UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Implementación eficiente de un método fuzzy en dos fases para el filtrado de imágenes médicas

Trabajo Fin de Máster

Máster Universitario en Computación en la Nube y de Altas Prestaciones / Cloud and High-Performance Computing

AUTOR/A: Tendero Delicado, Celia

Tutor/a: Vidal Gimeno, Vicente Emilio

Cotutor/a externo: ARNAL GARCIA, JOSEP

Director/a Experimental: CHILLARON PEREZ, MONICA

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Universitat Politècnica de València

Máster en Computación en la Nube y de Altas Prestaciones

Trabajo de Fin de Máster

Implementación eficiente de un método fuzzy en dos fases para el filtrado de imágenes médicas

Alumno:

Celia Tendero Delicado

Tutores:
Vicente E. Vidal Gimeno
Josep Arnal García
Mónica Chillarón Pérez

17 de julio de 2023

Este Trabajo Fin de Máster es parte del proyecto de I+D+i TED2021- 131091B-I00, financiado por MCIN/ AEI/10.13039/501100011033/ y por la Unión Europea NextGenerationEU/PRTR.

Agradecimientos

En primer lugar me gustaría agradecerle a mis tutores su apoyo durante la realización de este trabajo. A Vicent Vidal por su apoyo estos meses y por introducirme en el grupo de investigación de imagen médica. A Josep Arnal por su apoyo en todos los asuntos teóricos del tema de filtrado, a pesar de la distancia. A Mónica Chillarón por su apoyo y ayuda desde que entré en el grupo de investigación, especialmente durante la fase experimental de este trabajo.

También quiero agradecer a los compañeros del máster, han hecho de este año una muy buena experiencia. Y también a los profesores del máster, que me han despertado interés por nuevas áreas de la informática.

Por último, agradecerle a mi familia y amigos su apoyo incondicional siempre, y especialmente estos meses de realización del trabajo.

Resumen

Actualmente, existe un esfuerzo investigativo para reducir la dosis de radiación a la que se someten los pacientes cuando se hacen pruebas de TAC, sin perder la calidad diagnóstica de las mismas. El problema de las técnicas de reducción de dosis es que estas suponen un aumento del ruido de las imágenes de TC reconstruidas, por lo que para solventarlo aparecen técnicas de filtrado de imagen.

En este trabajo se estudia una de estas técnicas de filtrado para imagen médica, un filtro basado en la lógica difusa para ruido mixto gaussiano-impulsivo, y se adapta para su funcionamiento en paralelo en GPU. Además, se han desarrollado diversas versiones del filtro que optimizan su velocidad.

Se ha estudiado el filtro sobre diversos datasets, con imágenes de tomografía y de radio, de fantomas y de pacientes reales, con ruido simulado y por baja dosis, e incluso con sinogramas antes de su reconstrucción. Sobre estas imágenes se han estudiado los parámetros óptimos del algoritmo de filtrado, el rendimiento de las diversas versiones del filtro, su eficiencia sobre distintos tipos de ruido y su eficiencia en distintos puntos del proceso de reconstrucción de imagen de TC.

Palabras clave: GPU, memoria pinned, CUDA, imágenes de TC, métrica fuzzy, mejora de imagen médica, reducción de radiación, ruido mixto gaussiano-impulsivo, filtrado de ruido

Abstract

Nowadays, there has been an investigative effort put through for reducing radiation dosage in patients that are subjected to CAT scans without losing the diagnostic quality of these tests. The problem is that dose reduction techniques cause an increase in the noise present in the reconstructed CT images, thus the appearence of image filtering techniques aimed at solving this.

This work studies one of these medical image filtering techniques, a fuzzy filter for mixed gaussian-impulsive noise, and adapts it for parallel GPU operation. In addition, several versions of the filter have been developed for optimizing its speed.

The filter has been studied on several datasets, with tomography and radio images, phantoms and real patients, with simulated and low-dose noise, and even with sinograms before reconstruction. On these images we have studied the optimal parameters for the filtering algorithm, the performance of the several versions of the filter, its efficiency on several noise types and its efficiency on different points of the CT image reconstruction process.

Keywords: GPU, pinned memory, CUDA, CT imaging, fuzzy metrics, medical image enhancement, radiation reduction, Gaussian-impulsive mixed noise, noise filtering

Resum

Actualment, existeix un esforç investigador per a reduir la dosi de radiació a la qual se sotmeten els pacients quan es fan proves de TAC, sense perdre la qualitat diagnòstica d'aquestes. El problema de les tècniques de reducció de dosi és que aquestes suposen un augment del soroll en les imatges de TC reconstruïdes, per la qual cosa per a solucionar-ho apareixen tècniques de filtrat d'imatge.

En aquest treball s'estudia una d'aquestes tècniques de filtrat per a imatge mèdica, un filtre basat en la lògica difusa per a soroll mixt gaussià-impulsiu, i s'adapta per al seu funcionament en paral·lel en GPU. A més, s'han desenvolupat diverses versions del filtre que optimitzen la seua velocitat.

S'ha estudiat el filtre sobre diversos datasets, amb imatges de tomografia i de radi, de fantomes i de pacients reals, amb soroll simulat i per baixa dosi, i fins i tot amb sinogrames abans de la seua reconstrucció. Sobre aquestes imatges s'han estudiat els paràmetres òptims de l'algorisme de filtrat, el rendiment de les diverses versions del filtre, la seua eficiència sobre diferents tipus de soroll i la seua eficiència en diferents punts del procés de reconstrucció d'imatge de TC.

Paraules clau: GPU, memòria pinned, CUDA, imatges de TC, mètrica fuzzy, millora d'imatge mèdica, reducció de radiació, soroll mixt gaussià-impulsiu, filtrat de soroll

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En esta sección se describe y contextualiza el problema que se va a tratar en este trabajo, explicando la motivación tras el mismo y los objetivos que se han marcado.

1.1. Descripción del Problema

La Tomografía Computarizada es una prueba de imagen médica esencial para el diagnóstico de multitud de enfermedades, pero no se trata de una prueba inocua. En tomografías convencionales, los niveles de radiación necesarios para obtener una imagen diagnóstica aceptable suponen un mayor riesgo para multitud de pacientes, ya sea por su edad [1], su índice de masa corporal [2] o porque se sometan a repetidas pruebas en cortos periodos de tiempo [3]. Para tratar de minimizar la exposición a radiación ionizante de los pacientes aparecen las tomografías de baja dosis, que a base de reducir el voltaje y la corriente del tubo de rayos-x pueden reducir la dosis de radiación de la prueba.

El principal problema de las tomografías de baja dosis es que la reducción de la radiación supone que aumente el ruido de las imágenes reconstruidas, y esto puede dificultar su uso como prueba diagnóstica. Los orígenes de este ruido son principalmente las propiedades de los fotones de rayos-x y el ruido electrónico del detector [4]. Como solución, se han propuesto varias técnicas de filtrado que puedan eliminar el ruido gaussiano, impulsivo y de poisson de los sinogramas y de las imágenes reconstruidas [5]. Uno de estos métodos es el propuesto en [6].

1.2. Motivación

Es precisamente la reducción de la dosis de radiación de las tomografías computarizadas lo que motiva este trabajo. Conforme se reduce la radiación en la adquisición, aumenta el nivel de ruido, por lo que el filtrado es necesario para producir imágenes de diagnóstico válidas.

El generar imágenes de diagnóstico válidas a partir de imágenes obtenidas con menor dosis implica que se puedan realizar pruebas diagnósticas en más cantidad de pacientes, disminuyendo el riesgo de sufrir patologías derivadas de la exposición a radiación no solo de aquellos pacientes que por su edad o sus características fueran especialmente vulnerables, sino de cualquier paciente.

En situaciones de emergencia donde se utilizan escáneres de rayos-x portátiles, que por su tamaño no tienen la misma potencia, si la calidad de la imagen es suficiente se puede evitar que el paciente tenga que visitar el hospital para realizarse una prueba de TC. Esto reduce el riesgo de infecciones que supone una visita hospitalaria, permitiendo no aumentar la carga de trabajo de los servicios del hospital y sobretodo mejorando la calidad de atención a los pacientes, que podrán ser diagnosticados y atendidos in situ por el personal de emergencias [7].

Se utilizan también imágenes tomográficas y de rayos-x para la detección de objetos prohibidos en la seguridad en aeropuertos y para detectar anomalías en cadenas de producción de alimentos. Aumentar la calidad de las imágenes mediante el filtrado podría suponer acelerar los procesos de escaneo reduciendo el número de vistas necesario. Las técnicas de filtrado podrían utilizarse también en conjunción con las técnicas de diferenciación de materiales, ya que estas requieren alterar el voltaje del escáner [8].

La técnica de filtrado que se analiza en este trabajo obtiene muy buenos resultados para el ruido que aparece en reconstrucciones de TC de baja dosis, pero no es un proceso rápido. Para procesar todos los cortes que se obtienen en una prueba de TC mediante un filtro sin optimizar se necesita mucho tiempo, lo que en un hospital supondría tener la máquina bloqueada durante horas. La solución que se propone es la implementación de este filtro y variantes del mismo en GPU, para reducir los tiempos de cómputo.

1.3. Estado del Arte

El campo del filtrado de imágenes es muy amplio, existiendo múltiples filtros distintos para cada tipo de ruido existente [9, 5]. En lo concerniente a las imágenes médicas interesan los filtros que estén pensados para los ruidos gaussiano, impulsivo y de poisson, por ser los más comunes en este tipo de imágenes.

En [10] los autores estudian una técnica de filtrado para ruido impulsivo en imágenes a color. El ruido impulsivo está normalmente causado por el mal funcionamiento de los sensores y otro hardware durante el proceso de reconstrucción, transmisión o almacenamiento de imagen. Este tipo de ruido afecta algunos píxeles individuales, cambiando sus valores originales. El modelo de ruido impulsivo más común es el de Sal y Pimienta, también llamado ruido de valor fijo, que considera que el nuevo valor erróneo del píxel es un extremo dentro del rango de la señal.

En dicho trabajo proponen un algoritmo paralelo implementado en dos fases para eliminar el ruido impulsivo de una imagen digital utilizando computación tanto en CPU como GPU, basado en el concepto de *peer groups* y una métrica Euclídea.

En [11] los autores estudian una técnica de filtrado mixto gaussiano-impulsivo para imágenes a color. En la adquisición de imágenes es normal que se introduzca ruido gaussiano y es común que aparezca adicionalmente ruido impulsivo debido a problemas de almacenaje o transmisión. El algoritmo propuesto se encarga de filtrar ambos tipos de ruido simultáneamente, lo que reduce la complejidad computacional que causaría el pasar dos filtros distintos sobre la imagen, haciendo que sea más viable su uso en aplicaciones reales.

En [12, 13] se implementó este algoritmo para imágenes en escala de grises, para su uso en el filtrado de imagen médica. Los resultados de esta implementación fueron buenos, pero el filtro resultante no es lo suficientemente rápido para hacer filtrado en tiempo real. Este es el filtro sobre el que se realizarán las optimizaciones, en GPU solamente porque en el trabajo [12] ya se abordó en CPU utilizando OpenMP y MPI.

En [14] se propone el concepto de *fuzzy peer groups* para diseñar un filtro de imagen en color de dos pasos en cascada que pasa de un filtro impulsivo basado en reglas difusas a un método de filtrado medio difuso. Ambos filtros trabajan sobre el mismo *fuzzy peer group* para así reducir los costes computacionales. El filtro propuesto puede filtrar imágenes con ruido impulsivo, con ruido gaussiano y aquellas imágenes que presenten ambos tipos de ruido. Desafortunadamente el coste de este filtro hace imposible su uso en aplicaciones en tiempo real.

En [6] se adaptaron los filtros propuestos en este algoritmo para su uso con imágenes médicas, adicionalmente se optimizó y paralelizó para mejorar su coste computacional

y que así pudiese ser empleado en aplicaciones en tiempo real. La paralelización se realizó para sistemas multicore utilizando OpenMP y para GPU utilizando CUDA, alcanzando un speedup de más de 8300 en GPU respecto a la versión secuencial.

En [15] los autores proponen un filtro para imágenes de TC por rayos-x, para la eliminación de artefactos en anillo. Este filtro está pensado para tomografías de rayos-x de contraste en fases, que captura detalles en tejidos blandos, ya que el bajo contraste de estas imágenes y del tipo de ruido hace que otros filtros no sean aptos.

En [16] los autores aceleraron mediante el uso de CUDA los algoritmos de filtrado en paralelo, uno basado en el algoritmo NLM [17] y otro basado en el algoritmo KNN, pensados para filtrar ruido en imágenes de gran tamaño, como pueden ser las imágenes de satélite. La implementación en GPU consiguió un speedup de hasta 40 veces con respecto a la implementación paralela en CPU de estos filtros.

En años recientes se han estado estudiando técnicas basadas en redes neuronales para el procesamiento de imagen. En [18] se propone una técnica basada en redes neuronales convolucionales (CNN) para preservar las estructuras de la imagen tras el filtrado de imágenes de tomografía computarizada de baja dosis.

1.4. Objetivos

El principal objetivo de este trabajo es implementar un filtro eficiente paralelizado en CUDA que permita el filtrado de imágenes médicas en aplicaciones en tiempo real, a partir del filtro gaussiano-impulsivo seleccionado. Para alcanzar este objetivo principal se han marcado otros objetivos:

- Estudiar el algoritmo de filtrado de lógica difusa para así comprender los principios detrás de su uso en imágenes a color y cómo puede alterarse para su uso en imágenes médicas.
- Comprender los principios de la lógica difusa.
- Comprender las métricas de calidad de imagen.
- Realizar un análisis de la versión secuencial del filtro existente.
- Implementar dicho filtro secuencial de manera eficiente utilizando el paralelismo en GPU con CUDA.
- Estudiar las características del filtro para realizar optimizaciones sobre el mismo.

- Realizar un estudio sobre los distintos parámetros del filtro para determinar cuales son los óptimos en función de calidad.
- Realizar un estudio de la eficiencia del filtro en función de distintos parámetros y comparando con la versión secuencial.

1.5. Estructura del Documento

Se ha estructurado el documento en seis capítulos, seguidos de la bibliografía y de unos apéndices en los que se puede consultar los códigos Matlab desarrollados para la preparación de las imágenes.

En el Capítulo 1 se introduce el problema a tratar, explicando la motivación tras el mismo y estudiando el estado del arte en este área. Se explican también los objetivos generales y específicos que se tratan de alcanzar con nuestro trabajo.

En el Capítulo 2 se explican los fundamentos teóricos detrás del tema de estudio. Se explican brevemente los conceptos de TAC, lógica difusa, métricas de calidad, desarrollo en CUDA y se ponen en contexto los procesos de filtrado.

En el Capítulo 3 se explica el algoritmo detrás del filtro mixto gaussiano-impulsivo.

En el Capítulo 4 se realiza una breve explicación de la implementación secuencial del filtro, y se explica cómo es la implementación paralela, así como en qué difiere, por qué y las distintas versiones implementadas.

En el Capítulo 5 se explica el diseño de las pruebas y se analizan los resultados de los estudios sobre la eficiencia y calidad de los filtros implementados.

Por último, se cierra con las conclusiones en el Capítulo 6, así como qué líneas de trabajo futuro se han presentado tras el estudio.

1.5.1. Organización del trabajo

En la figura 1.1 mostramos un esquema del proceso que se ha seguido para la realización de este trabajo, partiendo del estudio bibliográfico y terminando con la redacción de este documento.

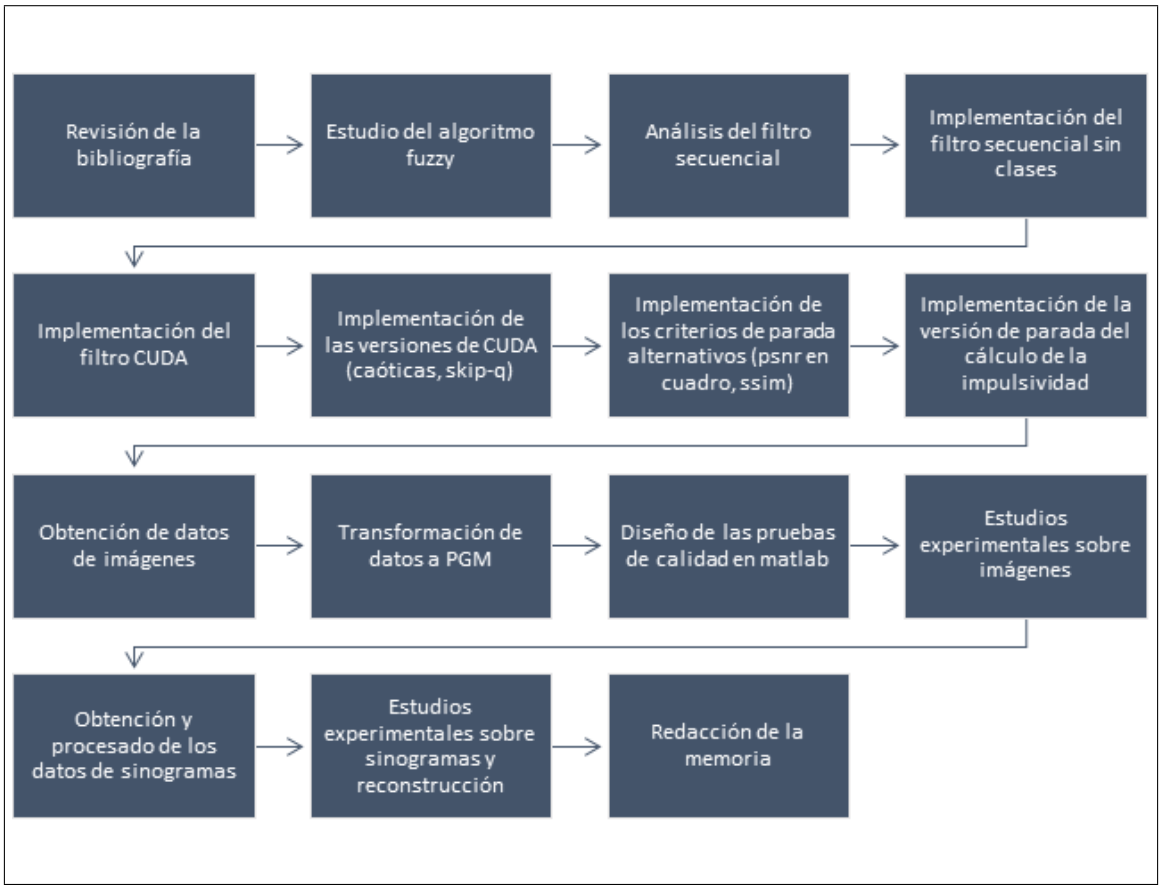

Figura 1.1: Esquema de la organización del trabajo realizado.

Capítulo 2

Fundamentos y Conceptos

En esta sección se explicarán los conceptos de la Tomografía Axial Computarizada y del filtrado de imagen relevantes para el estudio. También, se explicarán el entorno y la tecnología utilizada para el desarrollo del método eficiente para GPU.

2.1. Tomografía Axial Computarizada

Una tomografía es una imagen de corte de un objeto creada a partir de los datos de transmisión o reflexión producidos al proyectar un haz de rayos desde diferentes posiciones a su alrededor [19]. Normalmente este haz de rayos consiste en rayos-x, tal como fue originalmente propuesto en su escáner por Hounsfield en 1972.

El escáner original de Hounsfield utilizaba técnicas de reconstrucción algebraicas y producía imágenes en resolución 80x80. Con la introducción de algoritmos convolucionales de retroproyección para las reconstrucciones se consiguieron mejoras en eficiencia y exactitud, lo que permitió que los escáneres comerciales tengan sistemas capaces de reconstruir en resolución 256x256 y 512x512 píxeles.

Una proyección es la combinación de un set de integrales lineales, que a su vez son la representación de la atenuación total sufrida por un haz de rayos-x conforme atraviesa el objeto en línea recta.

Una proyección paralela es aquella que se obtiene colocando el detector y la fuente de rayos-x paralelamente en lados opuestos del objeto. En la figura 2.1 (a) se puede ver un esquema de la colocación del detector y la fuente sobre el plano xy.

Una proyección de haz en abanico es aquella que se obtiene con una fuente colocada

en una posición fija relativa a una linea de detectores. En la figura 2.1 (b) se puede ver el esquema de colocación para la proyección de haz en abanico.

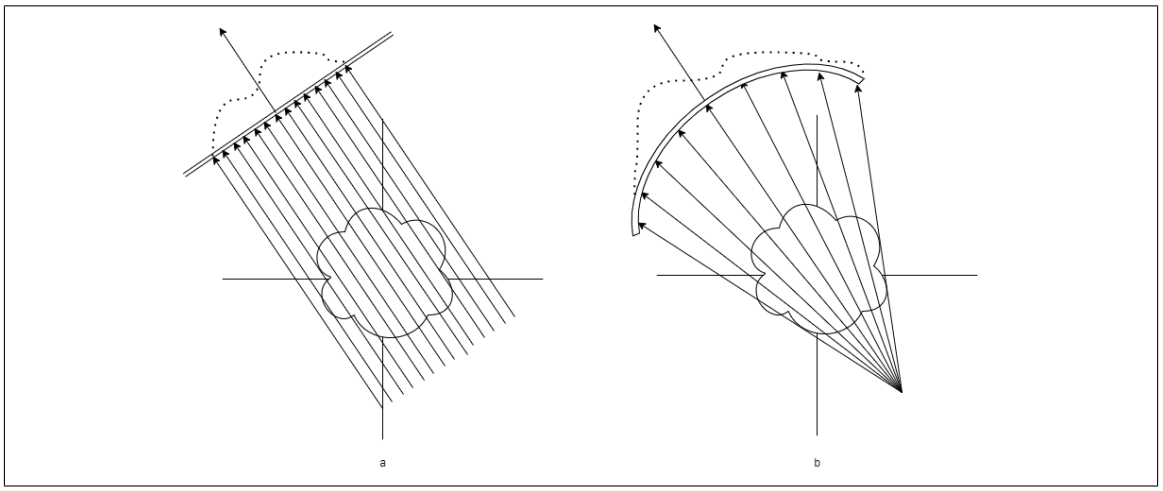

Figura 2.1: (a) Proyección paralela. (b) Proyección de haz en abanico.

Proyección paralela

Solo un número finito de proyecciones puede tomarse, por lo que la reconstrucción ha de hacerse interpolando desde los radiales. Dada que la densidad de los puntos radiales se vuelve más dispersa conforme nos alejamos del centro, el error se vuelve también mayor. Esto implica una mayor degradación en los elementos de alta frecuencia de la imagen.

El algoritmo de reconstrucción más frecuentemente utilizado es el de retroproyección filtrada simple. En este algoritmo cada proyección representa una medida casi independiente del objeto.

La adquisición de los datos en este tipo de proyecciones consiste en escanear en línea la longitud de la proyección, rotar los detectores y proyectores un ángulo dado, escanear en esa nueva línea y así hasta completar una rotación. Esto resulta en que se necesiten varios minutos tan solo para la adquisición de los datos.

Proyección de haz en abanico

La adquisición de los datos en las proyecciones en abanico requieren menos tiempo, pues en vez de escanear la línea se lanza el haz de rayos en abanico y se captura la

proyección toda a la vez. Existen dos tipos de proyecciones en abanico dependiendo de si las muestras se toman en intervalos equiespaciados o equiangulados.

El algoritmo de reconstrucción utilizado en este tipo de proyecciones es el de retro-proyección filtrada pesada.

Como mínimo es necesario medir las proyecciones de un objeto para ángulos entre 0 y 180^{o}, y en longitud igual a la anchura máxima del objeto. Esto en las proyecciones en abanico supone que el ángulo de proyecciones tenga que ser 180^{o} más el ángulo del haz.

Los sinogramas se tratan de las transformadas de Radón de las proyecciones (en coordenadas rectangulares o polares) en las que cada punto representa una integral de línea.

La forma convencional de conseguir imágenes de un objeto tridimensional es utilizar los métodos de proyección y reconstrucción de dos dimensiones en planos sucesivos. Otro método es el uso de haz en cono, utilizando un detector bidimensional.

Por otra parte existen varias técnicas para obtener imágenes de Tomografía Computarizada reduciendo la dosis de radiación absorbida por el paciente. Para ello, se puede reducir la corriente del tubo, reducir su voltaje o reducir el número de vistas, lo que se conoce como proyecciones dispersas.

