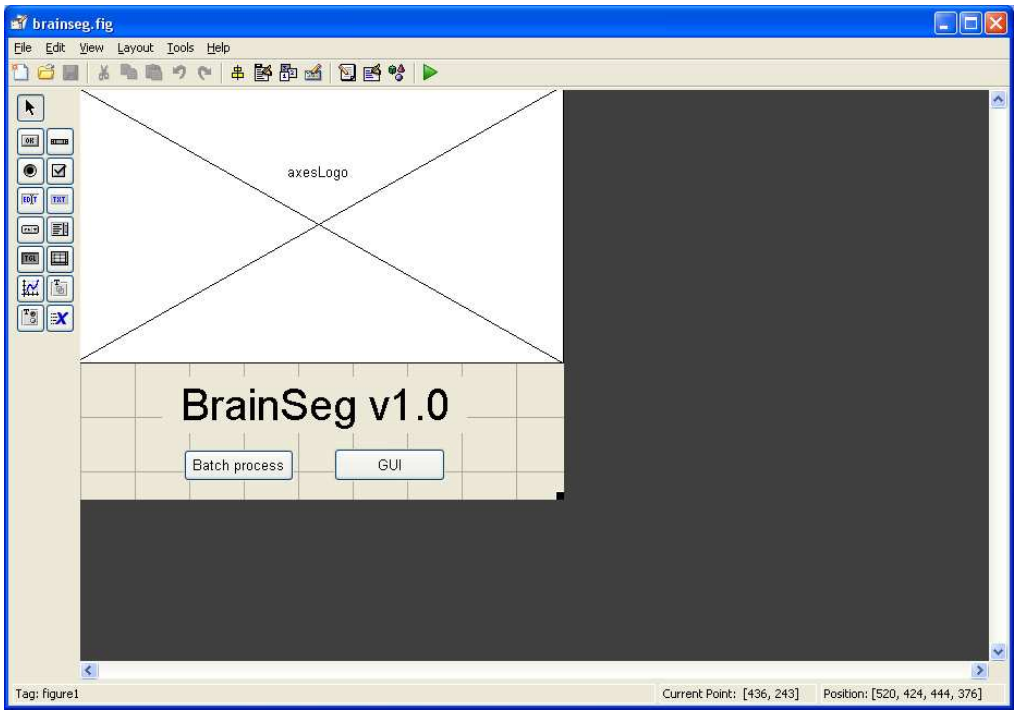


Figura 3.3. Pantalla de la herramienta GUIDE.

El principal motivo por el cual se ha elegido MATLAB como herramienta es precisamente su capacidad y eficacia para el tratamiento de matrices. Las imágenes médicas están almacenadas internamente en forma matricial, y es por lo tanto una herramienta óptima. Otra gran cualidad es la posibilidad de crear scripts, y son precisamente estos los que hemos utilizado a la hora de desarrollar el software. Así mismo, los módulos desarrollados por el grupo de imagen médica están escritos en matlab (los

scripts de matlab tienen extensión .m) junto con, en algunos casos, en lenguaje C (ficheros .mex).

3.3 SPM

Los recientes avances en técnicas de neuroimagen han contribuido a la proliferación de investigaciones y estudios relacionados con este campo. Los más vanguardistas utilizan una herramienta conocida como *Statistical Parametric Mapping* (SPM). SPM se ha convertido en el estándar de facto para la aplicación de métodos estadísticos en el análisis de imágenes. La herramienta se compone de una colección de funciones y una interfaz gráfica que se ejecuta directamente sobre MATLAB.

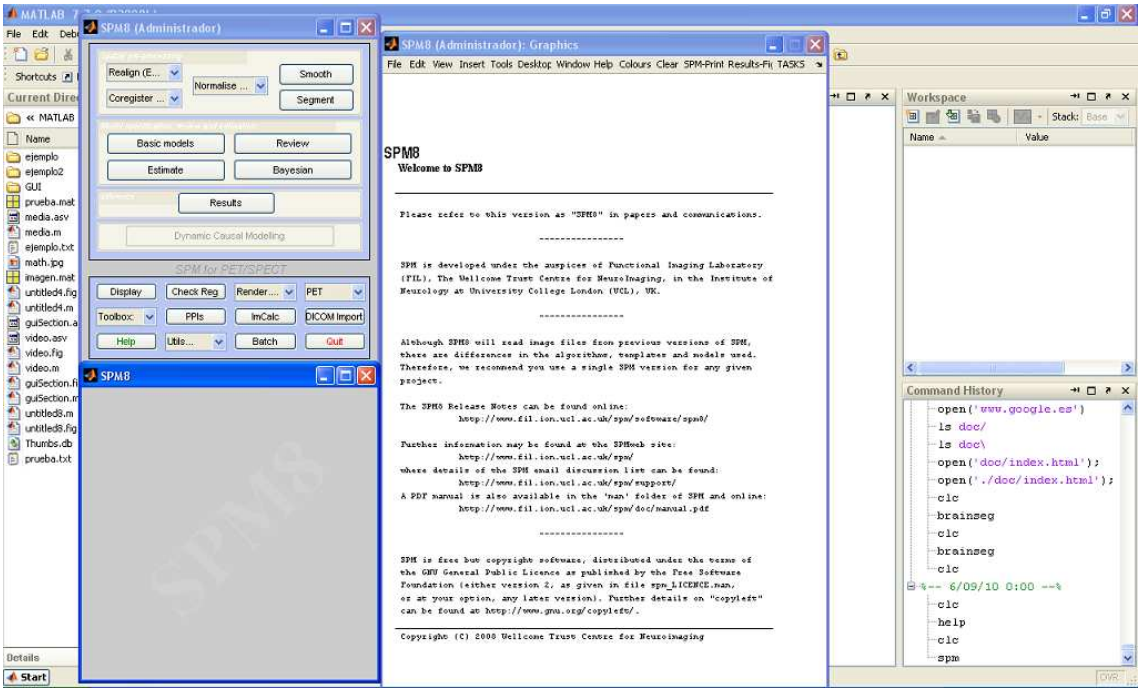


Figura 3.4. Interfaz gráfica de SPM8.

En concreto, la versión que se ha utilizado ha sido SPM8. SPM8 soporta el tratamiento de imágenes de RM, PET y SPECT. Una de las principales ventajas que tiene este software deriva de su carácter open source, ya que son muchos los investigadores (entre los que se incluyen informáticos, médicos, matemáticos, estadísticos, farmacéuticos y físicos) que a nivel mundial trabajan con este sistema. La aportación de cada uno de estos usuarios conlleva un desarrollo constante de la aplicación e incrementa su robustez frente a posibles errores o bugs. Además, SPM8 posee una buena granularidad de funciones que hace posible la creación de

procesos automatizados encapsulados en procesos por lotes (también conocidos como procesos batch o scripts). Es precisamente esta herramienta la que vamos a utilizar para poder leer los ficheros de imagen y obtener la matriz correspondiente para poder aplicar los tratamientos oportunos. Aunque los módulos desarrollados por el grupo de imagen médica hacen uso de más llamadas, para cargar los casos y sacar la información de la matriz utilizaremos principalmente vamos a utilizar estas dos llamadas:

spm_vol

```
V = spm_vol(P)
      P - a matrix of filenames.
      V - a vector of structures containing image volume information.

The elements of the structures are:
    V.fname   - the filename of the image.

    V.dim     - the x, y and z dimensions of the volume

    V.dt      - A 1x2 array.  First element is datatype (see spm_type).
                The second is 1 or 0 depending on the endian-ness.

    V.mat     - a 4x4 affine transformation matrix mapping from voxel coordinates to real
                world coordinates.

    V.pinfo   - plane info for each plane of the volume.
          V.pinfo(1,:)    - scale for each plane
          V.pinfo(2,:)    - offset for each plane
            The true voxel intensities of the jth image are given
             by: val*V.pinfo(1,j) + V.pinfo(2,j)
          V.pinfo(3,:)    - offset into image (in bytes).
            If the size of pinfo is 3x1, then the volume is assumed
            to be contiguous and each plane has the same scalefactor
            and offset.
```

spm_read_vols

```
Y = spm_read_vols(V)
 V   - vector of mapped image volumes to read in (from spm_vol)
 Y   - 4D matrix of image data, fourth dimension indexes images
```

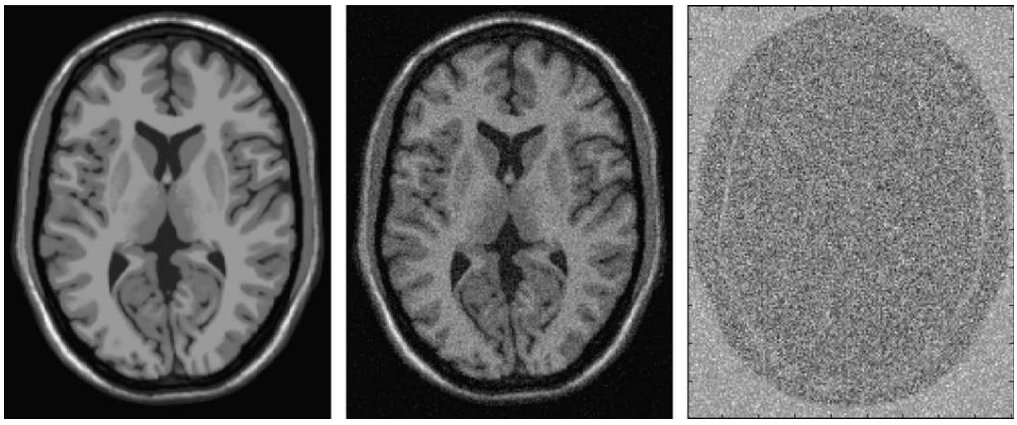
Básicamente estas dos llamadas nos van a servir para cargar una imagen a partir de su nombre de fichero (*spm_vol*) y posteriormente utilizar esta variable para cargar los datos propios del volumen (*spm_read_vols*). Con la llamada *spm_vol* además de cargar la imagen, nos devolverá información como el nombre de la imagen, las dimensiones del volumen y el tamaño de los vóxeles.

3.4 Módulos

El objetivo final de este proyecto es el desarrollo de una interfaz gráfica de usuario en entorno MATLAB para la integración de los módulos desarrollados por el grupo de imagen médica. En este punto se hará una breve descripción de cada uno de los módulos, dejando una descripción más detallada en los artículos citados en la bibliografía.

Filtrado de ruido

En el proceso de adquisición de imágenes es prácticamente imposible que se obtengan sin ningún tipo de ruido. El ruido en las adquisiciones puede deberse a varios factores que escapan al factor humano. Algunas de estas causas pueden ser la inhomogeneidades del campo magnético, radiaciones de fondo y movimientos del propio paciente dentro de la máquina.



(a) (b) (c)

Figura 3.5. Imagen sin ruido(a), imagen con un ruido estimado del 9%(b) y los residuos correspondientes (c).

Es necesario aplicar un buen filtrado a las imágenes, ya que la clasificación del tejido se hace en base al brillo de este, y el ruido puede causar una confusión a la hora de clasificar y por ello producir resultados indeseados. El módulo de filtrado puede realizar el filtrado propiamente dicho o solamente una estimación del ruido, realizándose esta como paso previo para realizar el filtrado, guardando además del fichero con el volumen sin ruido, otro fichero con los residuos.

Corrección de inhomogeneidad

Las imágenes médicas obtenidas por resonancia magnética suelen estar afectadas por inhomogeneidades en la intensidad de estas. Estas inhomogeneidades hacen mucho más complicado el proceso de obtener medidas cuantitativas. El método desarrollado por el grupo de imagen médica y el que se ha utilizado para el programa es un método que realiza esta corrección automáticamente, sin necesidad de supervisión y recibiendo como único argumento el nombre del fichero a tratar.

