RESTAURACIÓN DE VOLÚMENES DE MICROCOPÍA ELECTRÓNICA DE PARTÍCULAS INDIVIDUALES

Carlos Oscar Sanchez Sorzano
Fernando Fuentes Arija

ÍNDICE

1.-Resumen

Conocer la estructura de las moléculas (virus, proteinas,...) es muy importante para entender su funcionamiento. ¿Porqué microscopia electrónica? Llegados a este punto la EM es necesaria, debido a que los microscopios ópticos no tienen un aumento suficiente por culpa de la longitud de onda de los fotones, que es mucho más grande que la de los electrones.

Las imágenes procedentes de EM tienen asociado mucho ruido, y ahí es donde nos vamos a centrar. Partimos de una imagen 3D de una molécula obtenida mediante un TEM, la cual vamos a procesar utilizando una metodología de mejora del contraste ya existente, basada en aplicar de forma local una función no lineal a la imagen. Esta función se va a definir para los 10 intervalos de grises especificados por la escala de Munsell. Una vez implementado el método en Matlab, lo simulamos y los resultados obtenidos son positivos.

Partiendo de esta metodología y añadiendo ideas nuevas, hemos creado un nuevo método de mejora del contraste pensado para que supere al ya existente. En primer lugar hemos añadido una primera etapa que denominamos “enmascaramiento” y que se encarga de eliminar el ruido que rodea a la molécula. Para hacerlo consideramos que las seis caras de la imagen 3D sólo contienen ruido, y a partir esta muestra de ruido mediante inferencia estadística calculamos un intervalo de confianza, para poder discriminar entre lo que consideramos ruido y lo que no. Esta información se va a guardar en una máscara que vamos a procesar utilizando la operación de cierre (Morfología Matemática). En segundo lugar vamos a mejorar el proceso de Sobel, trabajando en 3D, y añadiendo unas máscaras tridimensionales para derivar no solo en “x” y en “y”, si no también en “z”. Esto nos proporcionará más información sobre el vecindario para calcular mejor los bordes de la imagen. La tercera mejora que aplicamos, es introducir el concepto de vecindario variable para el cálculo del Mean Edge Gray Value. Es decir en vez de procesar utilizando cubos de 3x3x3, los tamaños varían desde éste inicial hasta 9x9x9 como máximo. Para saber que tamaño de cubo debemos escoger para cada píxel, recurrimos de nuevo a la inferencia estadística, calculamos un intervalo de confianza para el cubo de tamaño menor, y si el cubo de tamaño superior entra dentro de ese intervalo de confianza, entonces podemos considerar que esos dos cubos son iguales, y por lo tanto cogemos el de mayor tamaño, ya que nos aporta más información. Finalmente aplicamos la función lineal sin aplicar ninguna mejora en este apartado. Una vez implementado todo esto en Matlab comparamos los resultados con la metodología original, y la nuestra resulta ser algo superior.

Finalmente hemos implementado el programa en C++ utilizando el entorno de desarrollo Eclipse, y hemos añadido una última mejora a la hora de calcular los bordes. Hemos sustituido los operadores de Sobel, por una metodología más compleja, que consiste en interpolar mediante B-splines la función de la molécula para más tarde derivarla y sacar así los bordes.

2.-Abstract

Knowing the structure of biological specimens is very important to understand how they operate. Why electron Microscopy? Reaching this spot EM is necessary because optical microscopes can not zoom enough because of the wavelenth of the photons is much bigger than the one of electrons.

The images obtained in EM have a lot of noise and that is where we are goning to work. We have a 3D image obtained by a TEM, that we are going to process using a method of contrast sharpening that already exists. This method applies locally a nonlinear funtion in each of the ten subintervals described by the Munsell's scale. We have programed this method in Matlab producing successful results.

Starting over this method we are going to add new ideas in order to improve this method and create one better. On firt place we have added a new process that removes the noise surrounding the specimen. We have made this by considering noise pixels that are far from the specimen, and then using statistics we can determine which pixels are likely to be noise and which are likely to belong to the specimen. On second place we are going to improve the Sobel operator working in 3D and using new masks. This will provide us more information about the neighborhood so we can compute better the edges. Finally, we are going to use a variable neighborhood that is going to grow from a size of 3x3x3 up to 9x9x9 using statistics. When we simulate our method in Matlab the results are better than the original method.

In a final stage we have programmed the method in C++ using Eclipse and we have added a final improvement in the compute of the edges removing the Sobel operators and introducing a much better method. We interpolate the function of the specimen using B-splines, and later we derivate to compute the edges.

3.-INTRODUCCIÓN

3.1.-El microscopio electrónico

Un microscopio electrónico emplea electrones en vez de fotones, y esto le permite obtener unos aumentos mucho mayores a los de un microscopio óptico, ya que la longitud de onda de los electrones es mucho menor a la de los fotones pertenecientes al espectro visible. Por ello mientras que los mejores microscopios ópticos pueden aumentar la imagen de objeto hasta 2000 veces, un microscopio electrónico puede hacerlo 1000000 veces. Debido a que los microscopios electrónicos, no utilizan luz, las imágenes obtenidas son en blanco y negro.

Principalmente existen dos tipos de microscopio electrónico, el microscopio electrónico de transmisión y el microscopio electrónico de barrido:

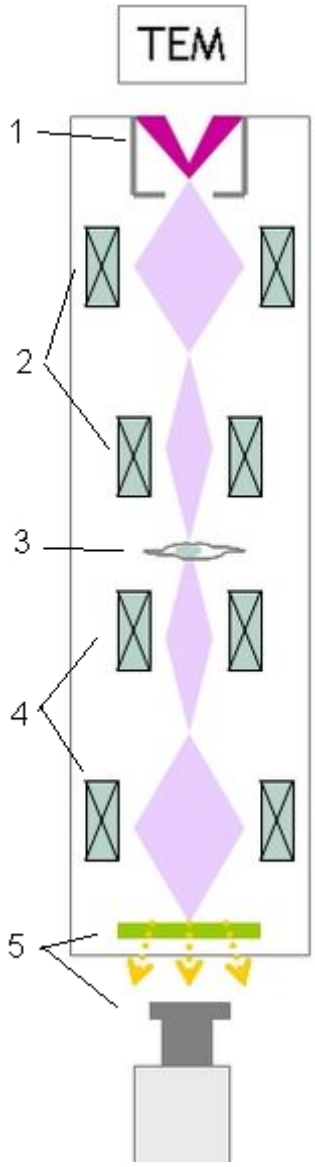


Figura (3.1)

El **microscopio electrónico de barrido** (TEM) funciona del siguiente modo: en primer lugar, hay un *cañón electrónico* (número 1 en la figura) que emite unos electrones, que son acelerados por un alto voltaje, en segundo lugar, nos encontramos unas *lentes magnéticas* (número 2 en la figura) cuya misión es crear campos que dirigen y enfocan el haz de electrones. En tercer lugar, nos encontramos con *la muestra* (número 3 de la figura), la cual debe ser cortada en finas capas cuyo grosor no puede superar los 2000 angstroms. En este punto, los electrones rebotan, son absorbidos o atraviesan la muestra, formando así la imagen. Después de la muestra nos encontramos unas *lentes magnéticas* (número 4 de la imagen) cuya misión es amplificar la imagen, que será recogida por una *placa fotográfica* o *una pantalla fluorescente* (número 5 de la figura) sensible al impacto de los electrones. Por lo general esta imagen es recogida por un computador y mostrada en una pantalla. Y todo este proceso, se realiza en el vacío, ya que el aire absorbe los electrones. Por lo tanto para conseguir un flujo ininterrumpido de electrones, se trabaja con bajas presiones de entre 10^{-4} y 10^{-8} KPa, esto se hace por dos razones: primero para permitir que haya una alta diferencia de voltaje, pero sin que se produzca un arco voltaico y segundo para reducir la frecuencia de las colisiones de los electrones con los átomos del aire a niveles despreciables. El tipo de imagen con el que vamos a trabajar ha sido obtenida con un microscopio de este tipo.

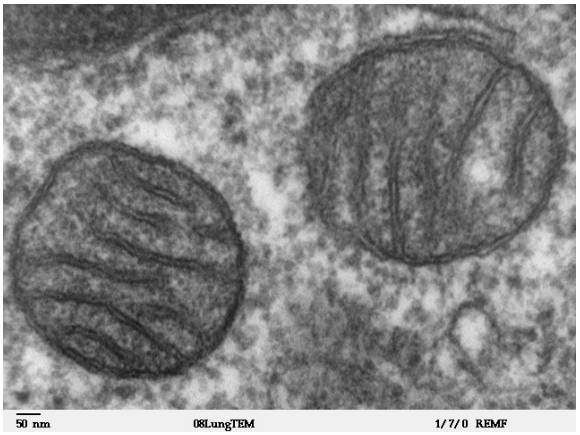


Figura (3.2)

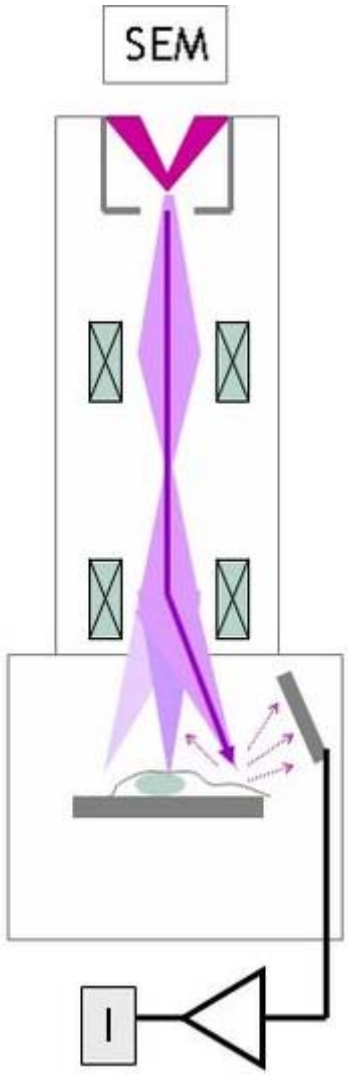


Figura (3.3)

El **microscopio electrónico de barrido** (SEM), se caracteriza por tener una gran profundidad de campo, lo que permite enfocar a la vez gran parte de la muestra. La preparación de la muestra, en este caso es más sencilla, ya que solo se requiere que sea conductora. Por lo general la muestra es recubierta de carbón o de una fina capa de oro, para proporcionarle a la muestra propiedades conductoras. La muestra es barrida con los electrones acelerados que viajan a través del cañón y luego un detector mide la cantidad de electrones enviados que arroja la intensidad de la zona de muestra, siendo capaz de mostrar figuras en tres dimensiones. Este tipo de microscopio permite una resolución de entre 3 y 20 nm. Al igual que en el microscopio MET se aceleran los electrones en un campo eléctrico, para aprovechar de esta manera su comportamiento ondulatorio, lo cual se lleva a cabo en la columna del microscopio, donde se aceleran por una diferencia de potencial de 1000 a 30000 voltios. Los electrones acelerados mediante voltajes pequeños se utilizan cuando las muestras son sensibles, como puede ser el caso de muestras biológicas. Mientras que los electrones acelerados por altos voltajes, se utilizan para muestras metálicas, ya que estas no suelen sufrir daños. Los electrones acelerados que salen del cañón son enfocados por las lentes condensadora y objetiva y al incidir estos en la muestra, esta vez observamos los electrones que rebotan, en vez de los que la atraviesan.

Los microscopios SEM son ampliamente utilizados en la biología celular, y aunque permiten una menor capacidad de aumento que el microscopio electrónico de transmisión, gracias a su capacidad para ver las imágenes en tres dimensiones permiten apreciar con mayor facilidad las texturas.

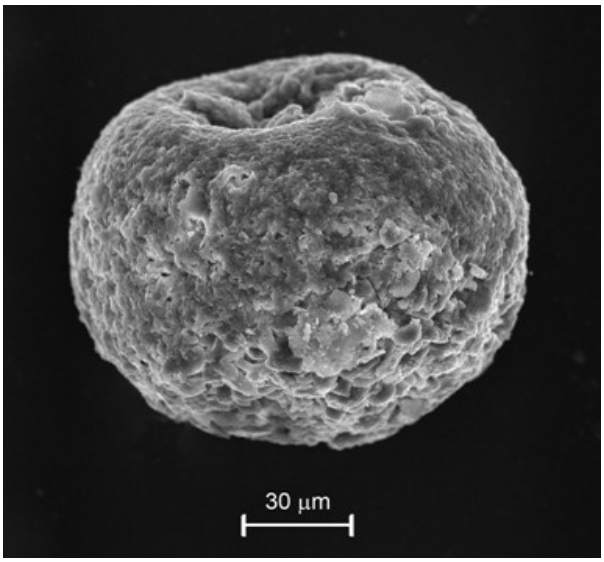


Figura (3.4)

3.2.-Procesamiento de imágenes y reconstrucción 3D en microscopia electrónica

Conocer las estructuras de los especimenes biológicos es crítico para entender sus funciones a todas las escalas y es crucial en las biociencias para complementar los estudios bioquímicos. La microscopia electrónica nos permite investigar estructuras de especimenes de multitud de dimensiones, desde estructuras celulares hasta macromoléculas y todo ello con una resolución baja, media, alta e incluso atómica.

■ Formación de la imagen en el microscopio electrónico

Como se ha explicado anteriormente, el funcionamiento de un microscopio electrónico de transmisión es muy similar al del microscopio óptico. La fuente de iluminación es un filamento (cátodo) que emite electrones desde lo alto de una columna cilíndrica de unos 2 metros. Luego los electrones son acelerados por un ánodo formando un haz que viaja por el vacío hasta atravesar la muestra. Luego los electrones que atraviesan la muestra son recogidos por las lentes magnéticas, y enfocados para formar un patrón que constituye la imagen. Las imágenes obtenidas por un microscopio electrónico de transmisión (TEM) pueden considerarse como proyecciones 2D de la muestra. Mediante la inclinación de la muestra en el TEM podemos obtener diferentes vistas, que luego serán necesarias para el proceso de reconstrucción tomográfica.

■ Preservación de las muestras bajo condiciones fisiológicas

Las muestras han de ser especialmente preparadas antes de exponerlas a los electrones, ya que las condiciones de vacío del TEM y la radiación de electrones pueden degradar las estructuras biológicas. Principalmente nos encontramos con dos técnicas para preparar las muestras, la tinción negativa y la criomicroscopía.

La tinción negativa consiste en cubrir la muestra con algún tipo de contraste que proteja el material biológico de los electrones y a la vez mejore el contraste de las imágenes obtenidas. Esta técnica solo nos proporciona información estructural sobre la superficie de la muestra, además el agente de tinción produce artefactos en las imágenes, limitándonos a una resolución media de unos 20 Å. Por estas razones la tinción negativa no se usa para estudios de alta resolución, no obstante es muy útil en las primeras etapas de la investigación donde lo que nos interesa es echar un pequeño vistazo para tener una primera idea de como es la muestra.

Actualmente la criomicroscopía es la técnica más común para preparar las muestras biológicas de los TEM. Esta técnica mantiene la muestra congelada en una capa de hielo a temperaturas criogénicas, lo que asegura la preservación de la muestra durante la exposición a los electrones. Las imágenes obtenidas mediante la criomicroscopía tienen información sobre la estructura de la muestra, y no solo de su superficie. En un principio esta técnica no implica ninguna limitación en cuanto a resolución.

El material biológico es muy sensible a la radiación, por lo tanto las dosis de electrones deberán mantenerse muy bajas, entre 5 y 20 electrones por $Å^2$.

■ Principios de la reconstrucción tomográfica

Los TEM proporcionan proyecciones 2D que contienen características estructurales de las diferentes capas de la estructura tridimensional de la muestra que han sido superpuestas en la dirección del haz de electrones. La estructura 3D de la muestra, se puede calcular a partir de un conjunto de imágenes obtenidas al observar la muestra desde distintos ángulos. Estas imágenes son obtenidas o bien cambiando la inclinación de la muestra dentro del TEM, o cogiendo imágenes de distintas muestras del mismo espécimen.

Los principios matemáticos de la construcción tomográfica están basados en el teorema de la sección central. Este teorema establece que la transformada de Fourier (FT) de una proyección 2D de un objeto 3D es la sección central de la transformada de Fourier del objeto 3D. (Figura 3.5)

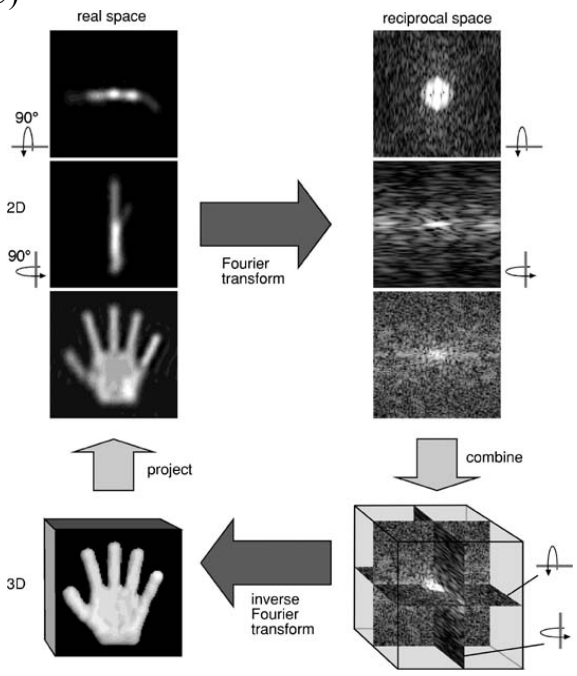


Figura (3.5)

Por lo tanto mediante la recolección de distintas proyecciones del espécimen desde distintos ángulos, se puede muestrear su transformada de Fourier 3D para más tarde calcular su estructura mediante una simple inversa de la FT. La limitada capacidad que tienen los TEM para inclinar las muestras, hace imposible obtener todas las posibles vistas de un espécimen y como consecuencia puede que haya regiones incompletas en el espacio de Fourier. Esta falta de información implica que el volumen reconstruido tiene resolución anisotrópica.

■ Procesamiento de imagen y reconstrucción 3D en microscopia electrónica

El problema general en la determinación de estructuras mediante microscopia electrónica, es la reconstrucción 3D de una estructura biológica a partir de un conjunto finito de proyecciones 2D de la misma. Desde el punto de vista de procesamiento de imágenes hay gran número de problemas como: la extremadamente baja señal a ruido

(SNR) de las imágenes, la determinación de la CTF (función de transferencia de contraste) y corrección de sus efectos, alineamiento y clasificación de imágenes, etc...

■ Aproximaciones a la determinación de estructuras en microscopia electrónica

Los distintos enfoques que se usan en la recopilación de datos y en las reconstrucciones 3D dependen de la naturaleza del espécimen y de la información estructural que se busque. Para los especimenes que se encuentren el dominio macromolecular se han definido unas metodologías específicas, como por ejemplo, para los arrays cristalinos de proteínas en 2D se emplea una técnica conocida como **EC** (electron crystallography) y para especimenes aislados, tanto asimétricos, como altamente simétricos se emplea la aproximación "**single particles**". Para especimenes mayores se utiliza el método **ET** (electron tomography).

Gracias a estas estrategias podemos sacar información estructural de los diferentes especimenes empleando la microscopia electrónica y obteniendo resoluciones de 40 - 100 Å para la ET, 6-30 Å para "single particles", e incluso resolución atómica para EC.

■ Single Particles

Para reconstruir macromoléculas biológicas en alta resolución mediante EM, se necesita un gran número de proyecciones para cubrir todo el espacio 3D de Fourier y una dosis muy baja de electrones. Hasta mediados de los 80 estos requerimientos solo eran posibles cuando el espécimen era un cristal 2D o era una partícula simétrica. En estos casos la formación de una imagen con bajas dosis de electrones es posible gracias a que se puede mejorar la SNR promediando la estructura repetida o usando las propiedades simétricas de la partícula. Estos dos métodos a día de hoy siguen siendo los que proporcionan una mejor calidad de reconstrucción, pero por desgracia muchas de las macromoléculas biológicas interesantes, ni son simétricas ni se pueden cristalizar.
La única forma de recoger la suficiente cantidad de información para hacer una reconstrucción 3D de una molécula no cristalizable y no simétrica utilizando bajas dosis de electrones, es combinar imágenes de un gran número de partículas. La clave para realizar una buena reconstrucción 3D, a partir de la información es identificar bien las proyecciones de las partículas en las imágenes y determinar con precisión su orientación. La máxima resolución que se puede alcanzar oscila entre los 6 y los 10 Å. Este procedimiento se divide en 6 partes: adquisición de la imagen, corrección de la CTF, análisis 2D, clasificación de la imagen, asignación angular y reconstrucción 3D.

1. Adquisición de la imagen

Las imágenes se obtienen de dos modos: o bien se digitaliza la imagen grabada en una película, o con un sensor CCD introducido en el TEM. La función de transferencia de la modulación (MTF) de cualquier dispositivo suele ser insignificante en comparación con las inexactitudes introducidas en la preparación de la muestra, la alta cantidad de ruido o las aberraciones del TEM. Por otro lado las condiciones de iluminación pueden variar mucho a lo largo de las distintas imágenes o incluso dentro de una misma imagen, generando variaciones en los niveles de gris que deberán ser corregidas más adelante. Otros preprocesamientos necesarios incluyen por ejemplo el denoising y el downsampling.

2. Corrección de la CTF

Como ya se ha explicado antes, todas las imágenes se ven afectadas por la CTF del microscopio electrónico, por lo tanto la corrección de la CTF es un tema principal en cualquier estudio estructural de alta resolución. Para corregir sus efectos, primero hay que estimar la CTF. Por lo general esto se hace en dos pasos, primero se calcula la densidad espectral de potencia (PSD) y luego se ajusta un modelo teórico de CTF a esa PSD.

3. Análisis 2D

En las primeras fases se analizan proyecciones 2D desde direcciones similares, para familiarizarse con las características principales de la estructura de la molécula. Este análisis ayuda a construir modelos, que posteriormente se utilizarán para hacer la clasificación de las imágenes, y para la asignación angular. Las herramientas principales usadas en esta etapa son: promediado de la imagen y descomposición de la imagen en sus armónicos de Fourier.

4. Clasificación de imágenes

Antes de empezar la reconstrucción 3D hay que asegurarse de que todas las imágenes proyectadas pertenecen a la misma estructura de la macromolécula, si no fuera así combinar las proyecciones de diferentes objetos en un mismo volumen produciría una versión borrosa. Por lo tanto las imágenes proyectadas se clasifican en grupos homogéneos utilizando técnicas clásicas de reconocimiento de patrones y de clustering.

5. Asignación angular

Una vez que tenemos una población de imágenes homogénea, hay que encontrar sus orientaciones relativas tridimensionales. Esto se puede hacer por ejemplo comparando imágenes experimentales con proyecciones simuladas por ordenador de modelos (ya conocidos) que se parezcan a la molécula bajo estudio. La dirección de proyección que mejor concuerde con el modelo simulado es la que se asigna. Esta búsqueda se puede realizar en varios espacios, como por ejemplo en la FT, ya que debido al teorema de la sección central (explicado anteriormente), sabemos que cualquier pareja de proyecciones comparte una línea común.

6. Reconstrucción 3D

Una vez que tenemos las orientaciones relativas del conjunto de imágenes, podemos pasar a la reconstrucción 3D, y para hacer esto existen un gran número de algoritmos, de entre los cuales los más usados son “weigted black-projection” y el algoritmo de expansión en series. El algoritmo “weigted black-projection”, está basado en el anteriormente comentado teorema de la sección central y en la inversión del volumen reconstruido en el espacio de Fourier.

3.3.-Objetivos, plan de trabajo y criterios de evaluación

1. Objetivos

Se va a partir de una metodología 2D no lineal y basada en la escala de Munsell para la mejora del contraste de una imagen, y se va a replantear el método, para mejorarlo y poderlo emplear en imágenes 3D de microscopía.

2. Plan de trabajo

Primero vamos a simular en Matlab una técnica actual de mejora del contraste sobre una imagen de prueba, luego mediante la lectura de diversos artículos sobre tratamiento digital de imágenes para sacar ideas y el empleo de técnicas estadísticas vamos a proponer una nueva técnica para la mejora del contraste de imágenes. Esta nueva técnica se simulará implementándola en Matlab para poder compararla de modo rápido con la antigua técnica y comprobar que efectivamente la nueva metodología es superior. Una vez se hayan realizado las dos simulaciones con la imagen de prueba, se procederá ha realizar una versión final en C++ utilizando unas librerías concretas para poder ejecutar el programa sobre un CLUSTER, debido a que el procesamiento de una imagen a tamaño normal supone tantos millones de operaciones, que no se podría realizar en un PC normal.

3. Criterios de evaluación

Mediante la técnica “Fourier Shell Correlation”, vamos a comparar la imagen ideal con la imagen defectuosa y con la procesada mediante nuestra función. De tal modo que nuestra imagen procesada debe estar más correlada con la ideal que la errónea.

4.-DESARROLLO Y RESULTADOS

Primero se va a escoger una metodología utilizada para la mejora del contraste en imágenes, y posteriormente se va a exponer el nuevo método diseñado, para al final comparar los resultados de ambos y comprobar que el nuevo método es mejor.
Tanto la nueva como la antigua metodología se van a implementar en Matlab, una vez hecho esto vamos a realizar nuestro método en C, y añadir unas mejoras para poder correr el programa sobre un ordenador multicore que utiliza unas librerías determinadas.

4.1.-METODOLOGÍA ACTUAL

Consideramos la aproximación del problema: "mejora no lineal del contraste de una imagen basado en la escala de Munsell". [6]

4.1.1.-Introducción

El contraste es una medida de la variación de la intensidad o del valor de gris en una región específica de una imagen. Esta región puede ser una pequeña ventana, en cuyo caso se estaría hablando de un concepto de contraste localizado, o por otro lado la región podría ocupar toda o casi toda la imagen y estaríamos ante un concepto global de contraste. Típicamente cuando se hace mejora de contraste uno se fija en la diferencia de los valores de gris en una pequeña ventana y no en toda la imagen.

Existen dos tipos básicos de técnicas para la mejora del contraste, las indirectas y las directas. En las indirectas el contraste es mejorado sin el cálculo ni la utilización de una medida cuantitativa de contraste, sin embargo en los métodos directos sí se calcula y usa una medida cuantitativa del contraste.
Esta aproximación utiliza ideas de ambos métodos, ya que se calcula una medida de contraste como en el método directo, pero a diferencia de este no utilizamos esta medida para mejorar directamente el contraste.

El cálculo de la medida local de contraste en el vecindario de un píxel localizado en las coordenadas (i,j) se hace del siguiente modo:

$$C_{ij} = \frac{\left| x_{ij} - \overline{E}_{ij} \right|}{x_{ij} + \overline{E}_{ij}} \qquad (1)$$

Donde x_{ij} es el valor de gris asociado al píxel de coordenadas (i,j) y $\overline{E}_{ij}$ es el "mean edge gray value" que se definirá con rigurosidad posteriormente, aunque podemos adelantar que intuitivamente es una media ponderada con los bordes de la imagen en el entorno de (i,j). Para esta metodología, se van a usar vecindarios de 3x3 píxeles para calcular C_{ij}. Un vecindario de mayor tamaño haría que el "mean edge gray value" y la medida de contraste fueran menos sensibles al ruido, pero también tendríamos una pérdida de los pequeños detalles.

El "mean edge gray value" se define en un vecindario N_{ij} del píxel en (i,j) como:

$$\overline{E}_{ij} = \frac{\sum_{(k,l)\in N_{ij}} \Delta_{kl} \cdot x_{kl}}{\sum_{(k,l)\in N_{ij}} \Delta_{kl}} \qquad (2)$$

Donde Δ_{kl} es el valor del borde, calculado mediante los operadores de Sobel asociados con el píxel en (k,l) y x_{kl} es el valor de gris del píxel en (k,l)
La ecuación (2) establece el "mean edge gray value" como la media del valores de gris ponderada por los valores del borde.

4.1.2.-Operadores Sobel

En primer lugar se va a definir como bordes de una imagen digital, las transiciones entre dos regiones de niveles de gris cuya diferencia sea significativa. Los bordes proporcionan mucha información, nos sirven para reconocer objetos o las fronteras entre los mismos.

Los detectores de bordes suelen basarse en el cálculo de un operador local de derivación, como es en el caso de Sobel, vamos a poner un ejemplo sencillo para entender qué ocurre con esta operación. Si nos fijamos en la Figura 1, vemos dos imágenes, una franja negra sobre fondo blanco, y una franja blanca sobre fondo negro. Si prestamos atención nos damos cuenta de que se obtiene una primera derivada positiva cuando pasamos de un nivel de gris a otro más claro y una negativa en caso contrario. De este modo usando la derivada podemos saber dónde hay un borde.

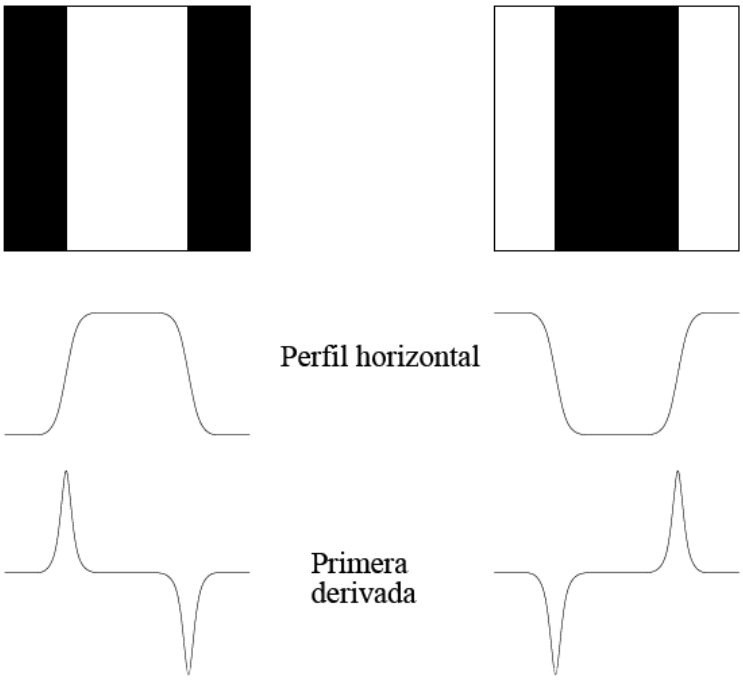


Figura 1

Como se ha comentado, el valor de la magnitud de la primera derivada nos sirve para detectar la presencia de bordes. La primera derivada en cualquier punto de la imagen vendrá dada por la magnitud del gradiente.

Z1	Z2	Z3
Z4	Z5	Z6
Z7	Z8	Z9

Figura 2: imagen I de 3x3 píxeles

El gradiente de una imagen I (Figura 2) en las coordenadas (x,y) se calcula como:

$$\nabla I = \begin{bmatrix} I_X \\ I_Y \end{bmatrix} = \begin{bmatrix} \partial I / \partial x \\ \partial I / \partial y \end{bmatrix} \qquad (3)$$

El vector gradiente siempre apunta en la dirección de la máxima variación de la imagen I en el punto (x,y).
Lo que nos interesa en la detección de bordes, es la magnitud de este vector, que se denomina simplemente como gradiente de la imagen, y se calcula:

$$\nabla I = \|\nabla I\| = \sqrt{I_X^2 + I_Y^2} \qquad (4)$$

Para agilizar la computación, la ecuación (4) se suele aproximar como:

$$\nabla I = \|\nabla I\| \approx |I_X| + |I_Y| \qquad (5)$$

Las máscaras de (operadores de) Sobel son las siguientes:

−1	0	1
−2	0	2
−1	0	1

(a)

−1	−2	−1
0	0	0
1	2	1

(b)

Figura (3): Operadores de Sobel

La máscara de la izquierda, Figura (3-a), es el gradiente en “x” y detecta los cambios horizontales, y la de la derecha, Figura (3-b), es el gradiente en “y”, y detecta los cambios verticales. Las derivadas según los operadores de la Figura (3) vienen dadas por las siguientes ecuaciones:

$$I_X = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \qquad (6)$$
$$I_Y = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

La dirección del vector gradiente puede ser calculada como:

$$\alpha(x,y) = \tan^{-1}\left(\frac{I_Y(x,y)}{I_X(x,y)}\right) \qquad (7)$$

Debajo de estas líneas en la Figura (4), vamos a mostrar un ejemplo en el que se ven 4 imágenes, la (a) corresponde a la imagen original en escala de grises, la (b) es la imagen procesada por los operadores de Sobel, la (c) es la resultante de calcular el gradiente en x, y la (d) la de calcular el gradiente en y.

(a) (b)

(c) (d)

Figura (4): ejemplo

Mirando la Figura (4-c) y (4-d), si nos fijamos en el cuadro y la puerta (por ejemplo) podemos corroborar de forma clara como el gradiente en x, ha detectado las líneas horizontales y como el gradiente en y ha detectado las verticales, tal y como se explicó anteriormente.

4.1.3.-Escala de Munsell

Este sistema fue creado por Henry Munsell en un intento de crear una "forma racional" de describir los colores. Este sistema consta de tres elementos claves, la "matriz", el "valor" y la "intensidad", cada color puede ser descrito mediante estos tres elementos. Cada uno de estos elementos está descrito por una escala, la de la "matriz" va de 0 a 100, el "valor" de 0 a 10, y la intensidad tiene la escala de saturación de color.
La "matriz" es la capacidad de distinguir entre los distintos colores. El "valor" es una escala que va del blanco al negro, pasando por los grises, estos son denominados colores neutrales, y no tienen matriz ya que no son cromáticos. La "intensidad" es usada para graduar el color.

4.1.4.-Funcionamiento del método

De los tres elementos con los que Munsell describe los colores, nosotros sólo vamos a quedarnos con el "valor", ya que trabajamos sobre una imagen en escala de grises. Munsell para determinar el "valor", escogió 10 trozos de gris, empezando en el negro (Valor = 0) y siguiendo con tonos cada vez más claros hasta acabar en el blanco (valor = 10). Definió la reflexión como, el total de la radiación reflejada de la luz que incide sobre una superficie, y asignó a cada "valor" (0-10) un valor de reflexión. Estableció la siguiente relación entre los dos parámetros:

$$V = 10\sqrt{R} \qquad (8)$$

Un estudio posterior estableció la relación entre el parámetro "valor" y la reflexión de forma más exacta como: ($R_{MgO} = 0{,}97$)

$$R = (1{,}2219 \cdot V - 0{,}23111 \cdot V^2 + 0{,}23951 \cdot V^3 - 0{,}021009 \cdot V^4 + 0{,}0008404 \cdot V^5)\frac{R_{MgO}}{100} \qquad (9)$$

Sustituyendo valores obtenemos la siguiente tabla:

Value	Reflectance Value	
	from Equation (8)	from Equation (9)
0	0	0
1	0.01	0.012
2	0.04	0.030
3	0.09	0.064
4	0.16	0.116
5	0.25	0.192
6	0.36	0.292
7	0.49	0.418
8	0.64	0.573
9	0.81	0.763
10	1.00	0.995

Figura (5)

La intensidad del píxel, viene dada por I=RL donde I es la intensidad, R la reflexión y L la iluminación. Entonces, si se asume constante el nivel de iluminación, la medida de reflexión se puede mapear en valores de intensidad entre 0 y 255.
A partir de los valores de la reflexión calculados mediante la ecuación (9) que podemos ver en la Figura (5), dividimos el intervalo [0,255] en 10 sub-intervalos: [0,3], (3,8], (8,16], (16,30], (30,49], (49,75], (75,107], (107,147], (147,196], (196,255].

El método de mejora de contraste coge un valor de gris (de un píxel) y lo lleva a uno de los umbrales, de su sub-intervalo correspondiente, que es determinado mediante el "mean edge gray value" asociado a dicho píxel.

$$f(x_{i,j}) = \begin{cases} a, & x_{i,j} \le \overline{E}_{ij} \\ b, & x_{i,j} > \overline{E}_{ij}, o\ x_{i,j} = \overline{E}_{ij} = b \end{cases} \qquad (10)$$

Lo que hace la ecuación (10), es coger el valor de gris de un píxel, y mirar el valor del "mean edge gray value" que tiene asociado, si el valor de gris es menor, lo iguala al umbral inferior de su intervalo [a,b], en este caso [a]. Si el valor de gris es mayor que su valor "mean edge gray value" asociado, entonces igualamos el valor de gris al umbral superior de su intervalo, es decir [b].

Conseguimos la máxima mejora de contraste, cuando el valor de gris mejorado resulta estar a la mayor distancia posible del "mean edge gray value". Esto ocurre con la ecuación (10), que sería una posible función de mejora de contraste, pero no podemos usarla por un problema que veremos más adelante, por lo tanto ahora debemos centrarnos en la elección de una ecuación de mejora de contraste adecuada.

