Segmenting with ITK (I)
Some notes

UNIVERSIDAD DE GRANADA

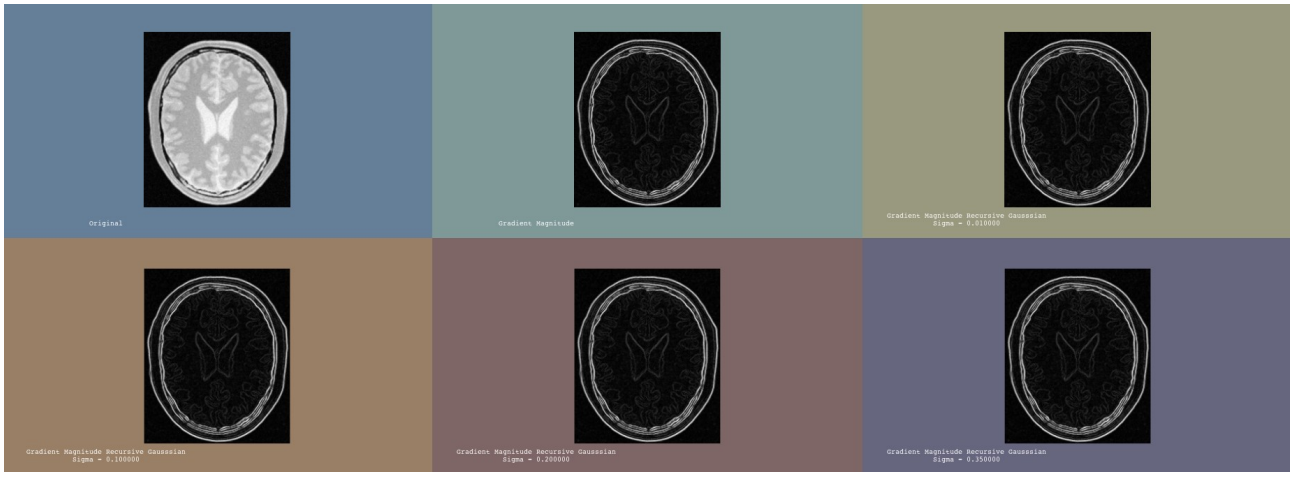
Autor: Víctor Torres de la Torre

Task 1

Write a program that reads an image from a file and displays the result of applying the previous
filters for calculating gradient magnitude to the input image. Try this program with the input images: BrainProtonDensitySlice256x256.png ,saltAndPepperNoise1.jpg , saltAndPepperNoise2.jpg , and gaussianNoise.jpg . You may read the source codes GradientMagnitudeImageFilter.cxx and GradientMagnitudeRecursiveGaussianImageFilter.cxx placed at ' ITK_DIR/Examples/Filtering' to get a clue about how to program this task.

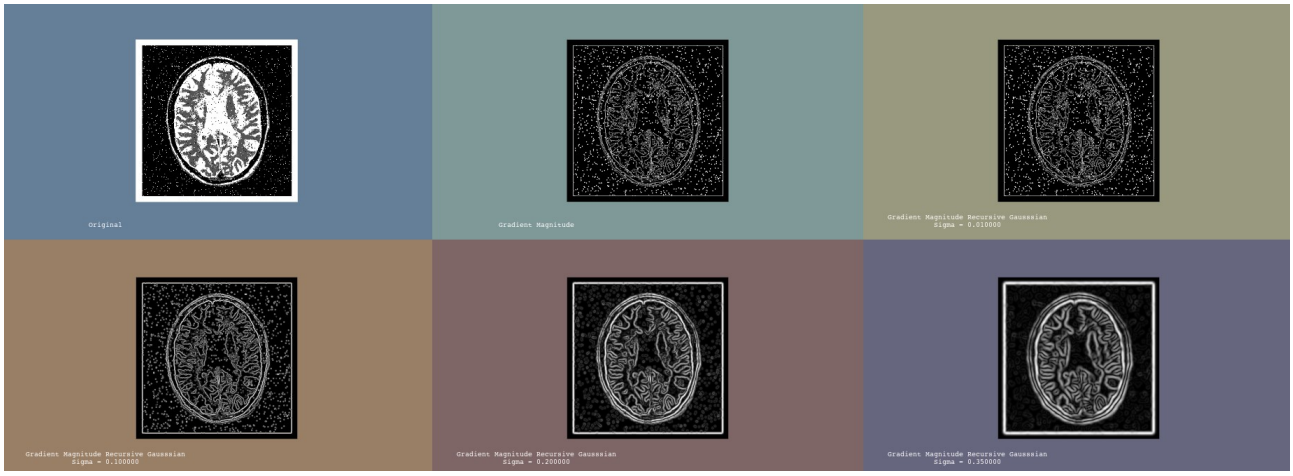
Para poder tener una idea clara de como funcionan ambos filtros, se ha decidido probar varios valores de sigma en GradientMagnitudeRecursiveGaussianFilterpara las diferentes imágenes.

Imagen 1. Sin ruido:



Cómo era de esperar, no hay ninguna diferencia significativa en ninguna de las imágenes obtenidas de ambos filtros, ni entre los diferentes valores de sigma. Como ya sabemos, la gran diferencia entre ambos filtros es si aplicamos o no técnicas de eliminación de ruido gaussiano antes de calcular los gradientes, es por este motivo, que no encontramos diferencias.