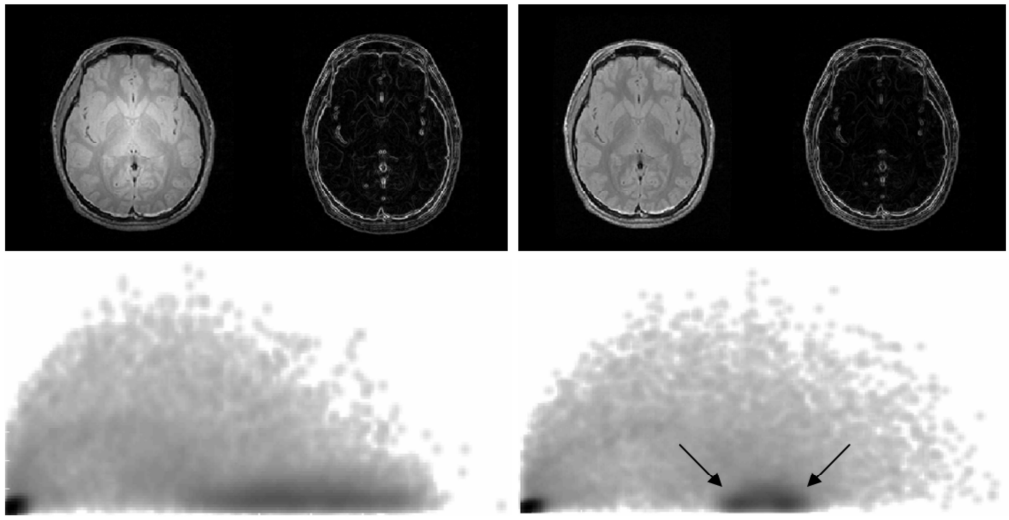


Figura 3.6. Resultados tras aplicar la corrección de inhomogeneidad a un volumen.

En los últimos años, la corrección de esta inhomogeneidad (normalmente referida como "bias correction") ha sido y es uno de los problemas que ha ganado mayor atención y para los cuales se han desarrollado numerosos métodos.

Extracción del parénquima

El paso previo a la segmentación es la extracción del cerebro. Este método se encarga de separar el parénquima del cráneo, ya que este último no es de nuestro interés para realizar la clasificación de tejidos. El método de extracción, además de separar el cerebro, genera una máscara que nos servirá para realizar la segmentación.

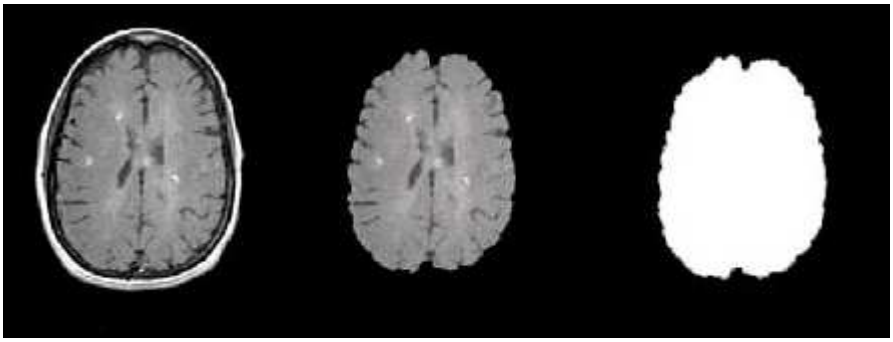


Figura 3.7. Volumen original (izquierda), parénquima extraído (centro), máscara correspondiente (derecha).

Como no existe el método perfecto, la máscara generada puede no ser del todo exacta, y por ello se ha implementado una opción en el programa desarrollado para corregir manualmente estas diferencias.

Clasificación del tejido

Es el último método y el paso final. Este método se encarga de la tarea más complicada y la más interesante a la vez. Básicamente se encarga, a partir del parénquima y de su máscara, de calcular el volumen que ocupa (en ml) de diferentes tejidos cerebrales, concretamente de materia blanca, materia gris y líquido cefalorraquídeo. Para que este cálculo sea lo más próximo posible a la realidad, los resultados de los métodos anteriores deben ser óptimos, siendo una parte fundamental la máscara generada en el punto anterior, y modificada en el caso de requerirlo.

Inicialmente, este módulo mostraba los resultados por pantalla, pero para la integración en el programa se ha optado por generar un fichero *.txt* con la información relativa (nombre de fichero, volumen del líquido cefalorraquídeo, volumen de materia gris, volumen de materia blanca, volumen total intracraneal, volumen normalizado del líquido cefalorraquídeo, volumen normalizado de materia gris, volumen normalizado de materia blanca y volumen cerebral normalizado).

Segmentación

El proceso de segmentación no es más que la ejecución secuencial de todos los anteriores, a partir de un fichero. Se ejecutan en orden los métodos de filtrado, corrección de inhomogeneidad, extracción del parénquima y clasificación de tejidos.

Habrá casos en los que el usuario únicamente quiera realizar únicamente el filtrado u otro método independiente del resto y habrá casos en los que le interese sea la clasificación de los tejidos.

Al igual que en el método de clasificación de tejidos, el método de segmentación generará un fichero *.txt* con la información antes descrita para cada volumen. Este método únicamente recibirá como argumento la ruta completa del fichero a segmentar.

4. Brainseg

El objetivo del presente proyecto es desarrollar una interfaz gráfica de usuario (GUI) que integre los métodos descritos en el punto anterior. Este era el objetivo principal de la aplicación. Para hacerla aún más completa, se han creado nuevas funciones, como la edición de máscara, que permiten un mejor resultado a la hora de realizar la segmentación de los volúmenes.

La aplicación se ha desarrollado bajo el entorno MATLAB, ya que como se ha comentado es el más extendido en el campo de imagen médica, y es una herramienta muy potente para trabajar con matrices. Con ayuda del editor visual GUIDE se ha creado el esqueleto de la interfaz, implementando las funcionalidades en ficheros script[3].

Según ha ido avanzando el desarrollo se han ido incluyendo nuevas características, y finalmente el programa se ha dividido en dos bloques principales, entre los que se puede elegir al lanzar la aplicación:

- GUI
- Batch process

GUI

Este bloque es el principal, en él podemos cargar un volumen, especificar qué orientación tiene este originalmente, y el programa lo reorienta siempre a orientación axial, tras lo cual lo muestra en pantalla.

Esta parte del programa incluye las opciones principales para aplicar los métodos antes descritos de forma individual o realizar la segmentación. Incluye además una herramienta de zoom con el que poder ver con más detalle una parte concreta del volumen, y la opción de mover el corte con el zoom aplicado. Así mismo también dispone de una herramienta de dibujado, en la cual una vez cargada una máscara, se puede activar para dibujar sobre las zonas que no han quedado bien definidas en el proceso de extracción del parénquima. Con esta herramienta activada, se podrá dibujar

[3] El código fuente se adjunta en el anexo del documento.

sobre el vóxel de la máscara haciendo click izquierdo, y borrar haciendo click derecho, con el radio elegido. También nos permite elegir la etiqueta que queremos escribir en la máscara, algo muy útil para los casos en los que la clasificación de tejidos va a ser de varios de ellos y queremos distinguir qué parte corresponde a cada tejido en la máscara.

Antes de poder aplicar todos estos cambios es necesario cargar una máscara, mediante un menú que permite cargar, descargar máscaras, y aplicar un porcentaje de opacidad, que varía del 0% (opaca) al 100% (transparente) en intervalos del 25%.

Una vez que se ha modificado la máscara, existe una opción en el apartado de edición que permite guardarla tras introducir el nombre del fichero nuevo. El apartado de configuración permite guardar el volumen con su orientación original o con la orientación reorientada a axial.

Los resultados obtenidos en los procesos que se hayan seleccionado se guardan dentro del directorio *results*. Dentro de este directorio se crea un subdirectorio con el nombre del fichero procesado, y dentro de este todos los resultados obtenidos de dicho fichero.

En la pantalla se muestra información del volumen como el nombre, el tamaño y la posición x, y, z correspondiente a la posición actual del puntero del ratón, así como el valor en ese punto.

Batch Process

Es el otro bloque general en que se ha dividido el programa. Este bloque se ha creado para poder procesar múltiples ficheros simultáneamente de forma automática, sin necesidad de intervención del usuario una vez cargados los ficheros y seleccionado los métodos que se van a ejecutar.

Se ha creado para poder cargar los múltiples casos a partir de un directorio o seleccionándolos individualmente. Además se ha incluido la opción de eliminar los ficheros intermedios generados por los métodos anteriores, así como la opción de guardar los resultados en el directorio *results* dentro del directorio del programa (como en el caso de GUI) o guardarlos en el directorio que el usuario especifique.

Al ejecutar los casos en este modo, se irán mostrando en la consola de MATLAB la evolución de cada método, informando cuando finalice cada uno y cuando termine la segmentación.

Como en el caso del método de clasificación de tejidos y segmentación, y al estar enfocado a procesar múltiples casos, cuando finalice cada volumen

se generará el informe correspondiente, figurando en cada línea el nombre del fichero para que sea más sencillo reconocer a qué paciente pertenece.

5. Resultados

El resultado del presente proyecto ha sido una interfaz intuitiva y de fácil uso, teniendo bien diferenciados los dos bloques principales al ejecutar el programa.

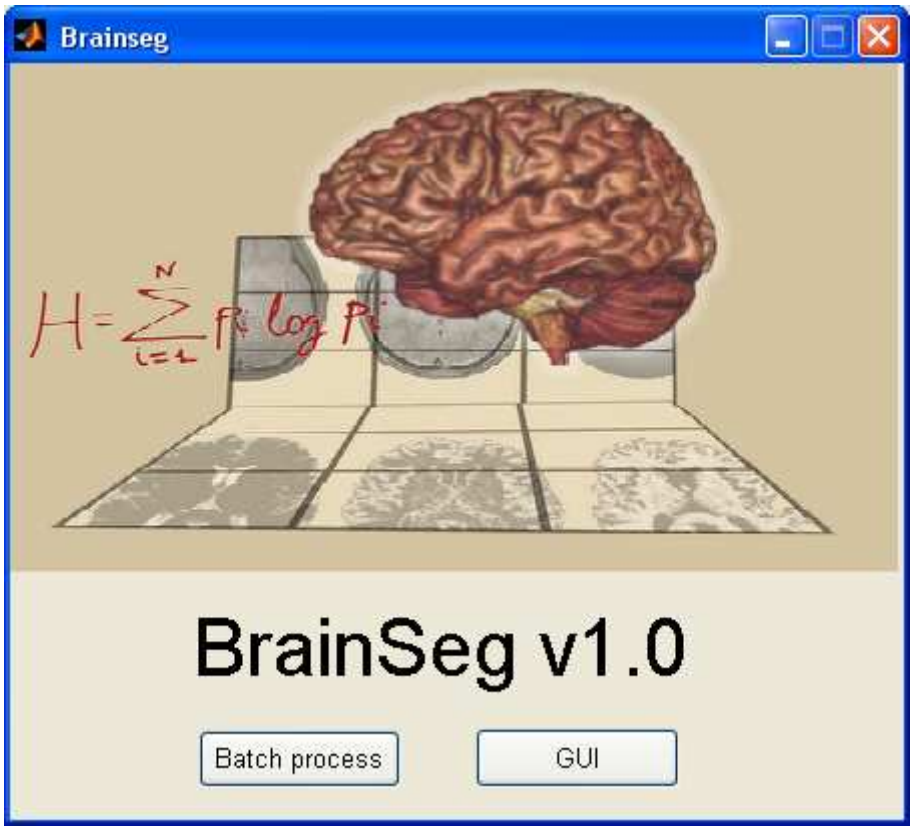


Figura 5.1. Pantalla principal de la aplicación.

La primera vez que se ejecuta la aplicación, preguntará por los directorios en los que tenemos instalados SPM y VBM[4] en el caso de no estar en el *path* de MATLAB.

[4] Colección de extensiones para el algoritmo de segmentación de SPM (versiones 2, 5 y 8).

Siguiendo lo explicado en el punto anterior, se tiene la pantalla de la opción de GUI en la que se nos mostrará el volumen junto con toda la información relativa al volumen y a la posición actual.