En la figura 2.2 se muestra un posible esquema del proceso de reconstrucción de imagen médica de Tomografía Computarizada. Se pueden distinguir tres fases distintas desde la adquisición hasta que se obtiene la imagen final:

1. **Sinograma** Los datos adquiridos del escáner se encuentran en forma de señal que se denomina sinograma. Este sinograma no suele salir sin ruido, por lo que en una primera fase debemos analizar los tipos de ruido presentes y filtrarlos acordemente. El tipo de ruido que puede haber presente en el sinograma puede originarse por motivos físicos o específicos de la máquina, o pueden originarse debido a las técnicas utilizadas para la reducción de dosis.

2. **Reconstrucción** A continuación, se reconstruye la imagen de TC a partir del sinograma filtrado. La reconstrucción puede realizarse utilizando métodos analíticos o algebraicos (ya sean directos o iterativos) aunque recientemente han empezado a aparecer técnicas de reconstrucción basadas en inteligencia artificial, utilizando redes neuronales y redes generativas antagónicas (GANs).

3. **Imagen** En métodos iterativos en una tercera fase se realiza una evaluación de la imagen obtenida. Si la imagen es de calidad suficiente, se da por buena y se para el proceso. Si fallase alguno de los criterios, se aplicaría un filtrado de la imagen y una técnica de aceleración.

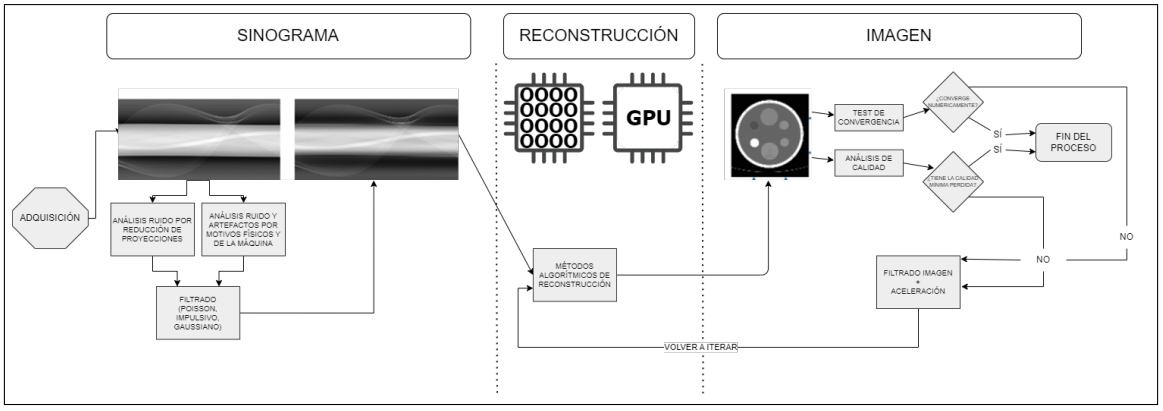

Figura 2.2: Proceso de reconstrucción de una imagen de TC

2.2. Lógica Fuzzy

Se utilizará el trabajo [20] para explicar los fundamentos de la lógica difusa o lógica fuzzy. La lógica fuzzy es un sistema preciso de racionamiento y computación en el que el objeto de estudio es impreciso e indeterminado. Aporta un sistema para deducir preguntas imprecisas para las que la respuesta no es verdadero o falso.

Conjuntos difusos Los conjuntos difusos se tratan de clases con límites difusos, son las respuestas a las preguntas imprecisas, y en vez de tomar valores binarios su valor se encuentra entre [0,1] tomando los límites del rango como verdadero (1) y falso (0).

En la figura 2.3 se muestra un ejemplo del conjunto difuso *Es de día*. Los límites de membresía a este conjunto son difusos, por ejemplo, la preposición *A las 7:30 es de día* es cierta solo la mitad del año por lo que su valor es 0,5.

Granulación La granulación es el particionado de un conjunto en gránulos. Los gránulos son conjuntos de valores de atributos agrupados por su indistinción, equivalencia, similitud, proximidad o funcionalidad.

En el ejemplo anterior, la granulación de las horas del día sería en los conjuntos *De día* y *De noche*.

Variables lingüísticas Las variables lingüísticas son variables granulares que tienen etiquetas lingüísticas para sus valores granulares. En nuestro ejemplo son los nombres que les hemos dado a los conjuntos, *De día* y *De noche*.

Reglas difusas if-then Las reglas if-then difusas tienen la forma siguiente: SI X es A ENTONCES Y es B. Donde A y B se tratan de conjuntos difusos. Son el tipo de reglas en las que se basa la lógica del algoritmo que estudiaremos.

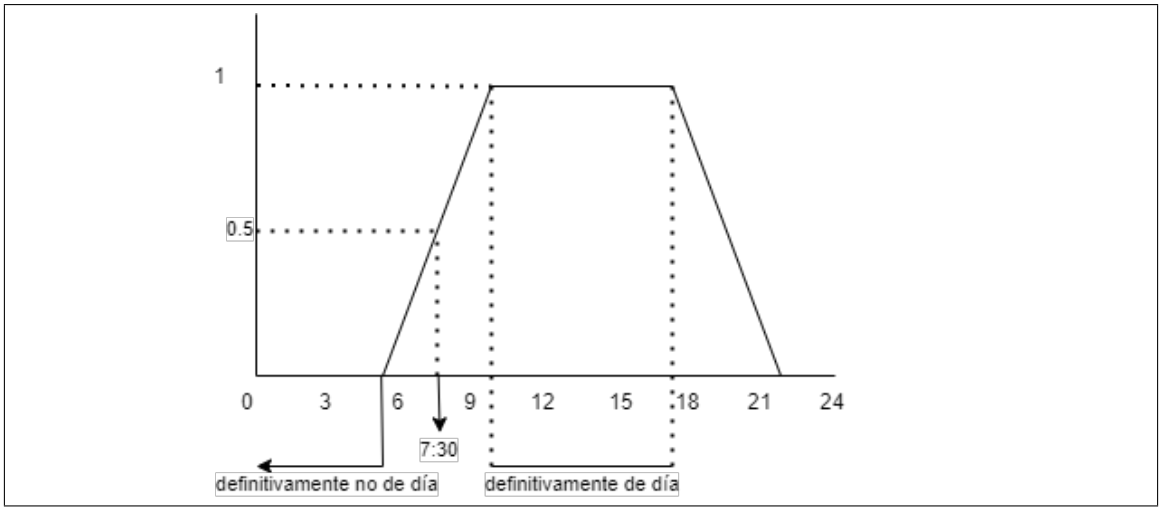

Figura 2.3: Conjunto difuso *Es de día*

2.3. Imágenes de Entrada

El conjunto de imágenes utilizado en los estudios proviene de distintos datasets, que se explicarán a continuación. Los filtros desarrollados toman como formato de entrada el formato PGM, por lo cual estos datos han tenido que ser adaptados. A continuación se hablará sobre la fuente de los datos de entrada y el formato de imagen.

2.3.1. Conjunto de datos

Dataset Mayo

Las imágenes de TC utilizadas para los estudios provienen de la base de datos de imagen médica de la clínica Mayo [21]. Esta base de datos cuenta con hasta 13.013.532 de imágenes de Tomografía Computarizada, almacenadas en formato DICOM anonimizado.

El formato DICOM (Digital Imaging and Communications in Medicine) es el estándar internacional de comunicación y gestión de imagen médica, está reconocido por el ISO como el estándar 12052 [22].

Las imágenes de TC han sido extraídas de la base de datos DICOM-CT-PD [23]. Se utilizarán tres conjuntos de imágenes reconstruidas en dosis completa y en baja dosis simulada de tórax (C012), abdomen (L064) y cabeza (N005), los conjuntos

estaban originalmente en unidades Hounsfield, en el apéndice A se muestra el script desarrollado para obtener las imágenes PGM de estos datos. Cada uno de los cortes tiene dimensiones de $512 * 512$ px. Los conjuntos en baja dosis se obtuvieron introduciendo ruido al dataset antes de la reconstrucción utilizando una versión modificada para ajustarse al escáner de la técnica descrita en [24]. Los conjuntos de abdomen y cabeza fueron modificados para simular exámenes adquiridos a un 25 % de la dosis completa, y el de tórax para simular un 10 % de la dosis completa. En la figura 2.4 se pueden observar un corte de cada uno de los conjuntos a dosis completa y a baja dosis simulada, y se puede apreciar como la imagen de tórax presenta más ruido que las de los otros conjuntos.

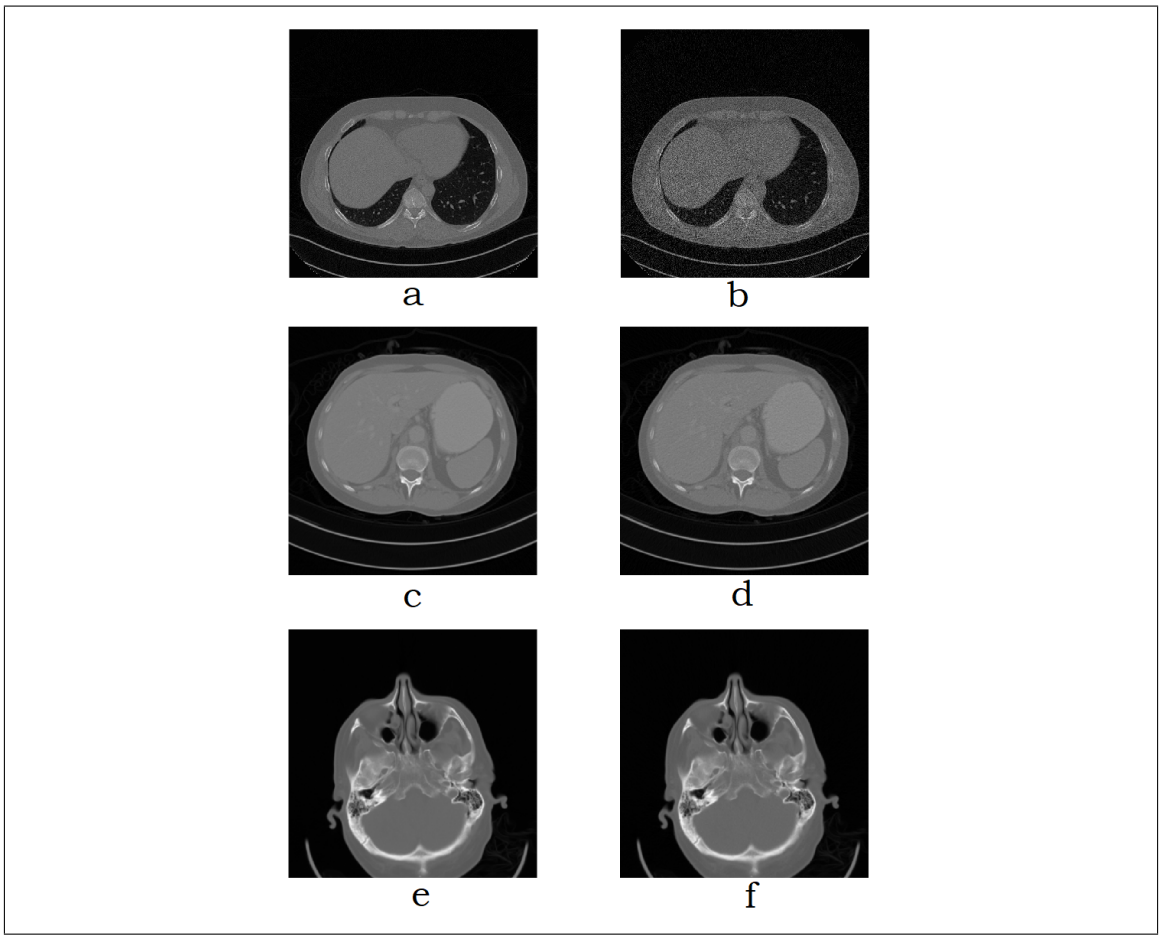

Figura 2.4: (a) Corte 200 de C012 en dosis completa. (b) Corte 200 de C012 en baja dosis. (c) Corte 100 de L064 en dosis completa. (d) Corte 100 de L064 en baja dosis. (e) Corte 10 de N005 en dosis completa. (f) Corte 10 de N005 en baja dosis.

Dataset Shepp-Logan

Para el estudio se utilizarán tanto conjuntos de datos de sinogramas como imágenes reconstruidas de TC.

Para el filtrado de sinogramas se utilizarán un conjunto a dosis completa (sin ruido) y con ruido de poisson del fantoma Shepp-Logan [25]. Se utilizó la librería Astra [26] en Matlab para generar los sinogramas e introducir de manera artificial ruido de poisson en el conjunto. En la figura 2.5 se puede ver un corte de las imágenes de sinograma en el conjunto sin ruido y con ruido, y de su reconstrucción. Las imágenes reconstruidas a partir de los sinogramas tienen dimensiones de $391 * 391px$, mientras que las imágenes PGM producidas a partir de los sinogramas tienen dimensiones $1600 * 1024\ px$.

Imágenes de grandes dimensiones

Se ha utilizado adicionalmente otra imagen de TC de un corte en vista axial de cerebro, obtenida del conjunto de datos de Radiopaedia, cortesía del profesor Frank Gaillard con rID 35508 [27]. En la figura 2.6 se puede ver la imagen del corte axial original y con el ruido mixto gaussiano-impulsivo introducido con Matlab. Estas imágenes tienen dimensiones de $1024 * 904\ px$.

También se utilizó una imagen de rayos-x del fantoma Alderson RANDO, un maniquí de torso de mujer consistente de un esqueleto humano sintético dentro de una masa comparable al tejido blando humano. Está diseñado para tener niveles de absorción de radiación similares a los del tejido humano. Este maniquí se escaneó en nuestro laboratorio con un dispositivo de imágenes AGFA, la imagen utilizada fue obtenida a 70 kV y 1 mA. En la figura 2.7 se puede ver el aspecto del fantoma y en 2.8 se puede ver la imagen original de radiografía del fantoma y con el ruido mixto gaussiano-impulsivo introducido con Matlab. Estas imágenes tienen dimensiones de $3730 * 3062\ px$, siendo significativamente más grandes que el resto de imágenes de prueba.

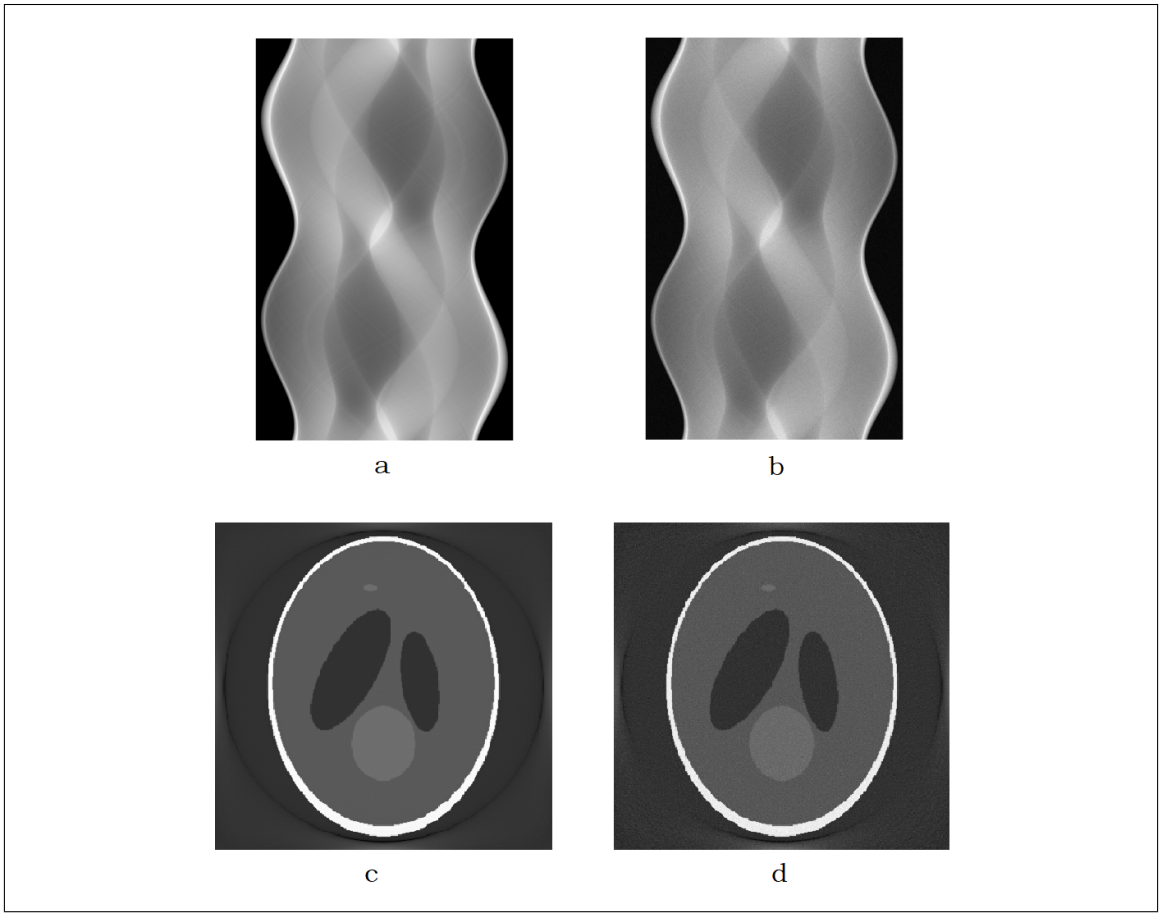

Figura 2.5: (a) sinograma, corte 200. (b) sinograma *(a)* con ruido de poisson. (c) Reconstrucción de *(a)*. (d) Reconstrucción de *(b)*.

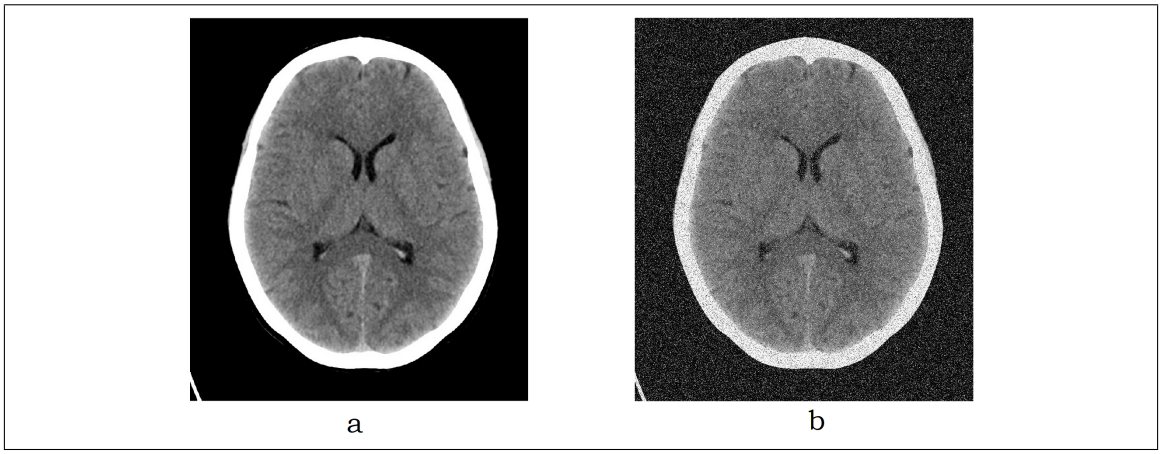

Figura 2.6: (a) Imagen de corte axial. (b) Imagen *(a)* con ruido gaussiano-impulsivo (g=20, i=0,2).

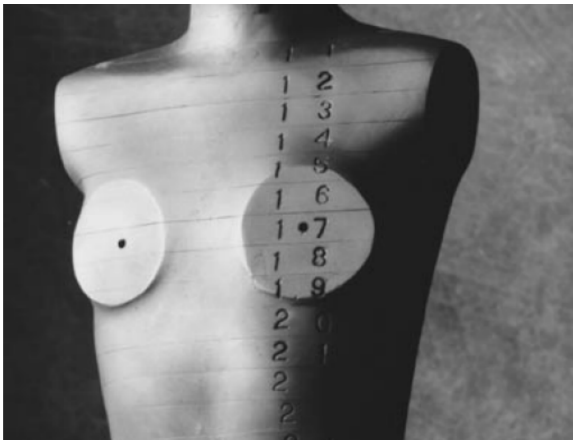

Figura 2.7: Fantoma Alderson RANDO. Imagen obtenida de [28].

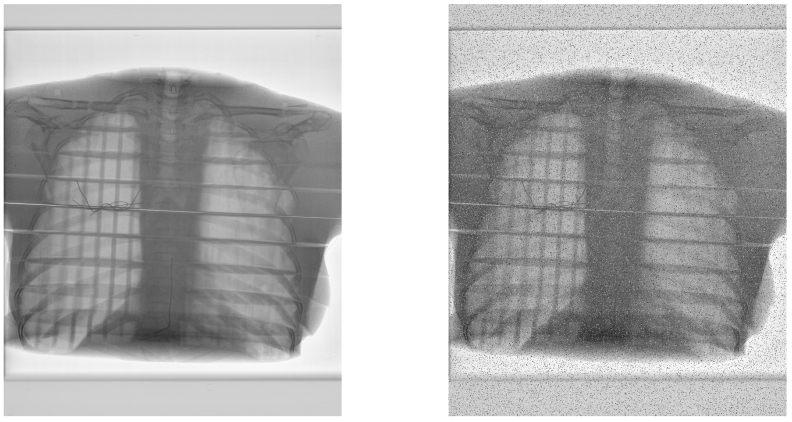

Figura 2.8: (a) Radiografía del fantoma de pecho. (b) Imagen *(a)* con ruido gaussiano-impulsivo (g=10, i=0,1).

2.3.2. Formato PGM

El formato PGM (Portable Gray Map) definido por el proyecto Netpbm se trata de una representación de una imagen en escala de grises en un array de enteros [29]. La estructura de un fichero PGM se compone por las cabeceras, que definen tipo de archivo (P2 o P5 para PGM), ancho, alto, el valor de gris máximo de la imagen (MaxVal) y el array de enteros en el rango [0,MaxVal], donde cada elemento representa un único píxel de la imagen. El valor del gris máximo puede tener valor de hasta 65536, pero si es menor o igual a 255 cada valor del array ocupa tan solo 1 byte. Todas las imágenes de prueba tienen un MaxVal inferior a 256.

El programa de filtrado sobre el que se trabaja toma como entrada datos en formato PGM. Es necesario, en el caso de las imágenes provenientes de la base de datos de la clínica Mayo, transformar el conjunto de datos almacenados en coma flotante a imágenes PGM. Para estos procesos de transformación se hace uso de Matlab.

2.3.3. Tipos de ruido

Se utilizará también Matlab para introducir distintos tipos de ruido a la imagen y así generar nuevos conjuntos de imágenes de prueba. Los tipos de ruido que se van a analizar, natural y simulados, son gaussiano, impulsivo y de poisson. En la figura 2.9 se puede ver la diferencia entre los distintos tipos de ruido. Se ha introducido el ruido sobre una imagen del conjunto C012 de la que se tiene una versión en baja dosis, para que se pueda apreciar la diferencia entre el ruido natural y el simulado.

Definimos la distribución del ruido de poisson y del ruido impulsivo a partir de [30], y el ruido de poisson a partir de [31]. El ruido gaussiano aditivo presenta la siguiente distribución de probabilidad, en función de la desviación estándar (σ):

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}. \tag{2.1}$$

Por otro lado, en el ruido impulsivo el pixel x* se obtiene de valores enteros aleatorios d distribuidos uniformemente en el intervalo [0,255] con probabilidad p:

$$x^* = \begin{cases} d & \textit{con probabilidad } p, \\ x^* & \textit{con probabilidad } 1-p. \end{cases} \tag{2.2}$$

Por último, el ruido de poisson se modela como una distribución gaussiana en la que la varianza depende de la probabilidad de fotones, su distribución de probabilidad es entonces:

$$p(x) = \frac{e^{-\lambda t}(\lambda t)^x}{x!}, \tag{2.3}$$

donde λ se trata del número esperado de fotones, que es equivalente a su varianza.