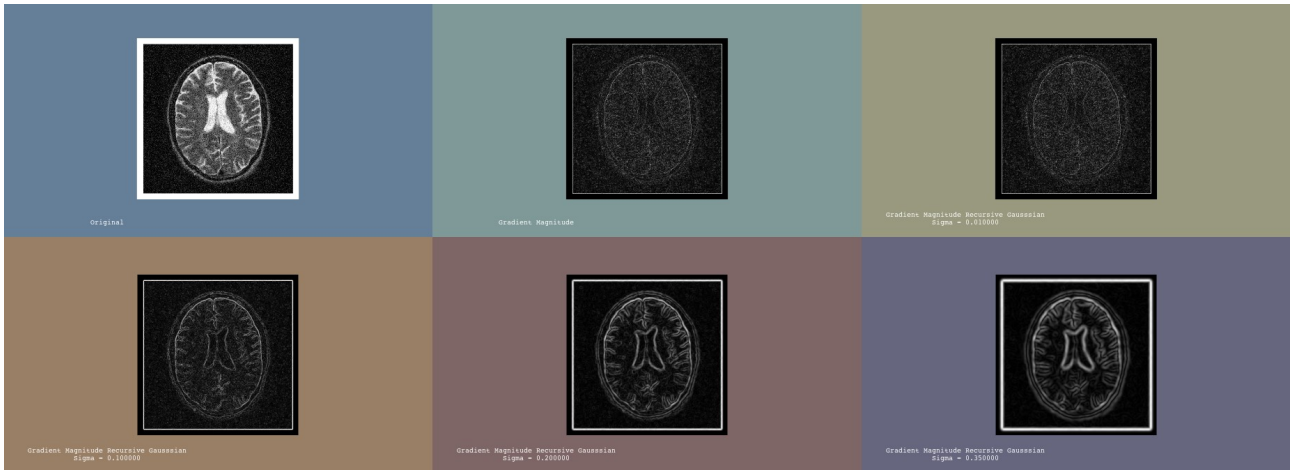
Imagen 2. Con ruido de tipo salt and peeper:



En este caso si que podemos ver grandes diferencias, obteniendo un resultado suficientemente aceptable con un valor de sigma 0.35. Se han probado varios valores

superiores y se ha llegado a la determinación de que 0.35 es el valor de sigma que da un resultado optimo que mantiene equilibrados los aspectos de eliminación de ruido y nitidez de la imagen.

Imagen 3. Con ruido de tipo gaussiano:



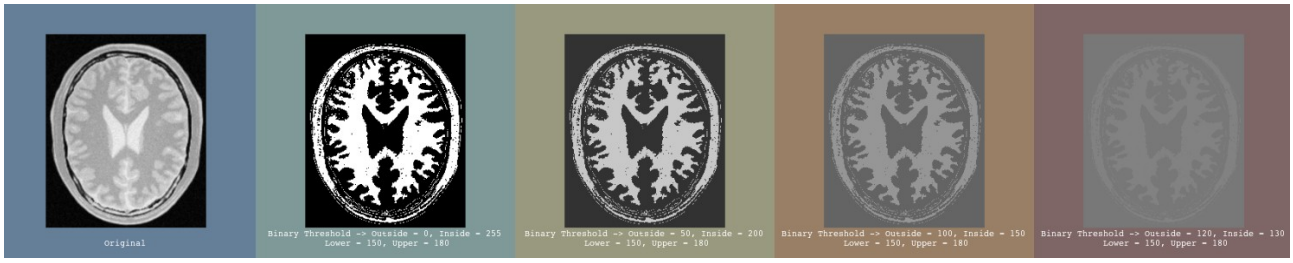
En este caso, aplicando los mismos valores de sigma que en el caso anterior, se puede ver que obtenemos una imagen con un ruido mínimo y una nitidez aceptable para un valor de sigma de 0.2. Observando la imagen con valor de sigma 0.35 vemos que hemos perdido una gran cantidad de nitidez para una mejora mínima en el ruido.

Task 2

Write a program that reads an image from a file and displays the result of applying the previous thresholding filters to the input image. The idea is to obtain a segmentation of some region of interest that you can identify in the image. Try this program with the input images: BrainProtonDensitySlice256x256.png , saltAndPepperNoise1.jpg , saltAndPepperNoise2.jpg , and gaussianNoise.jpg . You may read the source codes BinaryThresholdImageFilter.cxx and ThresholdImageFilter.cxx placed at ‘ ITK_DIR/Examples/Filtering ’ to get a clue about how to program this task.

1. En este caso vamos a comenzar viendo el algoritmo binaryThresholdFilter y como afectan los valores de sus parámetros a la visualización de la imagen tras aplicar el filtro. Para este primer experimento inicial usaremos la imagen sin ruido.

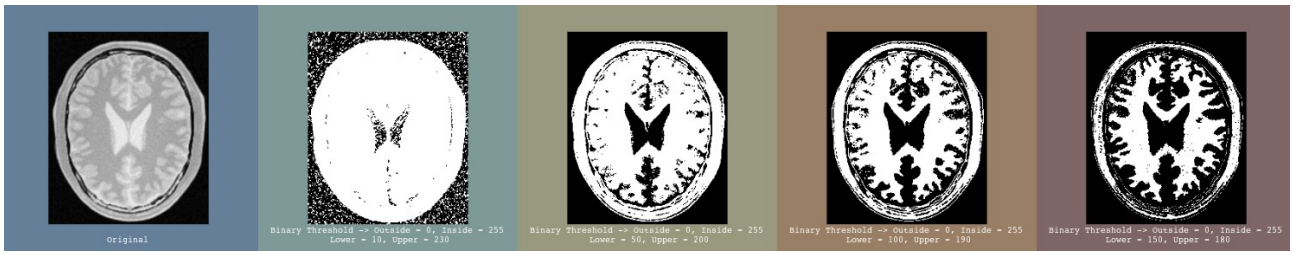
1.1 Valores de outside e inside:



Con estos parámetros podemos controlar que valor de intensidad damos a los pixeles que quedan fuera y dentro del umbral. Se puede observar que cuanto más cercanos sean los valores de estos dos parámetros, menor será el contraste lo que nos dificultará la visualización. Esto nos lleva a la

conclusión de que la mejor opción será separar lo máximo posible los valores de estos dos parámetros, quedando en 0 y 255 como en la primera imagen filtrada de la imagen superior.

