UNIVERSIDAD DEL BÍO-BÍO

FACULTAD DE INGENIERÍA
DEPTO. INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

"Estudio y evaluación de técnicas de disimulación de errores en la comunicación de imágenes sobre redes de sensores inalámbricos"

MAURICIO ANTONIO BAEZA RIQUELME, CHRISTOPHER ALEXIS CARRERA NOCHE.

TRABAJO DE TÍTULO PARA OPTAR A TÍTULO DE INGENIERO CIVIL EN AUTOMATIZACIÓN

Concepción - Chile
2016

UNIVERSIDAD DEL BÍO-BÍO

FACULTAD DE INGENIERÍA
DEPTO. INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

"Estudio y evaluación de técnicas de disimulación de errores en la comunicación de imágenes sobre redes de sensores inalámbricos"

MAURICIO ANTONIO BAEZA RIQUELME, CHRISTOPHER ALEXIS CARRERA NOCHE.

Profesores Guías
Dr. Cristian Durán Faúndez
Dr. Krzysztof Herman

Agradecimientos

En primer lugar darle gracias a Dios por todas aquellas personas que hemos conocido a lo largo de nuestra carrera universitaria, por nuestras familias que han estado ahí cuando las necesitamos, nombrar a cada uno seria demasiado largo sin embargo no podemos desmerecer el trabajo de nuestras madres que nos apoyan día a día sin importar cuan desordenados o porfiados seamos, su certeza de que podíamos hacer este trabajo posible, en ocasiones superaba a las nuestras.

También agradecer a nuestro profesor Cristian Durán Faúndez que nos acogió cuando estábamos perdidos y sin rumbo, por darnos esperanza y reemplazar todas las sensaciones de angustia que teníamos por confianza y seguridad, siempre tratábamos de hacer todo por nuestra cuenta y en los momentos en que el conocimiento no era suficiente el siempre nos sacaba del problema, también la exigencia de sus ramos nos hizo madurar no solo como estudiantes sino también como profesionales, por eso y mucho mas, gracias totales.

Agradecer a algunos profesores que nos dieron su apoyo cuando nos veían mal como el profesor Antipil quien nos alegraba con su humor y nos levantaba el ánimo o Don Pedro de pañol quien con su simpatía se gano nuestro respeto y cariño a tal punto que si el no estaba en pañol no pedíamos nada hasta que el llegara y también a nuestro profesor Luis Vera, con quien compartimos mucho en el CIMUBB.

A nuestra querida y hermosa Sandrita que en el ultimo año le pedimos toda la ayuda que no le pedimos en los 6 años de carrera y siempre estuvo dispuesta a atendernos con su sonrisa a pesar de llegar siempre fuera del horario de atención, darle solo las gracias es poco.

A los amigos y amigas que conocimos y que llegan con nosotros hasta este momento, siempre apoyándonos en cada ramo, cada tarde de estudio era mejor junto a ellos, algunos de esos amigos acompañándonos en aventuras extremas que quedaran grabadas en nuestra

memoria. Por aquellos y aquellas que alguna vez fueron amigos y la vida separo nuestros caminos, todos ellos nos dieron alegrías, penas y otros sentimientos que culminan en las personas que somos ahora.

Simplemente a todos ellos y a los que no nombramos pero que están en nuestras mentes y corazones (en especial al Sr. Toki por levantarnos todos los días temprano) les damos las gracias y les dedicamos nuestro trabajo de titulo.

Gracias, gracias por formar parte de esta parte de nuestras vidas...
Gracias totales.

Resumen

En la vida cotidiana y en la industria es frecuente el uso de redes inálambricas de visión, como por ejemplo, en la minería o en construcciones bajo tierra como túneles, incluso en el monitoréo de bosques. Debido al alto costo que conlleva la instalación de equipos alámbricos, y a la necesidad de equipos de bajo consumo, es que nos encontramos con la transmisión de imágenes de manera inalámbrica sin acuso de recepción.

Las transmisiones sin acuso de recepción llevan consigo un gran problema; en caso de perderse un paquete de datos en el el camino, no habrá manera de recuperarlo. Debido a esto, se han desarrollado una gran cantidad de investigaciones para apalear este problema, entre ellas, están las restauraciones en el nodo receptor mediante algoritmos de *inpainting* y *error concealment*, este ultimo enfocado específicamente en reconstruir imágenes que se han sido afectadas con perdida de información.

Debido a la gran cantidad de algoritmos, es importante establecer cual es el que ofrece mejor desempeño con respecto a sus pares y a otro tipo de técnicas.

En este trabajo se propone estudiar los algoritmos de *inpainting* y *error concealment* más conocidos y utilizados para realizar una comparación entre ellos. Para ello se utiliza la herramienta de simulación Sim-LIT, con la cual se desarrollan 240.000 simulaciones bajo parámetros de simulación preestablecidos.

Durante el desarrollo del trabajo se modelan los algoritmos para traspasarlos a lenguaje de programación, se analiza el resultado y se establece la comparación entre los distintos métodos.

Los resultados arrojados nos dejan como conclusión que se debe analizar los algoritmos bajo ciertas condiciones. Por ejemplo, para transmisión de imágenes en bloques de 8x8 el

Maximally smooth recovery es el más recomendable. Por otro lado, para demás tamaños de bloques el bicúbico resulta ser el más robusto a la hora de mantener resultados en las distintas simulaciones, además de tener una muy baja dispersión de resultados.

Los algoritmos de *inpainting* a pesar de entregar excelentes resultados con PSNR muy alto, también arrojan muchas simulaciones erróneas, con PSNR negativo lo que se refleja en imágenes completamente obscuras bajo los escenarios propuestos, por lo que no entregan la confianza para ser utilizados en sistemas reales de redes inalámbricas.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En la era tecnológica en la que nos encontramos inmersos la demanda de información se ha vuelto de gran interés, ya que lo queremos saber todo, y pareciera ser que entre más recóndita es la fuente de la información mayor interés tenemos en ella. Prueba de ello es que constantemente se están desarrollando herramientas que nos permiten recolectar datos de forma remota. Todos los datos generados por sistemas de transmisión no guiada, en algún momento de su trayectoria, desde la fuente hasta el usuario final, deben ser previamente tratados: modulación, compresión, reordenamiento y/o paquetizado de datos, y necesariamente transmitidos a través de un medio inalámbrico. Ejemplo de ello son sondas que exploran la superficie de otros planetas, o satélites utilizados para el monitoreo meteorológico, conectividad a internet mediante WiFi y en este caso, las redes de sensores inalámbricos.
Las redes de sensores inalámbricos de bajo consumo energético que carecen de acuso de recepción, tienen una alta probabilidad de perder información durante la transmisión. En el caso de las redes de sensores de visión, las pérdidas que preocupan son los paquetes de datos de las imágenes transmitidas. Con esto claro, en el presente trabajo de título, se propone como objetivo principal el estudiar y evaluar técnicas de disimulación de errores sobre imágenes digitales estáticas en escenarios de comunicación sobre redes de sensores inalámbricos de visión, bajo ciertas condiciones establecidas de pérdida y transmisión. Para poder llevar a cabo éste estudio se recurre a un entorno de simulación llamado Sim-LIT, el cual emula la transmisión de imágenes entre dos nodos inalámbricos.

Como objetivo final, se pretende estudiar y evaluar técnicas de disimulación de errores en imágenes digitales estáticas en escenarios de comunicación sobre redes de sensores inalámbricos de visión, utilizando para ello Sim-LIT. Esta herramienta de simulación permite modificar parámetros como porcentajes de perdida en la transmisión inalámbrica de imágenes, tamaño

del byte de los paquetes de transmisión, tipo de algoritmo de disimulación de errores en las imágenes recibidas, entre otras prestaciones.

Para lograr el objetivo principal, se deberá realizar un estudio bibliográfico, que permita identificar y conocer diferentes métodos de disimulación de errores en imágenes digitales. Construir una librería implementando los distintos métodos de disimulación de errores en el simulador Sim-LIT. Diseñar un banco de prueba que permita el análisis y comparación de los distintos métodos encontrados. Finalmente ejecutar y analizar los resultados de la simulación.

En el capítulo N°1: Se realiza el estudio del estado del arte de las técnicas de *inpainting* y *error concealment*, se hace un análisis de las investigaciones que han sido realizadas en el tema y algunos de los algoritmos desarrollados para la restauración y/o reconstrucción de imágenes, para luego seleccionar aquellos que serán utilizados en el estudio.

En el capítulo N°2: Se desglosan las ecuaciones de cada método para entender el funcionamiento y posteriormente se normalizan a sus respectivas implementaciones numéricas, con el fin de aplicarlas en lenguaje de programación.

En el capítulo N°3: Se da paso a la definición de los parámetros que se utilizan para realizar las simulaciones así como también la cantidad y el tipo de imágenes. También se aborda el tema de como agregar un nuevo algoritmo a Sim-LIT.

Capítulo N°4: La etapa de simulación es donde se presentan los resultados de cada una de las simulaciones valga la redundancia, para cada imagen, algoritmo y parámetro de transmisión. Los resultados se presentan en forma de diagramas de caja y bigotes con el fin de analizar la dispersión, la media, mediana y los valores atípicos de los resultados.

Capítulo N°5: En la sección de análisis se comparan los resultados y se establecen las tendencias, probabilidades y desempeño de cada algoritmo frente a distintos parámetros de transmisión. Para comparar los resultados se utilizan gráficos con la curva de la media de cada método, cada imagen y con cada parámetro de transmisión.

Finalmente, en el Capítulo N°6: Sección de comentarios y conclusiones se presentan las resoluciones con respecto a cada algoritmo estudiado bajo las condiciones dadas con el fin de llegar a los objetivos establecidos en una primera instancia.

Capítulo 2

Técnicas de restauración y ocultamiento de errores en imágenes

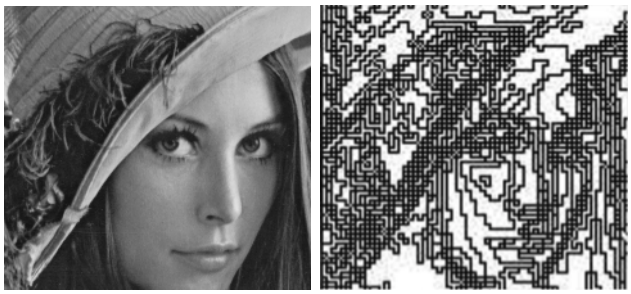
A través de los años, las tecnologías que se han desarrollado, permiten hoy en día un sin fin de aplicaciones prácticas y de bajo costo. Uno de los grandes avances tecnológicos, fue la creación de sistemas que actualmente permiten una comunicación sin enlaces guiados, como radios, televisores y telefonía móvil; sistemas que permiten la transferencia de contenido visual, como imágenes y videos.

Para poder transmitir la información de manera rápida y a bajo costo, se han desarrollado distintos algoritmos de compresión que permiten transferir la información haciendo uso eficiente de la memoria y también de la energía [54] [38], sin embargo, la perdida de paquetes de datos es un problema recurrente en este tipo de comunicación, y ocurre "cada vez que uno o más paquetes de datos se pierden en el camino", antes de llegar al nodo de destino [29]. En el caso de la transmisión de imágenes y videos, se han desarrollado algoritmos que permiten reconstruir áreas dañadas debido a la perdida de información en la transmisión o a la compresión. Estas técnicas son llamadas técnicas de restauración, o en ingles *inpainting* [33] y tienen como finalidad emular las restauraciones realizadas por artistas a pinturas y obras de arte pero, en este caso, de manera digital. También pueden ser utilizadas para rellenar pixeles o áreas de imágenes que se han visto afectadas por algún tipo de proceso [52]. Para ejemplificar, al momento de remover un objeto mediante edición, se genera un vacío que debe ser rellenado, es aquí en donde se aplican las técnicas de inpaiting [1]. También para eliminar texto, para descubrir cierta parte de una foto o imagen solapada por alguna mancha o daño [1], para hacer desaparecer un objeto [58] ó en la transmisión de videos con compresión. También son muy utilizados aquellos algoritmos que permiten ocultar errores de color o

profundidad debido a la perdida de información [22], sobre todo en transmisiones realizadas mediante redes móbiles como 3G o WiMAX, que permiten transmisión en streaming y están constantemente enfrentadas a perdidas de paquetes de datos [28].

2.1. *Inpainting*

Como se mencionó anteriormente, el *inpainting* se creo con la finalidad de restaurar imágenes de manera digital o recuperar una parte deteriorada. Para lograr ello es que diferentes algoritmos de restauración de errores en imágenes se han desarrollado a través de los años, los cuales permiten reconstruir imágenes de distintas formas, por ejemplo en 1995, F. Guichard y J.M. Morel, realizaron una investigación, que permitió demostrar la importancia que tiene el nivel de las líneas en una imagen tanto para poder representarlas, como también para poder entenderlas, y se definen como el límite de un conjunto de líneas, las cuales diferenciadas de todas las demás, permiten reconstruir un mapa topográfico [12], como el que se observa en la Figura 2.1, en donde se aprecia la representación en niveles de líneas de la imagen lena



(a) (b)

Figura 2.1: Imagen original a escala de grises (a) y su representacion en niveles de línea (b).

Haciendo uso de la detección de lineas, es que Simon Masnou y compañía lograron desarrollar un algoritmo llamado *level lines disocclusion*, que permite utilizar las líneas, como patrón para la reconstrucción de áreas y/o pixeles perdidos, como también para el mejoramiento de imágenes de mala calidad y dañadas [25]. Investigaciones y pruebas realizadas posteriormente por el mismo Simon Masnou [24] concluyeron que este tipo de algoritmos tienen desempeño deseable en cuanto a la recuperación de información de imágenes con ruido de hasta un 70 %.

Otro algoritmo de *inpainting* llamado en ingles *total variation model*, intenta imitar la percepción humana, restaurando un área de información perdida a partir de los pixeles que le rodean. Para ello utiliza una serie de ecuaciones no lineales que recorren la imagen y van rellenando el área de información perdida [4]. El proceso se realiza pixel a pixel y se necesita de varias iteraciones para lograr restaurar una imagen. Según la literatura, en áreas pequeñas ofrece un buen desempeño, pero al momento de recuperar bordes no entrega las prestaciones deseadas [21],[32].

Curvature-Driven Diffusion inpainting (CDD) en inglés, es otro tipo de algoritmo de recuperación de regiones dañadas que se presenta en la investigación [20], Este método no agrega ruido a la imagen resultante luego del proceso de reconstrucción como lo hacen otros algoritmos, los que no siempre producen resultados satisfactorios [32] o no están diseñados para trabajar con estructuras de imágenes que contienen muchos bordes y lineas [7]. Suli Li y Huiqin Wang, aseguran que CDD extiende el algoritmo de TV, siendo capaz de recuperar información de grandes áreas perdida, además de la información geométrica [21] ya que realizan el proceso de restauración de la misma forma, con la diferencia de que el CDD agrega un coeficiente de difusión en sus ecuaciones no lineales, este coeficiente provoca que la restauración se realice dependiendo de la intensidad de los pixeles que rodean al área restaurada.

Siguiendo con el análisis de las distintas técnicas de reconstrucción, en 2001 Andy Tsai, utilizó el algoritmo de evolución de curva de Mumford-Shah para poder segmentar y suavizar imágenes [43], con el fin de recuperar la calidad y eliminar el ruido en áreas grandes o también para la recuperación de información específica del contorno. Basado en este trabajo, es que se desarrollaron dos algoritmos de ocultación de errores [8], debido a que de por si el trabajo realizado en [43] no permite la recuperación de información perdida, sólo segmentación. En [8], se agrega a las ecuaciones presentadas en la investigación [43] un modelo de curva elástica. Esto permite no sólo segmentar, si no que también recuperar información perdida de manera muy precisa; inclusive la pérdida de información en un área grande, permitiendo recuperar las texturas de un objeto sobre una imagen. El proceso se basa en utilizar la información aledaña a el área a restaurar con el fin de analizar la textura de la imagen y reconstruir la imagen mediante una serie de iteraciones. La literatura demuestra la robustez de este algoritmo de recuperación en aplicaciones de restauración de imágenes, específicamente el color de estas [15]. También ha demostrado que puede servir en la medicina a la hora de remover ruido, reconectar y reconstruir la información dentro de una imagen con información orgánica [10].

2.2. *Error Concealment*

La ocultación de errores (Error Concealment) es una técnica utilizada en el procesamiento de señales que tiene como objetivo minimizar el deterioro de las señales, debido a datos faltantes, producido por la pérdida de paquetes, y a diferencia del *inpainting* el cual se diseñó para restaurar imágenes recorriéndolas completamente mediante varias iteraciones, el *error concealment* se dirige directamente al lugar de la imagen donde falta información y la restaura a partir de los pixeles que le rodean. La pérdida de paquetes se produce cuando estos están mal dirigidos, con retraso o están dañados.

Uno de los primeros algoritmos aplicados para interpolar áreas perdidas es el Bilineal, este se basa en el promedio simple de los 4 puntos que rodean al pixel en cuestión. Una de las grandes ventajas de este algoritmo es su simplicidad, bajo tiempo de procesamiento y el requerimiento de hardware, que en equipos de bajos recursos o en aplicaciones de tiempo real son factores importantes [44],[40].

La interpolación Bicúbica a diferencia de la interpolación Bilineal, utiliza los 8 pixeles para generar una superficie cubica, consiguiendo una reconstrucción suave a costa de mayor procesamiento de lineas de código, debido a la cantidad de ecuaciones, como se menciona en el articulo [14]. En este tipo de algoritmos, se genera un perímetro de los pixeles que rodean al área a interpolar, tomando en cuenta el direccionamiento horizontal y vertical del perímetro generado. Con la finalidad de obtener información detallada para reconstruir el área perdida, es que Sheila S. desarrolló un algoritmo bicúbico específicamente para la reconstrucción de paquetes de datos en la transmisión de imágenes, el cual interpola los pixeles mas cercanos haciendo uso de derivadas parciales de primer y segundo orden [14],[39].

Otro método de ocultamiento de errores es la reconstrucción mediante *Maximally Smooth Recovery* (MSR), dirigida a regiones de pixeles perdidos. El principio de funcionamiento es dividir bloque con información perdida en cuatro cuadrantes como se observa en la Figura 2.2, para luego comenzar una reconstrucción direccionada a cada uno de los cuatro puntos de la esquina del bloque, como se aprecia en la Figura 2.3. De esta manera cada uno de los pixeles reconstruidos de cada cuadrante, es dependiente de los pixeles conocidos más cercanos que le rodean.

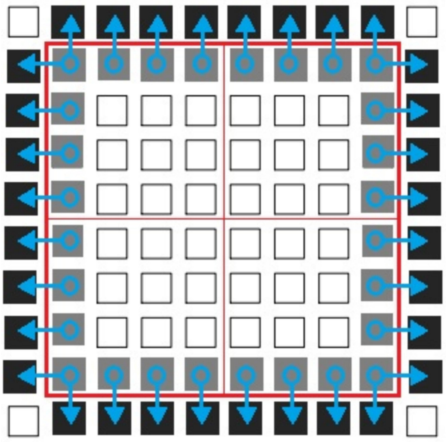


Figura 2.2: Bloque de pixeles perdidos, dividido en cuatro cuadrantes.

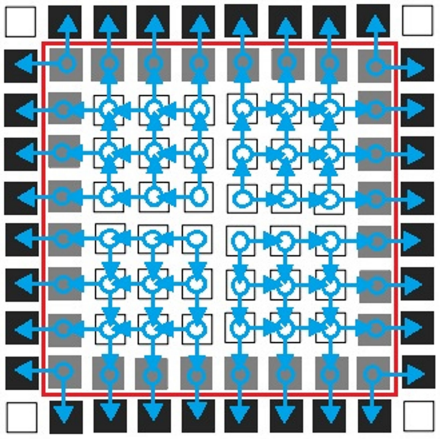


Figura 2.3: Reconstrucción direccionada a las esquinas del bloque por cuadrante.

Este algoritmo fue presentado en un principio por Yao Wang [47], con el fin de reconstruir imágenes transmitidas en redes de datos del tipo asíncronas y que han sido afectadas por pérdidas de paquetes de datos, dando como resultado bloques reconstruidos suavemente y dependientes de los pixeles más cercanos a cada uno de los cuadrantes.

En 2004, Katrin Meisinger y Andre Kaup [16], desarrollaron en su investigación un algoritmo para el ocultamiento de error en imágenes mediante extrapolación de frecuencia selectiva (FSE). Aquí utilizan un modelo isotropico como función de ponderación para correlacionar los pixeles cercanos al área faltante.

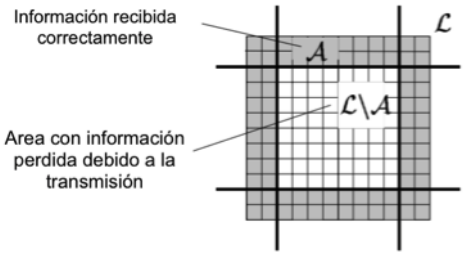


Figura 2.4: Ejemplificación de perdida de información y su vecindad.

El área conocida A de la Figura 2.4 es aproximada por un cierto modelo paramétrico y el área perdida se obtiene a partir de extrapolación [17].

La información recibida correctamente, es manipulada mediante la transformada de fourier discreta (DFT), con lo cual se obtiene un cierto modelo paramétrico. El modelo paramétrico es manipulado de tal manera que se llega a un conjunto de ecuaciones lineales que no tiene una única solución, ya que depende de iteraciones y en cada iteración el resultado cambia. Para obtener una solución, se aplica el principio de aproximación sucesiva, este algoritmo iterativo, calcula un coeficiente por cada iteración que realiza, por lo que el contenido de la imagen puede ser descrita como "una combinación lineal ponderada de unas pocas funciones seleccionadas"[17].

Este algoritmo permite extrapolar grandes áreas o bordes con información perdida, las áreas rodeadas por pixeles con tonalidades parecidas requieren solo una iteración, por otra parte las detalladas y con textura requiere muchas más [17]. Diversas investigaciones probaron este algoritmo para ser usado en transmisiones y comunicación concluyendo que tiene un gran desempeño en la recuperación de la calidad de imágenes dañadas en transferencias inalámbricas, debido a la adaptabilidad [34],[18].

Transmisión de imágenes y arquitectura de redes inalámbricas

Una red de sensores inalámbrica es un conjunto de equipos sensores de detección y nodos, capaz de realizar comunicación de manera inalámbrica entre ellos, permitiendo una gran flexibilidad de trabajo [37]. En un principio se considero caro y poco desarrollado para usarlo de manera robusta, pero poco a poco se han desarrollado diversas aplicaciones aprovechando las características de bajo consumo que entregan y el desempeño en lugares en donde el cableado es un problema debido al entorno de trabajo [19], como por ejemplo en agricultura de precisión [46], túneles viales [26], monitoreo de la calidad ambiental [53], detección de fuego [31] y monitoreo de maquinas y estructuras entre otras [13].

Dentro de este campo se han desarrollado variadas investigaciones para la transmisión de imágenes, muchas se enfocan en algoritmos de compresión para realizar transmisiones energéticamente eficientes, lo que permite un aumento en el tiempo de trabajo de un sistema autónomo [51], otras investigaciones hacen énfasis en disminuir el costo computacional [23] ó asegurar una reconstrucción de calidad de las imágenes transmitidas [42]. Otros artículos hacen hincapié en el proceso completo de compresión, transmisión y descompresión de la imagen en sensores de redes inalámbricas, aplicando algoritmos de *inpainting* para la recuperación de la información perdida [27]. A pesar de eso, son muchos investigadores los que dejan abiertas futuras investigaciones sin tener conclusiones certeras.

Sim-LIT

Sim-LIT es una herramienta desarrollada por Christopher Arredondo en 2015 para la simulación de la perdida de paquetes de transmisión inalámbrica de sensores. Este *Framework* dedicado a la simulación [2], permite configurar la perdida de datos en el proceso de transmisión, también ofrece algoritmos de reconstrucción, simulación de errores y una métrica para la calidad de la reconstrucción [11].

En la imagen 2.5 se aprecia parte del diagrama de clases de Sim-LIT. En el diagrama se muestran las clases que son necesarias para agregar los nuevos métodos, para ejemplificar, se observa la extension que se realiza a la clase *Rebuilder* y el manejo de las clases *images*, *pixel* y *DataType*, y las que en general se manejan para realizar las simulaciones.

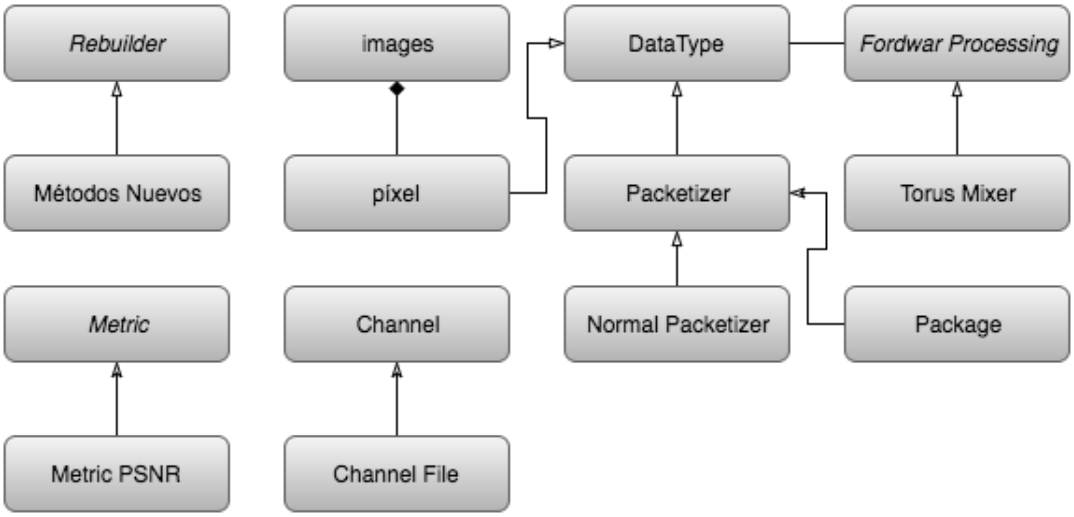


Figura 2.5: Diagrama de bloques de arquitectura de interes de Sim-LIT para agregar nuevos métodos de restauración de imágenes.

Algoritmos de cuantificación de calidad

Para cuantificar la calidad de una imagen que ha sido modificada con respecto a su par original, se puede encontrar una gran cantidad de alternativas, en la literatura observamos que uno de los indicadores más conocidos, es la relación señal-ruido (o en ingles *Peak Signal-to-Noise Ratio*) comúnmente conocido como PSNR, ya que entrega una medida de calidad basado en el error cuadrático entre la imagen original y la procesada, además tiene la ventaja de su simplicidad matemática [45]. Ha sido utilizado para calcular la calidad en investigaciones de compresión de imágenes y procesamiento de video, sin embargo la representatividad de la calidad visual que entregan es muy pobre [6]. Es por esto que se han desarrollado una gran cantidad de indicadores con el fin de determinar no solo la calidad de una imagen modificada a nivel de intensidad, si no que también el impacto visual que este proceso ha generado, siendo capaces de imitar la percepción humana para estimar la calidad de la imagen modificada, ejemplo de ello son el FSIM [56], IFC [35], IWSSIM [49], PIQA [9], SR-SIM [55], SSIM [48][50], VIF [36][57] y VSRN, este ultimo se caracteriza por el poco uso de memoria [6].

Cada uno de estos distintos indicadores tiene sus pro y contras, y la selección de alguno para la estimación de la calidad de imágenes depende de factores como recursos y área en que se utilizará el algoritmo.

Justificación del estudio y de los algoritmos de disimulación de errores

Según el estado del arte realizado, no hay en la literatura ninguna investigación que haya comparado algoritmos de *inpainting* y *error concealment* para la transmisión en nodos de redes inalámbricas, por lo que se justifica este trabajo de titulo.

De las muchas técnicas de ocultamiento de error e *inpainting* que existen, anteriormente se han expuesto algunas, debido a su relevancia en la historia de las investigaciones o por ser de las mas reconocidas y utilizadas en la literatura, cada una de estas técnicas presenta una ventaja o una característica distinta con respecto a otra, algunas se destacan por la velocidad de reconstrucción, otras por el poco hardware que utilizan o por la simplicidad de las ecuaciones. En el presente trabajo se pretende realizar una comparación entre algoritmos de *inpainting* y *error concealment*, para ser utilizado sobre redes de sensores inalambricos.

Para este trabajo se han seleccionado algoritmos de TV y CDD en el caso del *inpainting* y los de BIC, BI y MSR en el caso de los algoritmos de *error concealment* para realizar las pruebas por los siguientes motivos:

Disponibilidad de hardware: Todos los algoritmos pueden ser compilados y estudiados con el equipo con el que se cuenta, un ordenador con una unidad de procesamiento y el conocimiento para ejecutarlos.

Baja complejidad de programación: En sí, los algoritmos son de baja complejidad matemática, esto permite agilizar el proceso de homologación de las ecuaciones matemáticas a lenguaje de programación.

Velocidad de ejecución: Según la literatura, son algoritmos con una velocidad de ejecución alta, que los hace apetecibles para aplicaciones que trabajen en tiempo real.

Investigaciones: Finalmente, son algoritmos que han sido estudiados durante años por distintos investigadores, los que han desarrollado distintas aplicaciones y mejoras, lo que permite tener una base solida para que sean incluidos en este trabajo de título.

Para obtener el indice de calidad se ha seleccionado el indicador PSNR, debido a que muchos artículos e investigaciones lo han utilizado y lo siguen utilizado. Su ecuación se observa a continuación.

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \tag{2.1}$$

En donde MSE es el error cuadrático medio que se observa en la ecuación 2.2, para dos imágenes monocromas I y K de tamaños MxN (para imágenes en RGB el calculo se realiza para cada banda). MAX_I es el valor máximo que puede tomar un píxel, en el caso de las imágenes en escala de grises el valor máximo es 255.

$$MSE = \frac{1}{MN} \cdot \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \|I_{(i,j)} - K_{(i,j)}\|^2 \tag{2.2}$$

Capítulo 3

Técnicas y algoritmos seleccionados

3.1. TV *inpainting*

En la literatura existen muchos métodos para la realización del modelo matemático del TV y modelos minimizados para la generación de un algoritmo en lenguaje de programación, en este trabajo, se utilizara el modelo matemático presentado por Tony F. Chan en su investigación [5] y que se explica y desglosa en [4]. Basándose en las ecuaciones no lineales de *Euler-Lagrange*, se puede adaptar un modelo para el TV *inpainting* en el dominio de los pixeles, quedando como se muestra a continuación.

$$TV_{i,j} = \frac{w_o * u_{(i-1,j)} + w_e * u_{(i+1,j)} + w_s * u_{(i,j+1)} + w_n * u_{(i,j-1)}}{w_o + w_e + w_s + w_n} \tag{3.1}$$

Para entenderlo se muestra en la Figura 3.1 un plano cardinal en donde $u_{i,j}$ es el centro de coordenadas y en cuestión el pixel a interpolar.

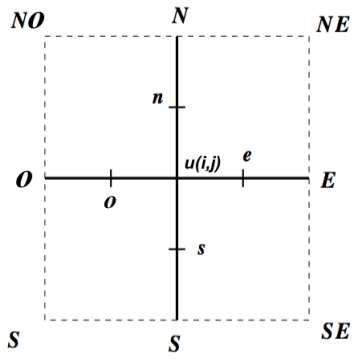


Figura 3.1: Mapa de coordenadas cardinales en donde $u_{i,j}$ es el pixel a restaurar.

De la ecuación 3.1 se tiene w, que son ponderaciones que varían según la posición u del pixel a restaurar, así se le da peso a los pixeles que estén mas cercanos y se logra una restauración optima. Las ponderaciones se calculan de la siguiente forma:

$$w_o = \frac{1}{\sqrt{|\nabla u_o| + \epsilon}} \tag{3.2}$$

$$w_e = \frac{1}{\sqrt{|\nabla u_e| + \epsilon}} \tag{3.3}$$

$$w_s = \frac{1}{\sqrt{|\nabla u_s| + \epsilon}} \tag{3.4}$$

$$w_n = \frac{1}{\sqrt{|\nabla u_n| + \epsilon}} \tag{3.5}$$

Aquí el valor absoluto del operador direccional nabla $|\nabla u|$ es distinto para cada ponderación y depende, como se explico anteriormente de la posición del pixel que se esta utilizando para el calculo, con respecto al pixel que se quiere interpolar, por otro lado ϵ es una constante para evitar indeterminación en la división. Finalmente los valores de $|\nabla u_n|$, $|\nabla u_s|$, $|\nabla u_e|$ y $|\nabla u_o|$ se calculan como se muestra a continuación:

$$|\nabla u_o| = \left(u_{(i,j)} - u_{(i-1,j)}\right)^2 - \left(\frac{\left(u_{(i-1,j-1)} - u_{(i-1,j+1)}\right)}{2}\right)^2 \tag{3.6}$$

$$|\nabla u_e| = \left(u_{(i,j)} - u_{(i+1,j)}\right)^2 - \left(\frac{\left(u_{(i+1,j-1)} - u_{(i+1,j+1)}\right)}{2}\right)^2 \tag{3.7}$$

$$|\nabla u_s| = \left(u_{(i,j)} - u_{(i,j-1)}\right)^2 - \left(\frac{\left(u_{(i-1,j-1)} - u_{(i+1,j-1)}\right)}{2}\right)^2 \tag{3.8}$$

$$|\nabla u_n| = \left(u_{(i,j)} - u_{(i,j+1)}\right)^2 + \left(\frac{\left(u_{(i-1,j+1)} - u_{(i+1,j+1)}\right)}{2}\right)^2 \tag{3.9}$$

Con esto se tiene la implementación numérica completa del TV para ser aplicada a lenguaje de programación.

3.2. CDD *inpainting*

Como se mencionó en el apartado 2.1, el CDD se crea a partir del TV. Tomando el modelo prepuesto por Tony F. Chan en [3] y la implementación numérica en [4] tenemos la siguiente ecuación que gobierna el CDD *inpainting* en el dominio de los pixeles, en donde k es el escalar de curvatura:

$$CDD_{i,j} = \frac{w_o * u_{(i-1,j)} + w_e * u_{(i+1,j)} + w_s * u_{(i,j+1)} + w_n * u_{(i,j-1)}}{w_o + w_e + w_s + w_n} \tag{3.10}$$

En donde al igual que en la sección anterior utilizaremos las coordenadas de la Figura 3.1 para entender el significado las ecuaciones, en este caso las ponderaciones. Cuando la función esté realizando el *inpainting*, k se encargará de reconstruir el diseño y unir lineas dentro de la imagen a restaurar, como se explica en el articulo citado [3]. Se observa que la ecuación tiene la misma estructura que el algoritmo de TV, y que el escalar de curvatura es en si el mismo modelo presentado en la ecuación para el TV. w dependen del coeficiente k como se muestra a continuación:

$$w_o = \frac{k}{\sqrt{|\nabla u_o| + \epsilon}} \tag{3.11}$$

$$w_e = \frac{k}{\sqrt{|\nabla u_e| + \epsilon}} \tag{3.12}$$

$$w_s = \frac{k}{\sqrt{|\nabla u_s| + \epsilon}} \tag{3.13}$$

$$w_n = \frac{k}{\sqrt{|\nabla u_n| + \epsilon}} \tag{3.14}$$

Se determina k realizando el mismo procedimiento que se vio en el TV *inpainting*, el cual es redundante y ϵ se utiliza para evitar indeterminaciones en la división. Llamaremos v las ponderaciones que se utilizan para determinar k por lo que nos queda:

$$k = \frac{v_o * u_{(i-1,j)} + v_e * u_{(i+1,j)} + v_s * u_{(i,j+1)} + v_n * u_{(i,j-1)}}{v_o + v_e + v_s + v_n} \tag{3.15}$$

Del mismo modo que en el método TV, las ecuaciones de determinan como sigue::

$$v_o = \frac{1}{\sqrt{|\nabla u_o| + \epsilon}} \tag{3.16}$$

$$v_e = \frac{1}{\sqrt{|\nabla u_e| + \epsilon}} \tag{3.17}$$

$$v_s = \frac{1}{\sqrt{|\nabla u_s| + \epsilon}} \tag{3.18}$$

$$v_n = \frac{1}{\sqrt{|\nabla u_n| + \epsilon}} \tag{3.19}$$

Donde ∇u se determina como se muestra:

$$|\nabla u_o| = \left(u_{(i,j)} - u_{(i-1,j)}\right)^2 - \left(\frac{\left(u_{(i-1,j-1)} - u_{(i-1,j+1)}\right)}{2}\right)^2 \tag{3.20}$$

$$|\nabla u_e| = \left(u_{(i,j)} - u_{(i+1,j)}\right)^2 - \left(\frac{\left(u_{(i+1,j-1)} - u_{(i+1,j+1)}\right)}{2}\right)^2 \tag{3.21}$$

$$|\nabla u_s| = \left(u_{(i,j)} - u_{(i,j-1)}\right)^2 - \left(\frac{\left(u_{(i-1,j-1)} - u_{(i+1,j-1)}\right)}{2}\right)^2 \tag{3.22}$$

$$|\nabla u_n| = \left(u_{(i,j)} - u_{(i,j+1)}\right)^2 + \left(\frac{\left(u_{(i-1,j+1)} - u_{(i+1,j+1)}\right)}{2}\right)^2 \tag{3.23}$$

Es así como se desglosa la ecuación 3.10 con la cual se realiza el CDD inpainting y sus ecuaciones pueden ser pasadas a lenguaje de programación sin mayor dificultad.