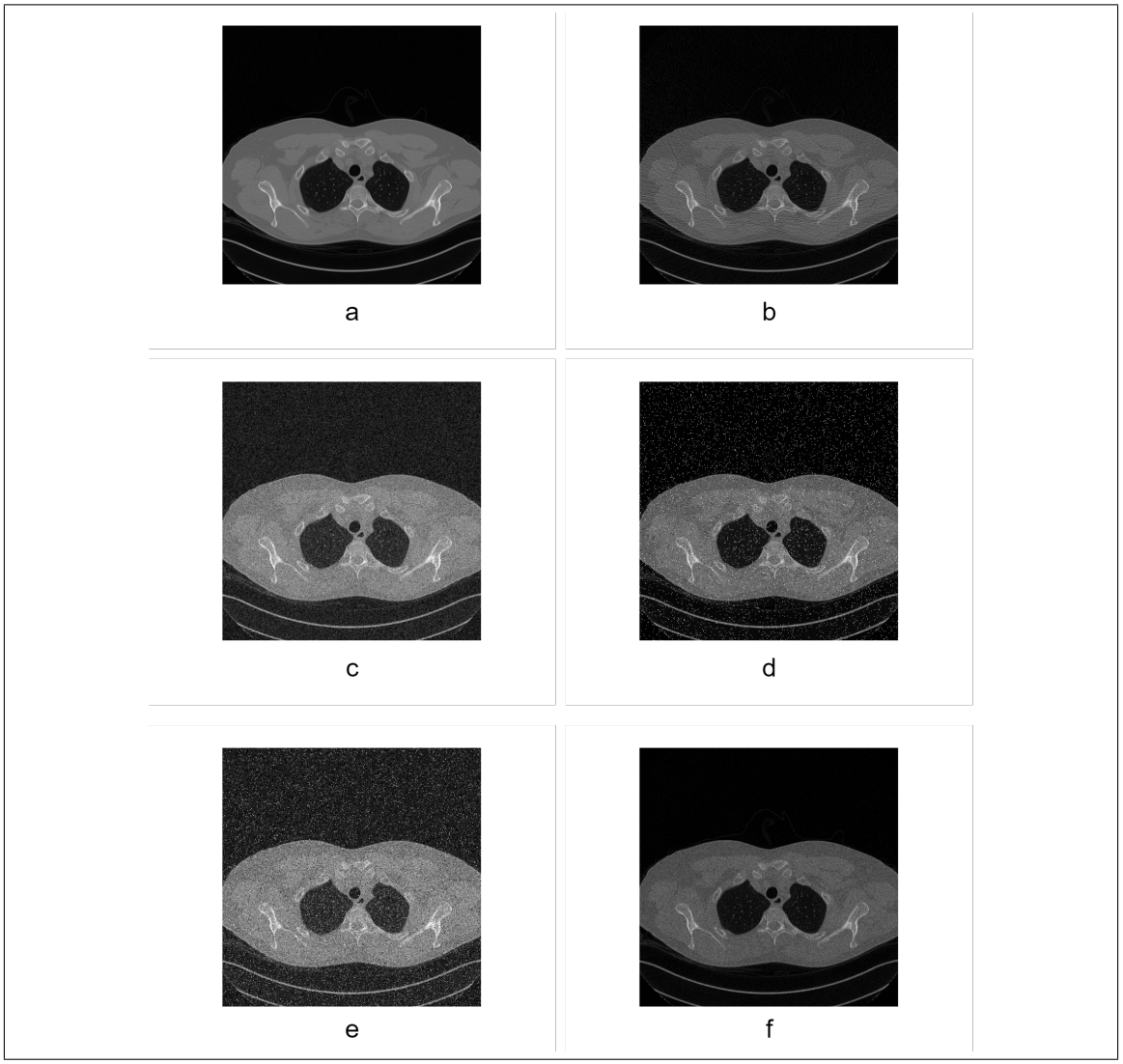

Figura 2.9: (a) Imagen sin ruido. (b) Imagen de baja dosis. (c) Imagen con ruido gaussiano simulado. (d) Imagen con ruido impulsivo simulado. (e) Imagen con ruido mixto gaussiano-impulsivo simulado. (f) Imagen con ruido de poisson simulado.

2.4. Métricas de calidad

Se utilizarán distintas métricas de calidad a la hora de evaluar los resultados, en el caso del PSNR el cálculo está incluido en el programa y se puede utilizar como un criterio de parada a la hora de estudiar las características y rendimiento del filtro. El resto se estudiarán mediante el uso de Matlab, que cuenta con funciones integradas para su cálculo.

2.4.1. MSE

El MSE (Mean Squared Error) se trata de una métrica de calidad de imagen computada haciendo la media de las diferencias de intensidad al cuadrado entre los píxeles de la imagen distorsionada y los de referencia. Cuanto más cercano a 0 sea el valor, mejor será la calidad. En la ecuación 2.4 se muestra como se calcula esta métrica, en esta ecuación x_i es el nivel de gris del pixel i la imagen original e y_i el de la imagen procesada [32].

$$MSE(x,y) = \left[\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2\right]^{1/2}, \tag{2.4}$$

donde n es el número de píxeles de la imagen (filas por columnas).

2.4.2. PSNR

El PSNR (Peak Signal to Noise Ratio) es una métrica de calidad de imagen derivada del MSE. En imágenes de 8 bits, como las PGM, los valores del PSNR pueden llegar hasta 50 dB [33], siendo mejor la calidad conforme mayor sea el valor. Tanto el PSNR como el MSE se tratan de métodos analíticos fáciles de calcular, pero no tienen por qué corresponderse con la calidad percibida. En la ecuación 2.5 se muestra como se calcula esta métrica a partir del PSNR para una imagen de 8 bits en escala de gris. En esta ecuación, los parámetros que se pasan a la función para el cálculo del MSE son x la imagen original y y la imagen procesada [34].

$$PSNR(x,y) = 10\log_{10}\left(\frac{MAX^2}{MSE(x,y)}\right), \tag{2.5}$$

donde $MAX = 255$ para el formato PGM de 8 bits.

2.4.3. SSIM

El SSIM (Structural Similarity Index) es una métrica de calidad que tiene en cuenta la estructura de la imagen original, siendo una métrica más adecuada a la calidad percibida. Los valores de SSIM están en el rango [0,1], para imágenes médicas se considera aceptables aquellas imágenes con SSIM superior a 0,9. En la ecuación 2.6 se muestra como calcular el SSIM a partir de la función de comparación de luminancia ($l(x,y)$), de contraste ($c(x,y)$) y de estructura ($s(x,y)$). Los parámetros α, β y γ son constantes de peso positivas [35].

$$SSIM(x,y) = l(\mathbf{x},\mathbf{y})^{\alpha} \cdot c(\mathbf{x},\mathbf{y})^{\beta} \cdot s(\mathbf{x},\mathbf{y})^{\gamma}. \tag{2.6}$$

En las ecuaciones 2.7 se muestra como calcular las funciones de comparación. $C1$, $C2$ y $C3$ son las constantes de regularización que se obtienen a partir del rango de valores de la imagen, para las PGM de 8 bits tendrán valores $C1 = 0{,}01 * 255^2$, $C2 = 0{,}03 * 255^2$ y $C3 = C2/2$.

$$\begin{aligned} l(\mathbf{x},\mathbf{y}) &= \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \\ c(\mathbf{x},\mathbf{y}) &= \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \\ s(\mathbf{x},\mathbf{y}) &= \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}. \end{aligned} \tag{2.7}$$

El parámetro μ se corresponde con la luminancia de la imagen, calculando la media del valor de los píxeles y σ se corresponde con el contraste, calculando la desviación estándar del valor de los píxeles. En las ecuaciones 2.8 y 2.9 se muestra como se calculan estas métricas.

$$\mu_x = \frac{1}{N}\sum_{n=1}^{N} x_n, \tag{2.8}$$

$$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(x_n - \mu_x)^2}. \tag{2.9}$$

2.5. Entorno de estudio

Para el estudio de los programas se hará uso del último servidor de cálculo adquirido por el grupo de investigación GCCAPPI del Departamento de Sistemas Informáticos y Computación, denominado Aurus.

Aurus se trata de un servidor con dos procesadores AMD EPYC 7282® con una frecuencia base de reloj de 2.8 GHz, con un total de 32 núcleos y 512 GiB de RAM. Esta máquina cuenta también con una GPU NVIDIA A100 con 80 GiB de RAM. Su disco principal se trata de un SSD CS900 de 2TB, y cuenta también con cuatro discos de almacenamiento Micron 7450 MAX de 12,8TB en una RAID0.

Esta máquina fue financiada por el Ministerio de Ciencia e Innovación a través de la Agencia Estatal de Investigación y por la Unión Europea - NextGenerationEU como parte del proyecto 'Procesamiento digital de imágenes CT mediante técnicas de HPC e IA' (TED2021 - 131091B - I00).

2.6. CUDA

El código de los programas creados está desarrollado en CUDA. CUDA es una plataforma de computación y modelo de programación en paralelo desarrollado por NVIDIA para computación en unidades de procesamiento gráficas (GPUs) [36]. A continuación se hará una breve explicación de las librerías y modelos de CUDA que se han utilizado.

2.6.1. Librería Thrust

Thrust es una biblioteca CUDA incluida en el CUDA Toolkit. Se trata de una biblioteca de algoritmos paralelos y estructuras de datos [37]. Provee una interfaz de alto nivel para la programación en GPU. Muchas de sus funciones sirven como sustituto en memoria de dispositivo de las funciones de la biblioteca estándar de C. Esto permite utilizar funciones que normalmente no se podrían acceder dentro de los kernels y utilizar operaciones aceleradas en GPU en el programa principal.

2.6.2. Memoria Pinned

Para acelerar el proceso de copia de datos a la memoria del dispositivo se hará uso de la memoria pinned [38]. La GPU no puede acceder directamente a los datos en memoria paginada del host, así que para transferir datos desde memoria paginada al dispositivo primero tiene que copiar los datos a un array en memoria pinned y transferir desde ahí, ver figura 2.10. Para evitar la copia de datos de paginada a pinned en el host se cargan directamente los datos en la memoria pinned, sin tener que pasar por paginada haciendo uso de cudaHostAlloc y cudaHostFree.

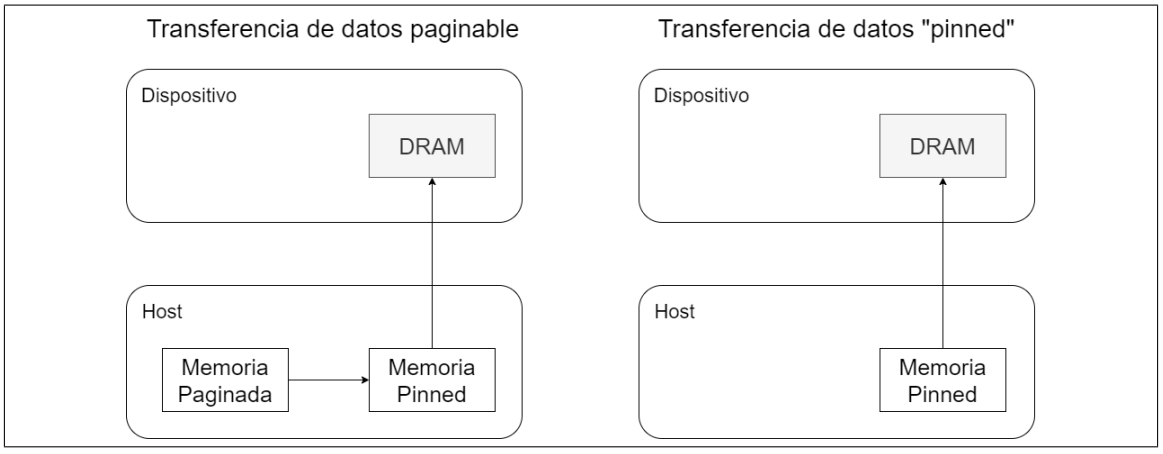

Figura 2.10: (a) Transferencia de datos paginable. (b) Transferencia de datos “pinned”.

Capítulo 3

Método de filtrado

El método de filtrado que se va a implementar en CUDA es el propuesto en [11]. Se parte de una implementación secuencial adaptada ya al formato PGM de las imágenes médicas desarrollado por [12]. A continuación se explicarán las distintas fases del filtro y se realizará un análisis de los procesos del filtro secuencial, con el objetivo de implementar los mismos procesos en CUDA.

3.1. Fases de filtrado

El filtro mixto gaussiano-impulsivo se basa en reglas fuzzy. En el proceso se evalúa individualmente cada píxel de la imagen con respecto al resto de píxeles de su ventana. En la figura 3.1 se muestra la estructura de la ventana dependiendo del píxel central seleccionado. Cada píxel se evalúa con respecto a dos métricas, su nivel de ruido impulsivo y su similitud con el resto de píxeles de la ventana, con los resultados de estas evaluaciones se realiza una media pesada del píxel con su ventana.

A continuación se verán en más detalle los procesos de evaluación del píxel y el algoritmo de reglas fuzzy que calcula el nuevo píxel.

3.1.1. Evaluación del ruido del píxel

En la primera fase, se le asigna al píxel F_i un grado de ruido impulsivo $\delta(F_i)$. Para ello se toma una ventana W centrada en F_i de tamaño n = 1,2,... y se reordena según su distancia en valor a F_i, etiquetando F_i como el primer píxel de la lista.

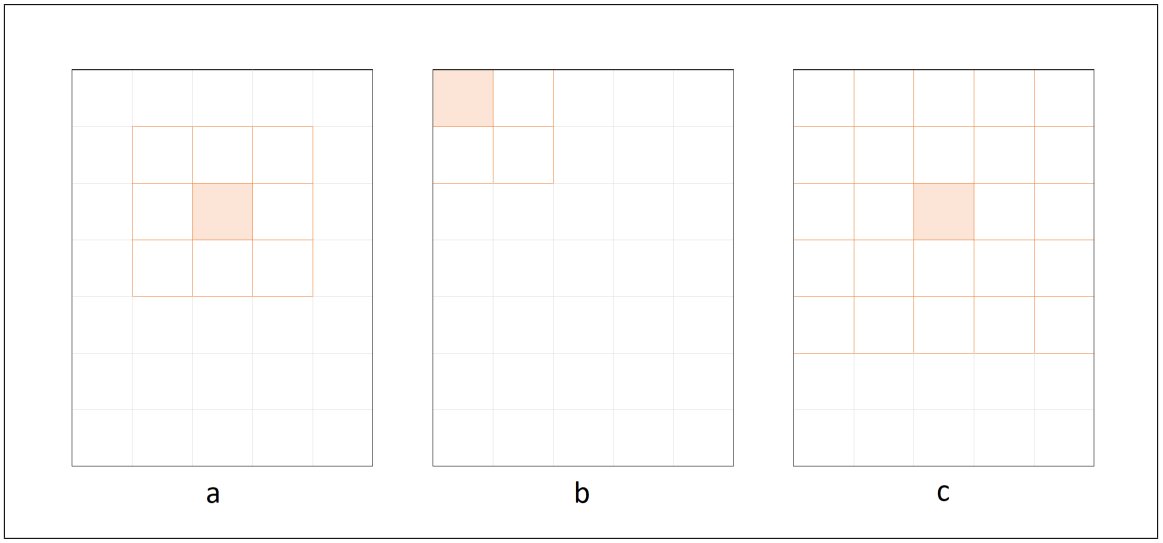

Figura 3.1: (a) Ventana de tamaño n=1. (b) Ventana de tamaño n=1 en una esquina. (c) Ventana de tamaño n=2.

En [11] esta métrica de la distancia se calcula mediante L_∞, que toma el valor absoluto máximo de la diferencia entre los píxeles en los canales RGB. En [12], al utilizarse el formato PGM esta métrica se simplifica, tomando simplemente el valor absoluto de la diferencia entre los píxeles.

$$ROD_r(F_i) = \sum_{j=0}^{r} L_\infty(F_i, F_{(j)}). \tag{3.1}$$

Para el cálculo del *ROD* (3.1) se utilizan los $r+1$ primer elementos de la lista ordenada de elementos de la ventana, siendo r un parámetro que indica el número de píxeles que computamos y que ha sido determinado experimentalmente de acuerdo con el tamaño de la ventana. En la ecuación, $F_{(j)}$ es el j-ésimo píxel en la ventana del píxel F_i. Un valor de ROD bajo indicará un grado de impulsividad bajo y por tanto similitud del píxel con sus vecinos, mientras que un valor alto indica que el píxel es muy distinto a sus vecinos y por tanto más ruidoso. Definimos el grado de impulsividad $\delta(F_i)$ mediante:

$$\delta(F_i) = \begin{cases} 0, & ROD \leq p_1, \\ \dfrac{ROD - p_1}{p_2 - p_1}, & p_1 < ROD < p_2, \\ 1, & p_2 \leq ROD. \end{cases} \tag{3.2}$$

3.1.2. Análisis de la similitud del píxel

En esta segunda fase, se analiza la similitud entre el píxel central y el resto de píxeles en la ventana. En [11] esta medida se calcula mediante L_1, que es la suma del valor absoluto de la diferencia entre los píxeles en los canales RGB, mientras que en [12], por el uso del formato PGM esta métrica será la misma que la que se toma para la distancia.

Para el cálculo de la similitud se vuelve a reordenar la ventana, pero en este caso con respecto a L_1. Se utilizan los primeros $m+1$ píxeles para calcular los grados de similitud alta, media y baja de la siguiente manera, representando x la distancia L_1 en las siguientes ecuaciones:

$$g_H(x) = \begin{cases} 1, & x \leq a, \\ \dfrac{-x}{(3a)} + \dfrac{4}{3}, & a < x < 4a, \\ 0, & 4a \leq x, \end{cases} \tag{3.3}$$

$$g_M(x) = \begin{cases} (x-a)/a, & a < x < 2a, \\ 1, & 2a \leq x \leq 3a, \\ (4a-x)/a, & 3a < x < 4a, \\ 0, & \textit{resto de casos}, \end{cases} \tag{3.4}$$

$$g_L(x) = 1 - g_H(x). \tag{3.5}$$

El parámetro a utilizado en las ecuaciones anteriores depende del nivel de ruido de la imagen 3.6.

$$a = 0{,}998\sigma + 1{,}960. \tag{3.6}$$

En la figura 3.2 se muestran los conjuntos difusos que componen la similitud en función de la distancia L_1 para un píxel.

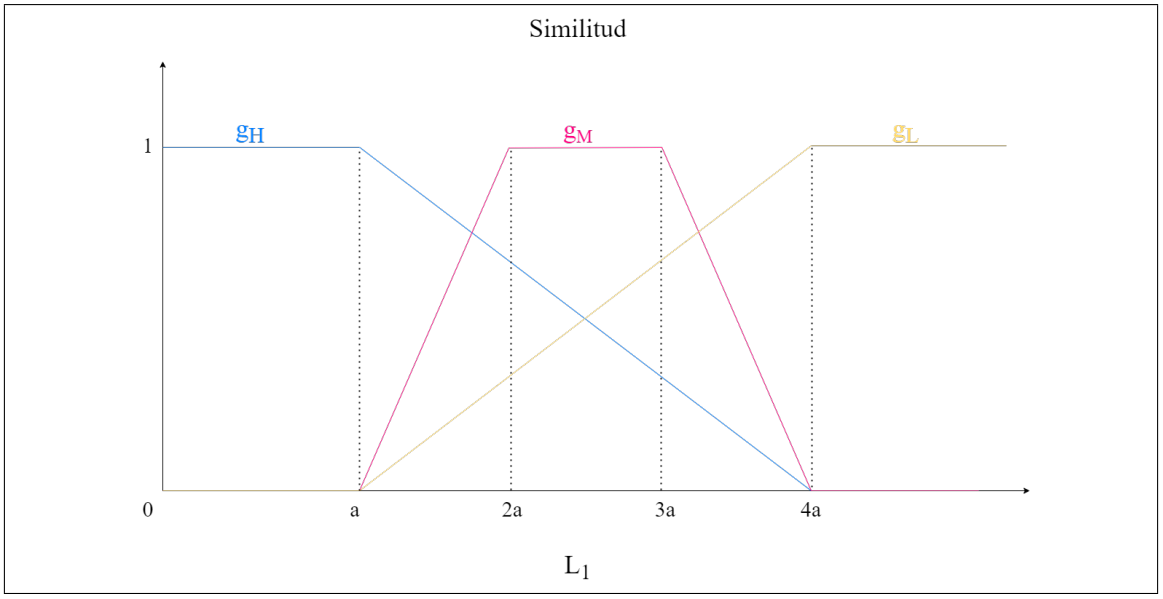

Figura 3.2: Similitud de un píxel x en función a L_1.

3.1.3. Sistema de reglas fuzzy

Se utiliza un sistema de reglas fuzzy para asignar peso a los píxeles de la ventana según las métricas calculadas.

1. IF (F^i no impulsivo AND F_0 impulsivo AND F_0 y F^i tienen similitud media) THEN w_i es un peso medio

2. IF (F^i no impulsivo AND F_0 impulsivo AND F_0 y F^i tienen similitud baja) OR (F^i no impulsivo AND F_0 no impulsivo AND F_0 y F^i tienen similitud alta) THEN w_i es un peso alto

3. IF (F^i es impulsivo)
   OR (F^i no impulsivo AND F_0 impulsivo AND F_0 y F^i tienen similitud alta)
   OR (F^i no impulsivo AND F_0 no impulsivo AND F_0 y F^i tienen similitud media)
   OR (F^i no impulsivo AND F_0 no impulsivo AND F_0 y F^i tienen similitud baja)
   THEN w_i es un peso bajo

Cada peso w_i se asocia con un grado de certeza v_L, v_M y v_S:

$$v_M(w_i) = \begin{cases} (2w_i - 1)/(2b - 1) + 1, & 1 - b < w_i \leq 0{,}5, \\ (1 - 2w_i)/(2b - 1) + 1, & 0{,}5 < w_i < b, \\ 0, & \textit{resto de casos}, \end{cases} \tag{3.7}$$

$$v_L(w_i) = \begin{cases} (w_i - 1)/(1-b) + 1, & b < w_i \leq 1, \\ 0, & resto\ de\ casos, \end{cases} \tag{3.8}$$

$$v_S(w_i) = \begin{cases} w_i/(b-1) + 1, & 0 \leq w_i \leq 1-b, \\ 0, & resto\ de\ casos. \end{cases} \tag{3.9}$$

El peso w_i de cada píxel se obtiene mediante la defuzzyficación, tal como muestra la ecuación 3.10 esto se calcula a partir de las áreas descritas por el grado de certeza del píxel x_i. En la figura 3.3 se muestran las áreas y los centros de gravedad asociados al peso, donde los trapecios de áreas están formados por los triángulos de los valores posibles del grado de certeza y los límites superiores marcados por las reglas fuzzy.

$$w_i = \frac{\sum_j A_j * C_{G_{jx}}}{\sum_j A_j}. \tag{3.10}$$

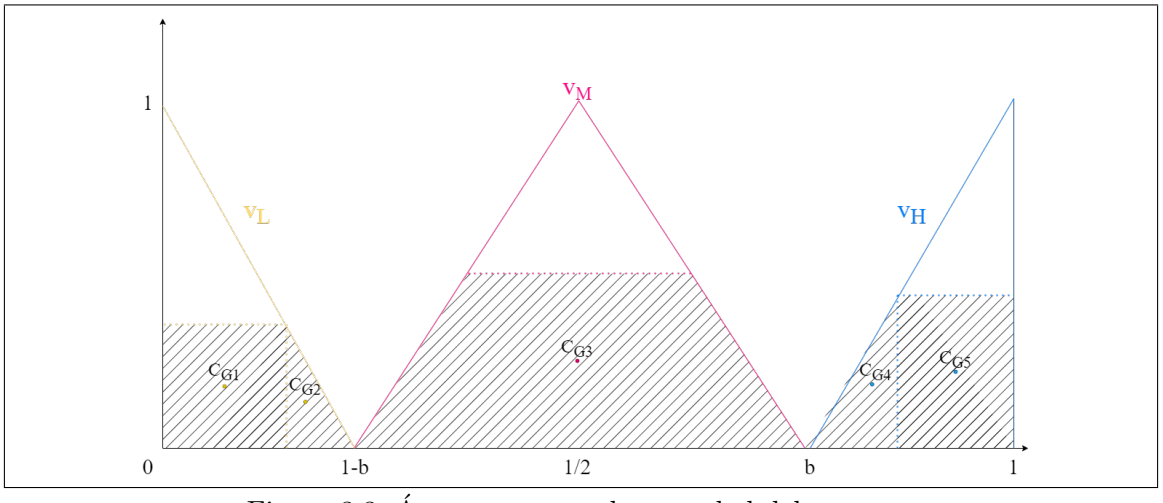

Figura 3.3: Áreas y centros de gravedad del peso w_i.

Capítulo 4

Implementación del filtro

4.1. Análisis de la versión secuencial

El programa que se utiliza como base para el desarrollo del filtro en GPU es el presentado en [12]. Está implementado en C++ y utiliza una librería de elaboración propia para la modelización de las imágenes PGM.