1.2 Valores de lower y upper:



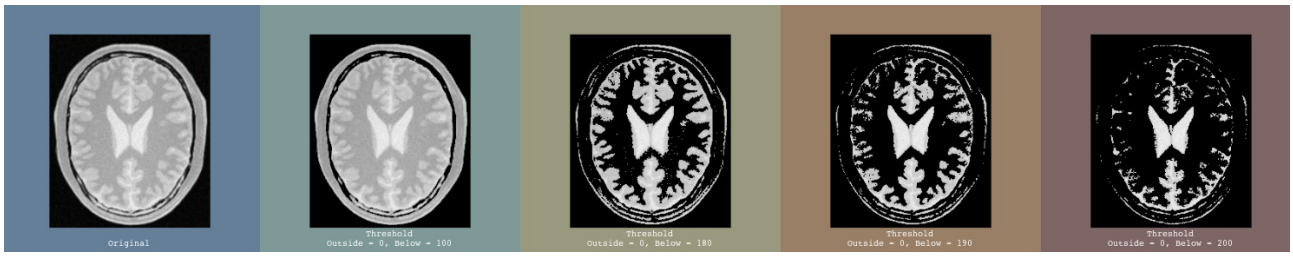
En este caso podemos observar como estos valores determinan el umbral, tras varios experimentos y teniendo en cuenta las visualizaciones obtenidas en este experimento, la visualización más adecuada se obtiene con los valores 150 y 180.

2. Veamos ahora el algoritmo ThresholdFilterType y como afectan los valores de sus parámetros a la visualización de la imagen tras aplicar el filtro. Para este segundo experimento también usaremos la imagen sin ruido.

2.1 Outside:

Empezamos jugando con el parámetro outside, vemos como dejando a 0 el valor de outside tenemos una mejor visualización de la imagen ya que este parámetro controla el valor intensidad de pixel que queda fuera del umbral.

2.2 Bellow:



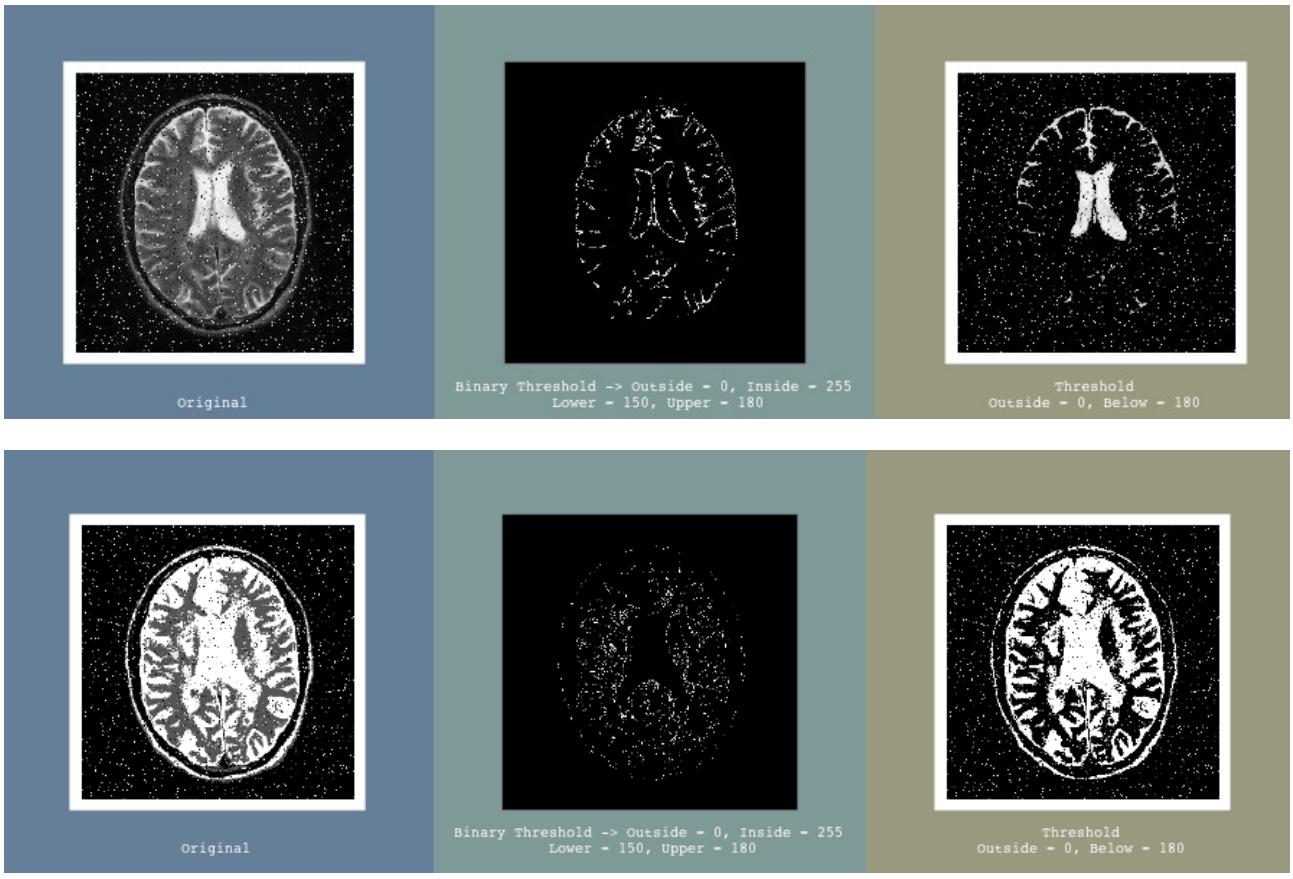
Jugando ahora con el umbral vemos que, tras varias pruebas en el experimento, el mejor valor de bellow por la calidad de detalle es 180.