3.3. Interpolación bilineal (BI)

La interpolación es relativamente sencilla, en primer lugar llamaremos i y j a las filas y columnas respectivamente, en la Figura 3.2, observamos que el pixel a interpolar esta en blanco y los cuatro pixeles mas cercanos en color verde.

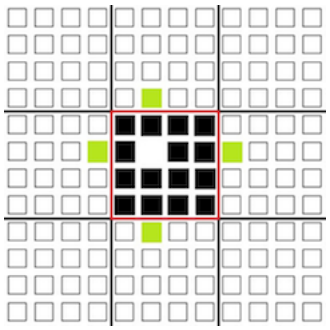


Figura 3.2: Principio de extrapolación bilineal con los 4 pixeles mas cercanos.

Así, el pixel a interpolar queda de la siguiente manera en función de (i, j):

$$Bili_{i,j} = \frac{u(i-1,j) + u(i+1,j) + u(i,j-1) + u(i,j+1)}{4} \tag{3.24}$$

Esta ecuación es bastante sencilla matemáticamente hablando para ser programada, y tiene la ventaja de no necesitar iteraciones.

3.4. Interpolación bicúbica (BIC)

Para realizar la interpolación Bicúbica, se utilizará el algoritmo presentado en la investigación [14], el cual fue diseñado para ser utilizado en escenarios de perdidas de paquetes de datos.

Lo primero es generar una cuadricula que rodee al pixel a reconstruir y se utilizan los puntos de cada esquina para formar una superficie cubica, también se utilizan los gradientes correspondientes entre cada punto $\partial f/\partial x$ y $\partial f/\partial y$, por ultimo también se hace uso de la respectiva derivada cruzada, siendo x e y utilizados para representar los cálculos en las filas y columnas respectivamente.

Para un punto (i, j) de una imagen digital se tendrían las siguientes ecuaciones no lineales que modelan al BIC:

$$f(x,y) = \left[2 \cdot f_{(i-1,)} + 2 \cdot f_{(i+1,)} + 2 \cdot f_{(,j+1)} + 2 \cdot f_{(,j-1)}\right]/8 \tag{3.25}$$

$$\partial f(x) = \left[\partial f/\partial x|_{(i-1,j-1)} - \partial f/\partial x|_{(i+1,j-1)} + \partial f/\partial x|_{(i-1,j+1)} - \partial f/\partial x|_{(i+1,j+1)}\right] \tag{3.26}$$

$$\partial f(y) = \left[\partial f/\partial y|_{(i-1,j-1)} + \partial f/\partial y|_{(i+1,j-1)} - \partial f/\partial y|_{(i-1,j+1)} - \partial f/\partial y|_{(i+1,j+1)}\right] \tag{3.27}$$

$$\partial^2 f(x,y) = \left[\partial^2 f/\partial x\partial y|_{(i-1,j-1)} - \partial^2 f/\partial x\partial y|_{(i+1,j-1)} - \partial^2 f/\partial x\partial y|_{(i-1,j+1)} + \partial^2 f/\partial x\partial y|_{(i+1,j+1)}\right] \tag{3.28}$$

De manera que el coeficiente de reconstrucción queda expresado de la siguiente forma:

$$Bic_{i,j} = \left[\frac{1}{4} \cdot f(x,y) + \frac{1}{16} \cdot \partial f(x) + \frac{1}{16} \cdot \partial f(y) + \frac{1}{64} \cdot \partial^2 f(x,y)\right] \tag{3.29}$$

3.5. Reconstrucción mediante MSR

Este algoritmo utiliza solo ecuaciones de primer orden, en primer lugar se debe dividir el bloque a reconstruir en cuatro cuadrantes, para ello se asume que el bloque siempre será cuadrado y de tamaño W, por lo tanto hay que desarrollar cuatro ecuaciones que recorran de manera distinta el bloque dependiendo en que cuadrante se encuentren.

Se asume que se tienen bloques de datos U(i,j), entonces para cada borde del bloque perdido (según la figura 2.2 del capítulo anterior) y en cada posición específica de ese borde,

se reemplaza por el valor del pixel mas cercano, así se genera un perímetro con valores de pixeles conocidos dentro del bloque. A partir de ahí, se realizan ciclos que van dependiendo de que cuadrante se este trabajando.

Para el primer cuadrante localizado horizontalmente entre 0 y $(W/2)-1$, y verticalmente entre 0 y $(W/2)-1$ entonces se tiene el siguiente conjunto de ecuaciones:

$$f_{(i,j)} = \begin{cases} i > 0; j > 0, & u_{i,j} = (f(i-1,j) + f(i,j-1))/2 \\ i = 0, & u_{i,j} = f(i-1,j) \\ j = 0, & u_{i,j} = f(i,j-1) \end{cases} \tag{3.30}$$

El segundo cuadrante esta posicionado desde $(W/2)$ hasta $W-1$ en el eje horizontal y de 0 hasta $(W/2)-1$ en el eje vertical, quedando las ecuaciones de la siguiente forma:

$$f_{(i,j)} = \begin{cases} i > 0; j < (W-1), & u_{i,j} = (f(i-1,j) + f(i,j+1))/2 \\ i = 0, & u_{i,j} = f(i-1,j) \\ j = W-1, & u_{i,j} = f(i,j+1) \end{cases} \tag{3.31}$$

El tercer cuadrante esta posicionado desde 0 a $(W/2)-1$ en el eje horizontal y de $(W/2)$ hasta $(W-1)$ en el eje vertical, así las ecuaciones quedan de la siguiente manera:

$$f_{(i,j)} = \begin{cases} i < (W-1); j > 0, & u_{i,j} = (f(i+1,j) + f(i,j-1))/2 \\ i = W-1, & u_{i,j} = f(i+1,j) \\ j = 0, & u_{i,j} = f(i,j-1) \end{cases} \tag{3.32}$$

Finalmente el cuarto cuadrante que se posiciona desde $(W/2)$ hasta $(W-1)$ en el eje horizontal y vertical, es reconstruido con las siguientes ecuaciones:

$$f_{(i,j)} = \begin{cases} i < (W-1); j < (W-1), & u_{i,j} = (f(i+1,j) + f(i,j+1))/2 \\ i = W-1, & u_{i,j} = f(i+1,j) \\ j = W-1, & u_{i,j} = f(i,j+1) \end{cases} \tag{3.33}$$

Así, con estas cuatro ecuaciones para cada cuadrante se recupera el bloque perdido como en la Figura 2.3 del capitulo anterior, siendo $u_{i,j}$ el valor del pixel recuperado.

Capítulo 4

Experimentos y desarrollo

4.1. Requisitos

Los requisitos para realizar este estudio son:

Un ordenador con una unidad de procesamiento capaz de procesar los algoritmos y también de ejecutar el entorno de simulación Sim-LIT, para ello es necesario sistema operativo Mac OS o Linux, ya que deben ser ejecutados en un sistema UNIX. Se deben tener las ecuaciones que rigen cada algoritmo y posteriormente su par en lenguaje de programación.

4.2. Implementación de algoritmos

Para implementar los algoritmos en Sim-LIT, hay que seguir un cierto orden. En primer lugar se deben homologar todas las ecuaciones vistas anteriormente en el capítulo 2, a lenguaje de programación C++. Tomando esto en cuenta se presentan los pasos para agregar una nueva clase Rebuilder a Sim-LIT.

4.2.1. Arquitectura Sim-LIT

En la carpeta *class*, están todas las clases que se utilizan para su funcionamiento, dentro de esta carpeta hay otra llamada *Rebuilder*, que contiene la clase Abstracta `Rebuilder.h` y las clases de ocultamiento de errores, encargada reconstruir los pixeles perdidos de una imagen, a partir de los bloques recibidos mas cercanos.

Todo tipo de método de ocultamiento de errores que se implemente debe ser extendido de la clase abstracta `Rebuilder.h`, ésta clase padre, define un único método que debe ser implementado por las clases herederas como se muestra en las siguientes lineas de código.

```
#indef CLASS_REBUILDER
#define CLASS_REBUILDER

#include "../Images.h"
#include "../DataBlock.h"
class Rebuilder{
public:
      virtual void hidden(Images* img, bool show_data, bool export_images,
         HEAD *header)=0;
      };
      #endif
```

La cabecera de imagen es una estructura que contiene el tamaño de la imagen original (width-height), el tamaño de cada bloque de la imagen (widthBlock-HeightBlock), la cantidad de canales de la imagen (1:GrayScale, 3:RGB) y la carpeta donde estarán los resultados de la simulación (folder).

Para crear la instancia de una clase, se utiliza el patrón de fábrica, en la carpeta creators están las fábricas para las clases que trabajan en la herramienta, en el caso del ocultamiento de errores, se tiene la clase RebuilderCreator. En la función RebuilderCreator, se ingresa un string, y lo compara con uno de los argumentos que se encuentran en los Define del archivo `arguments.h` de las siguientes lineas de código.

```
#define REBUILDER_METHOD "-rebuilder-method"
      #define AVERAGE_RBD "rebuilder-average"
      #define AVERAGE_PIXEL_RBD "rebuilder-average-pixel"
      #define AVERAGE_OPT_RBD "rebuilder-average-opt"
      #define AVERAGE_RAMANBABU2001 "rebuilder-ramanbabu2001"
      #define AVERAGE_TV "rebuilder-tv"
      #define AVERAGE_CDD "rebuilder-cdd"
      #define AVERAGE_BILI "rebuilder-bili"
      #define AVERAGE_TVBLOCK "rebuilder-tvblock"
      #define AVERAGE_MSR "rebuilder-msr"
```

4.2.2. Resumen de implementación de algoritmos

A continuación se muestra un resumen de la implementación de los algoritmos presentados en capítulo 3, en el entorno de simulación Sim-LIT.

1. Discretizar los algoritmos de inpainting y error concealment.
2. Homologar los algoritmos a lenguaje de programación en C++.
3. Crear la clase en la carpeta ..class/Rebuilder que extienda de Rebuilder.
4. Agregar el método a la fábrica.
5. Agregar en arguments.h el comando que definirá al método.

4.2.3. Métodos implementados

En el apartado anterior se han explicado las técnicas que serán utilizadas como parte fundamental para dar cumplimiento con los objetivos de este estudio, para poder utilizar éstas es necesario llevar sus ecuaciones matemáticas al lenguaje de programación C++. En total se agregaron cinco técnicas para restaurar imágenes en la directorio ../class/Rebuilder/.

1. RebuilderTV.h
2. RebuilderCDD.h
3. RebuilderBilineal.h
4. RebuilderBicubico.h
5. RebuilderMSR.h

Los algoritmos de cada una de estas técnicas en lenguaje de programación C++ se pueden ver en el anexo B.

4.3. Ejecución de Sim-LIT

4.3.1. Comandos de consola

Para ejecutar los algoritmos y tener acceso a las funciones y herramientas de Sim-LIT, hay que ingresar ciertos comandos mediante consola, los cuales se muestran a continuación.

- **./simlit:** Ejecutable.
- **-image-name:** comando que indica que a continuación viene la ubicación y el nombre la imagen de la imagen.
- **-packetizer-method:** Indica que viene el proceso de paquetizado, primero se lee la cantidad de bytes de cada paquete (es un byte por canal de cada pixel de la imagen), luego, se especifica el tipo de paquetizado, seguido de la cantidad de argumentos y los argumentos que necesita el paquetizado para operar.
- **-rebuilder-metod:** Este comando es muy importante ya que especifica que método de ocultamiento de errores será utilizado para tratar los paquetes de información perdida de la imagen.
- **-metric:** Indica las métricas que se aplicaran a la imagen, primero la cantidad y luego se enumeran las métricas definidas en arguments.h
- **-export-result:** Este comando guarda los resultados obtenidos en la simulación en un archivo de texto, al cual se le debe especificar el nombre.
- **-folder:** Se especifica la carpeta donde se guardarán los resultados.
- **-show-data-screen:** Indica que se mostrarán todos los datos posibles por pantalla, para los objetivos de nuestra tesis es irrelevante.
- **-export-images:** Indica si se exportarán imágenes durante todo el proceso de simulación.
- **-width-block:** Especifica el ancho de un bloque de pixeles.
- **-height-block:** Especifica el alto de un bloque de pixeles, junto con el comando anterior son muy importantes para realizar el proceso de error concealment.

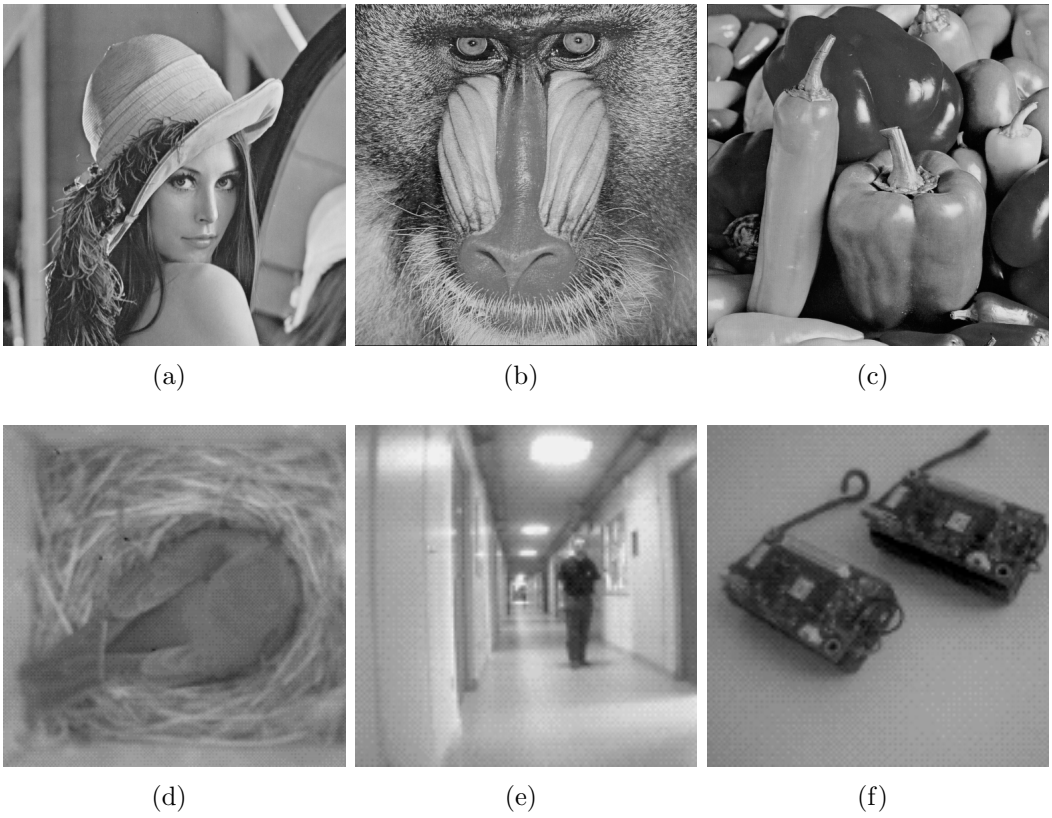
Capítulo 5

Simulación de algoritmos de disimulación de las imágenes

5.1. Parámetros y base de datos

Los parámetros con los que se trabajará serán los siguientes; Se utilizará tres tamaños de bloques distintos, 1x1, 2x2 y 8x8 en la simulación de la transmisión, además se utilizará la técnica de entrelazado *Torus Automorphisms*. Se realizarán simulaciones con pérdidas de datos de manera aleatoria que variarán de 10 % hasta 80 %, todo esto basándose en la investigación de perdida de paquetes de datos [30]. Las transmisiones se realizarán con paquetes de datos de 27 y 100 bytes por cada método de ocultación de errores y porcentaje de pérdida, dando como resultado un total de 240.000 simulaciones a realizar.

Se utilizará un set de seis imágenes de tres tamaños distintos, las que se observan en la Figura 5.1, de las cuales, algunas han sido utilizadas en diversas investigaciones.



(a) (b) (c)

(d) (e) (f)

Figura 5.1: Imágenes de Prueba: Lena 512x512 (a) Baboon 512x512 (b) Peppers 512x512 (c) Bird 200x200 (d) Corridor 128x128 (e) y motes 128x128 (f)

Para resultados representativo, se realizarán 200 simulaciones por cada porcentaje de error, que tendrá una ventana de 10 entre porcentaje, de modo que se realizarán pruebas con 10 %, 20 %, 30 %, 40 %, 50 %, 60 %, 70 % y 80 % de pérdida de paquetes de datos, realizar pruebas con paquetes perdidos bajo al 10 % o sobre el 80 % no tendría demasiada relevancia según [30], esto debido que a pérdidas menores, el resultado para los métodos de reconstrucción seria el mismo, por otro lado sobre el 80 %, la imagen prácticamente se pierde.

Ejecución de simulación

Teniendo claro los comandos anteriores, sus funciones y como utilizarlos, se procede a continuación a ejemplificar una sucesión de líneas de código para simular la transmisión de una imagen en bloques de 1x1 con paquetes de datos de 100 bytes, una perdida de paquetes del 20 % y reconstrucción mediante el método TV (Figura 5.2).

```
crixex — -bash — 80×24
Last login: Sun Oct  2 23:13:58 on ttys000
MacBookistopher:~ crixex$ ./simlit -image-name rrss/1.bmp -packetizer-method 100
 normal-packetizer -channel-method lost-from-file 1 files/loss_20_1.in -rebuilde
r-method rebuilder-tvblock -metric 1 metric-psnr -export-result resultados.txt -
folder resultados -width-block 1 -height-block 1
```

Figura 5.2: Ejemplificación de ejecución de comandos mediante consola para simulación en Sim-LIT

Luego de realizada las 240.000 simulaciones se obtuvieron los siguiente resultados, los cuales se ordenaran por imagen:

Los resultados que se obtuvieron para el TV, CDD y MSR en varios momentos de la simulación arrojaron restauraciones con valores cero debido a malas restauraciones, como hipótesis se tiene que el exceso de memoria ram que utilizan estos algoritmos provocan dichos resultados.

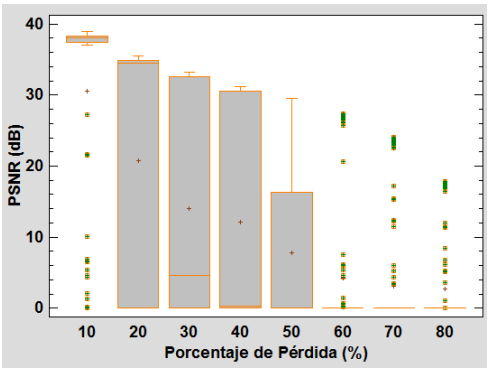
5.2. Imagen Nº1: Lena 512x512

A continuación, para la imagen Lena de la Figura 5.3 se tienen resultados para cada algoritmo de ocultamiento de error, paquete de dato y tamaño de bloque:

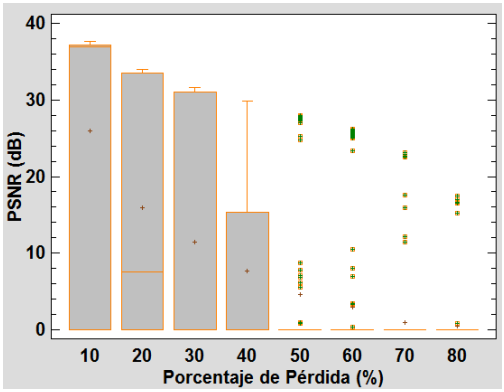
Figura 5.3: Lena 512x512.

Los resultados que se observan en la Figura 5.4 y Figura 5.6 para el TV y CDD *inpainting* respectivamente muestran una gran cantidad de errores en los ensayos al igual que en el MSR de la Figura 5.9 con bloques de 1x1.

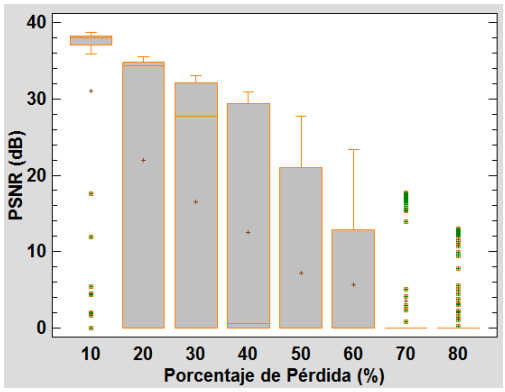
5.2.1. TV



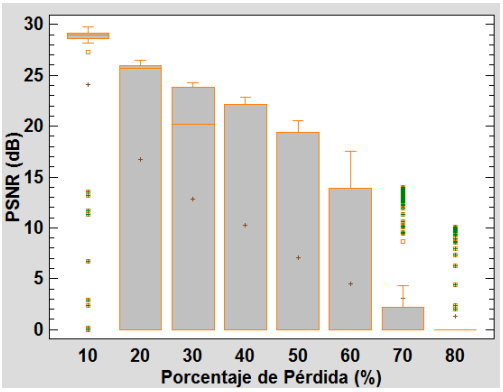
(a) block=1x1, byte TX=27



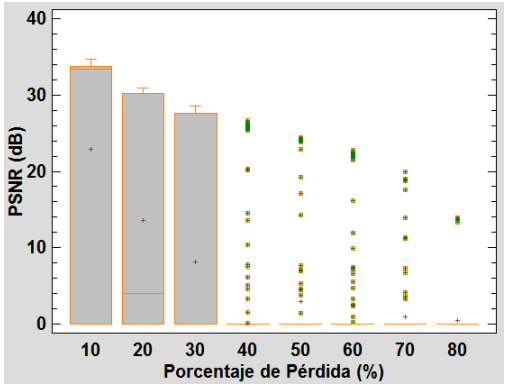
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.4: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de TV *inpainting*

En el diagrama 5.4a de bloques de 1x1 y paquetes de 27 bytes, se observa una alta dispersión de resultados, teniendo valores máximos entre 20dB y 40dB, y valores mínimos rondando los 0dB. Los valores de PSRN cercanos a cero, se deben a que el algoritmo de TV falló en la restauración de la imagen, entregando de resultado una reconstrucción como la que se observa en la Figura 5.5, en donde la imagen restaurada 5.5b está completamente en blanco con algunos cuadros negros. También se puede apreciar que en los únicos momentos en que la media se encuentra notoriamente cerca de los valores máximos de PSNR es cuando la pérdida de datos es de 10 %, por otro lado la mediana cae bruscamente entre el 50 % y el 60 %. Cuando la pérdida de paquetes de datos es de 10 %, la gran parte de los resultados se concentran cerca de los 40dB, con un 80 % de las pruebas realizadas con éxito, el 20 % restante son resultados con restauraciones fallidas como la de la Figura 5.5 (revisar tablas anexas A.1). Para un 20 % de pérdida, la dispersión aumenta, teniendo la mayor parte de sus resultados entre 30dB y 0dB. Desde el 30 % de pérdida de paquetes en adelante, la dispersión aumenta considerablemente en comparación a la dispersión entre 10 % y 20 % de pérdida, con una media prácticamente en 0dB, la cual se mantiene a la baja hasta llegar al 80 % de pérdida de paquetes de datos.

(a) Imagen con pérdida (b) Restauración fallida

Figura 5.5: Error en la restauración mediante TV

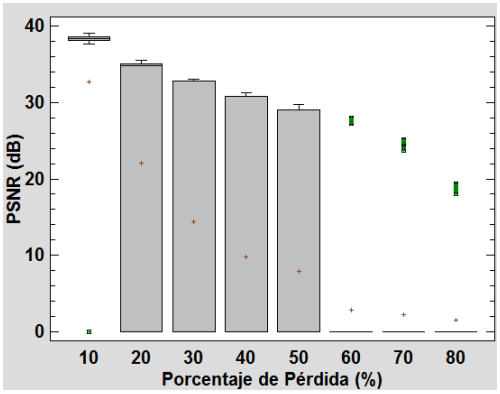
En el diagrama 5.4c con bloques de 1x1 pero en este caso paquetes de 100 bytes, son muy pocas las diferencias con los resultados arrojados en el diagrama 5.4a. Se observa que a diferencia de la transmisión mediante paquetes de 27 bytes, a 30 % de pérdida de datos la mediana esta mas cerca de los valores máximos, y que desde 40 % de pérdida de paquetes en adelante, esta cae bruscamente a valores de PSNR cercanos a los 0dB. La media tiene una tendencia muy similar a la del diagrama 5.4d y la dispersión de datos tiene una comportamiento similar.

Para transmisión en bloques de 2x2 y paquetes de datos de 27 bytes, se observa en el diagrama 5.4b la dispersión de datos es mayor que en los dos casos vistos anteriormente cuando la pérdida de datos es del 10 %. En 20 % de pérdida la dispersión aumenta bruscamente en comparación a los resultados para transmisiones en bloques de 1x1 y paquetes de 100 y 27 bytes. Después del 50 % de pérdida de paquetes, los resultados son prácticamente todos iguales, con valores cercanos a los 0dB, salvo algunos resultados que se mantienen entre 10dB y 40dB. En 70 % y 80 % de pérdida de paquetes de datos el porcentaje de pruebas exitosas en la restauración de imágenes fue de 4.5 % y 2.5 % respectivamente. La media solo se encuentra cerca de los valores máximos al 10 % de perdida, después de eso decae a valores por debajo de los 10dB de manera mas rápida que en los diagramas 5.4a y 5.4c, Por otro lado, los valores máximos se mantienen entre 20dB y 40dB.

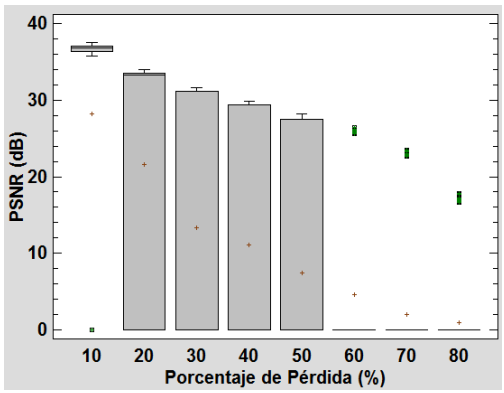
Los resultados para bloques de 2x2 y paquetes de 100 bytes del diagrama 5.4d, se muestran muy similares a los resultados en transmisión de bloques de 1x1. Comparando los resultados con los del diagrama 5.4b, se observa que el aumento del tamaño en el paquete de datos en bloques de 2x2, afecto los resultados. Con 100 bytes la dispersión es mucho menor a 10 % de pérdida y mayor en 60 % y 70 % en comparación a transmisión en bloques de 2x2 y 27 bytes. Con respecto a los demás resultados los puntos de la media se mantienen en promedio mas bajos, aproximadamente entre 1dB a 5dB.

La transmisión en bloques de 8x8 y 100 bytes del diagrama 5.4e, muestra que para porcentajes de pérdida sobre el 50 %, la mayor parte de los datos son cercanos a cero dB, lo que se refleja en una restauración fallida. Los valores máximos se observan, a medida que aumentan los porcentajes de pérdida, con valores entre el 35dB y 15dB. La media tiene una brusca caída entre el 10 % y 50 % de pérdida en comparación a todos los demás resultados. En general los resultados de el diagrama 5.4e son los peores de los cinco experimentos en el TV.

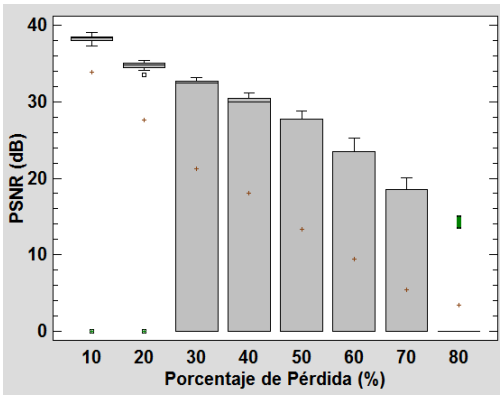
5.2.2. CDD



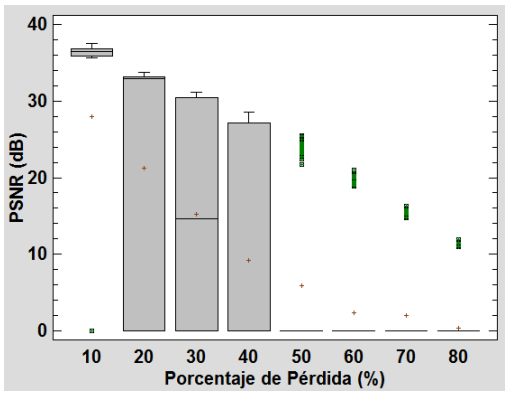
(a) block=1x1, byte TX=27



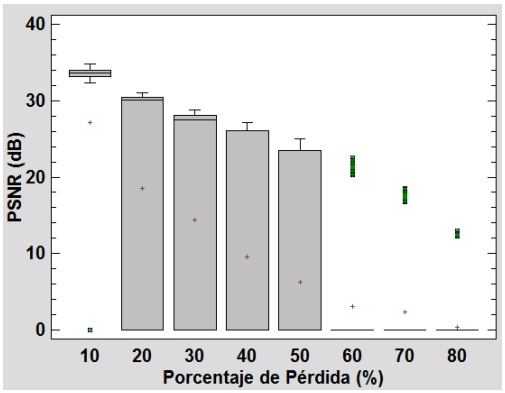
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.6: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de CDD *inpainting*

En el diagrama 5.6a para transmisión en bloques de 1x1 sobre paquetes de datos de 27 bytes, se aprecia que en 10 % de pérdida de paquetes, los resultados se agrupan cerca de los 40dB de PSNR, aun así existen resultados asilados que se escapan a 0dB, y que como se vio en el análisis para el TV, son malas restauraciones en donde el algoritmo falló. Entre el 20 % y 50 % de pérdida de paquetes de datos, se observa una gran dispersión entre el máximo y el mínimo en comparación a los demás porcentajes. De 50 % de pérdida en adelante los resultados son mayoritariamente cero dB, son muy pocas las simulaciones que tuvieron una restauración exitosa a tasas de perdidas altas. Los valores de media para cada porcentaje van bajando paulatinamente a medida que aumenta la pérdida de paquetes, la mediana de los resultados sobre 50 % de pérdida es prácticamente 0dB.

Si se comparan los resultados sobre paquetes de 27 bytes entre bloques de 1x1 y 2x2, se aprecia una gran similitud entre los diagramas, donde los único que se aprecia visualmente que varió, es el máximo de cada resultado para cada porcentaje de pérdida, siendo mayores en el diagrama 5.6a.

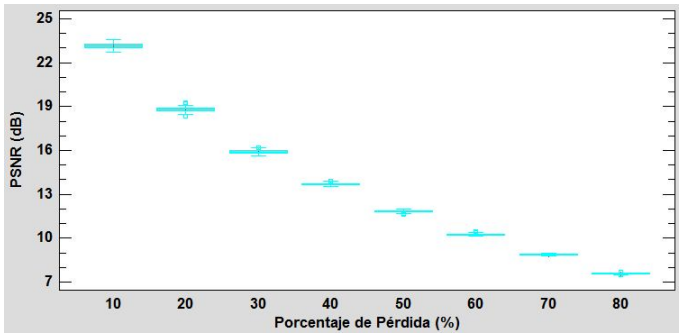
La transmisión en bloques de 1x1 sobre paquetes de 100 bytes, muestra resultados distintos a los resultados del diagrama 5.6a. A pesar de tener el mismo tamaño de bloque cuando se transmite sobre paquetes de 100 bytes, los resultados para 10 % y 20 % de pérdida se agrupan entre 32dB y 40dB, salvo resultados aislados de valores cercanos a 0dB. La tendencia de la media también es en promedio mejor que cuando se transmite sobre paquetes de 27 bytes. Entre porcentajes de pérdida de 60 % y 70 %, la dispersión es mayor que en el diagrama 5.6a, realizando casi el doble de restauraciones exitosas en comparación a transmisión en paquetes de 27 bytes (revisar tablas anexas A.1).

Los resultados para transmisión en bloques de 2x2 sobre paquetes de 100 bytes que se observan en el diagrama 5.6d, son muy parecidos a los resultados de transmisiones en bloques de 1x1 y 2x2 sobre paquetes de 27 bytes, salvo que para 50 % de pérdida tiene una menor dispersión de resultados en comparación a los demás experimentos, agrupando la mayor parte de los resultados cerca de los 0dB. La media para 30 % de pérdida de paquetes también difiere de los diagramas 5.6a y 5.6b, estando al medio de la dispersión. En todo lo demás son resultados son parecidos. Comparado ahora con la transmisión en bloques de 1x1 sobre 100 bytes, se observa que a pesar de tener el mismo tamaño de bytes, los resultados son muy distintos. En el diagrama 5.6d hay muy poca dispersión de resultados sobre los 50 % de datos perdidos, siendo la mayoría restauraciones fallidas, por otro lado en el diagrama 5.6c sobre

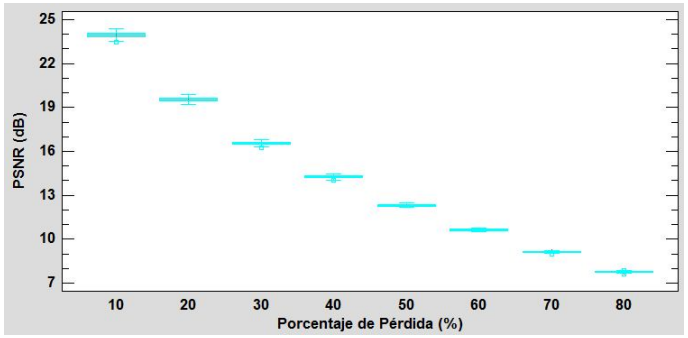
el mismo % de pérdida la dispersión es mayor al igual que la media.

Los resultados del diagrama 5.6e, con parámetros de transmisión en bloques de 1x1 sobre paquetes de 100 bytes, se parecen mucho a los resultados de los diagramas 5.6a y 5.6b, con dispersiones y medianas muy parecidas, sin embargo la media tiene una tendencia de caída mas abrupta en comparación a los otras dos simulaciones. Por otro lado a los 60 %, 70 % y 80 % de pérdida de datos, la cantidad de simulaciones con restauraciones exitosas es de 14 %, 13,5 % y 2.5 % respectivamente, siendo así la simulación con peor tasa de restauración exitosa en comparación a los resultados de los otros cuatro diagramas (revisar estadísticas anexas A.1).

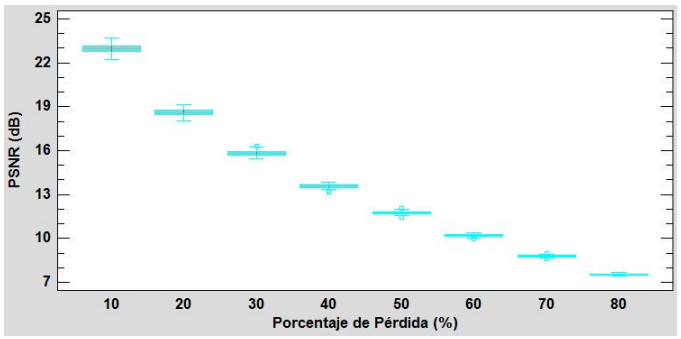
5.2.3. Bilineal



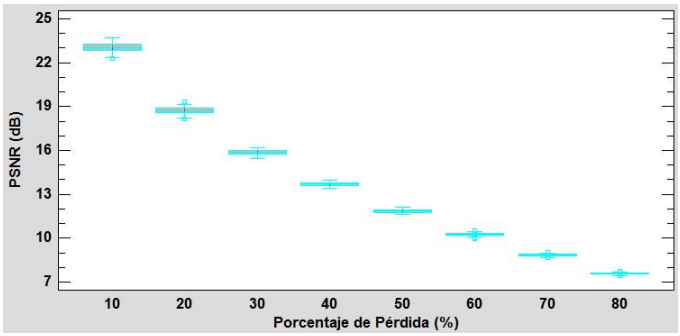
(a) block=1x1, byte TX=27



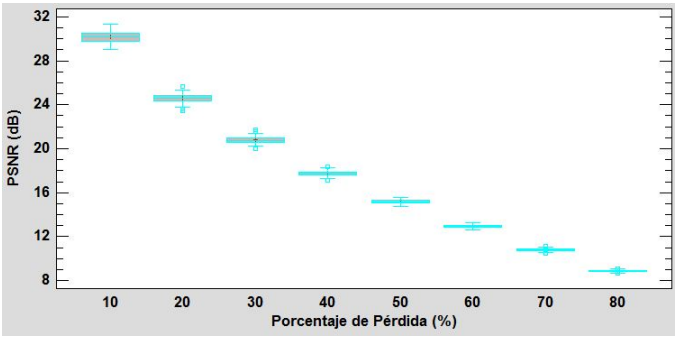
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.7: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bilineal

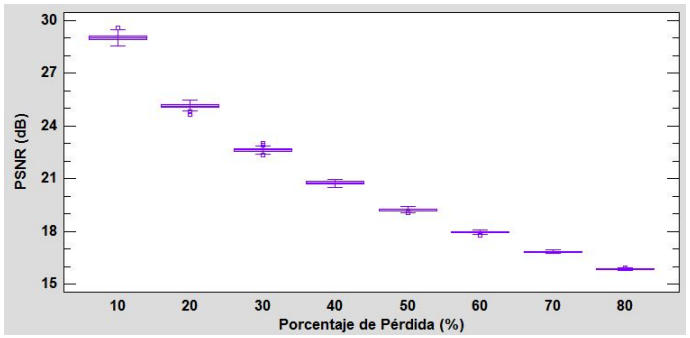
En el caso del algoritmo de disimilación de errores Bilineal, se puede observar en los cinco diagramas de la Figura 5.7 los resultados son muy parecidos entre si. La dispersión de datos en cada una de las simulaciones es muy baja, se realizaron el 100 % de las restauraciones con éxito. Se pueden notar algunas pequeñas observaciones, como por ejemplo los diagramas

5.7a, 5.7c y 5.7d, los resultados parecieran ser practicamente los mismos, por otro lado en transmisión de datos en bloques de 2x2 sobre paquetes de 100 bytes, se aprecia una media mayor a los tres diagramas mencionados anteriormente, la cual es notoria hasta el 30 % de pérdida de datos, después de ese porcentaje los valores de media del diagrama 5.7b se alinea con los valores de las otras tres simulaciones mencionadas.