El código en sí está en 5 archivos, esto simplificó la implementación en OpenMP en el trabajo original.

- **sec-gaussian-impulsive.cpp:** es el programa principal, cuenta con el parser de los datos de entradas y la llamada a la función principal.
- **common-gaussian-impulsive.cpp:** contiene la implementación en funciones de las ecuaciones del método de filtrado. Así como las funciones para actualizar el píxel de la imagen y la ventana.
- **pgm.cpp:** contiene la implementación de los métodos de la clase Pgm y de la clase PgmWindow.
- **gaussian-impulsive.h:** define los métodos que se implementan en common-gaussian-impulsive.cpp.
- **pgm.h:** contiene la definición de la clase Pgm y de la clase PgmWindow.

Tras analizar el código, visto lo intrínsecamente que está desarrollado en C++, no se encontró la forma de utilizar el programa secuencial como base para la versión CUDA. Se optó en su lugar por reescribir el programa sin utilizar clases, en la siguiente sección entraremos en más detalle respecto a lo que supone este cambio.

A continuación, en el Algoritmo 1 se muestra el pseudocódigo del proceso que se pretende implementar, tal como se utilizó en la implementación secuencial.

Algoritmo 1 Filtro secuencial

INPUT: Imagen ruidosa I, parámetros n, q, r, p_1, p_2, a, b
OUTPUT: Imagen filtrada I'
1) Inicialización
$I_0 = I$
for Iteración $it = 1, \ldots$ **do**
  Imagen $I_{it} = I_{it-1}$
  for x_i pixel $\in I_{it}$ **do**
    Tomar la ventana W $n \times n$ centrada en x_i

    for $j = 1, \ldots, q$ **do**
      2) Cálculo del grado de impulsividad

      Calcular $\delta(x_i)$ usando Ec. (3.2)

      3) Cálculo del grado de semejanza

      Ordenar los píxeles $x^j \in W$ según $d(x_i, x^j)$
      Seleccionar los q píxeles mas cercanos $x^1, \ldots, x^q$
      Calcular $g_H(x_i, x^j), g_M(x_i, x^j), g_L(x_i, x^j)$, usando Ecs. (3.3), (3.4), (3.5)

      4) Cálculo de los pesos medios mediante defuzzificación

      Calcular las reglas difusas para $\{x_i, x^j\}$
      Calcular el peso w_j correspondiente a x^j mediante Ecs. (3.7), (3.8), (3.9)
    end for

    5) Cálculo del nuevo valor para x_i

    $$\hat{x}_i = \frac{\sum_{j=1}^{q} \omega_j \cdot x^j}{\sum_{j=1}^{q} \omega_j}.$$

  end for
end for

4.2. Filtro en CUDA

Realizar el filtro en CUDA suponía una serie de desafíos. Primero se debía eliminar el uso de clases del filtro, y también reestructurar el algoritmo para poder utilizar kernels.

Es importante para la paralelización en CUDA que un hilo no llame a datos obtenidos en otro hilo diferente, esto supuso que se tuviera que separar el cálculo de la impulsividad del filtrado del píxel, por suerte esto es posible ya que la impulsividad sólo cambia entre iteraciones.

La eliminación de las clases Pgm y PgmWindow supuso también que ya no fuese necesario calcular las ventanas dentro del filtro. Como la estructura de la imagen no cambia entre iteraciones tampoco cambia qué píxeles conforman la ventana de cada píxel. En la versión secuencial esta información se guardaba en los objetos de tipo PgmWindow, junto con los valores de los píxeles que la conformaban, por lo que tenía que actualizarse cada iteración. Ahora la información estructural de las ventanas de la imagen se guarda en un par de vectores, uno de tamaño de la ventana y el otro de índices, por lo que se puede calcular solamente una vez.

La versión secuencial creada con estos cambios incluidos presentaba un rendimiento mucho peor que la versión secuencial original, debido a los bucles adicionales y el manejo de vectores de gran tamaño que no podían guardarse enteros en caché entre iteraciones. Sin embargo, esta reestructuración nos permitió crear tres kernels distintos para afrontar los puntos más computacionalmente costosos del programa:

- **Kernel 1:** Cálculo de los vectores de ventana. Cada hilo computa el tamaño de la ventana de un píxel y los índices de los píxeles de su ventana respecto a la imagen total.

- **Kernel 2:** Cálculo de los grados de impulsividad. Cada hilo computa el grado de impulsividad de un píxel y reordena su ventana en función de la distancia al centro.

- **Kernel 3:** Filtrado del píxel. Cada hilo computa la semejanza, los pesos fuzzy y el nuevo valor de un píxel.

En el Algoritmo 2 se muestra el pseudocódigo de la implementación realizada en CUDA. Los kernels mencionados antes se corresponden con los pasos del algoritmo, el kernel 1 con el paso 1, el kernel 2 con el paso 2 y el kernel 3 con los pasos 3, 4 y 5.

Algoritmo 2 Filtro en CUDA

INPUT: Imagen ruidosa I, parámetros n, q, r, p_1, p_2, a, b
OUTPUT: Imagen filtrada I'
1) Inicialización
$I_0 = I$
Calcular las ventanas de la imagen
for Iteración $it = 1, \ldots$ **do**
  Imagen $I_{it} = I_{it-1}$
  2) Cálculo del grado de impulsividad
  for x_i pixel $\in I_{it}$ **do**

    Calcular $\delta(x_i)$ usando Ec. (3.2)
    Ordenar los píxeles $x^j \in W$ según $d(x_i, x^j)$

  end for
  3) Cálculo del grado de semejanza
  for x_i pixel $\in I_{it}$ **do**
    for $j = 1, \ldots, q$ **do**

      Seleccionar los q píxeles mas cercanos $x^1, \ldots, x^q$
      Calcular $g_H(x_i, x^j), g_M(x_i, x^j), g_L(x_i, x^j)$, usando Ecs. (3.3), (3.4), (3.5)

      4) Cálculo de los pesos medios mediante defuzzificación

      Calcular las reglas difusas para $\{x_i, x^j\}$
      Calcular el peso w_j correspondiente a x^j mediante Ecs. (3.7), (3.8), (3.9)
    end for

    5) Cálculo del nuevo valor para x_i

    $$\hat{x}_i = \frac{\sum_{j=1}^{q} \omega_j \cdot x^j}{\sum_{j=1}^{q} \omega_j}.$$

  end for
end for

4.2.1. Kernel 1: Índices de ventana

Siguiendo los hitos marcados por el algoritmo se tiene que el kernel 1 se encarga de parte de la inicialización. Este kernel integra parte del constructor de PgmWindow de la versión secuencial.

Los parámetros de entrada de este kernel son las dimensiones de la imagen en filas y columnas, y las dimensiones de la ventana. Se computa cuales serán los índices de los píxeles en su ventana a partir del índice del hilo que ejecuta el kernel, siempre y cuando este se encuentre en el rango de la imagen.

Sus salidas son un vector de enteros con los índices de los píxeles que componen cada ventana y otro vector de enteros que indican cuantos píxeles componen cada ventana. Ver la figura 4.1 para conocer la estructura de estos vectores.

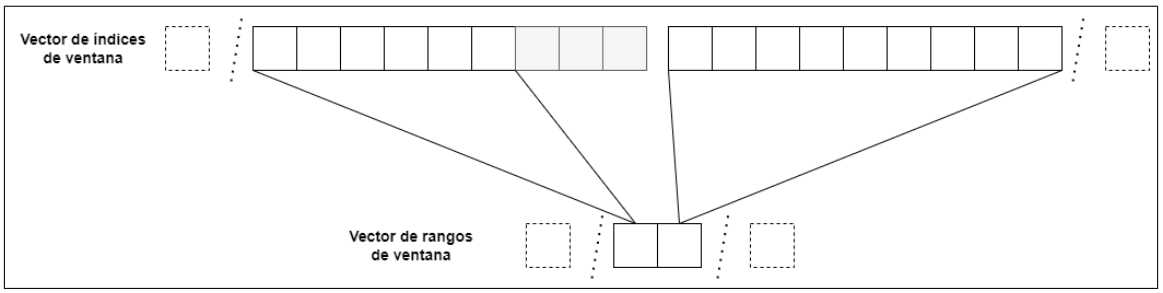

Figura 4.1: Estructura de los vectores de índices y de rangos (Ventana 3x3).

Manteniendo el stride fijo al tamaño máximo de la ventana se evitan problemas por asignación dinámica de memoria y se simplifica el acceso. Y cuando se necesite saber el tamaño de la ventana simplemente se accede al vector de rangos.

4.2.2. Kernel 2: Cálculo de la impulsividad

Era imprescindible para poder realizar la operación de filtrado paralelizada a nivel de píxel que se pudiera separar el cómputo de la impulsividad de la función del filtro fuzzy, ya que esta función no accede solamente a la impulsividad del píxel que estamos evaluando sino también a la de todos los píxeles de su ventana, y esta es una compartición de datos que no es posible en GPU. Ha sido posible separar el cálculo de la impulsividad del cómputo del filtro por el hecho de que la operación de filtrado no sobreescribe la imagen de entrada.

Este kernel toma como parámetros de entrada la imagen de entrada y los vectores de índices y rangos de ventana y como salida devuelve un vector con los grados de impulsividad de cada píxel de la imagen. En la figura 4.2 se puede observar cual es el

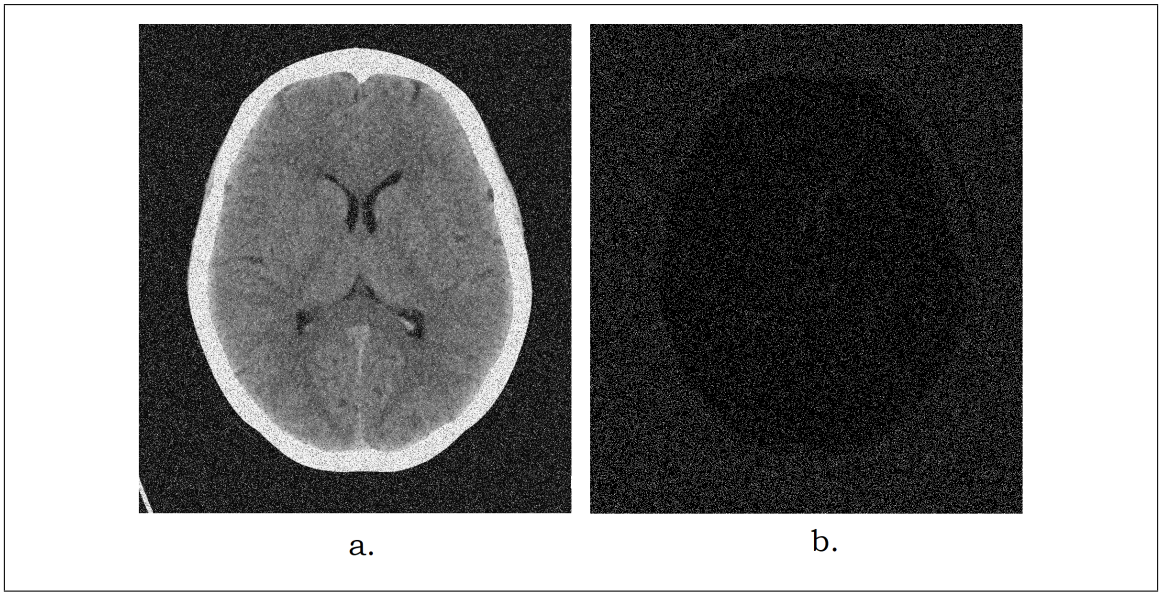

Figura 4.2: (a) Corte axial de TC con ruido gaussiano-impulsivo. (b) Impulsividad de (a).

resultado del cálculo de la impulsividad de la imagen ruidosa mapeando los valores del vector impulsividad del rango [0-1] al rango [0-255].

Se puede apreciar como el ruido en la imagen del corte axial se ve como ruido también en la imagen de impulsividad y también como el fondo de la figura es lo que tiene una densidad más exacta del nivel de ruido. En la imagen resultante se pueden intuir los contornos del cráneo y los detalles más oscuros, por el contraste que tienen con su entorno.

4.2.3. Kernel 3: Filtro fuzzy

Los parámetros de entrada de este kernel son: la imagen de entrada, la imagen original, los vectores de índices y rangos de ventana y el vector de impulsividad. La salida es la imagen filtrada.

El funcionamiento es esencialmente el mismo que en la versión secuencial, solo que para obtener los valores de los píxeles impulsivos no se llama a la función del cálculo de la impulsividad, sino que se obtienen del vector impulsividad que se calculó en el kernel anterior.

4.2.4. Implementación

Gestión de memoria

Uno de los desafíos que tiene la versión en CUDA es la necesidad de acceder a las imágenes desde la GPU. En la versión secuencial estas se almacenaban en vectores de dobles de la biblioteca estándar de C++ como parte de la clase Pgm, pero esto no es una opción en CUDA porque esta biblioteca es sólo accesible desde CPU.

Se exploraron dos opciones posibles, primero utilizando vectores de la librería thrust de CUDA y después arrays en memoria pinned.

Utilizar vectores en thrust simplificaba mucho la copia entre host y device. Es además completamente compatible con la biblioteca estándar, en la lectura de las imágenes generamos un vector std, y este se puede copiar directamente a un vector thrust en device.

```
1     std::vector<double> h_inputImage;
2     thrust::device_vector<double> d_inputImage;
3     h_inputImage = dataFromFile(pa.input_image.c_str(),false)
         ;
4     d_inputImage = h_inputImage;
```

Declarar vectores que solo se utilizan en GPU es también muy simple, el constructor se encarga de la asignación de memoria y con thrust::fill se puede inicializar el vector directamente en la GPU desde el programa principal. No se pueden utilizar los vectores directamente en los kernel, pero sí punteros a sus datos. En el siguiente extracto se puede ver cómo es el proceso de declarar, inicializar y preparar el vector impulsividad para su uso en un kernel:

```
1     thrust::device_vector<double> impulsivityDegrees(
         imageSize);
2     thrust::fill(impulsivityDegrees.begin(),
         impulsivityDegrees.end(), -1.0);
3     d_impulsivityDegrees = thrust::raw_pointer_cast(
         impulsivityDegrees.data());
```

Como técnica para la aceleración del programa se probó el uso de vectores en memoria pinned, utilizando *cudaHostAlloc* para asignar la memoria en host. En el siguiente extracto se puede ver cómo es diferente el proceso de copia de datos comparado con el ejemplo anterior en thrust:

```
1       CUDA_SAFE_CALL(cudaHostAlloc((void **)&image, imageSize*
           sizeof(double), 0));
2
3       h_original=dataFromFile(args.original_image.c_str());
4       CUDA_SAFE_CALL(cudaMalloc((void**)&d_original, (size_t)
           imageSize*sizeof(double)));
5       cudaMemcpyAsync(d_original, h_original, imageSize*sizeof(
           double), cudaMemcpyHostToDevice, 0);
```

El funcionamiento de ambas opciones es muy similar, pero se decidió desarrollar la versión final utilizando memoria pinned porque acelera la velocidad de copia entre host y device. Aun así, se sigue utilizando la librería thrust para algunas operaciones sobre los arrays.

Versión caótica

Se desarrollaron algunas variantes sobre el programa base, la primera de ellas es la que se denominó la versión caótica. Esta es una versión no determinista del filtro, en la que se modificó el tercer kernel para que en cada iteración el nuevo píxel se sobreescriba sobre la imagen de entrada en lugar de escribirse sobre una imagen de salida.

Por el paralelismo de los hilos de CUDA no es posible saber en qué orden se va a escribir sobre la imagen, por lo que distintas ejecuciones tendrán resultados distintos. Esta técnica causa también una convergencia más rápida, llegando a alcanzar el nivel máximo de calidad en menos iteraciones.

Versión sin píxel central (skip-q)

En el tercer paso del algoritmo, se itera sobre los píxeles de la ventana. Al haber reordenado la ventana en función a la distancia y contar esta con el píxel central, el filtro va a darle peso siempre al píxel previo. En esta versión del filtro se salta el primer elemento en la iteración sobre los q elementos para así evaluar el píxel solamente con respecto a sus vecinos.

La idea con este cambio es que el saltarse el píxel central se acelerará el filtro sin impactar a los resultados, ya que el tercer kernel tendrá una iteración menos que hacer y por tanto tardará menos.

Versión caótica con *skip-q*

En esta versión se combinaron los dos cambios anteriores, buscando acelerar aun más el filtro y obtener mejores resultados de calidad.

Estas son las principales versiones que se han desarrollado, pero hay más variantes sobre las mismas, se mencionan a continuación y se entrará en más detalle en el siguiente capítulo:

- **PSNR en cuadro:** Como criterio de parada en los casos de estudio se utiliza el PSNR, se ha creado una versión en la que el PSNR se calcula sólo en un cuadro de la imagen para así evitar la distorsión que pueda generar el fondo.
- **Parada por SSIM:** Otra versión que se ha desarrollado utiliza la métrica del SSIM en lugar del PSNR como criterio de parada del filtro, buscando así optimizar la calidad percibida. Para el criterio de parada mediante SSIM se probó el método en CUDA de la librería NPP [39], pero daba resultados muy distintos a la métrica en Matlab, por lo que los resultados no eran verificables. Finalmente el método que se utilizó fue el de Matlab convertido en librería C mediante la MATLAB Coder App [40].
- **Corte de impulsividad:** Se ha planeado desarrollar una versión en la que a partir de cierta iteración se deje de recalcular el vector de impulsividad.

En la figura 4.3 se puede ver un diagrama de las distintas versiones del filtro, indicando en qué punto difieren en el algoritmo. Las cuatro versiones del kernel 3 se han desarrollado para todas las versiones del criterio de parada y del corte del segundo kernel.

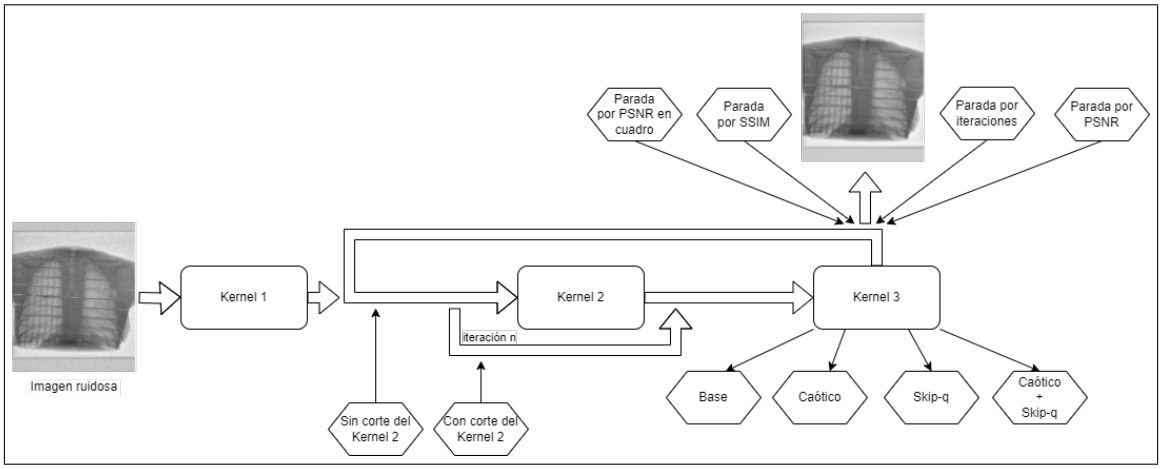

Figura 4.3: Diagrama del proceso de filtrado, con las distintas versiones del filtro implementadas.

Capítulo 5

Estudio de la calidad y rendimiento

Con el fin de determinar los parámetros óptimos de ejecución del filtro y qué versión de las implementadas es mejor se han diseñado una serie de pruebas experimentales. Todas las pruebas experimentales descritas a continuación han sido ejecutadas sobre la máquina Aurus, descrita en el Capítulo 2. Además, los datos utilizados han sido preprocesados para ser manipulados de forma correcta.

5.1. Diseño de las pruebas

Se han preparado unos conjuntos de pruebas que ejecutar en cascada. Primero hay que determinar cual es la mejor métrica de la calidad a utilizar durante la ejecución del filtro, a continuación se determinan los parámetros óptimos de ejecución en función de esa métrica y finalmente, utilizando estos parámetros se realizará un estudio del rendimiento de las distintas versiones del filtro.

Por otra parte, se realizará una serie de pruebas adicionales para estudiar los filtros sobre los distintos tipos de ruido y también otras pruebas para determinar en qué pasos del proceso de reconstrucción funciona mejor el filtro.

Para ello, se han diseñado una serie de scripts bash para poder automatizar la ejecución de las pruebas, estos scripts guardan las imágenes resultantes etiquetadas con las características que las producen y la salida en línea de comandos en un archivo CSV. A continuación se entrará en más detalle sobre qué conlleva cada prueba:

Prueba 1: Determinar la mejor métrica de calidad

Se estudiarán tres versiones de los filtros con distinto criterio de parada: PSNR, PSNR en cuadro (5 %, 25 %, 50 % y 80 % del tamaño de la imagen) y SSIM. Tras la ejecución de las pruebas se analizarán las imágenes resultantes en Matlab para determinar qué métrica obtiene mejores resultados, en el apéndice B mostramos uno de los scripts desarrollados para el análisis de calidad.

Las imágenes utilizadas para las pruebas son un subconjunto de imágenes de la tomografía denominada C012.

Prueba 2: Determinar los parámetros óptimos

Se realizará el estudio de las métricas en tres fases. Primero se estudiará la variación de los límites para el cómputo de la impulsividad, que determinan los parámetros p1 y p2 de la ecuación 3.1. A continuación se repetirán las pruebas variando el parámetro q. Y por último variando el parámetro r.

Las imágenes utilizadas para las pruebas del p1 y p2 son un subconjunto de 7 imágenes de la tomografía denominada N005, mientras que las pruebas para la q y la r se realizarán sobre un subconjunto de C012.

Prueba 3: Estudio del rendimiento

Con los parámetros obtenidos anteriormente se ejecutan las versiones paralelas del filtro para un número fijo de iteraciones. Se ejecuta también la versión secuencial del filtro sobre el mismo conjunto de imágenes, con los parámetros por defecto.

En este estudio, se utilizan los tres subconjuntos de imágenes de tomografía C012, L064 y N005, y para verificar el escalado del proceso se utilizan imágenes de mayor tamaño como son las imágenes de corte axial y de radio con ruido mixto $g = 10,\ i = 0{,}1$. Se ejecutarán también las pruebas sobre la imagen de corte axial con la versión paralelizada con OpenMP realizada en el trabajo [12], a fin de comparar el rendimiento entre las paralelizaciones en CPU y GPU.

Para estudiar el rendimiento utilizaremos el *speedup*, que definimos como:

$$S_p = \frac{T_{sec}}{T_p}, \tag{5.1}$$

donde T_{sec} y T_p se refieren al tiempo secuencial y paralelo respectivamente.

Prueba 4: Comparación de distintos tipos de ruido simulado

A partir del subconjunto en dosis completa de 7 imágenes N005, se obtienen distintas variaciones a las que les se ha introducido ruido gaussiano, ruido impulsivo, ruido mixto gaussiano impulsivo y ruido de poisson. Se realiza una evaluación inicial de la calidad tras aplicar los ruidos para poder apreciar el nivel de mejora tras el filtrado. En el apéndice C se muestra el script que genera las imágenes ruidosas y la función creada para añadir ruido de poisson.

También, se estudia la calidad obtenida por las distintas versiones del filtro sobre este conjunto de imágenes, comparando los resultados para los distintos tipos de ruido.

Prueba 5: Filtrado en distintas fases de la reconstrucción

Como se muestra en la figura 2.2 el filtrado puede realizarse en distintas fases del proceso de reconstrucción. Por ello se analiza para un conjunto de sinogramas una comparación de las imágenes reconstruidas en tres casos: aplicando el filtro sobre el sinograma, aplicando el filtro sobre la imagen reconstruida y aplicando el filtro sobre el sinograma y la imagen reconstruida.

5.2. Prueba 1: Determinar la mejor métrica de calidad

Como se describió en la sección anterior, se ejecutaron las distintas versiones del proceso del tercer kernel empleando los diversos criterios de parada y sobre el conjunto completo de imágenes de tomografía.

Los resultados de las métricas de calidad presentados a continuación han sido calculados mediante el uso de Matlab a partir de las imágenes producidas tras el filtrado.

En las figuras 5.1 y 5.2 se presentan los resultados del PSNR y SSIM, respectivamente, para el subconjunto C012 con distintos criterios de parada para las distintas versiones del filtro. La única diferencia que se aprecia en los resultados es entre las versiones del filtro para los valores de SSIM, para los valores de PSNR esta diferencia no es apreciable.