3. Conclusiones:

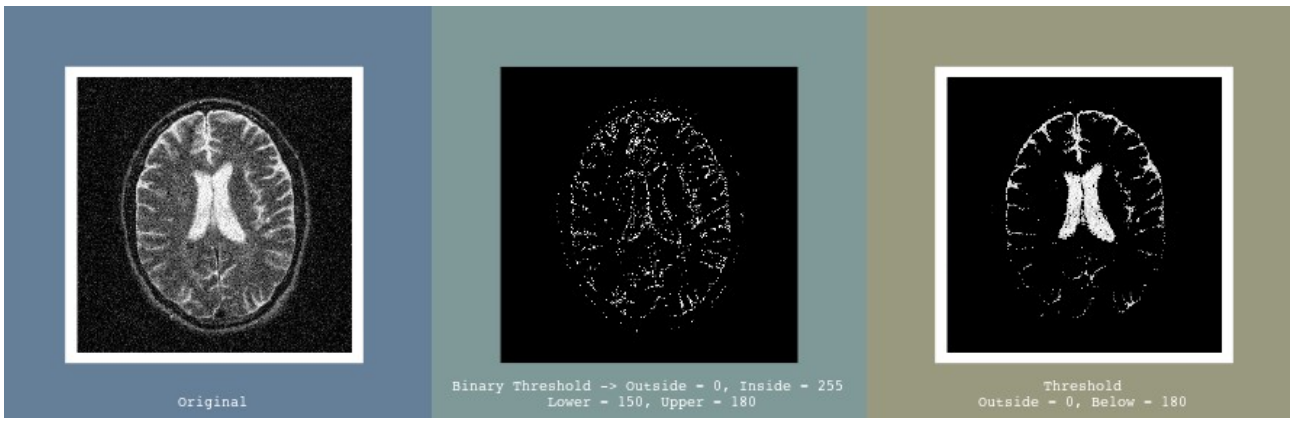
Para finalizar este task veamos las dos opciones más optimas de cada uno de los filtros con diferentes imágenes y sus respectivos tipos de ruido:

3.1 Sin ruido:

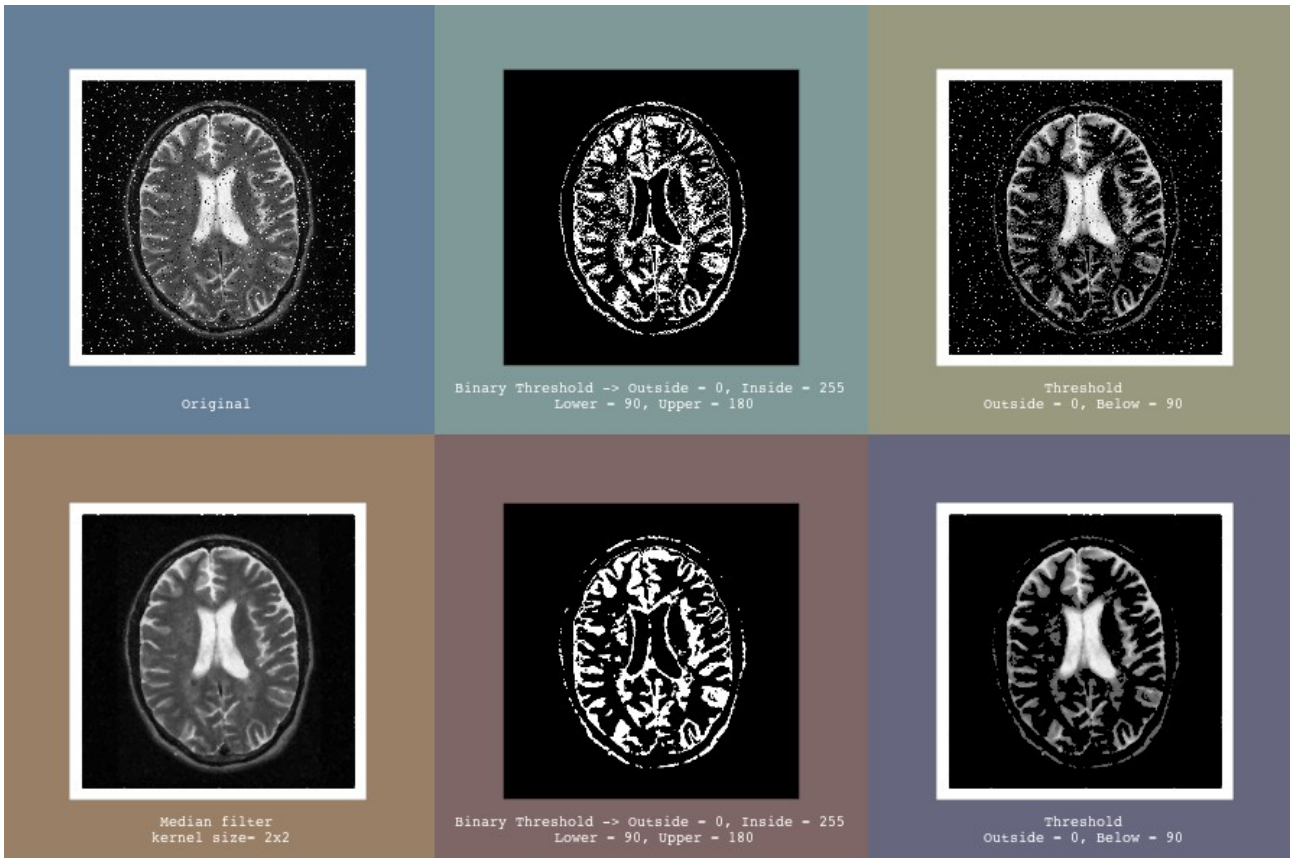
3.2 Con ruido salt and peeper:



3.3 Con ruido gaussiano:



Como vemos, este tipo de filtros no corrigen el ruido en las imágenes y esto nos lleva a una mala visualización que es poco útil si tenemos dicho ruido. Veamos que ocurre si aplicamos un filtro previo para eliminar el ruido.



Como vemos, aplicando un filtro para eliminar el ruido y jugando con los parámetros podemos obtener resultados mucho mejores.

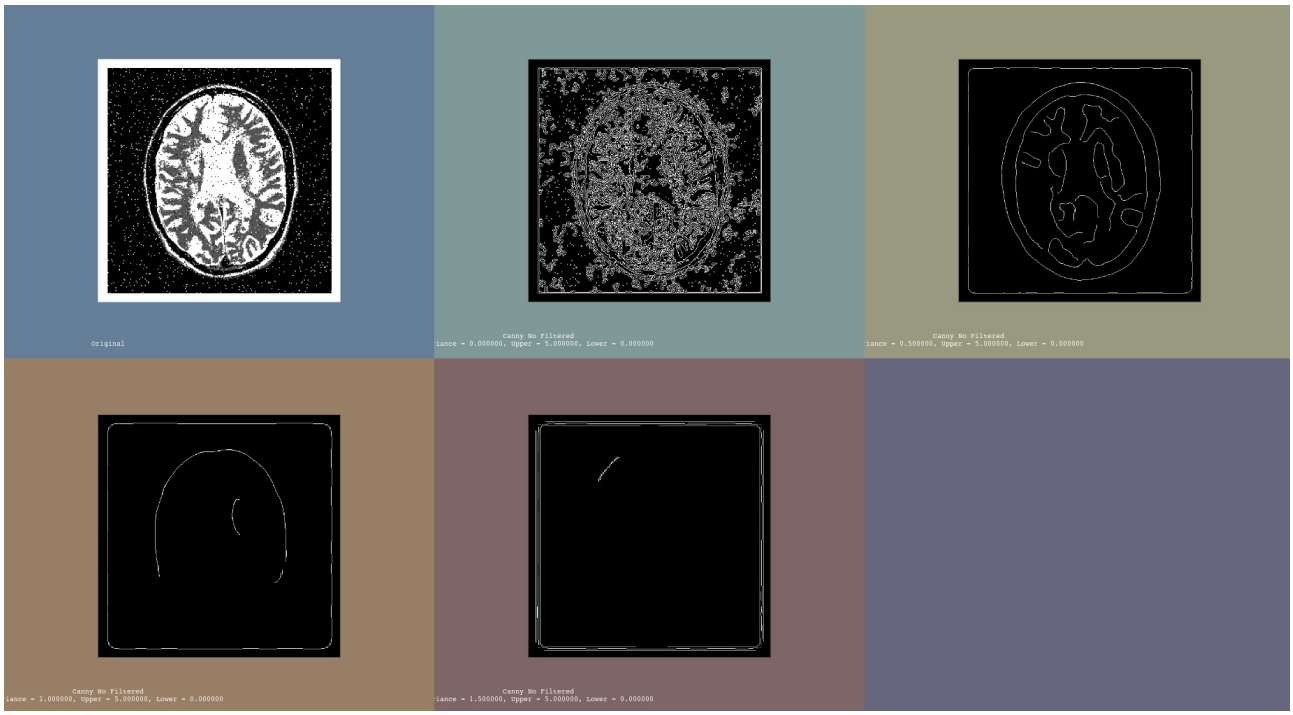
Task 3

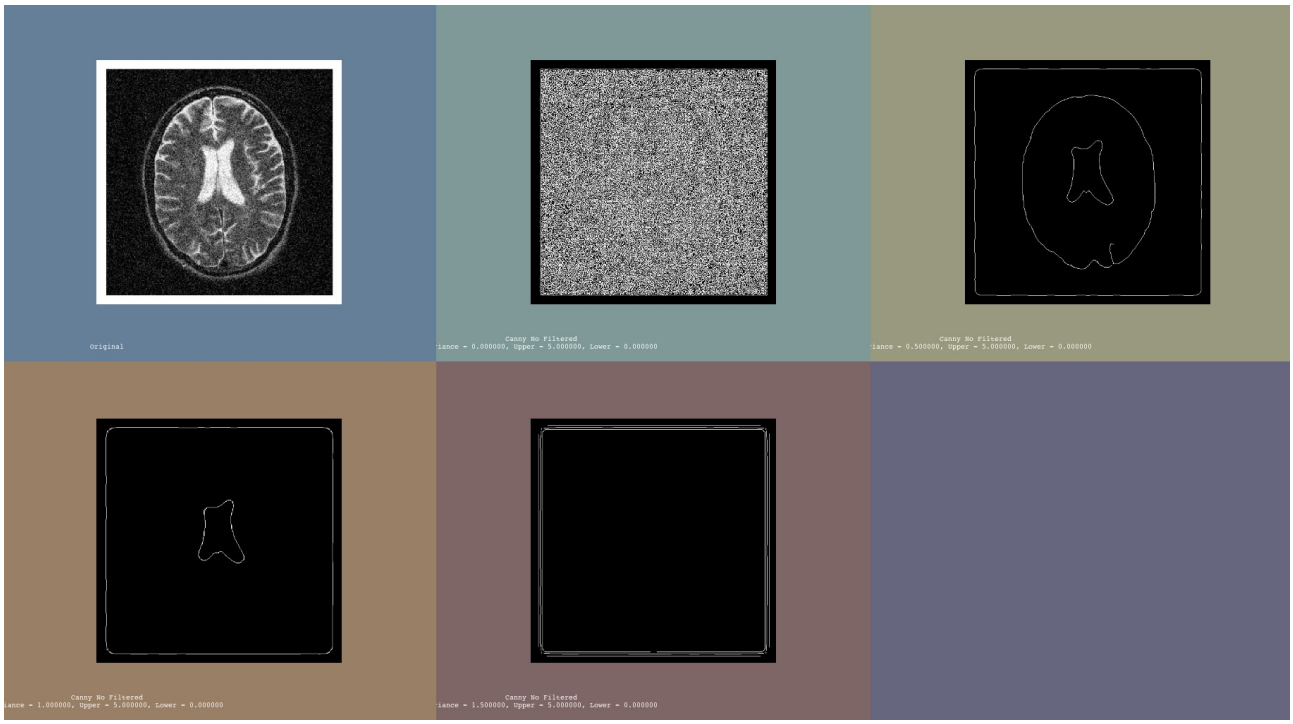
Write a program that reads an image from a file and displays the result of applying the CannyEdgeDetectionImageFilter to the