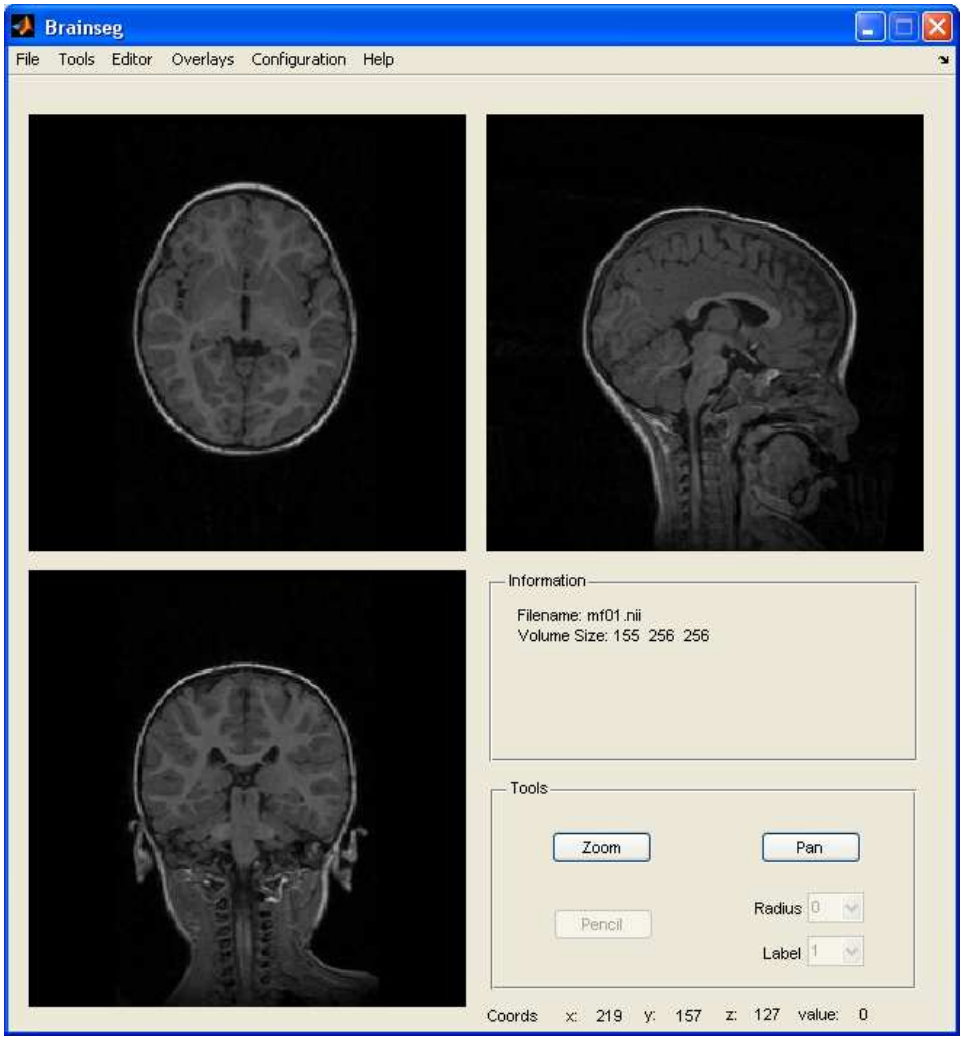


Figura 5.2. Pantalla de la opción GUI del programa.

En esta pantalla se mostrará el volumen cargado y reorientado a axial si es necesaria la reorientación. En la parte inferior se muestra la información de la posición actual, mientras que en el cuadro de *Information* se muestra el nombre del fichero y el tamaño del volumen.

En la parte superior están las distintas opciones, entre ellas está la de herramientas (Tools) en la que podemos elegir qué método se quiere aplicar al caso cargado, cargar una máscara (Overlays) y editar esta máscara para, como se ha comentado anteriormente, segmentar el cerebro (Editor).

En la parte inferior derecha está la zona de herramientas, en la que se puede aplicar el zoom, mover la imagen y, una vez activado el editor, corregir la máscara con una etiqueta y un tamaño de dibujado concreto.

Para cambiar el corte, con hacer click izquierdo en una vista del volumen se actualizarán las vistas de los otros dos cortes, de la misma forma, con la rueda del ratón se pueden cambiar los cortes de la vista actual.

La otra opción, *Batch Process*, muestra una interfaz muy simple e intuitiva:

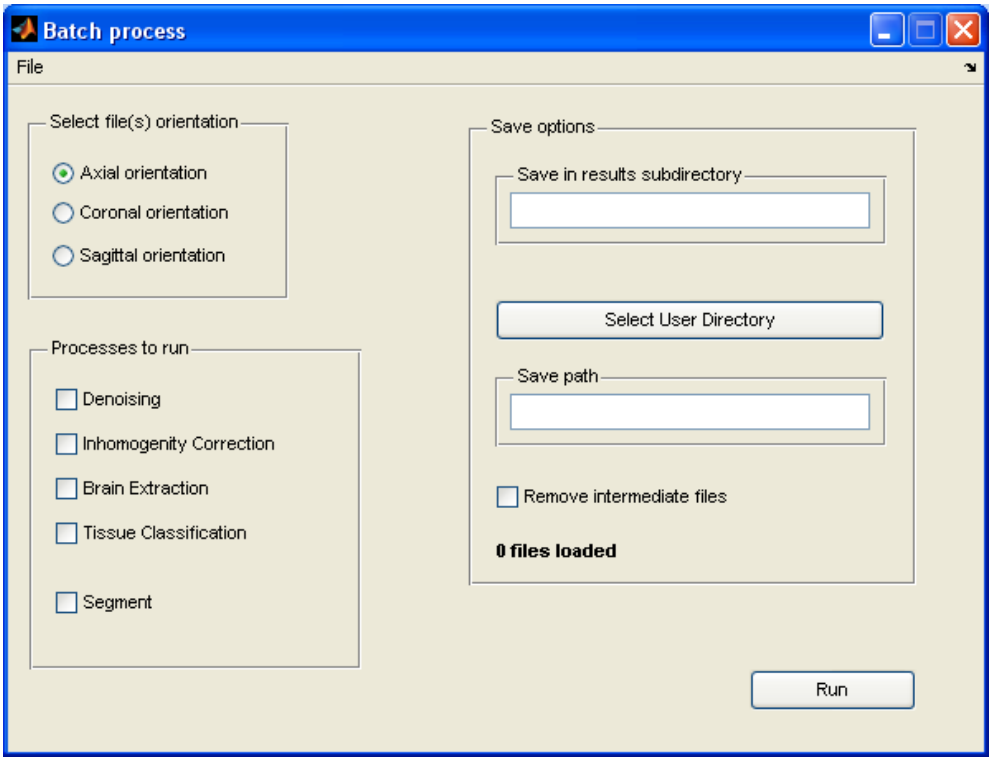


Figura 5.3. Pantalla de la opción Batch Process del programa.

En la pantalla de esta opción hay tres bloques; para especificar la orientación que tienen todos los volúmenes, seleccionar qué procesos queremos ejecutar y las opciones de guardado.

En estas últimas se dan dos posibles casos:

- Guardar dentro del subdirectorio results
- Seleccionar un directorio por el usuario

En el primer caso se guardarán los resultados (ficheros intermedios e información de la segmentación) dentro del subdirectorio *results* bajo otro subdirectorio que tendrá como nombre el que se introduzca en el campo de texto.

En el segundo caso los resultados se guardarán en la ubicación que especifique el usuario mediante un cuadro de diálogo.

6. Discusión y conclusiones

El proyecto comenzó a mediados de noviembre con la lectura de documentación sobre matlab y la programación de scripts para este, para posteriormente empezar a desarrollar una primera aproximación a la interfaz, que ha llevado unos ocho meses de desarrollo en etapas más o menos activas.

Como resultado de este proyecto se ha obtenido un software usable, en el que se ha hecho todo lo posible por automatizar todas las tareas de tratamiento de imágenes médicas, y hacerlo lo más sencillo posible.

Durante el desarrollo ha habido ciertas partes que han resultado más problemáticas, como la redimensión de los volúmenes independientemente de su tamaño original o las opciones de pincel y la iniciación a matlab.

Hasta ahora el uso de los métodos desarrollados por el grupo de imagen médica sólo se podían utilizar mediante la línea de comandos de matlab, por lo que se espera que esta herramienta resulte de gran utilidad para los operarios.

El software está abierto a futuras mejoras y actualizaciones en base a las propias mejoras en los métodos que integra.

Personalmente, creo que ha sido un proyecto bonito e interesante, en el que he aprendido muchas cosas sobre imagen médica y matlab, y que espero sirva de gran utilidad.

7. Agradecimientos

Esta es sin duda la parte más personal del documento, y en la que más nos pensamos qué decir y cómo decirlo, y a la vez quizás la parte más complicada.

Sin duda alguna, a las personas que más tengo que agradecer en la vida son a mi madre y a mi abuela. Siempre han estado ahí, desde pequeño, apoyándome y dándome consejos, educándome como mejor han creído y sacándome adelante siempre pese a las circunstancias. Si no fuese por ellas jamás podría haber estudiado lo que me gusta, ni estar donde estoy ahora, y ha sido por ellas por las que muchas veces he sacado fuerzas cuando pensaba que algo era demasiado complicado, algo que necesitaba esfuerzo y sacrificio. Han sabido enseñarme cómo deben ser las cosas y cómo no deben serlo, a luchar por un futuro y ser alguien. Les debo la vida.

A parte de mi familia, mi padre y mi tío, que también me han aconsejado, me han apoyado y han confiado en mí, y cada uno a su manera me han enseñado hacia dónde debía dirigir mi futuro.

A mi novia, Sylvia, que me ha apoyado siempre, ha sabido escucharme, me ha aconsejado y ha estado a mi lado en todo momento, siempre que lo he necesitado, y que es una parte fundamental en mi vida. Te quiero.

Y sin duda a mi director de proyecto, José, que ha hecho lo imposible por ayudarme siempre que me he atascado con algo, mostrando interés por que progresase. Gracias a él he aprendido muchas cosas con este proyecto, tanto en el ámbito informático como en el médico, y me ha dedicado un montón de horas de trabajo. Si todos los directores fuesen así, tendríamos un bonito recuerdo de nuestro último escalón al finalizar la carrera en lugar de un mero trámite para terminarla. Gracias José.

Y a mi abuelo, al que siempre llevaré en mi memoria.

8. Bibliografía y referencias

Artículos científicos

José V.Manjón, Juan J. Lull, José Carbonell-Caballero, Gracían García-Martí, Luis Martí-Bonmatí, Montserrat Robles. 2007. A nonparametric-MRI inhomogeneity correction method. Medical image analysis 11: 336-345.

Pierrick Coupé, Pierre Yger, Sylvain Prima, Pierre Hellier, Charles Kervrann, Christian Barillot. 2008. An optimized blockwise nonlocal means denoising filter for 3-D magnetic resonante images.

José V.Manjón, Juan J. Lull, José Carbonell-Caballero, Gracían García-Martí, Luis Martí-Bonmatí, Montserrat Robles. 2008. MRI denoising using non-local means. Medical image analysis 12: 514-523.

José V.Manjón, Jussi Tohka, Gracián García-Martí, José Carbonell-Caballero, Juan J.Lull, Luís Matí-Bonmatí, Montserrat Robles. 2008. Robust MRI brain tissue parameter estimation by multistage outlier rejection. Magnetic resonance in medicine 59: 866-873.

Tutoriales

Mª Victoria Lapuerta González, Ana Laverón Simavilla. Introducción a MATLAB.

Gonzalo Fernández de Córdoba Martos. Creación de interfaces gráficas de usuario con MATLAB.

Ashok Krishnamurthy, Siddharth Samsi. MATLAB for engineering applications.

Diego Orlando Barragán Guerrero. Manual de interfaz gráfica de usuario en MATLAB.

Emma Díez. Imágenes de resonancia magnética nuclear.