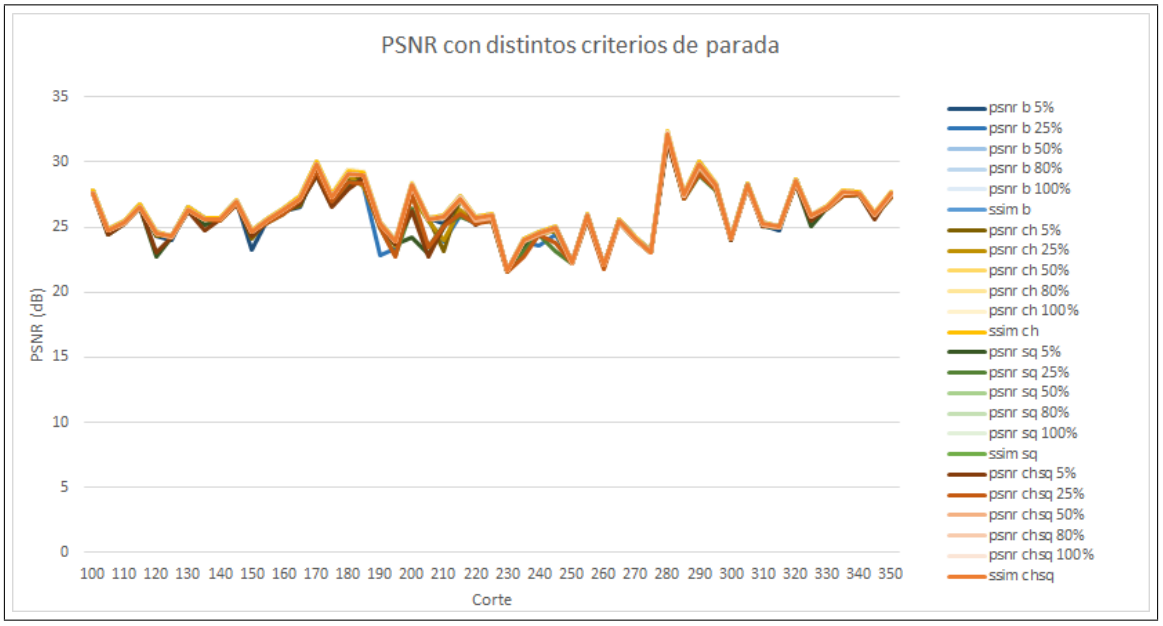

Figura 5.1: Gráfico del PSNR de las imágenes obtenidas con distintos criterios de parada, para cada versión del filtro.

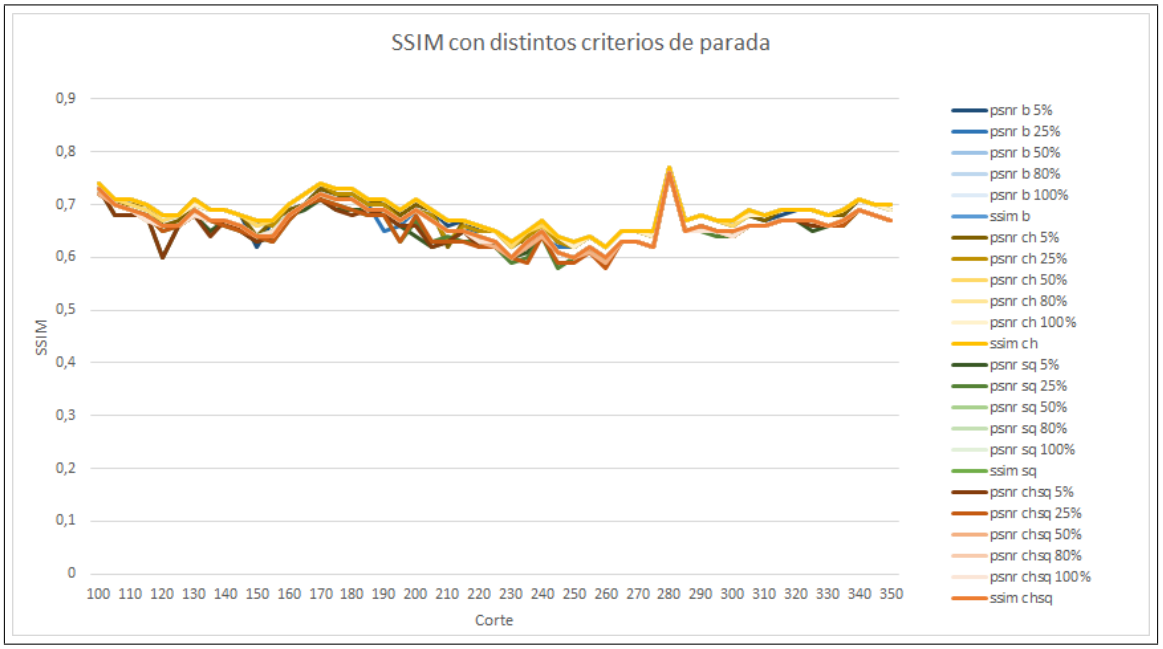

Figura 5.2: Gráfico del SSIM de las imágenes obtenidas con distintos criterios de parada, para cada versión del filtro.

	Versión Base					
Corte	**PSNR 5 % (PSNR)**	**PSNR 25 %(PSNR)**	**PSNR 50 % (PSNR)**	**PSNR 80 % (PSNR)**	**PSNR 100 % (PSNR)**	**SSIM (PSNR)**
150	23,26	24,71	24,73	24,73	24,73	24,72
155	25,67	25,66	25,74	25,74	25,74	25,74
160	26,19	26,29	26,44	26,44	26,44	26,43
165	26,55	27,3	27,42	27,42	27,42	27,41
170	29,66	29,86	30,08	30,08	30,08	30,07
175	27,27	27,14	27,53	27,53	27,53	27,52
180	28,57	29,01	29,38	29,38	29,38	29,33
185	29,02	28,18	29,24	29,25	29,26	29,19
190	25,26	22,88	25,39	25,39	25,4	25,38
195	23,74	23,37	23,94	23,96	23,96	23,96

Tabla 5.1: Resultados del PSNR final con distintos criterios de parada para 10 cortes del conjunto C012.

	Versión Base					
Corte	**PSNR 5 % (SSIM)**	**PSNR 25 % (SSIM)**	**PSNR 50 % (SSIM)**	**PSNR 80 % (SSIM)**	**PSNR 100 % (SSIM)**	**SSIM (SSIM)**
150	0,62	0,67	0,66	0,67	0,67	0,67
155	0,67	0,66	0,67	0,67	0,67	0,67
160	0,69	0,7	0,7	0,7	0,7	0,7
165	0,7	0,72	0,72	0,72	0,72	0,72
170	0,73	0,74	0,74	0,74	0,74	0,74
175	0,72	0,72	0,73	0,73	0,73	0,73
180	0,71	0,72	0,73	0,73	0,73	0,73
185	0,71	0,7	0,71	0,71	0,71	0,71
190	0,7	0,65	0,71	0,71	0,71	0,71
195	0,68	0,66	0,69	0,69	0,69	0,69

Tabla 5.2: Resultados del SSIM final con distintos criterios de parada para 10 cortes del conjunto C012.

En la tabla 5.1 se presentan los resultado de PSNR de las imágenes resultantes tras utilizar los distintos criterios de parada (PSNR 5 %, PSNR 25 %, PSNR 50 %, PSNR 80 %, PSNR 100 % y SSIM) en la versión base del filtro. En la tabla 5.2 se presentan los resultados de SSIM en estas mismas imágenes. Como se adelantaba en las figuras mostradas anteriormente, no existe una diferencia significativa en los resultados del uso de los distintos criterios de parada, salvo el criterio del PSNR en cuadro que toma solo el 5 % de la imagen en consideración, que obtiene resultados peores.

Es significativo en estos resultados que los valores alcanzados tanto para el PSNR como el SSIM indican que la calidad de la imagen resultante es mala. En la figura 5.3 se muestra el corte 170 en dosis completa, en baja dosis y filtrado, hemos elegido este corte por tener los mejores resultados en las tablas anteriores, pero se puede ver que el ruido en este conjunto es tan significativo que el filtro no lo elimina.

5.3. Prueba 2: Determinar los parámetros óptimos

En una aplicación real del filtro no existe una imagen de referencia con la que poder comparar los resultados, por lo que resulta esencial determinar los parámetros óptimos del filtro para producir los mejores resultados posibles. Los parámetros a

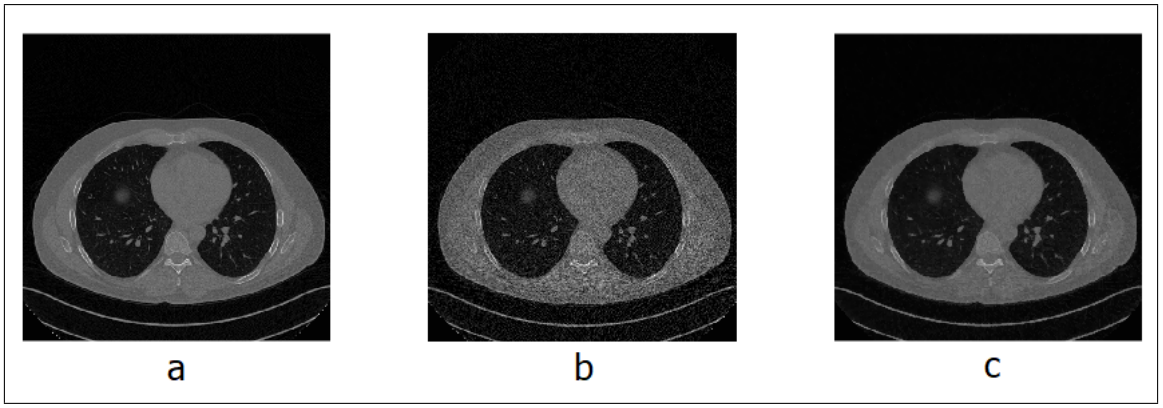

Figura 5.3: (a) Corte 170 del conjunto C012 a dosis completa. (b) A baja dosis. (c) Tras el filtrado (Versión base, corte por PSNR 100 %).

analizar son los siguientes:

- **Iteraciones:** al realizar los estudios de parámetros de calidad se obtendrá una aproximación a la iteración en la que el filtro deja de producir mejoras. Conocer este parámetro permitirá en un contexto real parar antes de que la mejora sea suficiente y evitar la degradación de la imagen por filtrado excesivo.
- **Parámetros p1 y p2:** se busca ajustar los límites del cálculo del ROD para mejorar la sensibilidad de la detección de la impulsividad.
- **Parámetro q:** se busca obtener la q óptima para cada versión del filtro. Inicialmente este valor se considera $q = 7$ para ventanas 3x3, pero este parámetro proviene de los estudios realizados sobre el algoritmo original de imágenes a color.
- **Parámetro r:** similar al parámetro q, este parámetro tiene un valor óptimo determinado en $r = 3$ para la ventana de 3x3 por los estudios del algoritmo original, pero es importante ajustarlo a las versiones desarrolladas del filtro.

Utilizando las pruebas realizadas en la sección anterior, se puede determinar el número de iteraciones que necesitan cada una de las versiones del proceso de filtrado para cada subconjunto de imágenes, con el criterio de corte por PSNR en imagen completa.

En la figura 5.4 se muestran las iteraciones que necesita cada método de filtrado para obtener el resultado óptimo para las imágenes del subconjunto C012, en la figura 5.5 se muestra lo mismo pero para el subconjunto L064. No solo hay una gran diferencia entre las iteraciones que necesitan los métodos, pero también entre las iteraciones que necesita cada corte del subconjunto y entre subconjuntos con niveles de ruido diferentes. No existe un número óptimo de iteraciones para cada método cuando

el nivel de ruido es alto, para ruido bajo las iteraciones antes de que el filtro cause degradación están en torno a $ite = 3$.

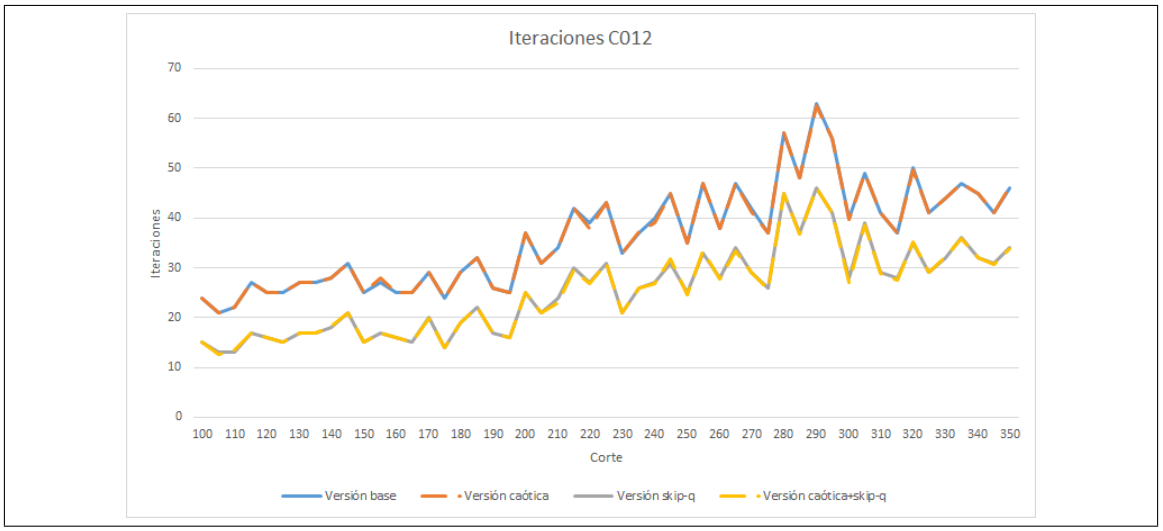

Figura 5.4: Gráfica de iteraciones en el subconjunto C012.

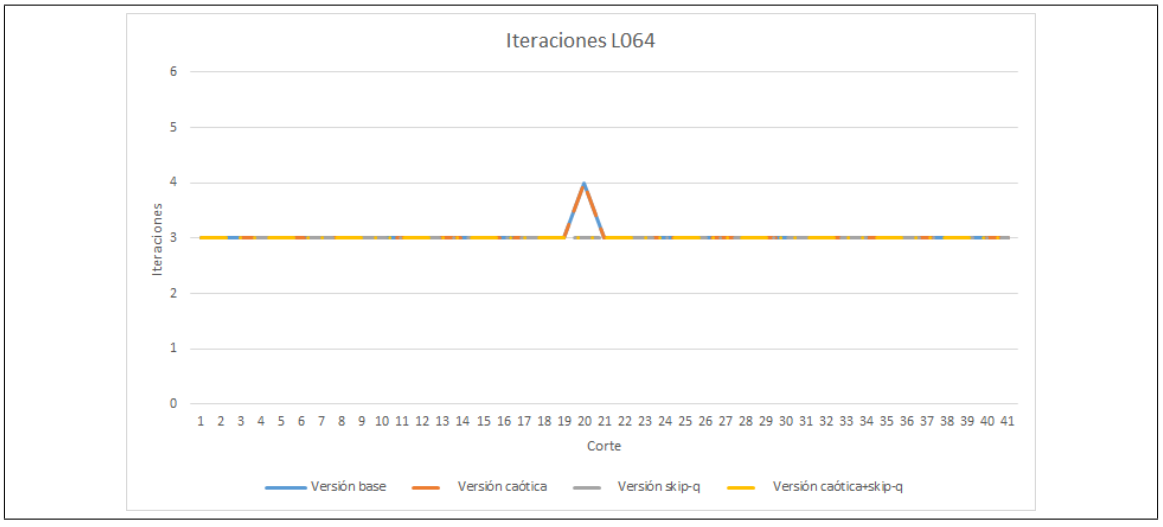

Figura 5.5: Gráfica de iteraciones en el subconjunto L064.

Como fue descrito en la sección de diseño (5.1), primero se ejecutaron las pruebas para determinar p1 y p2. Se empleó un conjunto de imágenes pequeño y con poco ruido para estudiar los parámetros ya que se quiere mejorar la sensibilidad de detección de ruido.

En la figura 5.6 se muestran los PSNR alcanzados para los cortes de N005 con los diferentes valores de p1 y p2 en la versión base del filtro, en todos los cortes los mejores valores de PSNR los alcanzan los valores de $p1 = 90$ y $p2 = 150$. Ya que el Kernel 2 es el mismo en todas las versiones, se realizó el estudio únicamente en la versión base.

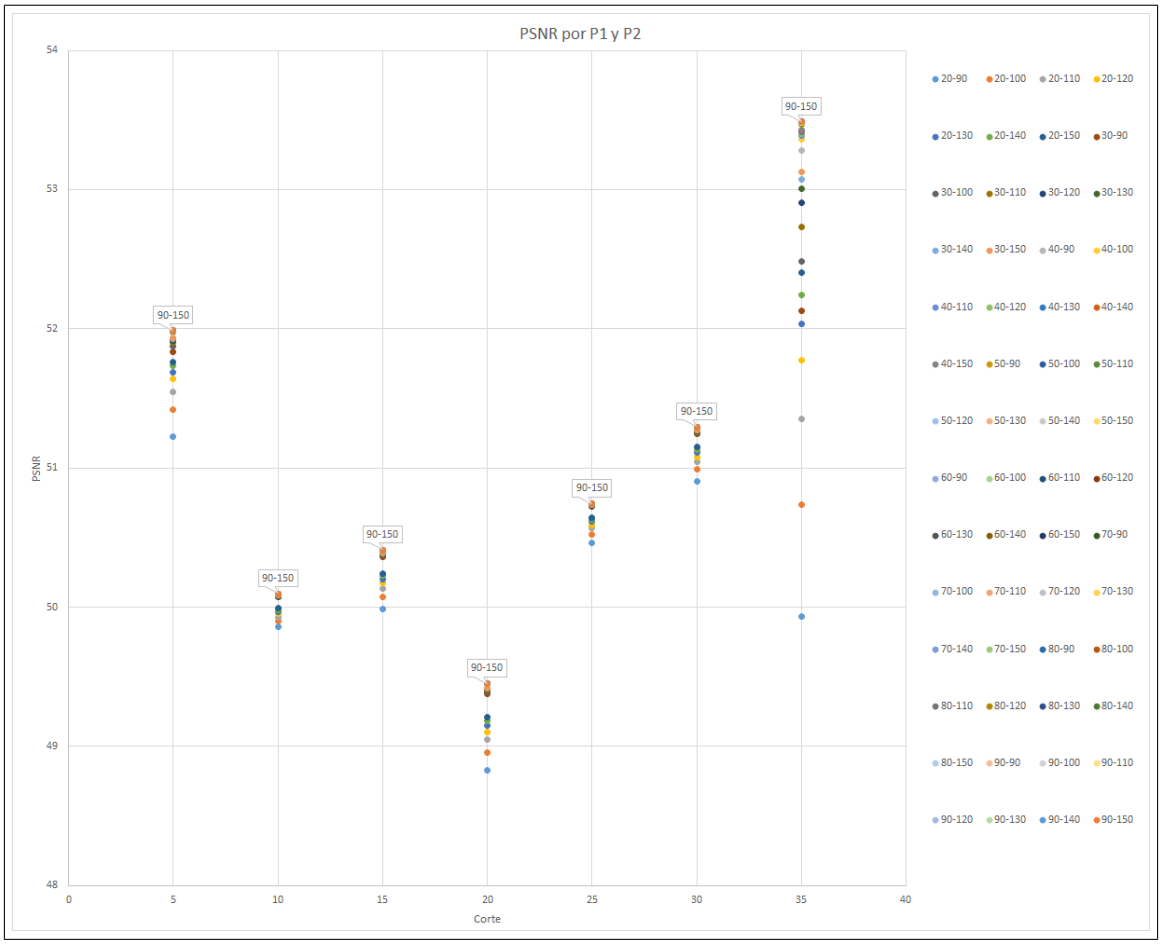

Figura 5.6: Gráfica de PSNR por p1-p2 en el subconjunto N005.

	Corte	255	260	265	270	275	280	285	290	295	300
Versión base	q=7	25,9891	22,1458	25,6249	24,1947	23,1656	32,4089	27,5932	30,0854	28,4166	24,2089
	q=8	26,0035	22,1554	25,6373	24,2174	23,1842	32,5504	27,6035	30,0862	28,4331	24,2181
	q=9	25,8763	22,127	25,542	24,1751	23,1486	32,2065	27,4206	29,6673	28,2009	24,1725
Versión caótica	q=7	25,98962	22,14583	25,62511	24,19542	23,16586	32,41055	27,59398	30,08554	28,4171	24,20959
	q=8	26,0036	22,15518	25,63638	24,21737	23,18421	32,54962	27,60278	30,08495	28,43332	24,218
	q=9	25,87557	22,12665	25,54106	24,17432	23,14763	32,20631	27,41929	29,66565	28,19989	24,17153
Versión skip-q	q=7	25,9041	22,1031	25,512	24,1285	23,1044	32,2521	27,4858	29,8776	28,261	24,1351
	q=8	25,8957	22,0974	25,5124	24,1331	23,0969	32,3578	27,4513	29,8503	28,2705	24,127
	q=9	25,726	22,0469	25,3624	24,054	23,0297	31,9759	27,205	29,4021	27,967	24,0428
Versión caótica + skip-q	q=7	25,90234	22,10288	25,51115	24,12961	23,10211	32,25246	27,48565	29,87447	28,26106	24,13521
	q=8	25,89493	22,09644	25,50773	24,13399	23,09676	32,35489	27,45191	29,85006	28,26704	24,12505
	q=9	25,7237	22,04518	25,36142	24,05116	23,02677	31,97171	27,20129	29,40267	27,96358	24,0407

Tabla 5.3: Resultados del PSNR para cada versión del filtro con distintos valores de q.

	Corte	255	260	265	270	275	280	285	290	295	300
Versión base	r=2	26,0066	22,1542	25,6308	24,2068	23,1708	32,4054	27,6008	30,0988	28,4338	24,2151
	r=3	25,9891	22,1458	25,6249	24,1947	23,1656	32,4089	27,5932	30,0854	28,4166	24,2089
	r=4	25,9226	22,1146	25,5493	24,1425	23,1286	32,3082	27,5111	29,9488	28,293	24,1687
	r=5	25,8192	22,0734	25,4055	24,0464	23,0621	32,1112	27,3821	29,6835	28,0615	24,09
Versión caótica	r=2	26,00641	22,15386	25,63177	24,20797	23,17099	32,40581	27,60129	30,09927	28,43372	24,21615
	r=3	25,99003	22,14593	25,62508	24,19545	23,16586	32,41061	27,59388	30,08549	28,41724	24,20965
	r=4	25,92337	22,11449	25,54968	24,14354	23,12903	32,30932	27,51223	29,94967	28,29361	24,16958
	r=5	25,81876	22,07296	25,4071	24,04809	23,0632	32,11355	27,38275	29,68591	28,06259	24,09005
Versión skip-q	r=2	25,9062	22,1035	25,5125	24,1287	23,105	32,2525	27,4871	29,8817	28,2642	24,1346
	r=3	25,9041	22,1031	25,512	24,1285	23,1044	32,2521	27,4858	29,8776	28,261	24,1351
	r=4	25,8859	22,0957	25,491	24,1189	23,0943	32,2337	27,4576	29,8234	28,2071	24,1235
	r=5	25,7949	22,0664	25,3843	24,0208	23,0477	32,0802	27,3416	29,6396	28,0362	24,0728
Versión caótica + skip-q	r=2	25,90554	22,1034	25,51179	24,12937	23,10294	32,25188	27,48773	29,8783	28,26535	24,13492
	r=3	25,90242	22,10258	25,51204	24,12956	23,10277	32,25332	27,48664	29,87704	28,26225	24,13581
	r=4	25,88664	22,09436	25,49037	24,12081	23,09266	32,23351	27,45737	29,82544	28,20685	24,12361
	r=5	25,79552	22,06685	25,38373	24,02388	23,04674	32,07585	27,34413	29,63816	28,03613	24,07366

Tabla 5.4: Resultados del PSNR para cada versión del filtro con distintos valores de r.

Las pruebas para los parámetros q y r se han lanzado sobre un subconjunto de C012 de los cortes 150 a 300. El parámetro q se ha medido en el rango [7-9] y el parámetro r en el rango [2-5]. Se lanzaron primero las pruebas de q, por lo que la r se calculó ya con la q óptima.