■ Selección de una buena función de mejora del contraste

El principal problema de la ecuación (10) es que reducimos el número de valores de gris de 255 a 11, y esto provoca una pérdida de información y detalle en la imagen de salida que causa bordes dentados.

Lo que se necesita es una función que se aproxime de forma suave a la ecuación (10) y que pase por los puntos (a,a), (E,E) y (b,b), donde (a) es el punto más bajo del intervalo [a,b], E es el "mean edge gray value" asociado a un píxel dado y (b) es el punto más alto del intervalo [a,b]. Y además la función tendrá que tener pendiente cero en los puntos finales del intervalo. En cada intervalo tendríamos una función con un aspecto aproximado al de la Figura (5).

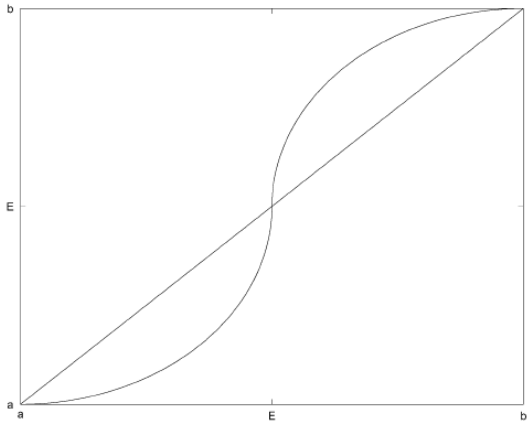


Figura (5)

Estos requerimientos que acabamos de exigir hacen que la función tenga una forma que nos proporcione una buena mejora del contraste para aquellos valores de gris que estén a cierta distancia del "mean edge gray value". Además también asegura que los sucesivos intervalos se unan de manera suave. Como consecuencia de la conexión entre cada una de las funciones de mejora de contraste, definidas para cada intervalo, tenemos una aproximación suave a la función escalera definida en la ecuación (10).

Ahora la mejor función, en el sentido de proveer el mayor contraste posible, tiene que tener pendiente infinita en E (mean edge gray value). Las funciones con pendiente finita en E, no consiguen mejorar tanto el contraste como las que tienen pendiente infinita en E.

Para explicar el motivo de esto, vamos a suponer que tenemos una función f1 con pendiente infinita en E y definida en [E,b]. Entonces f1 empieza en E con pendiente infinita, y esa pendiente irá decayendo (f1 es convexa) hasta alcanzar pendiente cero en b. Para un mismo punto, cualquier otra función con pendiente finita en E, tendrá un valor menor que f1, y esto es debido al hecho de que f1 tiene pendiente infinita en x=E, por lo tanto los valores se incrementan rápidamente en el vecindario de E.

Para encontrar la función óptima de mejora del contraste debemos buscar una función suave monotónica f(x) tal que:

$$\left| f(x) - E \right| \tag{11}$$

sea mayor que todas las funciones suaves y monotónicas posibles definidas en el intervalo [E,b]. Esto significa que para todo el resto de funciones suaves y monotónicas g(x) se debe cumplir:

$$\left| f(x) - E \right| \geq \left| g(x) - E \right| \tag{12}$$

Se supone que buscamos entre todas las funciones suaves y monotónicas, la función y(x) que maximice la integral:

$$\int_E^b \ln\left(\frac{y}{\sqrt{1+y'^2}} \right) dx \tag{13}$$

sujeto a la restricción:

$$\int_E^b y(x)dx \geq \int_E^b r(x)dx \tag{14}$$

para todas las funciones suaves y monotónicas r(x). Como y(x) y r(x) son monotónicas, esto implica dos cosas:

$$y(x) \geq r(x) \tag{15}$$

$$\left| f(x) - E \right| \geq \left| r(x) - E \right| \tag{16}$$

Maximizar la integral (13), es el equivalente a maximizar:

$$\int_E^b \ln(y)dx - \int_E^b \ln(\sqrt{1+y'^2})dx \tag{17}$$

Que a su vez es equivalente a maximizar:

$$\int_E^b \ln(y)dx \tag{18}$$

sujeto a la restricción de que

$$\int_E^b \ln\left(\sqrt{1+y'^2}\right)dx \tag{19}$$

es de longitud fija L' y sujeto a la restricción de los extremos. Y como la función logaritmo neperiano es monotónica, esto es equivalente a maximizar:

$$\int_E^b y\, dx \tag{20}$$

sujeto a la condición de que

$$\int_E^b \sqrt{1+y'^2}\,dx \tag{21}$$

es de longitud fija L y sujeto a la restricción de los extremos. Por lo tanto la función óptima de mejora del contraste es aquella que describe una curva de longitud fija y área máxima sobre un intervalo limitado por un punto final a la izquierda y otro a la derecha. Buscamos "y" tal que:

$$f^* = y + \lambda\sqrt{1+y'^2} \tag{22}$$

sea maximizado. La función y(x) tendrá la propiedad de

$$\int_E^b y(x)dx \geq \int_E^b r(x)dx \tag{23}$$

para todas las funciones suaves y monotónicas r(x). Como y(x) y r(x) son monotónicas, eso implica:

$$y(x) \geq r(x) \tag{24}$$

$$\left|f(x)-E\right| \geq \left|r(x)-E\right| \tag{25}$$

En este caso la función y(x), es la solución a un clásico problema isoperimétrico (explicado en 4.1.5) y se puede expresar como:

$$(x-c_1)^2 + (y-c_2)^2 = \lambda^2 \tag{26}$$

donde c_1, c_2 y λ son constantes reales. Ahora la restricción del extremo de la izquierda obliga a la función a pasar por el punto (E,E), además de que la pendiente de la función en este punto sea infinita, y la restricción del punto derecho obliga a la función a pasar por el punto (b,b). Si aplicamos estos puntos fijos como restricciones (x=E y x=b) en la ecuación (26), obtenemos las siguientes ecuaciones:

$$\begin{aligned}(E-c_1)^2 + (E-c_2)^2 &= \lambda^2 \\ (b-c_1)^2 + (b-c_2)^2 &= \lambda^2\end{aligned} \tag{27}$$

Si resolvemos este sistema para c_2 en función de c_1 nos queda:

$$c_2 = b + E - c_1 \tag{28}$$

Si derivamos respecto de x la ecuación (26), obtenemos la pendiente:

$$\frac{\partial y}{\partial x} = \frac{-(x-c_1)}{y-c_2} \tag{29}$$

Como se sabe que la pendiente es infinita para x=E, de la ecuación anterior obtenemos que $c_2 = E$, por lo tanto de la ecuación (28) sacamos que $c_1 = b$. Con las constantes determinadas sabemos que el centro del círculo es (E,b). De la ecuación (29) sacamos que la pendiente para x=b es cero, por lo tanto la función tiene una tangente horizontal en x=b. En consecuencia la función óptima en [E,b] es un cuarto de círculo.

De forma similar, para el intervalo [a,E] buscamos una y(x), pero esta vez los las restricciones de los extremos están invertidas. Es decir el extremo derecho exige que la función pase por el punto (E,E) y además se exige que en x=E la función tenga pendiente infinita. Por otro lado en endpoint izquierdo requiere que la función pase por el punto (a,a).

Repitiendo el proceso anterior, análogamente a la ecuación (28), obtenemos:

$$c_2 = a + E - c_1 \tag{30}$$

Basándose de nuevo en la función tiene pendiente infinita en x=E deducimos $c_1 = 1$ y $c_2 = E$. De la ecuación (29) observamos que para x=a la función tiene pendiente cero. Del mismo modo que para el caso anterior nos encontramos con una tangente horizontal en x=a. Por lo tanto la función óptima en el intervalo [a,E], es un cuarto de círculo.

Basándonos en todo lo anterior, escogemos una función de mejora de contraste, que consiste en dos cuartos de círculo, uno definido en [a,E] y otro definido en [E,b]. Estos dos cuartos, se unen de forma continua y suave en el punto [E,E]. Además de que f(x) tiene pendiente infinita en x=E, cumple una serie de propiedades para cualquier valor de gris x:

$$\begin{aligned} &Para \quad x \le E \rightarrow f(x) < x \\ &Para \quad x > E \rightarrow f(x) > x \end{aligned} \tag{31}$$

Estas propiedades son ciertas para todo x perteneciente al intervalo [a,b], excepto para x=a,E,b que son puntos fijos de f(x).

Para determinar la ecuación del cuarto de círculo definido en el intervalo [a,E], sabemos que su centro está localizado en el punto (a,E), por lo tanto su radio es E-a. Sustituimos es radio en la ecuación (26) del círculo:

$$(x-a)^2 + (y-E)^2 = (E-a)^2 \tag{32}$$

Despejamos de la ecuación (32) y obtenemos la función:

$$f_1(x) = E - \sqrt{(E-a)^2 - (x-a)^2} \tag{33}$$

Si dibujamos $f_1(x)$ obtenemos el siguiente cuarto de círculo:

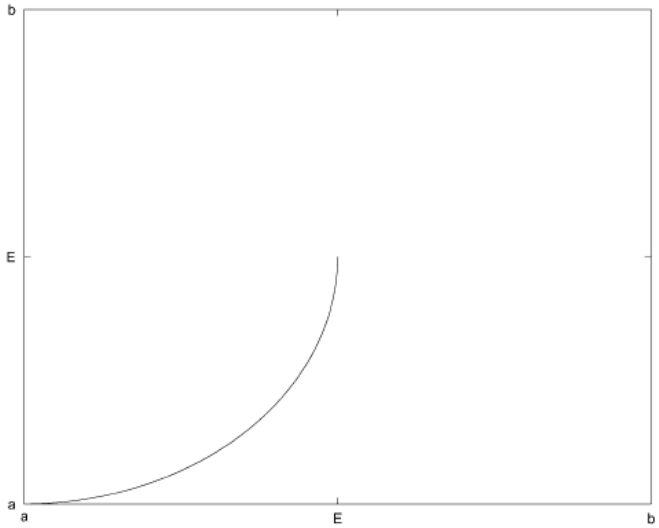


Figura (6)

De forma análoga, para determinar el cuarto de círculo definido en [E,b] , partimos de su centro en el punto (b,E) para determinar que su radio es b-E, y sustituyendo de nuevo sobre la ecuación (26):

$$(x-b)^2+(y-E)^2=(b-E)^2 \tag{34}$$

Despejamos de la ecuación (32) y obtenemos la función:

$$f_2(x)=E-\sqrt{(b-E)^2-(x-b)^2} \tag{35}$$

Si dibujamos $f_2(x)$ obtenemos el siguiente cuarto de círculo:

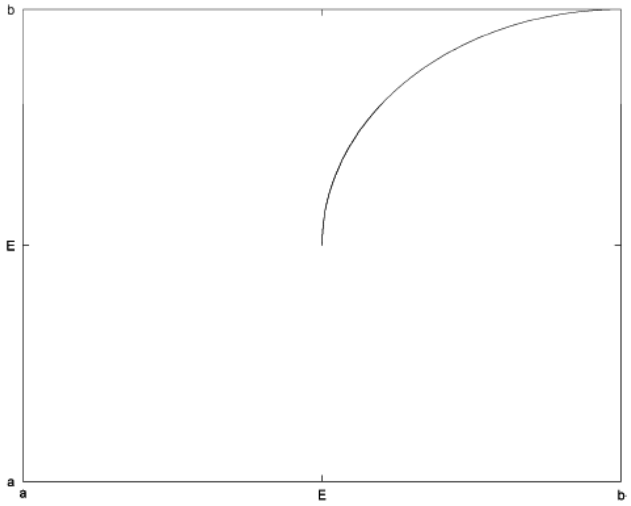


Figura (7)

Por lo tanto, la función de mejora de contraste óptima es:

$$f(x)=\begin{cases}E-\sqrt{(E-a)^2-(x-a)^2}, & x\le E\\ E-\sqrt{(b-E)^2-(x-b)^2} & x>E\end{cases} \tag{36}$$

Cuya función dibujada es la que ya adelantábamos en la Figura (5):

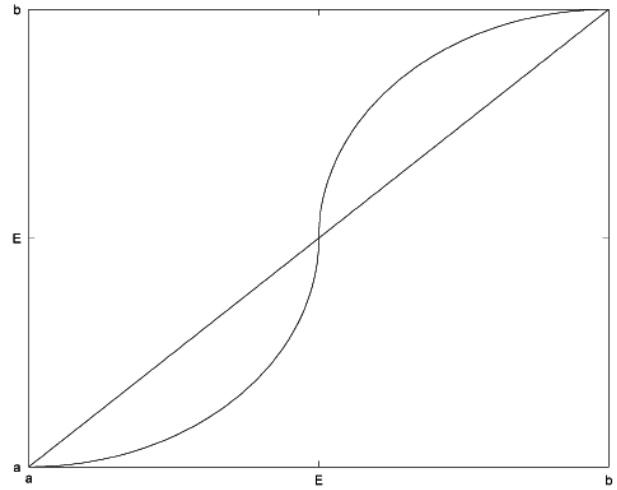


Figura (8)

Es posible que el "mean edge gray value" $\overline{E}_{ij}$ quede fuera del intervalo [a,b] que contiene al valor de gris x_{ij} . En cualquier caso el valor de gris mejorado $f(x_{ij})$ caerá dentro del intervalo [a,b].

$$\left.\begin{array}{l}\overline{E}_{ij} < a \\ x_{ij} \in [a,b]\end{array}\right\} f(x_{ij}) > x_{ij} \ \ para\ x_{ij} > \overline{E}_{ij} \rightarrow f(x_{ij}) > x_{ij} > \overline{E}_{ij}$$

$$\left.\begin{array}{l}\overline{E}_{ij} > b \\ x_{ij} \in [a,b]\end{array}\right\} f(x_{ij}) < x_{ij} \ \ para\ x_{ij} < \overline{E}_{ij} \rightarrow f(x_{ij}) < x_{ij} < \overline{E}_{ij}$$

■**Demostración:** Ahora vamos a demostrar que hemos mejorado el contraste de la imagen partiendo de la definición que dimos de contraste en la ecuación (1). Para la función de mejora del contraste f(x), la mejora de contraste viene dada por:

$$C'_{ij} = \frac{\left| f(x_{ij}) - \overline{E}_{ij} \right|}{f(x_{ij}) + \overline{E}_{ij}} \qquad (37)$$

Si $x_{ij} \le \overline{E}_{ij}$ sabemos que $f(x_{ij}) < x_{ij}$ por lo tanto:

$$\left| f(x_{ij}) - \overline{E}_{ij} \right| \ge \left| x_{ij} - \overline{E}_{ij} \right| \qquad (38)$$

Para demostrar que el contraste es mayor después de procesar la imagen con este método, hay que llegar a que la ecuación (37) es mayor que la ecuación (1):

$$\frac{\left| f(x_{ij}) - \overline{E}_{ij} \right|}{f(x_{ij}) + \overline{E}_{ij}} \ge \frac{\left| x_{ij} - \overline{E}_{ij} \right|}{x_{ij} + \overline{E}_{ij}} \qquad (39)$$

Partiendo de $f(x_{ij}) < x_{ij}$:

$$\frac{1}{f(x_{ij}) + \overline{E}_{ij}} > \frac{1}{x_{ij} + \overline{E}_{ij}}$$

$$\frac{f(x_{ij}) - \overline{E}_{ij}}{f(x_{ij}) + \overline{E}_{ij}} > \frac{\left| f(x_{ij}) - \overline{E}_{ij} \right|}{x_{ij} + \overline{E}_{ij}}$$

$$\frac{\left| f(x_{ij}) - \overline{E}_{ij} \right|}{x_{ij} + \overline{E}_{ij}} > \frac{\left| x_{ij} - \overline{E}_{ij} \right|}{x_{ij} + \overline{E}_{ij}} \quad \textit{por ecuación (38)}$$

Por lo tanto:

$$\frac{\left| f(x_{ij}) - \overline{E}_{ij} \right|}{f(x_{ij}) + \overline{E}_{ij}} \ge \frac{\left| x_{ij} - \overline{E}_{ij} \right|}{x_{ij} + \overline{E}_{ij}}$$

De modo que queda demostrado que $C'_{ij} \ge C_{ij}$, o lo que es lo mismo, el contraste de la imagen procesada es mayor que el de la imagen inicial sin procesar. Para $x_{ij} > \overline{E}_{ij}$ también se cumple la ecuación (38), de modo que la demostración es análoga.

4.1.5.-Problema Isoperimétrico

El problema isoperimétrico es el siguiente: "De entre todas las curvas planas cerradas que tienen una longitud fija, determinar cuál es la que encierra una mayor área en su interior". Tal y como lo describe la propia palabra isoperimétrico (isos=igual, perímetro=longitud del contorno).

Los antiguos griegos, ya se lo plantearon, e intuían que la solución era la circunferencia, pero no fueron capaces de demostrarlo. Este problema ha sido uno de los más importantes e influyentes de la historia de las matemáticas y de echo no se consiguió resolver hasta la segunda mitad del siglo XIX, cuando el matemático alemán Karl Weierstrass proporcionó una demostración completa en sus clases de la universidad de Berlín.

Ejemplo de problema Isoperimétrico: Sean dos puntos A=(a,0) y B=(b,0) en el eje x donde la distancia entre ellos está dada, es decir AB=l. El problema de hallar una curva que maximice el área entre ella y el eje x será:
Hallar la función f(x) de modo que:

$$I[f] = \int_a^b f(x)\,dx = máx \qquad (40)$$

con la restricción:

$$G[f] = \int_a^b \sqrt{1 + f'(x)^2}\,dx = l \qquad (longitud\ de\ arco) \qquad (41)$$

Solución: El problema se puede resolver partiendo de la desigualdad que relaciona el perímetro con el área de una curva cerrada en un plano.

$$4\pi A \le P^2 \qquad (42)$$

Siendo "P" el perímetro de la curva y "A" el área de la región encerrada por la misma. Ahora nos damos cuenta, que para el caso de un círculo de radio "r" , tenemos $A=\pi \cdot r^2$ y $P = 2\pi \cdot r$, que al introducirlos en la desigualdad esta se maximiza:

$$4\pi A \le P^2$$
$$4\pi \cdot \pi \cdot r^2 \le (2\pi \cdot r)^2$$
$$(2\pi \cdot r)^2 = (2\pi \cdot r)^2$$

La siguiente tabla muestra lo demostrado:

Polígono	Núm de lados	Perímetro	Área
Triángulo	3	1 m	0,0481 m²
Cuadrado	4	1 m	0,0625 m²
Hexágono	6	1 m	0,0721 m²
Octógono	8	1 m	0,0754 m²
Dodecágono	12	1 m	0,0777 m²
Isodecágono	20	1 m	0,0789 m²
Circunferencia	∞	1 m	0,0795 m²

Figura (9)

4.1.6.-Implementación y resultados

En una primera fase vamos a implementar los programas en Matlab para simular su funcionamiento sobre una imagen volumétrica de prueba de 64x64x64 píxeles y poder comprobar que nuestra mejora del contraste es superior.
La estructura del programa es la siguiente:

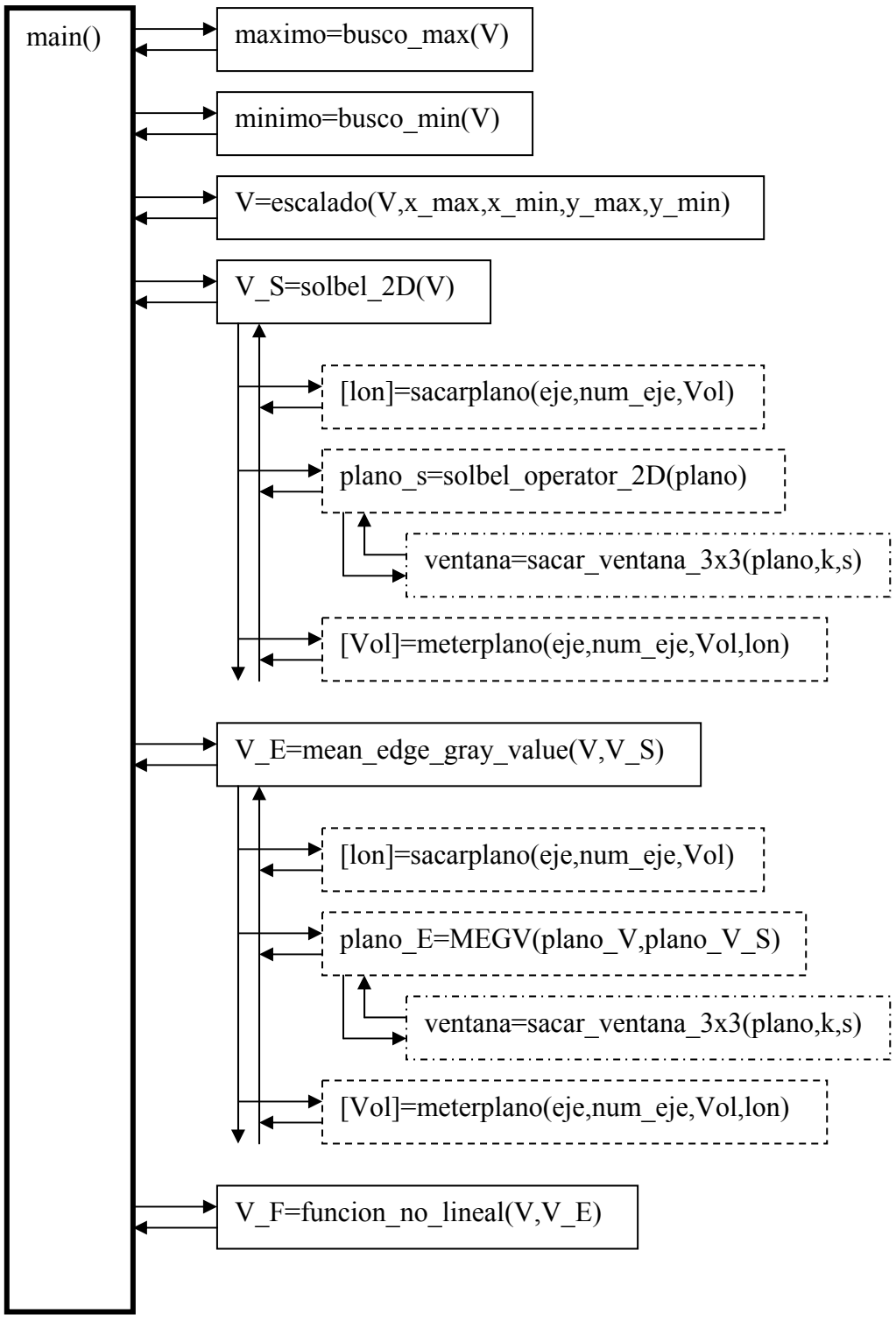


Figura (10)

Observando el archivo principal **main.m** *(ver código en el Apéndice A.1)* lo primero que hacemos es abrir la molécula (molecula64.vol) utilizando la función open_volume que nos devuelve una matriz de 64x64x64. El primer proceso que vamos a realizar es un escalado de los valores de la matriz entre 0 y 255. Para ello primero llamamos a las funciones **busco_max.m** *(ver código en el Apéndice A.2)* y **busco_min.m** *(ver código en el Apéndice A.3)* que nos van a devolver el valor más alto y más bajo de la matriz respectivamente. Luego procedemos a escalar la imagen usando la función **V=escalado(V,x_max,x_min,y_max,y_min)** *(ver código en el Apéndice A.4)* que recibe una matriz "V" con su valor máximo "x_max" y su valor mínimo "x-min", y devuelve la misma matriz "V", pero esta vez con todos los valores escalados entre "y_min" y "y_max", que en este caso son 0 y 255. (que son los niveles de gris) A la matriz de 64x64x64, la vamos a llamar volumen. Como mostrar el volumen completo sería excesivo, ya que habría que imprimir 64 planos, por simplicidad solo vamos a mostrar (por ejemplo) el plano (x,y,35) que se encuentra en el medio del volumen. El resultado que nos proporciona la función de escalado es el siguiente:

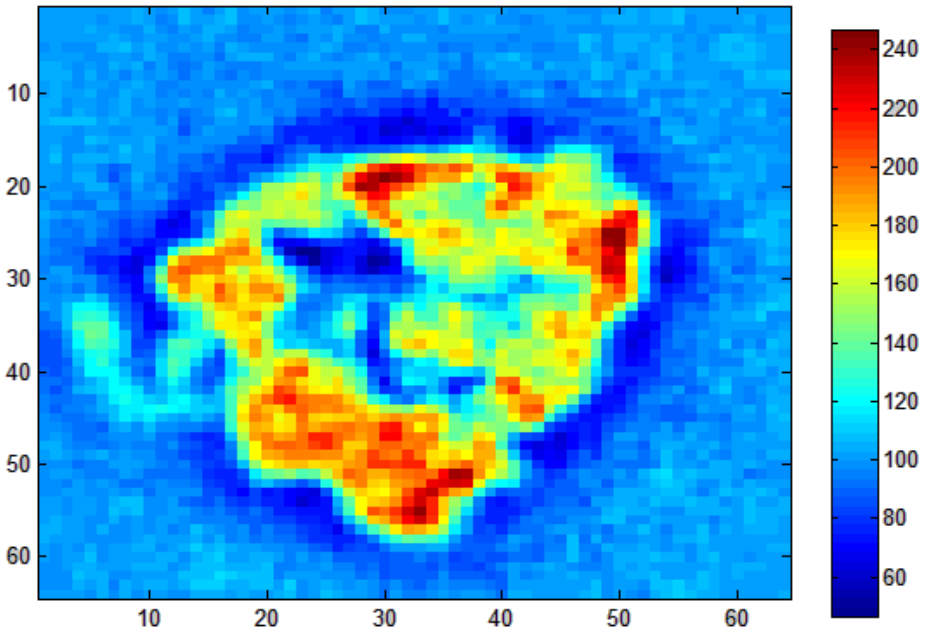


Figura (11): Plano (x,y,35) de V

Una vez que tenemos todo el volumen escalado entre 0 y 255, ahora podemos trabajar sobre él. El procesamiento que tenemos que hacer lo vamos a dividir en 3 fases fundamentales, una primera en la que aplicamos los operadores de sobel, para sacar los bordes de la imagen (volumétrica) , luego una segunda fase, donde utilizamos los bordes calculados anteriormente, para hacer una media ponderada con los mismos (mean edge gray value) y finalmente una última fase donde vamos a aplicar la función óptima de mejora del contraste que dedujimos en el apartado 4.1.4.

La primera fase la realiza la función **V_S=solbel_2D(V)** *(ver código en el Apéndice A.5)*, que en primer lugar hace una llamada a la función **[lon]=sacar plano (eje, num_eje,Vol)** *(ver código en el Apéndice A.6)* , a la cuál para un mismo "eje", en este caso el eje z, vamos sacando todos los planos "num_eje" del volumen "Vol". Una vez que hemos sacado un plano del volumen lo procesamos mediante la función **plano_s=sobel_operator_2D(plano)** *(ver código en el Apéndice A.7)* que aplica a un

plano "plano" las máscaras de Figura (3) explicadas en el apartado 4.1.2, devolviendo el resultado en el "plano_s". Como operamos usando ventanas 3x3 píxeles, hacemos uso de la función **ventana=sacar_ventana_3x3(plano,k,s)** *(ver código en el Apéndice A.8)*.
Una vez que tenemos el plano procesado por los operadores de Sobel, lo introducimos en el volumen "V_S", volumen procesado mediante Sobel, mediante la llamada a la función **[Vol]=meterplano(eje,num_eje,Vol,lon)** *(ver código en el Apéndice A.9)*.
Este procedimiento se repite para cada plano, desde z=1 (eje=3,num_eje=1) hasta z=64 (eje=3,num_eje=64) y el resultado es el siguiente:

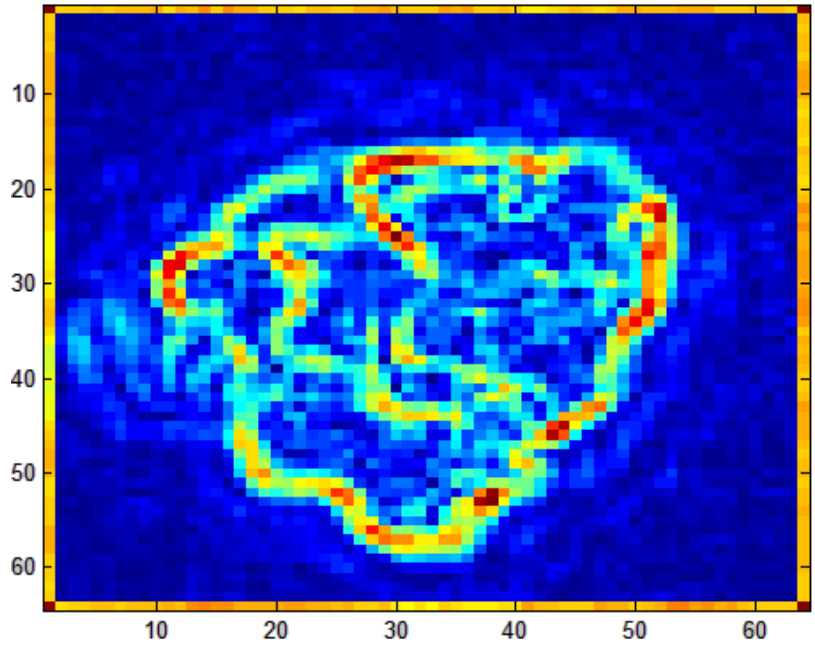


Figura (12): Plano (x,y,35) de V_S

La segunda fase es realizada por la función **V_E=mean_edge_gray_value(V,V_S)** *(ver código en el Apéndice A.10)* , la cuál opera del mismo modo comentado anteriormente, saca cada uno de los 64 planos que forman la molécula, y los procesa de forma individual mediante la función **plano_E=MEGV(plano_V,plano_V_S)** *(ver código en el Apéndice A.11)*. Esta función implementa la siguiente ecuación:

$$\overline{E}_{ij} = \frac{\sum_{(k,l)\in N_{ij}} \Delta_{kl} \cdot x_{kl}}{\sum_{(k,l)\in N_{ij}} \Delta_{kl}} \qquad (43)$$

donde N_{ij} es un vecindario de 3x3 píxeles que obtenemos mediante la función **ventana =sacar_ventana_3x3(plano,k,s)** anteriormente mencionada. Como resultado de ejecutar esta segunda fase, obtenemos un volumen V_E, que contiene todos los valores del "mean edge gray value" de cada píxel del volumen. A continuación mostramos un ejemplo del resutado:

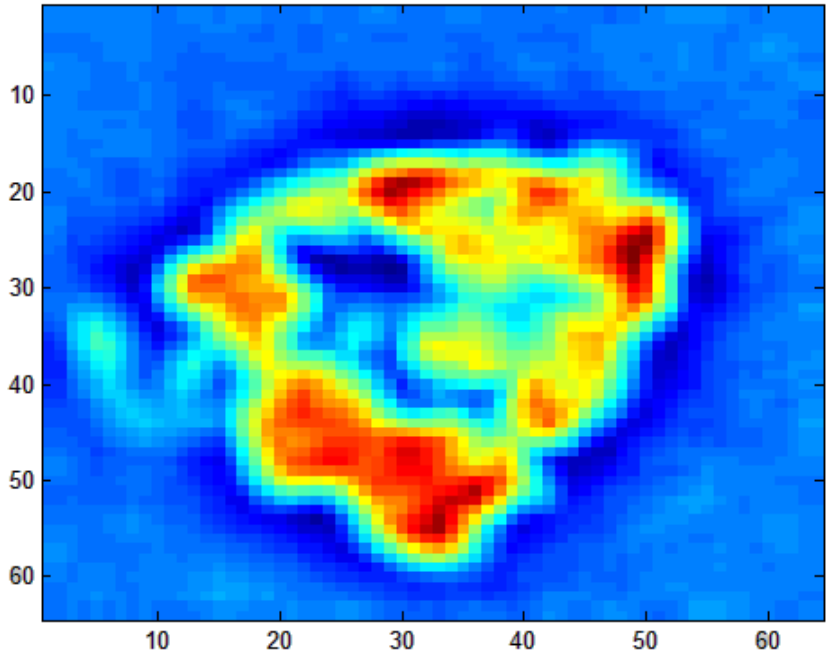


Figura (13): Plano (x,y,35) de V_E

Como podemos ver, la Figura (13) varía de forma más suave, ya que en el fondo es una media de los valores de la molécula ponderada por los bordes de la misma.

La última fase es llevada a cabo por la función **V_F=funcion_no_lineal(V,V_E)** *(ver código en el Apéndice A.12)* , la cual recibe el volumen "V" con todos los píxeles de la imagen tridimensional de la molécula y un volumen "V_E" con los valores del "mean edge gray value" asociados a cada píxel de "V". Con todos estos datos se implementa la función (36) y el resultado es un volumen V_F:

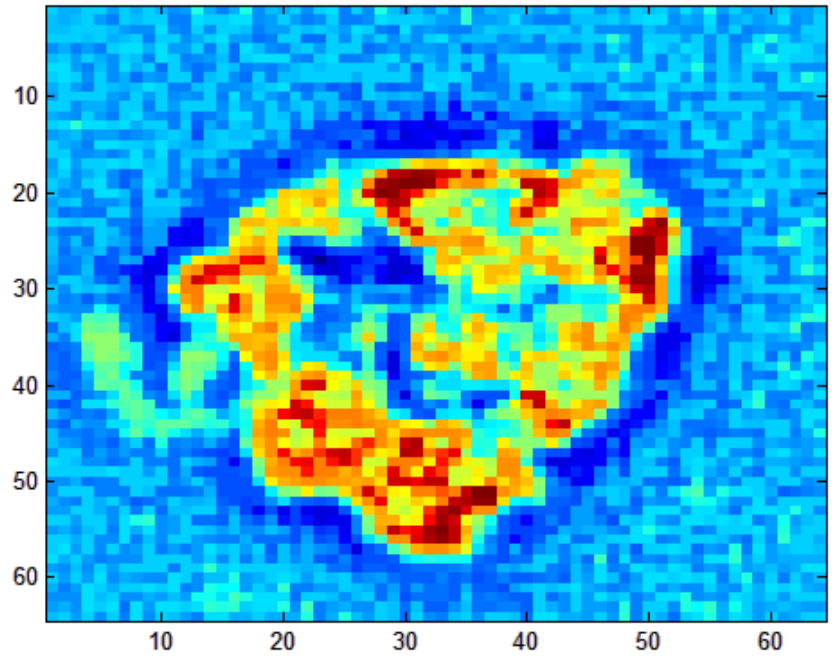


Figura (14): Plano (x,y,35) de V_F

4.1.7.-Conclusiones

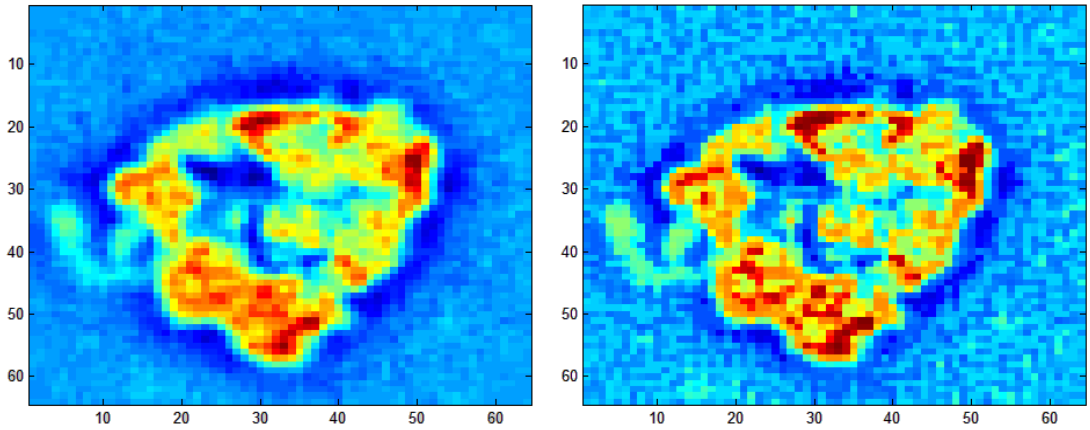


Figura (15)

Si nos fijamos en la Figura (15), a la izquierda tenemos un plano de la molécula sin procesar, y a la derecha el mismo plano pero procesado, a simple vista, podemos observar lo que demostramos en el apartado 4.1.4:

$$\frac{\left| f(x_{ij}) - \overline{E}_{ij} \right|}{f(x_{ij}) + \overline{E}_{ij}} \geq \frac{\left| x_{ij} - \overline{E}_{ij} \right|}{x_{ij} + \overline{E}_{ij}} \qquad (44)$$

el contraste de la imagen procesada es superior al de la imagen inicial, $C'_{ij} \geq C_{ij}$. Como contrapartida también hemos aumentado el contraste del ruido que rodea a la molécula. (los tonos azules)

4.2.-NUEVA METODOLOGÍA

Ahora se va a exponer la solución mejorada que se propone. Una vez explicada se procederá a ver los resultados, y a compararlos con los de la metodología actual, para ver si la nueva es mejor.