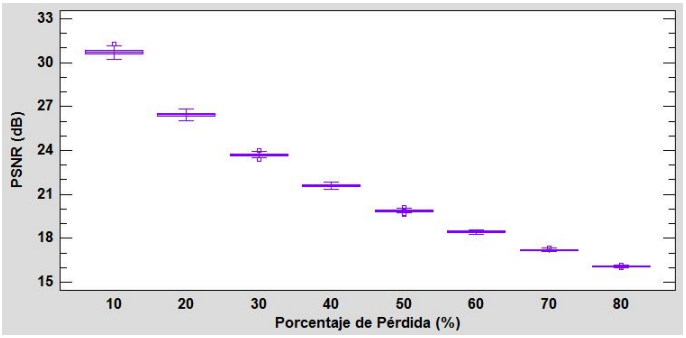
Un caso distinto ocurre en el diagrama 5.7e, en donde el promedio de los resultados es mayor a los cuatro diagramas restantes de la Figura 5.7, con una dispersión algo mayor a porcentajes de pérdida bajo la cual disminuye a medida que la perdida de paquetes aumenta.

Según las estadísticas anexadas en la sección A.1, los mejores resultados bajos los parámetros de nuestro trabajo, son cuando se transmite en bloques de 8x8 sobre paquetes de datos de 100 bytes. La máxima alcanza una valor de 31.30679dB y la mínima 28.99959dB para pérdida de paquetes del 10 %.

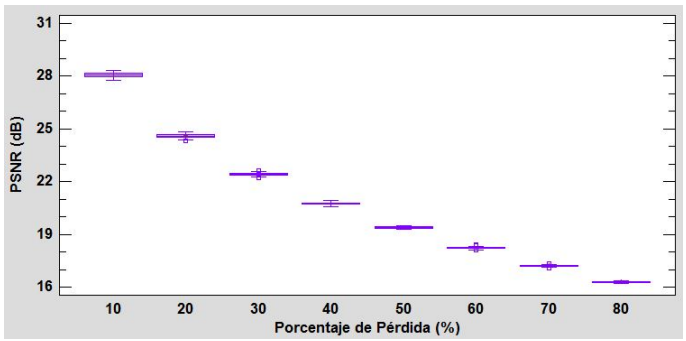
5.2.4. Bicúbico



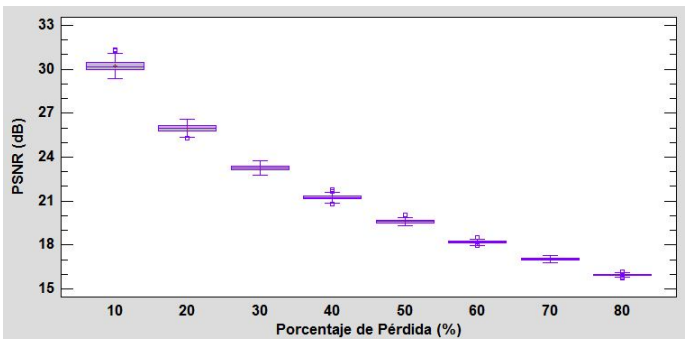
(a) block=1x1, byte TX=27



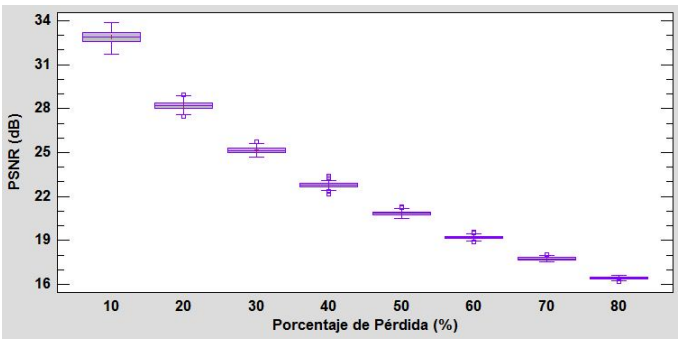
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.8: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bicúbico

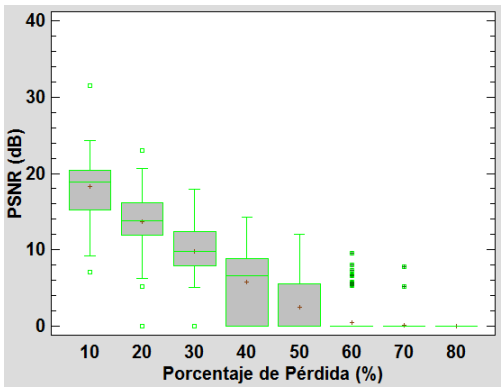
Los diagramas de la Figura 5.8 muestran resultados interesantes, para un mismo tamaño de bloque y a 10 % de pérdida de datos los resultados son mejores cuando el tamaño de paquete es menor, por otro lado sobre el 10 % de pérdida los valores tienden a igualarse. Por ejemplo el valor medio de los resultados para transmisiones en bloques de 1x1 sobre paquetes

de datos de 27 bytes es mayor que en transmisión de 1x1 sobre 100 bytes. Lo mismo ocurre con los diagramas 5.8b y 5.8d, en donde a pesar de tener el mismo tamaño de bloque, los resultados para 10 % de pérdida son mejores en paquetes de 27 bytes. Sobre el 10 % de pérdida de paquetes los resultados entre simulaciones de mismo tamaño tienden a igualarse.

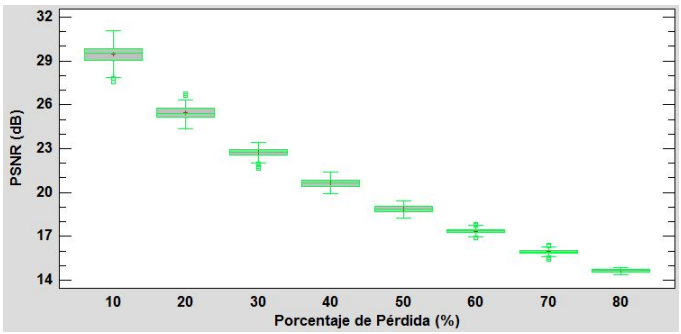
En general se puede observar que entre mas grande es el tamaño del bloque, los resultados tienden a ser mejores. Por ejemplo las restauraciones en bloques de 2x2 tienen mejor promedio que las de 1x1, así mismo las simulaciones en tamaño de bloque de 8x8 del diagrama 5.8e muestran en promedio mayor valor de PSNR a cada porcentaje de pérdida en comparación a los demás diagramas de la Figura 5.8.

Se realizaron el 100 % de las simulaciones con restauraciones exitosas con una tendencia de la media muy parecida en cada simulación del método Bicúbico.

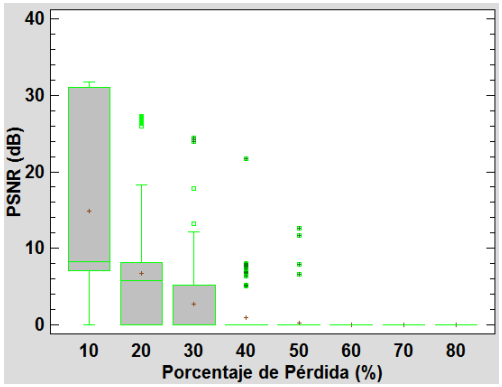
5.2.5. MSR



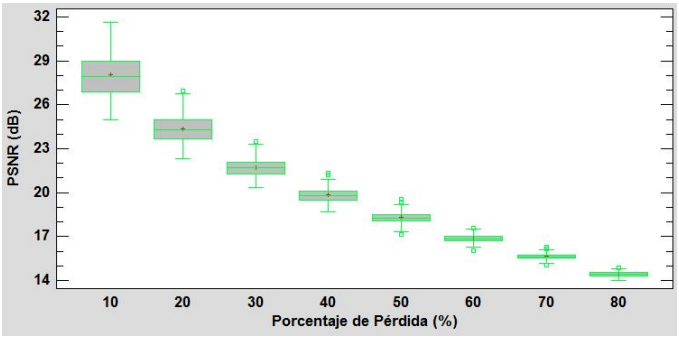
(a) block=1x1, byte TX=27



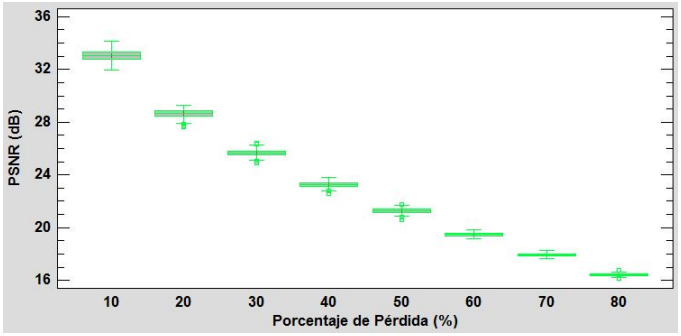
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.9: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment Maximal Smooth Recovery*

Para la restauración mediante MSR, se observa que para transmisión en bloques de 1x1 y paquetes de datos de 27 bytes, hay una dispersión muy parecida entre los distintos porcentajes de pérdida. También se observa que en el diagrama 5.9a la media y la mediana

se encuentran casi siempre al centro de las cajas de dispersiones. Cuando la pérdida es del 10 % y 20 % se observa que se escapan algunos valores del promedio, teniendo como valores máximos 31.47052dB y 23.04476dB respectivamente (revisar tablas anexas A.1). Del 40 % de pérdida de datos en adelante el porcentaje de pruebas exitosas comienza a disminuir, arrojando resultados cercanos a los 0dB, los cuales se producen debido a que el algoritmo falla al momento de la restauración. El valor mas crítico de simulaciones fallidas es cuando entre 70 % y 80 % de pérdida de datos, en donde el porcentaje de restauraciones exitosas es de 10 % y 0.5 % respectivamente.

Para el diagrama 5.9c que tiene como parámetros de transmisión bloques de 1x1 y tamaño de paquetes de datos de 100 bytes, se observa que en ninguno de los porcentajes de pérdida realizó el 100 % de las restauraciones con éxito. La dispersión de datos para perdidas del 10 % es casi el cuatro veces mayor que en el diagrama 5.9a, y en general también tiene un promedio de PSNR menor a este ultimo diagrama a cada porcentaje de pérdida. De los cinco diagramas de la Figura 5.9 el 5.9c es el con peores resultados, muchas fallas de reconstrucción, alta dispersión de datos y promedios de PSNR bajos.

El resto de los diagramas realizaron el 100 % de las simulaciones sin problemas, cuando la transmisión se realiza en bloques de 2x2 sobre paquetes de 27 bytes, tiene una dispersión de resultados menor en comparación a los diagramas 5.9a y 5.9c, con una tendencia de promedio muy parecida a los resultados de los métodos Bilineal y Bicúbico.

En el diagrama 5.9d se aprecia que a el aumento del tamaño del paquete de datos, afecto negativamente los resultados en transmisión de bloques de 2x2, ya que se aprecia un aumento en la dispersión de datos en perdidas de 10 %, 20 % y 30 % en comparación al diagrama 5.9b. El valor medio También se vio afectado, ya que según las tablas adjuntas en A.1), este bajo aproximadamente 1.5dB en cada porcentaje de pérdida en comparación a los resultados de 5.9b.

Finalmente en el diagrama 5.9e, para transmisión en bloques de 8x8 sobre paquetes de 100 bytes, los resultados muestran una poca dispersión de datos en comparación al resto de los diagramas de la Figura 5.9. El valor medio y máximo también esta por encima de las demás restauraciones del MSR. Se realizaron el 100 % de las simulaciones, habiendo fallas solamente en transmisiones de bloques de 1x1.

5.3. Imagen Nº2: Baboon 512x512

A continuación, para la imagen Baboon de la figura 5.10 se tienen resultados para cada algoritmo de ocultamiento de error, paquete de dato y tamaño de bloque:

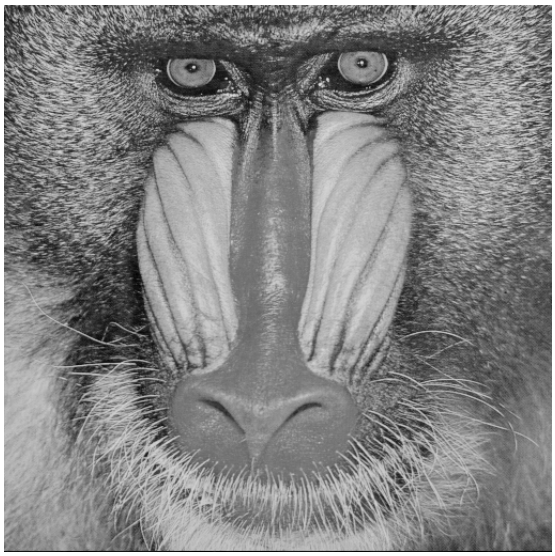
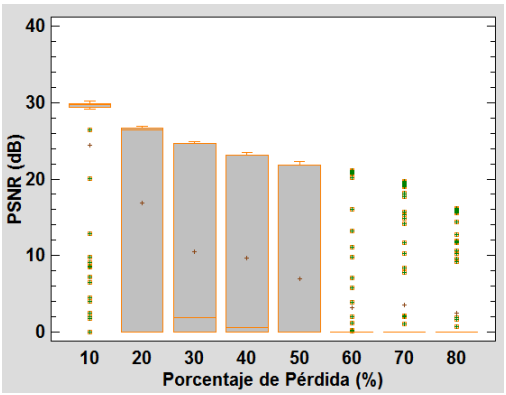


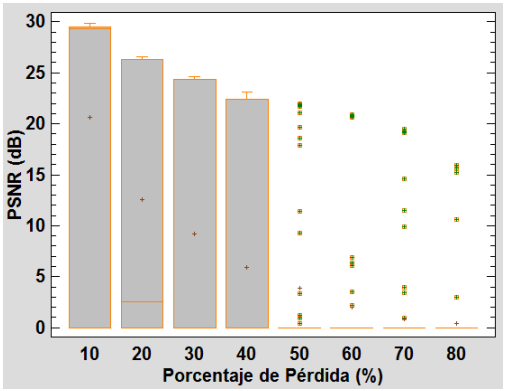
Figura 5.10: Baboon 512x512.

Los resultados de los ensayos de esta imagen al igual que anteriormente con lena, se puede observar que el TV y CDD *inpainting* provocan una gran cantidad de errores al momento de reconstruir la imagen dañada en las gráficas 5.12 y 5.11, el MSR *error concealment* de la figura 5.15 se aprecia la falta de robustez cuando se simula la transmisión de bloques de 1x1, sin embargo para bloques de 2x2 y 8x8 en cualquier tamaño de paquete utilizado, el resultado es robusto, teniéndose el 100 % de las simulaciones correctas, al igual que en el *error concealment* Bilineal y Bicúbico de las figuras 5.13 y 5.14.

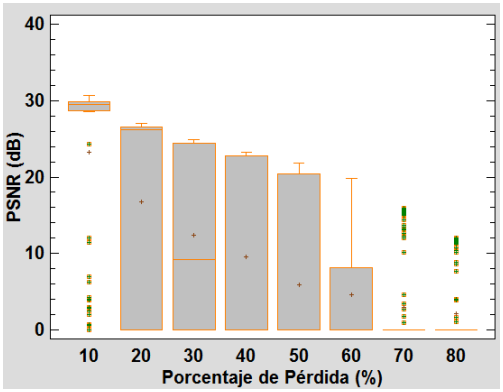
5.3.1. TV



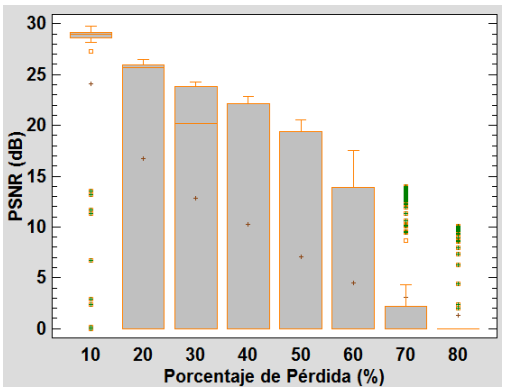
(a) block=1x1, byte TX=27



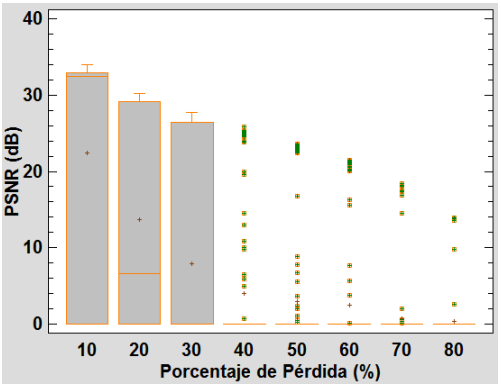
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.11: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de TV *inpainting*

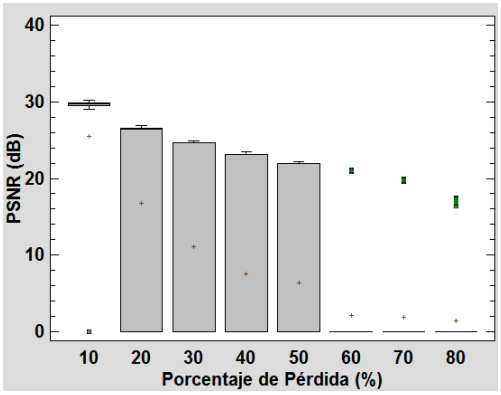
Si se analizan los resultados del método TV sobre la imagen Baboon de la Figura 5.11, y se comparan con los mismos resultados para la imagen Lena de la Figura 5.4, se aprecia que los resultados son prácticamente los mismos. Hay algunas diferencias que se comentan a continuación.

Para todos los diagramas de la Figura 5.11, se observa una disminución en los valores máximos de las simulaciones para cada porcentaje de pérdida de casi 10dB.

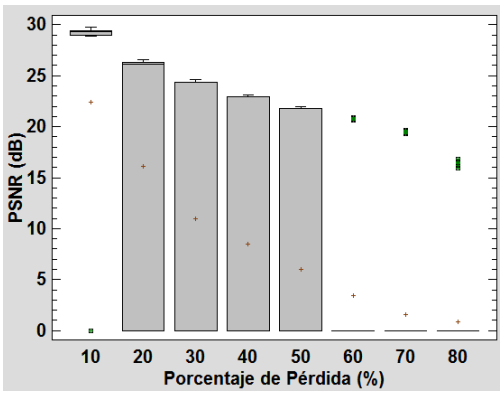
En la transmisión de bloques de 1x1 sobre paquetes de 100 bytes, se aprecia en el diagrama 5.11c que para la perdida del 30 % de paquetes de datos, la mediana se encuentra por bajo de la media, cosa que en los resultados del mismo diagrama en la imagen Lena es al revés.

El resto de los resultados son muy parecidos a los ya vistos en el análisis de la imagen Lena, con variaciones mínimas.

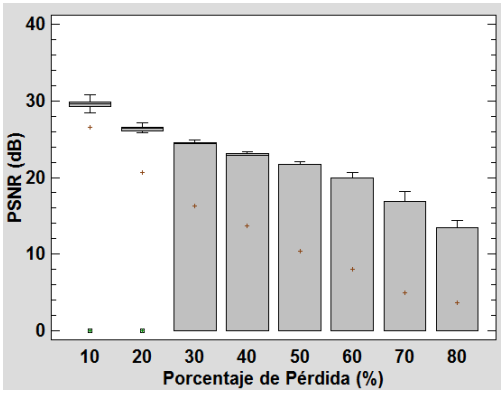
5.3.2. CDD



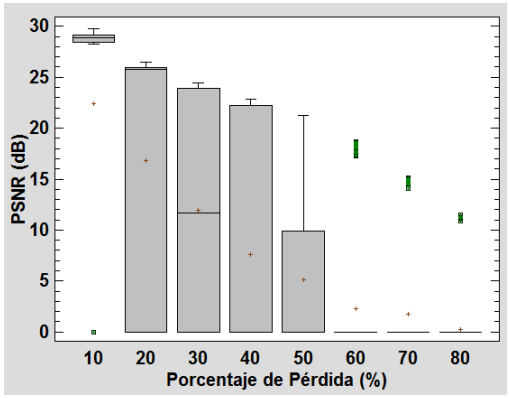
(a) block=1x1, byte TX=27



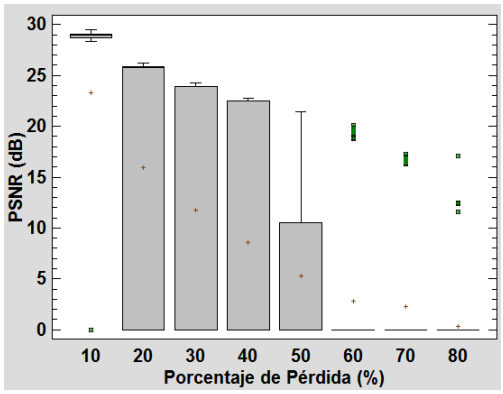
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.12: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de CDD *inpainting*

Al igual que en el análisis anterior, los resultados son prácticamente los mismos a los obtenidos en la imagen Lena con el método de restauración CDD. Algunas de las pocas diferencias se mencionan a continuación.

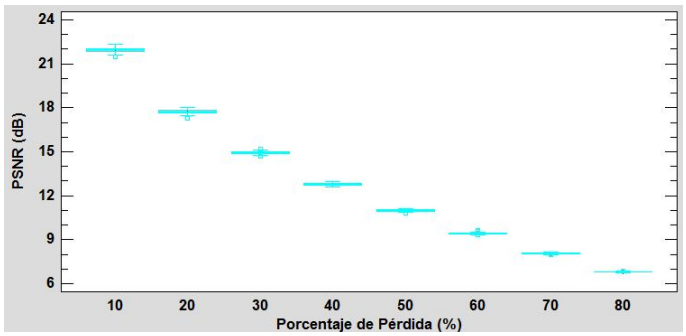
La tendencia de valores máximos de cada simulación, disminuyo en aproximadamente 10dB en comparación con los resultados de la imagen Lena para el algoritmo de CDD.

Los resultados de la transmisión en bloques de 1x1 sobre paquetes de datos de 100 bytes del diagrama 5.12c muestra una gran dispersión de datos entre los valores máximos y mínimos para pérdidas del 80 %, cosa muy distinta a lo que ocurre en el diagrama 5.6c, en donde para los mismos parámetros, la dispersión de resultados es baja, concentrándose la mayoría cerca de los valores mínimos de PSNR.

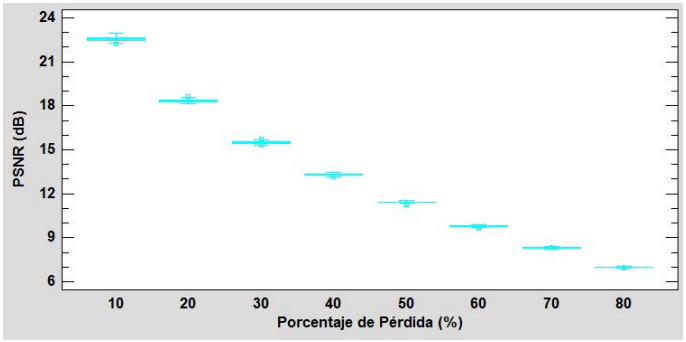
En los resultados del diagrama 5.12d, para pérdidas del 50 % de datos, los resultados están más dispersos que en los resultados para los mismos parámetros sobre la imagen Lena.

Finalmente en los resultados para bloques de 8x8 sobre paquetes de 100 bytes del diagrama 5.12e, si se compara con los resultados de los mismos parámetros para la imagen Lena, se observa una disminución en la dispersión de resultados para perdidas del 50 % de casi la mitad. El resto del análisis es el mismo aplicado a los diagramas del CDD sobre Lena.

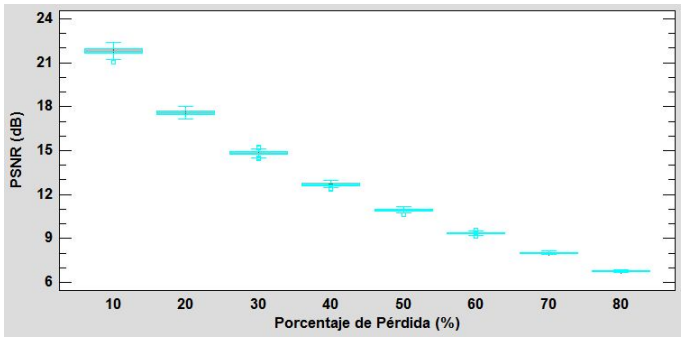
5.3.3. Bilineal



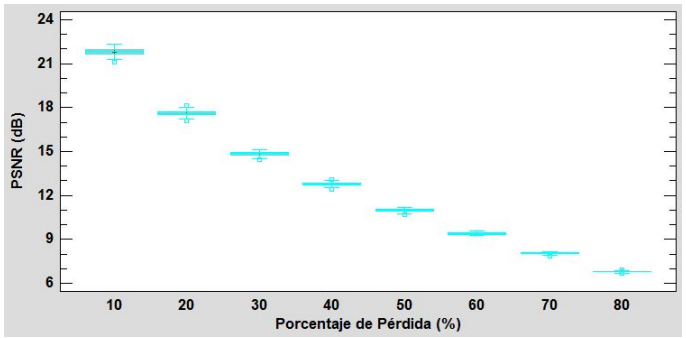
(a) block=1x1, byte TX=27



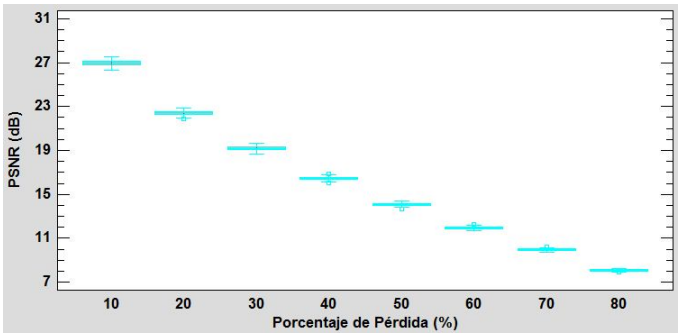
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



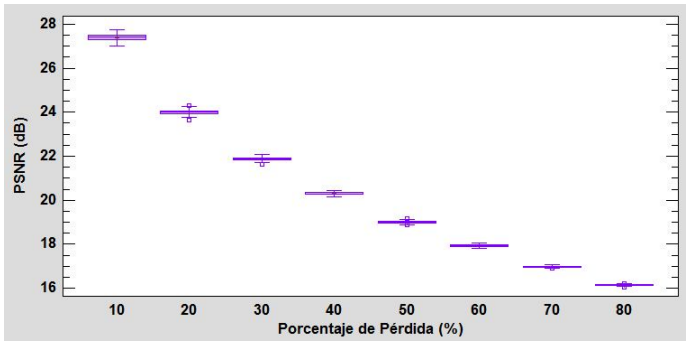
(e) block=8x8, byte TX=100

Figura 5.13: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bilineal

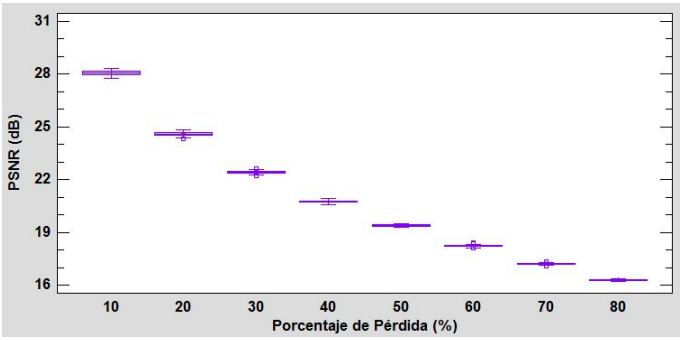
Si se comparan los resultados obtenidos en la imagen Baboon en la Figura 5.13, con los de la Figura 5.7, vemos que en transmisiones de bloques de 1x1 y 2x2 para cualquier tamaño de paquetes, se puede observar un comportamiento parecido, con la salvedad de que en los cuatro diagramas de 1x1 y 2x2 de la Figura 5.7, los valores máximos están aproximadamente

1dB por debajo de los resultados de la imagen Lena para los mismos parámetros. En el caso del diagrama 5.13e para tamaño de bloque de 8x8, la diferencia de valores máximos de PSNR es mayor con su par de la imagen Lena, aproximadamente 3dB.

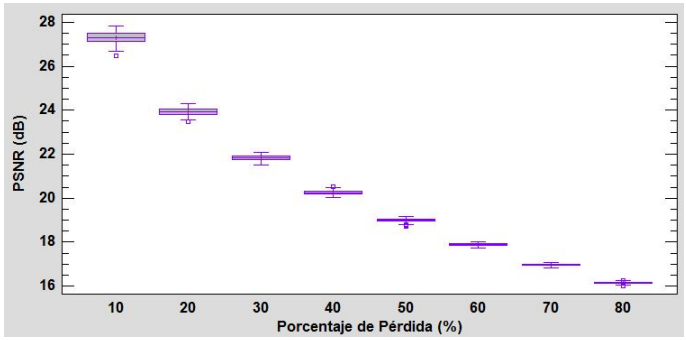
5.3.4. Bicúbico



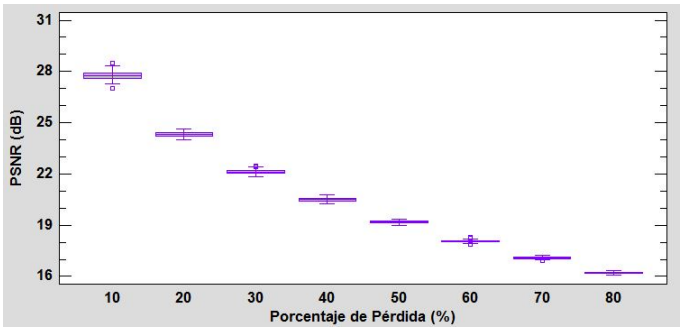
(a) block=1x1, byte TX=27



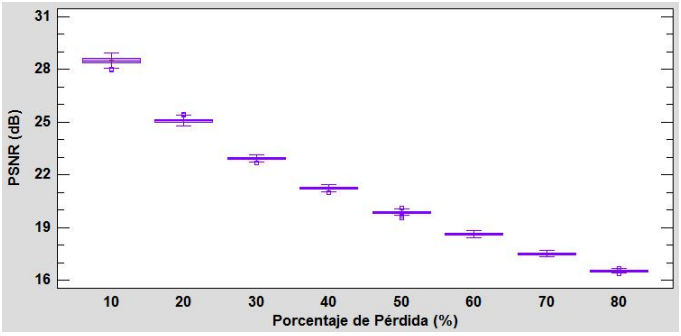
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100

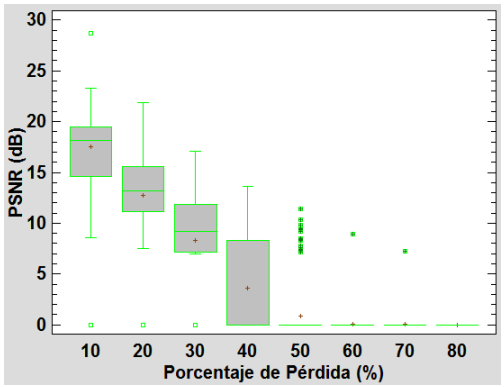


(e) block=8x8, byte TX=100

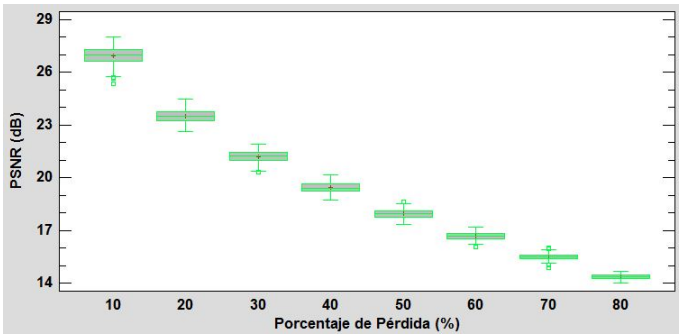
Figura 5.14: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bicúbico

Al igual que en el Bilineal, la tendencia de los resultados en la restauración Bicúbica es muy parecida a su par sobre la imagen. El valor medio para cada porcentaje de pérdida de los diagramas de la Figura 5.14 son entre 1dB a 3dB mas bajos en comparación a los resultados del Bicúbico sobre Lena.

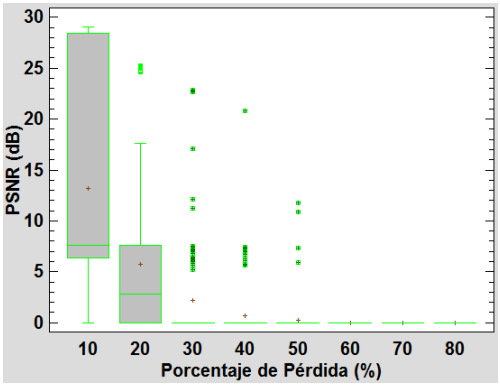
5.3.5. MSR



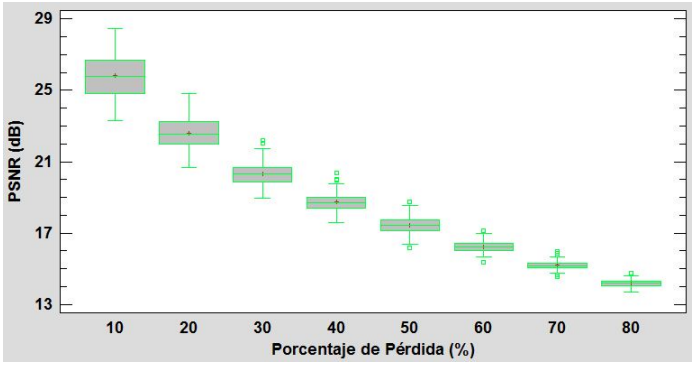
(a) block=1x1, byte TX=27



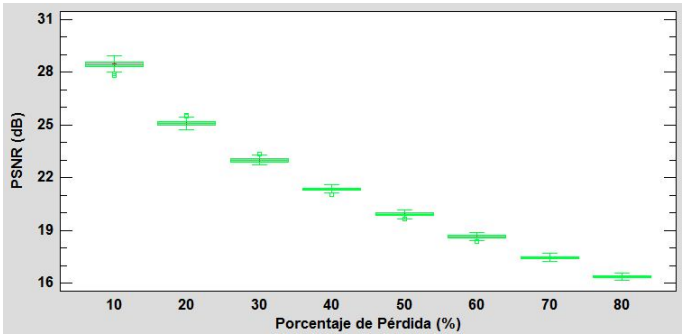
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.15: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment Maximal Smooth Recovery*

Nuevamente los resultados no varían mucho con respecto a la restauración mediante MSR sobre Lena. En este caso la media al igual que en las restauraciones anteriores sobre Baboon, están aproximadamente 2dB a 3dB por debajo de los valores medios arrojados por

la restauración sobre Lena. Esta diferencia aumenta a aproximadamente 4dB de diferencia en bloques de 8x8.