En la tabla 5.3 se muestran los resultados del PSNR para las distintas q en cada versión del filtro para 10 de los cortes filtrados. En la tabla 5.4 se muestran estos mismos valores para los distintos valores de r. Hemos resaltado en amarillo para cada corte cual es el valor mayor de PSNR, para poder ver más fácilmente qué valor es tendencia.

Con estos resultados se puede determinar que los parámetros óptimos de q y de r para cada versión del filtro son:

1. Versión base: $q = 8,\ r = 2$
2. Versión caótica: $q = 8,\ r = 2$
3. Versión *skip-q*: $q = 7/8,\ r = 2$
4. Versión caótica+skip-q: $q = 7,\ r = 2/3$

5.4. Prueba 3: Estudio del rendimiento

Se han realizado estudios del speedup sobre los conjuntos de imágenes C012, L064 y N005, y estudios de escalado sobre las imágenes de corte axial y de radio. Para calcular el speedup, y que la medida de este sea lo más adecuada posible, se ha calculado el tiempo por iteración sin contar los tiempos de carga de imagen para la versión secuencial y las distintas versiones en CUDA fijando las iteraciones del filtro en 60 para todos los casos.

En la figura 5.7 se muestra el speedup respecto a la versión secuencial en el conjunto C012, en la figura 5.8 se muestra el speedup en el conjunto L064 y en la figura 5.9 para el conjunto N005.

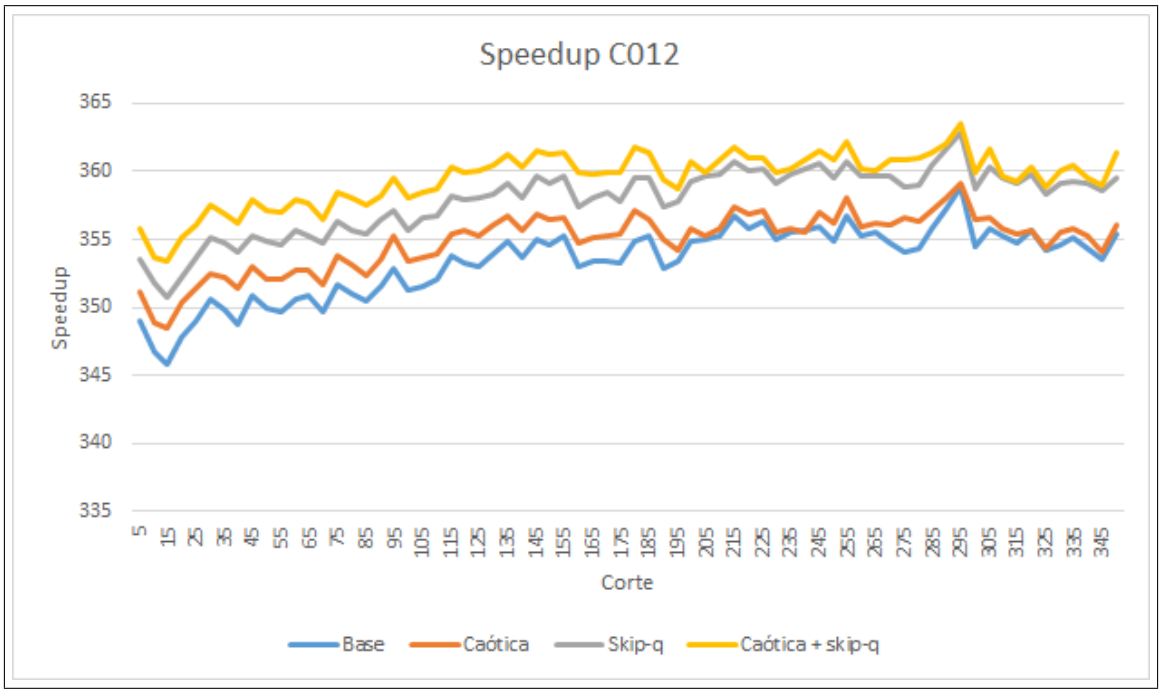

Figura 5.7: Gráfica del speedup en el subconjunto C012.

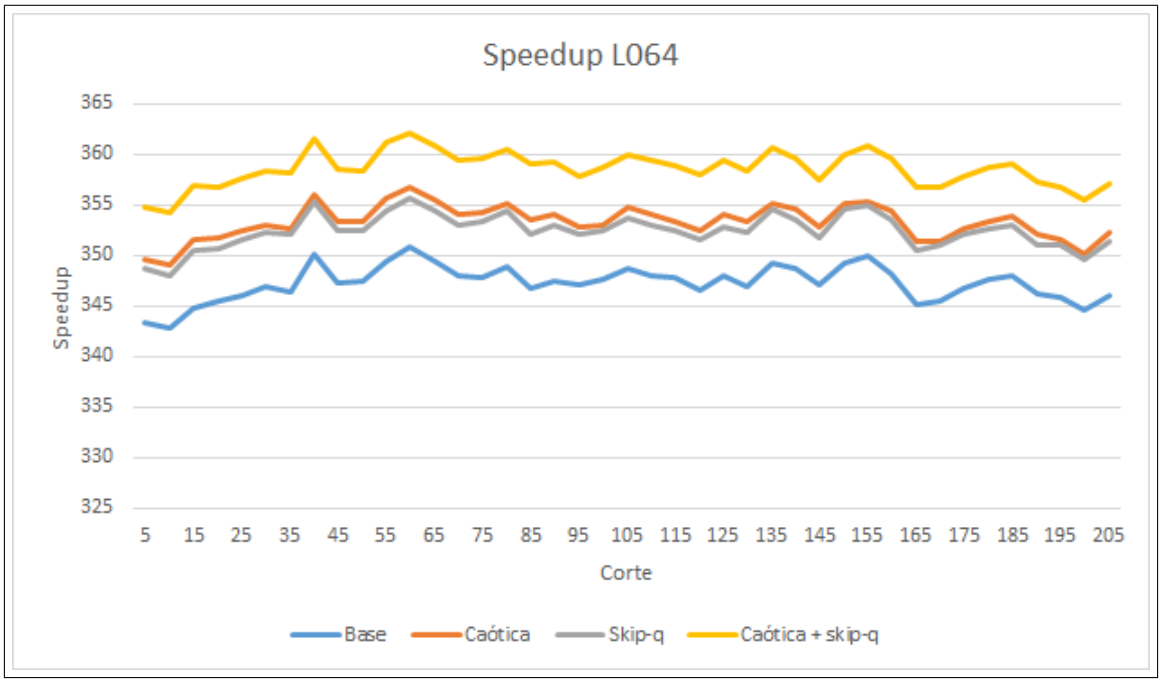

Figura 5.8: Gráfica del speedup en el subconjunto L064.

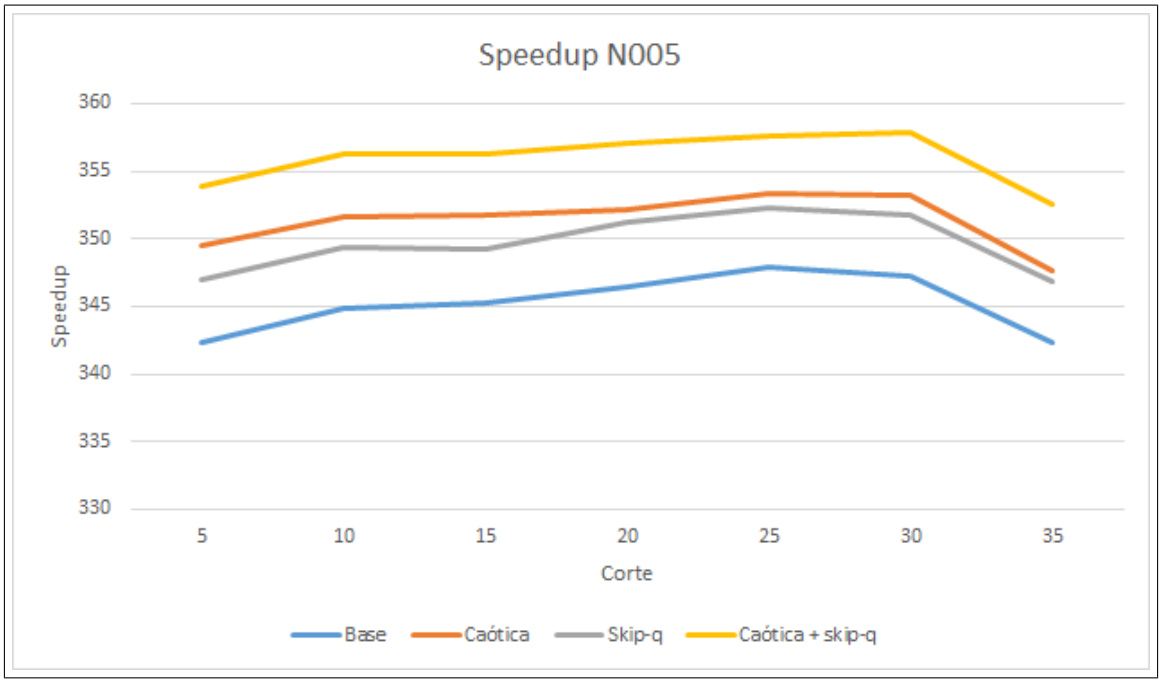

Figura 5.9: Gráfica del speedup en el subconjunto N005.

Las tres gráficas muestran claramente que el método que obtiene un mayor speedup se trata de la versión caótica del filtro con *skip-q*, mientras que el que obtiene una menor aceleración es la versión base.

Esto es consistente con los resultados esperados, pues las versiones caóticas obtienen aceleración al no tener que realizar la copia al vector de salida entre iteraciones y las versiones con *skip-q* tienen una iteración menos en el Kernel de filtrado. Como es natural, la combinación de estas dos modificaciones producen el método más rápido.

Con respecto a cual de las dos versiones obtiene una mayor aceleración, la respuesta es más incierta. En el conjunto C012 es el *skip-q* no-caótico el que obtiene mayor aceleración comparado con el caótico, pero en L064 y N005 esto es al revés. Podemos decir que con mayor nivel de ruido (C012) es más significativo en la velocidad el hacer una iteración menos en el bucle del filtrado.

En la tabla 5.5 se pueden ver los resultados del estudio del speedup sobre la imagen de Radio y la imagen de Corte Axial. Como en los estudios del speedup anteriores, hemos fijado las iteraciones en 60 y el speedup se ha calculado tomando los tiempos solo del método de filtrado divididos por el número de iteraciones, sin tener en cuenta los tiempos de carga y guardado de las imágenes.

Para las imágenes grandes el speedup es significativamente mayor que para los conjuntos anteriores, llegando hasta más de 550 para la imagen de Radio. Este incremento en el speedup se debe al tamaño de las imágenes, puesto que el método secuencial, al ser iterativo, escala mucho peor que el paralelo.

Verificamos con estas dos imágenes también que la versión más rápida que tenemos es la de *skip-q* caótica. Respecto a la ambigüedad de cual es más rápido entre la versión caótica base y la versión con *skip-q* no caótico, en ambas imágenes la versión más veloz es la caótica.

Puesto que ya existía una implementación de este filtro con paralelismo en memoria

	Radio				
	Secuencial	Base	Caótico	Skip-q	Caótico + Skip-q
Tiempo total (s)	318,479	0,6032186	0,5852211	0,5935888	0,5771309
Tiempo/Iteración	5,307983333	0,010053643	0,009753685	0,009893147	0,009618848
Speedup		**527,9661469**	**544,2028662**	**536,5313496**	**551,8314823**
	Axial				
	Secuencial	Base	Caótico	Skip-q	Caótico + Skip-q
Tiempo total (s)	25,8099	0,05459632	0,0531273	0,0542725	0,05228788
Tiempo/Iteración	0,430165	0,000909939	0,000885455	0,000904542	0,000871465
Speedup		**472,7406536**	**485,8123789**	**475,5612879**	**493,6115214**

Tabla 5.5: Resultados del Speedup para la imagen de Radio y la imagen de Corte Axial.

	Axial					
	Secuencial	2 hilos	4 hilos	8 hilos	16 hilos	32 hilos
Tiempo total	25,8099	12,8568	6,52189	3,47494	1,73225	0,903664
Tiempo/Iteración	0,430165	0,21428	0,108698167	0,057915667	0,028870833	0,015061067
Speedup		**2,0074902**	**3,957426439**	**7,427437553**	**14,89963937**	**28,56138944**

Tabla 5.6: Resultados del Speedup para la imagen de Corte Axial en OpenMP.

compartida [12], los autores nos han cedido el código con el fin de realizar la comparativa de los tiempos obtenidos en ambos casos. Se ha ejecutado el filtro en su versión con OpenMP en la misma máquina que la versión en CUDA (Aurus), sobre la imagen de corte axial, aumentando el número de hilos en potencias de dos hasta alcanzar el número de núcleos de la m´aquina. En la tabla 5.6 mostramos los resultados de tiempos y el speedup de esta versión, este speedup está cercano al teórico para este tipo de paralelización que sería igual al número de núcleos. Nos permite verificar que la versión en GPU es mucho más rápida, unas 17 veces más que la mejor de OpenMP para esta imagen, y cabe de esperar que para imágenes mayores esta diferencia aumente aun más, no porque la versión en CPU pierda prestaciones, pero porque la versión para GPU aumenta su aceleración con el tamaño.

5.5. Prueba 4: Comparación de distintos tipos de ruido simulado

En esta sección analizaremos cómo se comportan las versiones del filtro para distintos tipos de ruido. Se ha introducido ruido artificial mediante el uso de Matlab en las imágenes del conjunto N005, en la tabla 5.7 se pueden ver las métricas de calidad de estas imágenes ruidosas con respecto a las imágenes sin ruido, con esto se puede ver de qué calidad de imagen se parte. Se puede ver que el ruido que menos impacto causa es el ruido de poisson, ya que no modifica el fondo, mientras que el que más impacto causa en las métricas es el gaussiano-impulsivo, especialmente si nos fijamos en el SSIM. En la figura 5.10 se puede ver los resultados de introducir ruido sobre el corte 15 de N005.

Se han filtrado las imágenes con los parámetros óptimos determinados anteriormente y se ha analizado el ruido tras el filtrado con Matlab. Para poder determinar qué versión del método es la más eficiente, se han generado gráficas con los valores de PSNR y SSIM de las imágenes filtradas para cada tipo de ruido. Se han seleccionado las gráficas que muestran diferencias entre los métodos, aquellas en las que todos los métodos producen resultados idénticos se han obviado.

En la figura 5.11 se muestra el PSNR para las imágenes filtradas de ruido gaussiano. El método que genera mejores resultados queda ambiguo ya que la tendencia cambia en el tercer corte, esto puede deberse a que los cortes interiores son menos uniformes que los de los extremos, ya que muestran las estructuras de los ojos.

En la figura 5.12 se muestra el PSNR para el ruido impulsivo, es con diferencia el tipo de ruido que mejor se filtra. Los métodos de filtrado más eficientes son el base y el caótico.

En la figura 5.13 se muestra el SSIM para el ruido de poisson y también los métodos de filtrado más eficientes para este ruido son el base y el caótico.

Por último, en la figura 5.14 se muestra el PSNR para el ruido mixto y a diferencia de los dos anteriores, se obtienen mejores resultados con las versiones con *skip-q*.

El ruido que peor se ha filtrado ha sido el gaussiano, si nos fijamos en las imágenes producidas al incluir este ruido podemos ver que el introducir ruido gaussiano provoca una mayor distorsión de los fondos de la imagen, lo que puede sesgar los resultados. Una técnica que se puede utilizar para evitar este sesgo y dar métricas de calidad más adecuadas es el tomar la media de las métricas en determinadas regiones de interés (ROIs). Por la complejidad que supondría determinar estos ROIs para cada corte no se han utilizado en este estudio, pero sí se considerará para trabajos futuros.

Independientemente del comportamiento con cada ruido se puede apreciar claramente, basado en los resultados obtenidos, que la única diferencia en el filtrado entre los filtros es entre la versión base y la versión *skip-q*. Tanto la versión base como la versión caótica obtienen resultados muy similares, y tanto la versión con *skip-q* como la versión caótica con *skip-q* obtienen resultados similares también.

Esto da la libertad de utilizar tan sólo las versiones caóticas de los métodos base y con *skip-q*, ya que como se vio en la sección anterior son los más rápidos.

	PSNR				MSE				SSIM			
Corte	Gau.	Imp.	Poi.	Gau.-Imp.	Gau.	Imp.	Poi.	Gau.-Imp.	Gau.	Imp.	Poi.	Gau.-Imp.
5	17,09	16,93	24,66	14,12	1271,25	1317,68	222,48	2520,85	0,10	0,25	0,71	0,06
10	17,11	16,95	25,03	14,13	1263,79	1311,77	204,43	2513,87	0,12	0,27	0,70	0,08
15	17,12	17,17	23,08	14,24	1262,80	1247,55	319,87	2448,11	0,11	0,26	0,67	0,07
20	17,10	17,17	22,48	14,20	1266,52	1248,86	367,08	2474,61	0,12	0,27	0,66	0,08
25	17,12	16,94	22,14	14,15	1262,83	1314,20	396,94	2500,27	0,12	0,26	0,69	0,08
30	17,14	16,77	24,15	14,04	1257,59	1368,96	250,05	2566,92	0,10	0,26	0,72	0,06
35	17,16	16,42	24,58	13,88	1251,79	1483,58	226,67	2659,35	0,07	0,25	0,81	0,04

Tabla 5.7: PSNR, MSE y SSIM del conjunto N005 con ruido simulado.

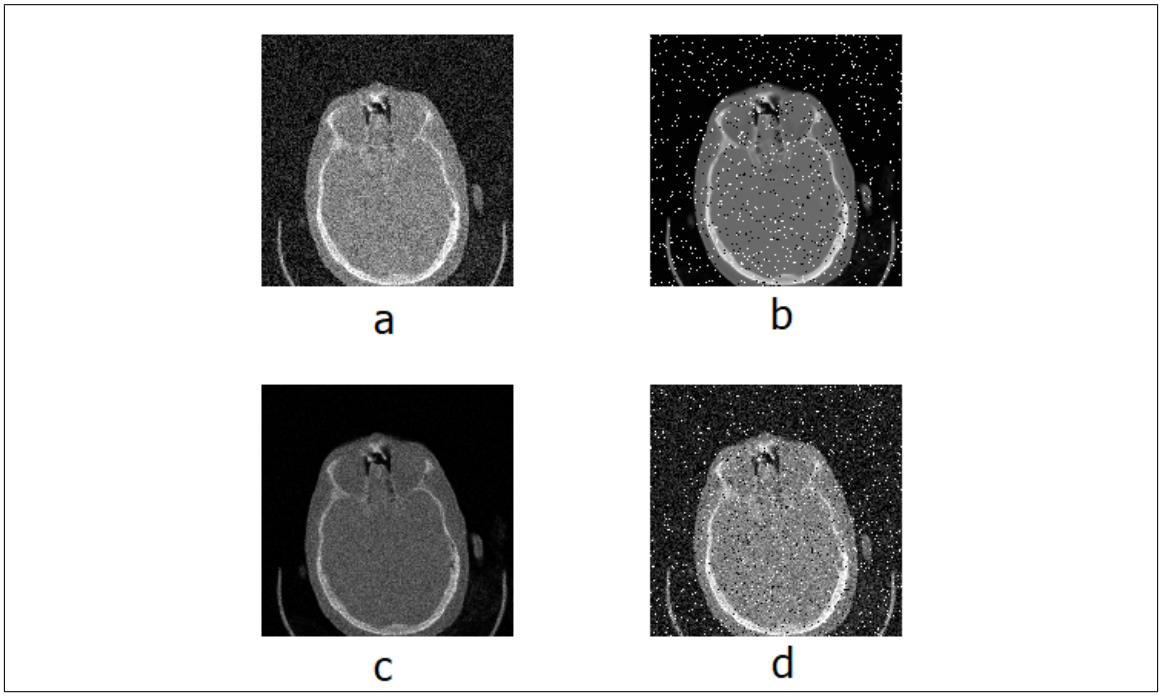

Figura 5.10: (a) Ruido gaussiano. (b) Ruido impulsivo. (c) Ruido de poisson. (d) Ruido mixto gaussiano-impulsivo.

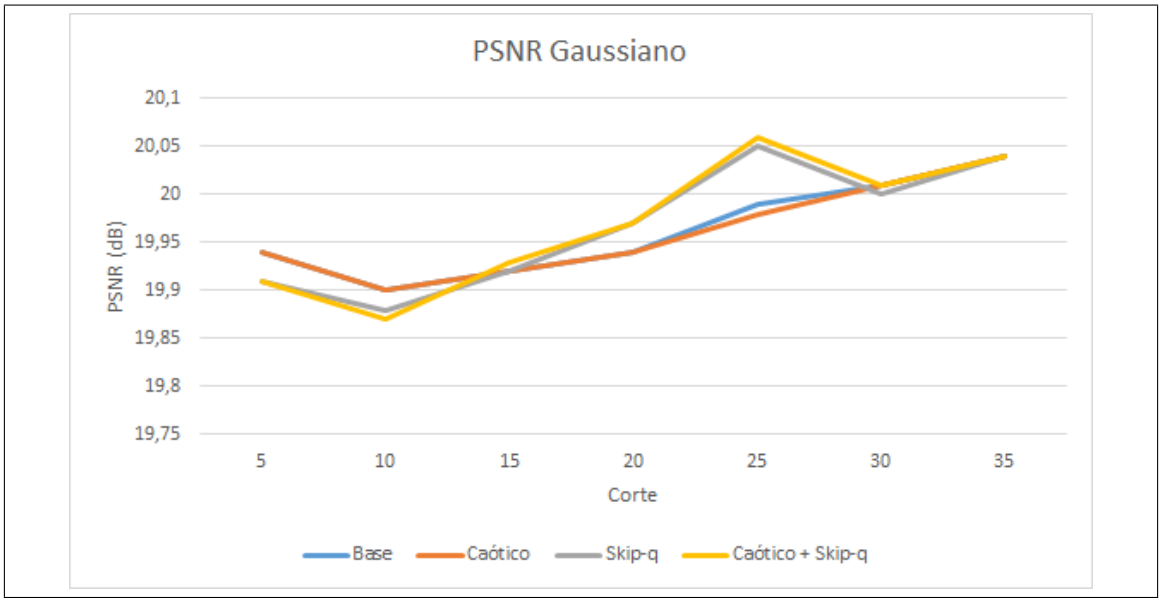

Figura 5.11: PSNR en imágenes con ruido gaussiano.

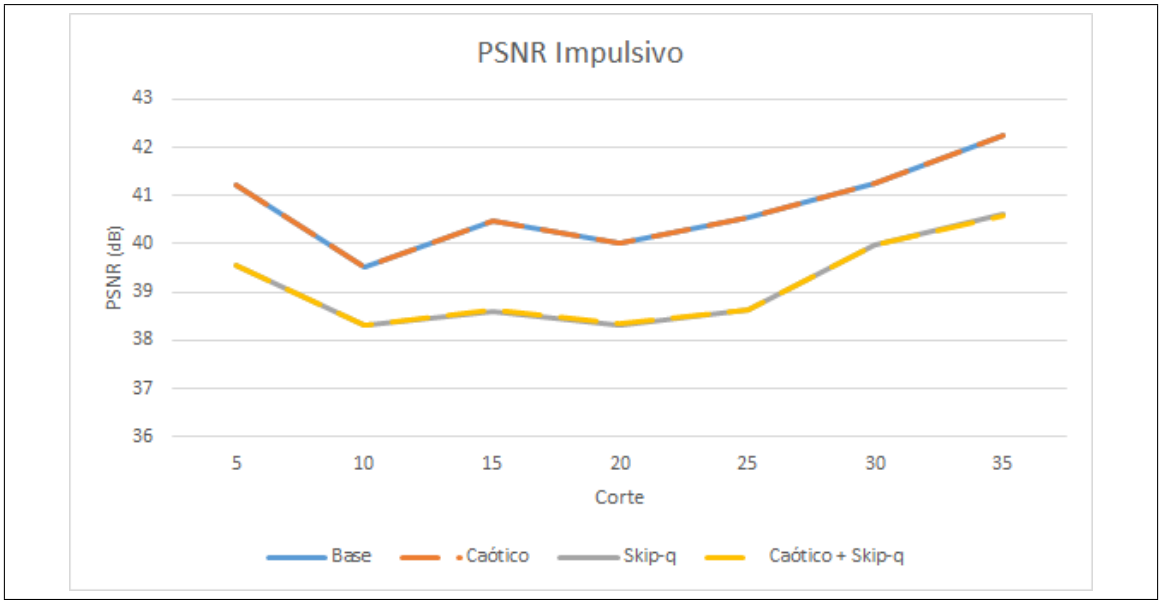

Figura 5.12: PSNR en imágenes con ruido impulsivo.