4.2.1.-Introducción

La metodología actual, como hemos visto anteriormente, se resumía en tres fases, una operación de Sobel, un cálculo del los "mean edge gray value" y finalmente la aplicación de una función no lineal. Para esta nueva metodología se va a proponer una cuarta fase y se van a replantear de diferente manera las fases de Sobel y cálculo del "mean edge gray value".

4.2.2.-Fase 1: Enmascaramiento

Por lo tanto esta nueva metodología consta de cuatro fases, la primera fase es un "denoising", en ella pretendemos eliminar el **ruido** que rodea a la molécula (los valores en tono azul) generado por el pegamento que se utiliza para mantenerla estática y por los cristales que se crean al congelar la misma. A este ruido lo llamaremos *background*.

El primer problema que se nos plantea es cómo detectar este ruido, para ello vamos a emplear un método estadístico. ¿Por qué usar método estadístico? Porque un método estadístico sirve para: "sacar conclusiones de un grupo de *individuos* a partir de la información que nos proporciona un subconjunto más o menos amplio de los mismos". En nuestro caso los individuos son los píxeles que forman el ruido que rodea a la molécula, de los cuales queremos sacar información, para definir qué es ruido y qué no lo es. Y para ello vamos emplear la inferencia estadística.

4.2.2.1.-Inferencia Estadística

La inferencia estadística "es aquella rama de la Estadística mediante la cual se trata de sacar conclusiones de una población en estudio, a partir de la información que proporciona una muestra representativa de la misma". También se conoce como Inferencia Inductiva, ya que sigue el método inductivo de razonamiento, es decir partiendo de unos hechos particulares se pretende llegar a una ley general.

La inferencia exacta resulta imposible ya que partimos de la muestra, obtenida por observación o experimentación, y ésta dispone sólo de información parcial. Sin embargo, podemos realizar inferencias inexactas, y medir el rango de error que cometemos.

Ahora vamos a explicar dos conceptos importantes, el primero es el de *población*, que es el conjunto de elementos sobre los que deseamos obtener la información, la población ha de estar perfectamente definida a la hora de comenzar el estudio. El segundo concepto importante, es la *muestra*, que es el subconjunto que se extrae de la población, este subconjunto se ha de escoger de una forma específica, para que la representatividad de la muestra quede garantizada.

Cada población se encuentra asociada a una variable aleatoria que denotaremos con X.

Una **variable aleatoria** X, es una función real definida en el espacio muestral asociado a un experimento aleatorio, Ω.

$$X : \Omega \rightarrow \Re$$

Cuando se realizan experimentos aleatorios resulta necesario cuantificar los resultados, de modo que se establece una relación funcional entre los elementos del espacio muestral (asociado al experimento) y los números reales.

Por lo general vamos a suponer que la población es infinita, o muy grande. Junto con la variable aleatoria asociada al la población, vamos a suponer un modelo de distribución de probabilidad que resuma sus características.

La **distribución de probabilidad** de una variable aleatoria X es una función que le asigna a cada suceso, su probabilidad de que ocurra.
Dada una variable aleatoria X, su función de distribución, Fx(x), es

$$F_X(x) = P(X \leq x)$$

Propiedades:
-Es una función continua por la derecha.
-Es una función monótona no decreciente.
-Se cumple:

$$\lim_{x \to -\infty} F(x) = 0$$

$$\lim_{x \to \infty} F(x) = 1$$

Llamamos función **densidad de probabilidad** de una variable aleatoria, $f(x)$, que describe la densidad de probabilidad en cada punto del espacio:

$$P[a \leq X \leq b] = \int_a^b f(x)dx$$

Siendo F(x) la función distribución tenemos:

$$F(x) = \int_{-\infty}^{x} f(u)du \qquad \text{o} \qquad f(x) = \frac{\partial}{\partial x} F(x)$$

Propiedades:
- $f(x) \geq 0$ para todo x
-El área total encerrada bajo la curva es igual a 1:

$$\int_{-\infty}^{\infty} f(x)dx = 1$$

- $\Pr(a \leq X \leq b) = \int_a^b f(x)dx = F(b) - F(a)$

La inferencia estadística puede dividirse en dos tipos según el conocimiento que se tenga sobre la distribución en la población:
1.-Inferencia Paramétrica: Se conoce la forma de la distribución pero se desconocen los parámetros. La inferencia se realiza sobre los parámetros desconocidos.
2.-Inferencia no Paramétrica: En este caso tanto los parámetros como la forma son desconocidos.

Otra clasificación atiende a la forma en que se estudian los parámetros:
1.-Estimación: Se intenta dar estimaciones de los parámetros desconocidos sin hacer hipótesis previas sobre posibles valores de los mismos. Aquí se engloba la estimación puntual, donde se busca dar un único valor por parámetro, y la estimación por intervalos, donde se busca dar un intervalo de valores probables.
2.-Contraste de Hipótesis: Se realizan hipótesis sobre los parámetros desconocidos, para más tarde comprobar su verosimilitud.

■ Estadísticos y Distribuciones:

Tenemos una población X, y tomando una muestra al azar, obtenemos un valor x_1. Los posibles valores de x_1 son todos los de X, por lo tanto x_1 puede considerarse como una realización particular de una variable X_1 con la misma distribución que X. Lo mismo ocurriría si volvemos a obtener de forma independiente un valor x_2, al igual que antes, podríamos considerarlo como una observación de una variable aleatoria X_2 que tuviese la misma distribución que X. Ahora continuamos este proceso n veces para obtener una muestra de tamaño n, de modo que tenemos n observaciones, $x_1, x_2, \ldots, x_n$ de n variables aleatorias $X_1, X_2, \ldots, X_n$ independientes y con la misma distribución.
"Sea X una variable aleatoria con f.d.p. f, y sean $X_1, X_2, \ldots, X_n$, n variables aleatorias independientes con la misma f.d.p. f que X. Se dice que $X_1, X_2, \ldots, X_n$, es una muestra aleatoria de tamaño n de f o bien n observaciones independientes de X".
Una vez que tenemos la muestra, ésta puede ser descrita usando sus características fundamentales, como la media o la desviación típica. A estas características las denominamos estadísticos. De manera más formal, se dice que "un estadístico es una función de los valores muestrales que no depende de ningún parámetro poblacional desconocido".
Como un estadístico, es una función de variables aleatorias, resulta que el estadístico es en sí otra variable aleatoria. Por ejemplo, la media muestral:

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{45}$$

es una variable aleatoria de la que tenemos una única observación:

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{46}$$

■ Distribuciones muestrales de la media y la desviación típica:

Sea $X_1, X_2, \ldots, X_n$, una muestra aleatoria de una población X en la que:

$$E(X) = \mu \qquad Var(X) = \sigma^2 \tag{47}$$

Si tenemos en cuenta que la esperanza de la suma de varias variables aleatorias es la suma de las esperanzas y que la varianza es la suma de las varianzas, y que si multiplicamos una variable por una constante la varianza queda multiplicada por la constante al cuadrado, entonces el valor esperado y la varianza del estadístico "media muestral" son:

$$E(\overline{X}) = E(\frac{1}{n}\sum_{i=1}^{n} X_i) = \frac{1}{n}\sum_{i=1}^{n} E(X_i) = \frac{1}{n} n\mu = \mu \quad (48)$$

$$Var(\overline{X}) = Var(\sum_{i=1}^{n} \frac{X_i}{n}) = \sum_{i=1}^{n} \frac{1}{n^2} Var(X_i) = n\frac{\sigma^2}{n^2} = \frac{\sigma^2}{n} \quad (49)$$

Si la población tiene una distribución normal, entonces la media muestral se distribuye con:

$$X \equiv N(\mu, \sigma^2) \rightarrow \overline{X} \equiv N(\mu, \frac{\sigma^2}{n}) \quad (50)$$

■ Estimación por intervalos

Definición: "un intervalo de confianza al $(1-\alpha)$ para la estimación de un parámetro poblacional θ que sigue una determinada distribución de probabilidad, es una expresión del tipo $[\theta_1, \theta_2]$ tal que $P[\theta_1 \leq \theta \leq \theta_2] = 1-\alpha$, donde P es la función distribución de probabilidad de θ".

Cuando queremos estimar un parámetro llamamos intervalo de confianza a un par de números entre los cuales estimamos que se encuentra el valor de ese parámetro con una determinada probabilidad de acierto. Estos dos números determinan un intervalo que se calcula a través de una muestra.
-La probabilidad de éxito está representada por $(1-\alpha)$ y se denomina nivel de confianza.
-Y α representa la posibilidad de fallar la estimación y se denomina nivel de significación.
Como es obvio, un intervalo más amplio tendrá más posibilidades de acierto, y uno más corto tendrá una estimación más precisa. El nivel de confianza suele tener de valores del 95% o 99%.

■ Intervalo de confianza para la media de una población normal de varianza conocida

Supongamos, por ejemplo, que disponemos de una variable aleatoria con función de distribución normal $N(\mu, \sigma^2)$ con σ conocida. Ahora obtenemos una muestra de tamaño n, y queremos estimar la media de la población, μ.
El estimador de la misma es la media muestral cuya distribución es conocida:

$$\overline{X} \equiv N(\mu, \frac{\sigma^2}{n}) \quad (51)$$

La cantidad

$$Z = \frac{\overline{X} - \mu}{\frac{\sigma}{\sqrt{n}}} \quad (52)$$

tendrá distribución estándar. Como ahora la distribución es N(0,1) los puntos $X_{-\alpha/2}$ y $X_{\alpha/2}$ son simétricos de modo que se cumple:

$$P[X_{-\alpha/2} \leq Z \leq X_{\alpha/2}] = 1-\alpha \quad (53)$$

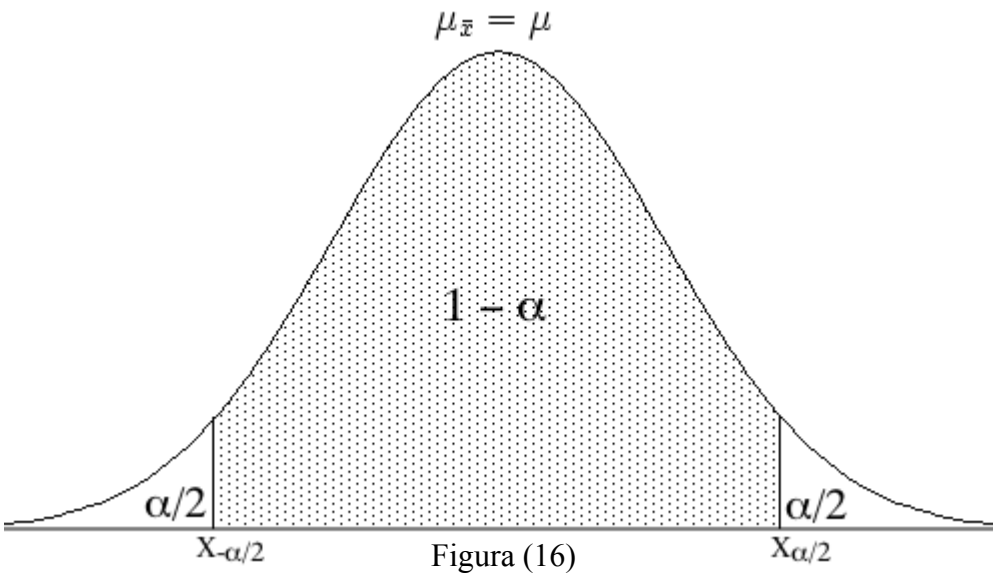


Figura (16)

Si sustituimos la ecuación (52) en la (53), obtenemos lo siguiente:

$$P[X_{-\alpha/2} \leq \frac{\overline{X} - \mu}{\frac{\sigma}{\sqrt{n}}} \leq X_{\alpha/2}] = 1 - \alpha \qquad (54)$$

Y si ahora despejamos la varianza y la media muestral:

$$P[\overline{X} - X_{\alpha/2}\frac{\sigma}{\sqrt{n}} \leq \mu \leq \overline{X} + X_{\alpha/2}\frac{\sigma}{\sqrt{n}}] = 1 - \alpha \qquad (55)$$

Viendo la ecuación (55), el intervalo de confianza es el siguiente:

$$I_{\mu}^{1-\alpha} = [\overline{X} - X_{\alpha/2}\frac{\sigma}{\sqrt{n}}\,;\, \overline{X} + X_{\alpha/2}\frac{\sigma}{\sqrt{n}}] = [\overline{X} \pm X_{\alpha/2}\frac{\sigma}{\sqrt{n}}] \qquad (56)$$

En realidad de todos los posibles valores de $\overline{X}$, nosotros tenemos un único valor obtenido de nuestra muestra $\bar{x}$.

Por lo tanto el intervalo queda:

$$I_{\mu}^{1-\alpha} = [\bar{x} \pm X_{\alpha/2}\frac{\sigma}{\sqrt{n}}] \qquad (57)$$

■Intervalo de confianza para la media de una población normal de varianza desconocida

Esta es la situación más común, es decir aquella en la que desconocemos el valor de la varianza y por lo tanto tendremos que estimar a partir de la muestra obtenida. Por sus buenas propiedades utilizaremos la cuasi-varianza muestral como estimador. La distribución muestral asociada a la cuasi-varianza es:

$$\frac{(n-1)\hat{S}^2}{\sigma^2} \equiv \chi^2_{n-1} \qquad (58)$$

Siendo:

$$\hat{S} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - x)^2$$

Si se combina la distribución normal asociada a las medias con la χ^2 obtenemos una distribución t de Student:

$$t = \frac{Z}{\sqrt{\frac{\chi^2_{n-1}}{n-1}}} = \frac{\frac{\overline{X}-\mu}{\sigma/\sqrt{n}}}{\sqrt{\frac{\frac{(n-1)\hat{S}^2}{\sigma^2}}{n-1}}} = \frac{\overline{X}-\mu}{\frac{\hat{S}}{\sqrt{n}}} \equiv t_{n-1} \qquad (59)$$

Repitiendo de forma análoga los cálculos de las ecuaciones (54) a (57), tenemos que el intervalo de confianza es:

$$I_{\mu}^{1-\alpha} = [\bar{x} \pm t_{n-1,\alpha} \frac{\hat{S}}{\sqrt{n}}] \qquad (60)$$

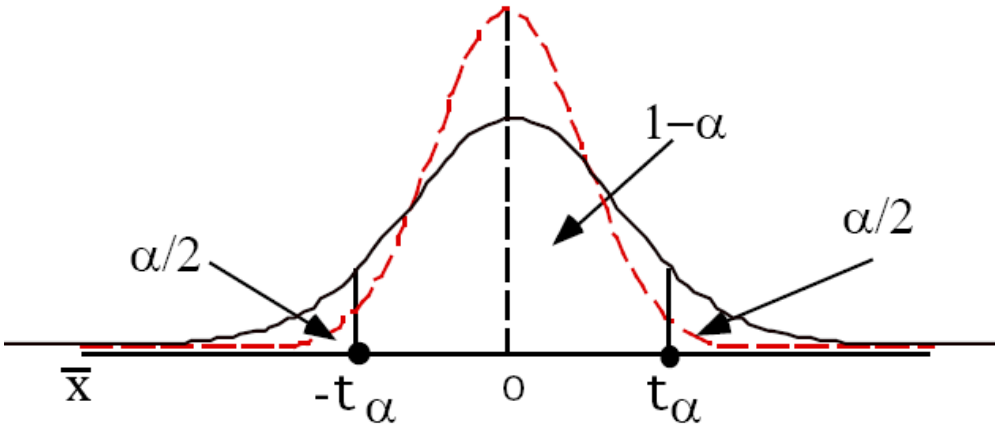


Figura (17)

4.2.2.2.-Implementación de la Inferencia estadística

Como sabemos que el ruido tiene una distribución normal, estamos hablando de inferencia paramétrica, y de entre todos los métodos de estimación vamos a elegir el de intervalos de confianza. Si nos fijamos en el nuevo programa principal **main.m** *(ver código en el Apéndice B.1)* podemos observar que esta tarea de detectar el ruido, se va a llevar acabo en la función **background_detection.m** *(ver código en el Apéndice B.2)* cuya estructura es la siguiente:

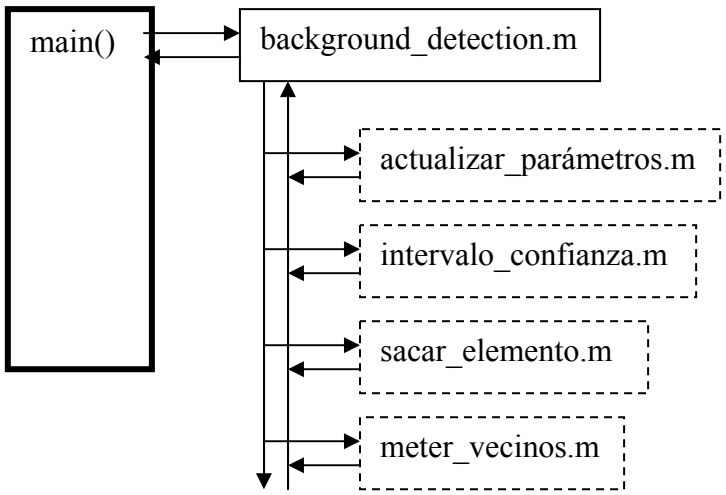


Figura (18)

Primero muestreamos por conglomerado, en este caso sabemos que cuanto más nos alejemos del centro, más posibilidades tenemos de encontrarnos píxeles de ruido y no de molécula, por lo tanto vamos a coger los seis planos que forman las caras del cubo. Es decir, para el caso del cubo de tamaño 64x64x64 vamos a coger como muestra los planos x=1,x=64,y=1,y=64,z=1 y z=64.

Luego como sabemos que el ruido es una variable aleatoria que se distribuye como una gausiana y desconocemos el valor de la varianza, la distribución que se elige para calcular los intervalos de confianza, es la siguiente:

$$\frac{\bar{x} - \mu_X}{\frac{S_X}{\sqrt{N_X}}} \equiv t_{Nx-1} \qquad (61)$$

Donde $\bar{x}$ es la media de la muestra obtenida, S_X la desviación estándar, N_X el número de píxeles que componen la muestra y μ_X la media que vamos a estimar.

$$P[t_{\alpha/2} \le t \le -t_{\alpha/2}] = 1 - \alpha \qquad (62)$$

Por lo tanto tenemos el intervalo:

$$t_{\alpha/2} \le t \le -t_{\alpha/2} \qquad (63)$$

Y sustituyendo la distribución:

$$t_{\alpha/2} \le \frac{\bar{x} - \mu_X}{\frac{S_X}{\sqrt{N_X}}} \le -t_{\alpha/2} \qquad (64)$$

Despejando ahora μ_X :

$$\bar{x} - t_{\alpha/2}\frac{S_X}{\sqrt{N_X}} \leq \mu_X \leq \bar{x} + t_{\alpha/2}\frac{S_X}{\sqrt{N_X}} \qquad (65)$$

No nos interesa tener S_X promediado por $\sqrt{N_X}$, así que el intervalo es el siguiente:

$$I = \bar{x} \pm t_{\alpha/2} S_X \qquad (66)$$

Los píxeles contiguos a los que forman la muestra los vamos a considerar posibles píxeles de background y los vamos a almacenar en una lista (vector columna), luego sacamos de esa lista un posible candidato a background utilizando la función **sacar_elemento.m** *(ver código en el Apéndice B.3)*, para definir si es o no background vamos a comprobar si se encuentra dentro del intervalo de confianza, calculado por la función **intervalo_confianza.m** *(ver código en el Apéndice B.4)* que implementa la ecuación (66) para calcular el intervalo. Si resulta que este píxel se encuentra dentro del intervalo, lo consideramos ruido (o background) y por lo tanto añadimos a todos sus vecinos (píxeles contiguos) a la lista de posibles candidatos a background utilizando la función **meter_vecinos.m** *(ver código en el Apéndice B.5)*. Finalmente actualizamos los valores de los estadísticos cada vez que localizamos 250 píxeles de ruido (o background) utilizando la función **actualizar_parámetros.m** *(ver código en el Apéndice B.6)* cada vez que la muestra se aumenta en 250 píxeles.

De este modo la función **background_detection.m**, devuelve un cubo (V_pixeles_ background) que vale “1” en las posiciones que hemos considerado ruido, y “0” en las que son molécula. Si hacemos por un corte en z=35 vemos lo siguiente:

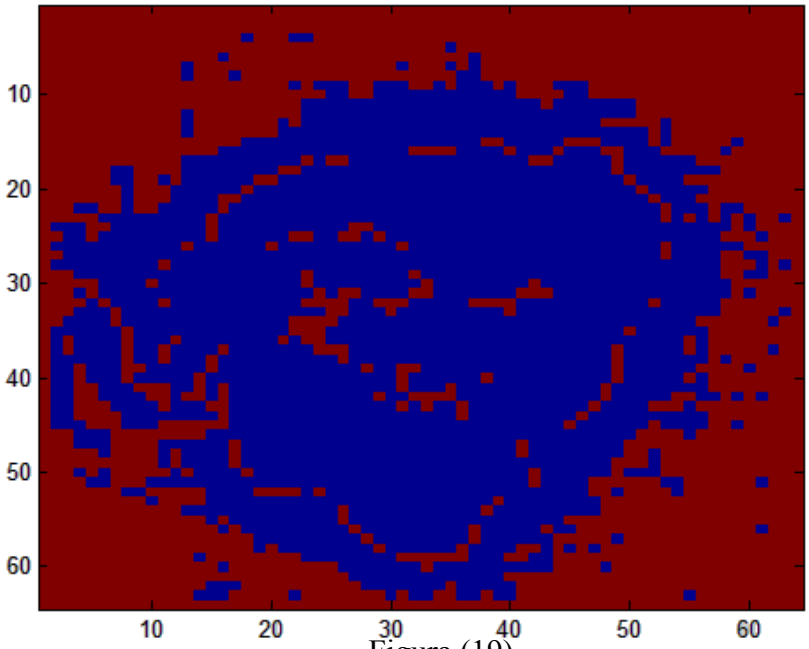


Figura (19)

Si nos fijamos en la Figura (19) observamos en color rojo, los píxeles que ha considerado como ruido, y en azul los píxeles que componen la molécula, que son los que nos interesan.

4.2.2.3.-Error α

Si nos fijamos bien en la Figura (19), veremos muchos píxeles azules aislados, rodeados de píxeles rojos, estos son probablemente ruido, pero cuyo valor estaba muy alejado de la media, y por lo tanto no han entrado dentro del intervalo de confianza, de modo que estos "píxeles azules aislados", son nuestro error α cometido al hacer inferencia.

Para visualizar mejor este problema vamos a fijarnos en el plano z=3, que está muy cerca de las caras del cubo, y por lo tanto debiera ser "todo ruido" (rojo), ya que cerca de las caras del cubo, no hay molécula.

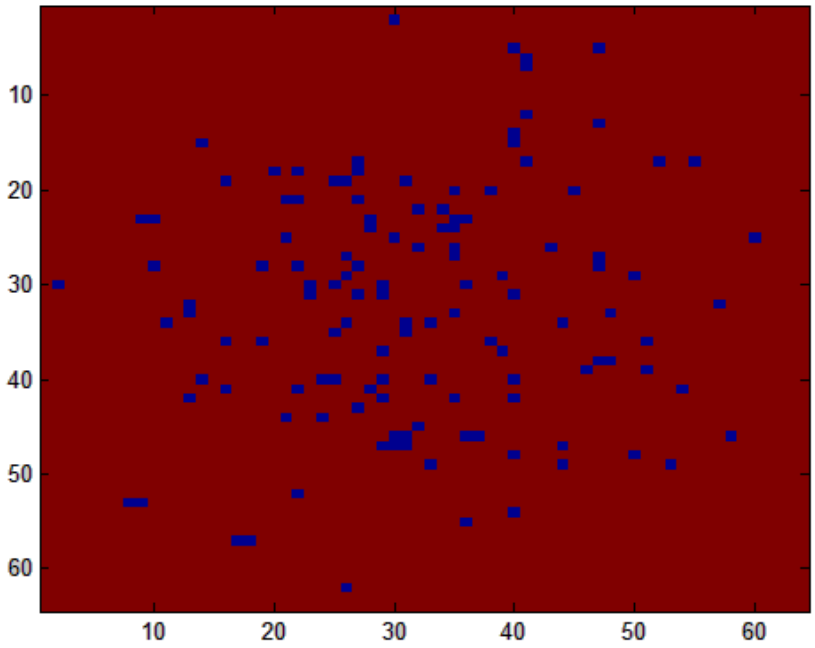


Figura (20)

Como hemos explicado anteriormente, la Figura (20) debiera ser un plano rojo, es decir todo ruido (o background), pero vemos "píxeles azules aislados", que son ruido, pero los hemos catalogado como molécula. Este error está contemplado en el α del intervalo de confianza. Para solventar este problema vamos a recurrir a una operación de la Morfología Matemática que se llama cierre, y que vamos a explicar a continuación.

4.2.2.4.-Morfología Matemática (cita [14])

La morfología matemática está basada en la teoría de conjuntos y se considera una técnica no lineal de tratamiento de señales. Sus aplicaciones son numerosas dentro del ámbito del procesamiento de imágenes, se emplea para la segmentación, restauración, detección de bordes, aumento de contraste, análisis de texturas, compresión, etc…

4.2.2.4.1.- Nociones sobre teoría de conjuntos

Los conjuntos se denotarán con mayúsculas (X,Y,Z,..) y los elementos contenidos en los conjuntos con minúsculas (p,q,r,..).

Definición 1: Dos conjuntos son iguales si están formados por los mismos elementos:

$$X = Y \Leftrightarrow (p \in X \Rightarrow p \in Y \text{ y } p \in Y \Rightarrow p \in X) \quad (67)$$

Definición 2: X es subconjunto de Y si todos los elementos de X pertenecen a Y:

$$X \subseteq Y \Leftrightarrow (p \in X \Rightarrow p \in Y) \quad (68)$$

Definición 3: La intersección de dos conjuntos X e Y es el conjunto de los elementos que pertenecen a ambos conjuntos:

$$X \cap Y = (p \mid p \in X \text{ y } p \in Y) \quad (69)$$

Esta última propiedad es importante en morfología y significa que $X \cap X = X$.

Definición 4: La unión de dos conjuntos se constituye por los elementos que pertenecen a uno o al otro:

$$X \cup Y = (p \mid p \in X \text{ o } p \in Y) \quad (70)$$

Definición 5: La diferencia entre conjuntos X e Y, la componen los elementos que pertenecen a X pero que no están incluidos en Y:

$$X \setminus Y = (p \mid p \in X \text{ y } p \notin Y) \quad (71)$$

Definición 6: La complementación de un subconjunto X, perteneciente a un conjunto Y (conjunto de referencia) se define como:

$$X\,c = (p \mid p \notin X \text{ y } p \in Y) \quad (72)$$

Definición 7: Sean X e Y dos conjuntos pertenecientes al conjunto Z. Para todo elemento $x \in X$ e $y \in Y$, es posible hacer corresponder una suma algebraica x+y. De esta manera se forma un nuevo conjunto denominado adicción de Minkowski y denotado por $X \oplus Y$:

$$X \oplus Y = \{x + y \mid x \in X, y \in Y\} \quad (73)$$

Definición 8: Dado un conjunto no vacío X, una relación binaria '$\leq$' en X es un orden parcial si cumple las siguientes propiedades:

1. $x \leq x$ (reflexiva).

2. $x \leq y$, $y \leq x$, implica que $x = y$ (antisimétrica).
3. $x \leq y$, $y \leq z$, implica que $x \leq y$ (transitiva).

Para cualquier $x, y, z \in X$. Un conjunto con una relación de este tipo será un conjunto que presenta un orden parcial y se denotará como $(X, \leq)$. El conjunto será totalmente ordenado si todos los elementos que lo componen son comparables, es decir: $x \leq y$ ó $y \leq x$, para cualquier par $(x,y) \in X$.

Definición 9: Sea $(X, \leq)$ un conjunto ordenado y $S \subset X$, un conjunto no vacío de X:

- Un elemento $x \in S$, es el menor elemento de S (mínimo) si $x \leq y$, para todo $y \in S$
- Un elemento $y \in S$, es el mayor elemento de S (máximo) si $x \leq y$, para todo $x \in S$.
- Un elemento $x \in X$, es cota inferior de S si $x \leq y$, para todo $y \in S$.
- Un elemento $y \in Y$, es cota superior de S si $x \leq y$, para todo $x \in S$.
- Un elemento $x \in X$, es extremo inferior o ínfimo de S si y sólo si es cota inferior de S y para toda cota inferior i de S se verifica que $i \leq x$. (Es la mayor de la cotas inferiores). Si este elemento existe es único y se denota por $\wedge$.
- Un elemento $y \in X$, es extremo superior o supremo de S si y sólo si es cota superior de S y para toda cota superior de s de S se verifica que $y \leq s$. (Es la menor de la cotas superiores). Si este elemento existe es único y se denota por $\vee$.

Definición 10: Un conjunto ordenado $(X, \leq)$ es un retículo completo si todos los subconjuntos de X poseen un ínfimo y un supremo.

Definición 11: Sean X e Y dos retículos completos. La relación f es una anamorfosis si y sólo si f es una biyección que conserva el ínfimo y supremo:

$$f(\wedge\{x_i \mid i \in I\}) = \wedge\{f(x_i) \mid i \in I\} \qquad (74)$$
$$f(\vee\{x_i \mid i \in I\}) = \vee\{f(x_i) \mid i \in I\}$$

para cualquier familia $\{x_i \mid i \in I\}$ en X, donde I es un conjunto de índices.

El concepto de retículo completo es la base para la formulación de la morfología matemática. Los operadores morfológicos de base deben conservar el orden presente en la estructura de retículo, deben ser crecientes. Un operador ψ , en un retículo completo X, es creciente si:

$$x \leq y \Rightarrow \psi(x) \leq \psi(y) \qquad (75)$$

4.2.2.4.2.-Propiedades principales de las transformaciones morfológicas

Toda operación morfológica está formada por el resultado de una o varias operaciones de conjuntos, como puede ser la unión, la intersección o la complementación. En una operación morfológica intervienen dos conjuntos X e Y, ambos subconjuntos del conjunto espacio Z. De los dos conjuntos, Y recibe el nombre de elemento estructurante, que para operar con X se desplazará a través del espacio Z.
Las operaciones morfológicas, son operaciones de conjunto $\psi(X)$, y tiene las siguientes propiedades:

1. **Invariabilidad a translación**:
$$\psi(X_p) = (\psi(X))_p \tag{76}$$
donde p es el factor de translación del conjunto.

2. **Compatibilidad con las homotecias**: Supongamos que λX es una homotecia, de modo que λ es una constante positiva que multiplica a las coordenadas de cada punto del conjunto X. Esta operación es equivalente a un cambio de escala. Si ψ no depende del cambio de escala tenemos que:
$$\psi(\lambda X) = \lambda\psi(x) \tag{77}$$

3. **Conocimiento local**: Se dice que una transformación morfológica ψ posee el principio de conocimiento local si para cualquier conjunto de puntos M, subconjunto del dominio N, la transformación del conjunto X restringido al dominio de M, y después restringido al dominio N, es equivalente a aplicar la transformación $\psi(X)$ y restringir el resultado en M:
$$\psi(X \cap N) \cap M = \psi(X) \cap M \tag{78}$$

4. **Continuidad**: Una transformación morfológica ψ no exhibe ningún cambio abrupto.

4.2.2.4.3.-Transformaciones morfológicas elementales

El objetivo de las transformaciones morfológicas es la extracción de estructuras geométricas en los conjuntos sobre los que se opera, mediante la utilización de otro conjunto de forma conocida, que llamaremos elemento estructurante. Hay dos tipos de transformaciones elementales, la erosión y la dilatación.

■ **La erosión**

Una erosión es una operación que conmuta con el ínfimo. Suponiendo que tenemos un retículo X, la operación erosión es una función $\varepsilon : X \to X$ que cumple:
$$\varepsilon(\underset{i\in I}{\wedge} x_i) = \underset{i\in I}{\wedge} \varepsilon(x_i) \tag{79}$$
siendo I cualquier conjunto de índices y x_i una colección arbitraria de valores pertenecientes a X.

La transformación por erosión, consiste en comprobar si el elemento estructurante Y está totalmente incluido dentro del conjunto X. Cuando no es así el resultado de la operación es el conjunto vacío.

Es decir cuando se tiene un retículo X y un elemento estructurante Y, la erosión se define como el conjunto de elementos x de X de forma que cuando el elemento estructurante se centra en x, Y queda contenido en X:

$$\varepsilon_Y(X) = \{x \mid Y_X \subseteq X\} \tag{80}$$

En la Figura que mostramos a continuación se puede ver un claro ejemplo de cómo actúa la erosión:

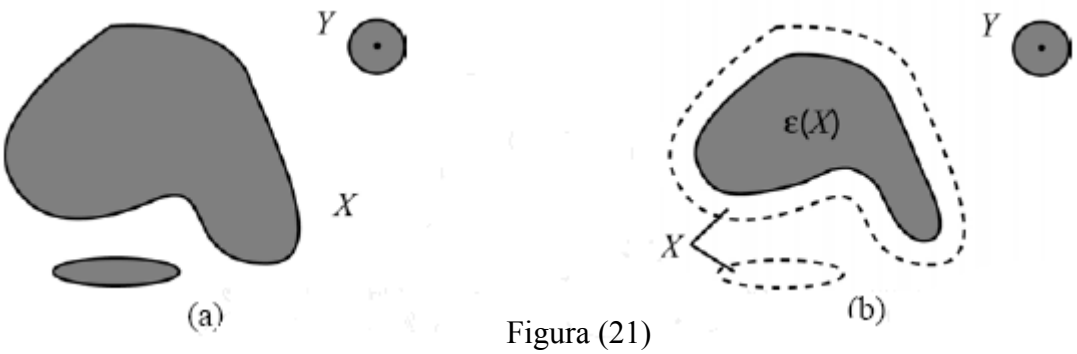


Figura (21)

Para el caso de imágenes binarias y escala de grises, considerando una imagen f y un elemento estructurante Y, definimos la transformación erosión $\varepsilon_Y(f)$ como el mínimo $(\wedge)$ de las translaciones de f por los elementos s de Y:

$$\varepsilon_Y(f) = \bigwedge_{s \in Y}(f_{-s}) \tag{81}$$

■ **La dilatación**

La dilatación es un operador $\delta : X \to X$ que conmuta con el supremo de una colección de valores:

$$\delta(\bigvee_{i \in I} x_i) = \bigvee_{i \in I} \delta(x_i) \tag{82}$$

siendo I cualquier conjunto de índices y x_i una colección arbitraria de valores pertenecientes a X.
El resultado de la dilatación, es el conjunto de puntos origen del elemento estructurante Y, siempre que este contenga algún elemento de X, al desplazarse por el espacio que contiene a ambos:

$$\delta_Y(X) = \{x \mid Y_x \cap X \neq 0\} \tag{83}$$

En la Figura siguiente mostramos un ejemplo de dilatación:

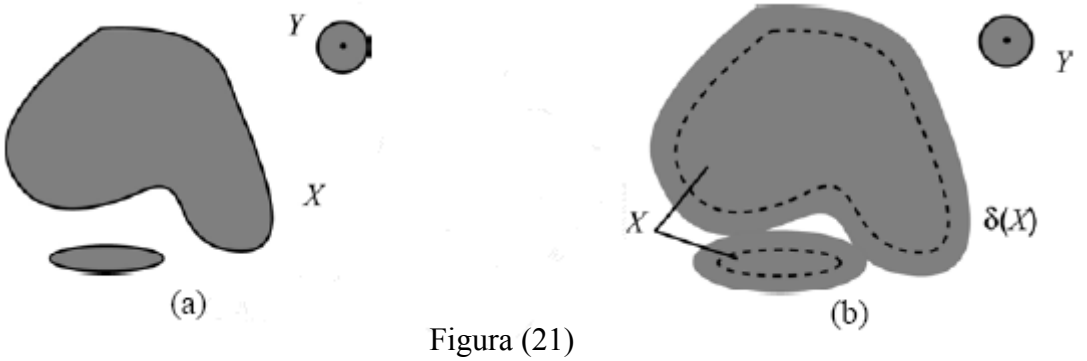


Figura (21)

Si estamos hablando de imágenes f binarias o de escala de grises la dilatación se puede definir como el máximo valor de las traslaciones de f.

$$\delta_Y(f) = \bigvee_{s \in Y} f_{-s} \tag{84}$$

4.2.2.4.4.-Propiedades de las operaciones básicas de erosión y dilatación

1. **Dualidad**: Las transformaciones erosión y la dilatación son operaciones duales con respecto a la complementación. Dicho de otro modo, una erosión es equivalente a la complementación de la dilatación de la imagen complementada con el mismo elemento estructurante y viceversa.

$$\varepsilon_Y = C\delta_Y C \tag{85}$$

2. **Crecientes**: Las transformaciones erosión y dilatación son crecientes. Esto quiere decir, que para dos imágenes f y g:

$$\begin{aligned} &Si \quad f \le g \;\Rightarrow\; \varepsilon(f) \le \varepsilon(g) \\ &Si \quad f \le g \;\Rightarrow\; \delta(f) \le \delta(g) \end{aligned} \tag{86}$$

3. **Extensividad y antiextensividad**: La operación de dilatación es extensiva, es decir para una imagen f, $f \le \delta(f)$, en cambio la operación erosión es antiextensiva: $\varepsilon(f) \le f$. En general se cumple:

$$\varepsilon(f) \le f \le \delta(f) \tag{87}$$

   Esta propiedad se puede comprobar claramente viendo un ejemplo par una señal unidimensional:

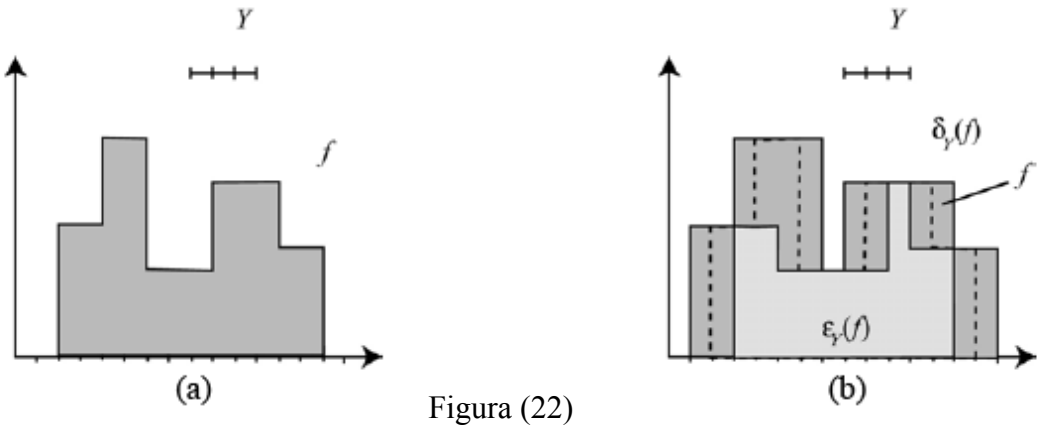


Figura (22)

4.2.2.4.5.-Apertura y cierre

Si tenemos un retículo X, las operaciones de dilatación $\delta(X)$ y erosión $\varepsilon(X)$, no admiten inversa, de modo que partiendo de $\delta(X)$ o $\varepsilon(X)$, no se puede obtener el retículo X. No obstante mediante el uso de operadores básicos nos podemos aproximar a X gracias a la propiedad de dualidad que poseen.