ANEXO

En esta sección se mostrará el código de los siguientes scripts implementados para el desarrollo del software:

- brainseg.m
- gui.m
- batch.m
- loadFile.m
- reorientation.m
- volReformat.m
- resizeVol.m
- save_volume.m
- run_batch.m

Brainseg.m

```
function varargout = brainseg(varargin)

% Begin initialization code – DO NOT EDIT

if(exist('defaults.mat','file'))
  load defaults;
  addpath(spmpath);
  addpath(vbmpath);
end;

isspm = findstr(path,'spm8');
if(isempty(isspm))
 spmpath=uigetdir(pwd,'Select SPM8 directory');
 addpath(spmpath);
end

isvbm = findstr(path,'vbm8');
if(isempty(isvbm))
 vbmpath=uigetdir(pwd,'Select VBM8 directory');
 addpath(vbmpath);
 save defaults spmpath vbmpath
end

% Add additional subdirectories
actual=pwd;
addpath([actual, '\modules']);
addpath([actual, '\modules\Denoising']);
addpath([actual, '\modules\BrainExtraction']);
addpath([actual, '\modules\Clasification']);



% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
```

```
                   'gui_OpeningFcn', @brainseg_OpeningFcn, ...
                   'gui_OutputFcn',  @brainseg_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

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


% --- Executes just before brainseg is made visible.
function brainseg_OpeningFcn(hObject, eventdata, handles, varargin)

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

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes brainseg wait for user response (see UIRESUME)
% uiwait(handles.figure1);
global arg
    arg.img_name = [];
    arg.axial=0;
    arg.coronal=0;
    arg.sagittal=0;
    arg.s=[];
    arg.csag=1;
    arg.ccor=1;
    arg.cax=1;

axes(handles.axesLogo)
handles.imagen=imread('logo.tif');
imagesc(handles.imagen)
axis off

%End of global var args


% --- Outputs from this function are returned to the command line.
function varargout = brainseg_OutputFcn(hObject, eventdata, handles)

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

% --- Executes on button press in guiButton.
function guiButton_Callback(hObject, eventdata, handles)

close;

gui;
```

```
% --- Executes on button press in batchButton.
function batchButton_Callback(hObject, eventdata, handles)

close;

batch;
```