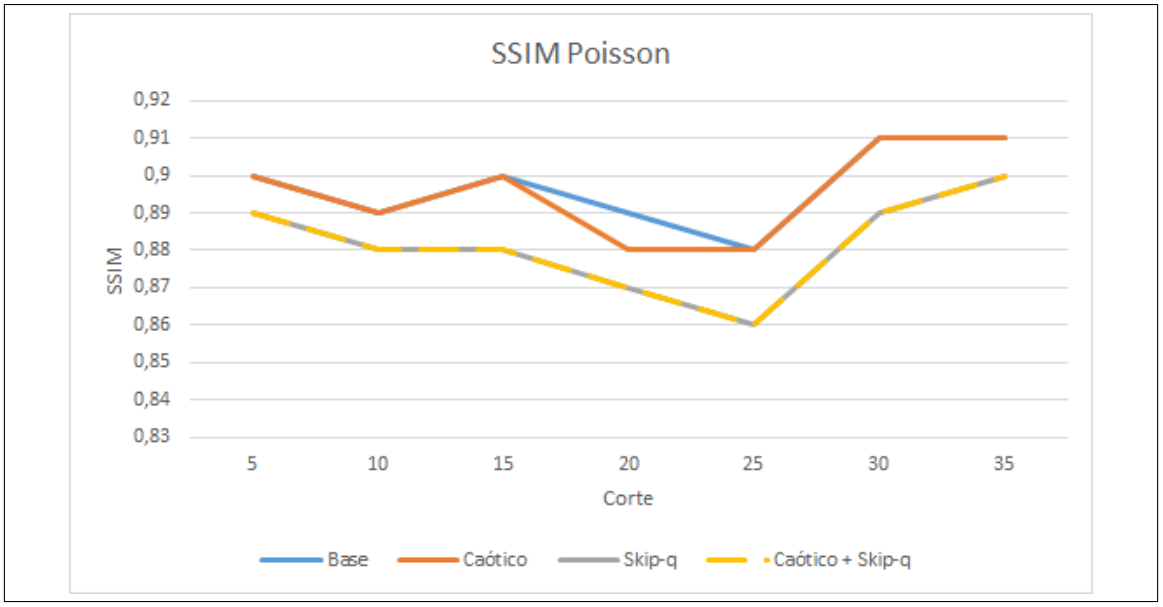

Figura 5.13: SSIM en imágenes con ruido de poisson.

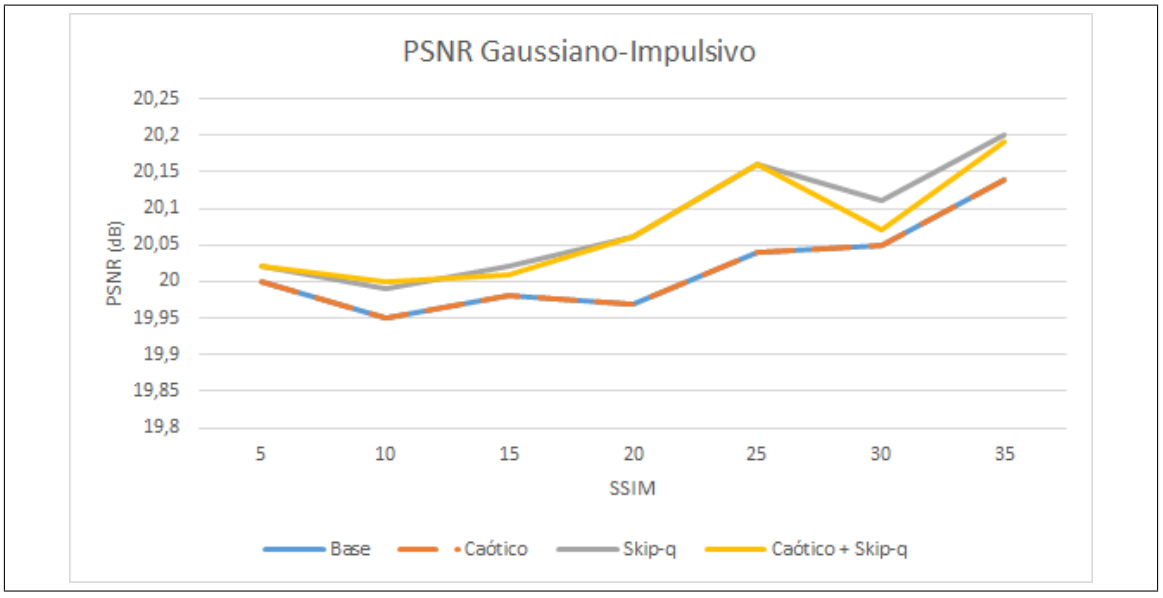

Figura 5.14: PSNR en imágenes con ruido gaussiano-impulsivo.

5.6. Prueba 5: Filtrado en distintas fases de la reconstrucción

5.6.1. Descripción del filtrado en distintas fases de la reconstrucción

Para el estudio sobre sinogramas se ha decidido evaluar sólo dos de los métodos de filtrado, las versiones caóticas base y *skip-q*, ya que en los estudios anteriores han demostrado tener un comportamiento idéntico a sus versiones no caóticas pero siendo más veloces. Este estudio consiste en analizar la calidad de las imágenes filtrando en distintas etapas del proceso de reconstrucción, en concreto filtrando sobre el sinograma, filtrando sobre la imagen reconstruida y filtrando sobre ambos. En este primer apartado explicaremos las fases de la reconstrucción y cómo han funcionado los filtros en cada una, y en la siguiente sección analizaremos la calidad de las imágenes resultantes del estudio.

Se han tomado 50 sinogramas del fantoma Shepp-Logan con y sin ruido, a partir de los cuales se puede realizar la reconstrucción de la imagen del fantoma.

Primero, como se muestra en la figura 5.15, se han filtrado los sinogramas ruidosos. Tanto el método caótico como el caótico con *skip-q* han sido capaces de producir una versión filtrada del sinograma, pero el caótico ha realizado más iteraciones que el *skip-q*.

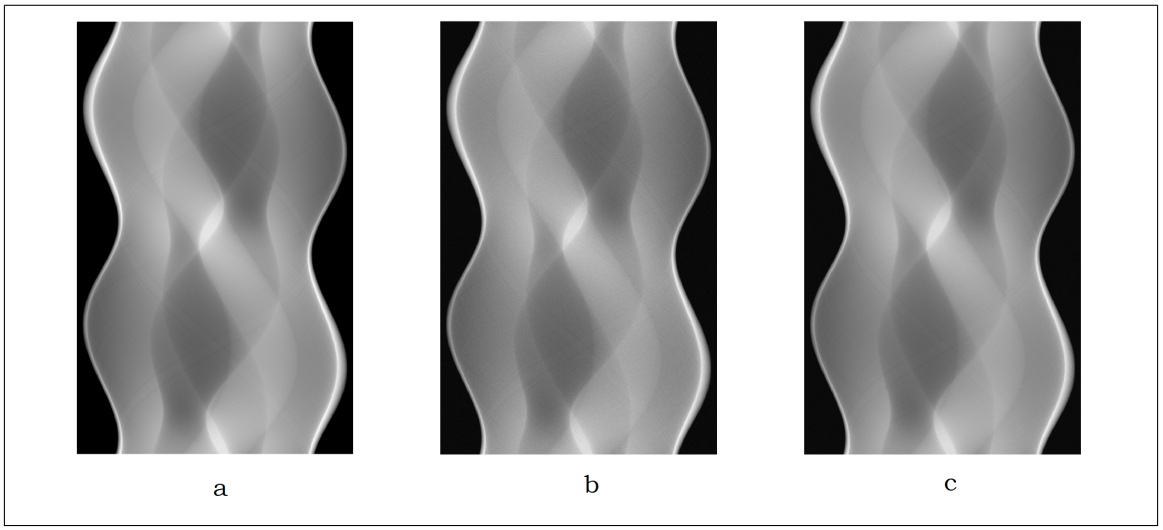

Figura 5.15: (a) sinograma de referencia. (b) sinograma con ruido. (c) sinograma *(b)* filtrado.

A continuación se han reconstruido las imágenes correspondientes a los sinogramas de referencia y ruidosos, y a las imágenes reconstruidas se les ha aplicado de nuevo el filtro. En la figura 5.16 se muestran las tres imágenes resultantes, al aplicar el método caótico. El método *skip-q* no ha sido capaz de producir una mejora sobre la reconstrucción del sinograma ruidoso.

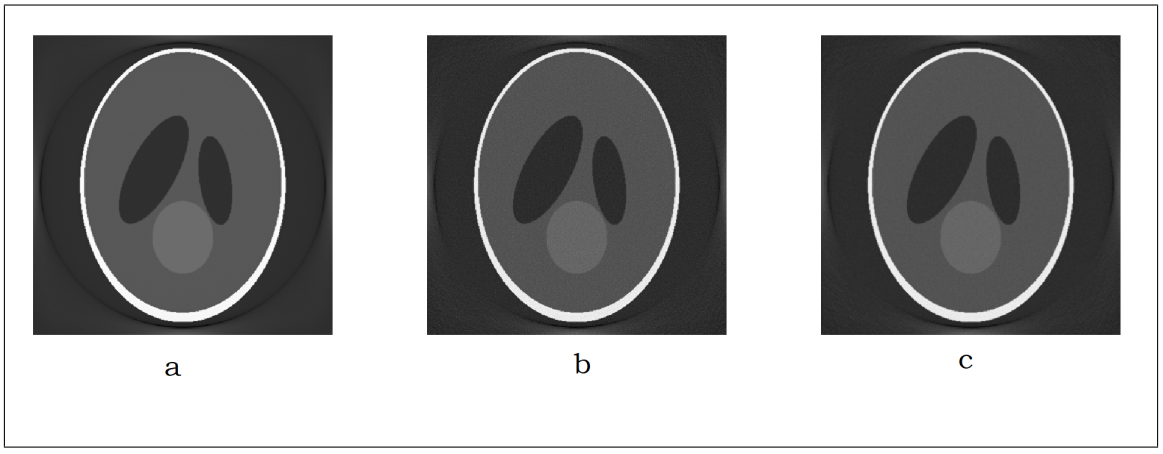

Figura 5.16: (a) Reconstrucción con el sinograma de referencia. (b) Reconstrucción con el sinograma con ruido. (c) Imagen *(b)* filtrada.

Por último, en la figura 5.17 se muestra el último proceso de filtrado, que se ha hecho sobre la reconstrucción del sinograma filtrado en la primera fase (ver figura 2.2). Tan solo el filtro caótico base ha sido capaz de producir una mejora sobre las imágenes, realizando dos iteraciones sobre todos los cortes.

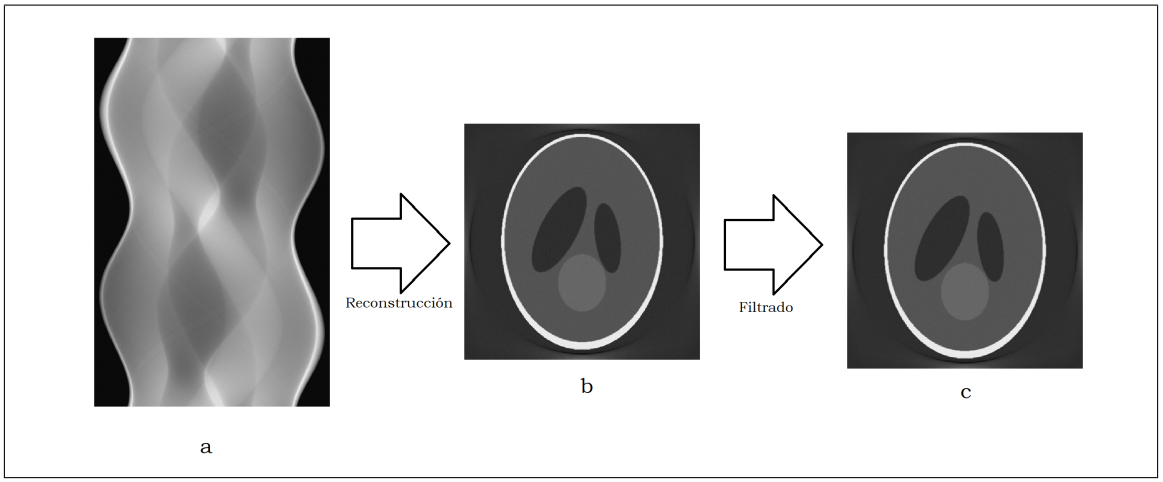

Figura 5.17: (a) sinograma con ruido filtrado. (b) Reconstrucción de *(a)*. (c) Imagen *(b)* filtrada.

5.6.2. Análisis de las imágenes reconstruidas

Se han tomado las imágenes reconstruidas de las fases anteriores y se ha analizado su calidad con ayuda de Matlab. Las imágenes que tenemos son:

1. Reconstrucción del sinograma ruidoso (figura 5.16 b). Se ha llamado a este conjunto de imágenes *Noisy* en las gráficas.
2. Imagen filtrada de la reconstrucción del sinograma ruidoso (figura 5.16 c). Se ha denominado a estas imágenes *Filtered* en las gráficas.
3. Reconstrucción a partir del sinograma filtrado (figura 5.17 b). Denominado *Sino filtered* en las gráficas.
4. Imagen filtrada de la reconstrucción a partir del sinograma filtrado (figura 5.17 c). Y estas imágenes son las que se han llamado *Sino filtered filtered* en las gráficas.

En la figura 5.18 se muestra la gráfica del PSNR de las imágenes mencionadas utilizando el método de filtrado caótico con *skip-q*. El PSNR de la imagen ruidosa está por encima de 30db, lo cual es una calidad buena, como se ha mencionado previamente este nivel alto de calidad de la imagen ruidosa ha provocado que el filtro no actúe sobre la reconstrucción. Lo mismo ha ocurrido con la reconstrucción del sinograma filtrado, que con un PSNR de hasta 35db en algunos cortes no ha podido filtrarse más con este método.

En la figura 5.19 se muestra la gráfica del PSNR utilizando el método de filtrado caótico base. Con este método sí se ha podido filtrar la imagen ruidosa reconstruida, con valores significativamente mejores con respecto a los iniciales. El filtrado sobre el sinograma obtiene resultados mucho mejores con respecto al filtrado sobre la imagen reconstruida ruidosa, pero el filtrado sobre la reconstrucción de esta imagen ya no presenta casi diferencia.

Si se comparan ambos métodos (véase la figura 5.20) se puede observar que ambos obtienen resultados similares cuando filtran sobre el sinograma, pero los resultados del método caótico son ligeramente mejores en mayor cantidad de cortes.

En la figura 5.21 se comparan las métricas del SSIM obtenidas por ambos métodos. En esta gráfica se observa que los resultados iniciales del filtrado sobre el sinograma son buenos para ambos métodos. El filtrado sobre la imagen reconstruida a partir del sinograma filtrado con el método caótico, que previamente aparentaba haber producido muy poca mejora al medir al PSNR, se ve que ha producido una mejora evidente con respecto a la métrica del SSIM, dando 0,97 para todos los cortes.

Para finalizar, con el fin de comparar mejor los resultados del filtrado en todas

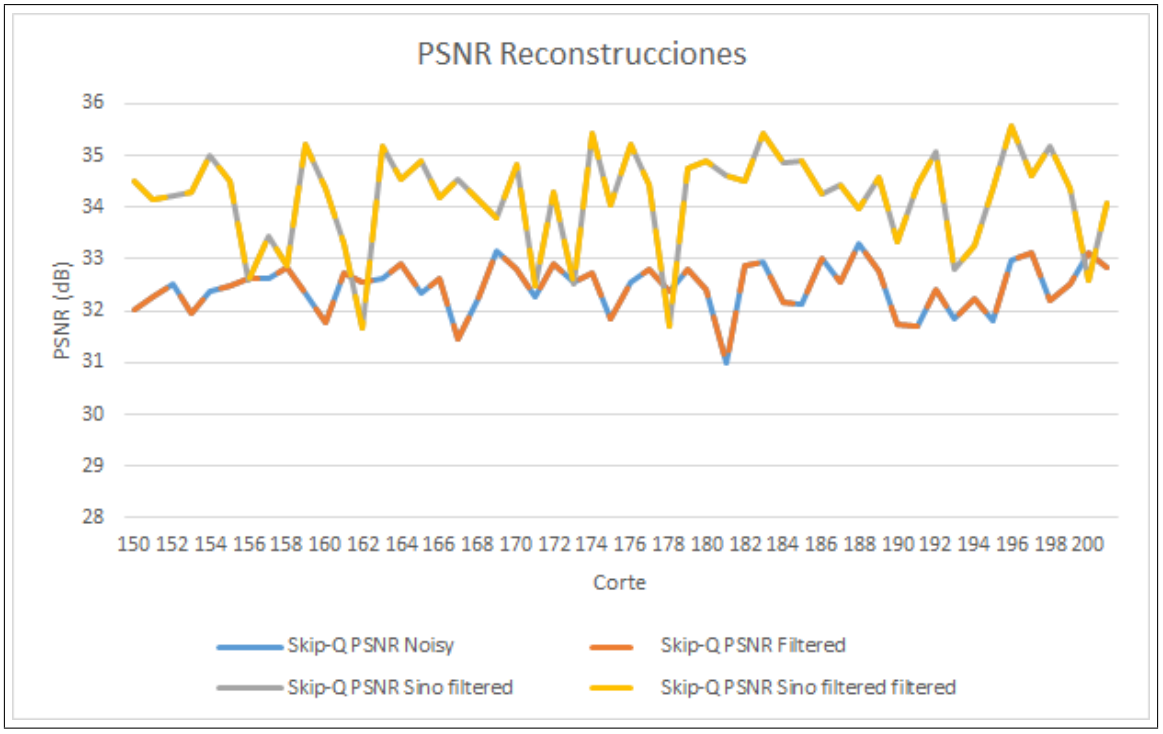

Figura 5.18: PSNR en las imágenes reconstruidas con el método Skip-Q.

las fases, se ha obtenido la diferencia entre las imágenes reconstruidas que hemos analizado y la imagen de referencia para el corte 200 (figura 5.22).

Se puede observar que en (a) hay una granularidad bastante uniforme en toda la imagen, con tan solo los bordes del fantoma distinguibles, después de filtrar esta imagen, en (b), desaparece un poco la granularidad, pero no se aprecia una diferencia significativa con respecto al caso anterior. Con respecto a las imágenes (c) y (d), ambas tienen mucha menos granularidad que las anteriores, indicando una reducción significativa del ruido sobre todo en las zonas oscuras, sin embargo las estructuras del fantoma son mucho más distinguibles. Concluimos que al filtrar el sinograma antes de la reconstrucción se elimina mejor el ruido pero se alteran los niveles de gris de la imagen reconstruida lo que explica los resultados más bajos de PSNR y altos de SSIM.

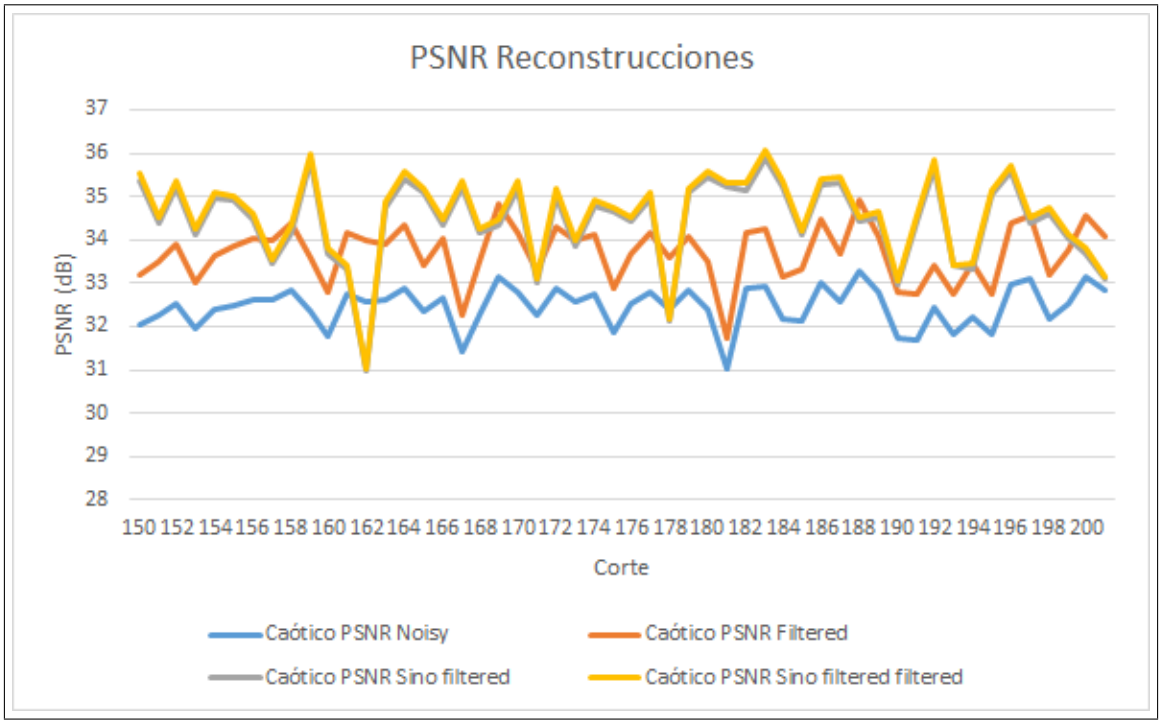

Figura 5.19: PSNR en las imágenes reconstruidas con el método Caótico.

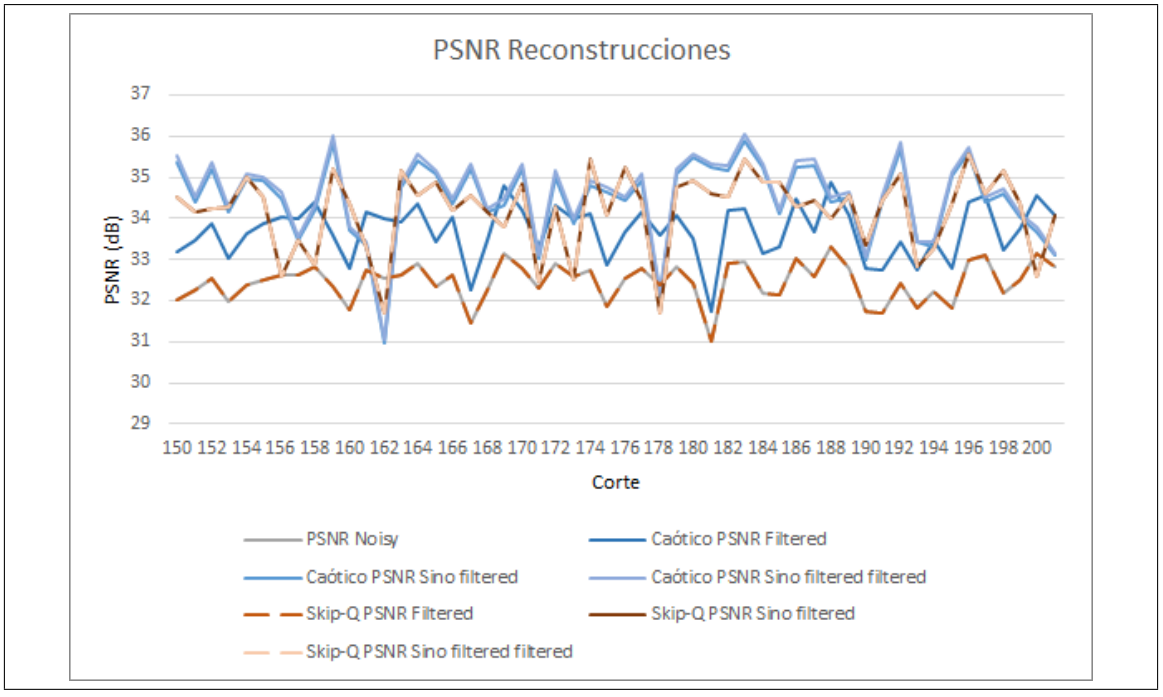

Figura 5.20: PSNR en las imágenes reconstruidas.