Una posibilidad de recuperar X una vez ha sido erosionado, es aplicar una dilatación justo después de la erosión, esta unión de operaciones es conocida como apertura. Si intentamos recuperar X después de una dilatación, erosionando, esta operación, es conocida como cierre.

En la morfología matemática, las operaciones de apertura y cierre, son los filtros básicos, con cuales se construyen los filtros más complejos.

$$Apertura: X \Rightarrow \varepsilon(X) = Y \Rightarrow \delta(Y)$$

$$Cierre: X \Rightarrow \delta(X) = Y \Rightarrow \varepsilon(Y)$$

■ **Apertura**

La apertura de una señal f por un elemento estructurante Y se denota por $\gamma_Y(f)$ y se define como la erosión de f por Y, seguida de la dilatación por el mismo elemento estructurante.

$$\gamma_Y(f) = \delta_Y(\varepsilon_Y(f)) \tag{88}$$

La apertura de una imagen es independiente del origen del elemento estructurante, puesto que si la erosión se corresponde con una intersección de translaciones, la dilatación que sigue es una unión de translaciones en dirección opuesta.

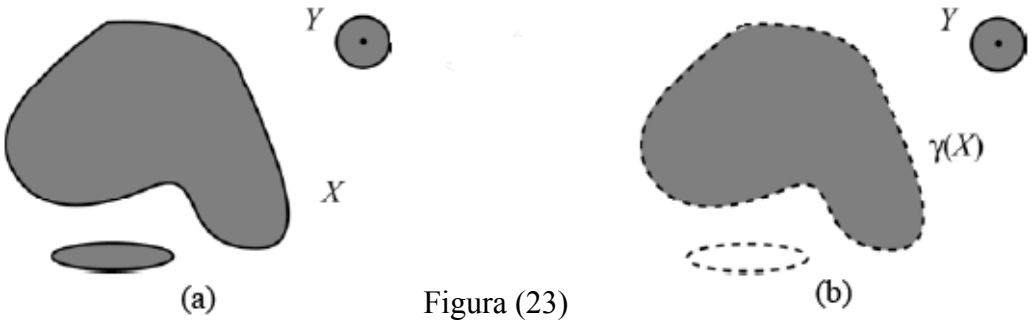


Figura (23)

Podemos ver en la Figura (23), como el elemento estructurante Y, hacer desaparecer en la erosión, una parte del conjunto X que no se recupera en la dilatación. Por lo tanto la apertura sirve para hacer desaparecer objetos de menor tamaño que el del elemento estructurante.

■ **Cierre**

El cierre de una señal f por un elemento estructurante Y se denota por $\varphi_Y(f)$ y se define como la dilatación de f por Y seguida de la erosión por el mismo elemento estructurante:

$$\varphi_Y(f) = \varepsilon_Y(\delta_Y(f)) \tag{89}$$

Como vimos que ocurría con la apertura, el cierre también es independiente del origen del elemento estructurante.

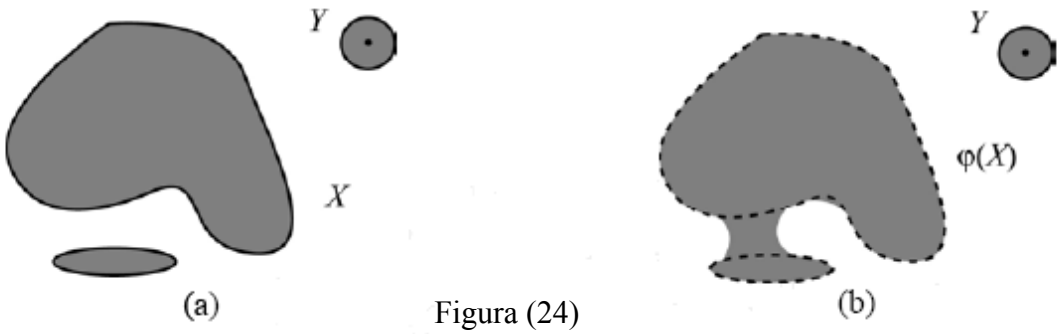


Figura (24)

Si nos fijamos en la Figura (24) , se puede intuir que el cierre es el espacio descrito por el elemento estructurante cuando es forzado a estar fuera de los conjuntos.

4.2.2.4.6.-Propiedades de la apertura y el cierre

1. **Dualidad:** La apertura y el cierre son operaciones duales con respecto a la complementación, esto significa que la apertura de una imagen es equivalente al complemento del cierre de la imagen complementada:

$$\gamma_Y = C\varphi_Y C \tag{90}$$

   El uso del cierre o la apertura para eliminar objetos no deseados en una imagen viene determinado por el tipo de imagen. Si los objetos no deseados son de tonalidades parecidas a la imagen (ej: ruido simétrico) tendremos que usar una combinación de cierres y aperturas.

2. **Relaciones de orden:** La operación cierre φ es una operación extensiva, y la operación apertura γ antiextensiva. Es decir, si tenemos una imagen I:

$$\gamma \le I \le \varphi \tag{91}$$

3. **Crecientes:** Tanto la operación de cierre como la de apertura son crecientes. Es decir para dos imágenes f y g:

$$\begin{aligned} &Si \quad f \le g \Rightarrow \gamma(f) \le \gamma(g) \\ &Si \quad f \le g \Rightarrow \varphi(f) \le \varphi(g) \end{aligned} \tag{92}$$

4.2.2.5.-Implementación de la Morfología Matemática

Realizamos el cierre con la función **mat_morph_cierre.m** *(ver código en el Apéndice B.7)* , en esta fase de simulación usamos una función de matlab (`bwmorph`) para realizar la operación de cierre plano a plano, hasta completar todo el cubo. Más adelante para versión final del programa en C, programaremos una versión 3D del cierre morfológico.

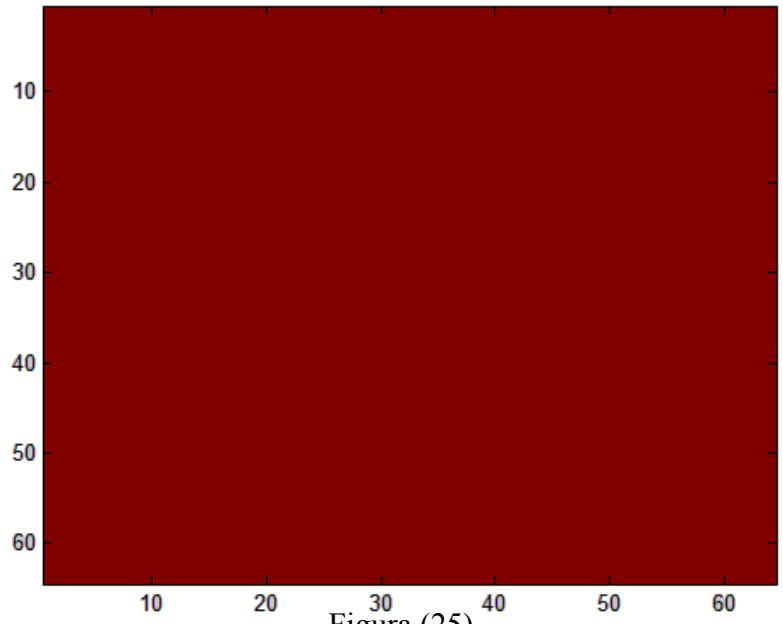


Figura (25)

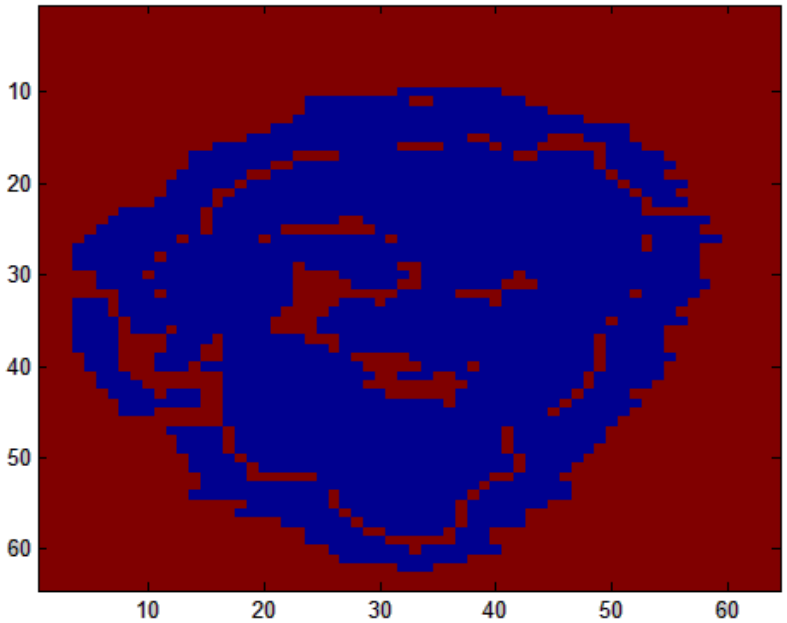


Figura (26)

La Figura (25), es un corte del resultado para z=3, si la comparamos con la análoga Figura (20), nos damos cuenta que todo el ruido que confundíamos con molécula (pixeles azules) ha desaparecido, tenemos un plano rojo (todo ruido). Para el caso de la Figura (26), que es un corte en z=35, ocurre lo mismo, si la comparamos con la (19) vemos como todos los píxeles azules aislados han desaparecido.

4.2.2.6.-Implementación y resultados

Finalmente después de todas estas operaciones utilizamos la función **homogeneizar.m** *(ver código en el Apéndice B.8)* para poner todo el ruido a un mismo valor, que va a ser su media. Por lo tanto la estructura final de la primera fase, es la siguiente:

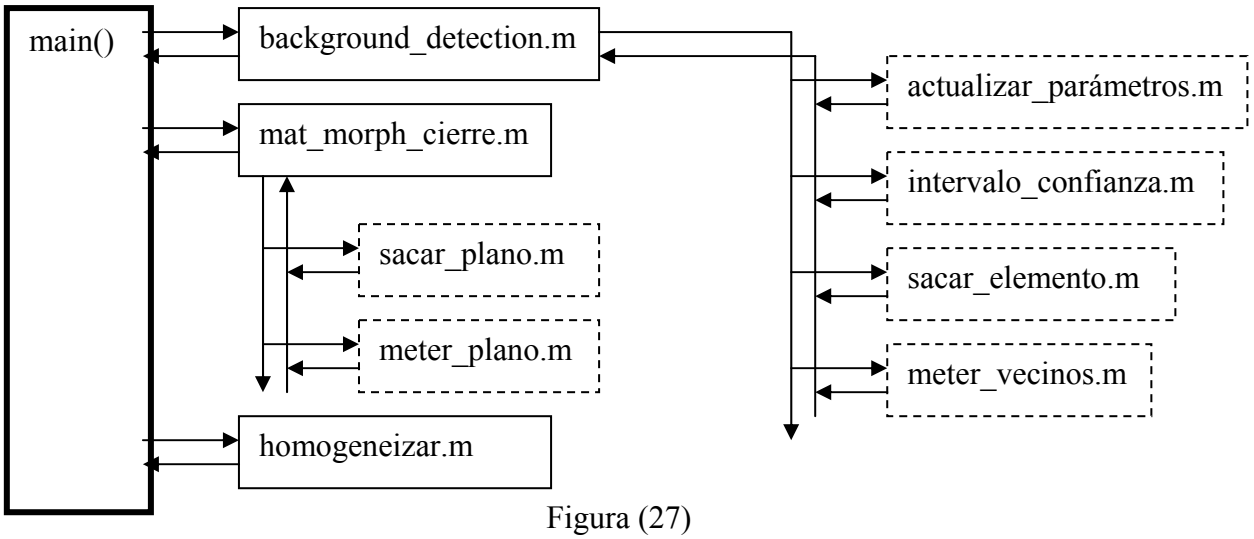


Figura (27)

Ahora se va a mostrar el plano z=35 del resultado obtenido de procesar la imagen con la primera fase:

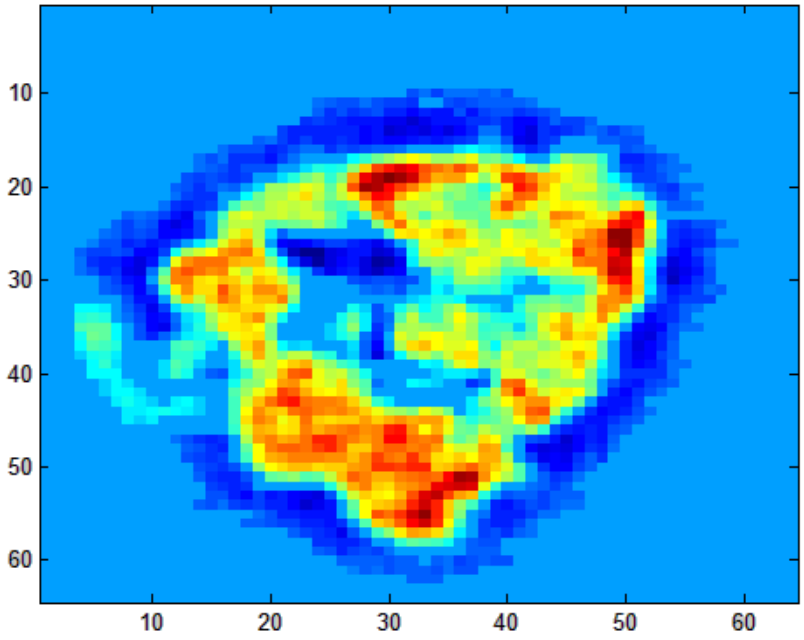


Figura (28)

Si nos fijamos en la Figura (28), y la comparamos con la Figura (11), se hace evidente que casi todo el ruido que teníamos rodeando la molécula en forma de diversas tonalidades del azul, ahora ha sido sustituido por un solo valor.

4.2.3.-Fase 2: Operadores Sobel

4.2.3.1.-Concepto 3D

Vamos a añadir un concepto nuevo, a partir de ahora vamos a trabajar en tres dimensiones, por lo general las técnicas que encontramos son para el procesamiento de imágenes en dos dimensiones. ¿Por qué es mejor trabajar en 3D? La respuesta es bastante obvia, al trabajar en 3D tenemos bastante más información, sobre el vecindario de un píxel, y por lo tanto podemos estimar con mayor precisión. Por ejemplo, al trabajar en 2D utilizábamos ventanas de 3x3, luego para un píxel tenemos una muestra compuesta por 8 vecinos, en cambio si trabajamos en 3D, y usamos una ventana de 3x3x3, para un píxel tenemos una muestra de 26 vecinos, es decir una muestra más de tres veces la muestra 2D. Por lo tanto al tener muestras de mayor tamaño los resultados obtenidos mediante el cálculo de probabilidades y estadística son mucho más precisos.

4.2.3.2.-Implementación y resultados

La función que realiza la segunda fase, es **sobel_3D.m** *(ver código en el Apéndice B.9)*, en vez de procesar plano a plano, esta vez tenemos una función **sacar_cubo_lado_n.m** *(ver código en el Apéndice B.10)* que se encarga de extraer cubos de tamaño 3x3x3. Luego estos cubos son procesados por la función **sobel_operator_3D.m** *(ver código en el Apéndice B.11)* que es la que detecta los bordes de la imagen mediante el empleo del gradiente usando los operadores explicados en el apartado "4.1.2.-Operadores Sobel" la única diferencia, es que ahora usamos unas máscaras distintas para derivar en 3D compuestas por 3 cubos:

$$\begin{bmatrix} -1 & -3 & -1 \\ -3 & -6 & -3 \\ -1 & -3 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 3 & 1 \\ 3 & 6 & 3 \\ 1 & 3 & 1 \end{bmatrix}$$

x-1 x x+1

$$\begin{bmatrix} 1 & 3 & 1 \\ 0 & 0 & 0 \\ -1 & -3 & -1 \end{bmatrix} \quad \begin{bmatrix} 3 & 6 & 3 \\ 0 & 0 & 0 \\ -3 & -6 & -3 \end{bmatrix} \quad \begin{bmatrix} 1 & 3 & 1 \\ 0 & 0 & 0 \\ -1 & -3 & -1 \end{bmatrix}$$

y-1 y y+1

$$\begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -3 & 0 & 3 \\ -6 & 0 & 6 \\ -3 & 0 & 3 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix}$$

z-1 z z+1

Figura (29)

Como se puede ver en la Figura (29), derivamos en el sentido del eje x con el cubo formado por los planos x-1,x y x+1, en el sentido del eje y con el cubo formado por los planos y-1,y,y+1, y en el z con el cubo formado por los planos z-1,z y z+1.

La estructura de esta fase es la siguiente:

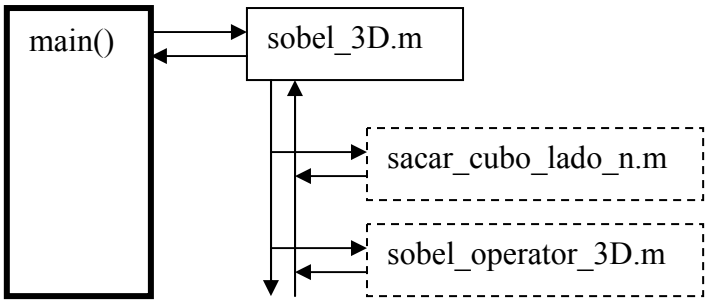


Figura (30)

El resultado que obtenemos es el siguiente:

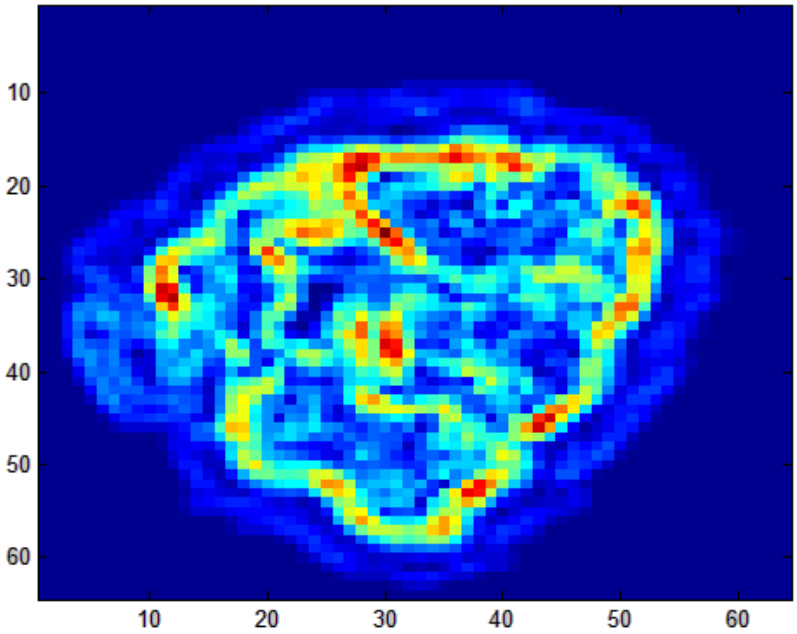


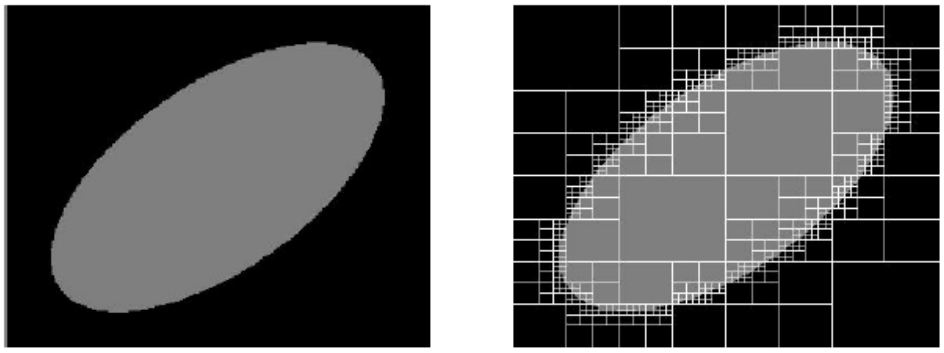
Figura (31)

La Figura (31), es un corte por el plano z=35 del volumen que contiene la imagen procesada mediante los operadores de Sobel, si la comparamos con la Figura (12), que corresponde con el mismo corte, pero del volumen procesado mediante los operadores 2D, vemos claramente que el nuevo proceso nos da una imagen en la que se ven mejor los bordes.

4.2.4.-Fase 3: Mean Edge Gray Value

4.2.4.1.-Vecindario de tamaño variable

El nuevo concepto que se implementa aquí, es el de vecindario de tamaño variable, como se comentó anteriormente, cuanto más grande sea la muestra (vecindario) mejor, ya que tenemos más información y esto nos permite estimar los estadísticos con mayor precisión. Por lo tanto, ahora no vamos a usar ventanas de 3x3x3 como vecindario del píxel x_{ij}, que ocupa la posición del píxel central del cubo, si no que el vecindario va a ser variable pudiendo tomar los siguientes tamaños 3x3x3, 5x5x5, 7x7x7 y 9x9x9. ¿Cómo decidimos el tamaño óptimo del vecindario? Se empieza cogiendo un vecindario de tamaño 3x3x3, y mediante inferencia estadística, utilizando intervalos de confianza (explicado en el apartado 4.2.2.1.-Inferencia estadística) establecemos si el cubo del siguiente tamaño, en este caso 5x5x5, se puede considerar también como vecindario del píxel x_{ij}, en caso afirmativo, porque nos encontremos en una zona con patrones muy similares, partimos del cubo 5x5x5 y lo comparamos con el siguiente más grande , el de 7x7x7 para ver si lo podemos considerar como posible vecindario, y esta cadena de cálculos se repite hasta llegar al tamaño máximo o hasta que no podamos considerar un cubo superior como vecindario. Una imagen que ilustra lo explicado es la siguiente:



(a) Figura (32) (b)

La Figura (32-a) muestra una imagen muy simple, es una eclipse en tono gris sobre un fondo negro, a la derecha (32-b) vemos unos posibles tamaños de vecindarios. Como es lógico, tanto en el fondo negro lejos del grano de arroz, como en el interior del grano de arroz, los tamaños del vecindario son muy grandes (ventanas grandes), ya que los patrones en estos sitios son iguales o muy similares, en cambio cerca del borde del grano de arroz se utilizan ventanas más pequeñas, ya que el patrón no es de un solo color, si no que existe el cambio de gris a negro y viceversa. En nuestro caso, en vez de tener un grano de arroz, tenemos una molécula, y por tanto después de ejecutar el programa, esperamos tamaños grandes de vecindarios en el fondo lejos de la molécula (y también en el interior de la molécula, si existen patrones parecidos) y pequeños vecindarios por los bordes de la molécula.

4.2.4.2.-Implementación y resultados

La función que realiza esta tercera fase es **mean_edge_gray_value_3D.m** *(ver código en el Apéndice B.12)* cuya estructura es la siguiente:

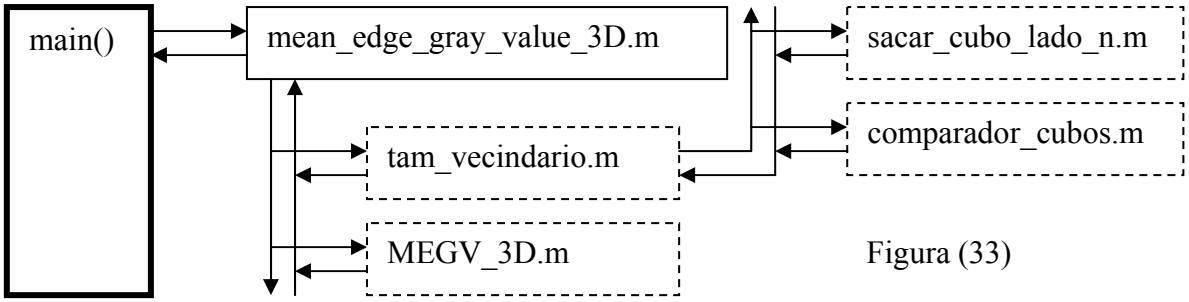


Figura (33)

Lo primero que hacemos, es calcular el tamaño de vecindario para cada píxel, y para eso utilizamos la función **tam_vecindario.m** *(ver código en el Apéndice B.13)*, ésta como hemos explicado emplea la función **sacar_cubo_lado_n.m** para ir sacando cubos de distinto tamaño, en todo momento tenemos dos cubos, uno de tamaño nxnxn y otro de tamano (n+2)x(n+2)x(n+2), de los cuales el pequeño ya hemos comprobado que lo podemos considerar vecindario de un píxel x_{ij}, luego cogemos esos dos cubos y los comparamos en la función **comparador_cubos.m** *(ver código en el Apéndice B.14)*, para ver si podemos considerar vecindario de x_{ij} al cubo de mayor tamaño. Esta comparación se hace utilizando intervalos de confianza.

En este caso como queremos comparar dos medias ,para ver si las podemos considerar o no iguales, y las varianzas son desconocidas y distintas, empleamos la siguiente distribución:

$$\frac{\bar{d}-\mu_d}{\sqrt{\frac{S_X^2}{N_X}+\frac{S_Y^2}{N_Y}}} \approx t_{\frac{\left(\frac{S_X^2}{N_X}+\frac{S_Y^2}{N_Y}\right)^2}{\frac{S_X^2}{N_X^2(N_X-1)}+\frac{S_Y^2}{N_Y^2(N_Y-1)}}} \tag{93}$$

Donde $\bar{d}=\bar{x}-\bar{y}$ es igual a la diferencia de la media del cubo pequeño con respecto al grande, S_x es la desviación estándar del cubo pequeño, S_Y es la desviación estándar del cubo grande, N_X el número de píxeles del vecindario (tamaño de la muestra) que compone cubo pequeño, N_Y es número de píxeles del vecindario del cubo grande y μ_d va a ser el intervalo que calculemos:

$$t_{\alpha/2} \le \frac{\bar{d}-\mu_d}{\sqrt{\frac{S_X^2}{N_X}+\frac{S_Y^2}{N_Y}}} \le t_{-\alpha/2} \tag{94}$$

Despejando μ_d del mismo modo que hacíamos en la ecuación (64), obtenemos:

$$I=[\bar{d}\pm\sqrt{\frac{S_X^2}{N_X}+\frac{S_Y^2}{N_Y}}\cdot t_{\alpha/2}] \tag{95}$$

Ahora hacemos una comprobación, si el intervalo I contiene al cero, podemos considerar que las medias son iguales, y por lo tanto nos quedamos con el cubo de mayor tamaño. Esta operación se repite para cada píxel de la imagen 3D hasta terminar, y el resultado es el siguiente:

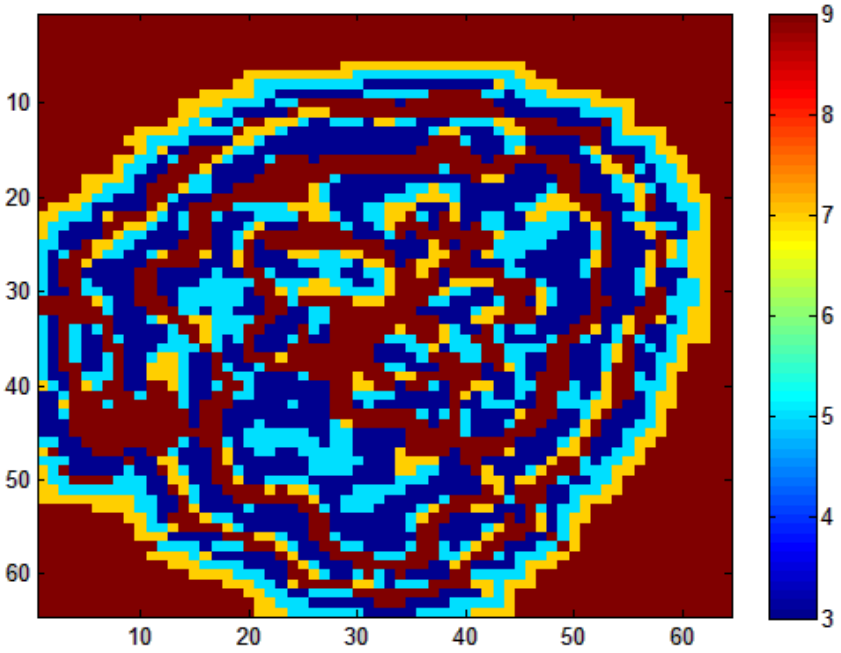


Figura (34)

En la Figura (34), podemos ver un corte por el plano z=35 del volumen V_tam, que contiene en cada píxel el tamaño del vecindario que le corresponde, y como habíamos supuesto en el apartado 4.2.4.1. por el exterior lejos de la molécula y en ciertas zonas similares de la molécula tenemos cubos de 9x9x9 (color rojo), y en cambio por las zonas donde hay contraste tenemos vecindarios de 3x3x3 o 5x5x5 (colores azules), por tanto funciona tal y como esperábamos. Una vez que tenemos calculado los tamaños de vecindario, calculamos el mean edge gray value con la función **MEGV_3D.m** *(ver código en el Apéndice B.15)* . Ahora mostramos como siempre el corte z=35 del volumen resultado V_E:

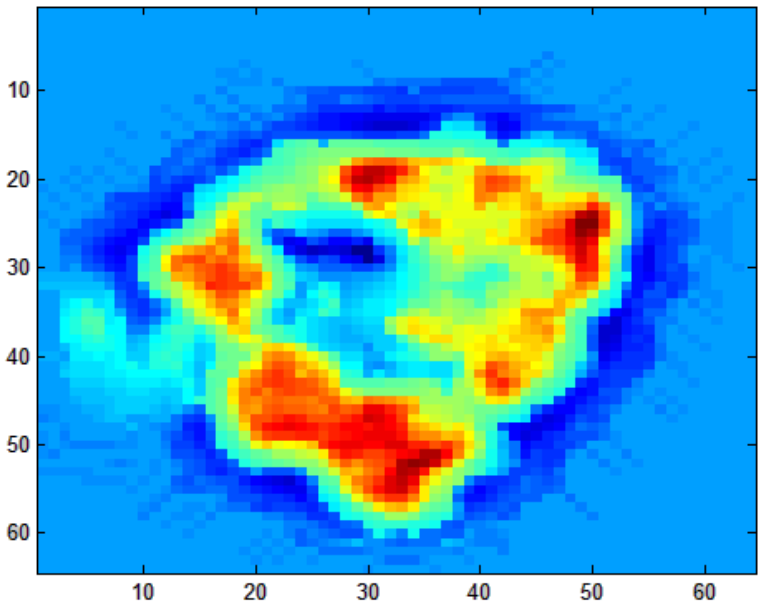
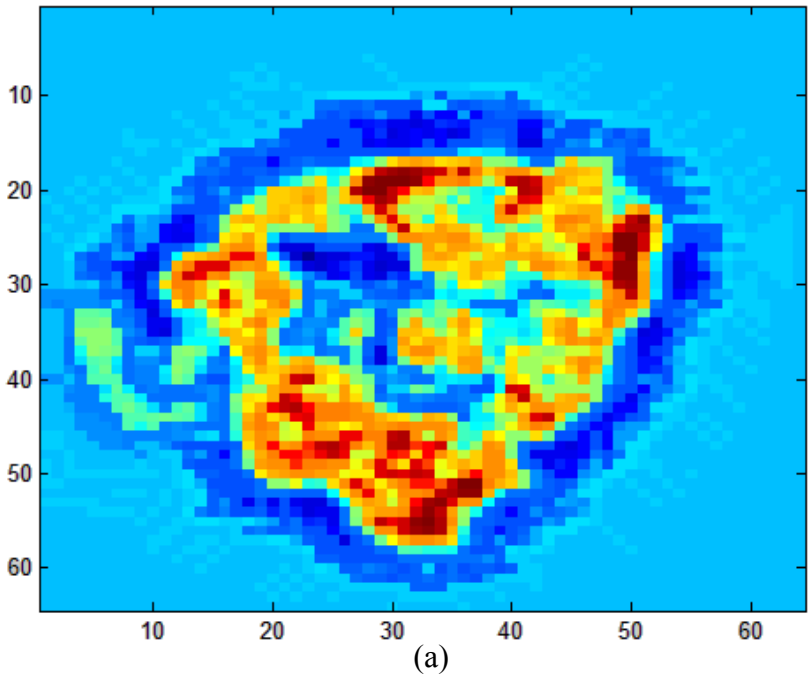


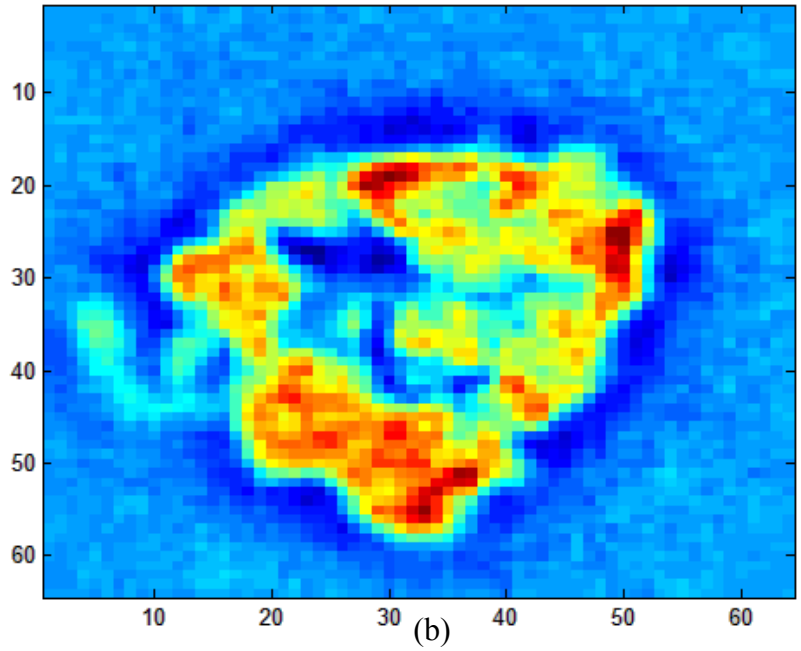
Figura (35)

4.2.4.-Fase 4: Función no lineal y resultado final

La función no lineal no sufre ningún tipo de cambio, ya ha sido explicada en el apartado 4.1.4, y el código se puede ver en el Apéndice A.12. De modo que simplemente mostramos el resultado de hacer un corte en z=35 al volumen V_F, resultado final de todo el procedimiento:



(a)



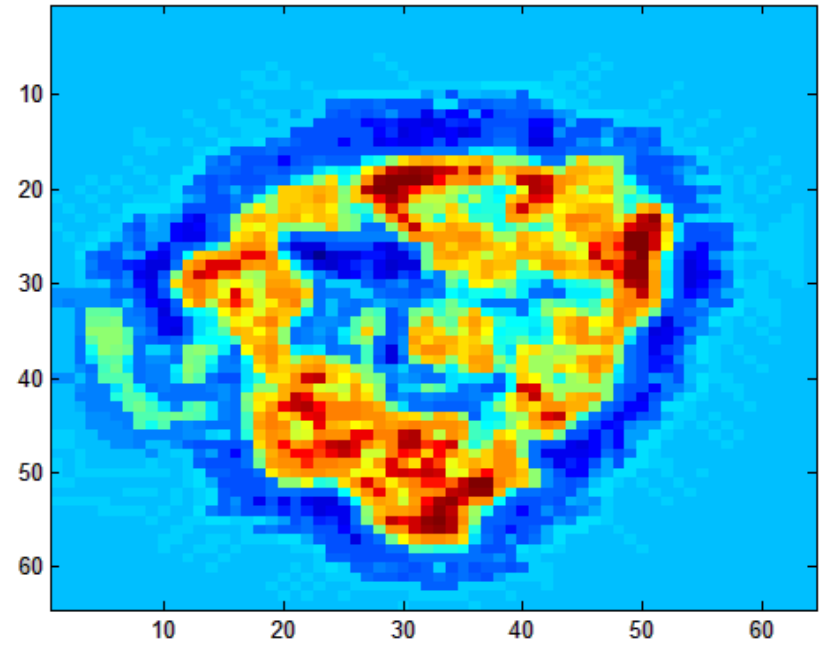
(b)

Figura (36)

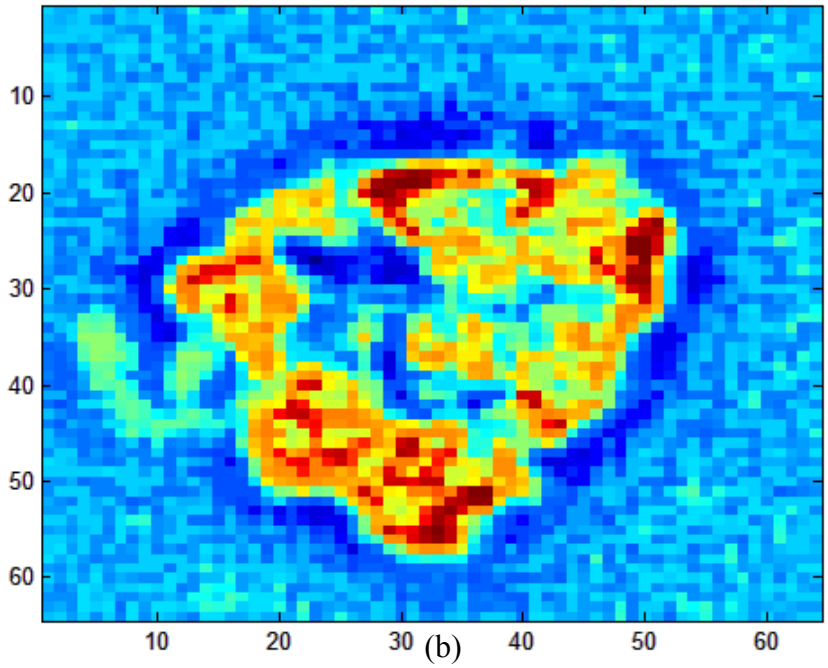
Si nos fijamos en la Figura (36-b) vemos el corte z=35 sobre la molécula original y en la Figura (36-a) tenemos un corte en el mismo plano el resultado final que ofrece nuestra metodología para la mejora del contraste, a simple vista podemos comprobar que la molécula procesada tiene mucho más contraste que la original, luego funciona correctamente.