input image. The idea is to obtain several results depending of the free parameters of the filter, together with using a previous noise reduction over an input image or providing the filter with the input image without any previous noise reduction step. Try this program with the input images:

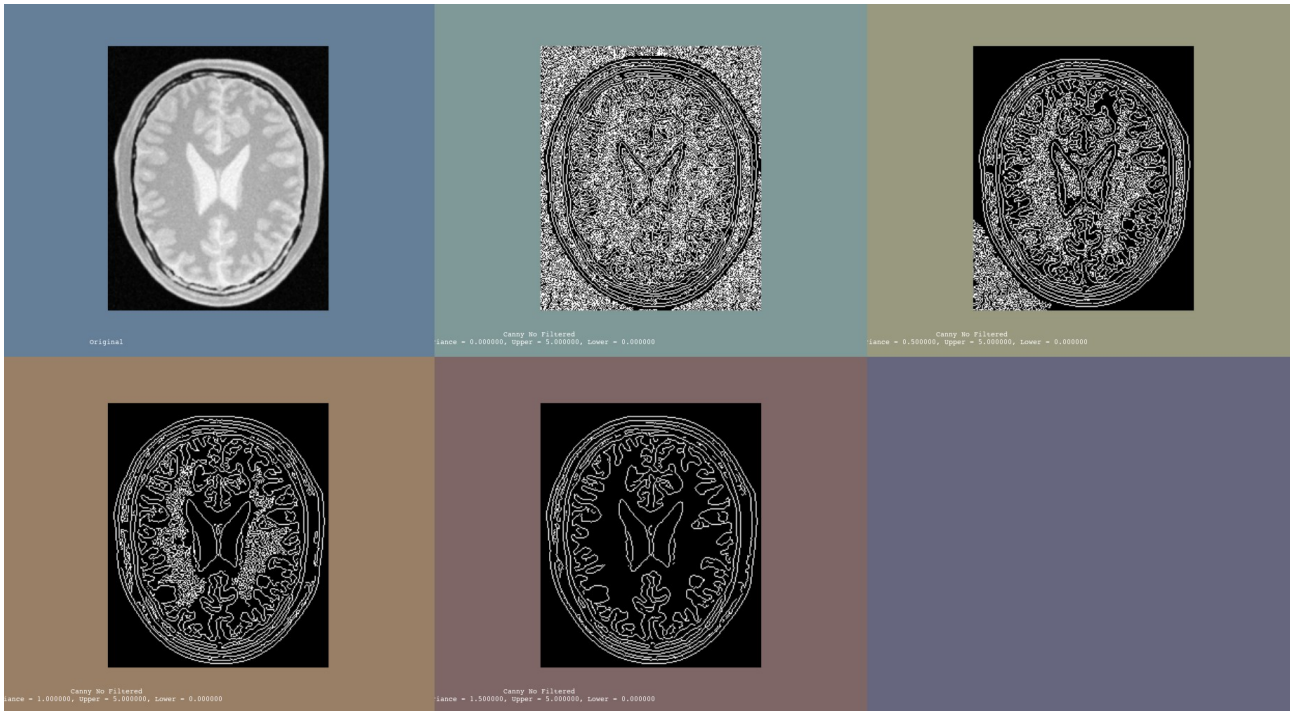
BrainProtonDensitySlice256x256.png , saltAndPepperNoise1.jpg , saltAndPepperNoise2.jpg , and gaussianNoise.jpg . You may read the source code CannyEdgeDetectionImageFilter.cxx placed at ' ITK_DIR/Examples/Filtering ' to get a clue about how to program this task.