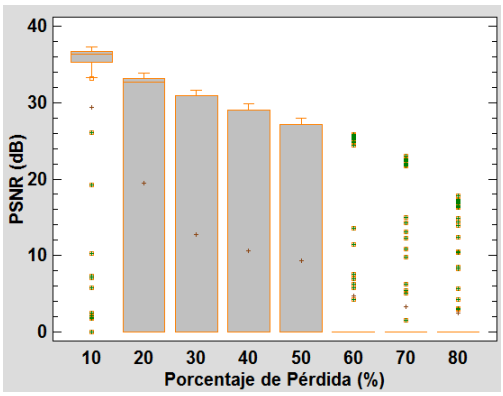
5.4. Imagen Nº3: Peppers 512x512

A continuación, para la imagen Peppers de la figura 5.16 se tienen resultados para cada algoritmo de ocultamiento de error, paquete de dato y tamaño de bloque:

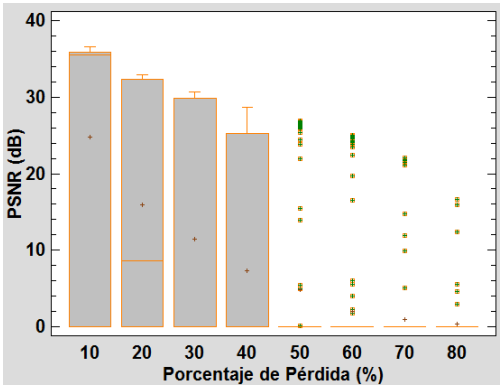
Figura 5.16: Peppers 512x512.

En los gráficos de la figura 5.17 y 5.18 se aprecia la dispersión de resultados de los algoritmos de TV y CDD al igual que en los gráficos 5.21a y 5.21c del MSR, los demás métodos entregan resultados con el 100 % de las simulaciones llevadas a cabo con éxito.

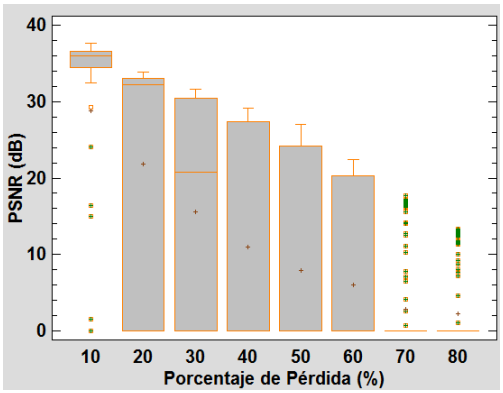
5.4.1. TV



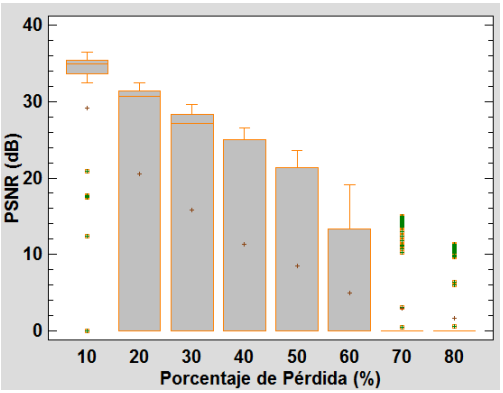
(a) block=1x1, byte TX=27



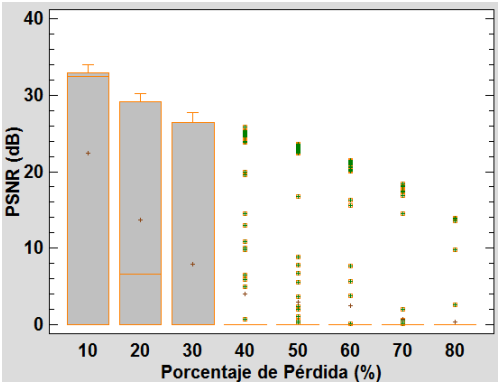
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100

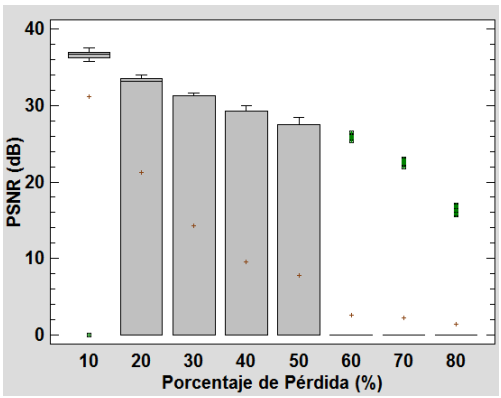


(e) block=8x8, byte TX=100

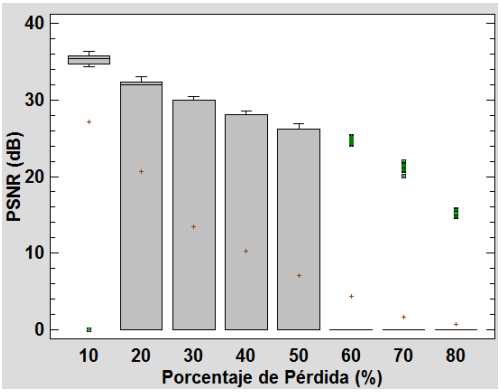
Figura 5.17: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de TV *inpainting*

Los resultados sobre la imagen Peppers que se observan en la Figura 5.17 tienen la misma forma gráfica que la restauración mediante TV sobre Baboon, sin embargo los valores máximos y mínimos se asemejan a los obtenidos sobre la imagen Lena, siendo en el caso de las imágenes de 512, Baboon la de menor valor medio en los resultados del TV.

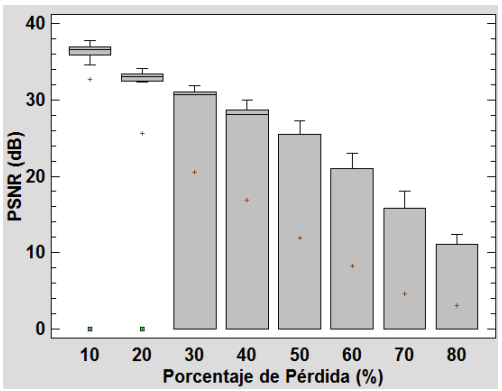
5.4.2. CDD



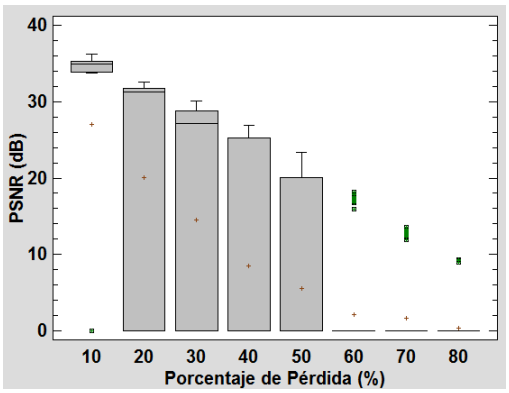
(a) block=1x1, byte TX=27



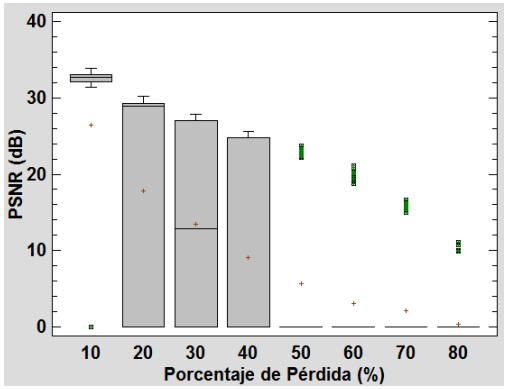
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



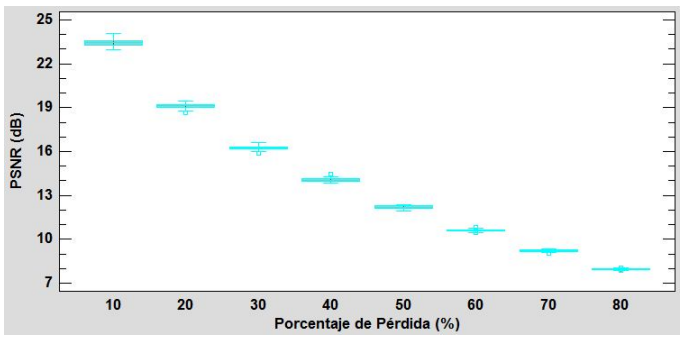
(e) block=8x8, byte TX=100

Figura 5.18: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de CDD *inpainting*

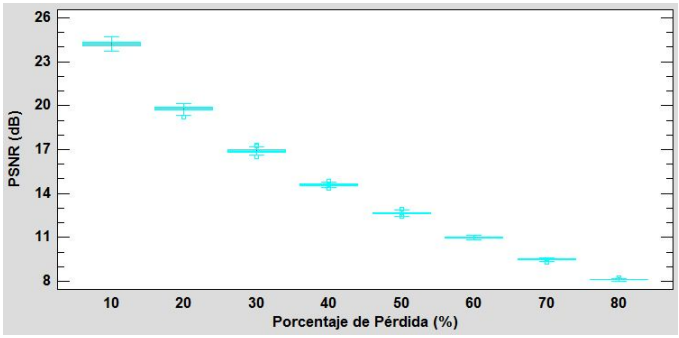
Los diagramas de caja de la Figura 5.18 varían muy poco con respecto a los resultados del CDD pare Lena y Baboon. Los valores de máximos y mínimos son prácticamente iguales a los resultados de Lena. Hay algunas salvedades, como por ejemplo el diagrama 5.18d muestra casi seis veces mas de dispersión de resultados en comparación al CDD sobre Lena a 60 % de pérdida de datos. En el mismo diagrama para pérdidas del 30 % se aprecia que la mediana de resultados se encuentra cerca de los valores máximos de PSNR, en cambio en CDD sobre lena y a mismo porcentaje la mediana se encuentra al centro de la caja de dispersión.

En el diagrama 5.18e de transmision en bloques de 8x8 y paquetes de 100 bytes, se observa que para pérdida de datos del 50 % la dispersión de resultados es aproximadamente siete veces más que los resultados sobre Lena a los mismos parámetros. En el mismo diagrama a porcentajes de pérdida de 30 % la mediana se encuentra al centro de la caja de dispersión, muy cerca de la media, por otro lado en los resultados sobre lena bajo los mismos parámetros la mediana se encuentra cerca de los valores máximos de PSNR. El resto de los resultados tienen el mismo análisis que los casos de CDD anterior.

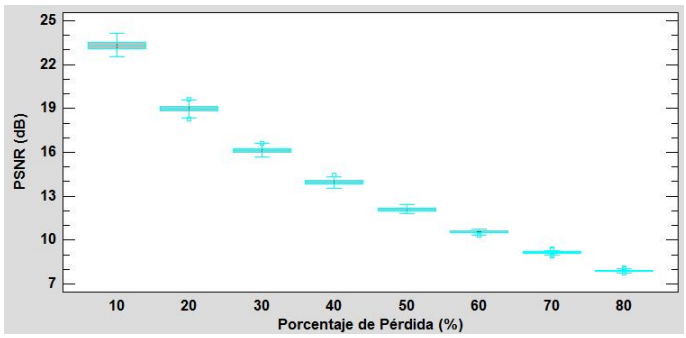
5.4.3. Bilineal



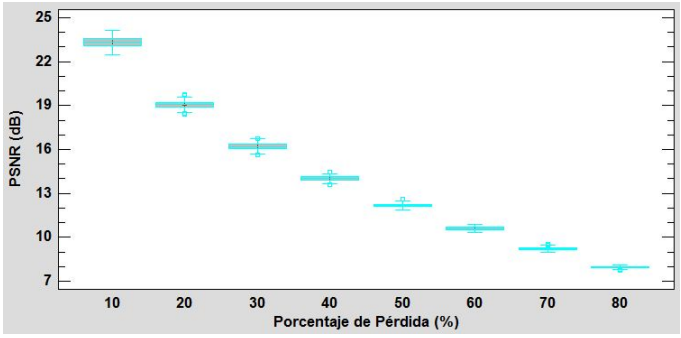
(a) block=1x1, byte TX=27



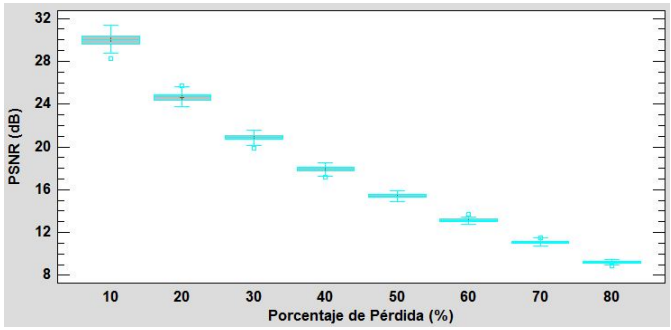
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100

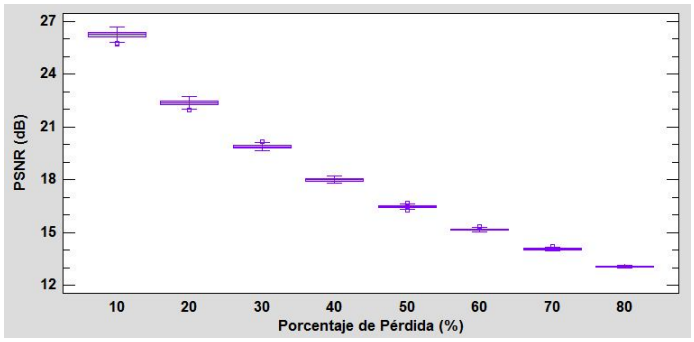


(e) block=8x8, byte TX=100

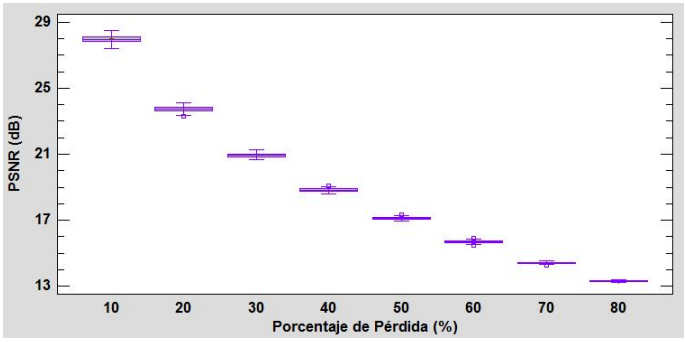
Figura 5.19: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bilineal

Visualmente los resultados de la Figura 5.19 son muy idénticos a los resultados del método Bilineal sobre Lena, por lo tanto se recomienda revisar dicho análisis el cual es aplicable a los resultados sobre la imagen Peppers.

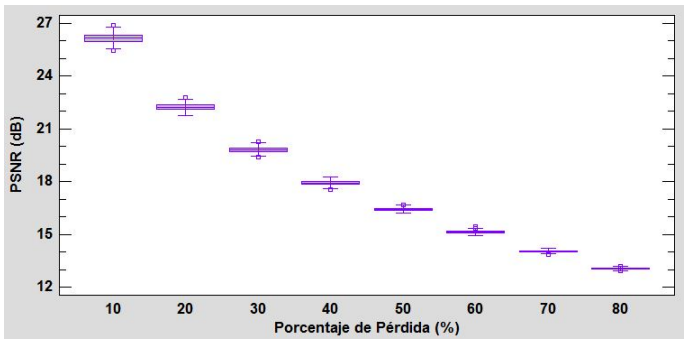
5.4.4. Bicúbico



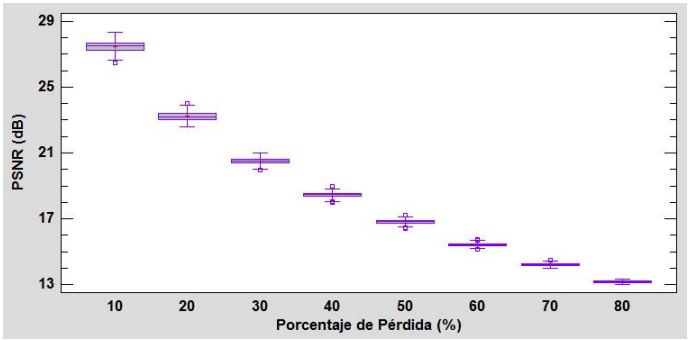
(a) block=1x1, byte TX=27



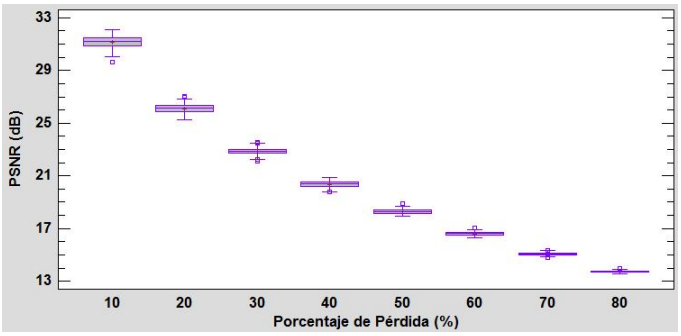
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100

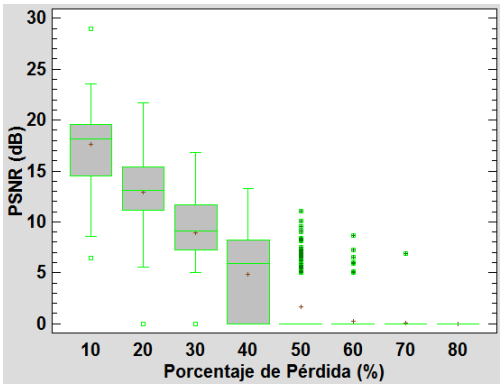


(e) block=8x8, byte TX=100

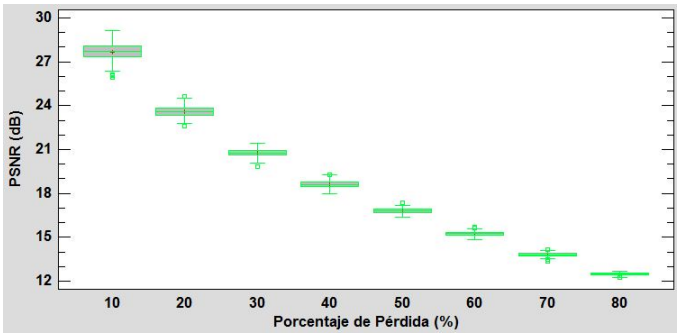
Figura 5.20: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bicúbico

Los resultados de las simulaciones para bloques de 1x1 y 2x2 de la Figura 5.20 no varían mucho de los resultados en la imagen Baboon, los que se pueden revisar en la Figura 5.14. En el caso de la transmisión en bloques de 8x8, los resultados del diagrama 5.20e son muy parecidos a los de Lena bajo los mismos parámetros.

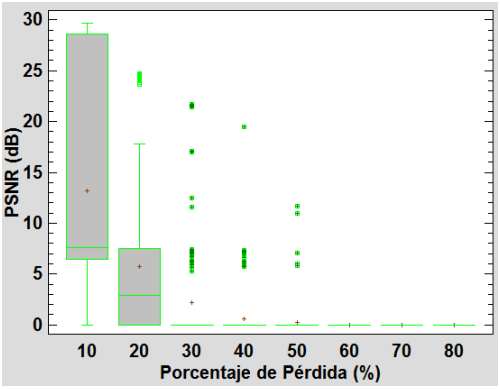
5.4.5. MSR



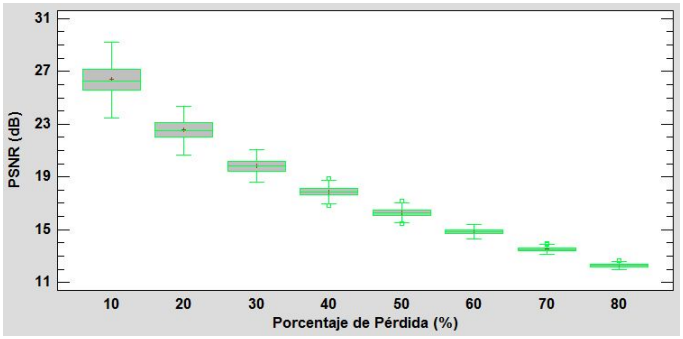
(a) block=1x1, byte TX=27



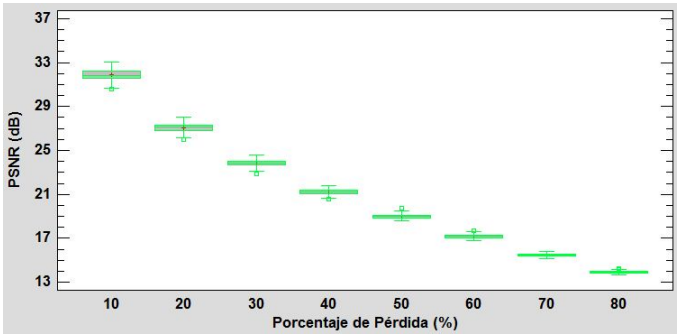
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.21: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment Maximal Smooth Recovery*

Al igual que en el Bicúbico, los resultados para bloques de 1x1 y 2x2 de la Figura 5.21 son idénticos a los resultados bajo los mismos parámetros sobre la imagen Baboon. Por otro lado para bloques de 8x8 del diagrama 5.21e, los resultados se asemejan al diagrama de MSR

sobre Lena bajo los mismos parámetros.

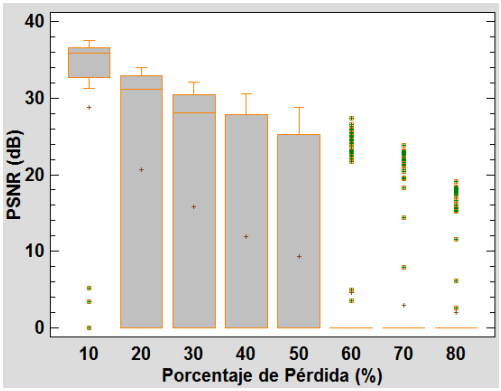
5.5. Imagen Nº4: Bird 200x200

A continuación, para la imagen Bird de la figura 5.22 se tienen resultados para cada algoritmo de ocultamiento de error, paquete de dato y tamaño de bloque:

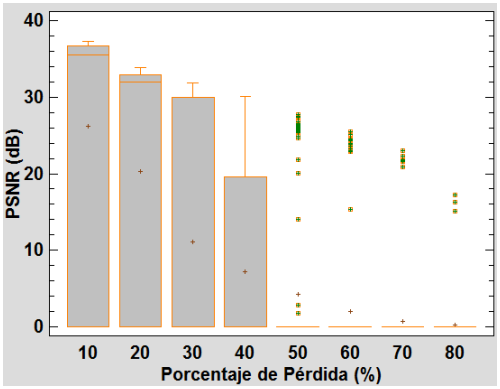
Figura 5.22: Bird 200x200.

En la imagen birds de 200x200, los resultados mantienen el mismo análisis que en las imágenes anteriores, con una pequeña variación en los resultados de los métodos TV y MSR los cuales presentan un aumento en la dispersión de resultados sin embargo a pesar de que el tamaño de la imagen disminuyó, los resultados tienen una tendencia similar.

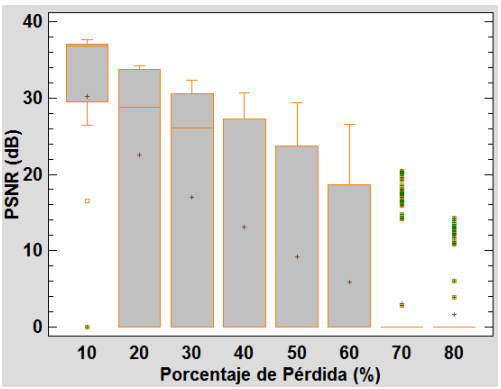
5.5.1. TV



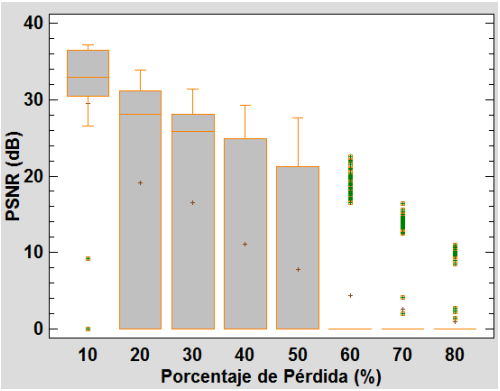
(a) block=1x1, byte TX=27



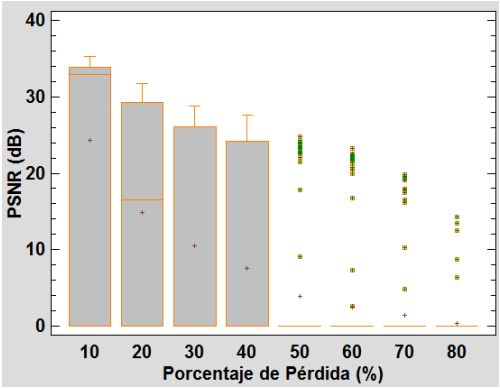
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.23: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de TV *inpainting*

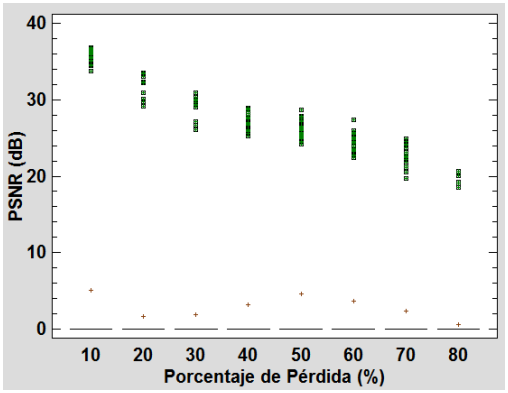
En Los diagramas de la Figura 5.23, se observa que para bloques de 1x1 sobre paquetes de 27 bytes, los resultados son muy parecidos a los diagramas de la imagen Peppers bajo los mismos parámetros de análisis. Hay algunas diferencias que se deben comentar, por ejemplo en el diagrama 5.23a para pérdidas del 30 % la mediana se encuentra cerca de los valores máximos de PSNR, por otro lado en los resultados para la imagen Peppers está rondando los 0dB. Otra diferencia es que para 80 % de pérdida de datos la dispersión de resultados en el diagrama 5.23a es casi nula, con la mayor parte de los resultados negativos, debido a restauraciones fallidas, en este caso el porcentaje de restauraciones realizadas con éxito fue de 11 % en comparación con el 16 % de restauraciones exitosas en la imagen Peppers. El resto de resultados del diagrama 5.23a es el mismo que en la imagen Peppers, para mayor análisis revisar los comentarios de la Figura 5.17.

Al igual que en el caso anterior, el diagrama 5.23b tiene resultados parecidos a su par del método TV en bloques de 2x2 y paquetes de 27 bytes sobre la imagen Peppers. Hay pequeñas diferencias como en la mediana para el 20 % de pérdida, la cual se encuentra cerca de los valores máximos de PSNR para ese porcentaje de pérdida. Por otro lado para 50 % de pérdida la dispersión es hasta cinco veces menor en comparación a los resultados sobre Peppers.

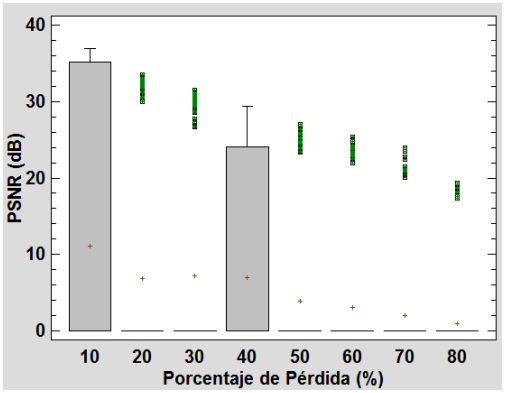
Los resultados de los diagramas 5.23c y 5.23d son prácticamente los mismos que en la imagen Peppers, con la diferencia que en 5.23c la dispersión de resultados para pérdidas del 80 % es poca en comparación a su par sobre la imagen Peppers.

El diagrama 5.23e de bloques de 8x8 con transmisión sobre paquetes de datos de 100 bytes, tienen una dispersión muy alta de resultados a perdidas del 50 % de datos en comparación a el mismo diagrama sobre Peppers. El resto de los resultados como media, mediana y tendencia de máximos y mínimos son casi los mismos.

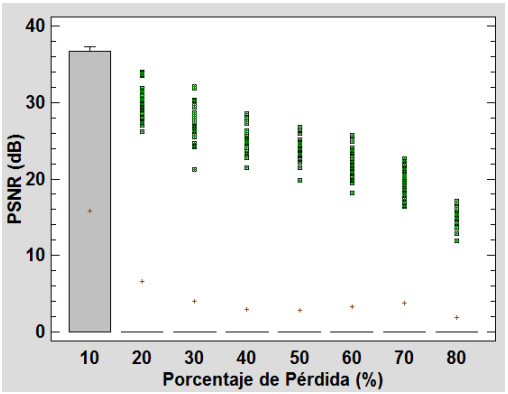
5.5.2. CDD



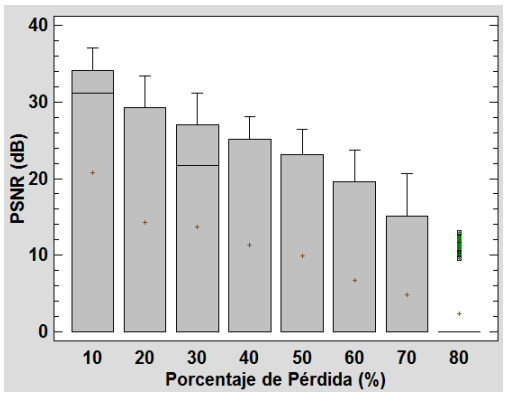
(a) block=1x1, byte TX=27



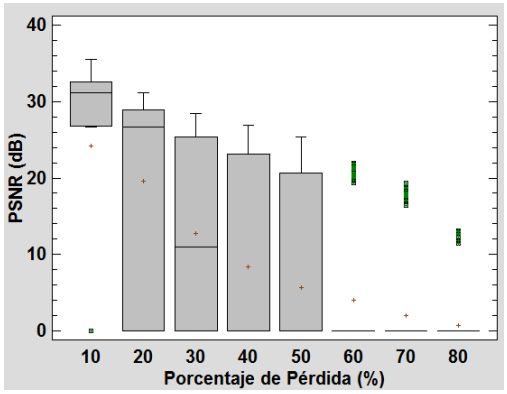
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.24: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de CDD *inpainting*

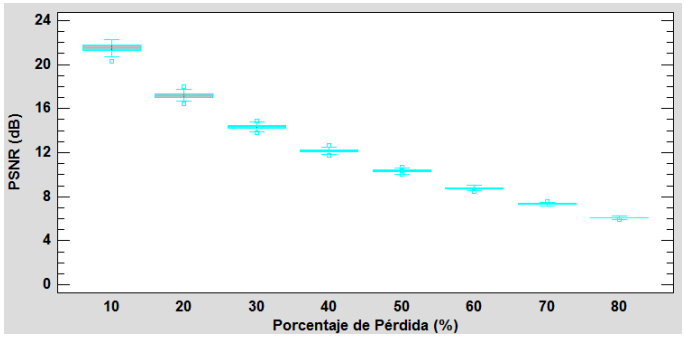
Para el diagrama 5.24a que tiene como características bloques de 1x1 y paquetes de 27 bytes, los resultados están a lo largo de todo el gráfico agrupados en los valores mínimos de las simulaciones, siendo la mayor parte de las restauraciones fallidas por parte del CDD. Para 10 % de pérdida de paquetes de datos el porcentaje de restauraciones exitosas fue del 14 %, y disminuye hasta llegar al 3 % de restauraciones exitosas a medida que aumenta la perdida de paquetes de datos (revisar tablas anexas en A.4).

Los resultados para el diagrama 5.24b muestra valores parecidos al diagrama 5.24a, con la diferencia que para porcentajes de pérdida de datos del 10 % y 40 % la dispersión de resultados va desde los valores mínimos a los máximos de PSNR. Lo mismo ocurre en el diagrama 5.24c para bloques de 1x1 y paquetes de 100 bytes, en donde se aprecian los mismos resultados que en el diagrama 5.24a, con una diferencia a 10 %, en donde se observa una gran dispersión de resultados en comparación al diagrama 5.24a.

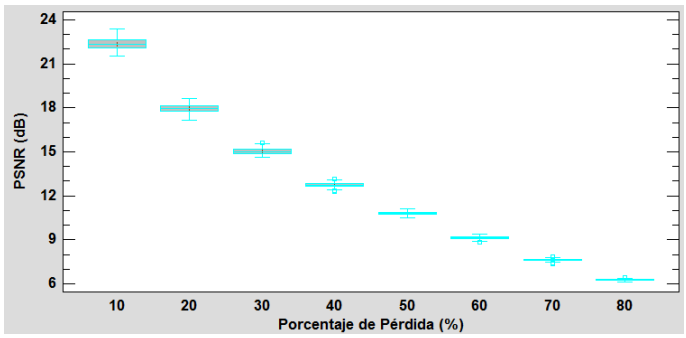
En el diagrama 5.24d para bloques de 2x2 y paquetes de 100 bytes, se aprecian resultados muy distintos a los demás diagramas de la Figura 5.24. Para porcentajes de pérdida entre 10 % y 70 % los resultados están dispersos entre el valor de PSNR máximo y mínimo. La mediana fluctúa entre el máximo y mínimo de PSNR y la media en todos los casos tiene valor negativo. Para porcentajes de pérdida del 80 % los resultados se agrupan en el valor mínimo de las simulaciones.

El diagrama para bloques de 8x8 y paquetes de datos de 100 bytes es el único que se parece a los resultados arrojados en la imagen Peppers. La única diferencia es que para porcentajes de pérdida de 50 % la dispersión de datos es mucho mayor en comparación a los resultados sobre la imagen Peppers.

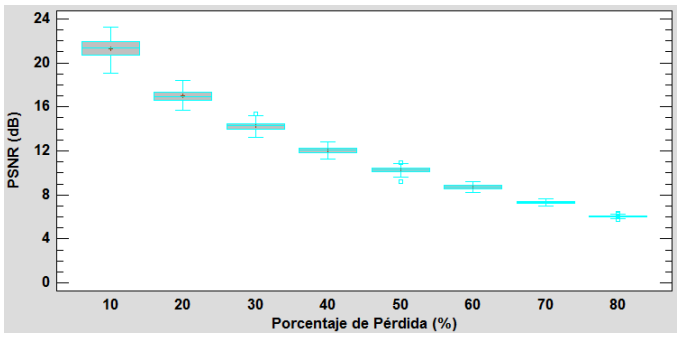
5.5.3. Bilineal



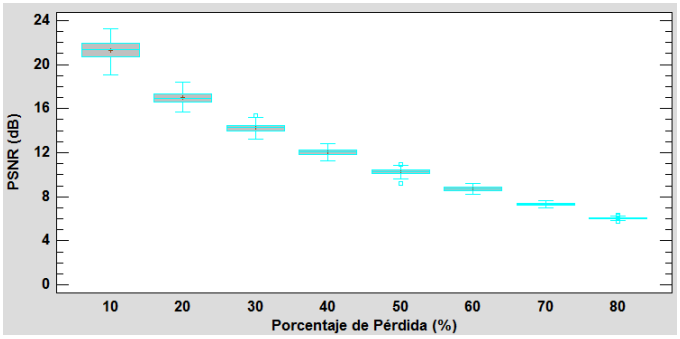
(a) block=1x1, byte TX=27



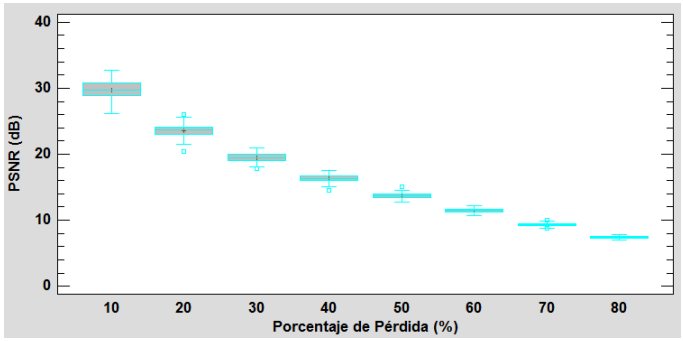
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100

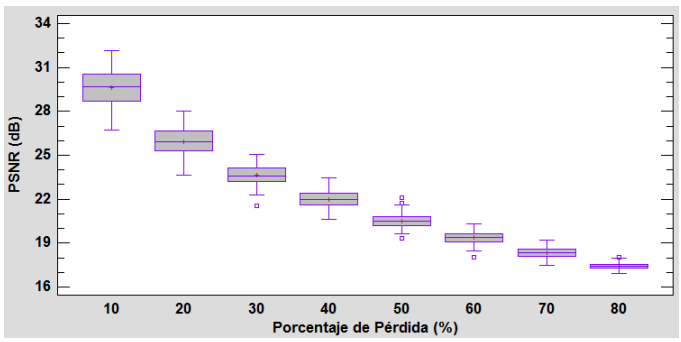


(e) block=8x8, byte TX=100

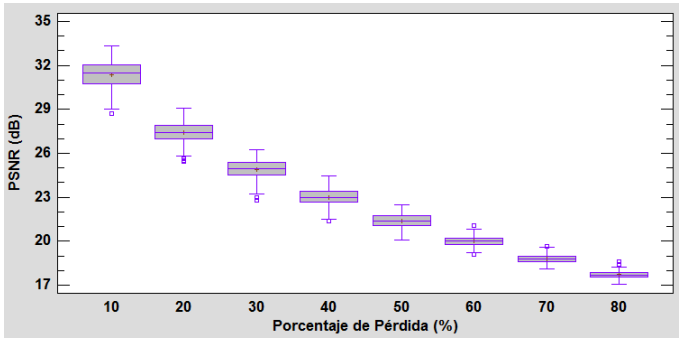
Figura 5.25: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bilineal

Según los resultados de los diagramas de la Figura 5.25, las simulaciones tienen la misma tendencia de resultados que en la imagen Peppers, con la diferencia que para cada porcentaje sobre el método bilineal sobre la imagen Birds, la media se encuentra aproximadamente 1dB por debajo de la media de los resultados sobre la imagen Peppers.