4.2.6.-Conclusiones de la comparación de métodos

Ahora vamos a comparar los dos resultados obtenidos con los dos métodos, para ver si a simple vista podemos comprobar mediante esta simulación que efectivamente nuestra técnica es mejor.



(a)



(b)

Figura (37)

Si nos fijamos en la Figura (37), la imagen (a) es pertenece a nuestro método y la imagen (b) a la metodología actual, de nuevo a simple vista podemos comprobar que nuestro método es un genera más constraste.

4.3.-METODOLOGÍA FINAL

La versión final de nuestra metodología se va a implementar en C++ utilizando el entorno de desarrollo Eclipse sobre un sistema operativo Linux. Todo ello para poder introducir esta nueva metodología como una clase de la librería de tratamiento digital de imágenes Xmipp del CNB.

4.3.1.-Introducción

El proceso a realizar se va a resolver del mismo modo que el explicado en el apartado 4.2, salvo con la excepción, de que ahora vamos a mejorar el cálculo de los bordes de la imagen sustituyendo los operadores de Sobel, por una técnica más compleja y precisa. Esta técnica consiste en calcular la función continua de la molécula, a partir de las muestras que tenemos, los píxeles. De este modo mediante el teorema de muestreo utilizamos las funciones B-Splines para interpolar la función continua, para luego derivarla, y calcular los bordes.

4.3.2.-Interpolación y ciencia

Dado un conjunto discreto de medidas (x_i , y_i), la interpolación es el arte de construir los puntos que se encuentran entre nuestras medidas (y que no tenemos) mediante una función continua $y = f(x)$ que tiene que pasar por las medidas conocidas, por lo tanto se tiene que cumplir que: $y_i = f(x_i)$. En el caso de las ingenierías y algunas ciencias, es frecuente disponer de un cierto número de puntos obtenidos por muestreo o a partir de un experimento y lo que se pretender es construir una función que los ajuste. En nuestro caso concreto de imágenes biomédicas nuestro conjunto discreto de medidas suele ser los valores de la imagen, $z_i = I(x_i, y_i)$, siendo z_i el valor de gris de la imagen y siendo (x_i , y_i) su localización en el espacio. Para imágenes a color, se puede descomponer el color en sus tres componentes (rojo, verde y azul), y cada uno del ellos impone una restricción de la misma forma que los valores de gris.

La interpolación es importante para saber el valor de nuestra imagen entre los píxeles conocidos, esto de gran utilidad para poder realizar rotaciones, translaciones, downsampling, upsampling, etc…
Dos formas frecuentes de interpolación son la interpolación lineal, y la interpolación polinómica.

■ Interpolación lineal
Este método es uno de los más sencillos y también de los menos precisos. En general la interpolación lineal intenta calcular el conjunto de nuevos valores entre dos puntos “a” y “b” de coordenadas (x_a , y_a) y (x_b , y_b) mediante la fórmula.

$$y = y_a + (x - x_a)\frac{y_b - y_a}{x_b - x_a} \tag{96}$$

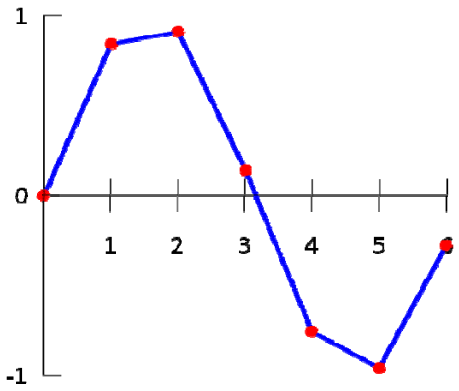


Figura (38)

En la Figura (38), vemos un ejemplo de interpolación lineal, donde los puntos rojos, son nuestro conjunto de muestras, y la línea azul representa la interpolación.

■ Interpolación polinómica

Dado un grupo de N medidas, podemos estimar un polinomio de grado N-1, que tiene N coeficientes y por tanto N grados de libertad, que pase a través de todas estas medidas. Este polinomio es único y viene dado por la siguiente fórmula:

$$f(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_{N-1}(x - x_0)(x - x_1)\ldots(x - x_{N-2}) \qquad (97)$$

Donde a_n viene dado por:

$$\begin{aligned} a_0 &= y_0 \\ a_n &= \sum_{i=0}^{n} \frac{y_i}{\prod_{\substack{j=0 \\ j \neq i}}^{n} (x_i - x_j)} \end{aligned} \qquad (98)$$

El uso de polinomios para interpolar está justificado por dos poderosos motivos, el primero es que son fáciles de manipular, derivar o integrar, y el segundo viene dado por un teorema de Weierstrass que establece que en un intervalo cerrado cualquier función continua puede ser aproximada de forma uniforme con el grado de precisión que se desee siempre que el polinomio sea de un grado suficiente.

4.3.3.-Muestreo

Bajo ciertas condiciones, una señal puede representarse y reconstruirse por completo partiendo del conocimiento de sus valores en puntos equiespaciados en el tiempo a los que llamaremos muestras. Esta propiedad se deriva de un resultado básico que se conoce como el "Teorema de muestreo" y que es de gran importancia, ya que nos sirve de puente entre las señales continuas y las discretas.

■ Representación de una señal continua mediante sus muestras

Por lo general no debiéramos esperar, que una señal continua pudiese especificarse de forma inequívoca mediante una secuencia de muestras igualmente espaciadas. Como se puede ver en la Figura (39), mostramos tres señales continuas totalmente distintas, que toman los mismos valores para los múltiplos enteros de T

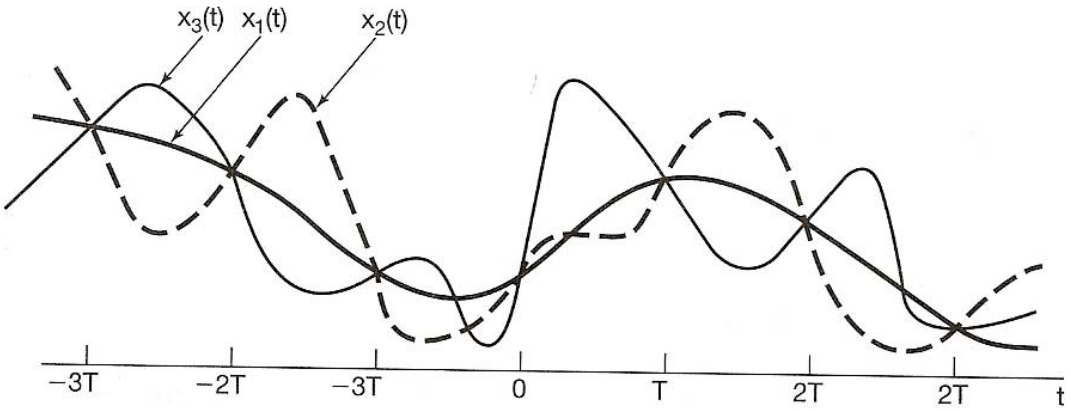


Figura (39)

Esto demuestra que hay un número infinito de señales que puede pasar por un conjunto de muestras. Sin embargo, si se cumplen ciertas condiciones, como que la señal esté limitada en banda o que las muestras estén lo suficientemente cercanas, entonces las muestras identifican unívocamente una señal, y por lo tanto podemos reconstruirla perfectamente.

■ Muestreo con tren de impulsos

Un método por el cual se puede muestrear de forma sencilla una señal continua a intervalos regulares, es mediante el uso de un tren de impulsos periódicos que multiplique a la señal continua x(t) que queremos muestrear.

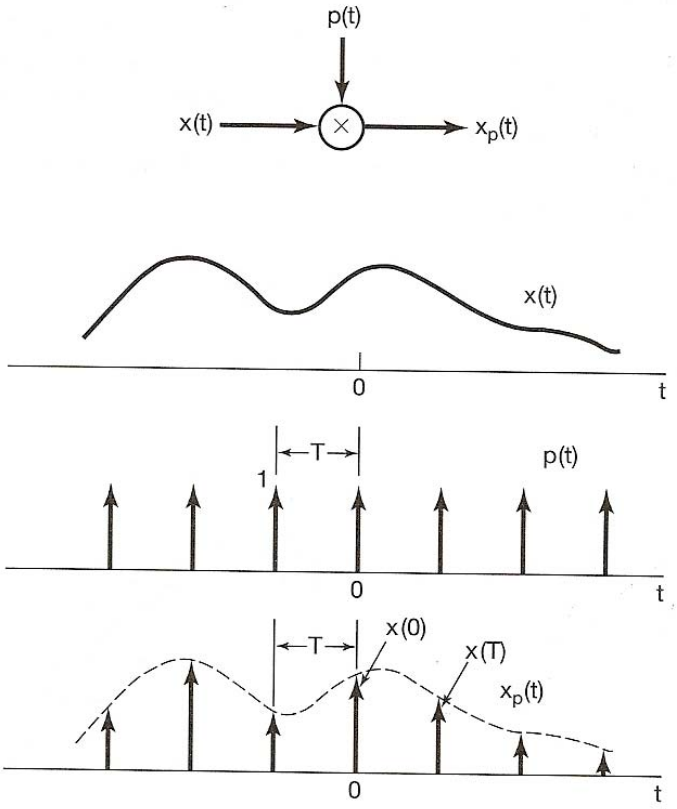


Figura (40)

Si nos fijamos en la Figura (40), p(t) es el tren de impulsos periódicos, x(t) la señal continua que queremos muestrear, T es el periodo de muestreo, $w_S = 2\pi/T$ la frecuencia fundamental y $x_p(t)$ la señal muestreada.

Por lo tanto en el dominio del tiempo tenemos:

$$x_p(t) = x(t)\,p(t) \tag{99}$$

Donde:

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{100}$$

Se sabe que al multiplicar x(t) por un impulso unitario se muestrea el valor de señal en el punto donde se encuentra el impulso, es decir:

$$x(t)\delta(t - t_0) = x(t_0)\delta(t - t_0) \tag{101}$$

Entonces si utilizamos esta propiedad y sustituimos p(t) en la ecuación (99), obtenemos un tren de impulsos con amplitudes iguales a las de x(t) en intervalos de distancia T, es decir:

$$x_p(t) = \sum_{n=-\infty}^{\infty} x(nT)\delta(t - nT) \tag{102}$$

Sea conocida la propiedad de la multiplicación:

$$r(t) = s(t)\,p(t) \overset{Fourier}{\leftrightarrow} R(jw) = \frac{1}{2\pi}[S(jw) * P(jw)] \tag{103}$$

La aplicamos a la ecuación (102) para obtener la función del tren de impulsos en el dominio de la frecuencia:

$$X_p(jw) = \frac{1}{2\pi}[X(jw) * P(jw)] \tag{104}$$

Si pasamos la ecuación (100) al dominio de la frecuencia obtenemos:

$$P(jw) = \frac{2\pi}{T} \sum_{k=-\infty}^{\infty} \delta(w - k \cdot w_s) \tag{105}$$

Sabemos que la convolución con un impulso desplaza la señal del siguiente modo:

$$X(jw) * \delta(w - w_0) = X(j(w - w_0)) \tag{106}$$

Por lo tanto si sustituimos la ecuación (105) en la (104) obtendremos lo siguiente:

$$X_p(jw) = \frac{1}{T} \sum_{k=-\infty}^{\infty} X(j(w - k \cdot w_s)) \tag{107}$$

Si nos fijamos en la ecuación (107) observamos que la señal muestreada en el espacio de fourier, es una función periódica de w que consiste en una superposición de réplicas de X(jw) desplazadas y escaladas por 1/T, como se muestra en la siguiente Figura.

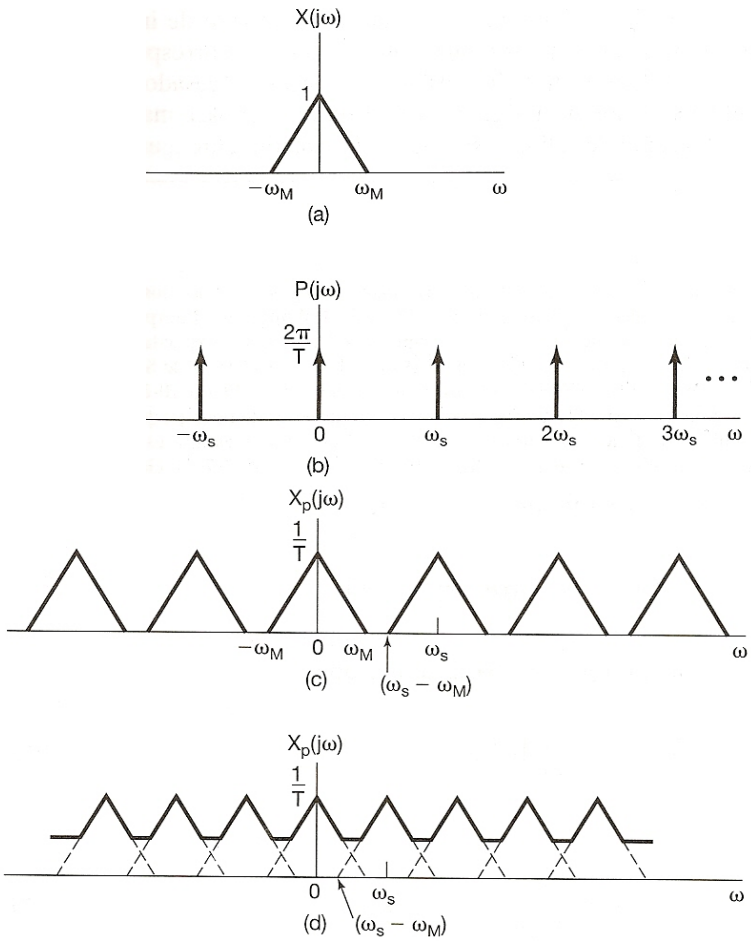


Figura (41)

Si nos fijamos en la Figura (41-c) , tenemos que $w_S < (w_S - w_M)$, o lo que es lo mismo $w_S > 2 \cdot w_M$, por lo que no hay solape entre las réplicas de X(jw), mientras que si nos fijamos en la Figura (41-d) como tiene $w_S < 2 \cdot w_M$ si que existe solape. Como consecuencia directa de esto se puede afirmar, que si $w_S > 2w_M$, entonces x(t) puede ser recuperada exactamente a partir de $x_p(t)$ por medio de un filtro paso bajo con ganancia T y una frecuencia de corte mayor que w_M y menor que $w_S - w_M$.
Resumiendo:

Teorema de muestreo:
"Sea x(t) una señal de banda limitada con $X(jw) = 0$ para $|w| > w_M$. Entonces x(t) se determina unívocamente mediante sus muestras x(nT), si

$$w_S > 2w_M \tag{108}$$

donde

$$w_S = \frac{2\pi}{T} \tag{109}$$

Dadas estas muestras, podemos reconstruir x(t) generando un tren de impulsos periódicos en el cual los impulsos sucesivos tengan amplitudes que correspondan a valores de muestras sucesivas. Este tren de impulsos es entonces procesado a través de un filtro paso bajo ideal con ganancia T cuya frecuencia de corte sea mayor que w_M y menor que $w_S - w_M$."

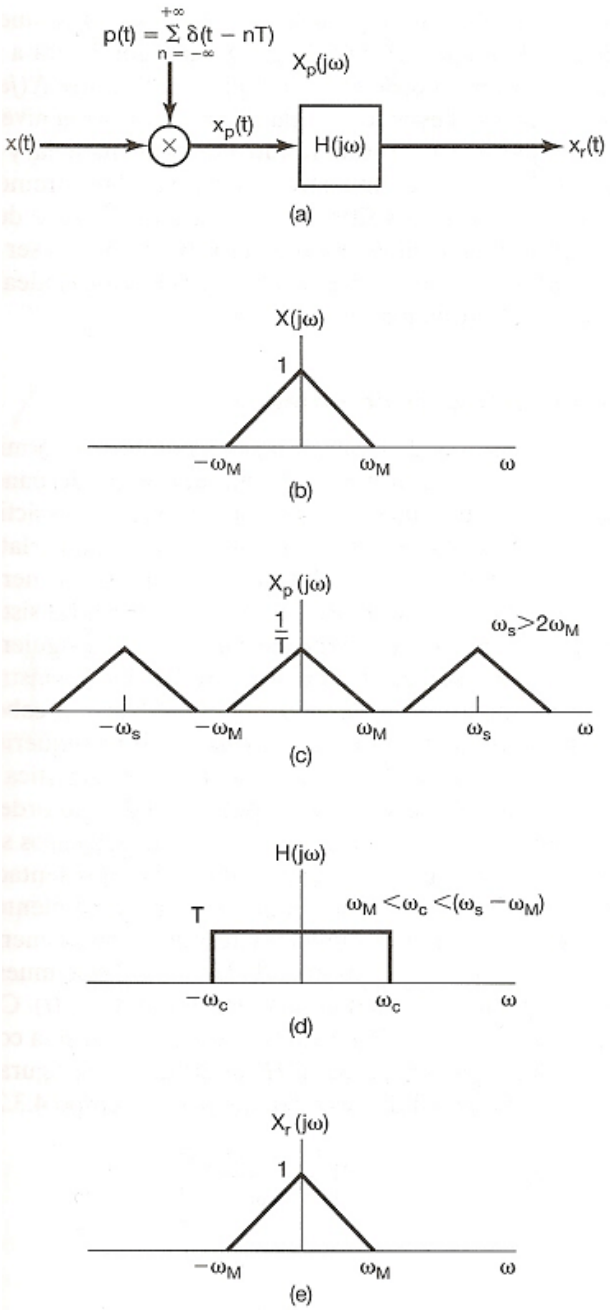


Figura (42)

En la Figura (42), mostramos como se puede recuperar una señal continua x(t) a partir de sus muestras $x_p(t)$ por medio de un filtro paso bajo ideal $H(jw)$ de amplitud T y frecuencia de corte $w_M < w_C < (w_S - w_M)$.

4.3.4.-Interpolación, muestreo y B-Splines

Una spline es una curva definida a trozos mediante polinomios, con todos los trozos suavemente conectados. Para obtener una representación continua de una señal discreta en una o más dimensiones se suele utilizar una spline. La concordancia puede ser exacta o aproximada. Aumentando el grado de la spline, pasamos gradualmente de una representación continua simple (interpolación lineal) a una representación continua caracterizada por un modelo de señal limitada en banda (spline de grado infinito).
La teoría de muestreo tradicional recomienda el uso de un filtro paso bajo no ideal cuando la señal de entrada no esté limitada en banda, a este filtro lo llamamos filtro anti-aliasing y en el caso de usar splines hay que sustituirlo por un filtro especificado por la reconstrucción.

Las curvas que más se usan son las B-splines debido a su eficiencia computacional derivada de su corto soporte. Se ha demostrado que las B-splines son las curvas con menor soporte para un grado de aproximación determinado. También se sabe que las B-splines cúbicas ofrecen un buen compromiso entre coste computacional y calidad de interpolación. Las B-splines son las funciones base preferidas gracias a su simple forma analítica, que hace que sean fáciles de manipular, a esto hay que añadir la interesante propiedad de que son maximalmente diferenciables y que sus derivadas se pueden calcular de forma recursiva.

■ Splines como funciones interpoladoras y funciones base

Las splines básicamente se utilizan para interpolar, (funciones base) para la reconstrucción de curvas y superficies suaves. Todas las explicaciones teóricas que vamos a realizar a partir de ahora se harán en 1D, para extenderlo al espacio bidimensional (2D) se hace utilizando el producto tensorial. Si $\varphi_{1D}(x)$ es una spline 1D, el producto 2D de una spline se define como:

$$\varphi_{2D}(x,y) = \varphi_{1D}(x) \cdot \varphi_{1D}(y) \tag{110}$$

■ Interpolación regularmente espaciada: Teorema generalizado del muestreo

Los polinomios no son las únicas funciones interpoladoras, si los puntos x_i están regularmente distribuidos, siendo $x_i = iT$ para un número entero i y un periodo de muestreo T, entonces Whittaker demostró que la serie

$$C(x) = \sum_{i=-\infty}^{\infty} y_i \sin c(\frac{x - x_i}{T}) \tag{111}$$

siendo

$$\sin c(x) = \frac{sen(\pi x)}{\pi x} \tag{112}$$

también interpola las muestras de entrada. C(x) es conocida como función cardinal. Shannon se dio cuenta de que esta representación era única para aquellas funciones cuya máxima frecuencia era inferior 1/2T Hz (como ya explicamos anteriormente en el teorema del muestreo). Este teorema no sólo sirve para las funciones limitadas en

banda, sino que se puede extender a un espacio mayor de funciones, como por ejemplo las funciones del espacio de Hilbert L_2:

$$L_2 = \left\{ f(x) : \Re \to \Re \mid \int_{-\infty}^{\infty} |f(x)|^2 dx < \infty \right\} \tag{113}$$

o todas las funciones que sean cuadrado integrables en el sentido de Lebesgue

$$\|f(x)\|^2 = \langle f, f \rangle < \infty \tag{114}$$

donde el producto interior entre dos funciones reales se define como

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(x) g(x) dx \tag{115}$$

El conjunto de funciones limitadas en banda, es un subconjunto de L_2, y el teorema generalizado de muestreo se define en este espacio del siguiente modo:

$$C(x) = \sum_{i=-\infty}^{\infty} c_i \cdot \varphi_i(x) \tag{116}$$

donde los c_i son una serie de coeficientes que se tienen que calcular a partir de la información de entrada (y_i) y $\varphi_i(x)$ es una versión desplazada de una función base:

$$\varphi_i(x) = \varphi(x - x_i) \tag{117}$$

En el caso particular de señales limitadas en banda, la función base utilizada es:

$$\varphi(x) = \sin c(\frac{x}{T}) \tag{118}$$

y el conjunto de todas las funciones limitadas en banda se abarca mediante la familia de funciones $\varphi_i(x)$. Dicho de otro modo, cualquier función limitada en banda, se puede expresar como una combinación lineal de un conjunto infinito de funciones $\varphi_i(x)$.
Si consideramos cualquier función φ en L_2 y el subespacio V generado por su translación de medida T

$$V = \left\{ f(x) = \sum_{i=-\infty}^{\infty} c_i \cdot \varphi_i(x) : c_i \in l_2 \right\} \tag{119}$$

donde l_2 es el espacio de todas las secuencias cuadrado sumables, entonces se puede probar de forma sencilla, que si un conjunto de funciones $\varphi_i(x)$ del espacio de Hilbert es ortonormal, entonces la proyección de cualquier función f de L_2 en el subespacio V, es

$$\tilde{f} = Pv\ f = \arg\min_{g \in v} \| f - g \| = \sum_{i=-\infty}^{\infty} \langle f, \varphi_i \rangle \varphi_i \tag{120}$$

Nótese que

$$\langle f, \varphi_i \rangle = \int_{-\infty}^{\infty} f(x)\varphi(x - x_i)dx = f(x) * \varphi(-x)\,|_{x=x_i} \tag{121}$$

el producto interior $\langle f, \varphi_i \rangle$ se puede calcular fácilmente muestreado la convolución lineal de $f(x)$ y $\varphi(-x)$ en $x = x_i$. De hecho, el filtro anti-aliasing utilizado para limitar la señal en banda antes de muestrearla corresponde a este producto interno.

El problema de usar $\varphi(x) = \sin c(x/T)$, es que la función sinc decae muy lentamente y las convoluciones que se necesitan hacer para calcular el producto interno son largas y por lo tanto poco prácticas. Por lo tanto lo que hay que hacer es buscar funciones que sean más atractivas computacionalmente hablando. Lo primero que se podría hacer, es relajar la condición de ortonormalidad y exigir solamente que el conjunto $\{\varphi_i\}$ defina una base de Riesz

$$A\|c_i\|_{l_2}^2 \le \left\| \sum_{i=-\infty}^{\infty} c_i \cdot \varphi_i(x) \right\|_{L_2}^2 \le B\|c_i\|_{l_2}^2 \quad \forall c_i \in l_2 \tag{122}$$

para dos constantes A y B dependientes de φ. La ortonormalidad es el caso especial en el que A=B=1. La condición de la izquierda asegura que las funciones base sean linealmente independientes, mientras que la condición de la derecha asegura que la norma de f esté limitada y por lo tanto el subespacio V es un subespacio válido de L_2.

Una importante restricción para las bases, es que sean capaces de representar cualquier función, con cualquier nivel de precisión deseado simplemente mediante la reducción de T. Esta condición se puede reformular como

$$\sum_{i=-\infty}^{\infty} \varphi_i(x) = 1 \quad \forall x \tag{123}$$

la suma de todas las funciones base es una función constante. La función $\varphi(x) = \sin c(x/T)$, cumple todas las condiciones comentadas hasta ahora, pero su soporte infinito hace que sea muy complicado calcular las convoluciones. De todos modos, hay otras funciones que también cumplen estos requisitos. La función más corta que los cumple, es la función rectangular:

$$\varphi(x) = \beta_0\left(\frac{x}{T}\right) = \begin{cases} 1 & |x| < T/2 \\ 0 & |x| > T/2 \end{cases} \tag{124}$$

que es la B-spline cardinal de grado 0.

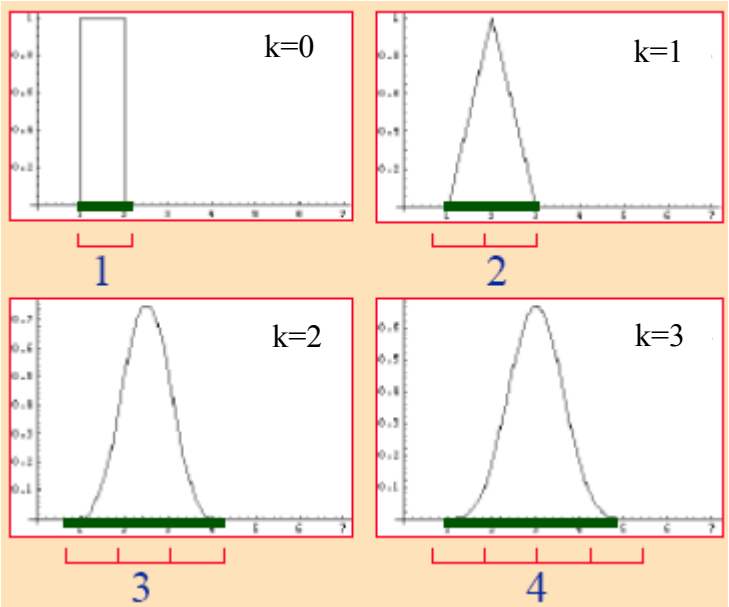


Figura (43)

La B-spline cardinal de orden n, se obtiene simplemente mediante la convolución de $\beta_0(x)$ consigo misma n veces. Por ejemplo la $\beta_1(x/T)$ es la función triangular definida entre –T y T, y $\beta_2(x/T)$ es una función parabólica definida entre –(3/2)T y (3/2)T. En general $\beta_n(x/T)$ es un polinomio simétrico definido a trozos de grado n cuyo dominio es $|x| < \frac{n+1}{2}T$. En las siguiente ecuaciones vamos a mostrar las B-splines cardinales de grados 1,2 y 3 para T=1. (fijarse en la Figura superior)

$$\beta_1(x) = \begin{cases} 1-|x| & |x| \le 1 \\ 0 & |x| > 1 \end{cases} \tag{125}$$

$$\beta_2(x) = \begin{cases} \frac{3}{4} - |x|^2 & |x| \le \frac{1}{2} \\ \frac{1}{2}\left(|x| - \frac{3}{2}\right)^2 & \frac{1}{2} < |x| < \frac{3}{2} \\ 0 & |x| > \frac{3}{2} \end{cases} \tag{126}$$

$$\beta_3(x) = \begin{cases} \frac{2}{3} + \frac{1}{2}|x|^2(|x|-2) & |x| \le 1 \\ \frac{1}{6}(2-|x|)^3 & 1 < |x| \le 2 \\ 0 & |x| > 2 \end{cases} \tag{127}$$

En general podemos expresar una B-spline cardinal de grado n como:

$$\beta_n(n) = \frac{1}{n!}\sum(-1)^k \binom{n+1}{k}\left(x-\left(k-\frac{n+1}{2}\right)\right)_+^n \tag{128}$$

Donde $$(x)_+^n = \begin{cases} x^n & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{129}$$

Las B-spline cardinales son polinomios construidos a trozos porque pueden representar mediante polinomios de grado n en $|x|$ que son diferentes para cada intervalo $iT \leq |x| < (i+1)T$. Todas estas funciones cumplen los requisitos anteriormente mencionados y además se pueden utilizar como funciones base para la representación de señales. Además su soporte en el espacio real es tan compacto que hace que los cálculos sean asumibles.
La representación de señales utilizando B-spline cardinales está íntimamente relacionada con la interpolación: el uso de β_0 es el equivalente a la interpolación de vecino más cercano, el uso de β_1 es el equivalente a la interpolación lineal. β_3 ha resulta ser un buen compromiso entre la complejidad de cálculo, el corto soporte y la aproximación de error. Por lo tanto el teorema de muestreo resulta ser un caso particular de interpolación y como hemos visto se puede generalizar para el espacio L_2, en vez de estar restringido a las señales limitadas en banda.

La pregunta ahora es: ¿cómo hacemos para calcular los coeficientes de representación c_i, de la ecuación (116)? Ya hemos visto una posibilidad para el caso de bases ortonormales en la ecuación (121), no obstante como norma general las B-spline cardinales no son ortonormales (excepto las de grado 0). Por lo general el cálculo de los coeficientes c_i es similar al caso ortonormal, la única diferencia es que en vez de usar la propia base φ_i, tenemos que usar una base dual $\tilde{\varphi}_i$.

$$\tilde{f} = Pv\ f = \sum_{i=-\infty}^{\infty} \langle f, \tilde{\varphi}_i \rangle \varphi_i \tag{130}$$

La base dual se define de forma única mediante la condición de biortogonalidad $\langle \tilde{\varphi}_i, \varphi_i \rangle = \delta_{i-j}$ y además hereda la propiedad de desplazamiento invariante de la función original $\tilde{\varphi}_i(x) = \tilde{\varphi}(x - x_i)$. Sin embargo, seguimos sin tener una forma clara de calcular las c_i porque no tenemos una fórmula general para calcular las bases duales. Se puede demostrar [1] que la transformada de Fourier de una $\tilde{\varphi}$ viene dada por:

$$\hat{\tilde{\varphi}}(j\Omega) = \frac{\hat{\varphi}(j\Omega)}{\sum_{k=-\infty}^{\infty} |\hat{\varphi}(j(\Omega + 2\pi k))|^2} \tag{131}$$

donde $\hat{f}(j\Omega)$ representa la transformada continua de Fourier de la función $f(x)$.

Afortunadamente Unser y sus colaboradores [2,3] obtuvieron una forma muy eficiente de calcular estos coeficientes utilizando filtros digitales. Actualmente ésta es la forma que se utiliza para calcular los coeficientes, en el caso de imágenes estos filtros se ejecutan en cada columna de la imagen, produciendo una nueva imagen de coeficientes. Luego se vuelven utilizar los filtros en cada fila de la nueva imagen para producir los coeficientes 2D del producto tensorial B-spline. Una vez que los coeficientes han sido generados, las imágenes se tratan como si fueran funciones continuas, además se guardan como si fueran un conjunto de coeficientes de B-spline cardinales.

Ahora nos preguntamos si podríamos diseñar un tipo de funciones base basadas en las B-spline y cuyos coeficientes cumplan $c_i = y_i$. Ésta sería una spline interpoladora y estaríamos ante una situación similar a la de la interpolación del teorema de Muestreo que mostrábamos en la ecuación (111). La siguiente función sería la de una spline interpoladora:

$$\varphi_{\text{int}}(x) = \sum_{i=-\infty}^{\infty} q_{\text{int}}[i]\varphi_i(x) \tag{132}$$

donde $q_{\text{int}}[i]$ es la secuencia l_2 definida como la inversa de la transformada de Z de

$$Q_{\text{int}}(z) = \frac{1}{\sum_{k=-\infty}^{\infty} \varphi(kT) z^{-k}} \tag{133}$$

siendo

$$\varphi(x) = \beta_n\left(\frac{x}{T}\right) \tag{134}$$

■ Error de aproximación

Un concepto importante relacionado con el teorema del muestreo que acabamos de explicar es: ¿Cómo de bien somos capaces de representar una función f cualquiera?. Vamos a llamar f_T a la aproximación obtenida al muestrear con periodo T. Ahora este problema se va a estudiar mediante la teoría de aproximación, que demuestra que los siguientes tres estamentos son equivalentes:

1. Si f es una función lo suficientemente suave (f pertenece a espacio de Sobolev W_2^L, es decir sus primeras L derivadas pertenecen al espacio L_2), entonces según T tiende a cero, existe una constante C independiente de f tal que: $\|f - f_T\| \le CT^L \|f^{(L)}\|$

2. Los primeros L momentos de φ son constantes. Siendo el momento

$$\sum_{i=-\infty}^{\infty} (x - x_i)^m \varphi_i(x) = \mu_m \tag{135}$$

para $m = 0,1,2,3,.....,L-1$

3. Los primeros L monomios se pueden representar de forma exacta, para cada monomio x^m $(m = 0,1,..., L-1)$ existen constantes $c_i \in l_2$ tal que

$$x^m = \sum c_i \varphi_i(x) \tag{136}$$

Una consecuencia importante que deriva del resultado anterior es que el error de aproximación de las diferentes funciones base depende sobre todo de su diseño, es decir, dados dos polinomios del mismo grado, uno de ellos puede que aproxime las funciones suaves de forma más rápida que el otro. A L lo llamamos orden de aproximación, y depende totalmente de los momentos de φ o del número de monomios que pueden ser representados, las B-splines de grado n tienen un orden de aproximación de L=n+1.