gui.m

```
function varargout = gui(varargin)

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

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


% --- Executes just before gui is made visible.
function gui_OpeningFcn(hObject, eventdata, handles, varargin)

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

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui wait for user response (see UIRESUME)
% uiwait(handles.guiWindow);

global vol;
global V;
global arg;
global res;
global f;
global origin;
global volImg;
global volOriginal;
global cutAxial;
global cutSagittal;
global cutCoronal;
global left;
global rect;
```

```
global brillo;
global sizeVolImg;
global maxSize;
global overlay;
global map;
global enable;
global resultspath;
global fname;
global limit_a;
global limit_s;
global limit_c;
global mapa;
global right;
global pencil;

% Initialize tools section

set(handles.disable_editor_menu, 'Enable', 'off'); % Both menus are
disabled by default
set(handles.save_mask_menu, 'Enable', 'off');
% End of init.

enable=0;
right=0;
left=0;
brillo=0;
overlay=0;
pencil=0;

if(isempty(arg.img_name))
  volOriginal=zeros(64,64,64);
  res=[1,1,1];
  arg.axial = 1;

  %Disable tool menus if acquisition is not loaded
  set(handles.denoising_menu, 'Enable', 'off');
  set(handles.correction_menu, 'Enable', 'off');
  set(handles.extraction_menu, 'Enable', 'off');
  set(handles.classification_menu, 'Enable', 'off');
  set(handles.segment_menu, 'Enable', 'off');
  set(handles.save_options_menu, 'Enable', 'off');
  set(handles.buttonZoom, 'Enable', 'off');
  set(handles.buttonPan, 'Enable', 'off');
  set(handles.buttonPencil, 'Enable', 'off');
  set(handles.popupmenu_size, 'Enable', 'off');
  set(handles.popupmenu_label, 'Enable', 'off');

  %Disable overlay options
  set(handles.load_overlay_menu, 'Enable', 'off');
  set(handles.unload_overlay_menu, 'Enable', 'off');
  set(handles.transparency_menu, 'Enable', 'off');

else

  V = spm_vol(arg.img_name);
  volOriginal=spm_read_vols(V);

  res(1)=abs(V.mat(1,1));
  res(2)=abs(V.mat(2,2));
```

```
  res(3)=abs(V.mat(3,3));

  origin(1)=V.mat(1,4);
  origin(2)=V.mat(2,4);
  origin(3)=V.mat(3,4);

  set(handles.denoising_menu, 'Enable', 'on');
  set(handles.correction_menu, 'Enable', 'on');
  set(handles.extraction_menu, 'Enable', 'on');
  set(handles.classification_menu, 'Enable', 'on');
  set(handles.segment_menu, 'Enable', 'on');
  set(handles.save_options_menu, 'Enable', 'on');
  set(handles.buttonZoom, 'Enable', 'on');
  set(handles.buttonPan, 'Enable', 'on');

  %Enable overlay options
  set(handles.load_overlay_menu, 'Enable', 'on');
  set(handles.unload_overlay_menu, 'Enable', 'on');
  set(handles.transparency_menu, 'Enable', 'on');

  %Set saving as axial by default
  set(handles.save_axial_menu, 'Checked', 'on');

  % Set information data in information box

  [infopath,infoname,infoext]=fileparts(arg.img_name);

  infoFilename=[infoname,infoext];

  set(handles.text_info, 'String', { ['Filename: ', infoFilename ] });
  set(handles.text_info, 'String', [get(handles.text_info, 'String'),
{['Volume Size: ', num2str(size(volOriginal)) ]} ]);

end

stemp=size(volOriginal);

% If the acquisition is axial
if (arg.axial == 1)
    colormap(gray);

    arg.s=size(volOriginal);

    % If all voxels are the same size, then call resizeVol function
    if (res(1) == res(2) && res(3)==res(1))
        volImg=resizeVol(volOriginal);
    else
        resMin=min(res); % Store the minimum value of res
        f=res/resMin; % Calculate new resolution value and call to the
interpolation function
        volTemp=volReformat(volOriginal, f);

        volImg=resizeVol(volTemp);
    end



    % createcolormap
    Imin=round(min(volImg(:)));
```

```
    volImg=volImg-Imin;
    Imax=round(max(volImg(:)));
    volImg=volImg*255/Imax;
    Imax=255;
    mapa=zeros(Imax,3);

    % If adquisition is axial copy Original volume in new vol
    vol=volOriginal;

    for i=1:Imax
      mapa(i,1)=(i-1)/(Imax-1);
      mapa(i,2)=(i-1)/(Imax-1);
      mapa(i,3)=(i-1)/(Imax-1);
    end
    colormap(mapa);

    sizeVolImg=size(volImg);

% For each image we'll show the central cut without labels
    axes(handles.axesAxial);
    cutAxial=round((sizeVolImg(3)/2));
    image(imrotate(volImg(:,:,cutAxial),90));
    set(handles.axesAxial, 'Tag', 'axial');
    axis off;

    axes(handles.axesSagital);
    cutSagittal=round((sizeVolImg(1)/2));
    sag(:,:)=volImg(cutSagittal,:,:);
    image(imrotate(sag,90));
    set(handles.axesSagital, 'Tag', 'sagittal');
    axis off;

    axes(handles.axesCoronal);
    cutCoronal=round((sizeVolImg(2)/2));
    cor(:,:)=volImg(:,cutCoronal,:);
    image(imrotate(cor,90));
    set(handles.axesCoronal, 'Tag', 'coronal');
    axis off;

end


% If the acquisition is coronal
if (arg.coronal == 1)
    colormap(gray);

    %Create new matrix to store fixed acquisition in axial orientation
     vol=zeros(stemp(1),stemp(3),stemp(2));

     % Copy axials images in the new matrix
     for i=1:stemp(2)
          vol(:,:,i)=imrotate(squeeze(volOriginal(:,i,:)),180);
     end


      tt=res(2);
      res(2)=res(3);
      res(3)=tt;
```

```
  tt=origin(2);
  origin(2)=origin(3);
  origin(3)=tt;

  arg.s=size(vol);

 % If all voxels are the same size, then call volImg function
if (res(1) == res(2) && res(3)==res(1))
    volImg=resizeVol(vol);
else
    resMin=min(res);
    f=res/resMin;
    volTemp=volReformat(vol, f);
    volImg=resizeVol(volTemp);
end

% createcolormap
Imin=round(min(volImg(:)));
volImg=volImg-Imin+1;
Imax=round(max(volImg(:)));
volImg=volImg*255/Imax;
Imax=255;


mapa=zeros(Imax,3);

for i=1:Imax
  mapa(i,1)=(i-1)/(Imax-1);
  mapa(i,2)=(i-1)/(Imax-1);
  mapa(i,3)=(i-1)/(Imax-1);
end
colormap(mapa);


 sizeVolImg=size(volImg);

 axes(handles.axesAxial);
 cutAxial=round((sizeVolImg(3)/2));
 image(imrotate(volImg(:,:,cutAxial),90));
 set(handles.axesAxial, 'Tag', 'axial');
 axis off;

 axes(handles.axesSagital);
 cutSagittal=round((sizeVolImg(1)/2));
 sag(:,:)=volImg(cutSagittal,:,:);
 image(imrotate(sag,90));
 set(handles.axesSagital, 'Tag', 'sagittal');
 axis off;

 axes(handles.axesCoronal);
 cutCoronal=round((sizeVolImg(2)/2));
 cor(:,:)=volImg(:,cutCoronal,:);
 image(imrotate(cor,90));
 set(handles.axesCoronal, 'Tag', 'coronal');
 axis off;
```

```
end


% If acquisition is sagittal
if (arg.sagittal == 1)
    colormap(gray);

    %Create new matrix to store fixed acquisition in axial orientation
    vol=zeros(stemp(3),stemp(1),stemp(2));

     % Copy axials images in the new matrix
     for i=1:stemp(2)
          vol(:,:,i)=imrotate(squeeze(volOriginal(:,i,:)), 90);
     end

    tt1=res(1);
    tt2=res(2);
    res(1)=res(3);
    res(2)=tt1;
    res(3)=tt2;

    tt1=origin(1);
    tt2=origin(2);
    origin(1)=origin(3);
    origin(2)=tt1;
    origin(3)=tt2;

    arg.s=size(vol);

    if (res(1) == res(2) && res(3)==res(1))
        volImg=resizeVol(vol);
    else
        resMin=min(res);
        f=res/resMin;
        volTemp=volReformat(vol, f);
        volImg=resizeVol(volTemp);
    end

    % createcolormap
    Imin=round(min(volImg(:)));
    volImg=volImg-Imin+1;
    Imax=round(max(volImg(:)));
    volImg=volImg*255/Imax;
    Imax=255;

    vol=double(vol);



    mapa=zeros(Imax,3);
    for i=1:Imax
      mapa(i,1)=(i-1)/(Imax-1);
      mapa(i,2)=(i-1)/(Imax-1);
      mapa(i,3)=(i-1)/(Imax-1);
    end
    colormap(mapa);
```

```
    sizeVolImg=size(volImg);


     axes(handles.axesAxial);
     cutAxial=round((sizeVolImg(3)/2));
     image(imrotate(volImg(:,:,cutAxial),90));
     set(handles.axesAxial, 'Tag', 'axial');
     axis off;

     axes(handles.axesSagital);
     cutSagittal=round((sizeVolImg(1)/2));
     sag(:,:)=volImg(cutSagittal,:,:);
     image(imrotate(sag,90));
     set(handles.axesSagital, 'Tag', 'sagittal');
     axis off;

     axes(handles.axesCoronal);
     cutCoronal=round((sizeVolImg(2)/2));
     cor(:,:)=volImg(:,cutCoronal,:);
     image(imrotate(cor,90));
     set(handles.axesCoronal, 'Tag', 'coronal');
     axis off;



end


% --- Outputs from this function are returned to the command line.
function varargout = gui_OutputFcn(hObject, eventdata, handles)

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


% Update views of volume, depending where x and y has been pressed
function position_update(handles, x, y, axedSelected)

global cutAxial;
global cutCoronal;
global cutSagittal;
global overlay;
global sizeVolImg;
global volImgFusion;
global volImg;

if ( ((x>0) && (x<sizeVolImg(1))) && ((y>0) && (y<sizeVolImg(1))) )

    % For each cut, if we are working with loaded overlay, we'll show
the
    % fusion image instead of normal volume.
    switch axedSelected
        case 'axial'

            axes(handles.axesCoronal);
            cutCoronal=(round((sizeVolImg(2)-y)));
            if (overlay == 1)
              cor(:,:)=volImgFusion(:,cutCoronal,:);
```

```
            else
              cor(:,:)=volImg(:,cutCoronal,:);
            end
            image(imrotate(cor,90));
            set(handles.axesCoronal, 'Tag', 'coronal');
            axis off;

            axes(handles.axesSagital);
            cutSagittal=round(((x)));
            if (overlay == 1)
              sag(:,:)=volImgFusion(cutSagittal,:,:);
            else
              sag(:,:)=volImg(cutSagittal,:,:);
            end
            image(imrotate(sag,90));
            set(handles.axesSagital, 'Tag', 'sagittal');
            axis off;



        case 'coronal'

             axes(handles.axesAxial);
             cutAxial=(sizeVolImg(2)-y);
             if (overlay == 1)
               ax(:,:)=volImgFusion(:,:,cutAxial);
             else
               ax(:,:)=volImg(:,:,cutAxial);
             end
             image(imrotate(ax,90));
             set(handles.axesAxial, 'Tag', 'axial');
             axis off;

             axes(handles.axesSagital);
             cutSagittal=round(((x)));
             if (overlay == 1)
               sag(:,:)=volImgFusion(cutSagittal,:,:);
             else
               sag(:,:)=volImg(cutSagittal,:,:);
             end
             image(imrotate(sag,90));
             set(handles.axesSagital, 'Tag', 'sagittal');
             axis off;

        case 'sagittal'

            axes(handles.axesAxial);
            cutAxial=(sizeVolImg(2)-y);
            if (overlay == 1)
                ax(:,:)=volImgFusion(:,:,cutAxial);
            else
                ax(:,:)=volImg(:,:,cutAxial);
            end
            image(imrotate(ax,90));
            set(handles.axesAxial, 'Tag', 'axial');
            axis off;

            axes(handles.axesCoronal);
            cutCoronal=(round((x)));
```

```
            if (overlay == 1)
              cor(:,:)=volImgFusion(:,cutCoronal,:);
            else
              cor(:,:)=volImg(:,cutCoronal,:);
            end
            image(imrotate(cor,90));
            set(handles.axesCoronal, 'Tag', 'coronal');
            axis off;

    end



end


function [vol]=volReformatOverlay(data,f)

s=round(f.*size(data));

vol=zeros(s);

sOver=size(data);

temp = zeros([s(1),s(2),sOver(3)]);

  for i=1:sOver(3)
    temp(:,:,i)=imresize(data(:,:,i),s(1:2),'bilinear');
  end

   for i=1:sOver(1)
    vol(i,:,:)=imresize(squeeze(temp(i,:,:)),s(2:3),'bilinear');
   end


% --- Executes on scroll wheel click while the figure is in focus.
function guiWindow_WindowScrollWheelFcn(hObject, eventdata, handles)

global volImg;
global volImgFusion;
global cutAxial;
global cutCoronal;
global cutSagittal;
global arg;
global overlay;
global sizeVolImg;

axedSelected=get(gca, 'Tag');

if (length(arg.img_name) ~=0)
    switch axedSelected

        case 'axial'
            axes(handles.axesAxial);
            cutAxial=cutAxial + eventdata.VerticalScrollCount;

            if ((cutAxial >= 1) && (cutAxial < sizeVolImg(3)))
                if (overlay == 1)
```

```
                    image(imrotate(volImgFusion(:,:,cutAxial),90));
                else
                    image(imrotate(volImg(:,:,cutAxial),90));
                end
            else
                cutAxial = cutAxial - eventdata.VerticalScrollCount;
            end

            set(handles.textCoordZ, 'String', cutAxial-1);

            set(handles.axesAxial, 'Tag', 'axial');
            axis off;
        case 'coronal'
            axes(handles.axesCoronal);
            cutCoronal=cutCoronal + eventdata.VerticalScrollCount;

            if ((cutCoronal >= 1) && (cutCoronal < sizeVolImg(3)))
                if (overlay == 1)
                    cor(:,:)=volImgFusion(:,cutCoronal,:);
                else
                    cor(:,:)=volImg(:,cutCoronal,:);
                end
              image(imrotate(cor,90));
            else
                cutCoronal = cutCoronal -
eventdata.VerticalScrollCount;
            end

            set(handles.textCoordY, 'String', cutCoronal-1);

            set(handles.axesCoronal, 'Tag', 'coronal');
            axis off;

        case 'sagittal'

            axes(handles.axesSagital);

            cutSagittal= cutSagittal + eventdata.VerticalScrollCount;

            if ((cutSagittal >= 1) && (cutSagittal < sizeVolImg(3)))
                if (overlay == 1)
                    sag(:,:)=volImgFusion(cutSagittal,:,:);
                else
                    sag(:,:)=volImg(cutSagittal,:,:);
                end
                image(imrotate(sag,90));
            else
                cutSagittal = cutSagittal -
eventdata.VerticalScrollCount;
            end

            set(handles.textCoordX, 'String', cutSagittal-1);

            set(handles.axesSagital, 'Tag', 'sagittal');
            axis off;
    end
end
```

```
% --- Executes on mouse press over figure background, over a disabled
or
% --- inactive control, or over an axes background.
function guiWindow_WindowButtonDownFcn(hObject, eventdata, handles)

 global volImg;
 global volImgFusion;
 global sizeVolImg;
 global left;
 global f;
 global rect;
 global overImg;
 global volImgOverlay;
 global offsetx;
 global offsety;
 global offsetz;
 global cutAxial;
 global cutSagittal;
 global cutCoronal;
 global arg;
 global enable;
 global pencil;
 global right;
 global sizeOverImg;

 axedSelected=get(gca, 'Tag');
 pos=get(gca,'CurrentPoint');
 x=round(pos(1,1));
 y=round(pos(1,2));


if strcmp(get(gcf,'SelectionType'),'normal')
    left=1;
    right=0;
end
if strcmp(get(gcf,'SelectionType'),'alt')
    left=0;
    right=1;
end

% If editor is enable and we want to draw
if(enable == 1 && pencil==1)

  if ( ((x>0) && (x<sizeVolImg(1))) && ((y>0) && (y<sizeVolImg(1))) )


            switch axedSelected
                case 'axial'

                    cutCoronal=(round((sizeVolImg(2)-y+1)));
                    cutSagittal=round(((x)));

                case 'coronal'

                     cutAxial=(sizeVolImg(2)-y)+1;
                     cutSagittal=round(((x)));

                case 'sagittal'
```

```
                    cutAxial=(sizeVolImg(2)-y+1);
                    cutCoronal=(round((x)));
            end

            x1=round((cutSagittal-offsetx)/f(1));
            y1=round((cutCoronal-offsety)/f(2));
            z1=round((cutAxial-offsetz)/f(3));


            sizeSelected = get(handles.popupmenu_size, 'Value')-1;


            % Left click
            if(left==1)

                labelMax=max(overImg(:));
                labelValue=get(handles.popupmenu_label, 'Value');

                for i=-sizeSelected:sizeSelected
                    for j=-sizeSelected:sizeSelected
                        for k=-sizeSelected:sizeSelected
                            newX=x1+i;
                            newY=y1+j;
                            newZ=z1+k;

                            d = sqrt(i*i + j*j + k*k);
                            if ( d > sizeSelected)
                                continue;
                            end

                            if( (newX>0 && newX<=sizeOverImg(1)) &&
(newY>0 && newY<=sizeOverImg(2)) && (newZ>0 && newZ<=sizeOverImg(3)) )

                                overImg(newX, newY, newZ)=labelValue;

                            end

                            newSagittal=cutSagittal+i;
                            newCoronal=cutCoronal+j;
                            newAxial=cutAxial+k;


                            if ( ((newSagittal>0) &&
(newSagittal<sizeVolImg(1))) && ((newCoronal>0) &&
(newCoronal<sizeVolImg(1))) && ((newAxial>0) &&
(newAxial<sizeVolImg(1))))
                                volImgOverlay(newSagittal, newCoronal,
newAxial)=(255*labelValue)/labelMax;
                                volImgFusion(newSagittal, newCoronal,
newAxial)=129 + (volImg(newSagittal, newCoronal,
newAxial).*(volImgOverlay(newSagittal, newCoronal, newAxial)))/(256);
                            end

                        end
                    end
                end
```

```
                if (labelValue > labelMax)
                    change_alfa(labelValue);
                end

            end % End of if left


            % Right click
            if(right==1)

                for i=-sizeSelected:sizeSelected
                    for j=-sizeSelected:sizeSelected
                        for k=-sizeSelected:sizeSelected
                            newX=x1+i;
                            newY=y1+j;
                            newZ=z1+k;

                            d = sqrt(i*i + j*j + k*k);
                            if ( d > sizeSelected)
                                continue;
                            end

                            if( (newX>0 && newX<=sizeOverImg(1)) &&
(newY>0 && newY<=sizeOverImg(2)) && (newZ>0 && newZ<=sizeOverImg(3)) )
                                overImg(newX, newY, newZ)=0;
                            end

                            newSagittal=cutSagittal+i;
                            newCoronal=cutCoronal+j;
                            newAxial=cutAxial+k;

                            volImgOverlay(newSagittal, newCoronal,
newAxial)=0;
                            volImgFusion(newSagittal, newCoronal,
newAxial)=0.5*volImg(newSagittal, newCoronal, newAxial);
                        end
                    end
                end

            end % End of if right

            axes(handles.axesAxial);
            image(imrotate(volImgFusion(:,:,cutAxial),90));
            set(handles.axesAxial, 'Tag', 'axial');
            axis off;
            zoom(1);

            axes(handles.axesSagital);
            sag(:,:)=volImgFusion(cutSagittal,:,:);
            image(imrotate(sag,90));
            set(handles.axesSagital, 'Tag', 'sagittal');
            axis off;
            zoom(1);

            axes(handles.axesCoronal);
            cor(:,:)=volImgFusion(:,cutCoronal,:);
            image(imrotate(cor,90));
            set(handles.axesCoronal, 'Tag', 'coronal');
            axis off;
```

```
            zoom(1);

   end
end

if pencil==0 && strcmp(get(gcf,'SelectionType'),'alt')

         if(strcmp(axedSelected,'axial'))
             rect=getrect(handles.axesAxial);
             rect=round(rect);

reg=volImg(rect(1):(rect(1)+rect(3)),rect(2):(rect(2)+rect(4)),cutAxia
l);
             m1=min(reg(:));
             m2=max(reg(:));
             CambiarBrillo(m1,m2);
             return;
         end

         if(strcmp(axedSelected,'sagittal'))
            rect=getrect(handles.axesSagital);
            rect=round(rect);

reg=volImg(rect(1):(rect(1)+rect(3)),rect(2):(rect(2)+rect(4)),cutSagi
ttal);
            m1=min(reg(:));
            m2=max(reg(:));
            CambiarBrillo(m1,m2);
            return;
         end

         if(strcmp(axedSelected,'coronal'))
            rect=getrect(handles.axesCoronal);
            rect=round(rect);

reg=volImg(rect(1):(rect(1)+rect(3)),rect(2):(rect(2)+rect(4)),cutCoro
nal);
            m1=min(reg(:));
            m2=max(reg(:));
            CambiarBrillo(m1,m2);
            return;
         end
end

position_update(handles, x, y, axedSelected);

if(length(arg.img_name) ~= 0)
        switch axedSelected
            case 'axial'
                axes(handles.axesAxial);
                set(handles.textCoordZ, 'String', cutAxial-1);
            case 'coronal'
                axes(handles.axesCoronal);
                set(handles.textCoordY, 'String', cutCoronal-1);
            case 'sagittal'
                axes(handles.axesSagital);
                set(handles.textCoordX, 'String', cutSagittal-1);
        end
end
```

```
% --------------------------------------------------------------------
function new_acquisition_Callback(hObject, eventdata, handles)
global overlay;

overlay=0;
loadFile;


% --- Executes on mouse motion over figure - except title and menu.
function guiWindow_WindowButtonMotionFcn(hObject, eventdata, handles)

 global volImg;
 global volImgFusion;
 global volImgOverlay;
 global overImg;
 global left;
 global right;
 global cutAxial;
 global cutSagittal;