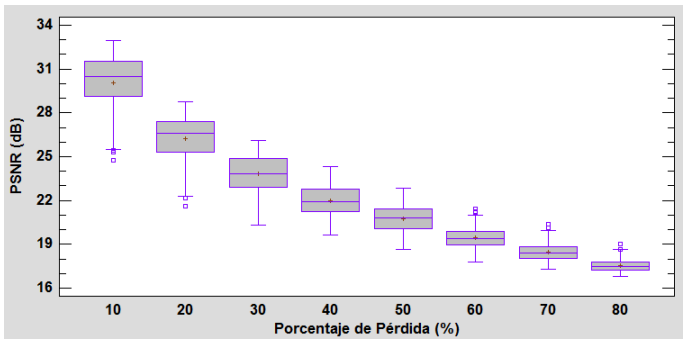
5.5.4. Bicúbico



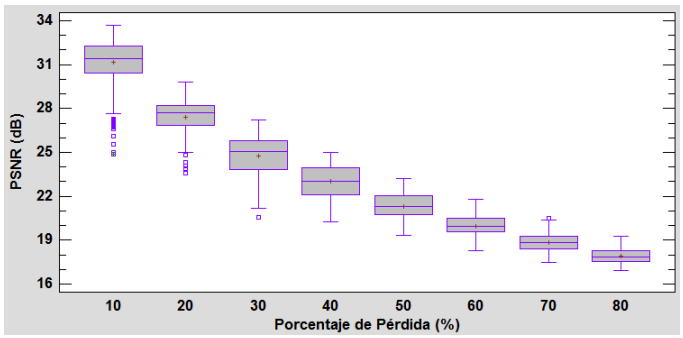
(a) block=1x1, byte TX=27



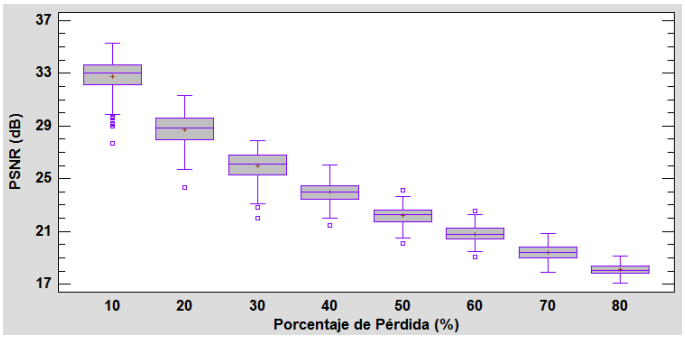
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

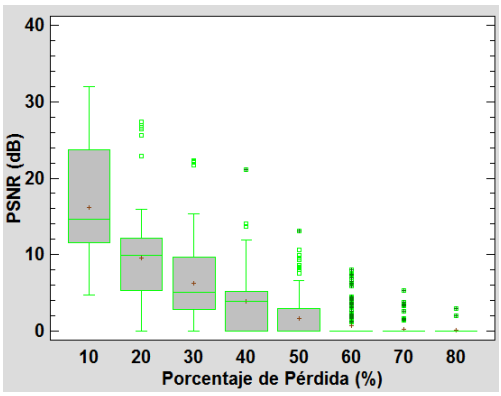
Figura 5.26: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bicúbico

Los resultados de la Figura 5.26 muestran que para los dos diagramas en bloques de 1x1 los resultados no varían mucho, hay un poco más de dispersión de datos en el diagrama 5.26c que tiene paquetes de datos de 100 bytes, pero la Media y la mediana son muy parecidas en ambos diagramas.

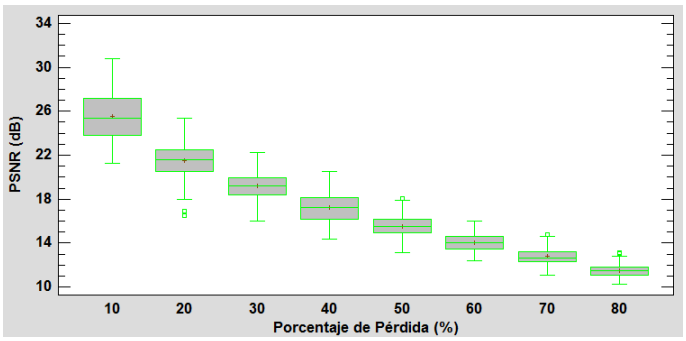
Los resultados para bloques de 2x2 también se parecen entre ellos, con la misma diferencia que se vio antes para los bloques de 1x1. En general la media se encuentra 1dB por sobre los resultados para transmisiones en bloques de 1x1.

Para transmisiones en bloques de 8x8 y paquetes de datos de 100 bytes, se observa que la media para 10 % de perdida de paquetes de datos se aproximadamente 3dB por sobre la media en transmisiones de bloques de 2x2. A medida que el porcentaje de pérdida aumenta esta diferencia va disminuyendo, y se igualan cuando los diagrama 5.26b, 5.26d y 5.26e llegan al 70 % de pérdida de paquetes de datos.

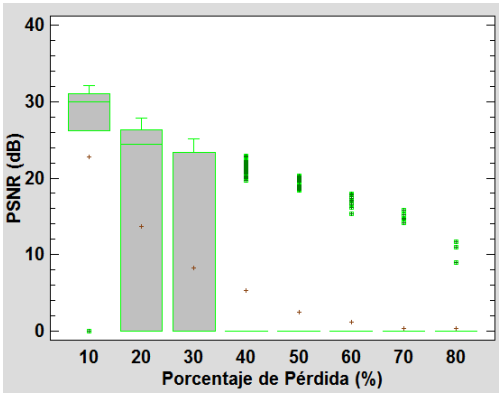
5.5.5. MSR



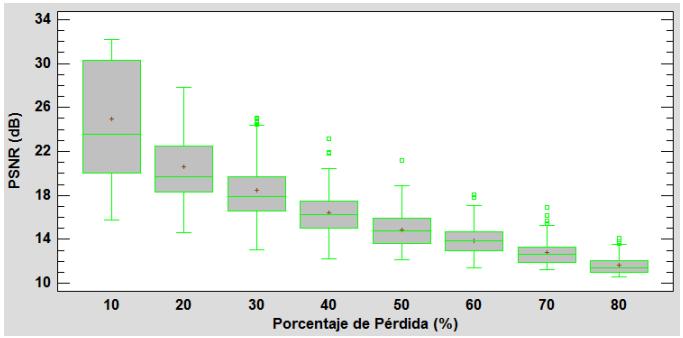
(a) block=1x1, byte TX=27



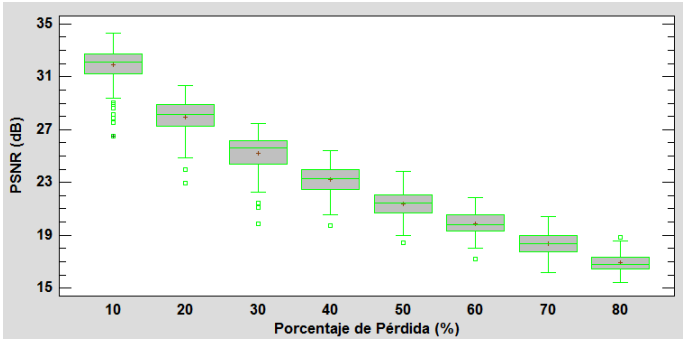
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.27: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment Maximal Smooth Recovery*

La Figura 5.27 muestra que para tamaños de bloques de 1x1, el algoritmo falla al momento de restaurar la imagen, en el caso de paquetes de datos de 100 bytes, la dispersión

de datos y la cantidad de restauraciones fallidas es mayor en comparación a la transmisión sobre paquetes de 27 bytes (revisar tablas anexas A.4).

La transmisión en bloques de 2x2 de los diagramas 5.27b y 5.27d tuvieron el 100 % de las simulaciones exitosas. Para transmisiones en paquetes de 27 bytes la dispersión es aproximadamente dos veces menor a la dispersión en paquetes de 100 bytes. De igual manera la media es aproximadamente 2dB mayor en paquetes de 27 bytes.

La transmisión en bloques de 8x8 y paquetes de 100 bytes del diagrama 5.27e, muestra que para cada porcentaje de pérdida de paquetes de datos, la media se encuentra aproximadamente 7dB por sobre los valores de PSNR medios de los resultados para la transmisión sobre paquetes de datos en bloques de 2x2. Se realizaron el 100 % de las simulaciones de manera exitosa.

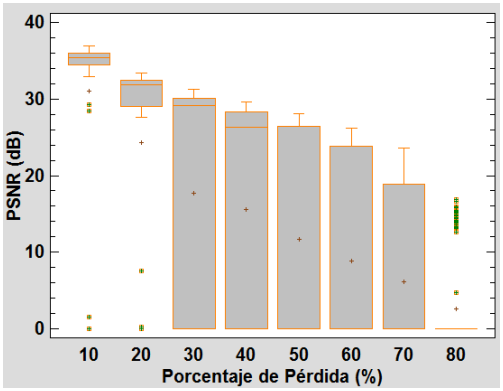
5.6. Imagen Nº5: Corridor 128x128

A continuación, para la imagen Corridor de la figura 5.28 se tienen resultados para cada algoritmo de ocultamiento de error, paquete de dato y tamaño de bloque:

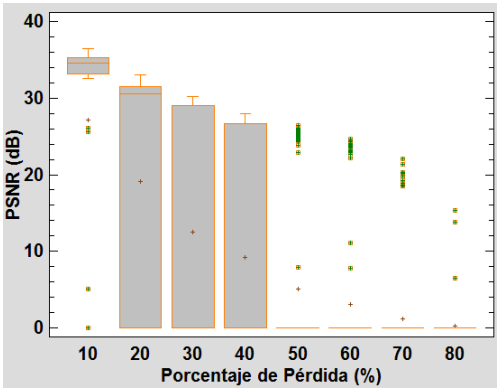
Figura 5.28: Corridor 128x128.

Los resultados muestran que al igual que en la imagen Bird, vuelve a aumentar levemente la dispersión de los resultados, principalmente en la figura 5.33 donde se observa como los resultados del MSR comienzan con una dispersión de datos muy alta la cual disminuye a medida que aumenta el porcentaje de pérdida, por otro lado, la gráfica 5.31e (para bloques de 8x8 y 100 bytes) del método Bilineal muestra una alza en el valor máximo de PSNR en comparación a las otras cuatro gráficas de la figura 5.31. Los demás algoritmos presentan resultados muy similares a los ya vistos en las imágenes de 512x512.

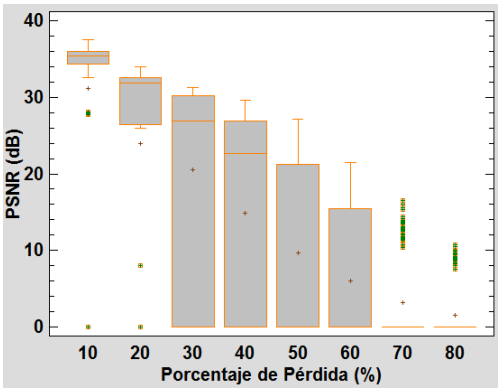
5.6.1. TV



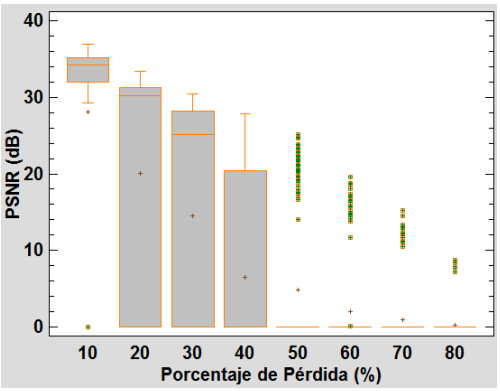
(a) block=1x1, byte TX=27



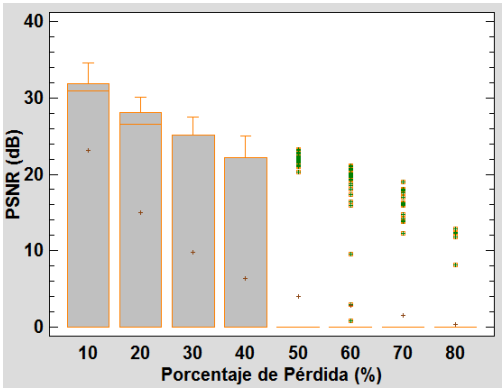
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.29: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de TV *inpainting*

En el diagrama 5.29a con transmisión de bloques de 1x1 y paquetes de 27 bytes, se observa que los valores máximos están generalmente entre 20dB y 40dB. Esto es igual para los diagramas 5.29b, 5.29c y 5.29d.

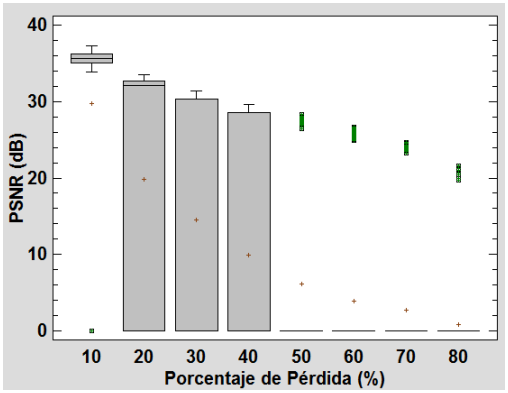
La transmisión en bloques de 1x1 y paquetes de 100 bytes, se parece al diagrama 5.29a. Cuando el porcentaje de pérdida es del 50 %, la mediana esta cerca del valor mínimo de PSNR, por el contrario del diagrama 5.29a, en donde la mediana esta por sobre la media.

Los resultados para transmisión en bloques de 2x2 de los diagramas 5.29b y 5.29d, son prácticamente iguales con pequeñas variaciones de dispersión de datos al 50 % de pérdida. Esta variación es irrelevante debido a que son en su mayoría restauraciones fallidas.

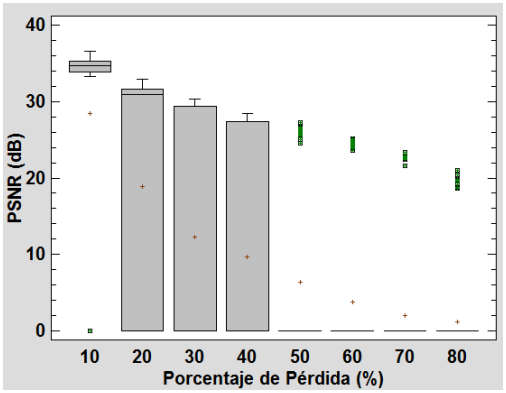
Cuando se transmite en bloques de 8x8, se obtienen resultados como los que se aprecian en el diagrama 5.29e, en donde los resultados son muy parecidos a los que se obtienen en la transmisión en bloques de 2x2 y paquetes de 27 bytes. Hay una diferencia en la mediana para porcentajes de pérdida de 30 %, ya que en el diagrama 5.29e esta se encuentra por sobre la media y en el diagrama 5.29b se encuentra muy por debajo de la mediana, cercano a los valores mínimos de PSNR.

En cuanto a la cantidad de simulaciones fallidas, se observa en las tablas anexadas A.1, A.3, A.2 y A.5, que estas no varían demasiado entre una imagen de 512x512 y otra de 128x128, hay diferencias de entre 5 % y 10 %, por lo que el tamaño de la imagen este trabajo de título no es motivo de falla en la reconstrucción del algoritmo.

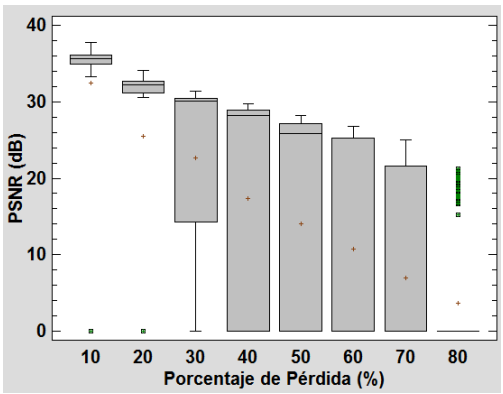
5.6.2. CDD



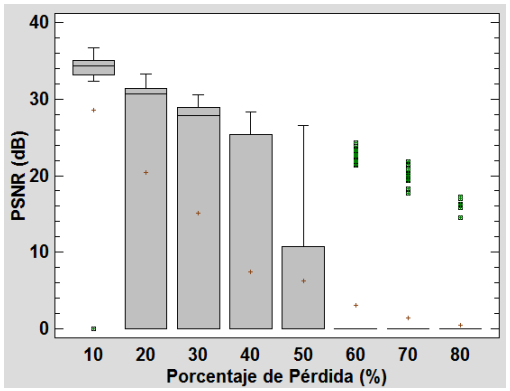
(a) block=1x1, byte TX=27



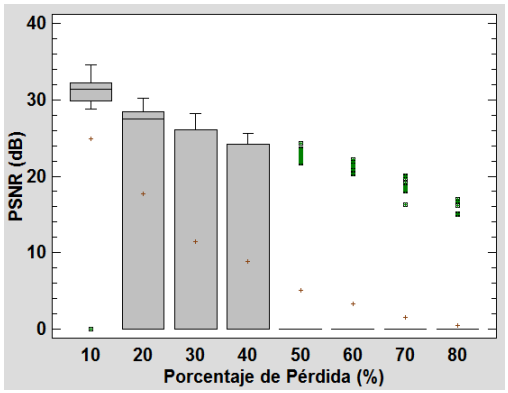
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.30: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de CDD *inpainting*

De resultados de los diagramas 5.30a de bloques de 1x1 y subfig-1:corridorcddB de bloques de 2x2, se observa que prácticamente no hay diferencia, entre los resultados, al parecer en este caso no afecta el cambio en el tamaño del bloque.

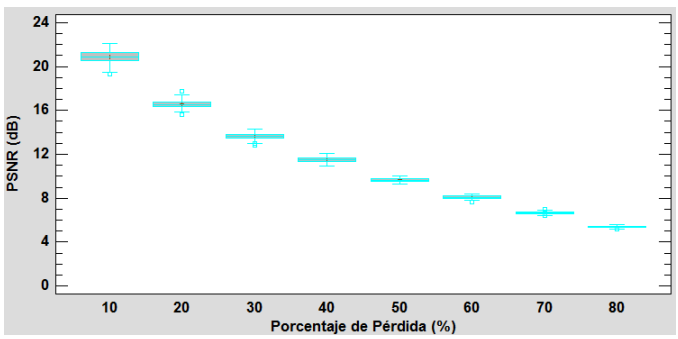
En el diagrama 5.30c bajo 10 % y 20 % de pérdida, los resultados se agrupan en mayor cantidad cerca de los 35dB. Para porcentajes mayores los valores se dispersan entre el valor máximo y mínimo de PSNR. A 80 % de pérdida de datos, los datos se acumulan en la zona negativa del gráfico, cerca del valor mínimo de PSNR, en este caso el porcentaje de éxito en las restauraciones es de 6 % (revisar tablas anexadas en A.5).

La transmisión en bloques de 2x2 y paquetes de datos de 100 bytes, muestran resultados parecidos a los de los diagramas 5.30a y 5.30b, sin embargo la mediana se encuentra muy cerca de los valores máximos de PSNR cuando la pérdida es de 30 %, cosa que no se cumple en los otros dos diagramas. Cuando el porcentaje de pérdida es del 50 %, se observa que la dispersión es aproximadamente el triple en comparación a los diagramas 5.30a y 5.30b. En el resto son muy similares.

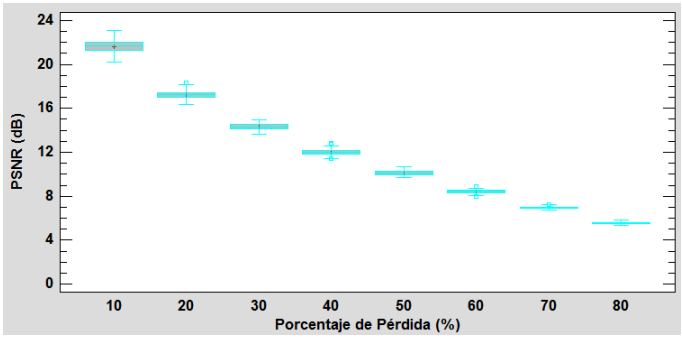
Finalmente, el diagrama 5.30e de bloques de 8x8 y paquetes de 100 bytes, es prácticamente igual a los diagramas 5.30a y 5.30b, con la salvedad de que en comparación a estos dos no tiene dispersión a 50 % de perdida de datos.

La cantidad de restauraciones fallidas se mantiene al igual que en las imágenes vistas anteriormente.

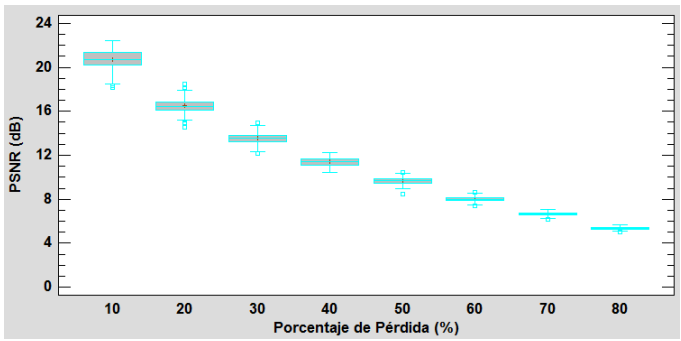
5.6.3. Bilineal



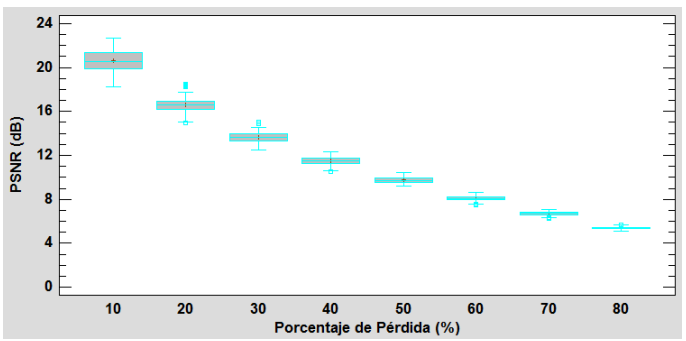
(a) block=1x1, byte TX=27



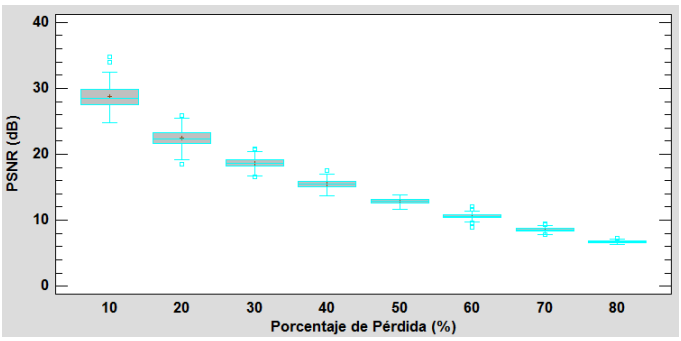
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

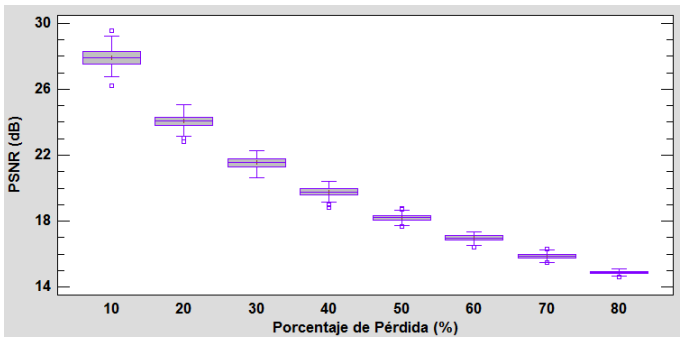
Figura 5.31: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bilineal

Los resultados de los diagramas 5.31a, 5.31b, 5.31c y 5.31d, muestran resultados prácticamente iguales. Hay algunas mínimas diferencias, por ejemplo para simulaciones con paquetes de datos de 100 bytes la dispersión en los resultados para 10 % y 20 % de pérdida es mayor en comparación a los otros tres diagramas. En si los valores medios son muy parecidos, con

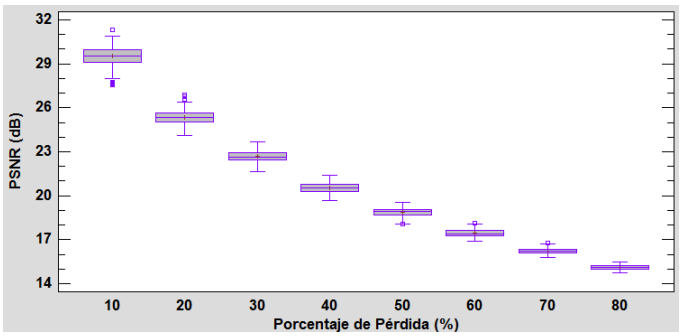
una tendencia similar a medida que aumenta el porcentaje de pérdida.

Para los resultados de simulaciones en bloques de 8x8 y paquetes de datos de 100 bytes del diagrama 5.31e, se observa que entre 10 % y el 30 % de pérdida de paquetes de datos, los valores medios tienen entre 11dB y 5dB en comparación a los otros cuatro diagramas de caja (revisar tablas anexadas en A.5).

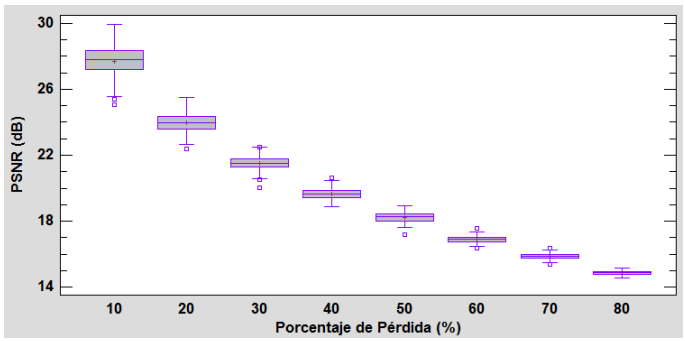
5.6.4. Bicúbico



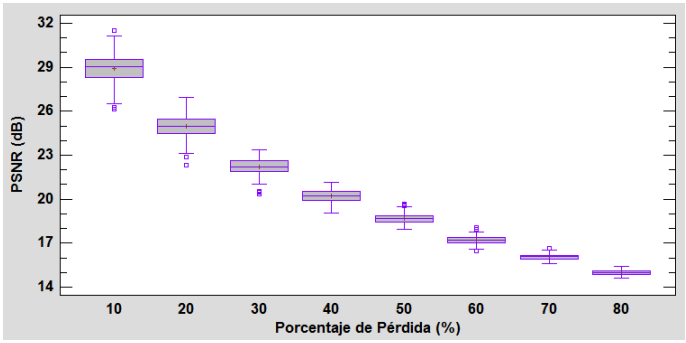
(a) block=1x1, byte TX=27



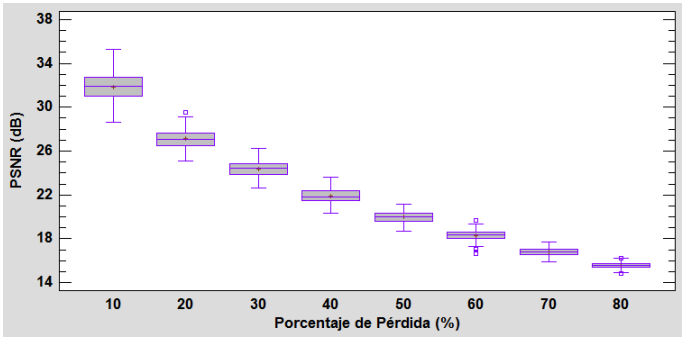
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.32: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bicúbico

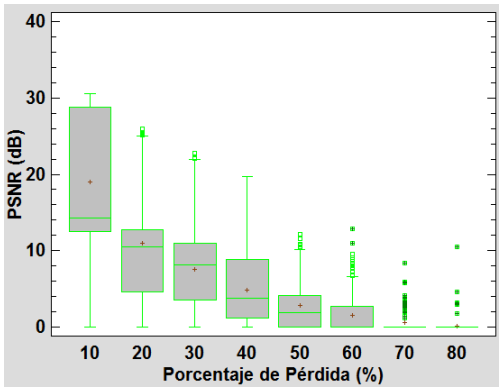
La tendencia de los resultados de los diagramas de la Figura 5.32 es muy parecida entre los diagramas 5.32a y 5.32c, que tienen como parámetros bloques de 1x1. La tendencia es prácticamente la misma al igual que los valores máximos y mínimos.

Lo mismo ocurre con los resultados para las simulaciones en bloques de 2x2, los diagramas 5.32b y 5.32d muestran gran similitud entre ellos, con una dispersión de resultados muy parecida.

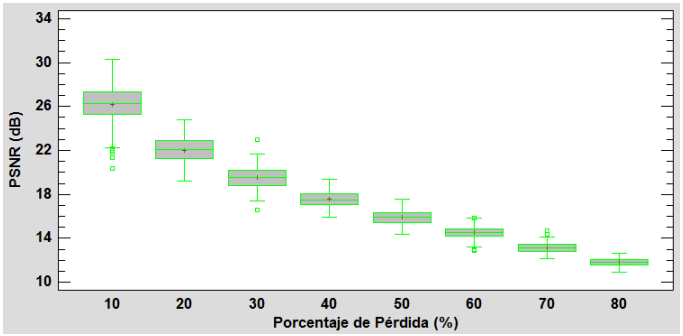
Comparando los resultados de las transmisiones en bloques de 1x1 con los resultados de las simulaciones en bloques de 2x2, se observa que en promedio la transmisión en bloques de 2x2 tiene resultados maximos con valores de PSNR hasta 9dB mayores a las simulaciones con bloques de 1x1. Esta brecha disminuye a medida que aumenta el porcentaje de pérdida de datos.

Si la transmisión en bloque de 2x2 demuestra tener mejor media que la de 1x1, lo mismo sucede con la transmisión en bloques de 8x8, el diagrama 5.32e muestra valores de PSNR de hasta 35.24341dB, siendo el máximo valor de todas los resultados de las simulaciones de la Figura 5.32.

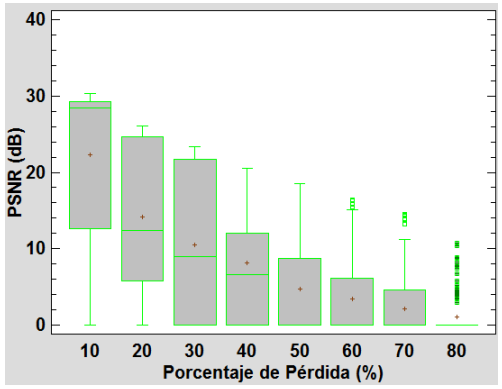
5.6.5. MSR



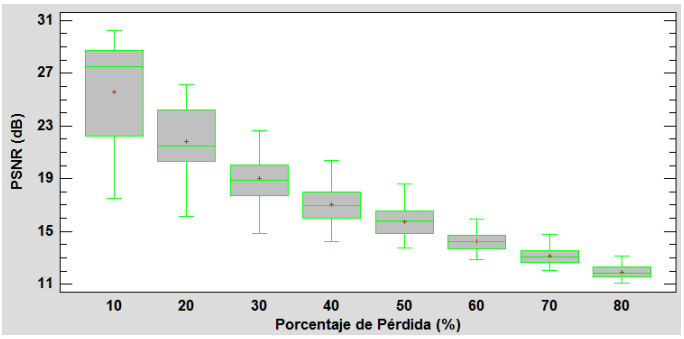
(a) block=1x1, byte TX=27



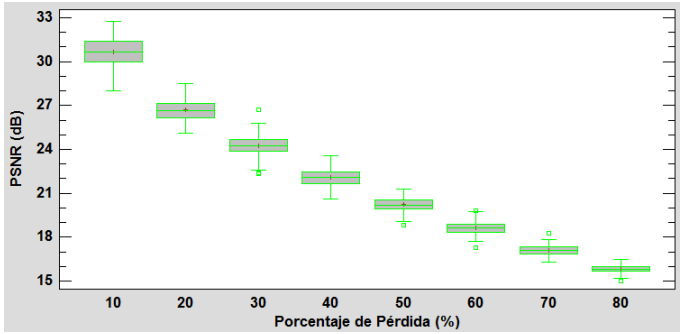
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.33: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment Maximal Smooth Recovery*

Los resultados de los diagramas 5.33a y subfig-1:corridormsrC, muestran resultados con restauraciones fallidas, entre estos dos diagramas se observa que la dispersión de datos para paquetes de 100 bytes, es casi el doble que en los resultados del diagrama de 27 bytes. Los

valores de la media, el máximo y mínimo tienen una tendencia muy similar a medida que el porcentaje de perdida de paquetes aumenta.

Para la transmisión en bloques de 2x2, se observa que en los diagramas 5.33b y 5.33d no hay restauraciones fallidas, eso indica el 100 % de las restauraciones realizadas de manera exitosa. La dispersión de resultados entre 10 % y 40 % de pérdidas de datos para el diagrama 5.33d es hasta tres veces mayor que la dispersión de datos del diagrama 5.33b bajo el mismo parámetro de análisis. A medida que el porcentaje de pérdida de datos aumenta, la dispersión de datos disminuye y se empareja con los resultados de el diagrama 5.33b.

Finalmente el diagrama 5.33e en donde el tamaño de bloque transmitido es de 8x8 y el paquete de 100 bytes, se observa que visualmente tiene una tendencia muy parecida al diagrama 5.33b, con una dispersión de datos prácticamente sin variación para cada porcentaje de pérdida. Sin embargo en la transmisión de bloques de 8x8 la media se encuentra aproximadamente por 6dB sobre la media del diagrama 5.33b.

5.7. Imagen Nº6: Motes 128x128

A continuación, para la imagen Motes de la figura 5.34 se tienen resultados para cada algoritmo de ocultamiento de error, paquete de dato y tamaño de bloque:

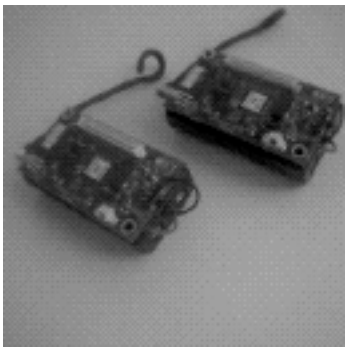
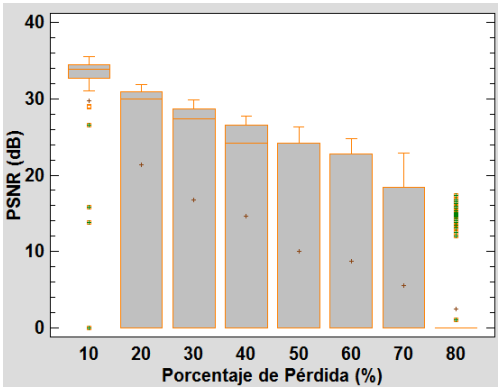


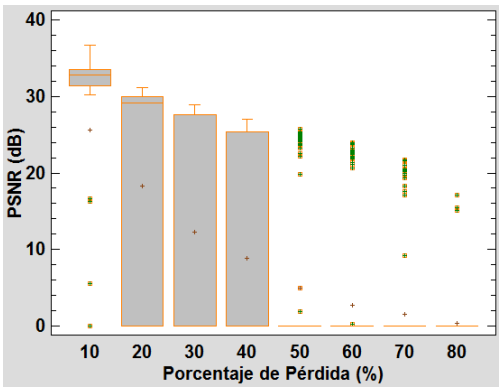
Figura 5.34: Motes 128x128.