Otra importante consecuencia es que para que la función f converja cuando $T \to 0$, la función base φ debe tener $L \geq 1$. Como bien es sabido las sinc filtradas no cumplen esta condición, por lo tanto ahora debemos intentar diseñar una familia φ de modo que tengan un orden de aproximación L con el mínimo soporte posible. Así es como se diseñaron el conjunto de funciones MOMS (Maximal-Order interpolation of Minimal Support) que no son más que combinaciones lineales de las B-spline de orden L y sus derivadas.[4]

4.3.5.-Implementación y resultados

Ahora se va a implementar la solución simulada en Matlab en C++ utilizando el entorno de desarrollo eclipse sobre un sistema operativo Linux SUSE. La programación se va a orientar a la rapidez de procesamiento, ya que la versión en Matlab tardaba dos semanas en terminar de procesar el volumen de prueba (ocho veces más pequeño que el volumen real). Por lo tanto vamos a optimizar el código, el cambio más importante, es que ahora no se va a dividir el programa en funciones, si no que se va a programar todo de forma seguida en el mismo fichero. Esto se hace así porque, aunque quede más ordenado y se entienda mejor al separar el programa en funciones, cada vez que se llama a una función hay un cambio de contexto que conlleva una serie de operaciones (como guardar el estado y las variables de la función que esté en ejecución y cargar la nueva función) que consumen una gran cantidad de tiempo.

Vamos a llamar a nuestra función "enhance_contrast", la definición de la misma la pueden ver en el fichero enhance_contrast.h *(ver Apéndice C.3)*. Y su implementación se encuentra en el fichero enhance_contrast.cpp *(ver Apéndice C.4)*. Si nos fijamos en este último, vemos que cada paso está numerado. Lo primero que hacemos es **escalar** la imagen 3D de grises entre 0 y 255 y guardamos el resultado en "PPP1_init.vol".

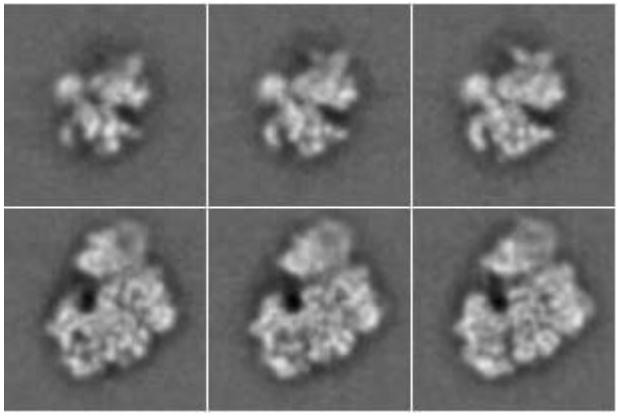


Figura (45)

En la Figura (45) vemos 6 cortes hechos al volumen "PPP1_init.vol" para la molécula entera *mirar Apéndice C.5*.

El siguiente paso que se lleva a cabo es **eliminar el ruido de fondo** o background, a pesar de que anteriormente hemos comentado que íbamos a implementar todo el programa de forma seguida, sin separarlo en funciones, aquí tenemos una excepción. El programa que elimina el background resulta ser una herramienta que puede ser útil en otros casos, así que lo vamos a implementar como una función a parte para que pueda ser usada de forma independiente por otros programadores que pueden llamarla desde la librería. Esta función se va ha llamar "detect_background", su declaración se va ha encontrar en el fichero "Filters.h" *(ver Apéndice C.1)* y su implementación en "filters.cpp" *(ver Apéndice C.2)*. Primero se crea un cubo "bg" que nos va a servir para controlar los píxeles que vamos procesando, este cubo va a valer "-1" en los píxeles que no hayamos visitado, "0" en los píxeles que consideremos de la molécula, "1" en los píxeles que consideremos background y "-2" para los píxeles que estén dentro de la lista por procesar. Luego creamos una lista que se va ha llamar "list_for_compute" y que va a contener las coordenadas de los píxeles que hay que computar. El siguiente paso es llenar la lista anterior con los píxeles que conforman las 6 caras del cubo, ya que a estos

los consideramos píxeles de ruido al encontrarse la molécula en el interior del cubo. Una vez hecho esto se calculan los estadísticos de lo que hemos considerado ruido para realizar un intervalo de confianza según lo explicado ya en los apartados 4.2.2.1 y 4.2.2.2. Con el intervalo de confianza ya podemos discriminar lo que consideramos ruido de lo que no, no obstante (cada 250 píxeles) según vamos detectando más píxeles de ruido, como tenemos más información sobre el ruido, volvemos a calcular los estadísticos, para recalcular el intervalo de confianza y ser así más precisos. Cada vez que consideramos que un píxel es de ruido comprobamos todos sus vecinos, y así consecutivamente, avanzando por lo tanto desde las caras del cubo hacia el interior del mismo. Con este procedimiento generamos un cubo que tiene a "1" los píxeles que hemos considerado background y a "0" los que consideramos molécula. Ahora procesamos este cubo utilizando la operación de cierre de morfología matemática (explicado en los apartados 4.2.2.4 y 4.2.2.5) para eliminar ciertos píxeles de ruido aislados que hemos considerado molécula. El resultado se graba en "PPPmask.vol".

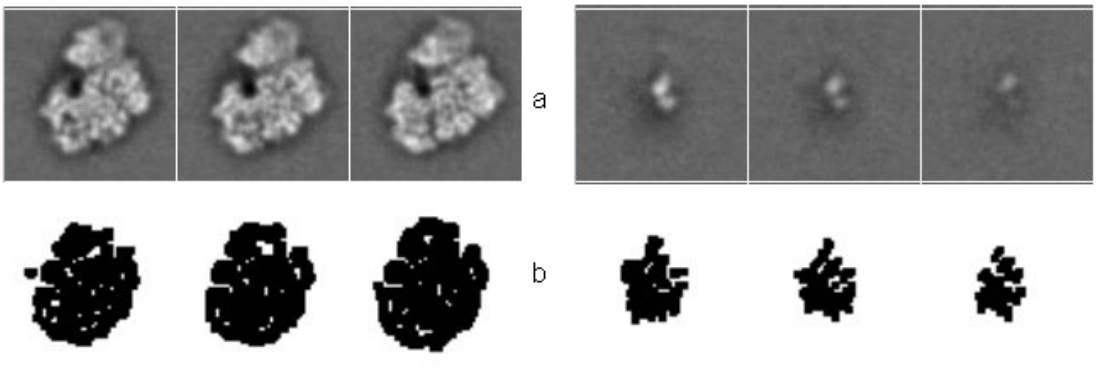


Figura (46)

En la Figura (46), podemos ver primero los cortes pertenecientes a la molécula sin procesar (imágenes a) y debajo de las mismas vemos cortes de la máscara (imágenes b) donde vemos de color negro los píxeles que hemos considerado que pertenecen a la molécula, y de blanco los que hemos considerado ruido de fondo. Si nos fijamos, se puede apreciar como están de color negro los píxeles en las imágenes (b) que parecen formar parte de la molécula en las imágenes (a). Los 64 planos del volumen "PPPmask.vol" se encuentran en el *Apéndice C.6.*

Lo siguiente que hacemos es eliminar del cubo original el ruido fondo, para ello usamos la máscara para saber que píxeles hemos determinado como ruido, hacemos el promedio de todos ellos y ponemos todos esos píxeles a ese valor.

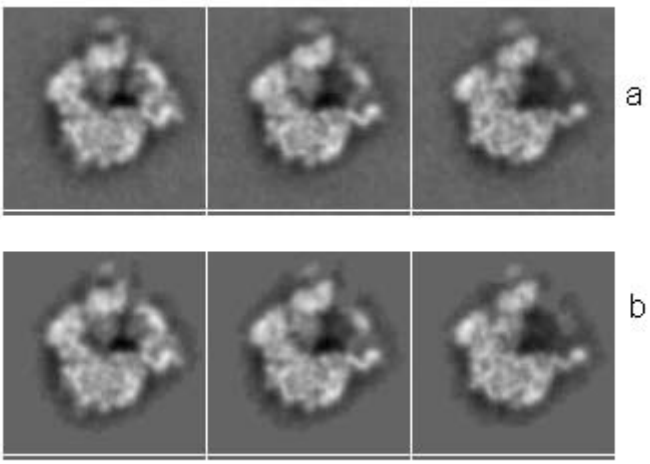


Figura (47)

En la Figura (47-a) podemos ver tres planos de la molécula con ruido de fondo, y en la (47-b) vemos como ha desaparecido casi todo el ruido fondo después de procesar la imagen. Para ver los 64 planos de la molécula sin ruido de fondo ir al *Apéndice C.7.*

La tercera fase es calcular **dónde se encuentran los bordes** y para eso utilizamos las B-splines (apartado 4.3.4) para interpolar la función que ha generado los píxeles de la molécula, luego se deriva la función en las tres dimensiones (x,y,z) como vimos en apartado 4.1.2.-Sobel y eso nos da los bordes de molécula.

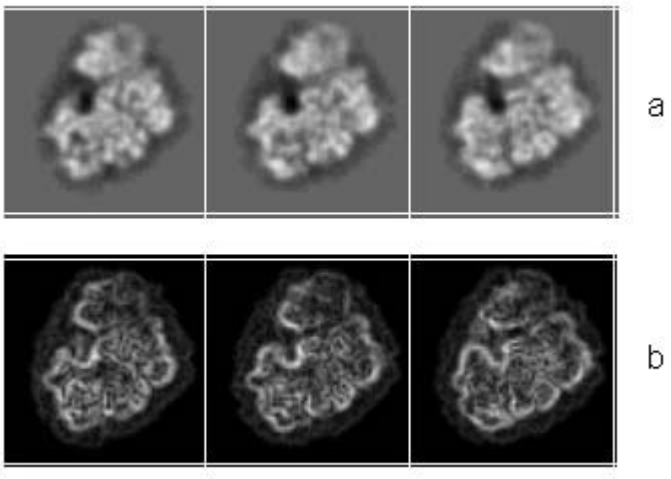


Figura (48)

En la Figura (48-a) vemos tres planos de la molécula, y en la Figura (48-b) vemos el resultado de calcular los bordes mediante la derivada de la función interpolada con las B-splines. Como se puede ver a simple vista el resultado es muy satisfactorio y los resultados son superiores a los obtenidos anteriormente mediante sobel. Para ver los 64 planos del volumen que contiene los bordes ir al *Apéndice C.8.*

El siguiente paso es calcular el **Mean Edge Gray Value** ya explicado en el apartado 4.1.4., para ello vamos a emplear un vecindario variable (apartado 4.2.4.1.) utilizando un intervalo de fianza para determinar hasta donde crecer (apartado 4.2.4.2.).
Este procedimiento, primero genera un volumen (vol_tam) que contiene el tamaño de vecindario que se ha de usar para cada píxel.

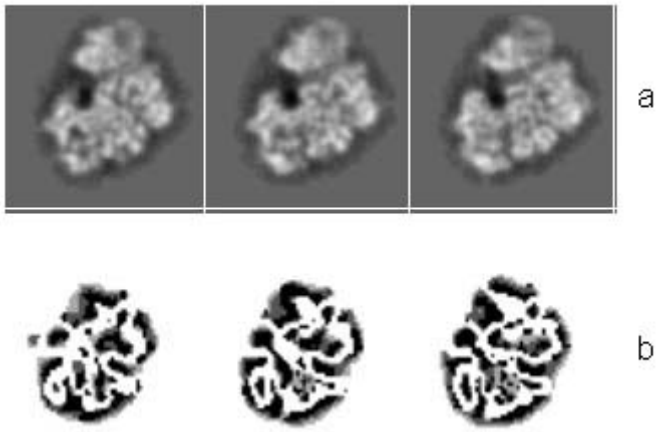


Figura (49)

En la Figura (49-a) tenemos tres planos de la molécula, y en la Figura (49-b) tenemos los tamaños de vecindarios que hay que tomar para cada píxel, los tonos claros o blancos son los vecindarios de mayor tamaño, es decir donde todos los píxeles son parecidos o siguen un patrón similar, como por ejemplo todo el ruido del exterior. Los tonos oscuros o negros, son los vecindarios de menor tamaño, es decir donde los píxeles no son parecidos, como por ejemplo en el interior, que es donde se encuentra la molécula. Para ver los 64 planos del volumen que contiene los tamaños del vecindario ir al *Apéndice C.9.* Una vez que tenemos los tamaños de vecindario, aplicamos la ecuación (2) a esos vecindarios para obtener el Mean Edge Gray Value.

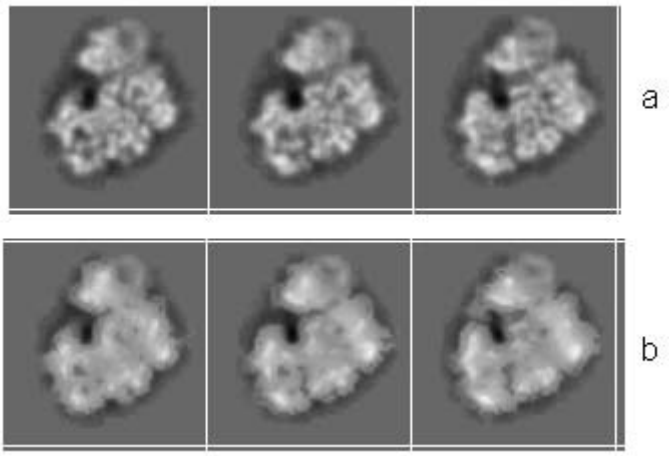


Figura (50)

En la Figura (50-a) mostramos tres planos de la molécula, y en la Figura (50-b) se ve el resultado de aplicar el MEGV a los mismos. Para ver los 64 planos del volumen con los valores del MEGV ir al *Apéndice C.10.*

Finalmente aplicamos la **función no lineal**, cuya teoría ya explicamos en el apartado 4.1.4. para mejorar el contraste en la imagen 3D, el resultado se guarda en PPP6_vol_f.vol.

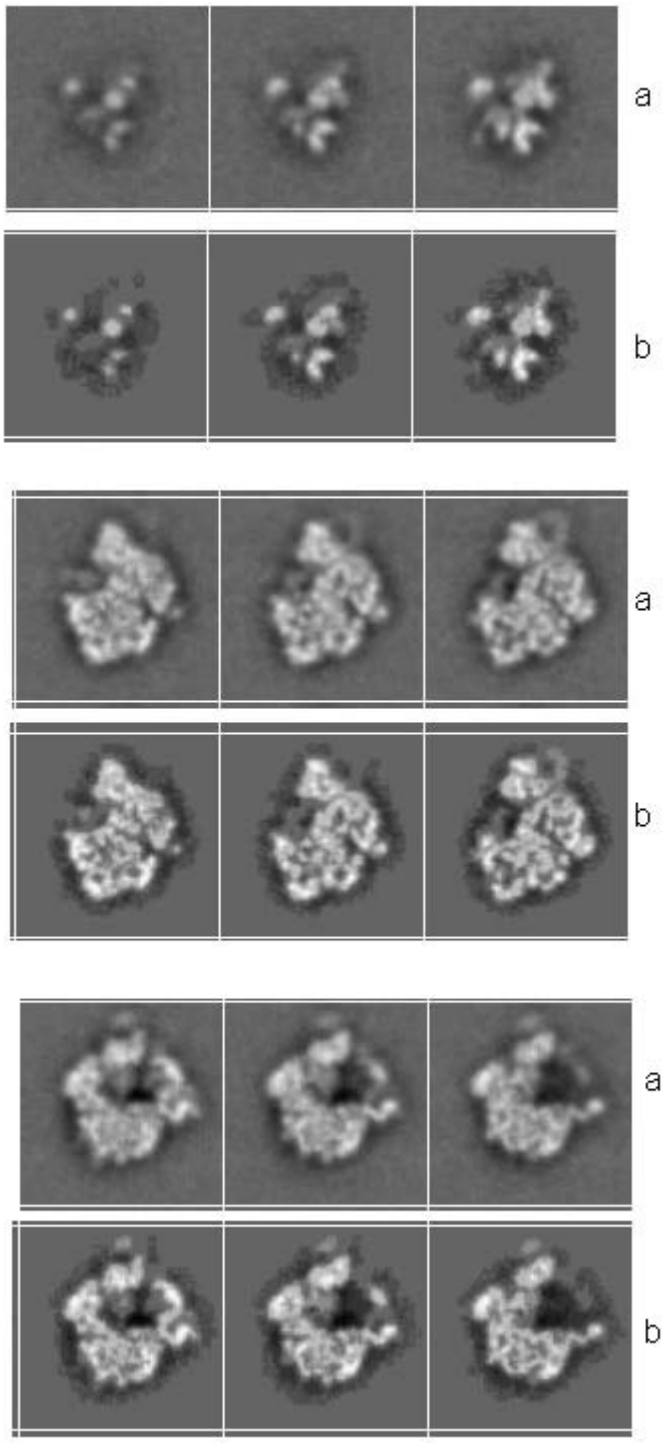


Figura (51)

Si nos fijamos en la Figura (51), tenemos la molécula sin procesar en las imágenes (a) y debajo, en las imágenes (b) la procesada. A simple vista podemos ver como el contraste es mejor en las imágenes (b). Luego el programa funciona correctamente. Para ver los 64 planos del volumen procesado ir al *Apéndice C.11.*

4.3.6.-Batería de pruebas

Una vez comprobado el programa con una imagen reducida de 64x64x64 píxeles (las imágenes mostradas hasta ahora), luego hemos pasado a realizar una batería de pruebas con 40 moléculas distintas de 128x128x128 píxeles, y los resultados han sido muy positivos en todas ellas.

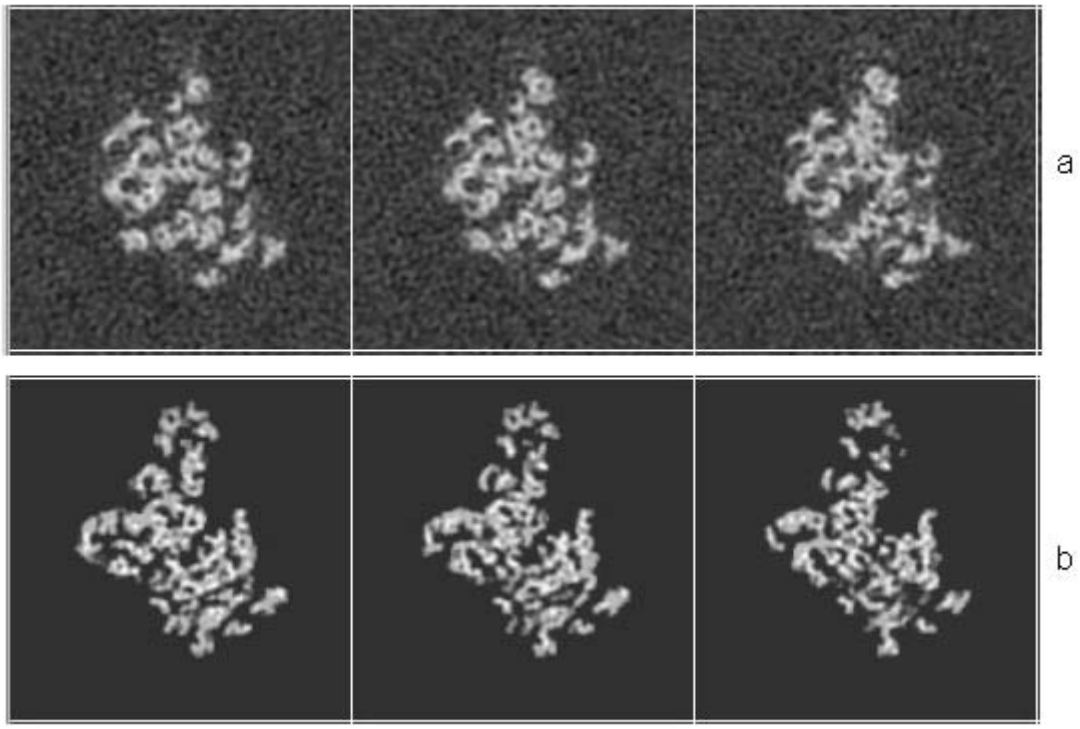


Figura (52)

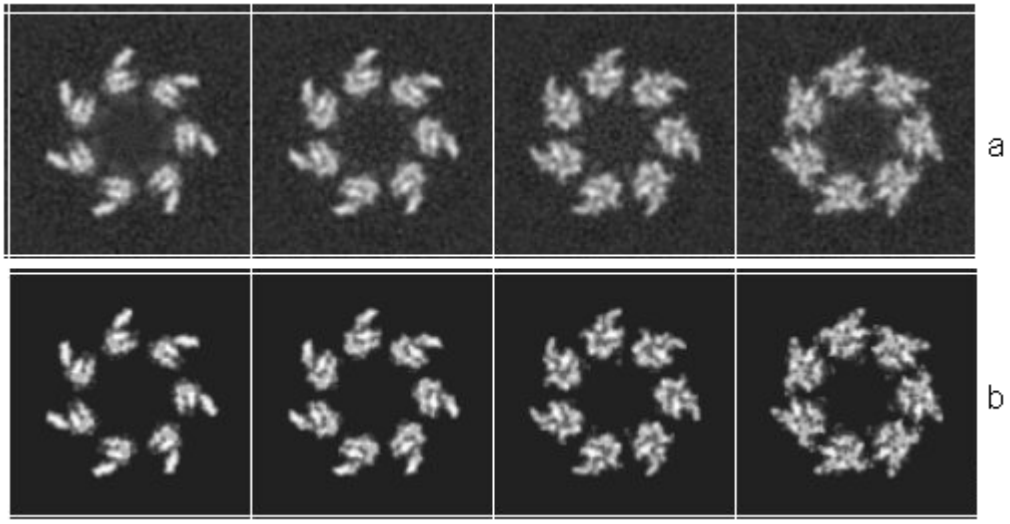


Figura (53)

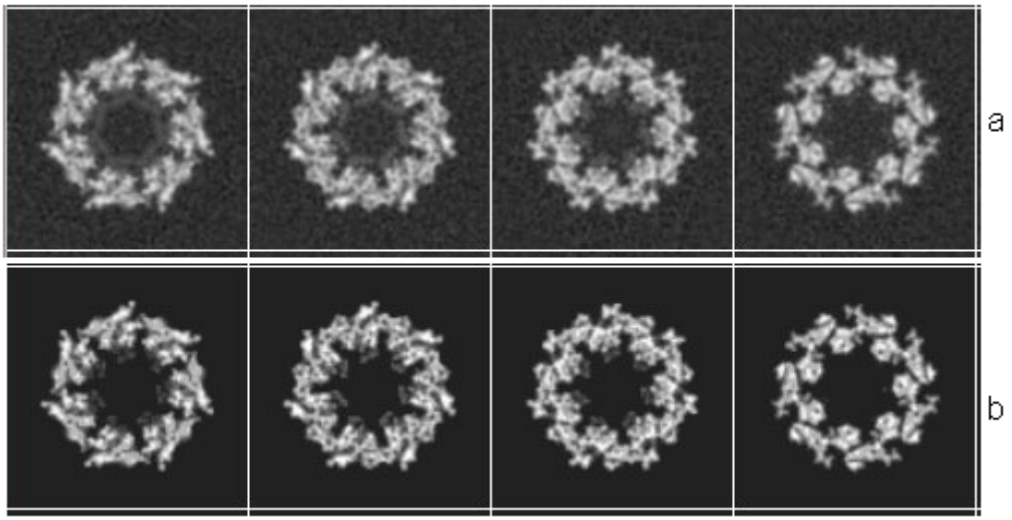


Figura (54)

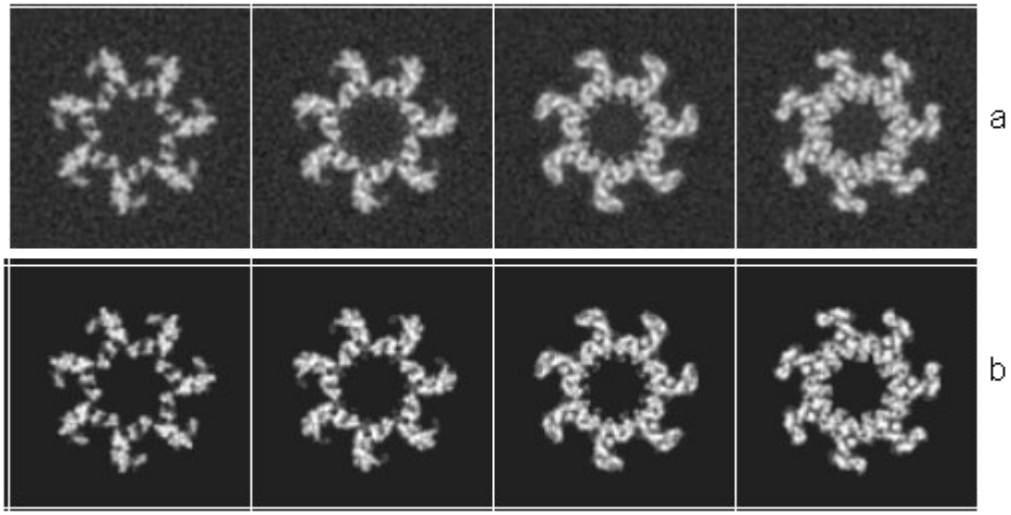


Figura (55)

En las Figuras 52 a 55, encontramos distintos ejemplos de moléculas procesadas, donde la imagen (a) es siempre la original y la (b) la procesada por nuestro programa. En todos los casos observamos una clara mejora.

4.3.7.-Fourier Shell Correlation

Ahora vamos a utilizar un método para comprobar de forma objetiva que nuestro programa mejora la imagen. Para ello vamos a partir de una imagen simulada que es perfecta, a esta imagen perfecta le vamos a añadir el tipo de ruido que introduce un microscopio electrónico. Luego procesamos la imagen con ruido para obtener nuestra imagen mejorada.

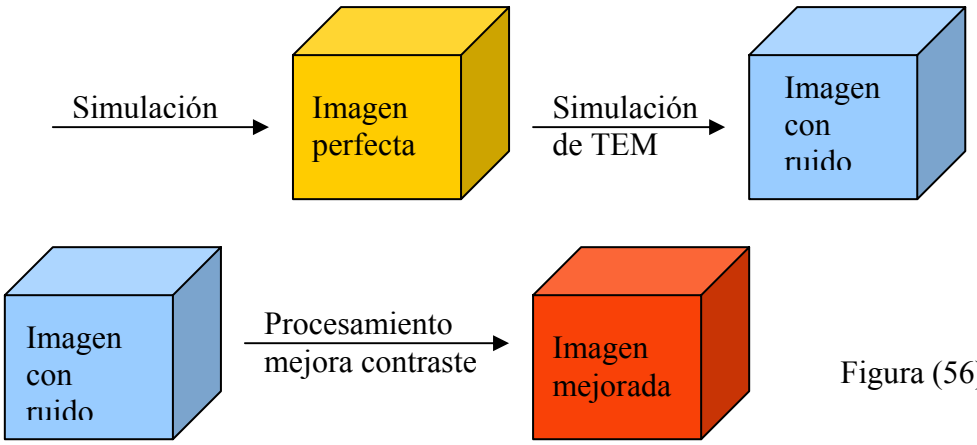


Figura (56)

A partir de este punto, vamos a utilizar la FSC (Fourier Shell Correlation) para comparar las imágenes, la FSC nos proporciona una curva que nos indica la correlación a lo largo de la frecuencia. Por lo tanto si nuestra imagen mejorada correla mejor con la perfecta que la imagen con ruido, entonces podemos afirmar de forma objetiva que hemos mejorado la imagen.

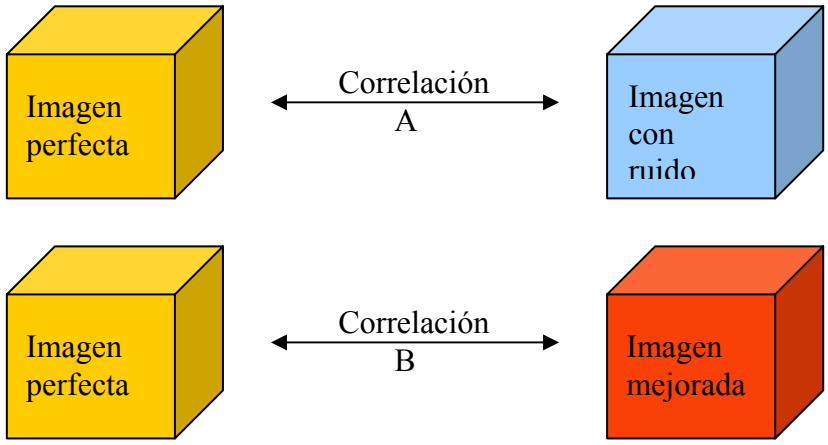


Figura (57)

Fijándonos en la Figura (57), si la curva de la correlación B, es mayor que la curva de correlación A, entonces significa que nuestra imagen mejorada se ha aproximado más a la imagen perfecta. Conocemos la siguiente propiedad de la correlación:

$$Corr\{f_1(r), f_2(r)\} = TF^{-1}\{F_1(R)F_2^*(R)\} \qquad (142)$$

Siendo "r" un vector del espacio tridimensional en el tiempo, y "R" un vector tridimensional en la frecuencia. De forma equivalente, podemos escribir:

$$\rho_{f1,f2}(r_0) = \int_{\Re_3} f_1(r)\, f_2(r+r_0)\, dr = \int_{\Re_3} F_1(R)F_2^*(R)\cdot e^{j\langle R,r_0\rangle} dR \tag{143}$$

Si ahora lo pasamos al espacio discreto, obtenemos:

$$\sum_r f_1(r)\, f_2(r+r_0) = \sum_R F_1(R)F_2^*(R)e^{j\langle R,r_0\rangle}\Delta R \tag{144}$$

Ahora vamos a normalizar la ecuación (144) para que quede comprendida entre 1 y -1:

$$\frac{\rho_{f1,f2}(r_0)}{\sqrt{\rho_{f1,f1}(0)\rho_{f2,f2}(0)}} = \frac{\sum_R F_1(R)F_2^*(R)e^{j\langle R,r_0\rangle}\Delta R}{\sqrt{\sum_R \|F_1(r)\|^2 e^{j\langle R,0\rangle}\Delta R \sum_R \|F_2(r)\|^2 e^{j\langle R,0\rangle}\Delta R}} = \tag{145}$$

En el denominador nos queda ΔR^2 , que al salir fuera de la raíz, se elimina con el ΔR del numerador, y como un producto vectorial por cero es cero las exponenciales del denominador quedan como 1:

$$= \sum_R \frac{F_1(R)F_2^*(R)}{\sqrt{\sum_R \|F_1(r)\|^2 \sum_R \|F_2(r)\|^2}} e^{j\langle R,r_0\rangle} = \sum_{\|R\|}\sum_{R\in Shell} \frac{F_1(R)F_2^*(R)}{\sqrt{\sum_R \|F_1(r)\|^2 \sum_R \|F_2(r)\|^2}} e^{j\langle R,r_0\rangle} = \tag{146}$$

Ahora eliminamos la exponencial del numerador evaluando la ecuación en el origen, es decir para $r_0 = 0$. Y evaluamos el sumatorio, para una mitad ($R_X \geq 0$).

$$\begin{aligned} &= \sum_{\|R\|}\sum_{R\in Shell} \frac{F_1(R)F_2^*(R)}{\sqrt{\sum_R \|F_1(r)\|^2 \sum_R \|F_2(r)\|^2}} = \sum_{\|R\|}\sum_{Rx\geq 0} \frac{F_1(R)F_2^*(R) + F_1(-R)F_2^*(-R)}{\sqrt{\sum_R \|F_1(r)\|^2 \sum_R \|F_2(r)\|^2}} = \\ &= \sum_{\|R\|}\sum_{Rx\geq 0} \frac{F_1(R)F_2^*(R) + F_1^*(R)F_2(R)}{\sqrt{\sum_R \|F_1(r)\|^2 \sum_R \|F_2(r)\|^2}} = \sum_{\|R\|}\sum_{Rx\geq 0} \frac{2\,\mathrm{Re}\{F_1(R)F_2^*(R)\}}{\sqrt{\sum_R \|F_1(r)\|^2 \sum_R \|F_2(r)\|^2}} = \end{aligned} \tag{147}$$

Ahora vamos a intentar eliminar el "2" que multiplica en el numerador, demostrando que el valor de la mitad que no hemos evaluado ($R_X < 0$) es igual al de la mitad evaluada:

$$\begin{aligned} &\sum_{Rx\leq 0} \mathrm{Re}\{F_1(R)F_2^*(R)\} \xrightarrow{R'=-R} \sum_{Rx'\geq 0} \mathrm{Re}\{F_1(-R')F_2^*(-R')\} = \\ &= \sum_{Rx\geq 0} \mathrm{Re}\{F_1^*(R)F_2(R)\} = \sum_{Rx\geq 0} \mathrm{Re}\{F_1(R)F_2^*(R)\} \end{aligned} \tag{148}$$

De modo que continuando el desarrollo (147), nos queda que la FSC se calcula:

$$FSC(R, Shell) = \sum_{\|R\|}\sum_{R\in Shell} \frac{\mathrm{Re}\{F_1(R)F_2^*(R)\}}{\sqrt{\sum_R \|F_1(r)\|^2 \sum_R \|F_2(r)\|^2}} \tag{149}$$

Como el primer proceso de denoising (4.2.2.) realmente no mejora la calidad de imagen, si no que elimina el ruido que rodea a la molécula, vamos a modificar nuestro programa para poder saltarnos la fase de denoising.

Ahora cogemos una imagen tridimensional simulada, aplicamos un algoritmo que introduce ruido, y luego mejoramos el contraste utilizando nuestro método. Una vez que tenemos todas las imágenes calculamos la FSC con un programa diseñado para ello, y el resultado obtenido es el siguiente:

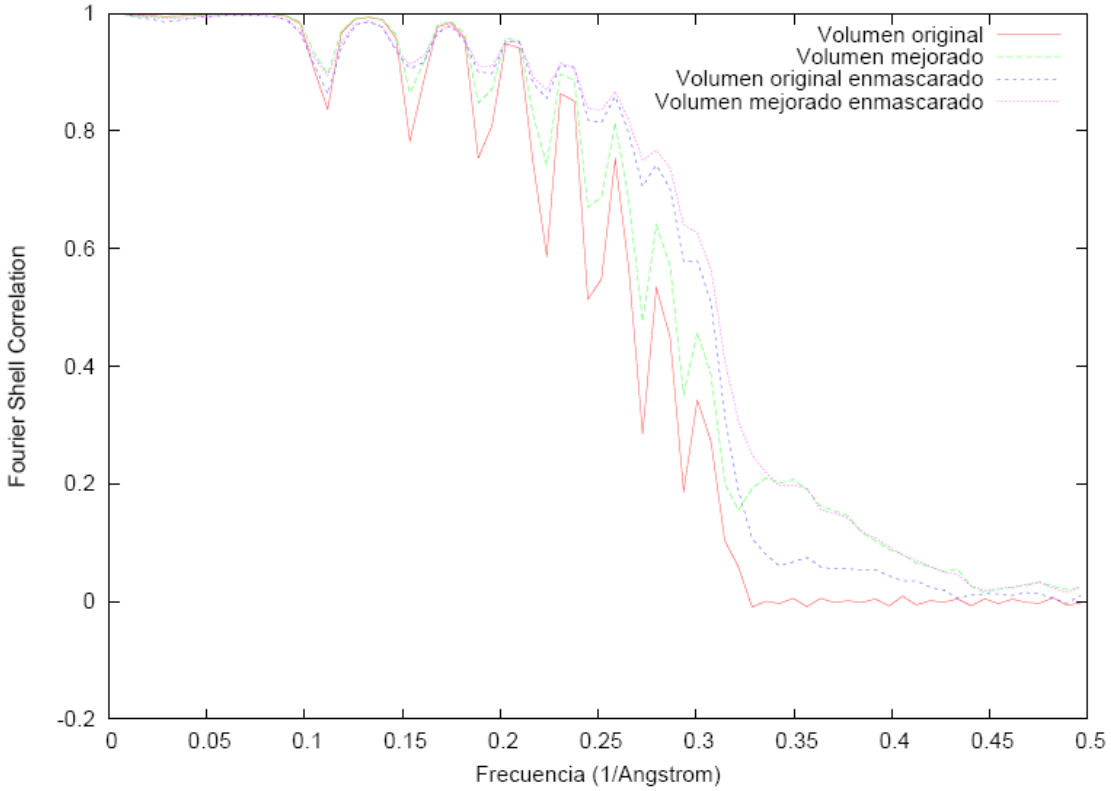


Figura (58)

Si nos fijamos en la Figura (58), podemos comparar el volumen con ruido (línea roja) con nuestro volumen mejorado sin aplicar el denoising (línea verde), podemos ver como la línea verde siempre es superior a la roja, lo que indica que nuestra imagen mejorada correla mejor con la imagen perfecta. Por lo tanto nuestra metodología mejora la calidad de la imagen.