 global cutCoronal;
 global arg;
 global f;
 global offsetx;
 global offsety;
 global offsetz;
 global enable;
 global pencil;
 global sizeVolImg;
 global sizeOverImg;

 s=sizeVolImg;

 axedSelected=get(gca, 'Tag');

 posPointer=get(gca,'CurrentPoint');

 if(length(arg.img_name)~=0)

     switch axedSelected
        case 'axial'
            xP=round(posPointer(1,1));
            yP=(s(2)-round(posPointer(1,2)));
            if(xP>0 && xP<=s(1) && yP>0 && yP<=s(2))
                set(handles.textCoordX, 'String', xP-1);
                set(handles.textCoordY, 'String', yP-1);
                set(handles.textCoordZ, 'String', cutAxial-1);
                set(handles.valor, 'String', volImg(xP,yP,cutAxial));
            end
        case 'coronal'
            xP=round(posPointer(1,1));
            yP=(s(3)-round(posPointer(1,2)));
            if(xP>0 && xP<=s(1) && yP>0 && yP<=s(3))
                set(handles.textCoordX, 'String', xP-1);
                set(handles.textCoordZ, 'String', yP-1);
                set(handles.textCoordY, 'String', cutCoronal-1);
                set(handles.valor, 'String',
volImg(xP,cutCoronal,yP));
            end
```

```
        case 'sagittal'
            xP=round(posPointer(1,1));
            yP=(s(3)-round(posPointer(1,2)));
            if(xP>0 && xP<=s(1) && yP>0 && yP<=s(3))
                set(handles.textCoordY, 'String', xP-1);
                set(handles.textCoordZ, 'String', yP-1);
                set(handles.textCoordX, 'String', cutSagittal-1);
                set(handles.valor, 'String',
volImg(cutSagittal,xP,yP));
            end
      end

 end


axedSelected=get(gca, 'Tag');
pos=get(gca,'CurrentPoint');
x=round(pos(1,1));
y=round(pos(1,2));


if ( (left == 1 || right==1) && (enable == 1) && (pencil == 1))

    if ( ((x>0) && (x<sizeVolImg(1))) && ((y>0) && (y<sizeVolImg(1))))


            switch axedSelected
                case 'axial'

                    cutCoronal=(round((sizeVolImg(2)-y+1)));
                    cutSagittal=round(((x)));

                case 'coronal'

                     cutAxial=(sizeVolImg(2)-y+1);
                     cutSagittal=round(((x)));

                case 'sagittal'

                    cutAxial=(sizeVolImg(2)-y+1);
                    cutCoronal=(round((x)));

            end

            x1=round((cutSagittal-offsetx)/f(1));
            y1=round((cutCoronal-offsety)/f(2));
            z1=round((cutAxial-offsetz)/f(3));

            sizeSelected = get(handles.popupmenu_size, 'Value')-1;


            if(left==1)

                labelMax=max(overImg(:));
                labelValue=get(handles.popupmenu_label, 'Value');

                for i=-sizeSelected:sizeSelected
```

```
                    for j=-sizeSelected:sizeSelected
                        for k=-sizeSelected:sizeSelected
                            newX=x1+i;
                            newY=y1+j;
                            newZ=z1+k;

                            d = sqrt(i*i + j*j + k*k);
                            if ( d > sizeSelected)
                                continue;
                            end

                            if( (newX>0 && newX<=sizeOverImg(1)) &&
(newY>0 && newY<=sizeOverImg(2)) && (newZ>0 && newZ<=sizeOverImg(3)) )

                                overImg(newX, newY, newZ)=labelValue;

                            end

                            newSagittal=cutSagittal+i;
                            newCoronal=cutCoronal+j;
                            newAxial=cutAxial+k;



                            if ( ((newSagittal>0) &&
(newSagittal<sizeVolImg(1))) && ((newCoronal>0) &&
(newCoronal<sizeVolImg(1))) && ((newAxial>0) &&
(newAxial<sizeVolImg(1))))
                                volImgOverlay(newSagittal, newCoronal,
newAxial)=(255*labelValue)/labelMax;
                                volImgFusion(newSagittal, newCoronal,
newAxial)=129 + (volImg(newSagittal, newCoronal,
newAxial).*(volImgOverlay(newSagittal, newCoronal, newAxial)))/(256);
                            end

                        end
                    end
                end


                if (labelValue > labelMax)
                    change_alfa(labelValue);
                end
            end

            if(right==1)

                for i=-sizeSelected:sizeSelected
                    for j=-sizeSelected:sizeSelected
                        for k=-sizeSelected:sizeSelected
                            newX=x1+i;
                            newY=y1+j;
                            newZ=z1+k;

                            d = sqrt(i*i + j*j + k*k);
                            if ( d > sizeSelected)
                                continue;
                            end
```

```
                                if( (newX>0 && newX<=sizeOverImg(1)) &&
(newY>0 && newY<=sizeOverImg(2)) && (newZ>0 && newZ<=sizeOverImg(3)) )
                                    overImg(newX, newY, newZ)=0;
                                end

                                newSagittal=cutSagittal+i;
                                newCoronal=cutCoronal+j;
                                newAxial=cutAxial+k;

                                volImgOverlay(newSagittal, newCoronal,
newAxial)=0;
                                volImgFusion(newSagittal, newCoronal,
newAxial)=0.5*volImg(newSagittal, newCoronal, newAxial);
                            end
                        end
                    end
                end

                axes(handles.axesAxial);
                image(imrotate(volImgFusion(:,:,cutAxial),90));
                set(handles.axesAxial, 'Tag', 'axial');
                axis off;
                zoom(1);

                axes(handles.axesSagital);
                sag(:,:)=volImgFusion(cutSagittal,:,:);
                image(imrotate(sag,90));
                set(handles.axesSagital, 'Tag', 'sagittal');
                axis off;
                zoom(1);

                axes(handles.axesCoronal);
                cor(:,:)=volImgFusion(:,cutCoronal,:);
                image(imrotate(cor,90));
                set(handles.axesCoronal, 'Tag', 'coronal');
                axis off;
                zoom(1);

    end
end

if(left==1 && pencil == 0)

    position_update(handles, x, y, axedSelected);

    switch axedSelected
        case 'axial'
            axes(handles.axesAxial);
        case 'coronal'
            axes(handles.axesCoronal);
        case 'sagittal'
            axes(handles.axesSagital);
    end
end



% --- Executes on mouse press over figure background, over a disabled
or
```

```
% --- inactive control, or over an axes background.
function guiWindow_WindowButtonUpFcn(hObject, eventdata, handles)

global left;
global right;

right=0;
left=0;


function CambiarBrillo(min,max)
    global overlay;
    global mapa;

    min=ceil(min);
    if (min == 0)
        min=1;
    end

    max=ceil(max);
    if (max == 0 || max == 1)
        max=2;
    end


    for i=1:min
          mapa(i,1)=0;
          mapa(i,2)=0;
          mapa(i,3)=0;
    end

    for i=min:max
          mapa(i,1)=(i-min+1)/(max-min+1);
          mapa(i,2)=(i-min+1)/(max-min+1);
          mapa(i,3)=(i-min+1)/(max-min+1);
    end

    for i=max:256
          mapa(i,1)=1;
          mapa(i,2)=1;
          mapa(i,3)=1;
    end

    if (overlay == 1)
        change_alfa(0);
    else
        colormap(mapa);
    end

function run_denoising_menu_Callback(hObject, eventdata, handles)

nV=save_volume;
denoise(nV.fname);

function noise_estimation_menu_Callback(hObject, eventdata, handles)

global volOriginal;
```

```
disp('Estimating noise...');
h=RicianSTD(volOriginal);
disp(['Noise estimated: ', num2str(h)]);

function correction_menu_Callback(hObject, eventdata, handles)

nV=save_volume;
NUCorrect(nV.fname);


function extraction_menu_Callback(hObject, eventdata, handles)

nV=save_volume;
brainextract(nV.fname);

function classification_menu_Callback(hObject, eventdata, handles)

nV=save_volume;
clasify(nV.fname);

function segment_menu_Callback(hObject, eventdata, handles)

nV=save_volume;
nV.fname
segment(nV.fname);
[path, name, ext]=fileparts(nV.fname);
fileName=[path, '\crisp_', name, ext];
load_mask(fileName);


function showVolume(overlay, handles)

global cutAxial;
global cutCoronal;
global cutSagittal;
global volImg;
global volImgFusion;

    if (overlay == 1)

        axes(handles.axesAxial);
        image(imrotate(volImgFusion(:,:,cutAxial),90));
        set(handles.axesAxial, 'Tag', 'axial');
        axis off;

        axes(handles.axesSagital);
        sag(:,:)=volImgFusion(cutSagittal,:,:);
        image(imrotate(sag,90));
        set(handles.axesSagital, 'Tag', 'sagittal');
        axis off;

        axes(handles.axesCoronal);
        cor(:,:)=volImgFusion(:,cutCoronal,:);
        image(imrotate(cor,90));
        set(handles.axesCoronal, 'Tag', 'coronal');
        axis off;

    elseif (overlay == 0)
```

```
        axes(handles.axesAxial);
        image(imrotate(volImg(:,:,cutAxial),90));
        set(handles.axesAxial, 'Tag', 'axial');
        axis off;

        axes(handles.axesSagital);
        sag(:,:)=volImg(cutSagittal,:,:);
        image(imrotate(sag,90));
        set(handles.axesSagital, 'Tag', 'sagittal');
        axis off;

        axes(handles.axesCoronal);
        cor(:,:)=volImg(:,cutCoronal,:);
        image(imrotate(cor,90));
        set(handles.axesCoronal, 'Tag', 'coronal');
        axis off;
    end


function load_mask(fileNameOverlay)

global res;
global alfa;
global volImg;
global volImgOverlay;
global volImgFusion;
global overlay;
global overImg;
global f;
global V;
global VOverlay;
global mapa;
global sizeOverImg;


if ~(fileNameOverlay == 0)
    overlayFile=strcat(fileNameOverlay);
    VOverlay = spm_vol(overlayFile);

    if (V.dim ~= VOverlay.dim)
        errordlg('Volume and mask dimensions must be the same',
'Dimension mismatch');
        return;
    end

    overImg=spm_read_vols(VOverlay);

    sizeOverImg=size(overImg);

    resOverlay(1)=abs(VOverlay.mat(1,1));
    resOverlay(2)=abs(VOverlay.mat(2,2));
    resOverlay(3)=abs(VOverlay.mat(3,3));

    resMin=min(res);

    f=resOverlay/resMin;
```

```
    volTemp=volReformat(overImg, f);

    volImgOverlay=resizeVol(volTemp);

    volImgOverlay=volImgOverlay*256/max(volImgOverlay(:));

    overlay=1;


    alfa=0.75;
    volImgFusion=0.5*volImg;
    ind=find(volImgOverlay>0);
    volImgFusion(ind)=129 + (volImg(ind).*(volImgOverlay(ind)))/(256);

    if(max(overImg(:))>7)
        disp('error: max number of tags is 7.');
        return;
    end

    cm=max(overImg(:));
    colors=[1,0,0;0 1 0; 0 0 1; 1 1 0; 0 1 1;1 0 1;1 1 1];
    size(mapa);
    mcm=mapa(1:2:end,:);
    for i=1:128
      mcm(128+i,1)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),1);
      mcm(128+i,2)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),2);
      mcm(128+i,3)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),3);
    end

    colormap(mcm);

    showVolume(overlay, handles);

end

% --------------------------------------------------------------------
function load_overlay_menu_Callback(hObject, eventdata, handles)

global res;
global alfa;
global volImg;
global volImgOverlay;
global volImgFusion;
global overlay;
global overImg;
global f;
global V;
global VOverlay;
global mapa;
global sizeOverImg;


[fileNameOverlay path]=uigetfile({'*.img; *.nii'}, 'Choose your
overlay file');
```

```
if ~(fileNameOverlay == 0)
    overlayFile=strcat(path, fileNameOverlay);
    VOverlay = spm_vol(overlayFile);

    if (V.dim ~= VOverlay.dim)
        errordlg('Volume and mask dimensions must be the same',
'Dimension mismatch');
        return;
    end

    overImg=spm_read_vols(VOverlay);

    sizeOverImg=size(overImg);

    resOverlay(1)=abs(VOverlay.mat(1,1));
    resOverlay(2)=abs(VOverlay.mat(2,2));
    resOverlay(3)=abs(VOverlay.mat(3,3));

    resMin=min(res);

    f=resOverlay/resMin;

    volTemp=volReformat(overImg, f);

    volImgOverlay=resizeVol(volTemp);

    volImgOverlay=volImgOverlay*256/max(volImgOverlay(:));

    overlay=1;


    alfa=0.75;
    volImgFusion=0.5*volImg;
    ind=find(volImgOverlay>0);
    volImgFusion(ind)=129 + (volImg(ind).*(volImgOverlay(ind)))/(256);

    if(max(overImg(:))>7)
        disp('error: max number of tags is 7.');
        return;
    end

    cm=max(overImg(:));
    colors=[1,0,0;0 1 0; 0 0 1; 1 1 0; 0 1 1;1 0 1;1 1 1];
    size(mapa);
    mcm=mapa(1:2:end,:);
    for i=1:128
      mcm(128+i,1)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),1);
      mcm(128+i,2)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),2);
      mcm(128+i,3)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),3);
    end

    colormap(mcm);

    showVolume(overlay, handles);
```

```
end



function change_alfa(newMax)

global alfa;
global volImg;
global volImgOverlay;
global volImgFusion;
global overImg;
global mapa;

volImgFusion=0.5*volImg;
ind=find(volImgOverlay>0);
volImgFusion(ind)=129 + (volImg(ind).*(volImgOverlay(ind)))/(256);

cm=max(overImg(:));

if (newMax > cm)
    cm = newMax;
end

colors=[1,0,0;0 1 0; 0 0 1; 1 1 0; 0 1 1;1 0 1;1 1 1];
mcm=mapa(1:2:end,:);
for i=1:128
  mcm(128+i,1)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),1);
  mcm(128+i,2)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),2);
  mcm(128+i,3)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),3);
end

colormap(mcm);



function unload_overlay_menu_Callback(hObject, eventdata, handles)

global overlay;
global mapa;
global enable;

overlay=0;
colormap(mapa);

enable=0;
set(handles.enable_editor_menu, 'Enable', 'on');
set(handles.disable_editor_menu, 'Enable', 'off');
set(handles.save_mask_menu, 'Enable', 'off');
set(handles.buttonPencil, 'Enable', 'off');
set(handles.popupmenu_size, 'Enable', 'off');
set(handles.popupmenu_label, 'Enable', 'off');

showVolume(overlay,handles);

function transparency_0_Callback(hObject, eventdata, handles)
```

```
global alfa;

alfa=0;
change_alfa(0);
function transparency_25_Callback(hObject, eventdata, handles)

global alfa;
alfa=0.25;
change_alfa(0);

function transparency_50_Callback(hObject, eventdata, handles)

global alfa;
alfa=0.5;
change_alfa(0);


function transparency_100_Callback(hObject, eventdata, handles)

global alfa;
alfa=1;
change_alfa(0);

function transparency_75_Callback(hObject, eventdata, handles)

global alfa;
alfa=0.75;
change_alfa(0);

function enable_editor_menu_Callback(hObject, eventdata, handles)

global enable;
global overlay;
global overImg;
global volImg;
global res;
global alfa;
global sizeOverImg;
global volOriginal;
global volImgFusion;
global mapa;
global f;
global V;
global VOverlay;
global arg;

if (isempty(arg.img_name))
    errordlg('You have to load volume image before enable the editor
tool.', 'No Volume Loaded');

else

    if (overlay == 0)
        st=size(volOriginal);

        overImg=zeros(st(1), st(2), st(3));

        sizeOverImg=size(overImg);
```

```
        if (enable == 0)
            enable=1;
            set(handles.enable_editor_menu, 'Enable', 'off');
            set(handles.disable_editor_menu, 'Enable', 'on');
            set(handles.save_mask_menu, 'Enable', 'on');
            set(handles.buttonPencil, 'Enable', 'on');
            set(handles.popupmenu_size, 'Enable', 'on');
            set(handles.popupmenu_label, 'Enable', 'on');

        end

        VOverlay = V;

        resOverlay(1)=abs(VOverlay.mat(1,1));
        resOverlay(2)=abs(VOverlay.mat(2,2));
        resOverlay(3)=abs(VOverlay.mat(3,3));


        resMin=min(res);

        f=resOverlay/resMin;

        volTemp=volReformat(overImg, f);

        volImgOverlay=resizeVol(volTemp);

        volImgOverlay=volImgOverlay*256/max(volImgOverlay(:));

        overlay=1;

        alfa=0.75;
        volImgFusion=0.5*volImg;
        ind=find(volImgOverlay>0);
        volImgFusion(ind)=129 +
(volImg(ind).*(volImgOverlay(ind)))/(256);

        if(max(overImg(:))>7)
            disp('error: max number of tags is 7.');
            return;
        end

        cm=max(overImg(:));

        if (cm == 0)
            cm = 1;
        end

        colors=[1,0,0;0 1 0; 0 0 1; 1 1 0; 0 1 1;1 0 1;1 1 1];
        size(mapa);
        mcm=mapa(1:2:end,:);
        for i=1:128
          mcm(128+i,1)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),1);
          mcm(128+i,2)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),2);
          mcm(128+i,3)=alfa*((i-1)/127) + (1-
alfa)*colors(ceil(double(i*cm)/128),3);
```

```
        end

        colormap(mcm);

        showVolume(overlay, handles);



    else
        if (enable == 0)
            enable=1;
            set(handles.enable_editor_menu, 'Enable', 'off');
            set(handles.disable_editor_menu, 'Enable', 'on');
            set(handles.save_mask_menu, 'Enable', 'on');
            set(handles.buttonPencil, 'Enable', 'on');
            set(handles.popupmenu_size, 'Enable', 'on');
            set(handles.popupmenu_label, 'Enable', 'on');

        end
    end

end

function disable_editor_menu_Callback(hObject, eventdata, handles)

global enable;

if (enable == 1)
    enable=0;
    set(handles.enable_editor_menu, 'Enable', 'on');
    set(handles.disable_editor_menu, 'Enable', 'off');
    set(handles.save_mask_menu, 'Enable', 'off');
    set(handles.buttonPencil, 'Enable', 'off');
    set(handles.popupmenu_size, 'Enable', 'off');
    set(handles.popupmenu_label, 'Enable', 'off');
end


function save_axial_menu_Callback(hObject, eventdata, handles)

set(handles.save_axial_menu, 'Checked', 'on');
set(handles.save_original_menu, 'Checked', 'off');


function save_original_menu_Callback(hObject, eventdata, handles)

set(handles.save_axial_menu, 'Checked', 'off');
set(handles.save_original_menu, 'Checked', 'on');

function exit_menu_Callback(hObject, eventdata, handles)

close all


function save_mask_menu_Callback(hObject, eventdata, handles)

global overImg;
global VOverlay;
```

```
[path,filename,ext]=fileparts(VOverlay.fname);
filename=[filename,ext];
[fileNameOverlay path]=uiputfile(filename, 'Save your overlay file');

if (fileNameOverlay ~= 0)

    VOverlay.fname=fileNameOverlay;
    spm_write_vol(VOverlay, overImg);

    disp('Mask saved!');
end


% --- Executes on button press in buttonZoom.
function buttonZoom_Callback(hObject, eventdata, handles)

global volImgFusion;
global cutCoronal;
global cutAxial;
global cutSagittal;
global limit_a;
global limit_s;
global limit_c;

if ( get(handles.buttonZoom, 'Value') == 1 )
    set(handles.buttonPan, 'Value', 0);
    set(handles.buttonPencil, 'Value', 0);
    set(handles.buttonPencil, 'ForegroundColor', 'Black');
    pan off;

limit_a = getappdata(handles.axesAxial,'zoom_zoomOrigAxesLimits');
limit_s = getappdata(handles.axesSagital,'zoom_zoomOrigAxesLimits');
limit_c = getappdata(handles.axesCoronal,'zoom_zoomOrigAxesLimits');

zoom on;
zoom reset;

else
    zoom off;
end

function buttonPan_Callback(hObject, eventdata, handles)

if ( get(handles.buttonPan, 'Value') == 1 )
    set(handles.buttonZoom, 'Value', 0);
    set(handles.buttonPencil, 'Value', 0);
    set(handles.buttonPencil, 'ForegroundColor', 'Black');
    zoom off;

    pan on;
else
    pan off;
end


function buttonPencil_Callback(hObject, eventdata, handles)
```

```
global pencil;
global volImgFusion;
global cutCoronal;
global cutAxial;
global cutSagittal;
global limit_a;
global limit_s;
global limit_c;


if ( get(handles.buttonPencil, 'Value') == 1 )
    set(handles.buttonPencil, 'ForegroundColor', 'Red');

    set(handles.buttonZoom, 'Value', 0);
    zoom off;

    set(handles.buttonPan, 'Value', 0);
    pan off;

    pencil = 1;

    zoom reset;

else
    set(handles.buttonPencil, 'ForegroundColor', 'Black');
    pencil = 0;

    setappdata(handles.axesAxial,'zoom_zoomOrigAxesLimits',limit_a);
    setappdata(handles.axesSagital,'zoom_zoomOrigAxesLimits',limit_s);
    setappdata(handles.axesCoronal,'zoom_zoomOrigAxesLimits',limit_c);

    axes(handles.axesAxial);
    image(imrotate(volImgFusion(:,:,cutAxial),90));
    set(handles.axesAxial, 'Tag', 'axial');
    axis off;

    axes(handles.axesSagital);
    sag(:,:)=volImgFusion(cutSagittal,:,:);
    image(imrotate(sag,90));
    set(handles.axesSagital, 'Tag', 'sagittal');
    axis off;

    axes(handles.axesCoronal);
    cor(:,:)=volImgFusion(:,cutCoronal,:);
    image(imrotate(cor,90));
    set(handles.axesCoronal, 'Tag', 'coronal');
    axis off;

end


% --- Executes during object creation, after setting all properties.
function popupmenu_size_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes during object creation, after setting all properties.
function popupmenu_label_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function documentation_menu_Callback(hObject, eventdata, handles)

dir=[pwd, '\doc\index.html'];
open(dir);

function about_menu_Callback(hObject, eventdata, handles)

open about.fig;
```