En la figura 5.37 se aprecia como el algoritmo de reconstrucción Bilineal tienen una alta dispersión de datos a porcentajes de perdida bajo, con valores de PSNR muy altos lo cual no ocurre en las imágenes grandes de 512x512. El MSR nuevamente muestra un aumento en la dispersión de resultados (figura 5.39) como es la tendencia con las ultimas dos imágenes.

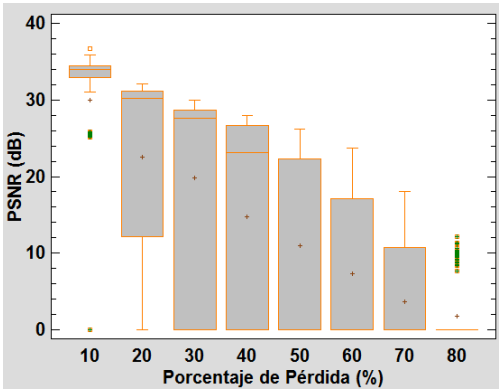
5.7.1. TV



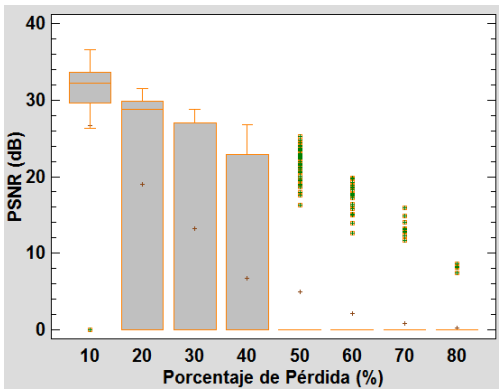
(a) block=1x1, byte TX=27



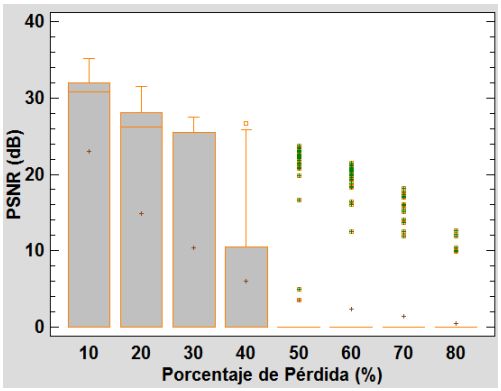
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.35: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de TV *inpainting*

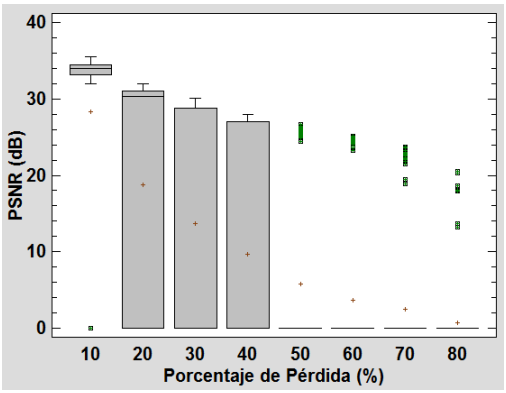
Los resultados que se observan en la Figura 5.35 tiene muy pocos cambios en comparación a los resultados del método de restauración TV sobre la imagen Corridor. Algunas diferencias son que para el 20 % de pérdida de datos en el diagrama 5.35b, hay una dispersión aproximadamente seis veces mayor en comparación a los resultados bajo los mismos parámetros de análisis sobre la imagen Corridor. Lo mismo ocurre para el mismo porcentaje de pérdida en bloques de 1x1 y paquetes de 100 %, en donde la dispersión de resultados es casi el triple que en la imagen Corridor.

Para el diagrama 5.35d y a perdida de datos del 50 % se aprecia una dispersión de resultados unas cinco veces menor a los resultados de la misma simulación sobre la imagen Corridor, además la mediana, para perdidas del 30 % en este caso se encuentra cerca de los 0dB y en la simulación sobre Corridor esta cerca de los valores máximos de PSNR.

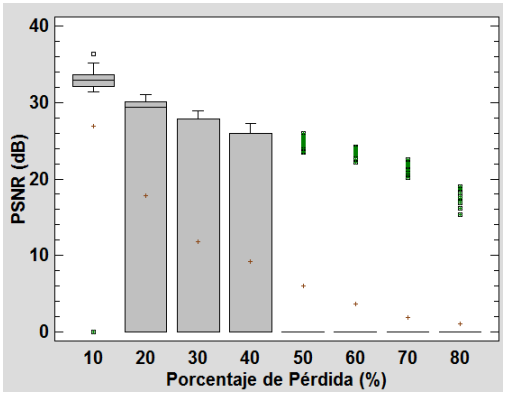
Para el diagrama 5.35e de bloques de 8x8 se aprecia que en porcentajes de pérdida del 10 % la dispersión de resultados es aproximadamente cuatro veces mayor a las simulaciones bajo los mismos parametros de analisis sobre la imagen Corridor. A 20 % de pérdida de paquetes la mediana esta muy cerca del valor mínimo de PSNR y por debajo de la media, por el contrario de los resultados de la simulación en la imagen Corridor, en donde la mediana rondaba los 10dB.

Para mas análisis sobre los datos se puede revisar los resultados de la imagen Corridor para el método TV, ya que tienen mucha semejanza en resultados. En comparación a las imágenes de tamaño mayor de este trabajo, la perdida de paquetes de datos se mantiene similar (revisar tablas anexas A.5).

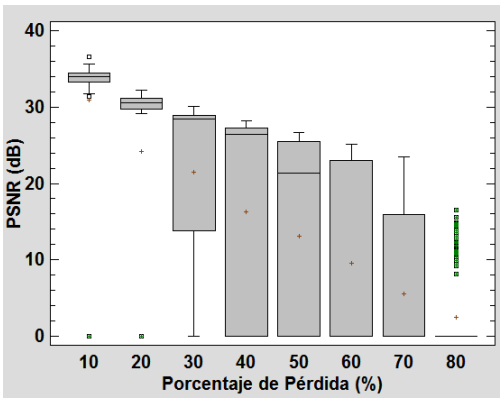
5.7.2. CDD



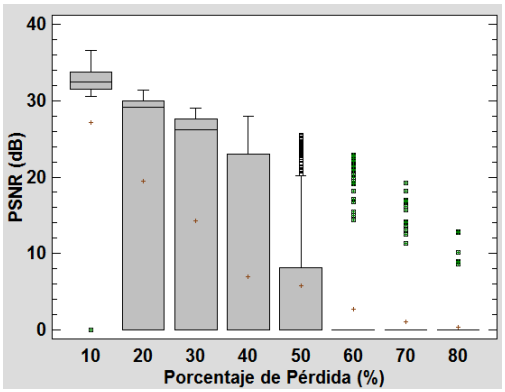
(a) block=1x1, byte TX=27



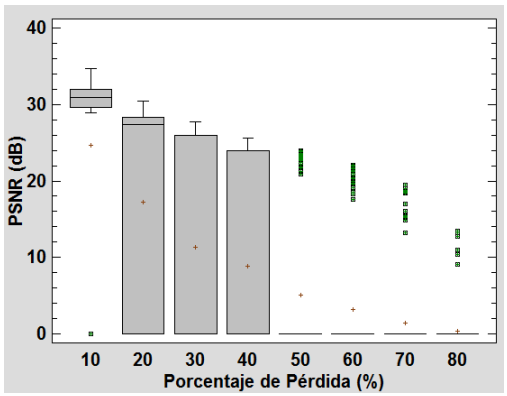
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100

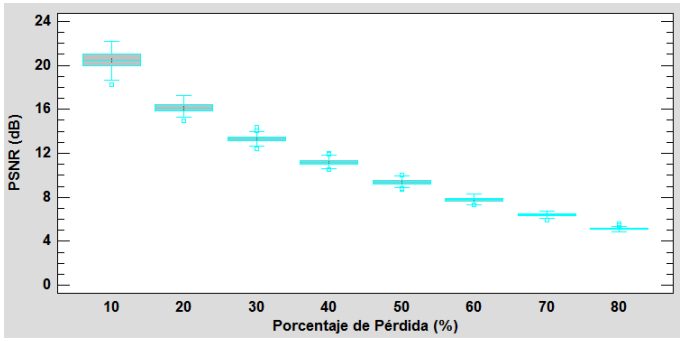


(e) block=8x8, byte TX=100

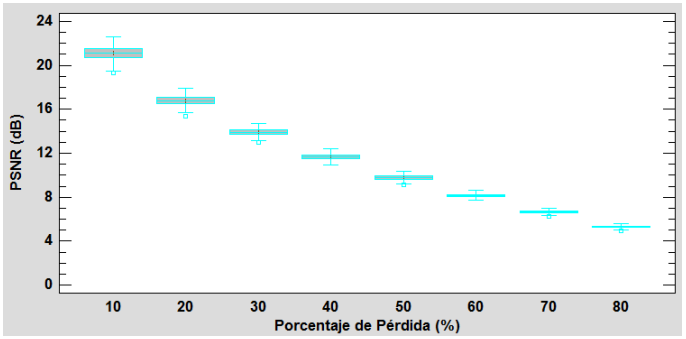
Figura 5.36: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de CDD *inpainting*

Los resultados de la Figura 5.36 son casi iguales a los resultados observados en la imagen Corridor para el método de restauración CDD. Tienen prácticamente la misma tendencia, mediana, media, máximos y mínimos.

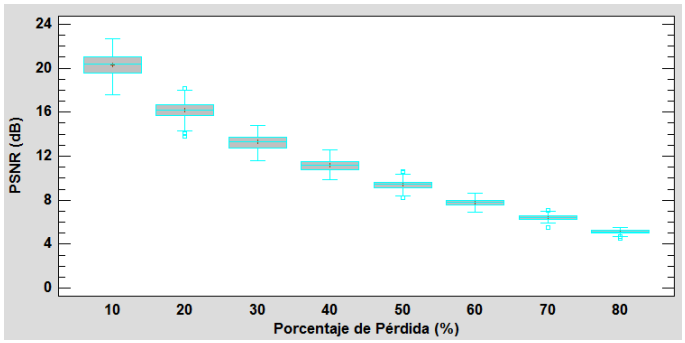
5.7.3. Bilineal



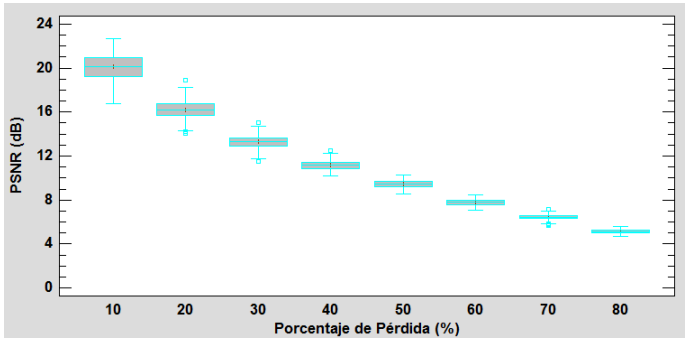
(a) block=1x1, byte TX=27



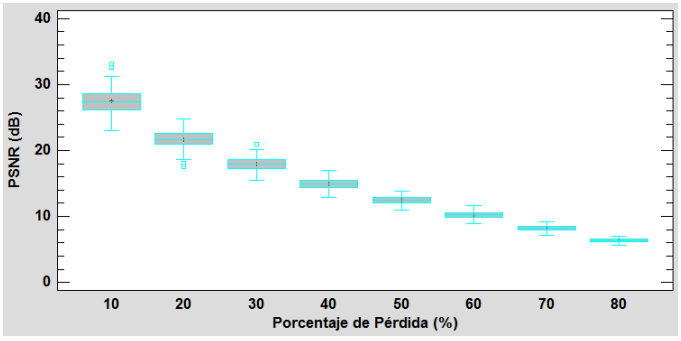
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100

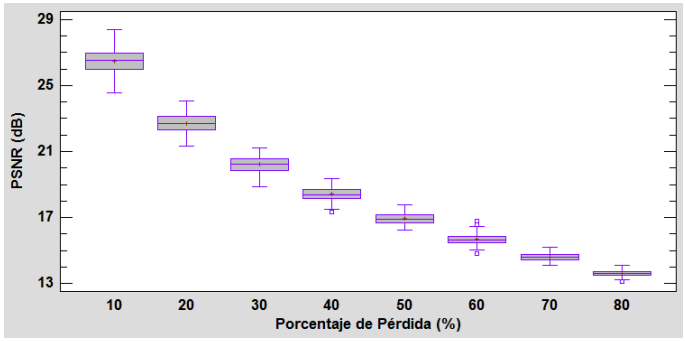


(e) block=8x8, byte TX=100

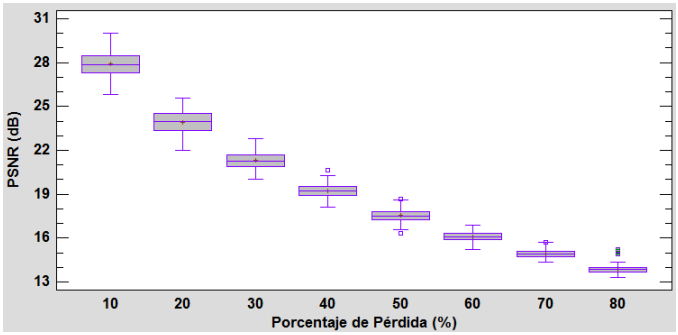
Figura 5.37: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bilineal

Los resultados del Bilineal observados en la Figura 5.37 muestra resultados casi calcados a los resultados arrojados en la simulación sobre Corridor bajo los mismos parámetros. Para un mayor análisis dirigirse a la Figura 5.31.

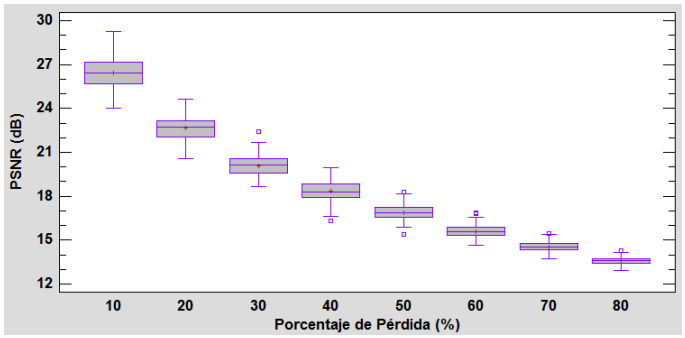
5.7.4. Bicúbico



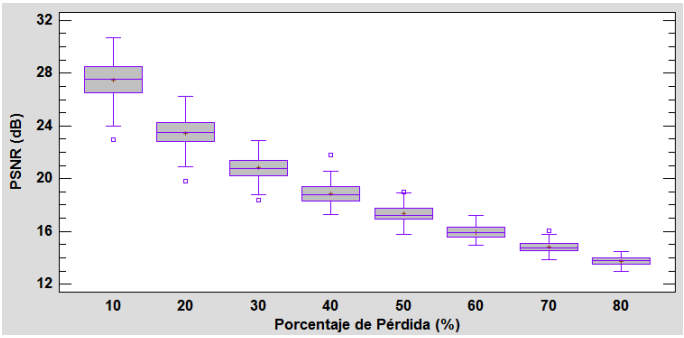
(a) block=1x1, byte TX=27



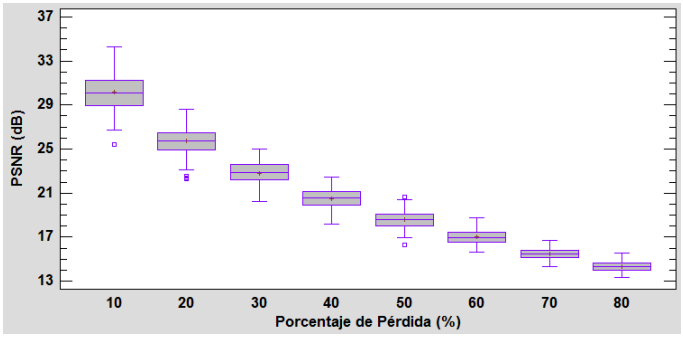
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100

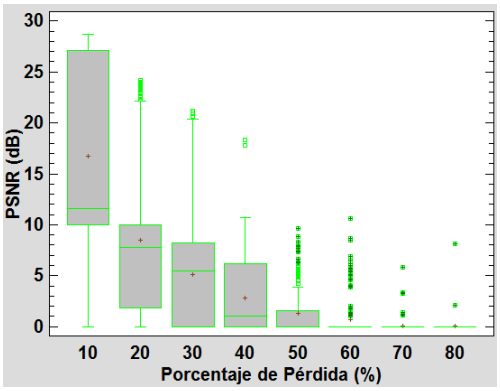


(e) block=8x8, byte TX=100

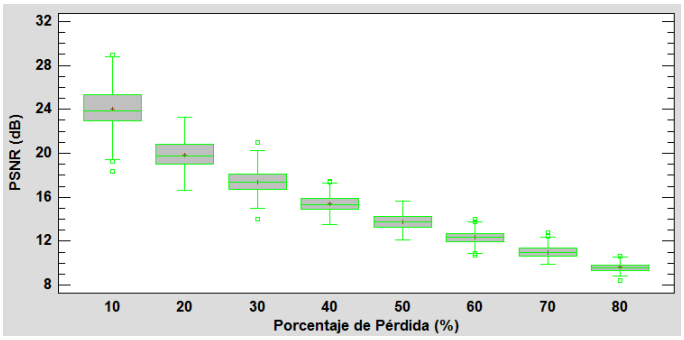
Figura 5.38: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment* Bicúbico

Se observa en los diagramas de la Figura 5.38 los resultados tienen una tendencia muy similar a los observados en la Figura 5.32 para la imagen Corridor. La única diferencia apreciable es que en los diagramas para bloques de 1x1, 2x2 y 8x8, la media esta aproximadamente 2dB por debajo de la media de los diagramas de la imagen Corridor.

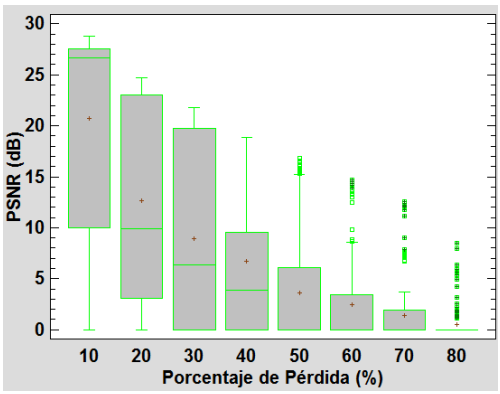
5.7.5. MSR



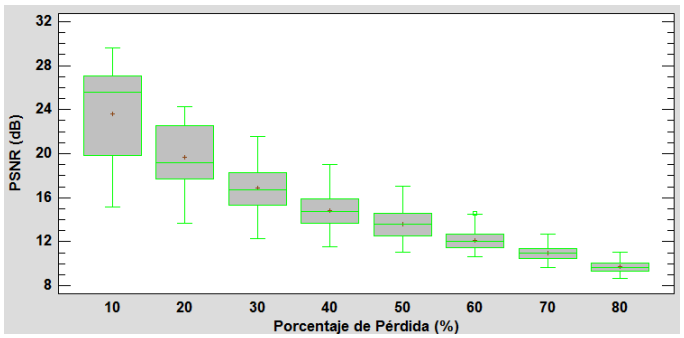
(a) block=1x1, byte TX=27



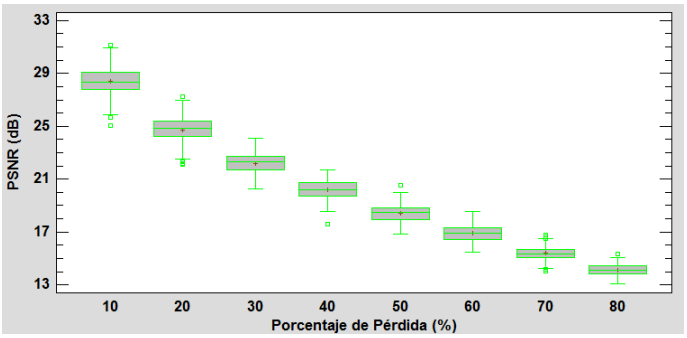
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 5.39: Gráficos de dispersión de los resultados para cada tamaño de bloque y paquete de datos para el algoritmo de *error concealment Maximal Smooth Recovery*

La tendencia a mantener los resultados de la imagen Corridor se mantienen, los valores medios de todos los diagramas esta aproximadamente 1dB por debajo de la media de los diagramas del MSR sobre Corridor. para un análisis mayor dirigirse a la Figura 5.33.

Capítulo 6

Análisis de resultados

A continuación se procede al análisis de los resultados obtenidos con mayor detalle, se utilizo la herramienta StatPlus para obtener estadísticas de los resultados obtenidos en las simulaciones las cuales se tabularon en el Anexo A. Los resultados se exponen en gráficos con curvas de la media, debido a que permiten observar el comportamiento promedio de los resultados y el nivel con respecto a otros métodos de reconstrucción de imágenes. Los gráficos se agrupan por imagen, lo que significa que por cada imagen habrá cinco gráficos de la media, uno por cada tamaño de bloque y paquete de datos, además cada gráfico tendrá cinco etiquetas, para identificar cada método de restauración de imagen, como se observa en la Figura 6.1 para analizar de manera ordenada los resultados.

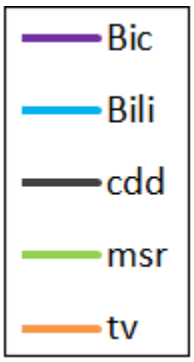
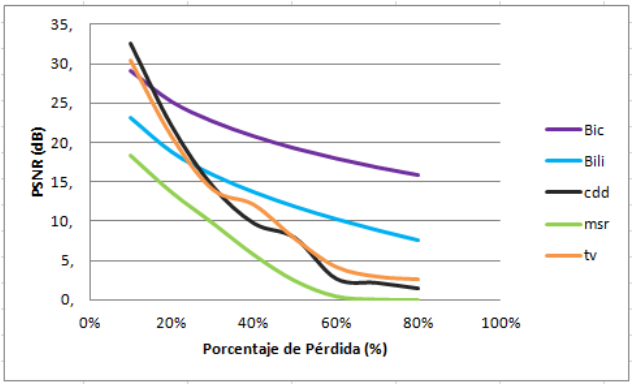
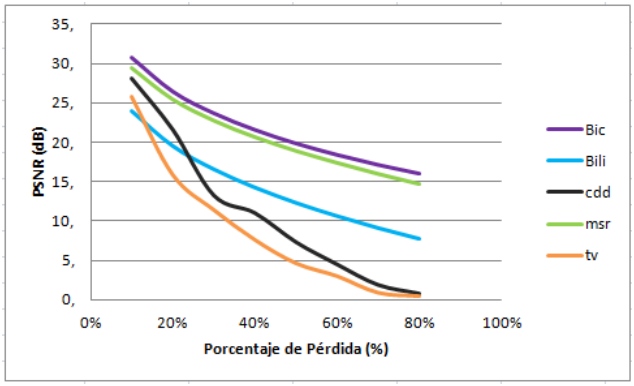


Figura 6.1: Etiquetas para cada algoritmo de reconstrucción en los graficos de comparación.

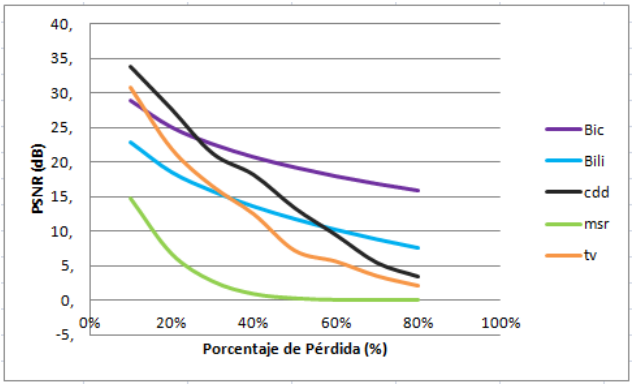
6.1. Comparación de métodos sobre imagen Lena



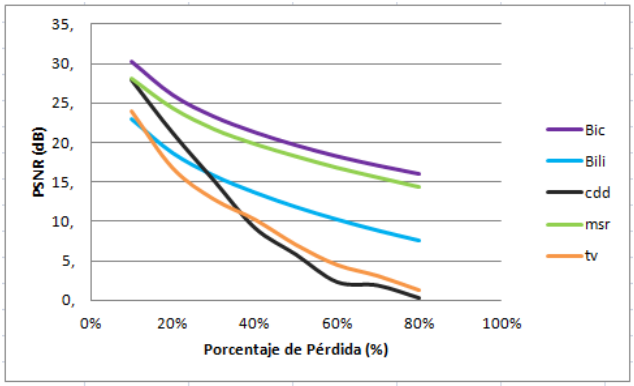
(a) block=1x1, byte TX=27



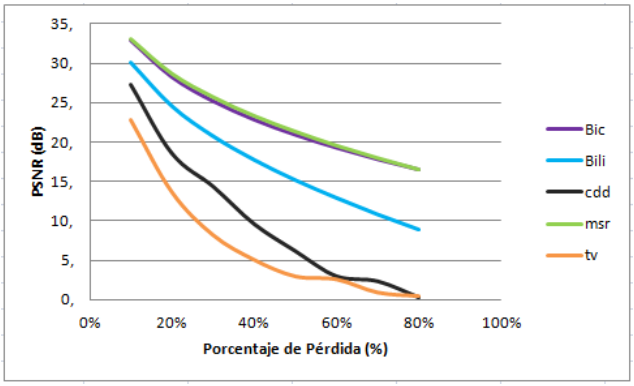
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 6.2: Comparación entre los cinco métodos utilizados en la reconstrucción de la imagen Lena de 512x512 con distintos parámetros de transmisión

De la a imagen Lena de 512x512 se obtienen las siguientes observaciones:

Observando cada gráfico se observa que el algoritmo de reconstrucción Bicúbico (que según los diagramas de caja tenia muy poca dispersión en los resultados), en los cinco casos, tiene un comportamiento muy constante en los resultados. Sus valores de PSNR fluctúan entre 30 y 15 dB en gran parte de los ensayos según las tablas estadísticas A.1.

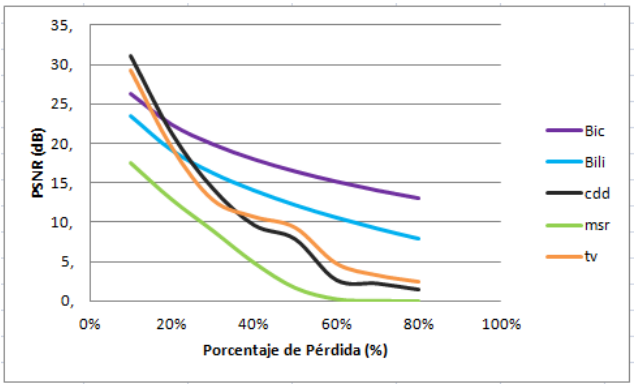
El Algoritmo de reconstrucción Bilineal, tiene un comportamiento muy parecido al Bicúbico, sin embargo los valores medios en los que ronda son muy por debajo, teniendo como promedio valores de PSNR entre 23 y 7 (A.1).

El CDD tiene en promedio el PSNR mas alto de todos los métodos para porcentajes de pérdida del 10 % en bloques de 1x1 que se utilizaron sobre esta imagen, llegando inclusive a tener un valor de PSNR de 39.05 (A.1), sin embargo se tiene mas de dispersión de datos que el Bicúbico, además, no se realizaron el 100 % de las pruebas exitosas, arrojando resultados erróneos con valores de PSNR cercanos a 0dB, los que iban en aumento a medida que aumentaba la perdida de datos en la transmisión. A medida que aumenta la pérdida de datos, la tendencia del promedio del PSNR cae bruscamente.

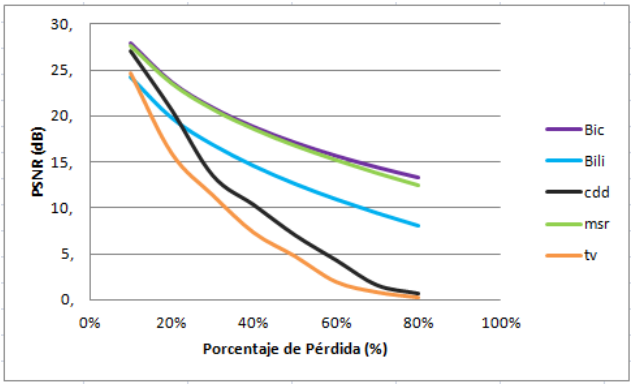
Con el MSR ocurre algo singular, en los gráficos 6.2a y 6.2c de bloques de 1x1 se observa que su curva está por debajo de las otras, esto debido a la gran cantidad de datos dispersos, algunas de las simulaciones inclusive fueron erróneas como se observa en las tablas anexadas A.16 y A.18, sin embargo, para bloques de 2x2 y 8x8 la dispersión es muy poca, dando como resultado una curva muy parecida y cercana al Bicúbico e incluso mejor, como se aprecia en la Figura 6.2e para bloques de 8x8.

Reconstrucción mediante TV tiene los valores medios de PSNR mas altos junto con el CDD a porcentajes de perdida del 10 % sobre bloques de 1x1, y comparten la cantidad de errores que se presentan a medida que aumenta la perdida de paquetes de datos. Tiene también una dispersión en los resultados muy grande, que se puede observar en los gráficos del capitulo anterior.

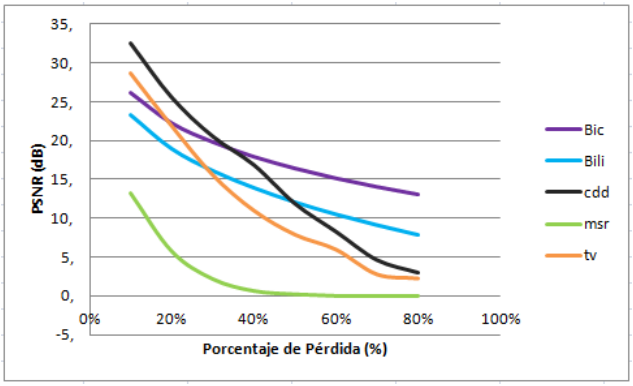
6.2. Comparación de métodos sobre imagen Baboon



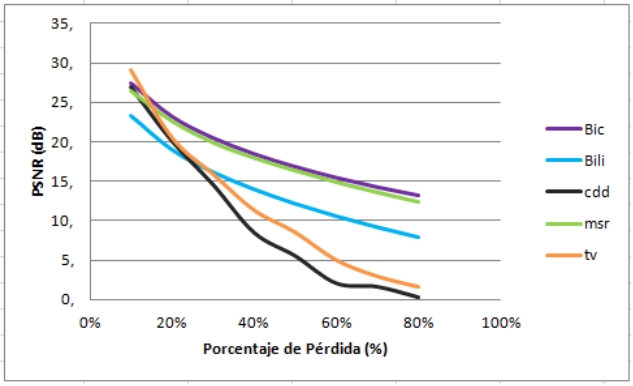
(a) block=1x1, byte TX=27



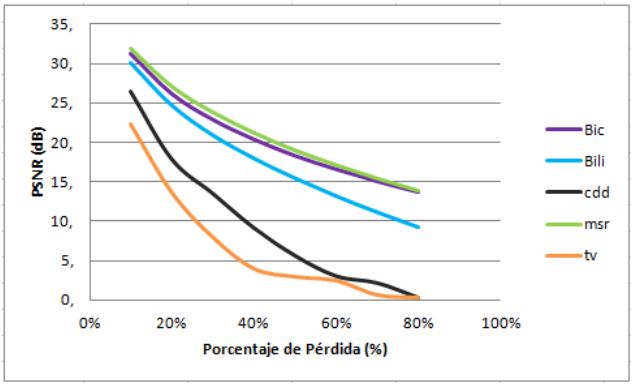
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 6.3: Comparación entre los cinco métodos utilizados en la reconstrucción de la imagen Baboon de 512x512 con distintos parámetros de transmisión

De la a imagen Baboon de 512x512 se obtienen las siguientes observaciones:

La reconstrucción mediante algoritmo Bicúbico muestra una buena linealidad en comparación a las demás, la dispersión de resultados es muy baja independiente del tamaño de los bloques o de los paquetes de datos utilizados, en promedio la linea de tendencia se varia muy poco en los cinco gráficos de la figura 6.3. El valor medio del PSNR cambia muy poco dentro de cada porcentaje de perdida, manteniéndose siempre sobre 26 dB en las simulaciones para porcentajes de pérdida del 10 %, y sobre 13 dB para porcentajes de perdida del 80 % (A.2).

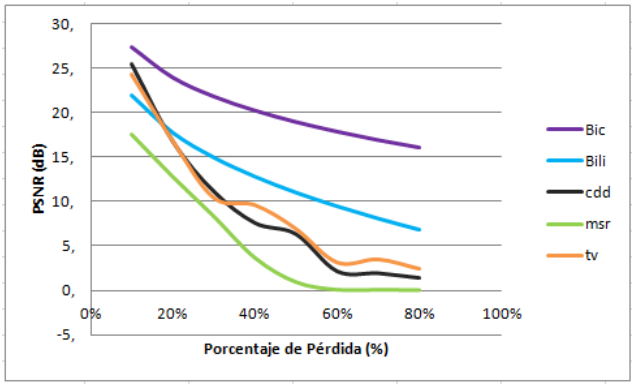
Los resultados de simulaciones para el algoritmo Bilineal, son al igual que el Bicúbico con muy poca dispersión de resultados y tienen la misma forma de tendencia, sin embargo el valor medio para la misma pérdida de paquetes, es muy bajo en comparación al Bicúbico. En los gráficos 6.3a y 6.3c entrega mejores resultados que el MSR, pero en el resto de las gráficas, es el de peor desempeño dentro de los métodos de *error concealment*.

Dejando la cantidad de resultados erróneos a un lado y dejando solo los resultados coherentes, el CDD tiene los resultados con mejores valores medios en cada simulación (A.2). Se observa en los gráficos de la figura 6.3 que debido a la gran cantidad de restauraciones erróneas, la curva promedio del CDD queda por debajo de los métodos de *error concealment*.

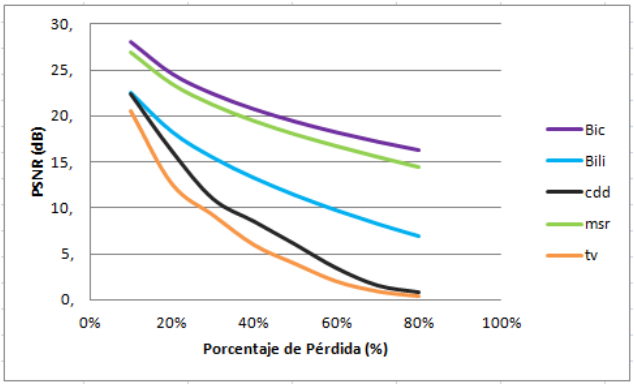
El algoritmo de reconstrucción mediante MSR entrega resultados muy parecidos a los del Bicúbico en los gráficos 6.3b y 6.3d, inclusive en la gráfica 6.3e los valores medios son mejores que el bicúbico. También se observa una tendencia mas lineal, sin embargo cuando el tamaño de los bloques a enviar son a nivel de pixel (6.3a y 6.3c), el MSR falla, arrojando una diversa cantidad de resultados muy dispersos y erróneos, en las tablas de estadísticas anexas se observa el porcentaje de resultados fallidos A.2.

Los valores de PSNR máximos que arroja la restauración mediante TV son muy altos, muy parecidos al del CDD, con una tendencia del promedio muy parecida y errática debido a los valores cercanos a 0dB, entregando para bloques de 1x1 transmitidos en paquetes de 27 bytes una curva de la media muy errática en comparación a los demás gráficos.