4.3.8.-Conclusiones

En este PFC hemos intentado crear nueva una metodología de tratamiento digital de imágenes para la mejora del contraste, partiendo de un artículo [6] y luego añadiendo ideas propias. Entre las ideas que hemos aplicado, se encuentra un enmascaramiento para eliminar el ruido, mediante el uso de intervalos de confianza y matemática morfológica, la utilización de vecindarios de tamaño variable, operar trabajando en 3D y el uso de B-Splines para interpolar funciones y así obtener derivadas más precisas.

Todo esto ha sido implementado en diversos formatos, y hemos comprobado los resultados utilizando el método Fourier Shell Correlation, el cuál ha corroborado de forma objetiva que nuestra nueva metodología efectivamente mejora la imagen.

5.-APÉNDICES

A.1.-main.m

```
function [V,V_S,V_E,V_F]=main()
% Programa principal
% Aumento de contraste no lineal basado en la escala de Munsell
% Este ejercicio se va a simular en 2D
% Fernando Fuentes Arija , Julio 2009

%introducimos el nombre del volumen que queremos leer (no funciona)
nombre=input('Introduzca el nombre del volumen a leer: ','s');
% El modo depende de como el microprocesador ordena los bytes
% puede ser ieee-le o ieee-be.
modo='ieee-be';
V=open_volume(nombre,modo);

% comprobamos el tamaño
[x y z]=size(V);
% Busco el máximo
maximo=busco_max(V);
% Busco el mínimo
minimo=busco_min(V);

%Salvamos el volumen
save('volumen','V');

% escalamos el volumen para que todos los voxel estén entre 0 y 250
V=escalado(V,maximo,minimo,255,0);

% Aplicamos los operadores de Sobel
V_S=solbel_2D(V);

% Calculamos Eij  "mean edge gray value"
V_E=mean_edge_gray_value(V,V_S);

%Aplicamos la función no lineal
V_F=funcion_no_lineal(V,V_E);

%Salvamos los resultados
save('resultados','V','V_S','V_E','V_F');
```

A.2.-busco_max.m

```
function maximo=busco_max(V)
% Función
% Esta función busca el máximo dentro de un volumen

% sacamos el tamaño
[x y z]=size(V);
% variable que guarda el máximo
maximo=V(1,1,1);

for i=1:x

    for j=1:y

        for k=1:z
            num=V(i,j,k);
            if num > maximo
               maximo=num;
            end % del if
        end % del for de k

    end % del for de j

end % del for de i
```

A.3.-busco_min.m

```
function minimo=busco_min(V)
% Función
% Esta función busca el máximo dentro de un volumen

% sacamos el tamaño
[x y z]=size(V);
% variable que guarda el máximo
minimo=V(1,1,1);

for i=1:x

    for j=1:y

        for k=1:z
            num=V(i,j,k);
            if num < minimo
               minimo=num;
            end % del if
        end % del for de k

    end % del for de j

end % del for de i
```

A.4.-escalado.m

```
function V=escalado(V,x_max,x_min,y_max,y_min)
% Función
% Esta función escala un dominio V que va desde x_min hasta x_max a
% otro dominio que va de y_min a y_max. En este caso el dominio es 3D
% El cambio de dominio se hace mediante una transformación lineal del
% tipo y=a+bx.
% Fernando Fuentes Arija , Julio 2009

% sacamos el tamaño
[x y z]=size(V);

% Calculamos a
num_a=y_max-((y_min*x_max)/x_min);
den_a=1-(x_max/x_min);
a=num_a/den_a;

%Calculamos b
b=(y_max-y_min)/(x_max-x_min);

for i=1:x

    for j=1:y

        for k=1:z
            % cogemos el valor
            num=V(i,j,k);
            % aplicamos el escalado
            num_new=a+b*num;
            % guardamos los cambios
            V(i,j,k)=num_new;

        end % del for de k

    end % del for de j

end % del for de i
```

A.5.-sobel_2D.m

```
function V_S=solbel_2D(V)
% Función
% Esta función recibe un volumen V lo divide en planos 2D, y aplica
% los operadores de Solbel para sacar los bordes y devuelve un
% volumen V_S ya procesado
% Fernando Fuentes Arija , Julio 2009

%creamos el volumen que vamos a devolver procesado
V_S=V;

% sacamos el tamaño
[x y z]=size(V);

% vamos a fijar Z y sacar todos los planos (x,y)
for i=1:z
    % sacamos un plano numero i
    plano=sacarplano(3,i,V);
    % utilizamos una función que me aplica solbel a un plano
    plano_s=sobel_operator_2D(plano);
    % metemos el plano en el nuevo volumen
    V_S=meterplano(3,i,V_S,plano_s);

end % del for de i
```

A.6.-sacarplano.m

```
function [lon]=sacarplano(eje,num_eje,Vol)
% eje=1  --->fijamos las x
% eje=2  --->fijamos las y
% eje=3  --->fijamos las z
% num_eje=valor  ---> Valor de la coordenada fijada

%calculamos el tamaño del volumen
[i j k]=size(Vol);

% Preparamos un plano en blanco
switch eje
    case 1,
        lon=zeros(j,k); %plano q devolvemos
    case 2,
        lon=zeros(i,k); %plano q devolvemos
    case 3,
        lon=zeros(j,k); %plano q devolvemos
end % del switch de eje

%Eje=1 fijamos la x
if eje==1
  for y=1:j
    for z=1:k
      lon(y,z)=Vol(num_eje,y,z);
    end %del z
  end %del y
end %del if de eje

%Eje=2 fijamos la y
if eje==2
  for x=1:i
    for z=1:k
      lon(x,z)=Vol(x,num_eje,z);
    end %del z
  end %del x
end %del if de eje

%Eje=3 fijamos la z
if eje==3
  for x=1:i
    for y=1:j
      lon(x,y)=Vol(x,y,num_eje);
    end %del y
  end %del x
end %del if de eje
```

A.7.-sobel_operator_2D.m

```
function plano_s=sobel_operator_2D(plano)
% Función
% Esta función recibe un plano, aplica los operadores de solbel al
% plano y devuelve el plano procesado.
% Fernando Fuentes Arija , Julio 2009

% creamos el nuevo plano
plano_s=plano;

% sacamos el tamaño del plano
[f c]=size(plano);

% empezamos a recorrer todos pixeles
for k=1:f
   for s=1:c
     %Llamamos a la función q nos devuelve la ventana de 3x3
     ventana=sacar_ventana_3x3(plano,k,s);
     % ventana de sobel H1
     H1=[-1 -2 -1; 0 0 0; 1 2 1];
     % ventana de sobel H2
     H2=[-1 0 1; -2 0 2; -1 0 1];
     % calculamos G1
     G1=ventana.*H1;

sum_G1=G1(1,1)+G1(1,2)+G1(1,3)+G1(2,1)+G1(2,2)+G1(2,3)+G1(3,1)+G1(3,2)
+G1(3,3);
     M_G1=abs(sum_G1);
     % calculamos G2
     G2=ventana.*H2;

sum_G2=G2(1,1)+G2(1,2)+G2(1,3)+G2(2,1)+G2(2,2)+G2(2,3)+G2(3,1)+G2(3,2)
+G2(3,3);
     M_G2=abs(sum_G2);

     % el nuevo valor del pixel es
     plano_s(k,s)=M_G1+M_G2;


   end % del for de s
end % del for de k
```

A.8.-sacar_ventana_3x3.m

```
function ventana=sacar_ventana_3x3(plano,k,s)
% Función
% Esta función saca una ventana de 3x3 pixeles, a partir de las
coordenadas
% (k,s) del pixel central. f es el número de filas del plano, y c el
% número de columnas
% Fernando Fuentes Arija , Julio 2009
[f c]=size(plano);

     x=k; % num fila
     y=s; % num columna

     % Vamos a sacar una ventana de 3x3 considerando la ventana
     %           A  B  C
     %           D  E  F
     %           G  H  I

     % PIXEL A (p-1,p-1)
     x_A=x-1;
     y_A=y-1;
     if x_A==0 | y_A==0 | x_A>f | y_A>c
         pixel_A=0;
     else
         pixel_A=plano(x_A,y_A);
     end % del if

     % PIXEL B (p-1,p)
     x_B=x-1;
     y_B=y;
     if x_B==0 | y_B==0 | x_B>f | y_B>c
         pixel_B=0;
     else
         pixel_B=plano(x_B,y_B);
     end % del if

     % PIXEL C (p-1,p+1)
     x_C=x-1;
     y_C=y+1;
     if x_C==0 | y_C==0 | x_C>f | y_C>c
         pixel_C=0;
     else
         pixel_C=plano(x_C,y_C);
     end % del if

     % PIXEL D (p,p-1)
     x_D=x;
     y_D=y-1;
     if x_D==0 | y_D==0 | x_D>f | y_D>c
         pixel_D=0;
     else
         pixel_D=plano(x_D,y_D);
     end % del if

     % PIXEL E (p,p)
     pixel_E=plano(x,y);
```

```
% PIXEL F (p,p+1)
     x_F=x;
     y_F=y+1;
     if x_F==0 | y_F==0 | x_F>f | y_F>c
         pixel_F=0;
     else
         pixel_F=plano(x_F,y_F);
     end % del if

     % PIXEL G (p+1,p-1)
     x_G=x+1;
     y_G=y-1;
     if x_G==0 | y_G==0 | x_G>f | y_G>c
         pixel_G=0;
     else
         pixel_G=plano(x_G,y_G);
     end % del if

     % PIXEL H (p+1,p)
     x_H=x+1;
     y_H=y;
     if x_H==0 | y_H==0 | x_H>f | y_H>c
         pixel_H=0;
     else
         pixel_H=plano(x_H,y_H);
     end % del if

     % PIXEL I (p+1,p+1)
     x_I=x+1;
     y_I=y+1;
     if x_I==0 | y_I==0 | x_I>f | y_I>c
         pixel_I=0;
     else
         pixel_I=plano(x_I,y_I);
     end % del if

     % sacamos la ventana de 3x3
     ventana=[pixel_A pixel_B pixel_C ; pixel_D pixel_E pixel_F ;
pixel_G pixel_H pixel_I];
```

A.9.-meterplano.m

```
function [Vol]=meterplano(eje,num_eje,Vol,lon)
% eje=1  --->fijamos las x
% eje=2  --->fijamos las y
% eje=3  --->fijamos las z
% num_eje=valor  ---> Valor de la coordenada fijada

%calculamos el tamaño del volumen
[i j k]=size(Vol);
%Eje=1 fijamos la x
if eje==1
  for y=1:j
    for z=1:k
      Vol(num_eje,y,z)=lon(y,z);
    end %del z
  end %del y
end %del if de eje

%Eje=2 fijamos la y
if eje==2
  for x=1:i
    for z=1:k
      Vol(x,num_eje,z)=lon(x,z);
    end %del z
  end %del x
end %del if de eje

%Eje=3 fijamos la z
if eje==3
  for x=1:i
    for y=1:j
      Vol(x,y,num_eje)=lon(x,y);
    end %del y
  end %del x
end %del if de eje
```

A.10.-mean_edge_gray_value.m

```
function V_E=mean_edge_gray_value(V,V_S)
% Función
% Esta función recibe un volumen V y la procesación del mismo mediante
% los operadores de sobel en V_S, y devuelve la mean edge gray value

%creamos el volumen que vamos a devolver procesado
V_E=V;

% sacamos el tamaño
[x y z]=size(V);

% vamos a fijar Z y sacar todos los planos (x,y)
for i=1:z
    % sacamos un plano numero i
    plano_V=sacarplano(3,i,V);
    % sacamos un plano numero i
    plano_V_S=sacarplano(3,i,V_S);
    % utilizamos una función que calcula E
    plano_E=MEGV(plano_V,plano_V_S);
    % metemos el plano en el nuevo volumen
    V_E=meterplano(3,i,V_E,plano_E);

end % del for de i
```

A.11.-MEGV.m

```
function plano_E=MEGV(plano_V,plano_V_S)
% Función
% Esta función recibe dos planos, uno normal y el otro con los
% operadores de sobel aplicados, y devuelve un plano con los valores del
% Mean Edge Gray Value

% creamos el plano
plano_E=plano_V;
% sacamos el tamaño del plano
[f c]=size(plano_V);

% empezamos a recorrer todos pixeles
for k=1:f
   for s=1:c
     %Sacamos ventana de la imagen normal
     ventana_V=sacar_ventana_3x3(plano_V,k,s);
     %Sacamos ventana de la imagen procesada
     ventana_V_S=sacar_ventana_3x3(plano_V_S,k,s);
     %calculamos el E
     mul=ventana_V.*ventana_V_S;
     num=sum(mul(:));
     den=sum(ventana_V_S(:));
     E=num/den;
     plano_E(k,s)=E;

   end % del for de s
end % del for de k
```

A.12.-funcion_no_lineal.m

```
function V_F=funcion_no_lineal(V,V_E)
% Función
% Esta función saca una ventana de 3x3 pixeles, a partir de las
% coordenadas(k,s) del pixel central. f es el número de filas del
% plano, y c el número de columnas

%creamos el volumen q vamos a devolver
V_F=V;
% saco el tamaño
[x y z]=size(V);
% variables umbral
a=0; % umbral inferior
b=0; % umbral superior

for i=1:x
    for j=1:y
        for k=1:z
            % sacamos el pixel
            x=V(i,j,k);
            % saco la E del pixel
            E=V_E(i,j,k);
            % sacamos el rango al que pertenece x
            if x>=0 & x<=3
                a=0;
                b=3;
            elseif x>3 & x<=8
                a=3;
                b=8;
            elseif x>8 & x<=16
                a=8;
                b=16;
            elseif x>16 & x<=30
                a=16;
                b=30;
            elseif x>30 & x<=49
                a=30;
                b=49;
            elseif x>49 & x<=75
                a=49;
                b=75;
            elseif x>75 & x<=107
                a=75;
                b=107;
            elseif x>107 & x<=147
                a=107;
                b=147;
            elseif x>147 & x<=196
                a=147;
                b=196;
            elseif x>196 & x<=255
                a=196;
                b=255;
            end % del if
```

```
% Con todos los datos, aplicamos la función
            if x<=E
                x_new=E-sqrt(((E-a)^2)-((x-a)^2));
            elseif x>E
                x_new=E+sqrt(((b-E)^2)-((x-b)^2));
            end

            V_F(i,j,k)=x_new;

        end % del for de k
    end % del for de j
end % del for del i
```

B.1.-main.m

```
function [V,V_S,V_E,V_F]=main()
% Programa principal
% Aumento de contraste no lineal basado en la escala de Munsell
% Este ejercicio se va a simular en 2D
% Fernando Fuentes Arija , Julio 2009

%introducimos el nombre del volumen que queremos leer (no funciona)
nombre=input('Introduzca el nombre del volumen a leer: ','s');
% El modo depende de como el microprocesador ordena los bytes
% puede ser ieee-le o ieee-be.
modo='ieee-be';
V=open_volume(nombre,modo);

% comprobamos el tamaño
[x y z]=size(V);
% Busco el máximo
maximo=busco_max(V);
% Busco el mínimo
minimo=busco_min(V);

% escalamos el volumen para que todos los voxel estén entre 0 y 250
V=escalado(V,maximo,minimo,255,0);

% Lo primero que vamos ha hacer, es detectar el background en la
% imagen de microscopía
[V,V_pixeles_background,media_background]=background_detection(V);

% Ahora plicamos un cierre (morfologia matematica) para eliminar
[V_pixeles_background_mm]=mat_morph_cierre(V_pixeles_background);

% Ahora ponemos todos los píxeles del background ("1" en el volumen
% V_pixeles_background_mm) al valor de "media_background"
[V]=homogeneizar(V,V_pixeles_background_mm,media_background);

% Aplicamos los operadores de Sobel
V_S=sobel_3D(V);

% Calculamos Eij  "mean edge gray value"
V_E=mean_edge_gray_value_3D(V,V_S);

%Aplicamos la función no lineal
V_F=funcion_no_lineal(V,V_E);

%Salvo los resultados
save('resultado.mat','V','V_S','V_E','V_F');
```

B.2.-background_detection.m

```
function
[V,V_pixeles_background,media_background]=background_detection(V)

% Esta función recibe un volumen V con la molécula, e intenta definir
% cuál es el fondo, devuelve:
% V= El volumen con el fondo a cero
% V_pixeles_background= Volumen que tiene los píxeles pertenecientes
% al background a "1", y a "0" el resto.

% Sacamos el tamaño del cubo
[x y z]=size(V);
% Creamos el cubo
V_pixeles_background=zeros(x,y,z);
% Volumen con los pixeles visitados a "1" y el resto a "0"
V_visitado=zeros(x,y,z);

% Tenemos que calcular la desviación , número pixeles y media del
% background inicial. Consideramos como background inicial las seis
caras
% del cubo
cara_1=V(1,:,:);
cara_2=V(x,:,:);
cara_3=V(:,1,:);
cara_4=V(:,y,:);
cara_5=V(:,:,1);
cara_6=V(:,:,z);
% Hemos visitado las seis caras
V_visitado(1,:,:)=1;
V_visitado(x,:,:)=1;
V_visitado(:,1,:)=1;
V_visitado(:,y,:)=1;
V_visitado(:,:,1)=1;
V_visitado(:,:,z)=1;
% Las seis caras las consideramos background
V_pixeles_background(1,:,:)=1;
V_pixeles_background(x,:,:)=1;
V_pixeles_background(:,1,:)=1;
V_pixeles_background(:,y,:)=1;
V_pixeles_background(:,:,1)=1;
V_pixeles_background(:,:,z)=1;

% Puntos que forman mi background inicial
puntos_background=[cara_1(:);cara_2(:);cara_3(:);cara_4(:);cara_5(:);
cara_6(:)];
% Número de puntos del background
[num_background c]=size(puntos_background);
% Media del background
media_background=(sum(puntos_background))/num_background;
% Calculamos la desviacion estandar
var_background=var(puntos_background);
des_background=sqrt(var_background);

% Creamos la lista con los elementos por procesar, esta lista tendrá
% una fila por pixel del siguiente modo "x y z".
lista_por_procesar=[];
```

```
for xx=2:1:x-1 % caras 1 y 2 de z=cte
    for yy=2:1:y-1
        lista_por_procesar=[lista_por_procesar;xx yy 2];
        V_visitado(xx,yy,2)=1;
        lista_por_procesar=[lista_por_procesar;xx yy z-1];
        V_visitado(xx,yy,z-1)=1;
    end % del for de yy
end % del for de xx
for xx=2:1:x-1 % caras 1 y 2 de y=cte
    for zz=2:1:z-1
        lista_por_procesar=[lista_por_procesar;xx 2 zz];
        V_visitado(xx,2,zz)=1;
        lista_por_procesar=[lista_por_procesar;xx y-1 zz];
        V_visitado(xx,y-1,zz)=1;
    end % del for de yy
end % del for de xx
for yy=2:1:y-1 % caras 1 y 2 de x=cte
    for zz=2:1:z-1
        lista_por_procesar=[lista_por_procesar;2 yy zz];
        V_visitado(2,yy,zz)=1;
        lista_por_procesar=[lista_por_procesar;x-1 yy zz];
        V_visitado(x-1,yy,zz)=1;
    end % del for de yy
end % del for de xx

% Variables para actualizar los valores del background cada "n"
% muestras
n=250;
cont_muestras=250;
% Variable para determinar cuando dejamos de buscar pixeles que
% pertenezcan al background
busqueda=1;  % Cuando se vacíe la lista por procesar lo ponemos a
% cero

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% CORE DE LA FUNCIÓN  %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while busqueda==1 % La busqueda acaba cuando se vacíe la lista por
% procesar
    % Aquí  actualizamos y calculamos los intervalos de confianza si
    % es necesario
    if n==cont_muestras
      % Actualizamos las variables estadísticas del background
      num_background,media_background,des_background]=actualizar_para
      metros(puntos_background);
      % Calculamos los intervalos de confianza
      [A B]=intervalo_confianza(des_background,num_background,
      media_background);
      % Ponemos el contador a cero
      cont_muestras=0;
    end % del if de n
```

```
% Sacamos el primer pixel de la lista
    [lista_por_procesar,elemento]=sacar_elemento(lista_por_procesar);
    % Sacamos las coordenadas del pixel
    x_e=elemento(1,1);
    y_e=elemento(1,2);
    z_e=elemento(1,3);
    % Sacamos el valor
    valor=V(x_e,y_e,z_e);

        % comprobamos si podemos considerar el pixel del vecindario
        if (A<=valor) && (valor<=B)  % es background
            % Lo marcamos como background
            V_pixeles_background(x_e,y_e,z_e)=1;
            % Lo introducimos en los puntos background
            puntos_background=[puntos_background;valor];
            % Actualizamos el contador xq el bg ha sido modificado
            cont_muestras=cont_muestras+1;
            % Consideramos todos sus vecinos como posibles candidatos
            % a background
            [lista_por_procesar,V_visitado]=meter_vecinos(x_e,y_e,
             z_e,lista_por_procesar,V_visitado);
        end % if del A y B

    % Si no hay más elementos por procesar salimos del while
    [f c]=size(lista_por_procesar);
    tam_lista=f % IMPORTANTE luego quitar
    if f==0 % la lista está vacía
         busqueda=0; % Hemos acabado de buscar el background
    end % del if
end % del while de busqueda
```

B.3.-sacar_elemento.m

```
function [cola,elemento]=sacar_elemento(cola)
% Esta función recibe una "cola", (que es un lista en forma de
% columna), y saca el primer elemento de la cola, quitándolo de la
% misma

% Sacamos el tamaño
[f c]=size(cola);

% frontera
if f==1 % dejamos vacía la cola
    % Sacamos el primer elemento
    elemento=[cola(1,1) cola(1,2) cola(1,3)];
    % Devolvemos una cola vacía
    cola=[];
else % la cola tiene más de un elemento
    % Sacamos el primer elemento
    elemento=[cola(1,1) cola(1,2) cola(1,3)];
    % Quitamos ese primer elemento de la cola
    cola=cola(2:f,:);
end % del if de f
```

B.4.-intervalo_confianza.m

```
function [A B]=intervalo_confianza(Sx,Nx,x_med)

% Calculamos la inversa de la t de student
t=tinv(0.995,Nx-1);
% Calculamos el umbral inferior
A=x_med-(t*Sx);
B=x_med+(t*Sx);
```

B.5.-meter_vecinos.m

```
function [lista V_visitado]=meter_vecinos(x,y,z,lista,V_visitado)
% Esta función mete los pixeles vecinos de pixel=(x,y,z) en la
"lista"

% Calculamos las coordenadas de los vecinos
for i=x-1:x+1
    for j=y-1:y+1
        for k=z-1:z+1
            % comprobamos si ese pixel ha sido ya visitado
            vis=V_visitado(i,j,k);
            if vis==0 % no ha sido visitado
                % Lo metemos en la lista
                lista=[lista; i j k];
                % Lo consideramos visitado
                V_visitado(i,j,k)=1;
            end % del if

        end % del for de k
    end % del for de j
end % del for de i
```

B.6.-actualizar_parámetros.m

```
function[num_background,media_background,des_background]=actualizar
_parametros(puntos_background)
% Esta función recibe los píxeles del background, y devuelve el
número de
% pixeles, la media y la desviación típica de los mismos

% Número de puntos del background
[num_background c]=size(puntos_background);
% Media del background
media_background=(sum(puntos_background))/num_background;
% Calculamos la desviacion estandar
var_background=var(puntos_background);
des_background=sqrt(var_background);
```

B.7.-mat_morph_cierre.m

```
function [V]=mat_morph_cierre(V)
% Esta función aplica la operación de "apertura" de morfología
matemática
% para eliminar el ruido aislado

% Saco el tamaño del volumen
[x y z]=size(V);

% Vamos a sacar todos los planos y a procesarlos
for i=1:x
% Sacamos un plano
[plano]=sacarplano(1,i,V);
% Procesamos el plano
plano_p = bwmorph(plano,'close');
% Metemos el plano procesado
[V]=meterplano(1,i,V,plano_p);
end % del for de i

% Las caras sabemos q son background
V(1,:,:)=1;
V(x,:,:)=1;
V(:,1,:)=1;
V(:,y,:)=1;
V(:,:,1)=1;
V(:,:,z)=1;
```

B.8.-homogeneizar.m

```
function [V]=homogeneizar(V,V_lista,valor)
% Esta función recibe un volumen V, y un volumen V_lista, este último
% tiene valores binarios, en la posición donde "V_lista" tenga los
% 1os pone el valor de "valor" en esa misma posición en "V".

% Sacamos el tamaño de los volúmenes
[x y z]=size(V);
% Recorremos el volumen binario
for i=1:x
    for j=1:y
        for k=1:z
            % Sacamos el la identidad del pixel
            bg=V_lista(i,j,k);
            if bg==1 % ese pixel forma parte del background
                % Ponemos ese pixel a la media del background
                V(i,j,k)=valor;
            end % del if
        end % del for de k
    end % del for de j
end % del for de i
```

B.9.-sobel_3D.m

```
function V_S=sobel_3D(V)
% Función
% Esta función recibe un volumen V
% Fernando Fuentes Arija , Julio 2009

%creamos el volumen que vamos a devolver procesado
V_S=zeros(size(V));

% sacamos el tamaño
[x y z]=size(V);

% vamos a sacar todas las posiciones del cubo
for i=2:x-1
    for j=2:y-1
        for k=2:z-1
            % sacamos un cubo
            cubo=sacar_cubo_lado_n(V,i,j,k,3);
            % utilizamos una función que me aplica solbel
            pixel_s=sobel_operator_3D(cubo);
            % metemos el pixel procesado
            V_S(i,j,k)=pixel_s;
        end % del for de k
    end % del for de j
end % del for de i
```

B.10.-sacar_cubo_lado_n.m

```
function [cubo,origen]=sacar_cubo_lado_n(V,p_x,p_y,p_z,n)
% Esta función saca un cubo de lado n (nxnxn) a partir de un cubo
% más grande V y el pixel central del mismo (p_x,p_y,p_z)
% Fernando Fuentes Arija , Julio 2009

% Calculamos lo que nos tenemos q desplazar desde el pixel central
W=(n-1)/2;

% Saco el tamaño del cubo para saber los límites
[lim_x lim_y lim_z]=size(V);

% Defino desde donde a donde voy a leer en el cubo grande
x_i=max(1,p_x-W);
x_f=min(lim_x,p_x+W);
y_i=max(1,p_y-W);
y_f=min(lim_y,p_y+W);
z_i=max(1,p_z-W);
z_f=min(lim_z,p_z+W);

cubo=V(x_i:x_f,y_i:y_f,z_i:z_f);
origen=[x_i-p_x  y_i-p_y  z_i-p_z];
```

B.11.-sobel_operator_3D.m

```
function pixel_s=sobel_operator_3D(cubo)
% Esta función recibe un plano, aplica los operadores de solbel al
% plano y devuelve el plano procesado.
% Fernando Fuentes Arija , Julio 2009

% Matriz para el gradiente en x
x_1=[-1 -3 -1; -3 -6 -3; -1 -3 -1];
x_2=[0 0 0; 0 0 0; 0 0 0];
x_3=[1 3 1; 3 6 3; 1 3 1];

% Matriz para el gradiente en y
y_1=[1 3 1; 0 0 0; -1 -3 -1];
y_2=[3 6 3; 0 0 0; -3 -6 -3];
y_3=[1 3 1; 0 0 0; -1 -3 -1];

% Matriz para el gradiente en z
z_1=[-1 0 1; -3 0 3; -1 0 1];
z_2=[-3 0 3; -6 0 6; -3 0 3];
z_3=[-1 0 1; -3 0 3; -1 0 1];

% Necesito 3 cubos
cubo_x=zeros(3,3,3);
cubo_y=zeros(3,3,3);
cubo_z=zeros(3,3,3);

% Procesamos el gradiente para las x
cubo_x(:,:,1)=cubo(:,:,1).*x_1;
cubo_x(:,:,2)=cubo(:,:,2).*x_2;
cubo_x(:,:,3)=cubo(:,:,3).*x_3;

sum_x=sum(cubo_x(:));
M_x=abs(sum_x);

% Procesamos el gradiente para las y
cubo_y(:,:,1)=cubo(:,:,1).*y_1;
cubo_y(:,:,2)=cubo(:,:,2).*y_2;
cubo_y(:,:,3)=cubo(:,:,3).*y_3;

sum_y=sum(cubo_y(:));
M_y=abs(sum_y);

% Procesamos el gradiente para las z
cubo_z(:,:,1)=cubo(:,:,1).*z_1;
cubo_z(:,:,2)=cubo(:,:,2).*z_2;
cubo_z(:,:,3)=cubo(:,:,3).*z_3;

sum_z=sum(cubo_z(:));
M_z=abs(sum_z);

% Calculamos el nuevo valor del pixel
pixel_s=M_x+M_y+M_z;
```

B.12.-mean_edge_gray_value_3D.m

```
function V_E=mean_edge_gray_value_3D(V,V_S)
% Esta función recibe un volumen V y la procesación del mismo      %
mediante los operadores de sobel en V_S, y devuelve la mean edge %
gray value
% Fernando Fuentes Arija , Julio 2009

% % Ahora calculamos el tamaño de los vecindarios variables
[V_tam]=tam_vecindario(V);

% Sacamos el tamaño
[x y z]=size(V);
% Variable donde guardamos el volumen procesado
V_E=zeros(x,y,z);

% Procesamos todo el cubo con su vecindario correspondiente

for i=1:x
    for j=1:y
        for k=1:z
            % Sacamos el tamaño de ventana del pixel
            tam_op=V_tam(i,j,k);
            % sacamos el cubo del volumen original de tamaño optimo
            [cubo_V,origen_V]=sacar_cubo_lado_n(V,i,j,k,tam_op);
            % sacamos un cubo del volumen procesado del tamaño del  %
            cubo_V

            [cubo_V_S,origen_V_S]=sacar_cubo_lado_n(V_S,i,j,k,tam_op);
            % utilizamos una función que calcula E
            pixel_E=MEGV_3D(cubo_V,cubo_V_S);
            % metemos el pixel en el nuevo volumen
            V_E(i,j,k)=pixel_E;

        end % del for de k
    end % del for de j
end % del for de i
```

B.13.-tam_vecindario.m

```
function [V_tam]=tam_vecindario(V)
% Esta función recibe un volumen "V" con la imagen de la molécula, y
% calcula el vecindario variable para cada pixel (mediante el uso de
% intervalos de confianza)

% Definimos el tamaño máximo del vecindario
n_tam=9;
% Sacamos el tamaño del volumen
[x y z]=size(V);
% Variable para guardar los tamaños
V_tam=zeros(x,y,z);

% contador para ver el % del proceso
cont=0;   % IMPORTANTE2
num_p=x*y*z; % IMPORTANTE2

% Tenemos que procesar todos los pixeles
for i=1:x
    for j=1:y
        for k=1:z  % Estamos trabajando con el pixel (i,j,k)
            % Variable para seguir buscando
            encontrado=0;
            % La primera ventana es un cubo de 3x3x3
            tam_cubo=3;
            [cubo_p,origen_p]=sacar_cubo_lado_n(V,i,j,k,tam_cubo);

            while encontrado==0
            % Saco un cubo más grande
            tam_cubo=tam_cubo+2;
            % Sacamos el cubo grande
            [cubo_g,origen_g]=sacar_cubo_lado_n(V,i,j,k,tam_cubo);
            % Comparo los cubos
            [iguales]=comparador_cubos(cubo_p,cubo_g);
                if (iguales==1)&&(tam_cubo<n_tam) % sigo buscando
                    % Ahora el cubo grande es el pequeño
                    cubo_p=cubo_g;
                elseif iguales==0  % hemos encontrado el tamaño optimo
                    encontrado=1;
                    % Metemos el tamaño optimo
                    V_tam(i,j,k)=tam_cubo-2;
                else % hemos llegado a un tamaño de cubo maximo
                    encontrado=1;
                    % Metemos el tamaño optimo
                    V_tam(i,j,k)=9;
                end % del if

            end % del while

            % hemos acabado de procesar un pixel
            cont=cont+1;   % IMPORTANTE2
            tam_betas_porcentaje=(100*cont)/(num_p)  % IMPORTANTE2

        end % del for de k
    end % del for de j
end % del for de i
```

B.14.-comporador_cubos.m

```
function [iguales]=comparador_cubos(cubo_p,cubo_g)
% Esta función recibe un cubo, y calcula sus estadísticos
% "m" media
% "S" desviación típica
% "N" número de pixeles

% Sacamos los tamaños
[xp yp zp]=size(cubo_p);
[xg yg zg]=size(cubo_g);
% Sacamos el número de pixeles
Np=xp*yp*zp;
Ng=xg*yg*zg;
% Sacamos la media
mp=(sum(cubo_p(:))/Np);
mg=(sum(cubo_g(:))/Ng);
% Calculamos la desviacion estandar
S2p=var(cubo_p(:));
S2g=var(cubo_g(:));

% Calculamos el número de grados de libertad
num=((S2p/Np)+(S2g/Ng))^2;
den=(S2p/((Np-1)*Np^2))+(S2g/((Ng-1)*Ng^2));
gl=num/den;

% Calculamos el umbral t de la distribución "t de student"
if gl>1
    t=tinv(0.975,gl);
else
    t=0;
end % del if
% Calculamos el rango
k=sqrt((S2p/Np)+(S2g/Ng));
d=mp-mg;
A=d-(k*t);
B=d+(k*t);

% Variable
iguales=0; % no son iguales
% Si el rango de la diferencia de medias contiene al cero, podemos
% considerar que son iguales
if (A<0) && (0<B)
    iguales=1;
end % del if
% Si la varianza es cero, son iguales xq estamos en el background
if t==0
    iguales=1;
end % del if
```

B.15.-MEGV_3D.m

```
function pixel_E=MEGV_3D(cubo_V,cubo_V_S)
% Función
% Esta función recibe dos cubos, uno normal y el otro con los operadores
% de sobel aplicados, y devuelve el pixel con el valor del Mean Edge
% Gray Value
% Fernando Fuentes Arija , Julio 2009

% Calculamos el Mean Edge Gray Value
 mul=cubo_V.*cubo_V_S;
 num=sum(mul(:));
 den=sum(cubo_V_S(:));
 E=num/den;
 pixel_E=E;
```

C.1.-detect_background (filters.h)

```
/** Detect background
 * @ingroup Filters
 * This function recieves a Matrix3D vol, and try to find the background
 * assuming that all the outside planes contain background, and apply
 * interval confidence, were alpha is the probabity to fail.
 * Mask is of the same size of vol, and is the solutiÃ³n, mask have
 * value 1 if background else value 0
*/
void detect_background(const Matrix3D<double> &vol, Matrix3D<double>
&mask, double alpha, double &final_mean);
```