batch.m

```
function varargout = batch(varargin)

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

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


% --- Executes just before batch is made visible.
function batch_OpeningFcn(hObject, eventdata, handles, varargin)

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

% Update handles structure
guidata(hObject, handles);

global checkboxActive;
checkboxActive = 0;
global numFiles;
numFiles=0;
global opts;
```

```
opts.segment=0;
opts.denoising=0;
opts.correction=0;
opts.extraction=0;
opts.clasify=0;
opts.remove=0;
opts.reorient=1;
opts.orientation='axial';


% --- Outputs from this function are returned to the command line.
function varargout = batch_OutputFcn(hObject, eventdata, handles)

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

function select_files_batch_menu_Callback(hObject, eventdata, handles)

global numFiles;
global files;

[fileName pathBatch]=uigetfile({'*.img; *.nii'}, 'Choose your
acquisition file(s)', 'MultiSelect', 'on');

tam=size(fileName);
if(tam(1,2) >1)

    if ( ischar(fileName(1,1)) )
            numFiles=1;
            set(handles.text_files_loaded, 'String',
[num2str(numFiles), ' file loaded']);
            files=[pathBatch,char(fileName)];
    else
            selected=size(fileName);
            numFiles=selected(1,2);
            set(handles.text_files_loaded, 'String',
[num2str(numFiles), ' files loaded']);

            for i=1:numFiles
              filestemp(i,:)=([pathBatch,char(fileName(i))]);
            end
            files=filestemp;
    end

end


function select_directory_batch_menu_Callback(hObject, eventdata,
handles)

global files;
global numFiles;
global opts;

directoryPath = uigetdir('Choose your acquisitions folder');

% If we've selected one directory..
```

```
if (directoryPath ~=0)

    lista1=ls([directoryPath, '/*.nii']);
    lista2=ls([directoryPath, '/*.img']);

    s1=size(lista1);
    s2=size(lista2);

    files=char(zeros((s1(1)+s2(1)),max(s1(2),s2(2))));
    s=size(files);

    for i=1:s1(1)
      files(i,1:length(lista1(i,:)))=lista1(i,:);
    end
    for i=(s1(1)+1):s(1)
      files(i,1:length(lista2(i-s1(1),:)))=lista2(i-s1(1),:);
    end

    tam=size(files);
    numFiles=tam(1,1);

    if (numFiles == 1)
        set(handles.text_files_loaded, 'String', [num2str(numFiles), '
file loaded']);
    elseif (numFiles > 1)
        set(handles.text_files_loaded, 'String', [num2str(numFiles), '
files loaded']);
    end

    for i=1:numFiles
        filestemp(i,:)=[directoryPath,'\',files(i,:)];
    end
    files=filestemp;

end


function exit_batch_menu_Callback(hObject, eventdata, handles)

global files;
global numFiles;
clearvar files;
clearvar numFiles;
close;


function editPath_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function button_set_path_Callback(hObject, eventdata, handles)

global savePath;
```

```
savePath = uigetdir('Choose your acquisitions folder');

if (savePath ~= 0)
    set(handles.editPath, 'String', savePath);
end


function pushbutton_run_Callback(hObject, eventdata, handles)

global checkboxActive;
global numFiles;
global files;
global opts;

global pathSave;

if( (isempty(get(handles.editPath, 'String'))) &&
(isempty(get(handles.editBrainseg, 'String'))) )
    errordlg('You must specify the destination path before run batch
process.', 'No Path Specified');
elseif (checkboxActive == 0)
    errordlg('You must choose at least one process to run.', 'No
Processes Chosen');
elseif (numFiles == 0)
    errordlg('Load any file before run.', 'No File(s) Loaded');


else

    warning('OFF', 'MATLAB:MKDIR:DirectoryExists'); % Disable warning

    disp('Creating directories and storing data in axial
orientation...');

        if (~isempty(get(handles.editPath, 'String'))) % If user
entered a complete path
            pathSave=get(handles.editPath, 'String');
        else
            userSelect=get(handles.editBrainseg, 'String'); %If user
typed a personal dir under brainseg\results\
            mkdir(['results\', userSelect]);
            pathSave=[pwd, '\results\', userSelect];
        end

        opts.nfiles=numFiles;
        opts.save=pathSave;

        if (get(handles.radio_axial, 'Value') == 1)
            opts.orientation='axial';
        end
        if (get(handles.radio_coronal, 'Value') == 1)
            opts.orientation='coronal';
        end
        if (get(handles.radio_sagittal, 'Value') == 1)
            opts.orientation='sagittal';
        end

        if (get(handles.checkbox_segment, 'Value') == 1)
            opts.segment=1;
```

```
        end

        if (get(handles.checkbox_remove_intermedied, 'Value') == 1)
            opts.remove=1;
        end

        if (get(handles.checkbox_denoising, 'Value') == 1)
            opts.denoising=1;
        end

        if (get(handles.checkbox_correction, 'Value') == 1)
            opts.correction=1;
        end

        if (get(handles.checkbox_extraction, 'Value') == 1)
            opts.extraction=1;
        end

        if (get(handles.checkbox_classification, 'Value') == 1)
            opts.clasify=1;
        end


        run_batch(files,opts);

        close;
end

function checkbox_denoising_Callback(hObject, eventdata, handles)

global checkboxActive;

if (get(hObject, 'Value') == 1)
    checkboxActive=1;
else
    checkboxActive=0;
end


function checkbox_correction_Callback(hObject, eventdata, handles)

global checkboxActive;

if (get(hObject, 'Value') == 1)
    checkboxActive=1;
else
    checkboxActive=0;
end

function checkbox_extraction_Callback(hObject, eventdata, handles)

global checkboxActive;

if (get(hObject, 'Value') == 1)
    checkboxActive=1;
else
    checkboxActive=0;
end
```

```
function checkbox_segment_Callback(hObject, eventdata, handles)

global checkboxActive;

if (get(hObject, 'Value') == 1)
    set(handles.checkbox_denoising, 'Value', 1);
    set(handles.checkbox_correction, 'Value', 1);
    set(handles.checkbox_extraction, 'Value', 1);
    set(handles.checkbox_classification, 'Value', 1);
    checkboxActive=1;
else
    set(handles.checkbox_denoising, 'Value', 0);
    set(handles.checkbox_correction, 'Value', 0);
    set(handles.checkbox_extraction, 'Value', 0);
    set(handles.checkbox_classification, 'Value', 0);
    checkboxActive=0;
end


function checkbox_classification_Callback(hObject, eventdata, handles)

global checkboxActive;

if (get(hObject, 'Value') == 1)
    checkboxActive=1;
else
    checkboxActive=0;
end

function panel_orientation_SelectionChangeFcn(hObject, eventdata,
handles)

if (hObject == handles.radio_axial)
    opts.orientation='axial';
end
if (hObject == handles.radio_coronal)
    opts.orientation='coronal';
end
if (hObject == handles.radio_sagittal)
    opts.orientation='sagittal';
end


function panel_orientation_CreateFcn(hObject, eventdata, handles)
opts.orientation='axial';

function editBrainseg_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

loadFile.m