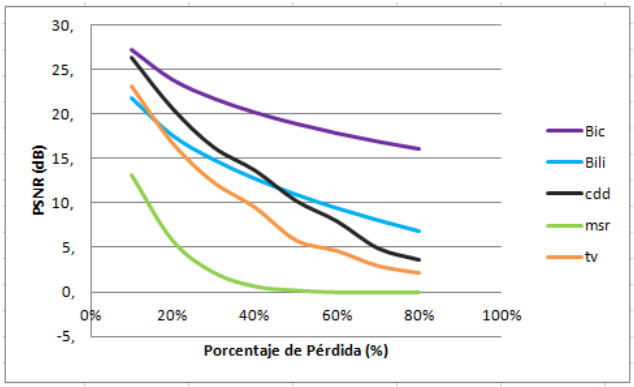
6.3. Comparación de métodos sobre imagen Peppers



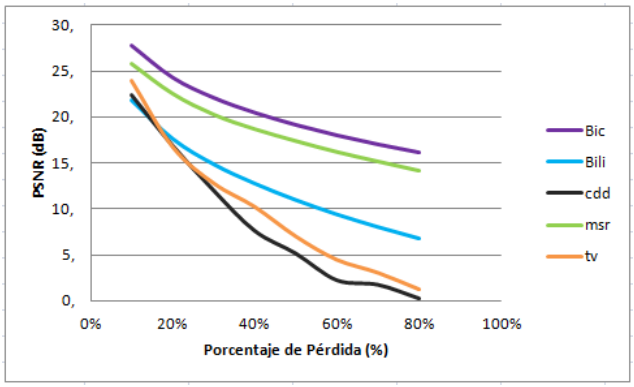
(a) block=1x1, byte TX=27



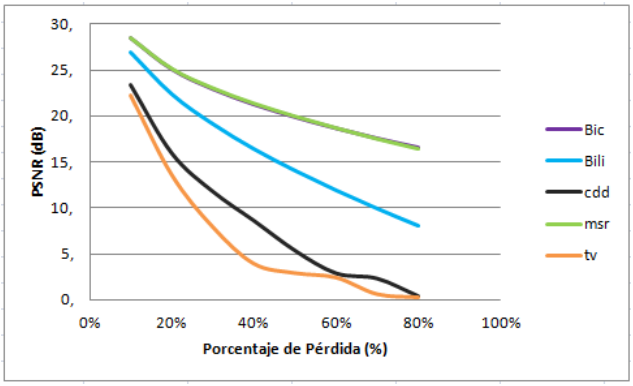
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 6.4: Comparación entre los cinco métodos utilizados en la reconstrucción de la imagen Peppers de 512x512 con distintos parámetros de transmisión

De la a imagen Peppers de 512x512 se obtienen las siguientes observaciones:

En esta imagen el comportamiento no es diferente de las dos imágenes anteriores, el algoritmo Bicúbico sigue teniendo un comportamiento constante en todos los casos, con valores medios mejores que el Bilineal y el MSR (excepto en el gráfico 6.4e).

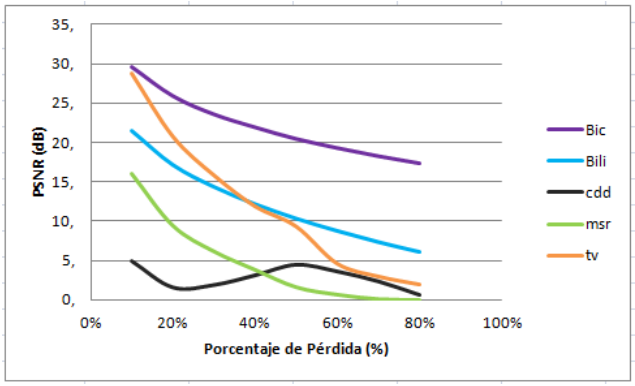
Los resultados para el Bilineal mantienen su tendencia paralela al Bicúbico, solo que ahora la brecha que se observa en todos los gráficos, entre los valores promedio del Bilineal y el Bícubico, es mayor.

El CDD mantiene su tendencia de obtener resultados con PSNR altos y con poca dispersión a 10 % de pérdida de datos, a excepción de los resultados para pérdidas del 80 % de los paquetes de datos, en donde siempre tiene un decaimiento con respecto a sus pares.

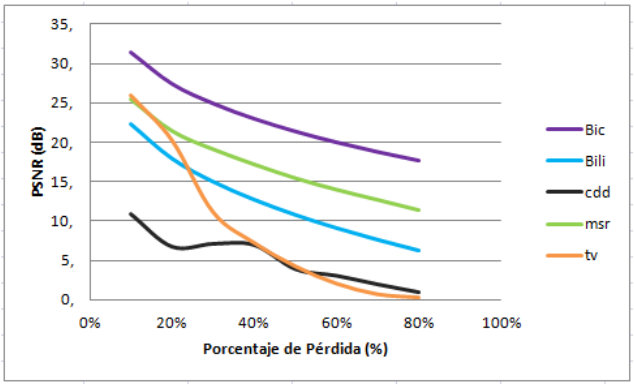
El algoritmo de MSR muestra la misma dispersión de valores en los gráficos de bloques de 1x1 agregando un poco mas de datos dispersos a la simulación con bloques de 2x2 y paquetes de 100 bytes, observado en el gráfico 6.4d. Para bloques de 2x2 se observa que el MSR mantiene una curva paralela a la del Bicúbico, por otra parte para bloques de 8x8 se solapan.

Por su parte en el TV *inpainting*, se observa dispersión de datos en todos los gráficos de la figura 6.4 que no varia mucho a las imágenes anteriores al igual que el porcentaje de simulaciones erróneas observadas en las tablas de estadísticas indexadas A.3. La tendencia de la media es muy parecida, con muchos valores de 0dB.

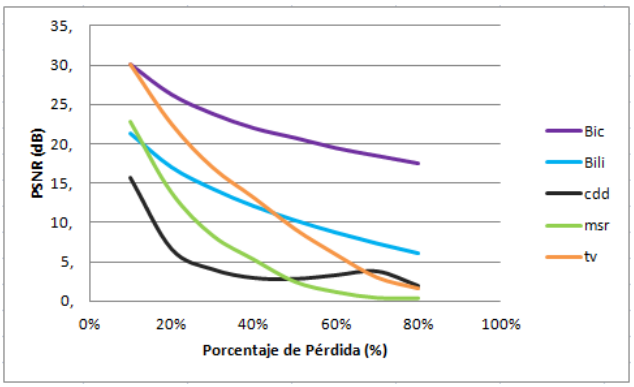
6.4. Comparación de métodos sobre imagen Bird



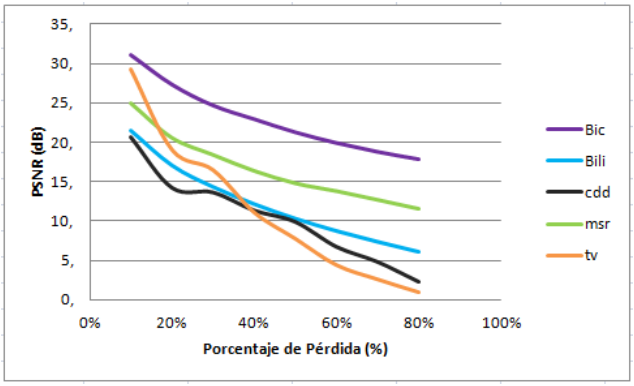
(a) block=1x1, byte TX=27



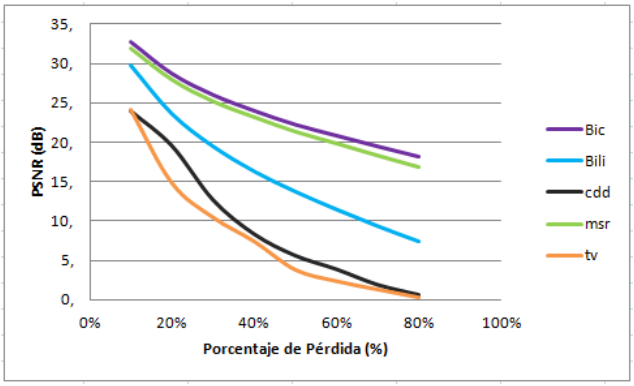
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 6.5: Comparación entre los cinco métodos utilizados en la reconstrucción de la imagen Bird de 200x200 con distintos parámetros de transmisión

De la a imagen Bird de 200x200 se obtienen las siguientes observaciones:

La restauración de la imagen mediante el algoritmo Bicúbico, arrojó resultados en términos medios, ya que los valores promedios se encuentran por debajo de los valores máximos de PSNR entregados por el CDD y el TV. La línea de tendencia se mantiene por sobre los valores medios de los algoritmos de *inpainting* a pesar de haber cambiado a una imagen mas pequeña (200x200). En las gráficas de las sub-imágenes 6.5b y 6.5d las cuales utilizan bloques de 2x2, se observa una leve alza en el valor del PSNR medio con respecto a los gráficos de dispersión de bloques de 1x1, así mismo, los resultados para la transmisión en bloques de 8x8 fueron superiores a los de 1x1 y 2x2.

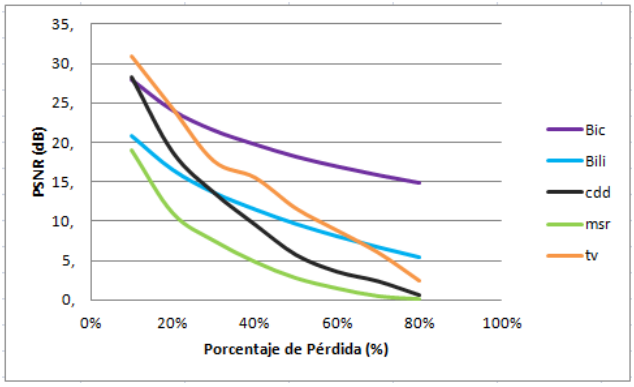
El algoritmo de reconstrucción Bilineal se mantuvo con valores de PSNR bajos en comparación a los demás algoritmos, excepto en la transmisión en bloques de 8x8, en donde para una pérdida de 10 % tiende a igualar a todos los demás métodos. Aun así la tendencia de la media, al igual que el Bicúbico es mejor que el CDD y TV. También se observó en el capítulo anterior, que para porcentajes de perdidas de paquetes de datos bajos, la dispersión de los resultados es mayor, y esta disminuye a medida que aumenta el porcentaje de perdida.

El CDD en general sigue siendo el algoritmo con mejor evaluación de valores máximos de PSNR, sin embargo de todas las simulaciones, muchas siguen siendo erróneas, lo que provoca que la tendencia de la media sea muy mala en comparación a los algoritmos de *error concealment*.

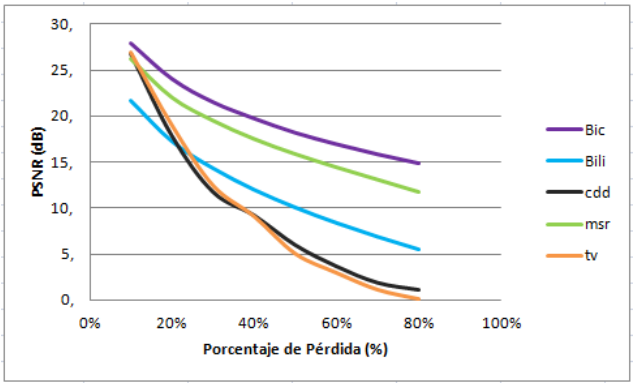
El MSR Nuevamente sigue muy de cerca al Bicúbico, teniendo el mismo análisis que en las imágenes anteriores.

Finalmente el algoritmo TV al igual que el CDD y MSR presenta mayor dispersión de resultados en comparación a las imágenes de mayor tamaño a bloques de 1x1. Es el peor de los algoritmos en general, con mucha dispersión y poca cantidad de simulaciones realizadas de manera exitosa.

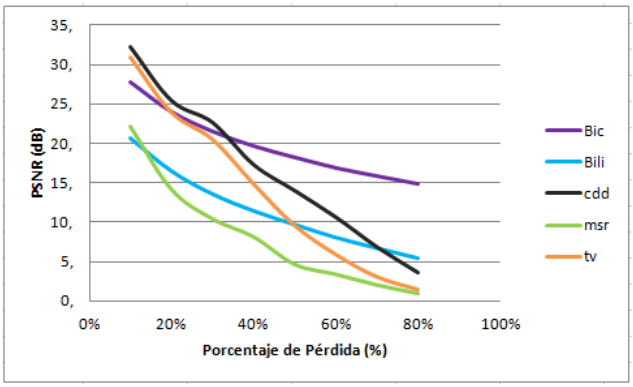
6.5. Comparación de métodos sobre imagen Corridor



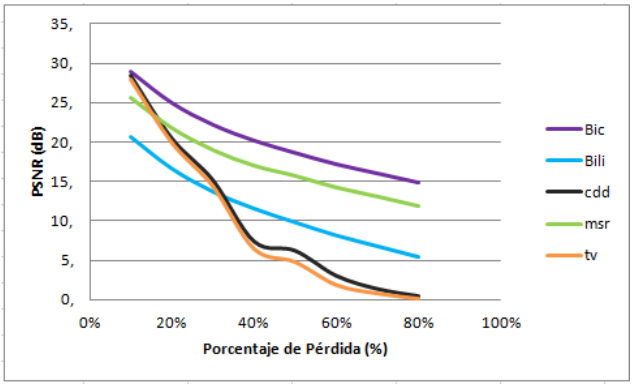
(a) block=1x1, byte TX=27



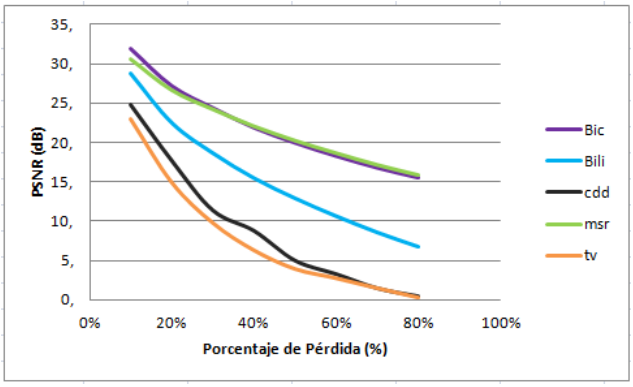
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 6.6: Comparación entre los cinco métodos utilizados en la reconstrucción de la imagen Corridor de 128x128 con distintos parámetros de transmisión

De la a imagen Corridor de 128x128 se obtienen las siguientes observaciones:

La restauración por parte del algoritmo Bicúbico al igual que en la imagen Bird, tiene una media de resultados más alta a medida que el tamaño de bloque de transmisión es mas grande. El gráfico de la sub-figura 6.6e muestra que hasta el 40 % de perdida de paquetes de datos, los valores medios de PSNR se ubicaron por sobre los demás gráficos de la figura 6.6. También se observa que a partir de bloques de 2x2 éste método ofrece mejores resultados desde el 10 % en comparación a los métodos de *inpainting*.

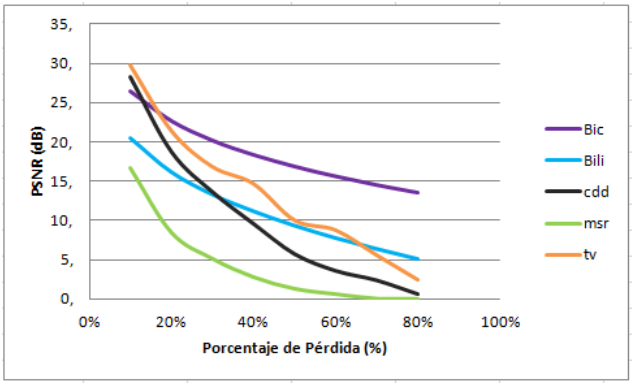
Bilineal nuevamente es el peor evaluado de todos los métodos de *error concealment*, solo tiene una leve reivindicación en el gráfico de la sub-figura 6.6e, en el que se observa como intenta igualar el valor medio de los demás métodos cuando la perdida es del 10 %, pero al mismo tiempo muestra una gran dispersión de valores según los gráficos del capítulo 5.

CDD *inpainting* con su gran dispersión de valores en comparación a las imágenes de 512x512, sigue liderando como el método con valor de PSNR máximo más alto puesto que ocupa junto al TV.

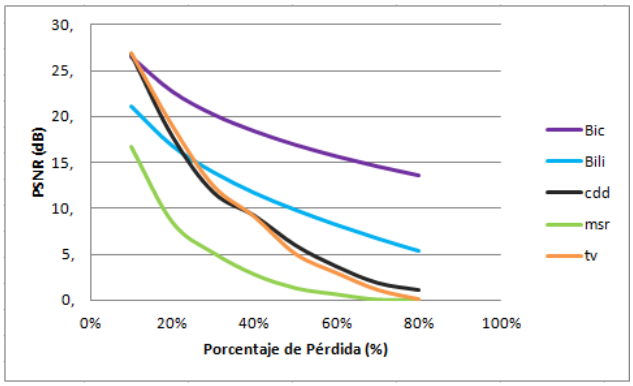
Al parecer, la reconstrucción mediante MSR solo muestra resultados competitivos en comparación a los demás métodos, cuando la transmisión de imágenes se realiza en bloques de 8x8, el gráfico de la sub-figura 6.6e muestra resultados muy compactos en cada porcentaje de perdida, tiene una tendencia muy parecida al Bicúbico, a tal punto que se solapan. Por otro lado en los demás gráficos aparece con una tendencia a fallar cuando se transmiten imágenes a nivel de pixel con promedios muy parecidos a los del CDD y TV.

La reconstrucción mediante TV no tiene en este caso, nada nuevo para analizar, sigue arrojando una gran cantidad de simulaciones erróneas con mucha dispersión de valores y con una linea de tendencia media por debajo del CDD.

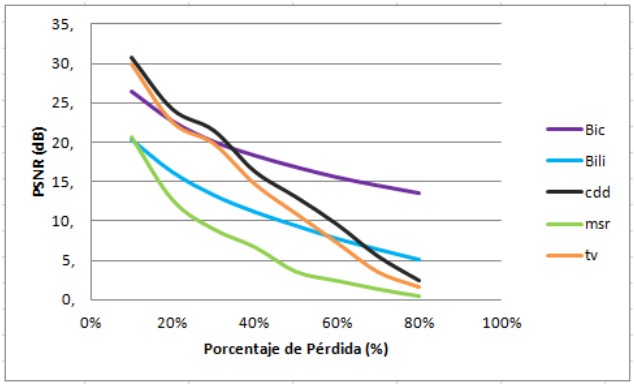
6.6. Comparación de métodos sobre imagen Motes



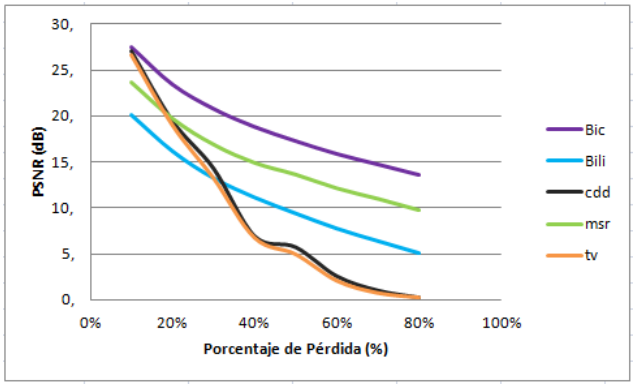
(a) block=1x1, byte TX=27



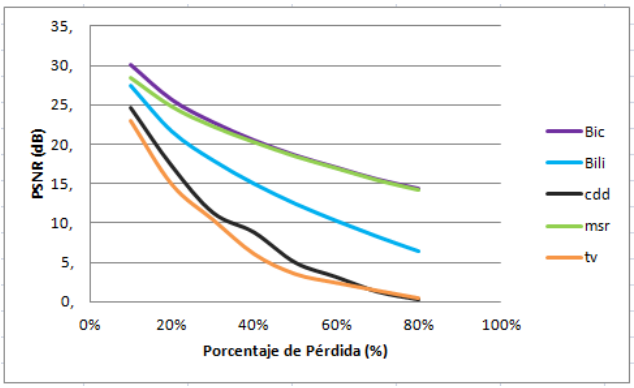
(b) block=2x2, byte TX=27



(c) block=1x1, byte TX=100



(d) block=2x2, byte TX=100



(e) block=8x8, byte TX=100

Figura 6.7: Comparación entre los cinco métodos utilizados en la reconstrucción de la imagen Motes de 128x128 con distintos parámetros de transmisión

De la a imagen Motes de 128x128 se obtienen las siguientes observaciones:

En general son los mismos resultados de las imágenes anteriores, independiente del tamaño de estas, los gráfico muestran tendencias de comportamiento similares, variando solo los valores máximos y mínimos.

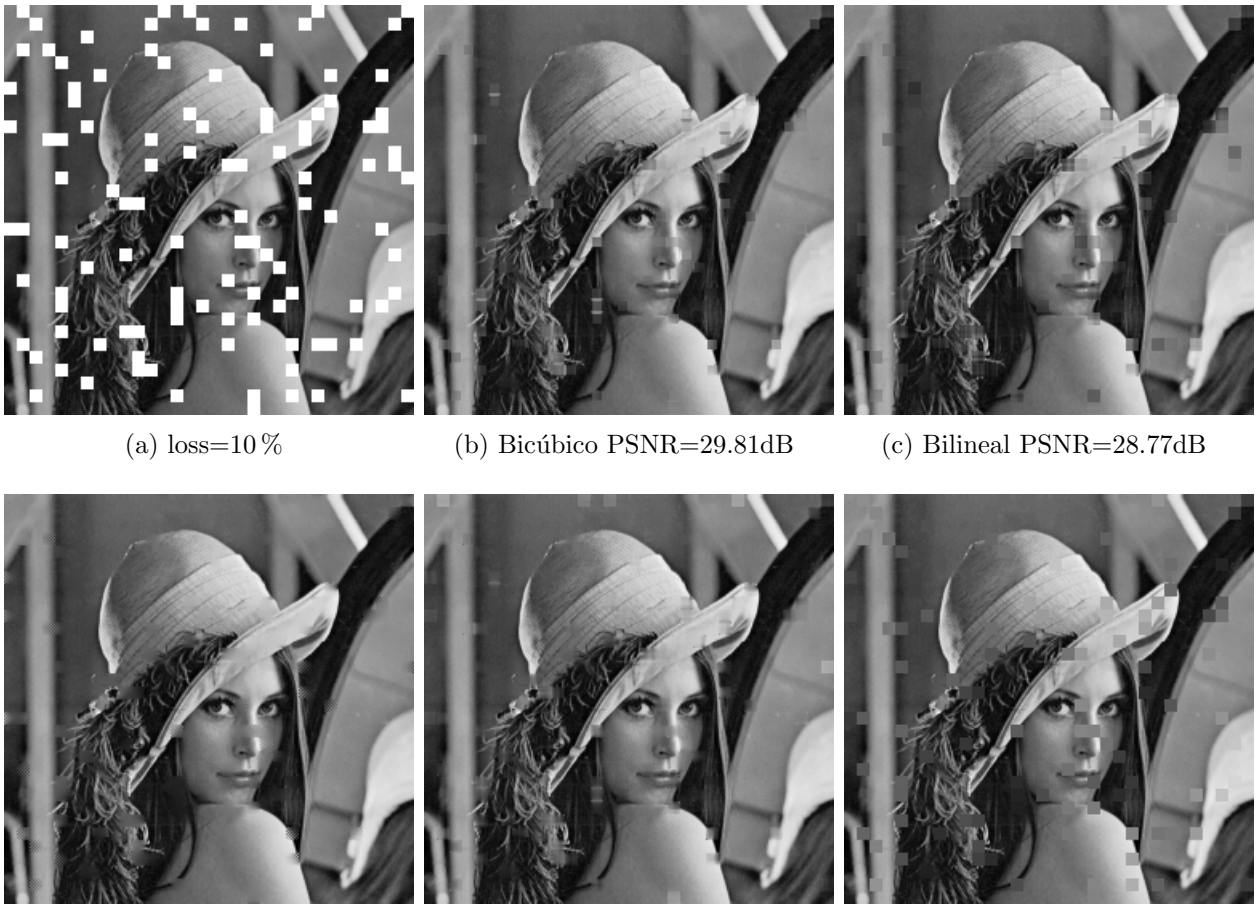
6.7. Análisis y comentarios

Los análisis anteriores a las seis imágenes de prueba permiten sacar algunas conclusiones con respecto a las simulaciones, por ejemplo en todas las imágenes el método Bicúbico mostró ser el de menos dispersión en cualquier porcentaje de perdida, por otro lado el CDD y el TV son los algoritmos que tienen en promedio mejor PSNR pero traen consigo una gran cantidad de simulaciones fallidas que entregan como resultado PSNR cercanos a 0dB, esto puede deberse a la gran cantidad de funciones no lineales por las que se componen los algoritmos provocando que al recorrer la imagen realizando el *inpainting* no siempre entregue un valor coherente, viéndose muy afectado al momento de recuperar bloques perdidos en los bordes o al centro de otros bloques perdidos, además el TV en si tiene una dispersión de valores mucho mayor que el CDD, esto demuestra la ventaja del CDD con respecto al TV al contar con el coeficiente de difusión dentro de sus lineas de código. El MSR demostró una gran eficacia con respecto al Bicúbico y el Bilineal en imágenes grandes (512x512) cuando se transmitía en bloques de 8x8, esto se puede explicar debido a que el MSR trabaja en cuadrantes, en bloques de 1x1 no tiene cuadrantes, por eso entregaba tantos resultados distintos y erróneos, en bloques de 2x2 solo tiene un pixel por cuadrante por lo que no se desempeña completamente sin embargo en bloques de 8x8 tiene cuatro pixeles por bloque, es por eso que en algunos casos supera al Bicúbico. Por ultimo el método de restauración Bilineal fue al igual que el Bicúbico muy constante en todas las simulaciones, con un promedio de PSNR en general bajo comparados con los algoritmos de *inpainting* pero con muy poca dispersión de datos, como se vio en la sección 3.3 del capítulo 5, es un método muy simple por lo que no se pueden esperar mejores resultados.

Para analizar el desempeño de los algoritmos de manera visual, se realizaron simulaciones con una imagen aleatoria, en este caso lena, se decidió transmitir en bloques de 8x8 sobre paquetes de datos de 100 bytes, ya que, como se mencionó anteriormente, la mayoría de los algoritmos mantiene la tendencia de sus resultados a excepción del MSR el cual presenta una mejora con los parámetros mencionados. Se decidió presentar los resultados para perdidas

del 10 %, 50 % y 80 %, ya que para resultados mas detallados se pueden observar las gráficas de la sección anterior

los resultados de estas simulaciones pueden ser observados en las figuras 6.8, 6.9 y 6.10. En primer lugar si se observan detenidamente los resultados cuando la perdida es de 10 %, visualmente se aprecia que el CDD es quien mejor reconstruye la imagen seguido del MSR, el Bicúbico también da como resultado una imagen suave pero no tanto como el MSR, por otro lado si se pone atención a la sub-figura 6.8a se puede observar que cuando se pierden paquetes de datos contiguos, se genera una especie de horizonte entre los dos bloques reconstruidos, con el MSR sucede los mismo, con la diferencia que en este caso el suavizado permite que el detalle sea difuminado.



(a) loss=10 %

(b) Bicúbico PSNR=29.81dB

(c) Bilineal PSNR=28.77dB

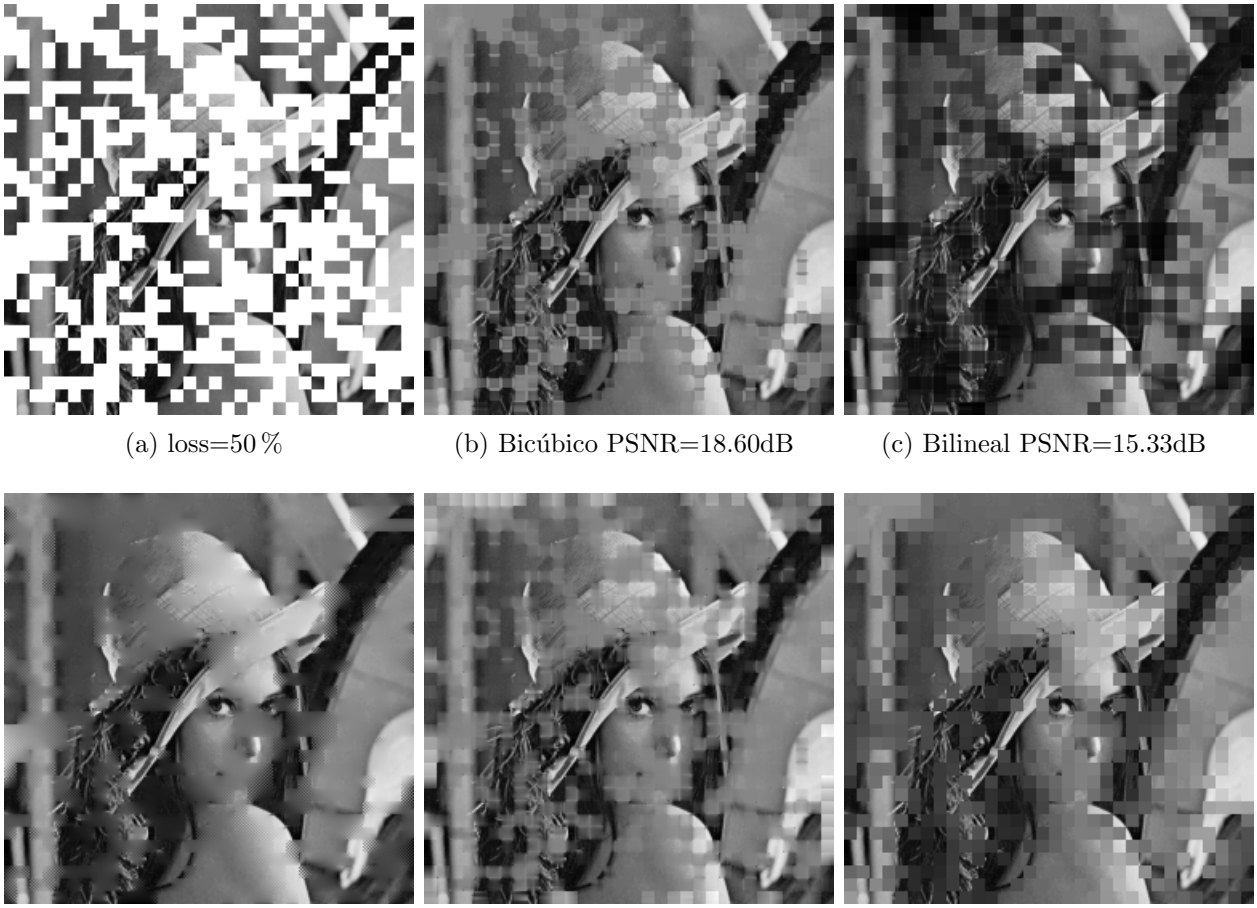
(d) CDD PSNR=30.34dB

(e) MSR PSNR=30.15dB

(f) TV PSNR=27.06dB

Figura 6.8: Restauración con distintos algoritmos de imagen lena de 512x512 transmitida en bloques de datos de 8x8 sobre paquetes de 100 bytes con 10 % de perdida de paquetes de datos

Cuando la perdida de paquetes de datos es de un 50 % como en la figura 6.9, cuesta identificar la imagen que esta detrás la de la sub-figura 6.9a, sin embargo con un poco de imaginación y el algoritmo de *inpainting* CDD, podemos ver que es lena con un PSNR de 21.08, siendo en esta ocasión, nuevamente el mejor evaluado, visualmente recupera de manera suave los tonos, además se observa que distingue cuando pasa bruscamente de un tono fuerte a uno mas débil y viceversa, como por ejemplo en los bordes del sombrero o en la estructura de fondo. Si observamos el Bicúbico y el MSR, se sobresale la suavidad de la reconstrucción de este último con respecto al Bicúbico, aun así su PSNR es menor, por otra parte la sub-figura 6.9d no recupera los bloques del los bordes como en la sub-figura 6.9b lo que en parte explicaría el la diferencia entre los indices de calidad. Lo mismo ocurre con el TV *inpainting* de la sub-figura 6.9f, que tiene un PSNR mayor a la imagen restaurada por MSR sin embargo visualmente la imagen 6.9d esta mas suavemente reconstruida, por otro lado la imagen 6.9f a pesar de no tener bloques suaves, estos están mas en armonía con los pixeles que los rodean. El bilineal tiende a fallar reconstruyendo una imagen con manchas oscuras, bastante distantes de los demás algoritmos.

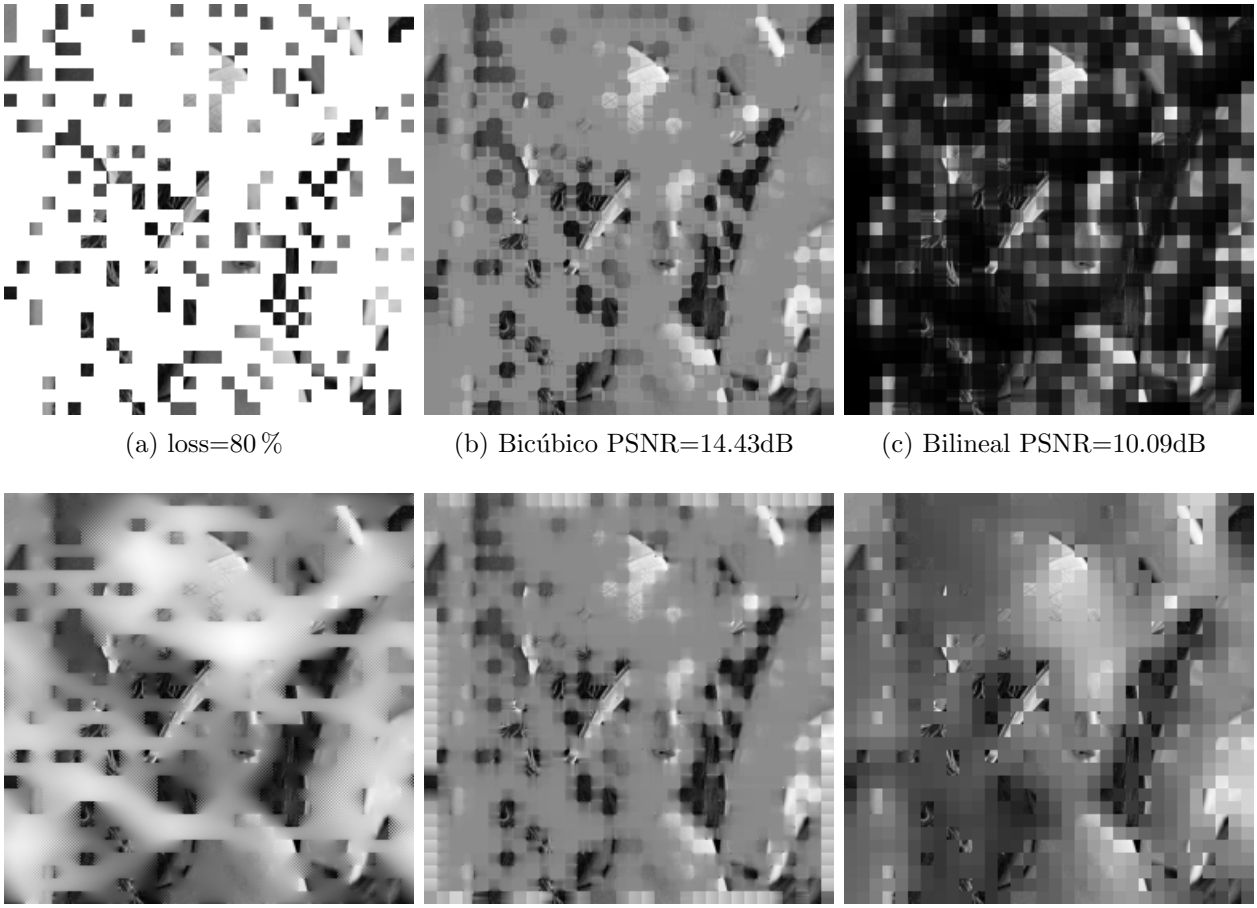


(a) loss=50 % (b) Bicúbico PSNR=18.60dB (c) Bilineal PSNR=15.33dB

(d) CDD PSNR=21.08dB (e) MSR PSNR=18.30dB (f) TV PSNR=19.37dB

Figura 6.9: Restauración con distintos algoritmos de imagen lena de 512x512 transmitida en bloques de datos de 8x8 sobre paquetes de 100 bytes con 50 % de perdida de paquetes de datos

Finalmente, para una perdida de datos del 80 % la imagen prácticamente es irreconocible (figura 6.10), la recuperación mediante TV, es la que tiene mayor PSNR y visualmente, la que mas se acerca a los tonos de la imagen, sin embargo la información restante de la imagen se pierde completamente, como rasgos faciales o textura de la imagen, los demás métodos arrojan o manchones grises como en el caso del CDD, MSR y Bicúbico u oscuros en el caso del Bilineal. Visualmente se observa el resultado de las gráficas de la sección anterior en donde el CDD entregaba los mejores resultados en la reconstrucción de las imágenes hasta que llegaba a una perdida de información del 80 %, en ese momento en la mayoría de las ocasiones bajaba el valor medio de PSNR, lo que se comprueba en la imagen 6.10d de la figura 6.10.



(a) loss=80 % (b) Bicúbico PSNR=14.43dB (c) Bilineal PSNR=10.09dB

(d) CDD PSNR=11.24dB (e) MSR PSNR=13.96dB (f) TV PSNR=15.93dB

Figura 6.10: Restauración con distintos algoritmos de imagen lena de 512x512 transmitida en bloques de datos de 8x8 sobre paquetes de 100 bytes con 80 % de perdida de paquetes de datos

Capítulo 7

Conclusiones y comentarios

En el presente trabajo de título se realizaron un total de 240.000 simulaciones de transmisiones inalámbricas de imágenes con perdida de datos, que tomó un aproximado de tres semanas, los resultados obtenidos se tabularon en el capítulo 5 y en el Anexo A, posteriormente se analizaron esos datos en el capítulo 6, a partir de esa información se puede llegar a las siguiente conclusiones:

De los cinco algoritmos estudiados, ninguno tuvo un desempeño 100 % deseable, todos tuvieron sus ventajas y desventajas sobre otros, por lo tanto se dividirán las conclusiones por parámetros de análisis.