3.1 cannyEdgeDetectionFilterNoFiltered

Para empezar a ver como afectan los parámetros del filtro a la visualización de la imagen, vamos a ver como los diferentes valores de la variable varianza afecta a imágenes con diferentes ruidos y sin ruido:

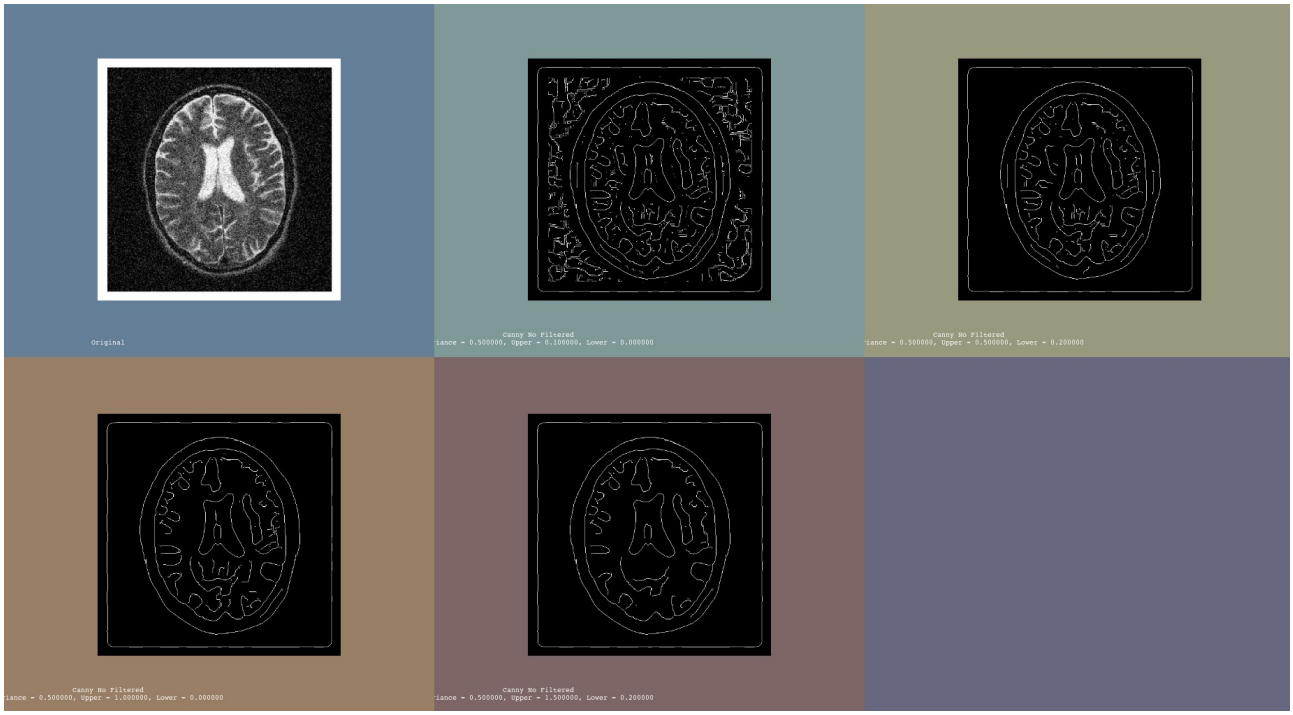






Podemos observar que mientras en imágenes con ruido los mejores resultados se obtienen con valores de la varianza en torno a 0.5, en la imagen sin ruido obtenemos mejores resultados con varianzas mayores, en torno a 1.5.

Veamos que ocurre ahora si jugamos con los parámetros de umbral:

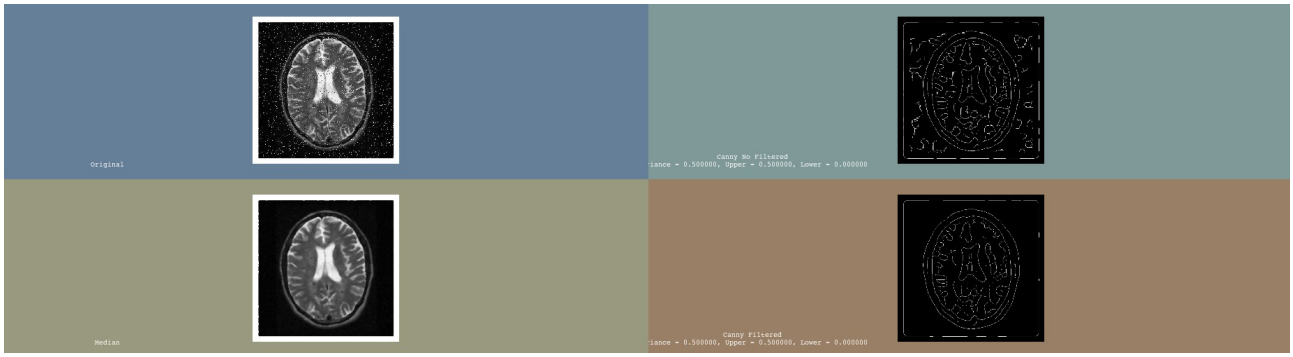


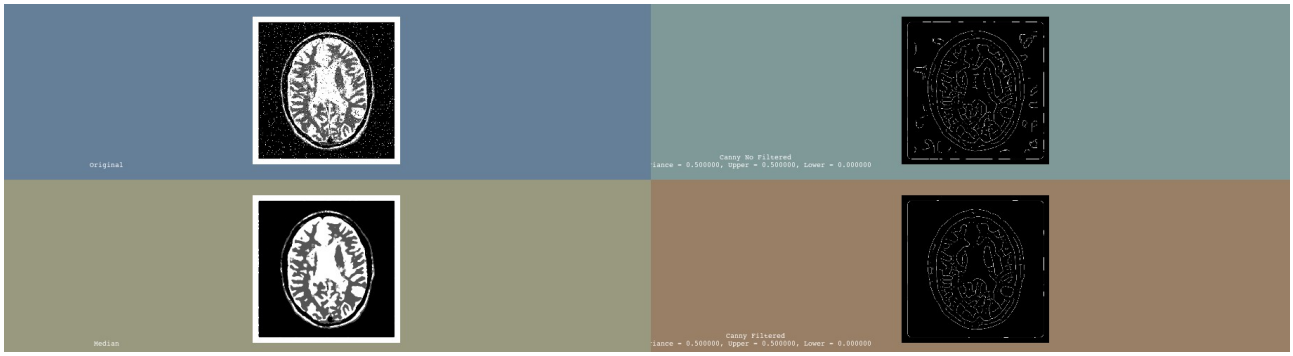
Como podemos ver, jugando con los parámetros de umbral se pueden obtener resultados aceptables en imágenes con y sin ruido. Para las imágenes con ruido necesitamos valores de umbral muy pequeños y tienden a perder detalle como era de esperar. Sin embargo, cuando tratamos imágenes sin ruido, obtenemos buenos resultados con valores de umbral mas altos obteniendo además más detalle en la imagen.

3.2 cannyEdgeDetectionFilterFiltered

En este caso vamos a aplicar un filtro de eliminación de ruido previo a la detección de ejes y vamos a ver como se comporta frente al caso anterior:

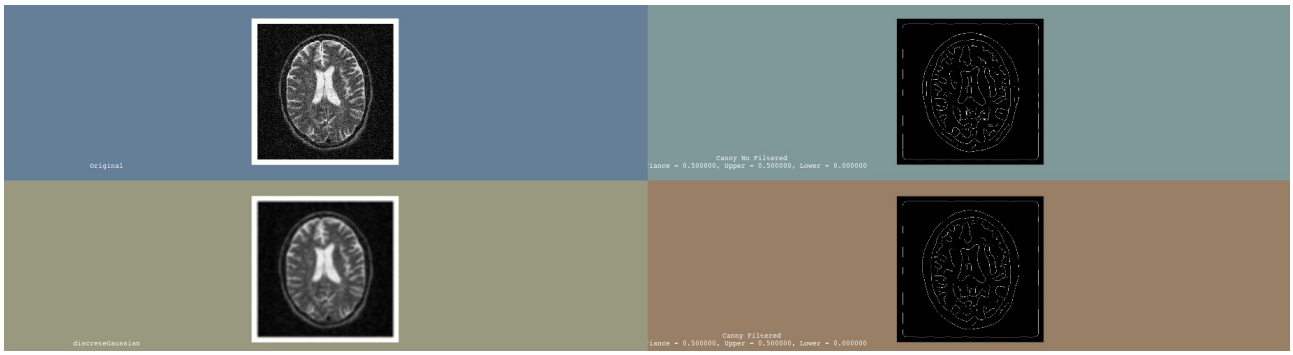
3.2.1 Imagen con ruido salt and peeper:





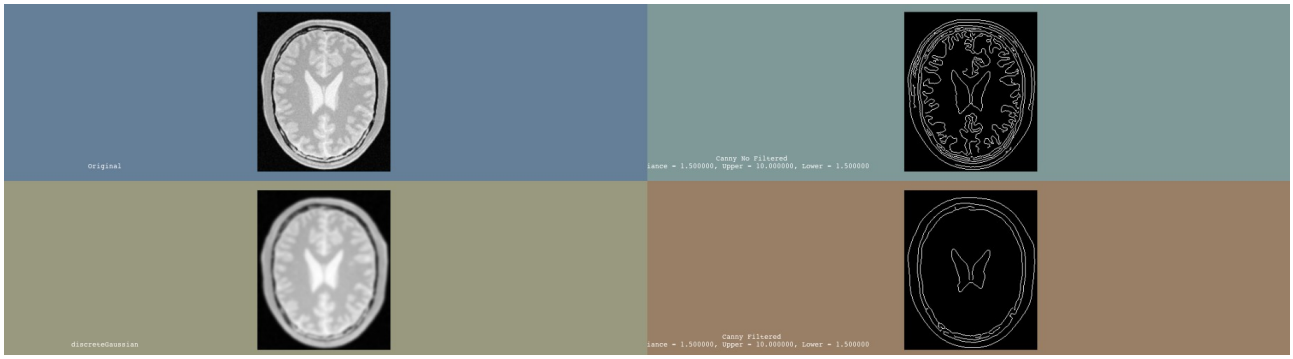
Como se observa, aplicando un filtro de mediana a las imágenes con ruido y manteniendo los parámetros del algoritmo, obtenemos un resultado mucho más limpio en la segmentación.

3.2.2 Imagen con ruido de tipo gaussiano:



Como se observa, aplicando un filtro de tipo gaussiano a la imagen con ruido y manteniendo los parámetros del algoritmo, obtenemos un resultado más limpio en la segmentación aunque en menor medida que en el caso anterior.

3.2.3 Imagen sin ruido:



Como era de esperar, si suavizamos una imagen que no tiene ruido lo que conseguimos es difuminar los bordes o limites de intensidad, esto implica que la visualización de la imagen pierda una gran cantidad de detalle. En el caso de tener una imagen limpia de ruido no es conveniente aplicar filtros de ruido ya que se pierde información de la imagen.