```
function varargout = loadFile(varargin)

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

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

function loadFile_OpeningFcn(hObject, eventdata, handles, varargin)

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

% Update handles structure
guidata(hObject, handles);

function varargout = loadFile_OutputFcn(hObject, eventdata, handles)

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

function editFilePath_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function btnBrowse_Callback(hObject, eventdata, handles)

global arg;

[fileName path]=uigetfile({'*.img; *.nii'}, 'Choose your acquisition
file');

if ~(fileName == 0)
    arg.img_name=strcat(path, fileName);

    V=spm_vol(arg.img_name);
    vol=spm_read_vols(V);

    % Show cut 100 preview
    s=size(vol);
    axes(handles.axesPreview);
    cor(:,:)=vol(:,:,round(s(3)/2));
    colormap(gray);
```

```
    imagesc(imrotate(cor,90));
    axis off;


end

function axesPreview_CreateFcn(hObject, eventdata, handles)

axis off;
return

function btnOk_Callback(hObject, eventdata, handles)

close 'Load acquisition file';
gui

return


function btnGroupCuts_CreateFcn(hObject, eventdata, handles)

global arg
    arg.axial=1;
    arg.coronal=0;
    arg.sagittal=0;

function btnGroupCuts_SelectionChangeFcn(hObject, eventdata, handles)

global arg

if (hObject == handles.radiobuttonAxial)
    arg.axial=1;
    arg.coronal=0;
    arg.sagittal=0;
elseif (hObject == handles.radiobuttonCoronal)
    arg.axial=0;
    arg.coronal=1;
    arg.sagittal=0;
else
    arg.axial=0;
    arg.coronal=0;
    arg.sagittal=1;
end
```

reorientation.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function reorients coronal or sagittal volume into axial.
% In:
%   - arg: To know if volume is in axial, coronal or sagittal
%          orientation.
%   - file: Absolut path to the file to reorient.
%
% Out:
%   - completePath: Absolut path to the reoriented file
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [completePath] = reorientation(arg, file, pathSave)


  V = spm_vol(file);

  volOriginal=spm_read_vols(V);

  res(1)=abs(V.mat(1,1));
  res(2)=abs(V.mat(2,2));
  res(3)=abs(V.mat(3,3));

  origin(1)=V.mat(1,4);
  origin(2)=V.mat(2,4);
  origin(3)=V.mat(3,4);

  stemp=size(volOriginal);

  if (strcmp(arg, 'axial'))

      volReoriented=volOriginal;

  elseif (strcmp(arg, 'coronal'))
      volReoriented=zeros(stemp(1),stemp(3),stemp(2));

      for i=1:stemp(2)

volReoriented(:,:,i)=imrotate(squeeze(volOriginal(:,i,:)),180);
      end

      tt=res(2);
      res(2)=res(3);
      res(3)=tt;

      tt=origin(2);
      origin(2)=origin(3);
      origin(3)=tt;

  elseif (strcmp(arg, 'sagittal'))

    %Create new matrix to store fixed acquisition in axial orientation
    volReoriented=zeros(stemp(3),stemp(1),stemp(2));

    % Copy axials images in the new matrix
    for i=1:stemp(2)
         volReoriented(:,:,i)=imrotate(squeeze(volOriginal(:,i,:)),
90);
    end

    tt1=res(1);
    tt2=res(2);
    res(1)=res(3);
    res(2)=tt1;
    res(3)=tt2;

    tt1=origin(1);
    tt2=origin(2);
    origin(1)=origin(3);
```

```
    origin(2)=tt1;
    origin(3)=tt2;


  end

  [path,name,ext]=fileparts(file);

  fname=[name,ext];

  V.dim=size(volReoriented);
  V.fname=[pathSave,'\ax_',fname];

  V.mat(1,1)=-res(1);%-res(1);
  V.mat(2,2)=res(2);
  V.mat(3,3)=res(3);

  V.mat(1,4)=abs(origin(1));
  V.mat(2,4)=-abs(origin(2));
  V.mat(3,4)=-abs(origin(3));

  spm_write_vol(V,volReoriented);

  completePath=V.fname;
```

volReformat.m

```
% Interpolation function
function [vol]=volReformat(data,f)

p=find(f==max(f));
p=p(1);

s=round(f.*size(data));
vol=zeros(s);

if(p==1 || p==2)
  for i=1:s(3)
    vol(:,:,i)=squeeze(imresize(data(:,:,i),s(1:2),'bilinear'));
  end
else
  for i=1:s(1)
    vol(i,:,:)=imresize(squeeze(data(i,:,:)),s(2:3),'bilinear');
  end
end
```

resizeVol.m

```
% Create volImage volume.
function[volImage] = resizeVol(vol)

    global offsetx;
    global offsety;
    global offsetz;
    global ox;
    global oy;
    global fx;
```

```
    global fy;
    global maxSize;

    sizeLocal=size(vol);
    minSize=min(sizeLocal);
    maxSize=max(sizeLocal);


    offsetx=round((maxSize-sizeLocal(1))/2);
    offsety=round((maxSize-sizeLocal(2))/2);
    offsetz=round((maxSize-sizeLocal(3))/2);


    % New 3x3 matrix with the maxSize of original Volume.
    volImage=zeros(maxSize,maxSize,maxSize);


    if (offsetx == 0)
        ox=(offsetx+1);
    else
        if (mod(sizeLocal(1), 2) ~= 0)
            ox=offsetx;
        else
            ox=(offsetx+1);
        end
    end


    if (offsety == 0)
        oy=(offsety+1);
    else
        if (mod(sizeLocal(2), 2) ~= 0)
            oy=offsety;
        else
            oy=(offsety+1);
        end
    end


    fx=min((maxSize-offsetx), maxSize);
    fy=min((maxSize-offsety), maxSize);

    for i=1:sizeLocal(3)
        volImage(ox:fx,oy:fy,(i+offsetz))=vol(:,:,i);
    end
```

save_volume.m

```
% This function is called before run any module to store volume as
axial
% orientation or original orientation. That orientation must be chosen
in
% Options menu.
function [nV]=save_volume

global resultspath;
global fname;
```

```
global vol;
global volOriginal;
global arg;
global V;
global res;
global origin;

handles = guihandles(gcf);

nV=V;

% Check if we have to save as axial or as original adquisition type.
if (arg.axial==0 && strcmp(get(handles.save_axial_menu,
'Checked'),'on'))

    % Assume that some image are loaded, because menu is enabled, so
don't
    % check again.

        [path,name,ext]=fileparts(arg.img_name);
        resultspath=[pwd,'\results\',name,'\'];

        fname=[name,ext];
        mkdir(resultspath);

        nV.dim=size(vol);
        nV.fname=[resultspath,'ax_',fname];

        nV.mat(1,1)=-res(1);%-res(1);
        nV.mat(2,2)=res(2);
        nV.mat(3,3)=res(3);

        %nV.mat(1:3,4)=(size(vol).*res)/2;
        %nV.mat(2:3,4)=-V.mat(2:3,4);
        nV.mat(1,4)=abs(origin(1));
        nV.mat(2,4)=-abs(origin(2));
        nV.mat(3,4)=-abs(origin(3));

        spm_write_vol(nV,vol);

else % If save as original is checked

        [path,name,ext]=fileparts(arg.img_name);
        resultspath=[pwd,'\results\',name,'\'];

        fname=[name,ext];
        mkdir(resultspath);

        nV.fname=[resultspath,fname];
        spm_write_vol(nV,volOriginal);
end
```

run_batch.m

```
function run_batch (files, opts)
```

```
% opts
% pause

for i=1:opts.nfiles

    if (opts.nfiles == 1)
        axFile=files;
    else
        axFile=files(i,:);
    end

    if (opts.reorient == 1)
       axFile=reorientation(opts.orientation,axFile,opts.save);
    end

    [path,name,ext]=fileparts(axFile);
    fname=[name, ext];

            % Some messages
            disp('#####################################');
            disp(['# Processing file ', fname]);
            disp(['# File number ', num2str(i), ' of ',
num2str(opts.nfiles), '.']);
            disp('#####################################');



            % Check if user has checked tissue clasificattion
            if (opts.segment == 1)
                disp(' ');
                disp('Applying Segmentation on file ...');
                segment(axFile);

                if (opts.remove == 1)
                    [path,name,ext]=fileparts(axFile);
                    fileName=[name,ext];

                    % Delete intermedied files...
                    delete(axFile);
                    delete([path,'\res_', fileName]);
                    delete([path,'\f_', fileName]);
                    delete([path,'\e_', fileName]);
                    delete([path,'\mask_', fileName]);
                end

                return;
            end

            % Check if user has checked denoising
            if (opts.denoising == 1)
                disp(' ');
                disp('Applying Denoising on file ...');
                denoise(axFile);

                % Update variable path
                [path,name,ext]=fileparts(axFile);
                fname=['f', name, ext];
                axFile=[path, '\', fname];
            end
```

```
            % Check if user has checked inhomogenity correction
            if (opts.correction == 1)
                disp(' ');
                disp('Applying Inhomogenity Correction on file ...');
                NUCorrect(axFile);

                % Update variable path
                [path,name,ext]=fileparts(axFile);
                fname=['m_', name, ext];
                axFile=[path, '\', fname];
            end



            % Check if user has checked brain extraction
            if (opts.extraction == 1)
                disp(' ');
                disp('Applying Brain Extraction on file ...');
                brainextract(axFile);

                % Update variable path
                [path,name,ext]=fileparts(axFile);
                fname=['e_', name, ext];
                axFile=[path, '\', fname];
            end


            % Check if user has checked tissue clasificattion
            if (opts.clasify == 1)
                disp(' ');
                disp('Applying Brain Extraction on file ...');
                clasify(axFile);
            end

end % Loop end

    disp(['All data stored under ', opts.save, ' directory.']);

    warning('ON', 'MATLAB:MKDIR:DirectoryExists'); % Enable again
```