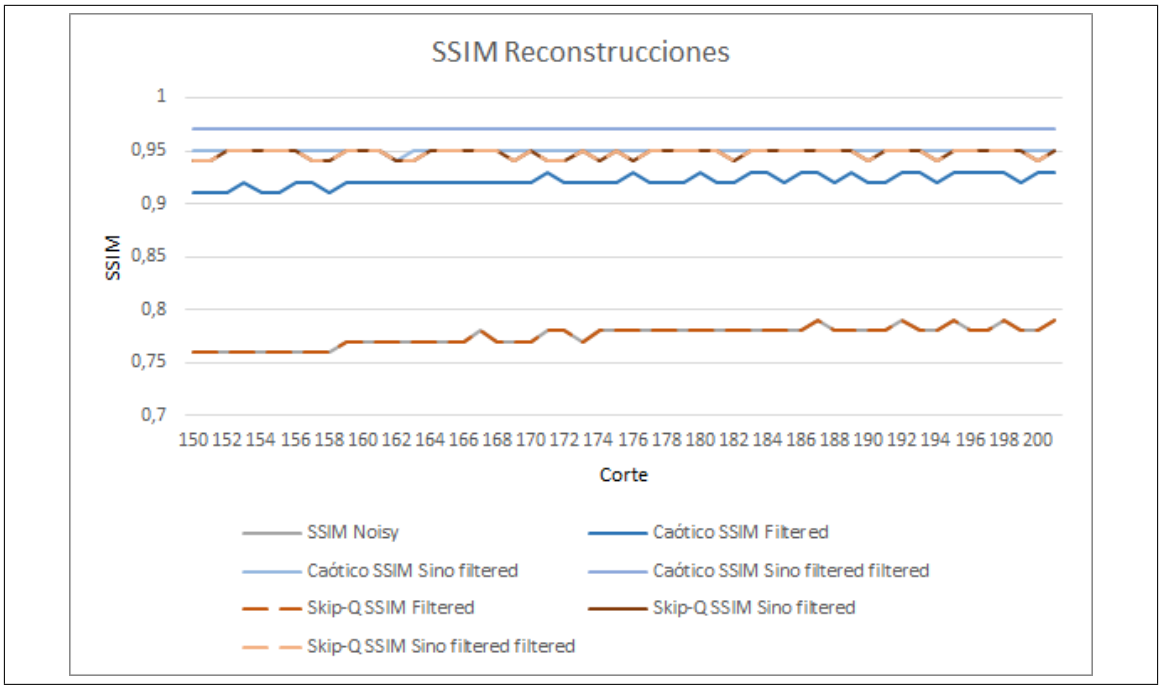

Figura 5.21: SSIM en las imágenes reconstruidas.

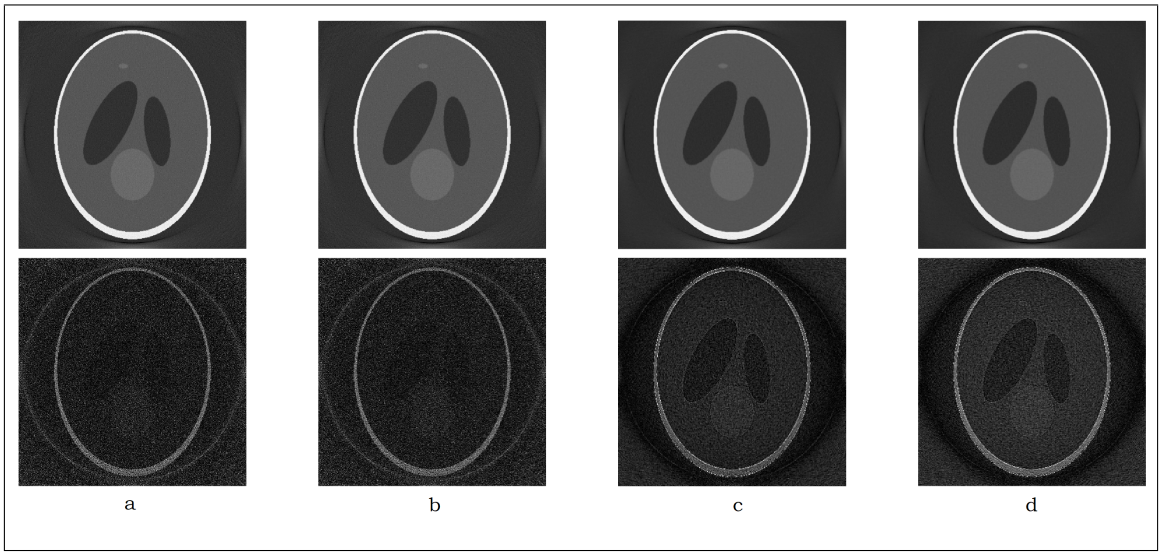

Figura 5.22: En la parte superior: (a) la imagen de reconstrucción ruidosa, (b) la imagen *(a)* filtrada, (c) la imagen de reconstrucción del sinograma filtrado, (d) la imagen *(c)* filtrada. En la parte inferior su diferencia con la imagen de referencia.

Capítulo 6

Conclusiones y Trabajo Futuro

A continuación se va a hacer un repaso de las conclusiones sacadas de este trabajo, los objetivos que se han conseguido y qué líneas de trabajo se nos han planteado a raíz de lo estudiado.

6.1. Conclusiones

En el desarrollo de este trabajo y los estudios realizados se han alcanzado la mayoría de los objetivos que se marcaron en un principio.

Mediante el estudio del algoritmo y de la implementación original se ha podido desarrollar una implementación eficiente en CUDA del filtro gaussiano-impulsivo de lógica difusa. Se ha conseguido adaptar el algoritmo para su funcionamiento en paralelo, eliminando dependencias de datos y sustituyendo clases por punteros, y se han encapsulado los pasos en tres kernels. Al probar la versión secuencial del algoritmo desarrollado este conjunto de cambios parece ralentizar significativamente el filtro, pero son cambios esenciales para la paralelización en GPU. Con todo esto se ha conseguido el objetivo que se marcó del estudio y adaptación del filtro secuencial utilizando el paralelismo de la GPU de manera eficiente.

Se han realizado estudios y mejoras sobre la versión base del filtro desarrollado en GPU, desarrollando otras tres versiones de este. Dos de las versiones desarrolladas son las denominadas caóticas, que se aprovechan de la ausencia de dependencia de datos para acelerar el kernel de filtrado, escribiendo directamente sobre los mismos datos de entrada y así haciendo que no sea necesario la sobre-escritura de punteros entre iteraciones.

Se han estudiado los parámetros del algoritmo de filtrado, y se ha determinado que, en general para casi todas las versiones del filtro, los parámetros que obtienen mayor calidad son $p1 = 90,\ p2 = 150,\ r = 2,\ q = 8$. Estos parámetros son algo diferentes a los que se obtuvieron en el estudio del algoritmo original, esencialmente lo que se ha hecho es aumentar el rango en el que la impulsividad de los píxeles está en una función lineal dependiente del ROD lo que suaviza los valores de impulsividad en general. Combinado con el aumento del parámetro q en uno, esta versión toma mayor cantidad de píxeles en la ventana y cada uno de estos tienen una influencia menor sobre el resultado final.

Se ha estudiado también qué métrica de calidad obtiene mejores resultados cuando se utiliza como criterio de parada. Aunque los resultados no han concluido que ninguna métrica sea mejor que otra, este estudio ha permitido descartar las hipótesis en las que nos basábamos al desarrollar estos criterios. El PSNR en cuadro no obtiene resultados distintos al PSNR en la imagen completa, por lo que se comprueba que el impacto que tiene la presencia del fondo negro en las imágenes de tomografía a la hora de calcular el PSNR tras el filtrado no es significativo. También verificamos que el SSIM óptimo se alcanza en un punto muy similar al PSNR óptimo, por lo que no existe ventaja por utilizarlo como criterio de parada. Aparte de analizar la calidad de filtrado mediante métricas objetivas de calidad de imagen, presentamos algunas imágenes que muestran visualmente la calidad de las imágenes filtradas.

Con respecto a la calidad del filtro hay que apuntar que se han obtenido peores resultados en general para las versiones *skip-q*, pero la diferencia con el resto de versiones es mínima. Además, al estudiar ruidos simulados, se obtuvo que las versiones *skip-q* funcionaban mejor en imágenes con ruido mixto gaussiano-impulsivo. La mayor diferencia ha ocurrido en la prueba realizada sobre los sinogramas, donde esta versión no ha sido capaz de filtrar las imágenes reconstruidas, que tenían muy poco ruido. Por lo tanto, dependiendo de la situación y del tipo de ruido que se pretende filtrar, el usuario final podrá elegir una versión u otra.

Las distintas versiones desarrolladas sobre el filtro mejoran significativamente su velocidad. De los resultados obtenidos en el estudio de aceleración, con las imágenes de tamaño 512x512, se ha determinado que la versión más rápida del filtro es la denominada caótica con *skip-q* y llega a ser casi 365 veces más rápida que la versión secuencial, con una diferencia insignificante en la calidad al ser comparada al resto de versiones. Aun así, la versión del filtro que denominamos caótica, que es la que mejores resultados de calidad ha obtenido en la mayoría de pruebas, no se queda atrás en velocidad, llegando a ser casi 360 veces más rápida que la secuencial en imágenes de resolución 512x512. Cuando estudiamos la aceleración en imágenes de mayores dimensiones alcanzamos valores de aceleración de hasta 550 para la versión caótica con *skip-q* y 544 para la caótica. Se extrae de este estudio que la aceleración obtenida

es mayor conforme mayor es la resolución de la imagen filtrada, pero sería necesario continuar este estudio con un mayor banco de imágenes grandes para verificar este resultado. Es considerable también la mejora que se ha obtenido con respecto a la paralelización en CPU, ya que los resultados que obtenemos son hasta 17 veces mejores que para la versión OpenMP ejecutada en 32 núcleos.

Por último, aunque las implementaciones CUDA desarrolladas obtienen excelentes valores de speedup, si los comparamos con aquellos obtenidos por los filtros estudiados en el estado del arte que conseguían una aceleración de hasta 8000 veces, nuestra optimización no alcanza ese nivel. Es posible que el filtro estudiado no sea compatible con este tipo de dramática aceleración, una de las posibles causas es el uso de reglas fuzzy de tipo if-then, ya que la aparición de sentencias de este tipo en los kernel puede afectar el rendimiento. Aun así, los tiempos por iteración alcanzados para imágenes pequeñas, del orden de 10^{-4} segundos, muestran que el filtro es apto para integrarlo en aplicaciones en tiempo real.

6.2. Trabajo Futuro

Como resultado del trabajo desarrollado se plantean diversas líneas de trabajo futuro que permitirán seguir explorando la aplicación de métodos de filtrado basados en lógica fuzzy a la imagen médica.

Se plantea realizar un nuevo estudio de una versión en la que el vector impulsividad se deje de recalcular a partir de cierta iteración. Para ello, sería necesario estudiar las siguientes cuestiones:

- ¿Cuando se corta el cálculo de la impulsividad? ¿Depende del nivel de ruido?
- ¿Funciona tanto para imágenes de radiografía como de tomografía? ¿Funciona para todo tipo de ruido? ¿Funciona para todo tamaño de imagen?
- ¿Es compatible con los parámetros de p1, p2 y r determinados anteriormente?

La aceleración que se conseguiría con esta implementación es significativa, ya que el segundo kernel es el más costoso temporalmente.

Se ha implementado el uso de la memoria pinned para acelerar la copia de datos, pero existen otros sistemas de copia de datos que podrían producir mejoras como son el uso de memoria compartida o memoria unificada. La implementación de estos sistemas de memoria requerirían una reestructuración significativa del filtro desarrollado.

Otra limitación del filtro es el uso del formato PGM como punto intermedio entre las imágenes en formato DICOM y el filtro. Al pasar a enteros de 8 bits, se pierde

gran cantidad de información que estaba presente en los datos de DICOM, donde las imágenes se almacenan como matrices de dobles. Habría que rediseñar la carga de datos, para que acepte matrices de datos en lugar de imágenes PGM, pero el resto del filtro podría seguir funcionando sin cambios mayores.

De cara a otros trabajos futuros, interesaría realizar un estudio analizando los tipos y niveles de ruido presentes en los datos que tenemos con ruido natural para poder generalizar su simulación. Con esta información, se podría analizar el número de iteraciones óptimas para cada tipo y nivel de ruido para que el filtro no cause una mayor degradación de la imagen.

Relacionado con esto, se ha planteado reunir un conjunto de imágenes para que los sanitarios miembros del grupo de investigación puedan evaluar la eficacia de los resultados del filtrado. Se buscará determinar hasta qué nivel de filtrado se mantiene la información de las imágenes y si existen niveles de ruido para los que el resultado del filtrado empeore la imagen.

Como extensión a un entorno real de las pruebas realizadas sobre los sinogramas, sería interesante integrar el filtro con algún proceso de reconstrucción de imagen médica de Tomografía Computarizada. Se integraría tras la fase de adquisición, para realizar el filtrado del sinograma en conjunto con el filtro del proveedor, y dentro de la fase de reconstrucción iterativa.

Por último, también relacionado con la integración del filtro en casos reales, sería interesante evaluar la eficiencia del filtro en máquinas con las características de los ordenadores de los hospitales que realizan estas operaciones actualmente. Esto permitiría determinar si se consigue el filtrado en tiempo real que buscábamos con la implementación en CUDA.

Referencias

[1] Mark S Pearce et al. "Radiation exposure from CT scans in childhood and subsequent risk of leukaemia and brain tumours: a retrospective cohort study". En: *The Lancet* 380.9840 (2012), págs. 499-505.

[2] I Musdalifah, S Suryani y N Rauf. "Relation between body mass index and position of chest examination as a function of entrance surface dose". En: *Journal of Physics: Conference Series*. Vol. 1341. 8. IOP Publishing. 2019, pág. 082007.

[3] Madan M Rehani et al. "Patients undergoing recurrent CT scans: assessing the magnitude". En: *European radiology* 30 (2020), págs. 1828-1836.

[4] Xinhui Duan et al. "Electronic noise in CT detectors: impact on image noise and artifacts". En: *American Journal of Roentgenology* 201.4 (2013), W626-W632.

[5] Mónica Chillarón, Vicente Vidal y Gumersindo Verdú. "Evaluation of image filters for their integration with LSQR computerized tomography reconstruction method". En: *PLoS one* 15.3 (2020), e0229113.

[6] Josep Arnal et al. "A parallel fuzzy algorithm for real-time medical image enhancement". En: *International Journal of Fuzzy Systems* 22 (2020), págs. 2599-2612.

[7] S Mark, D Henderson y J Brealey. "Taking acute medical imaging to the patient, the domiciliary based X-ray response team". En: *Radiography* 28.2 (2022), págs. 550-552.

[8] Junsung Park, Geunyoung An y Hee Seo. "Design optimization of X-ray security scanner based on dual-energy transmission imaging with variable tube voltage". En: *Journal of Instrumentation* 18.01 (2023), pág. C01063.

[9] Manoj Diwakar y Manoj Kumar. "A review on CT image noise and its denoising". En: *Biomedical Signal Processing and Control* 42 (2018), págs. 73-88.

[10] María G Sánchez et al. "Image noise removal on heterogeneous cpu-gpu configurations". En: *Procedia Computer Science* 29 (2014), págs. 2219-2229.

[11] Joan-Gerard Camarena et al. "A simple fuzzy method to remove mixed Gaussian-impulsive noise from color images". En: *IEEE Transactions on Fuzzy Systems* 21.5 (2012), págs. 971-978.

[12] Ignacio Encinas Rubio. "Algoritmos paralelos para la reducción de ruido mixto gaussiano-impulsivo en imágenes médicas". Trabajo Fin de Grado. Alicante: Universidad de Alicante, mayo de 2022.

[13] Josep Arnal, Juan B Pérez y Vicente Vidal. "A Parallel Fuzzy Method to Reduce Mixed Gaussian-Impulsive Noise in CT Medical Images". En: *Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery: Volume 1*. Springer. 2020, págs. 975-982.

[14] Samuel Morillas, Valentín Gregori y Antonio Hervás. "Fuzzy peer groups for reducing mixed Gaussian-impulse noise from color images". En: *IEEE Transactions on Image Processing* 18.7 (2009), págs. 1452-1466.

[15] Linda CP Croton et al. "Ring artifact suppression in X-ray computed tomography using a simple, pixel-wise response correction". En: *Optics express* 27.10 (2019), págs. 14231-14245.

[16] Lesia Mochurad y Roman Bliakhar. "Comparison of the Efficiency of Parallel Algorithms KNN and NLM Based on CUDA for Large Image Processing." En: *CMIS*. 2022, págs. 238-249.

[17] Xiaobo Zhang. "Two-step non-local means method for image denoising". En: *Multidimensional Systems and Signal Processing* 33.2 (2022), págs. 341-366.

[18] Ju Zhang et al. "A novel denoising method for low-dose CT images based on transformer and CNN". En: *Computers in Biology and Medicine* (2023), pág. 107162.

[19] Avinash C Kak y Malcolm Slaney. *Principles of computerized tomographic imaging*. SIAM, 2001.

[20] Lotfi A Zadeh. "Fuzzy logic". En: *Granular, Fuzzy, and Soft Computing*. Springer, 2023, págs. 19-49.

[21] C McCollough et al. "Low dose ct image and projection data [data set]". En: *The Cancer Imaging Archive* 10 (2020).

[22] ISO/TC 215 Health informatics. *ISO 12052:2017. Health informatics — Digital imaging and communication in medicine (DICOM)*. Ago. de 2017. URL: `https://www.iso.org/standard/72941.html`.

[23] Taylor R Moen et al. "Low-dose CT image and projection dataset". En: *Medical physics* 48.2 (2021), págs. 902-911.

[24] Lifeng Yu et al. "Development and validation of a practical lower-dose-simulation tool for optimizing computed tomography scan protocols". En: *Journal of computer assisted tomography* 36.4 (2012), págs. 477-487.

[25] Lawrence A Shepp y Benjamin F Logan. "The Fourier reconstruction of a head section". En: *IEEE Transactions on nuclear science* 21.3 (1974), págs. 21-43.

[26] *The ASTRA Toolbox — ASTRA Toolbox 2.1.0 documentation*. en. URL: `https://www.astra-toolbox.com/`.

[27] Craig Hacking y Frank Gaillard. "Normal brain CT". En: *Radiopaedia.org* (abr de 2015). DOI: 10.53347/rid-35508. URL: https://doi.org/10.53347/rid-35508.

[28] María G Sánchez et al. "Estimación de la Reducción de la radiación de dosis usando el método de Difusión No-Lineal en Radiografías Computarizadas". En: *Nuclear España* (2012).

[29] Jef Poskanzer. *PGM Format Specification*. 1991. URL: https://netpbm.sourceforge.net/doc/pgm.html.

[30] Konstantinos Plataniotis y Anastasios N Venetsanopoulos. *Color image processing and applications*. Springer Science & Business Media, 2000.

[31] Samuel W Hasinoff. "Photon, Poisson Noise." En: *Computer Vision, A Reference Guide* 4.16 (2014), pág. 1.

[32] Hans Marmolin. "Subjective MSE measures". En: *IEEE transactions on systems, man, and cybernetics* 16.3 (1986), págs. 486-489.

[33] Umme Sara, Morium Akter y Mohammad Shorif Uddin. "Image quality assessment through FSIM, SSIM, MSE and PSNR—a comparative study". En: *Journal of Computer and Communications* 7.3 (2019), págs. 8-18.

[34] Alain Hore y Djemel Ziou. "Image quality metrics: PSNR vs. SSIM". En: *2010 20th international conference on pattern recognition*. IEEE. 2010, págs. 2366-2369.

[35] Alan C Brooks, Xiaonan Zhao y Thrasyvoulos N Pappas. "Structural similarity quality metrics in a coding context: exploring the space of realistic distortions". En: *IEEE Transactions on image processing* 17.8 (2008), págs. 1261-1273.

[36] *CUDA Toolkit Documentation 12.1 Update 1*. en. URL: https://docs.nvidia.com/cuda/index.html.

[37] *Thrust — NVIDIA Developer*. en. Oct. de 2018. URL: https://developer.nvidia.com/thrust.

[38] *How to Optimize Data Transfers in CUDA C/C++ — NVIDIA Technical Blog*. en-US. ago de 2022. URL: https://developer.nvidia.com/blog/how-optimize-data-transfers-cuda-cc/.

[39] *NVIDIA 2D Image And Signal Performance Primitives (NPP): IQA*. URL: https://docs.nvidia.com/cuda/npp/group__image__quality__assessment.html.

[40] *Generate C Code by Using the MATLAB Coder App - MATLAB Simulink - MathWorks España*. es. URL: https://es.mathworks.com/help/coder/gs/generating-c-code-from-matlab-code-using-the-matlab-coder-project-interface.html.

Apéndice A

Script para generación de imágenes PGM a partir de imágenes en unidades Hounsfield

```
  clc
clear
load("images_full_low.mat");
a = size(C012_images_low,3);
for i=5:5:a
    su = "images/C012_"+i+"_full.pgm";
    sl = "images/C012_"+i+"_low.pgm";
    U = C012_images_full(:,:,i);
    L = C012_images_low(:,:,i);
    U0 = uint8(255 * mat2gray(U));
    L0 = uint8(255 * mat2gray(L));
    imwrite(U0,su);
    imwrite(L0,sl);
end
a = size(L064_images_low,3);
for j=5:5:a
    su = "images/L064_"+j+"_full.pgm";
    sl = "images/L064_"+j+"_low.pgm";
    U = L064_images_low(:,:,j);
    L = L064_images_low(:,:,j);
    U0 = uint8(255 * mat2gray(U));
    L0 = uint8(255 * mat2gray(L));
    imwrite(U0,su);
```

```
24      imwrite(L0,sl);
25  end
26  a = size(N005_images_low,3);
27  for k=5:5:a
28      su = "images/N005_"+k+"_full.pgm";
29      sl = "images/N005_"+k+"_low.pgm";
30      U = N005_images_low(:,:,k);
31      L = N005_images_low(:,:,k);
32      U0 = uint8(255 * mat2gray(U));
33      L0 = uint8(255 * mat2gray(L));
34      imwrite(U0,su);
35      imwrite(L0,sl);
36  end
```

Apéndice B

Script para analizar la calidad (ejemplo de la prueba del parámetro p1 y p2)

```
  clc
clear
load("images_full_low.mat");
format = '%d;%d;%d;%5.2f;%5.2f;%5.2f;%5.2f;%5.2f;%5.2f\n';
a = size(N005_images_low,3);
file = fopen("images/calidad_N005_P1P2.csv",'w');
for p1=20:10:90
    for p2=90:10:150
        for i=5:5:a
            su = "images/N005_"+i+"_full.pgm";
            sl = "images/N005_"+i+"_low.pgm";
            so2 = "images/N005_"+i+"_"+p1+"_"+p2+"_out.pgm";
            U = imread(su);
            L = imread(sl);
            O2 = imread(so2);
            psnr0 = psnr(L,U);
            psnr2 = psnr(O2,U);
            mse0 = immse(L,U);
            mse2 = immse(O2,U);
            ssim0 = ssim(L,U);
            ssim2 = ssim(O2,U);
            fprintf(file,format,i,p1,p2,psnr0,psnr2,mse0,mse2
               ,ssim0,ssim2);
```

```
23          end
24      end
25  end
26  fclose(file);
27  clear
```

Apéndice C

Scripts para añadir ruido a las imágenes

```
  clc
clear
load("images_full_low.mat");
a = size(N005_images_low,3);
for k=5:5:a
    s0 = "images/N005_"+k+"_gau.pgm";
    s1 = "images/N005_"+k+"_imp.pgm";
    s2 = "images/N005_"+k+"_poi.pgm";
    s3 = "images/N005_"+k+"_gauimp.pgm";
    U = N005_images_full(:,:,k);
    U0 = uint8(255 * mat2gray(U));
    UD = double(imread("images/N005_"+k+"_full.pgm"));
    SG = imnoise(U0, 'gaussian', 0.1);
    SI = imnoise(U0, 'salt & pepper');
    SP = add_poisson_noise(UD,10);
    SP = uint8(255 * mat2gray(SP));
    SGI = imnoise(SG, 'salt & pepper');
    imwrite(SG,s0);
    imwrite(SI,s1);
    imwrite(SP,s2);
    imwrite(SGI,s3);
end
clear
```

```
function I_out = add_poisson_noise_double(I,level)
  % scale to [0,1]
  max_I = max(I(:));
  IScaled = I ./ max_I;
  % add poison noise
  P_A = IScaled * power(10,-level);
  P_B = imnoise(P_A, 'poisson');
  P_C = P_B * power(10,level);
  % to density
  I_out = max_I .* P_C;
end
```