C.2.-detect_background (filters.cpp)

```
/* Detect background ------------------------------------------------------ */
void detect_background(const Matrix3D<double> &vol, Matrix3D<double> &mask,
      double alpha,double &final_mean)
{

      // 2.1.-Background detection-----------------------------------------
            Matrix3D<double> bg; // We create the volumen with
            bg.resize(vol);           // -1:not visited 0:mol 1:background
            bg.initConstant(-1);  // -2:in the list

            // Ponemos las seis caras de esta variable como visitadas inicializamos
            // la cola de pÃxeles por visitar
            std::queue<int> list_for_compute; // Lista del modo
                                              // [x1,y1,z1,...,xi,yi,zi] que
                                              // contiene los pixeles por procesar
            std::vector<double> bg_values; // Vector con los valores del background
            int xdim=XSIZE(bg);
            int ydim=YSIZE(bg);
            int zdim=ZSIZE(bg);
            FOR_ALL_DIRECT_ELEMENTS_IN_MATRIX3D(bg)
            {
                  if(j==0 || j==xdim-1 || i==0 || i==ydim-1 || k==0 || k==zdim-1)
                  {   // Visited coord
                        DIRECT_VOL_ELEM(bg,k,i,j)=1;
                        // We introduce the values of the background
                        bg_values.push_back(DIRECT_VOL_ELEM(vol,k,i,j));
                  }
                  if( (j==1 || j==xdim-2) && (i!=0) && (i!=ydim-1) && (k!=0) &&
                    (k!=zdim-1) )
                  {   // Coord for compute for x
                        list_for_compute.push(j);
                        list_for_compute.push(i);
                        list_for_compute.push(k);
                  }
                  if( (i==1 || i==ydim-2) && (j>1) && (j<xdim-2) && (k!=0) &&
                    (k!=zdim-1) )
                  {   // Coord for compute for y
                        list_for_compute.push(j);
                        list_for_compute.push(i);
                        list_for_compute.push(k);
                  }
                  if( (k==1 || k==zdim-2) && (j>1) && (j<xdim-2) && (i>1) &&
                    (i<ydim-2) )
                  {   // Coord for compute for y
                        list_for_compute.push(j);
                        list_for_compute.push(i);
                        list_for_compute.push(k);
                  }
            } // end of FOR_ALL_ELEMENTS

            // We work until the list_for_compute is empty
            int n=250; //each 250 pixels renew stats
            int cont=250; //We start here for compute stat for first time
            double A; // A and B are numbers such the interval of confidence is
                      //[A,B]
            double B; //
```

```
        float z=icdf_gauss(1-alpha/2);
         while(!list_for_compute.empty())
         {

               //We compute stat when is needed
               if(cont==n)
               {
                     // Calculamos los estadÃsticos
                     double avg, stddev, minval, maxval;
                     computeStats(bg_values,avg,stddev,minval,maxval);
                     final_mean=avg;
                     // Calculamos el intervalo de confianza
                     A=avg-(z*stddev);
                     B=avg+(z*stddev);
                     cont=0; // aumentamos el contador
                     //std::cout << "Pulsa una tecla\n";
                     //          char c; std::cin >> c;
               } // end of if
               // Now we start to take coords from the list_for_compute
               int x_coord=list_for_compute.front(); list_for_compute.pop();
               int y_coord=list_for_compute.front(); list_for_compute.pop();
               int z_coord=list_for_compute.front(); list_for_compute.pop();
               // Is visited
               DIRECT_VOL_ELEM(bg,z_coord,y_coord,x_coord)=-2;
               //We take the value
               double value=DIRECT_VOL_ELEM(vol,z_coord,y_coord,x_coord);
               // We see if is background or not
               if (A<=value && value<=B)
               {
                   // We now is background
                   DIRECT_VOL_ELEM(bg,z_coord,y_coord,x_coord)=1;
                   // We introduce the values of the background
                   bg_values.push_back(value);
                   // We update the cont variable
                   cont++;


                   // We add his neighbours in the list_for_compute
                   for(int xx=x_coord-1;xx<=x_coord+1;xx++)
                     for(int yy=y_coord-1;yy<=y_coord+1;yy++)
                           for(int zz=z_coord-1;zz<=z_coord+1;zz++)
                                 //We see if it has been visited
                                 if (DIRECT_VOL_ELEM(bg,zz,yy,xx)==-1)
                                  // not visited
                                 {
                                       // So we include it in the list
                                       list_for_compute.push(xx);
                                       list_for_compute.push(yy);
                                       list_for_compute.push(zz);
                                       // Is in the list
                                       DIRECT_VOL_ELEM(bg,zz,yy,xx)=-2;
                                 }
               } // end of if
               else
               {
                  // Isn't background
                     DIRECT_VOL_ELEM(bg,z_coord,y_coord,x_coord)=0;
               }
         } // end of while
```

```
            // Now we change not visited for mol
            FOR_ALL_DIRECT_ELEMENTS_IN_MATRIX3D(bg)
                  if (DIRECT_VOL_ELEM(bg,k,i,j)==-1)
                        DIRECT_VOL_ELEM(bg,k,i,j)=0;

            // End of 2.1-------------------------------------------------------------

            // 2.2.-Matematical Morphology
            Matrix3D<double> bg_mm; // We create the output volumen
            bg_mm.resize(vol);
            closing3D(bg,bg_mm,26,0,1);
            // Output
          //typeCast(bg_mm,mask);
            mask=bg_mm;
}
```

C.3.-enhance_contrast.h

```
/***************************************************************************
*
* Authors:    Carlos Oscar            coss@cnb.csic.es (2010)
*             Fernando Fuentes
*
* Unidad de  Bioinformatica of Centro Nacional de Biotecnologia , CSIC
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
* 02111-1307  USA
*
*  All comments concerning this program package may be sent to the
*  e-mail address 'xmipp@cnb.csic.es'
***************************************************************************/

#ifndef ENHANCE_CONTRAST_H
#define ENHANCE_CONTRAST_H

#include <string>
#include <data/progs.h>
#include <data/morphology.h>
#include <data/filters.h>

#include <queue>
#include <vector>
#include <iostream>


/// @defgroup EnhanceContrast Enhance constrast
/// @ingroup ReconsLibraryPrograms

/// Parameters for enhance contrast program
/// @ingroup Denoise
class EnhanceContrast_parameters: public Prog_parameters
{
public:
      // Confidence level for background identification
      double alpha;

      // Lower intensity (%)
      double lowerIntensity;

      // Remove the background
      bool removeBg;

      // Save mask
      FileName fnMask;
```

```
public:
    /** Empty constructor
     */
    EnhanceContrast_parameters();

    /** Read parameters from command line
     */
    void read(int argc, char** argv);

    /** Produce side info.
     *
     * The DWT type is translated and set
     */
    void produce_side_info();

    /** Show parameters. This function calls show_specific.
     */
    void show();

    /** Usage. This function calls usage_specific.
     */
    void usage();

    /** Enhance constrast of an image.
     */
    void enhance(Matrix2D< double >& img);

    /** Enhance contrast of a volume.
     */
    void enhance(Matrix3D< double >& vol);

};

#endif
```

C.4.-enhance_contrast.cpp

```
/***************************************************************************
 *
 * Authors:    Carlos Oscar            coss@cnb.csic.es (2010)
 *             Fernando Fuentes
 *
 * Unidad de  Bioinformatica of Centro Nacional de Biotecnologia , CSIC
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
 * 02111-1307  USA
 *
 *  All comments concerning this program package may be sent to the
 *  e-mail address 'xmipp@cnb.csic.es'
 ***************************************************************************/

#include "enhance_contrast.h"
#include <data/args.h>

#define DEBUG

// Empty constructor -------------------------------------------------------
EnhanceContrast_parameters::EnhanceContrast_parameters():
    Prog_parameters()
{
      alpha=0.01;
}

// Read from command line --------------------------------------------------
void EnhanceContrast_parameters::read(int argc, char **argv)
{
    Prog_parameters::read(argc, argv);
    alpha=textToFloat(getParameter(argc,argv,"-alpha","0.01"));
    lowerIntensity=textToFloat(getParameter(argc,argv,"-lowerIntensity","0.05"));
    removeBg=checkParameter(argc,argv,"-removeBackground");
    fnMask=getParameter(argc,argv,"-saveMask","");
    produce_side_info();
}

// Produce side info -------------------------------------------------------
void EnhanceContrast_parameters::produce_side_info()
{
}

// Show --------------------------------------------------------------------
void EnhanceContrast_parameters::show()
{
```

```
    Prog_parameters::show();
    std::cout << "Alpha: " << alpha << std::endl
              << "lowerIntensity: " << lowerIntensity << std::endl
                    << "removeBackground: " << removeBg << std::endl
                    << "saveMask: " << fnMask << std::endl;
}

// Usage ---------------------------------------------------------------------s
void EnhanceContrast_parameters::usage()
{
    Prog_parameters::usage();
    std::cerr << "  [-alpha <a=0.01>]         : Confidence interval for background
identification\n"
              << "  [-lowerIntensity <%>]     : Only process if is hier\n"
                    << "  [-removeBackground]       : Remove the noise of the
background\n"
              << "  [-saveMask <filename=\"\"]  : Filename for the background
mask\n"
    ;
}

// Enhance image -------------------------------------------------------------
void EnhanceContrast_parameters::enhance(Matrix2D<double> &img)
{
}

// Enhance volume ------------------------------------------------------------
void EnhanceContrast_parameters::enhance(Matrix3D<double> &vol)
{

      // 1.-Scale volume between 0 and 255----------------------------------
      double minVal, maxVal;
      vol.computeDoubleMinMax(minVal,maxVal);
      vol.rangeAdjust(0,255);
      VolumeXmipp save;
    #ifdef DEBUG
      save()=vol; save.write("PPP1_init.vol");
    #endif

      // 2.-Elimination of the background-----------------------------------
      Matrix3D<double> mask;
      double bg_mean;
      detect_background(vol,mask,0.01,bg_mean);
      #ifdef DEBUG
      save()=mask; save.write("PPPmask.vol");
      #endif

      // We change 0<->1
      FOR_ALL_ELEMENTS_IN_MATRIX3D(mask)
      {
            if (VOL_ELEM(mask,k,i,j)==1) {VOL_ELEM(mask,k,i,j)=0; }
            else {VOL_ELEM(mask,k,i,j)=1; }
      } // Now if 0:background and if 1:mol
      #ifdef DEBUG
      save()=mask; save.write("PPPmask_c.vol");
      #endif
```

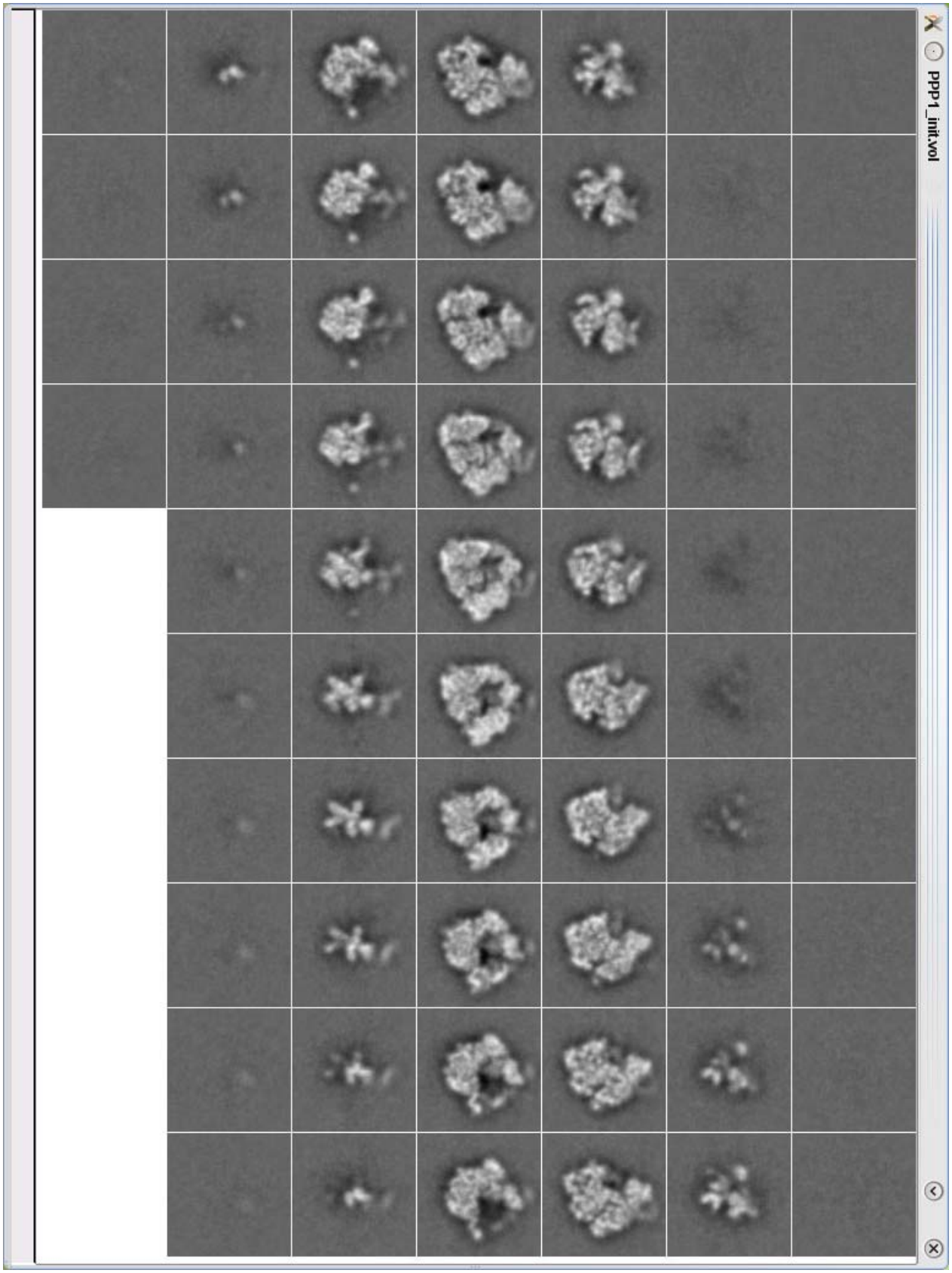
```
    if (fnMask!="")
    {
        save()=mask; save.write(fnMask);
    }
    if (removeBg==true)
    {
        // We put all the background with the same value
        FOR_ALL_DIRECT_ELEMENTS_IN_MATRIX3D(vol)
            if (DIRECT_VOL_ELEM(mask,k,i,j)==0)
                DIRECT_VOL_ELEM(vol,k,i,j)=bg_mean;
    }
    #ifdef DEBUG
    save()=vol; save.write("PPP2_no_bg.vol");
    #endif

    // 3.-BSplines + diff = edges-----------------------------------------
    Matrix3D<double> BSpline_coefs; // We create the volumen tha is going to
    BSpline_coefs.resize(vol);          // contain the BSpline coefficients
    // We compute the BSpline Coefficients
    vol.produceSplineCoefficients(BSpline_coefs,3);
    // We create the Edge volume
    Matrix3D<double> vol_edge; // We create the volumen tha is going to
    vol_edge.resize(vol);         // contain the BSpline coefficients

    FOR_ALL_ELEMENTS_IN_MATRIX3D(vol)
    {
        double V_dx;
        double V_dy;
        double V_dz;
        V_dx=BSpline_coefs.interpolatedElementBSplineDiffX(j,i,k,3);
        V_dy=BSpline_coefs.interpolatedElementBSplineDiffY(j,i,k,3);
        V_dz=BSpline_coefs.interpolatedElementBSplineDiffZ(j,i,k,3);
        VOL_ELEM(vol_edge,k,i,j)=sqrt((V_dx*V_dx)+(V_dy*V_dy)+(V_dz*V_dz));

    }
    #ifdef DEBUG
    save()=vol_edge; save.write("PPP3_edge.vol");
    #endif
    // 4.-MEAN EDGE GRAY VALUE

    // 4.1.-Variable Neighbourhood
#define COMPUTE_STATISTICS(V,N,k,i,j,avg,stddev,cubeSize) \
    {\
            int k0,kF,i0,iF,j0,jF; \
          k0=XMIPP_MAX(k-N,STARTINGZ(V)); \
          kF=XMIPP_MIN(k+N,FINISHINGZ(V)); \
          i0=XMIPP_MAX(i-N,STARTINGY(V)); \
          iF=XMIPP_MIN(i+N,FINISHINGY(V)); \
          j0=XMIPP_MAX(j-N,STARTINGX(V)); \
        jF=XMIPP_MIN(j+N,FINISHINGX(V)); \
         double sum=0, sum2=0; \
         for (int kk=k0; kk<=kF; kk++) \
                for (int ii=i0; ii<=iF; ii++) \
                    for (int jj=j0; jj<=jF; jj++) \
                    { \
                        double v=VOL_ELEM(V,kk,ii,jj); \
                        sum+=v; \
                        sum2+=v*v; \
                    } \
            cubeSize=(kF-k0+1)*(iF-i0+1)*(jF-j0+1); \
            avg=sum/cubeSize; \
```

```
                    stddev=sum2/cubeSize-avg*avg; \
                    stddev=sqrt(XMIPP_MAX(stddev,0)); \
              }

        // We use gaussian because we have a T-Student with more than 100
        float z=icdf_gauss(1-(0.01/2));  // degrees of freedom
        // We create a volume with the neighboor sizes
        Matrix3D<double> vol_tam;
        vol_tam.resize(vol);
        vol_tam.initConstant(4);

        FOR_ALL_ELEMENTS_IN_MATRIX3D(vol)
        {
              if (VOL_ELEM(mask,k,i,j)==1)
              {
                    // only compute if the pixel is not background
                    double cubeDim=1; // 3x3x3
                    double newCubeDim=1; // 3x3x3
                    double max_tam=4; // 9x9x9
                    double avgSmall, stddevSmall, NSmall;

        COMPUTE_STATISTICS(vol,cubeDim,k,i,j,avgSmall,stddevSmall,NSmall);
                    bool found=false;
                          while (!found)
                          {
                                // Big Cube
                                newCubeDim=newCubeDim+1;
                                double avgBig, stddevBig, NBig;

        COMPUTE_STATISTICS(vol,newCubeDim,k,i,j,avgBig,stddevBig,NBig);
                                // Computing the degrees of freedom
                                      //double num, den;

        //num=(((stddevSmall*stddevSmall)/NSmall)+((stddevBig*stddevBig)/NBig))*
                                      //
        (((stddevSmall*stddevSmall)/NSmall)+((stddevBig*stddevBig)/NBig));
                                      //den=((stddevSmall*stddevSmall)/((NSmall-
1)*(NSmall*NSmall)))+
                                      //    ((stddevBig*stddevBig)/((NBig-
1)*(NBig*NBig)));
                                      //double df;
                                      //df=num/den; // Degrees of freedom
                                // Confidence Intervals --> Comparison of two cubes
                                double K, d, A, B;

        K=sqrt(((stddevSmall*stddevSmall)/NSmall)+((stddevBig*stddevBig)/NBig));
                                d=avgSmall-avgBig;
                                A=d-(K*z);
                                B=d+(K*z);
                                // If the interval [A,B] contains the cero there are
the same
                                if ((A<0) && (0<B))
                                {  // equal continue or not
                                      if (newCubeDim>=max_tam)
                                      {     // not continue
                                            found=true;
                                            VOL_ELEM(vol_tam,k,i,j)=newCubeDim;
                                      }
                                      else
```

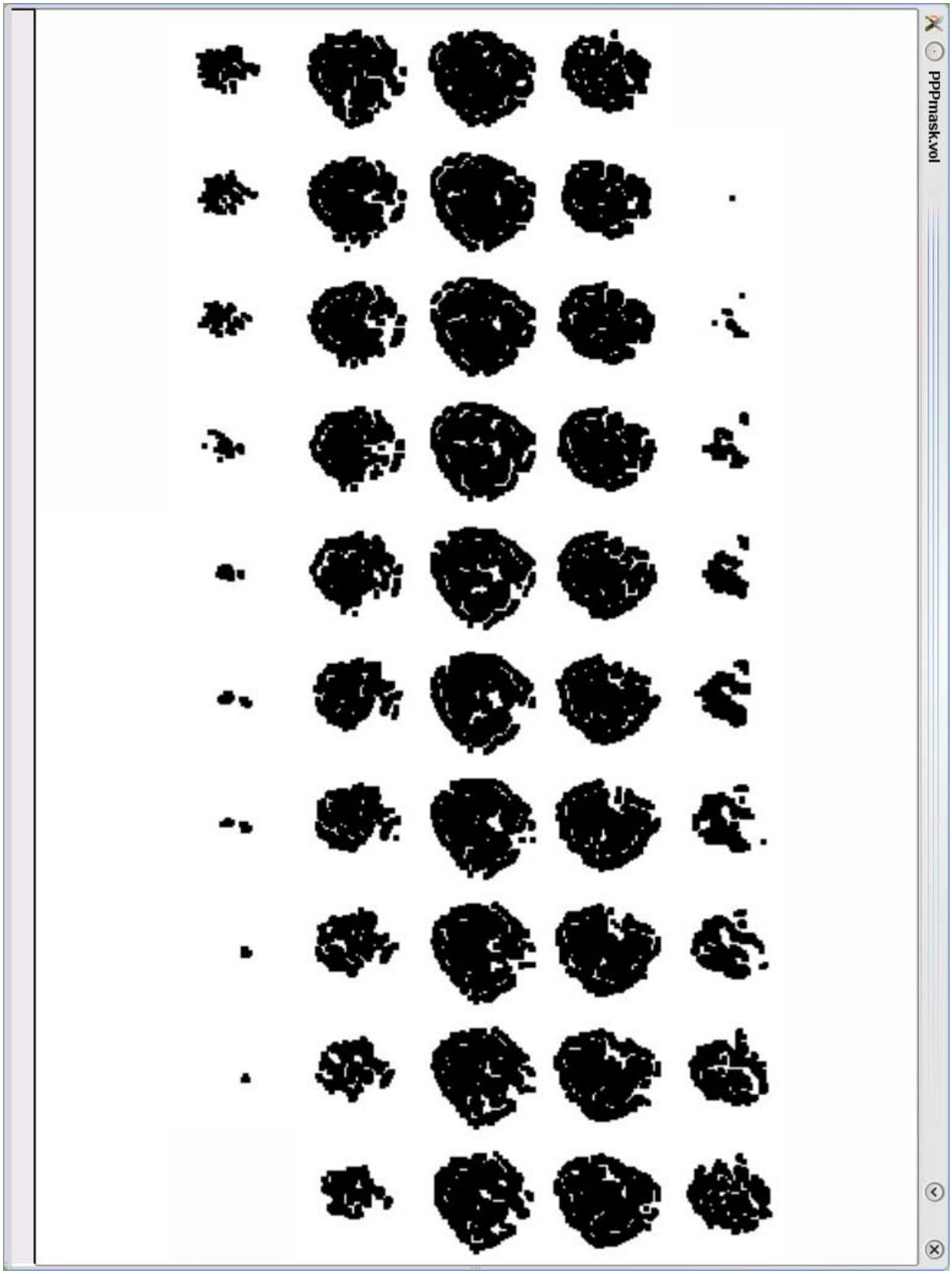
```
                                    {     // continue searching
                                          avgSmall=avgBig;
                                          stddevSmall=stddevBig;
                                          NSmall=NBig;
                                    }
                              }
                              else // Not equal -> we stop searching
                              {
                                    found=true;  // Size found
                                    VOL_ELEM(vol_tam,k,i,j)=newCubeDim-1; // it's
the latest
                              }

                        } // end of while
            } // end of if
      } // end of FOR_ALL_ELEMENTS
      #ifdef DEBUG
      save()=vol_tam; save.write("PPP4_vol_tam.vol");
      #endif

      // 4.2.- Compute the mean edge gray value
      Matrix3D<double> VolxEdge;
      VolxEdge=vol;
      VolxEdge*=vol_edge;
      // Macro for compute de Mean Edge Gray Value
      #define COMPUTE_MEGV(VxE,V_E,N,k,i,j,pixel) \
      {\
                  int k0,kF,i0,iF,j0,jF; \
               k0=XMIPP_MAX(k-N,STARTINGZ(V_E)); \
               kF=XMIPP_MIN(k+N,FINISHINGZ(V_E)); \
               i0=XMIPP_MAX(i-N,STARTINGY(V_E)); \
               iF=XMIPP_MIN(i+N,FINISHINGY(V_E)); \
               j0=XMIPP_MAX(j-N,STARTINGX(V_E)); \
             jF=XMIPP_MIN(j+N,FINISHINGX(V_E)); \
              double sum1=0, sum2=0; \
              for (int kk=k0; kk<=kF; kk++) \
                        for (int ii=i0; ii<=iF; ii++) \
                              for (int jj=j0; jj<=jF; jj++) \
                              { \
                                    double v1=VOL_ELEM(VxE,kk,ii,jj); \
                                    sum1=sum1+v1; \
                                    double v2=VOL_ELEM(V_E,kk,ii,jj); \
                                    sum2=sum2+v2; \
                              } \
                  pixel=sum1/sum2; \
            }
            Matrix3D<double> Vol_E;
            Vol_E=vol;
            double tam, pixel;
            // We compute the MEGV for all the volume
            FOR_ALL_ELEMENTS_IN_MATRIX3D(vol)
            {     // Only compute if no background
                  if (VOL_ELEM(mask,k,i,j)==1)
                  {
                        tam=VOL_ELEM(vol_tam,k,i,j);
                        COMPUTE_MEGV(VolxEdge,vol_edge,tam,k,i,j,pixel);
                        VOL_ELEM(Vol_E,k,i,j)=pixel;
                  }
            }
```

```
            #ifdef DEBUG
            save()=Vol_E; save.write("PPP5_vol_megv.vol");
            #endif

            //5.-Nonlinear function

            Matrix3D<double> vol_f=vol;
            double a, b, x, E;
            double threshold=lowerIntensity*255;

            FOR_ALL_ELEMENTS_IN_MATRIX3D(vol)
            {
                  x=VOL_ELEM(vol,k,i,j);
                  if (VOL_ELEM(mask,k,i,j)==1 && x>threshold)
                  {
                  E=VOL_ELEM(Vol_E,k,i,j);
                  // We search the interval
                  if ((x>=0)  && (x<=3))   { a=0;   b=3;   }
                  else if ((x>3)   && (x<=8))   { a=3;   b=8;   }
                  else if ((x>8)   && (x<=16))  { a=8;   b=16;  }
                  else if ((x>16)  && (x<=30))  { a=16;  b=30;  }
                  else if ((x>30)  && (x<=49))  { a=30;  b=49;  }
                  else if ((x>49)  && (x<=75))  { a=49;  b=75;  }
                  else if ((x>75)  && (x<=107)) { a=75;  b=107; }
                  else if ((x>107) && (x<=147)) { a=107; b=147; }
                  else if ((x>147) && (x<=196)) { a=147; b=196; }
                  else { a=196; b=255; }
                  // nonlinear function
                  double x_new;
                  if (x<=E) { x_new=E-sqrt(((E-a)*(E-a))-((x-a)*(x-a))); }
                  if (x>E)  { x_new=E+sqrt(((b-E)*(b-E))-((x-b)*(x-b)));  }
                  VOL_ELEM(vol_f,k,i,j)=x_new;
                  }
            }
            #ifdef DEBUG
            save()=vol_f; save.write("PPP6_vol_f.vol");
            #endif

            //6.-Re-Scale (now we put the original values diferent from [0-255])
            double x_s,x_ns;
            FOR_ALL_ELEMENTS_IN_MATRIX3D(vol_f)
            {
                  x_s=VOL_ELEM(vol_f,k,i,j);
                  x_ns=((maxVal-minVal)/255)*(x_s+((255*minVal)/(maxVal-minVal)));
                  VOL_ELEM(vol_f,k,i,j)=x_ns;
            }
            // OUTPUT
            vol=vol_f;


} // End of enhance
```

C.5.- PPP1_init.vol

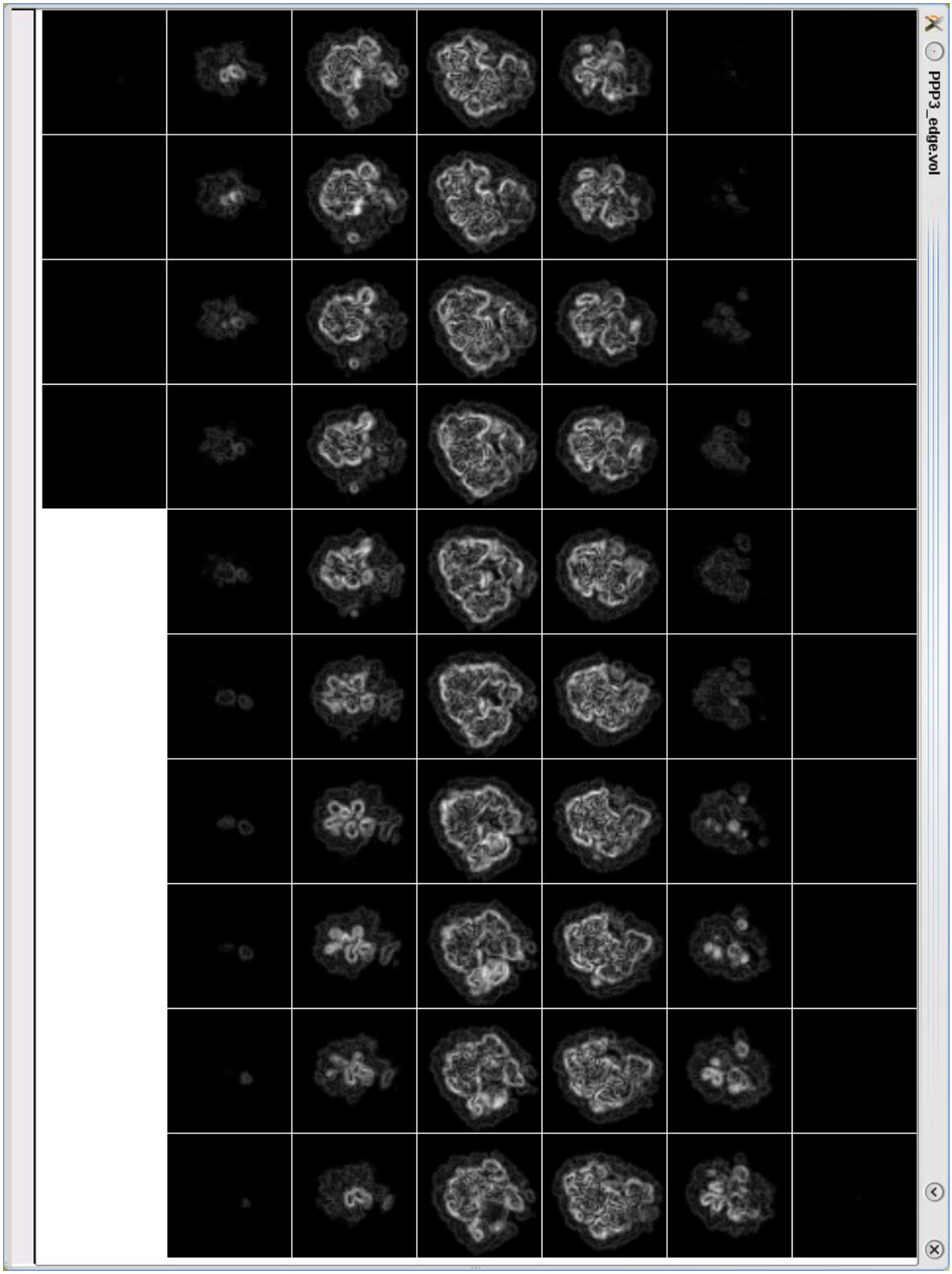


C.6.-PPPmask.vol

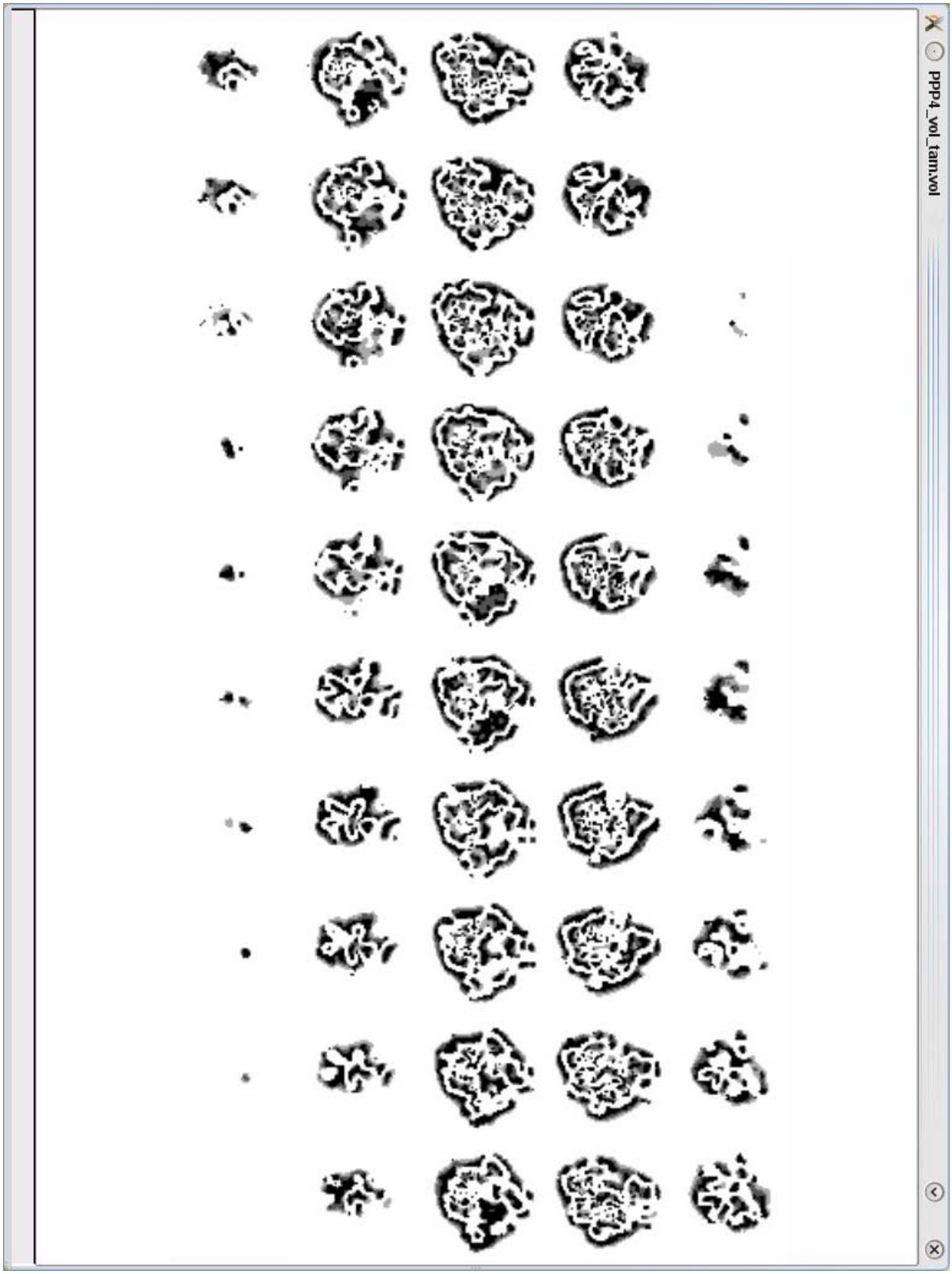


C.7.-PPP2_no_bg.vol

C.8.-PPP3_edge.vol



C.9.-PPP4_vol_tam.vol



C.10.-PPP5_vol_megv.vol

C.11.-PPP6_vol_f.vol

6.-BIBLIOGRAFÍA

Bibliografía mencionada

[1] Unser, M., "Sampling -50 years after Shannon" Proc. IEEE, 88, pp.569-587, 2000.

[2] Unser, M., Aldroubi, A. and Eden, M., "B-Spline signal processing: Part I – Theory," IEEE Trans. Signal Processing, 41, pp. 821-832, 1993.

[3] Unser, M., Aldroubi. A. and Eden, M., "B-Spline signal processing: Part II – Efficient design and applications," IEEE Trans. Signal Processing, 41, pp. 834.848, 1993.

[4] Blu, T., Thevenaz, P. and Unser, M., "MOMS: Maximal-Order interpolation of Minimal Support," IEEE Trans. Image Processing, 10, pp. 1069-1080, 2001

[5] Piegl, L. And Tiller, W., The NURBS book, Springer, 1997

[6] Sean C. Matz "A Nonlinear Image Contrast Sharpening Approach Based on Munsell's Scale" IEEE Trans. on image processing, Vol. 15, No.4, pp.900-909, 2006

Bibliografía utilizada

[6] Sean C. Matz "A Nonlinear Image Contrast Sharpening Approach Based on Munsell's Scale" IEEE Trans. on image processing, Vol. 15, No.4, pp.900-909, 2006

[7] Charles Kervrann And Jerome Boulanger, "Optimal Spatial Adaptation for Patch-Based Image Denoising", Trans. on image processing, Vol. 15, No.10, pp.2866-2878, 2006

[8] Zujun Hou and T.S. Koh, "Image Denoising Using Robust Regresion" IEEE Trans. Signal Processing, Vol 11, NO. 2, pp. 243.246, 2004

[9] Antoni Buades and Bartolomeu Coll, "A non-local algorithm for image denoising" IEEE 2005

[10] Michael Unser, "B-Spline Signal Processing" IEEE Transaction on signal processing, Vol. 41, NO. 2, 1993

[11] Mona Mahmoudi "Fast Image and Video Denoising via Nonlocal Means of Similar Neighborhoods", IEEE Signal Processing, Vol.12, NO. 12, 2005

[12] Marin Van Heel "Similarity measures between images", Ultramicroscopy, NO. 21, 1987

[13] José Fernandez and Carlos Oscar Sanchez, "Image Processing and 3D Reconstruction in Electron Microscopy" IEEE Signal Processing Magazine, 2006

[14] Francisco Gabriel Ortiz, "Procesamiento morfológico de imágenes en color.", Ed. Biblioteca Virtual Miguel de Cervantes, 2002