El algoritmo que presentó los valores máximos de PSNR fue el algoritmo de *inpainting* CDD, seguido de cerca por el TV, ambos por encima de los demás algoritmos, seguidos por el Bicúbico y el MSR quedando finalmente el Bilineal como el peor evaluado por valor máximo de PSNR.

El método de restauración con menos dispersión de datos fue siempre el Bicúbico, seguido del Bilineal y el MSR. El algoritmo de reconstrucción mediante CDD al igual que el TV entregan poca robustez debido a la alta tasa de fallas y la gran dispersión de datos en los resultados finales, causadas por las mismas.

Visualmente el CDD reconstruye las imágenes de mejor manera en comparación a los demás, pero solo a porcentajes de pérdida bajo, a porcentajes de perdida altos sobre el 60 % o 70 % el MSR toma ventajas por sobre los demás, al igual que el Bicúbico.

Como conclusión final, se establece que a partir de lo anterior, el algoritmo mas robusto para reconstruir imágenes con perdida de datos en las transmisión inalámbrica es el Bicúbico, debido a la gran robustez de funcionamiento en los distintos porcentajes de pérdidas, a distintos tamaños de bloques y a paquetes de datos distintos. Por otro lado la cantidad de ecuaciones y la dificultad para desarrollar computacionalmente el Bicúbico es mucho menor comparada con el CDD y TV lo que se refleja en una restauración mucho mas rápida, esto se refleja positivamente en un consumo energético menor. Por otro lado si la transmisión se realiza en bloques de 8x8 o mayores se recomienda el uso del MSR, ya que al trabajar en cuadrantes, sus resultados son mucho mejor en bloques de datos grandes. No se recomienda el uso de algoritmos de *inpainting* debido a la poca confiabilidad en la reconstrucción de imágenes, además la gran cantidad de ecuaciones no lineales hacen que estos algoritmos sean lentos en comparación a los demás métodos, considerando además que funcionan bajo condiciones iterativas.

Tomando en cuenta que una restauración aceptable, sobre una red inalámbrica, debe tener un PSNR por sobre los 20dB [41], el Bicubico en general cumple con lo anterior hasta el 50 % de pérdida de paquetes de datos en las tres imágenes de 512x512 bajo las que se aplicaron los parámetros de simulación. En la imagen de 200x200 Bird, se garantizan resultados aceptables hasta el 60 % de pérdida de paquetes de datos. Finalmente en las dos imágenes de 128x128, este umbral de confianza en la reconstrucción disminuye en comparación al resto de las imágenes, llegando hasta el 40 % de paquetes perdidos en los que ofrece una reconstrucción aceptable.

Bibliografía

[1] Lakshmi Bhanu Bangaru y Mr. Vimal Gupta. Object removal by kriging interpolation technique. *IEEE*, 2015.

[2] Yacine Bazi, Moufida Maimour, y Bouabdellah Kechar. Evalvsn : a new tool for video quality evaluation in wireless sensor networks. *IEEE*, 2014.

[3] Tony F. Chan y Jianhong Shen. Non-texture inpainting by curvature-driven diffusions (cdd). 2000.

[4] Tony F. Chan y Jianhong Shen. Mathematical models for local nontexture inpaintings. *SIAM Journal on Applied Mathematics*, 62(3):1019–1043, 2001.

[5] Tony F. Chan, Jianhong Shen, y Hao-Min Zhou. Total variation wavelet inpainting. *Mathematical Imagin and Vision*, 25(1):107–125, 2006.

[6] Damon M. Chandler y Sheila S. Hemami. Vsnr: A wavelet-based visual signal-to-noise ratio for natural images. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 16(9), 2007.

[7] Antonio Criminisi, Patrick Perez, y Kentaro Toyama. Region filling and object removal by exemplar-based image inpainting. *TRANSACTIONS ON IMAGE PROCESSING*, 13(9), 2004.

[8] SELIM ESEDOGLU y JIANHONG SHEN. Digital inpainting based on the mumford–shah–euler image model. *European Journal of Applied Mathematics*, (4):353–370, 2002.

[9] Xuan Fei, Liang Xiao, Yubao Sun, y Zhihui Wei. Perceptual image quality assessment based on structural similarity and visual masking. *Signal Processing: Image Communication*, 27:772–783, 2012.

[10] Zhilin Feng, Shuiming Chi, Jianwei Yin, Duanyang Zhao, y Xiaoming Liu. A variational approach to medical image inpainting based on mumford-shah model. *IEEE*, 2007.

[11] Christopher Arredondo Flores. Desarrollo e implementación de un software para el estudio de pérdida de paquetes en la transmisión de imágenes mediante redes de transmisión con pérdidas. 2015.

[12] F. Guichard y J.M. Morel. Partial differential equation and image iterative filtering. *tutorial of ICIP*, 1995.

[13] Gregory Hackmann, Weijun Guo, y Guirong Yan. Cyber-physical codesign of distributed structural health monitoring with wireless sensor networks. *ICCSP*, 2010.

[14] Sheila S. Hemami y Robert M. Gray. Subband-coded image reconstruction for lossy packet networks. *IEEE Transactions on Image Processing*, 6(4):523–539, 1997.

[15] Miyoun Jung, Xavier Bresson, Tony F. Chan, y Luminita A. Vese. Nonlocal mumford-shah regularizers for color image restoration. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 20(6), 2011.

[16] Meisinger K y Kaup A. Spatial error concealment of corrupted image data using frequency selective extrapolation. En *Proceedings of the international conference on acoustics, speech and signal processing*, págs. 209–12. 2004.

[17] André Kaup, Katrin Meisinger, y Til Aach. Frequency selective signal extrapolation with applications to error concealment in image communication. *International Journal of Electronics and Communications*, págs. 147–156, 2005.

[18] Ján Koloda, Jurgen Seiler, André Kaup, Victoria Sánchez, y Antonio M. Peinado. An error-based recursive filling ordering for image error concealment. *ICIP*, págs. 2517–2521, 2014.

[19] Timilehin Labeodana, Christel De Bakkerb, y Alexander Rosemann. On the application of wireless sensors and actuators network in existing buildings for occupancy detection and occupancy-driven lighting control. *Energy and Buildings*, (127):75–83, 2016.

[20] Shengfeng Li, Rujing Wang, Jin Xie, y Yi Dong. Exemplar image inpainting by means of curvature-driven method. En IEEE, ed., *International Conference on Computer Science and Electronics Engineering*. 2012.

[21] Suli LI y Huiqin WANG. Image inpainting using curvature-driven diffusions based on p-laplace operator. En IEEE, ed., *Information and Control*. 2009.

[22] Wen-Nung Lie y Guan-Hua Lin. Error concealment for the transmission of h.264/avc-compressed 3d video in color plus depth format. *J. Vis. Commun. Image R.*, 32:237–245, 2015.

[23] Qin Lu, Wusheng Luo, Jidong Wang, y Bo Chen. Low-complexity and energy efficient image compression scheme for wireless sensor networks. *Computer Networks*, (52):2594–2603, 2008.

[24] Simon Masnou. Disocclusion : a variational approach using level lines. 2002.

[25] Simon Masnou y Jean-Michel Morel. Level lines based disocclusion. 1998.

[26] LUCA MOTTOLA y GIAN PIETRO PICCO. Not all wireless sensor networks are created equal: A comparative study on tunnels. *ACM Transactions on Sensor Networks*, 7(2), 2010.

[27] G. Nikolakopoulos y D. Tsitsipis P. Stavrou. A dual scheme for compression and restoration of sequentially transmitted images over wireless sensor networks. *Ad Hoc Networks*, (11):410–426, 2013.

[28] Jae-Young Pyun. Error concealment aware streaming video system over packet-based mobile networks. *IEEE*, 2008.

[29] Dhwani R., Bhadra Charmi A., Joshi Priya R., Soni Nikita P., y Vyas Rutvij H. Jhaveri. Packet loss probability in wireless networks: A survey. IEEE, 2015.

[30] Eric Orellana Romero, Javier SanMartin Hernandez, Cristian Duran Faundez, Vincent Lecuire, y Katherine Zapata Quiñones. Evaluation of block interleaving techniques for robust image communication in wireless camera sensor networks. *Wireless Sensors (ICWISE)*, págs. 90–95, 2014.

[31] Nikolay Samotaeva, Alexey Vasilievb, y Alexander Pisliakovb. Detection of smokeless pyrolysis of organic materials by metal oxide gas sensor. En *EUROSENSORS 2014, the XXVIII edition of the conference series*, 87, págs. 1322–1325. 2014.

[32] Guillermo Sapiro y Dario L. Ringach. Anisotropic diffusion of multivalued images with applications to color filtering. *Transactions on image processing*, 5(11), 1996.

[33] Chaitali P. Sathe y Shubhalaxmi P. Hingway. Image restoration using inpainting. *International Journal of Advance Research in Computer Science and Management Studies*, 2(1), 2014.

[34] Jurgen Seiler y André Kaup. Fast orthogonality deficiency compensation for improved frequency selective image extrapolation. *ICASSP*, 2008.

[35] Hamid Rahim Sheikh, Alan C. Bovik, y Gustavo de Veciana. An information fidelity criterion for image quality assessment using natural scene statistics. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 2004.

[36] Hamid Rahim Sheikh, Alan C., y BovikAlan C. Bovik. Image information and visual quality. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 2004.

[37] Denis Spirjakin y Alexander M. Baranov. Investigation of heating profiles and optimization of power consumption of gas sensors for wireless sensor networks. *Sensors and Actuators A*, 2016.

[38] Enyan Suna, Xuanjing Shena, y Haipeng Chen. A low energy image compression and transmission in wireless multimedia sensor networks. *procedia Engineering*, 15:3604–3610, 2011.

[39] William T., William H., Saul A., y Brian P. *Numerical Recipes in C*, tomo 2. CAMBRIDGE UNIVERSITY PRESS, 1992.

[40] Guijin Tang y Xiuchang Zhu. Error concealment for stereoscopic images using boundary smooth degree. En *2009 15th Asia-Pacific Conference on Communications*, págs. 459–461. IEEE, 2009.

[41] Nikolaos Thomos, Nikolaos V. Boulgouris, y Michael G. Strintzis. Optimized transmission of jpeg2000 streams over wireless channels. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 15(1):54–68, 2006.

[42] Feng Tian, Jia Liu, Enyan Sun, y Chuanyun Wang. An energy efficient and load balancing distributed image compression algorithm in wmsns. *Procedia Engineering*, (15):3421–3427, 2011.

[43] Andy Tsai, Anthony Yezzi, y Alan S. Willsky. Curve evolution implementation of the mumford–shah functional for image segmentation, denoising, interpolation, and magnification. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 10(8), 2001.

[44] S. Tsekeridou, F. Alaya Cheikh, M. gabbouj, y I. Pitas. Motion fields estimation by combined vector rational and bilinear interpolation for mpeg-2 error concealment. En *Signal Processing Conference, 2000 10th European*, págs. 1–4. 2000.

[45] Deepak S. Turaga, Yingwei Chen, y Jorge Caviedes. No reference psnr estimation for compressed pictures. *Signal Processing: Image Communication*, (19):173–184, 2004.

[46] Aqeel ur Rehman, Abu Zafar Abbasi, y Noman Islam. A review of wireless sensors and networksápplications in agriculture. *Computer Standards Interfaces*, 2011.

[47] Yao Wang, Qin-Fan Zhu, y Leonard Shaw. Maximally smooth image recovery in transform coding. *IEEE Transactions on Communications*, 41(10):1544–1551, 1993.

[48] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, y Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 13(4), 2004.

[49] Zhou Wang y Qiang Li. Information content weighting for perceptual image quality assessment. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 20(5), 2011.

[50] Zhou Wang, Ligang Lu, y Alan C. Bovik. Video quality assessment based on structural distortion measurement. *SIGNAL PROCESSING: IMAGE COMMUNICATION*, 19(1), 2004.

[51] Huaming Wu y Alhussein A. Abouzeid. Energy efficient distributed image compression in resource-constrained multihop wireless networks. *Computer Communications*, (28):1658–1668, 2005.

[52] Xuewen Wu, Na Liu, y Yunyun Song. Image inpainting algorithm based on adaptive template direction. *CISP*, (6th), 2013.

[53] Guobao Xu, Weiming Shen, y Xianbin Wang. Applications of wireless sensor networks in marine environment monitoring: A survey. *sensors*, 14:16932–16954, 2014.

[54] Hanaa ZainEldin, Mostafa A. Elhosseini, y Hesham A. Ali. Image compression algorithms in wireless multimedia sensor networks: A survey. *Ain Shams Engineering*, 6:481–490, 2014.

[55] Lin Zhang y Hongyu Li. Sr-sim: A fast and high performance iqa index based on spectral residual. *ICIP*, págs. 1473–1476, 2012.

[56] Lin Zhang, Lei Zhang, Xuanqin Mou, y David Zhang. Fsim: A feature similarity index for image quality assessment. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 20(8), 2011.

[57] Lin Zhang, Lei Zhang, Xuanqin Mou, y David Zhang. A comprehensive evaluation of full reference image quality assessment algorithms. *ICIP*, 2012.

[58] Celia A. Zorzo, Barcelos Marcos A, Batista Adriana M. Martins, y Antonio Carlos Nogueira. Level lines continuation based digital inpainting. *Computer Graphics and Image Processing*, págs. 1530–1834, 2004.

Apéndice A

Tablas estadísticas de resultados

A.1. Estadísticas Lena

Cuadro A.1: Estadistica bicúbica block 1x1; pkg 27 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,02187	25,13197	22,63779	20,76378	19,23767	17,95589	16,84256	15,8573
Varianza	0,03078	0,01653	0,01078	0,00853	0,00479	0,00318	0,00185	0,00105
Error Est·ndar (de la Media)	0,01241	0,00909	0,00734	0,00653	0,00489	0,00399	0,00304	0,00229
MÌnimo	28,56581	24,63048	22,36842	20,5229	19,0369	17,79962	16,73354	15,77705
M·ximo	29,56123	25,4529	23,01187	20,9604	19,41495	18,08862	16,95641	15,94064
Rango	0,99541	0,82241	0,64345	0,4375	0,37805	0,289	0,22286	0,1636
Mediana	29,0184	25,13653	22,63141	20,75885	19,24613	17,95384	16,84129	15,85645
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.2: Estadistica bicúbica block 2x2; pkg 27 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	30,72651	26,4448	23,70319	21,60564	19,88833	18,45571	17,19208	16,08216
Varianza	0,03883	0,02063	0,01221	0,00948	0,00569	0,00429	0,00278	0,00149
Error Est·ndar (de la Media)	0,01393	0,01016	0,00781	0,00689	0,00533	0,00463	0,00373	0,00273
MÌnimo	30,21679	26,04996	23,36202	21,34849	19,62375	18,28806	17,07195	15,98774
M·ximo	31,2691	26,80788	24,0259	21,84482	20,12277	18,59414	17,34988	16,18269
Rango	1,0523	0,75792	0,66388	0,49632	0,49902	0,30608	0,27793	0,19495
Mediana	30,72537	26,44636	23,70718	21,60006	19,88401	18,46222	17,19447	16,08219
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.3: Estadistica bicúbica block 1x1; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	28,92155	25,03521	22,5909	20,70008	19,20322	17,91711	16,81774	15,83483
Varianza	0,05665	0,0288	0,02098	0,01331	0,00825	0,00579	0,00347	0,00205
Error Est·ndar (de la Media)	0,01683	0,012	0,01024	0,00816	0,00642	0,00538	0,00417	0,0032
MÌnimo	28,13262	24,52075	22,17147	20,33745	18,95565	17,72399	16,65866	15,72125
M·ximo	29,49545	25,42021	23,0177	20,99351	19,52317	18,11139	16,95536	15,94389
Rango	1,36284	0,89946	0,84623	0,65606	0,56751	0,3874	0,29671	0,22264
Mediana	28,89928	25,04961	22,58955	20,69626	19,20228	17,91477	16,82043	15,83817
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.4: Estadistica bicúbica block 2x2; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	30,2011	25,97079	23,24004	21,24738	19,60244	18,20047	17,01969	15,96098
Varianza	0,14323	0,06247	0,03537	0,02679	0,01703	0,00876	0,00677	0,0037
Error Est·ndar (de la Media)	0,02676	0,01767	0,0133	0,01157	0,00923	0,00662	0,00582	0,0043
MÌnimo	29,36565	25,27605	22,75343	20,80922	19,29157	17,95371	16,82183	15,77646
M·ximo	31,30135	26,58229	23,74422	21,77785	20,04528	18,48901	17,26356	16,15481
Rango	1,93569	1,30623	0,99079	0,96863	0,7537	0,53531	0,44173	0,37835
Mediana	30,15935	25,97069	23,23779	21,24627	19,61036	18,2127	17,01701	15,96527
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.5: Estadistica bicúbica block 8x8; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	32,89029	28,22394	25,15912	22,80072	20,86914	19,21774	17,76039	16,44495
Varianza	0,17474	0,08038	0,04078	0,02845	0,02217	0,01427	0,00867	0,00593
Error Est·ndar (de la Media)	0,02956	0,02005	0,01428	0,01193	0,01053	0,00845	0,00658	0,00544
MÌnimo	31,68456	27,48676	24,71408	22,17217	20,50753	18,91044	17,53546	16,21559
M·ximo	33,86757	28,96803	25,72928	23,3679	21,3028	19,57066	18,04932	16,65173
Rango	2,18301	1,48127	1,0152	1,19573	0,79527	0,66022	0,51386	0,43614
Mediana	32,91899	28,21742	25,14549	22,80754	20,87555	19,22073	17,75236	16,44915
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.6: Estadistica bilineal block 1x1; pkg 27 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	23,13378	18,80407	15,90239	13,69434	11,85092	10,2685	8,8651	7,5763
Varianza	0,03037	0,02102	0,0106	0,00684	0,00414	0,00253	0,00159	0,00075
Error Est·ndar (de la Media)	0,01232	0,01025	0,00728	0,00585	0,00455	0,00356	0,00282	0,00194
MÌnimo	22,69392	18,3287	15,65912	13,52315	11,65382	10,1446	8,77473	7,49348
M·ximo	23,56134	19,25171	16,18694	13,93112	12,00517	10,42759	8,97616	7,65989
Rango	0,86742	0,92301	0,52782	0,40797	0,35136	0,28299	0,20143	0,16641
Mediana	23,14306	18,81209	15,89341	13,69314	11,85331	10,26615	8,86392	7,57549
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.7: Estadistica bilineal block 2x2; pkg 27 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	23,93747	19,5197	16,57063	14,26276	12,31942	10,64464	9,13656	7,76585
Varianza	0,02355	0,01621	0,00841	0,00665	0,00429	0,00333	0,00238	0,00108
Error Est·ndar (de la Media)	0,01085	0,009	0,00648	0,00576	0,00463	0,00408	0,00345	0,00233
MÌnimo	23,47051	19,19481	16,25598	14,03048	12,17906	10,49685	8,97524	7,63777
M·ximo	24,35968	19,86355	16,8134	14,47014	12,48429	10,79192	9,252	7,86663
Rango	0,88917	0,66873	0,55743	0,43967	0,30522	0,29507	0,27676	0,22886
Mediana	23,93059	19,52598	16,5714	14,26725	12,31635	10,64801	9,14106	7,76336
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.8: Estadistica bilineal block 1x1; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	22,96404	18,60518	15,81894	13,58775	11,76985	10,1975	8,80292	7,54313
Varianza	0,08447	0,0454	0,02307	0,01571	0,01055	0,00719	0,00417	0,00202
Error Est·ndar (de la Media)	0,02055	0,01507	0,01074	0,00886	0,00726	0,006	0,00457	0,00318
MÌnimo	22,25208	18,00874	15,44347	13,1808	11,43301	9,93842	8,62738	7,41239
M·ximo	23,66911	19,1234	16,28379	13,8435	12,04701	10,39696	8,98255	7,65453
Rango	1,41703	1,11466	0,84033	0,6627	0,61401	0,45854	0,35518	0,24214
Mediana	22,95414	18,60664	15,81797	13,58816	11,75891	10,20132	8,80828	7,54757
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.9: Estadistica bilineal block 2x2; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	23,05141	18,73362	15,86681	13,69272	11,85688	10,26202	8,85712	7,57883
Varianza	0,08526	0,04205	0,02168	0,01613	0,01205	0,00762	0,00464	0,00302
Error Est·ndar (de la Media)	0,02065	0,0145	0,01041	0,00898	0,00776	0,00617	0,00481	0,00389
MÌnimo	22,2534	18,13599	15,45407	13,40503	11,59826	9,96629	8,67025	7,423
M·ximo	23,72461	19,30304	16,18133	13,96112	12,12055	10,48764	9,05606	7,73433
Rango	1,4712	1,16705	0,72726	0,55609	0,52229	0,52136	0,38581	0,31133
Mediana	23,04191	18,73805	15,86809	13,69914	11,85807	10,26458	8,85585	7,57602
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.10: Estadistica bilineal block 8x8; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	30,16537	24,60591	20,78664	17,75826	15,18793	12,91838	10,8116	8,88143
Varianza	0,22955	0,12418	0,06667	0,04777	0,02988	0,01907	0,01392	0,00626
Error Est·ndar (de la Media)	0,03388	0,02492	0,01826	0,01545	0,01222	0,00977	0,00834	0,0056
MÌnimo	28,99959	23,41637	20,0017	17,09563	14,77697	12,58355	10,4336	8,66984
M·ximo	31,30679	25,64313	21,68724	18,32379	15,57977	13,27836	11,13314	9,0841
Rango	2,30721	2,22677	1,68555	1,22816	0,80281	0,6948	0,69954	0,41426
Mediana	30,17513	24,62637	20,76783	17,75081	15,18732	12,91878	10,81608	8,88171
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.11: Estadistica cdd block 1x1; pkg 27 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	38,47021	35,03131	32,80036	30,95275	29,30547	27,53247	24,57284	18,80563
Varianza	0,06195	0,03298	0,02092	0,03083	0,0249	0,04211	0,16421	0,11596
Error Est·ndar (de la Media)	0,0192	0,01624	0,01551	0,0223	0,02167	0,04708	0,09828	0,08793
MÌnimo	37,71486	34,54204	32,37918	30,49608	29,00593	27,21563	23,94446	18,24954
M·ximo	39,05026	35,48898	33,10414	31,30981	29,79297	27,92954	25,09296	19,35923
Rango	1,3354	0,94693	0,72496	0,81373	0,78703	0,71392	1,1485	1,10969
Mediana	38,49188	35,04322	32,81257	30,98341	29,28544	27,54278	24,73153	18,72022
% de Èxito	84	62,5	43,5	31	26,5	9,5	8,5	7,5

Cuadro A.12: Estadistica cdd block 2x2; pkg 27 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	36,91284	33,47202	31,21925	29,45333	27,7819	26,0077	23,06349	17,20235
Varianza	0,07337	0,03508	0,02921	0,02745	0,02348	0,02947	0,06971	0,15657
Error Est·ndar (de la Media)	0,02204	0,01656	0,01865	0,01926	0,02125	0,02944	0,06601	0,1319
MÌnimo	35,73499	33,02293	30,77651	29,12085	27,426	25,62819	22,74146	16,61278
M·ximo	37,51842	33,96144	31,65294	29,8765	28,16235	26,43471	23,48673	17,84428
Rango	1,78343	0,93851	0,87643	0,75565	0,73635	0,80652	0,74528	1,23151
Mediana	36,90717	33,47647	31,22245	29,4609	27,77881	26,02175	22,97537	17,20792
% de Èxito	75,5	64	42	37	26	17	8	4,5

Cuadro A.13: Estadistica cdd block 1x1; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	38,3548	34,92685	32,59369	30,38777	27,64631	23,76665	19,09643	14,26345
Varianza	0,10785	0,07973	0,08086	0,20014	0,46149	0,43128	0,20291	0,07644
Error Est·ndar (de la Media)	0,02475	0,02246	0,02494	0,04101	0,06933	0,07389	0,06019	0,04033
MÌnimo	37,29451	33,49749	31,67057	28,98474	25,19887	22,54764	18,20869	13,6733
M·ximo	39,04624	35,48557	33,22878	31,19992	28,77795	25,2292	20,0517	14,809
Rango	1,75174	1,98808	1,55821	2,21519	3,57909	2,68157	1,84301	1,1357
Mediana	38,40118	34,95109	32,63353	30,45829	27,74644	23,68706	19,00762	14,23769
% de Èxito	88	79	65	59,5	48	39,5	28	23,5

Cuadro A.14: Estadistica cdd block 2x2; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	36,67673	33,11364	30,43599	27,44757	23,83545	19,68079	15,36879	11,37959
Varianza	0,15138	0,11107	0,14011	0,34455	0,524	0,29954	0,12291	0,09946
Error Est·ndar (de la Media)	0,03156	0,02946	0,03743	0,07171	0,10341	0,11172	0,07012	0,12875
MÌnimo	35,72009	32,0074	29,18324	25,71726	21,68535	18,86751	14,7204	11,00601
M·ximo	37,52979	33,83242	31,19892	28,58147	25,46514	20,94809	16,29792	11,86231
Rango	1,80971	1,82502	2,01568	2,86421	3,77979	2,08058	1,57752	0,8563
Mediana	36,67977	33,15285	30,46723	27,56222	23,91306	19,58283	15,33351	11,35231
% de Èxito	76	64	50	33,5	24,5	12	12,5	3

Cuadro A.15: Estadistica cdd block 8x8; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	33,79216	30,35029	28,119	26,26775	24,21879	21,36329	17,48887	12,54571
Varianza	0,19814	0,09639	0,06704	0,07165	0,11739	0,29544	0,22903	0,08288
Error Est·ndar (de la Media)	0,03508	0,02811	0,02564	0,03133	0,04798	0,10272	0,0921	0,12875
MÌnimo	32,33888	29,63549	27,51995	25,72336	23,4815	20,2447	16,71767	12,20826
M·ximo	34,85593	31,01964	28,75671	27,12983	25,02853	22,55255	18,51051	12,99362
Rango	2,51706	1,38414	1,23675	1,40647	1,54703	2,30785	1,79284	0,78536
Mediana	33,81123	30,37556	28,11353	26,2655	24,22981	21,34686	17,44717	12,53967
% de Èxito	80,5	61	51	36,5	25,5	14	13,5	2,5

Cuadro A.16: Estadistica msr block 1x1; pkg 27 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	18,29465	13,71205	9,96673	6,87448	4,77884	3,24013	2,82082	1,43351
Varianza	13,30557	11,42703	10,80509	8,44311	5,63267	3,07074	2,54075	NaN
Error Est·ndar (de la Media)	0,25793	0,23903	0,23243	0,21136	0,18704	0,18074	0,35642	NaN
MÌnimo	7,05255	3,41671	2,09939	1,13476	1,15255	1,0187	1,03646	1,43351
M·ximo	31,47052	23,04476	17,95704	14,26847	12,0322	9,50555	7,73551	1,43351
Rango	24,41797	19,62805	15,85765	13,1337	10,87965	8,48686	6,69905	0,
Mediana	18,83482	13,81669	9,77698	6,68084	4,63338	2,91262	2,27029	1,43351
% de Èxito	100	100	100	94,5	80,5	47	10	0,5

Cuadro A.17: Estadistica msr block 2x2; pkg 27 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,43205	25,43391	22,73227	20,63835	18,87499	17,35583	15,95143	14,65559
Varianza	0,42191	0,19336	0,10295	0,08072	0,04904	0,03396	0,02245	0,01246
Error Est·ndar (de la Media)	0,04593	0,03109	0,02269	0,02009	0,01566	0,01303	0,0106	0,00789
MÌnimo	27,57717	24,34746	21,63858	19,93697	18,27128	16,89506	15,40911	14,36415
M·ximo	31,04507	26,72696	23,45745	21,42702	19,45316	17,85401	16,38999	14,88105
Rango	3,4679	2,3795	1,81886	1,49005	1,18188	0,95895	0,98088	0,51691
Mediana	29,50478	25,43236	22,74977	20,63356	18,87607	17,36552	15,9407	14,65731
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.18: Estadistica msr block 1x1; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	16,06628	9,39667	6,19924	4,63332	3,5615	2,00432	1,999	2,42146
Varianza	129,75462	51,59526	26,67216	8,9783	6,38527	2,34497	1,82309	NaN
Error Est·ndar (de la Media)	0,83748	0,57882	0,4695	0,35814	0,35384	0,46171	0,45007	NaN
MÌnimo	0,68601	0,59029	0,39359	0,01225	0,04169	0,52564	0,42877	2,42146
M·ximo	31,81779	27,3038	24,45196	21,75894	12,62294	4,62262	4,502	2,42146
Rango	31,13178	26,71351	24,05837	21,74668	12,58125	4,09698	4,07323	0,
Mediana	8,21044	7,73371	4,90556	4,48111	3,21911	1,21718	2,0528	2,42146
% de Èxito	92,5	77	60,5	35	25,5	5,5	4,5	0,5

Cuadro A.19: Estadistica msr block 2x2; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	28,02858	24,32962	21,70781	19,83584	18,28856	16,86792	15,63994	14,44728
Varianza	2,29166	0,88503	0,40329	0,21436	0,15006	0,06625	0,03779	0,02668
Error Est·ndar (de la Media)	0,10704	0,06652	0,0449	0,03274	0,02739	0,0182	0,01375	0,01155
MÌnimo	24,9515	22,30912	20,32391	18,69286	17,17827	16,05472	15,0745	14,00621
M·ximo	31,64458	26,94692	23,51266	21,30882	19,52189	17,59567	16,26204	14,89801
Rango	6,69308	4,6378	3,18876	2,61596	2,34362	1,54096	1,18754	0,8918
Mediana	27,90191	24,30088	21,73543	19,80002	18,26793	16,86173	15,62883	14,45014
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.20: Estadistica msr block 8x8; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	33,05305	28,6368	25,66786	23,26436	21,27367	19,49573	17,90471	16,44483
Varianza	0,16775	0,09959	0,06021	0,04354	0,03486	0,02271	0,01521	0,00867
Error Est·ndar (de la Media)	0,02896	0,02231	0,01735	0,01475	0,0132	0,01066	0,00872	0,00659
MÌnimo	31,9879	27,67549	24,91138	22,59828	20,59903	19,13521	17,61939	16,13247
M·ximo	34,13127	29,31275	26,38644	23,7797	21,78729	19,83857	18,24149	16,74085
Rango	2,14338	1,63726	1,47506	1,18142	1,18826	0,70336	0,6221	0,60838
Mediana	33,06663	28,65074	25,66877	23,26607	21,27656	19,49848	17,89424	16,45136
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.21: Estadistica tv block 1x1; pkg 27 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	37,51372	34,27414	31,71002	30,16455	27,89612	26,59732	21,34172	16,68605
Varianza	12,31861	8,11907	13,94397	7,64183	12,05369	1,42654	17,1571	3,04467
Error Est·ndar (de la Media)	0,27747	0,26231	0,40743	0,31503	0,48146	0,21806	0,79715	0,31857
MÌnimo	10,04631	15,49825	11,89309	10,42187	12,20599	20,62535	11,41401	11,38962
M·ximo	38,9465	35,47196	33,20289	31,22739	29,5416	27,33071	24,06739	17,84402
Rango	28,90019	19,97371	21,3098	20,80552	17,33561	6,70536	12,65338	6,4544
Mediana	38,19259	34,86343	32,64381	30,672	28,88391	26,82174	23,22821	17,17158
% de Èxito	80	59	42	38,5	26	15	13,5	15

Cuadro A.22: Estadistica tv block 2x2; pkg 27 bytes; img1

1	**10 %**	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	36,75564	31,66339	30,21734	27,81981	27,40129	24,86648	19,02306	16,55642
Varianza	6,95483	28,29794	14,6427	21,85533	0,43815	10,09517	23,62712	0,66468
Error Est·ndar (de la Media)	0,22368	0,53736	0,44483	0,64216	0,11701	0,66251	1,62026	0,3646
MÌnimo	11,77965	10,24515	10,91261	10,54108	24,78844	10,51739	11,43629	15,24974
M·ximo	37,71014	34,03467	31,68565	29,82984	27,94589	26,17378	23,08388	17,45119
Rango	25,93049	23,78952	20,77304	19,28876	3,15746	15,65639	11,64759	2,20145
Mediana	37,06381	33,5614	31,22196	29,41866	27,60316	25,4584	22,55504	16,63156
% de Èxito	69,5	49	37	26,5	16	11,5	4,5	2,5

Cuadro A.23: Estadistica tv block 1x1; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	37,71314	34,49217	32,01473	28,75398	24,90866	21,01184	16,74544	12,33962
Varianza	7,00127	2,68713	0,79269	9,31661	13,1952	5,3425	0,51301	0,30263
Error Est·ndar (de la Media)	0,20725	0,14662	0,08859	0,33107	0,48981	0,32366	0,11186	0,10215
MÌnimo	11,88579	17,32869	24,97536	11,21833	10,08973	12,56353	13,90436	10,74749
M·ximo	38,74822	35,48873	33,02609	30,98979	27,73443	23,36098	17,6895	12,98714
Rango	26,86244	18,16004	8,05073	19,77146	17,6447	10,79745	3,78515	2,23966
Mediana	38,13182	34,71493	32,1107	29,53502	25,84646	21,59422	16,92228	12,45295
% de Èxito	81,5	62,5	50,5	42,5	27,5	25,5	20,5	14,5

Cuadro A.24: Estadistica tv block 2x2; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	35,79393	30,81134	28,33271	25,29832	21,90899	17,51482	13,89402	10,5839
Varianza	12,80327	17,93146	12,23054	9,72283	5,51808	3,37858	0,76712	0,02685
Error Est·ndar (de la Media)	0,27941	0,36718	0,32471	0,30144	0,25784	0,2373	0,13515	0,03576
MÌnimo	13,15941	15,74511	12,45494	11,46335	11,0575	10,75518	11,3674	10,30491
M·ximo	37,80022	33,84422	30,62852	27,90999	24,38618	19,5843	15,26886	11,00296
Rango	24,6408	18,09911	18,17358	16,44664	13,32869	8,82912	3,90146	0,69805
Mediana	36,67862	32,59481	29,62594	26,25657	22,47562	18,15684	14,12671	10,56125
% de Èxito	82	66,5	58	53,5	41,5	30	21	10,5

Cuadro A.25: Estadistica tv block 8x8; pkg 100 bytes; img1

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	32,2285	28,8759	26,52941	24,75716	23,20536	21,37925	16,59702	13,67786
Varianza	25,01168	20,31482	17,85254	14,28655	6,94985	6,49268	12,2888	0,04911
Error Est·ndar (de la Media)	0,42268	0,46991	0,55008	0,60525	0,5497	0,55604	1,16851	0,09047
MÌnimo	10,60577	11,12894	10,62904	10,3363	14,274	11,90913	11,13654	13,30541
M·ximo	34,68803	30,96569	28,53444	26,64576	24,45825	22,74896	19,88908	13,95076
Rango	24,08226	19,83675	17,9054	16,30946	10,18425	10,83983	8,75254	0,64535
Mediana	33,58593	30,22515	27,84536	26,11163	24,21772	22,12806	18,7665	13,70153
% de Èxito	70	46	29,5	19,5	11,5	10,5	4,5	3

A.2. Estadísticas Baboon

Cuadro A.26: Estadistica bicúbica block 1x1; pkg 27 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	26,25024	22,36799	19,87973	17,99057	16,46293	15,17629	14,05761	13,07291
Varianza	0,03555	0,01632	0,00936	0,0076	0,00518	0,00323	0,00218	0,00103
Error Est·ndar (de la Media)	0,01333	0,00903	0,00684	0,00616	0,00509	0,00402	0,0033	0,00227
MÌnimo	25,72134	21,94686	19,632	17,78885	16,25426	15,04561	13,95537	12,98116
M·ximo	26,70011	22,72488	20,18483	18,21154	16,65464	15,34664	14,19654	13,15184
Rango	0,97877	0,77803	0,55283	0,4227	0,40039	0,30104	0,24116	0,17068
Mediana	26,24994	22,37247	19,88326	17,98739	16,46304	15,17526	14,05397	13,07366
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.27: Estadistica bicúbica block 2x2; pkg 27 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,98535	23,72757	20,9302	18,8293	17,12246	15,67406	14,41637	13,30174
Varianza	0,03793	0,02416	0,0144	0,0092	0,00579	0,00419	0,00272	0,00156
Error Est·ndar (de la Media)	0,01377	0,01099	0,00848	0,00678	0,00538	0,00458	0,00369	0,0028
MÌnimo	27,43244	23,30043	20,64513	18,58741	16,94295	15,4868	14,24737	13,21653
M·ximo	28,47504	24,13716	21,26275	19,08014	17,31998	15,89753	14,56192	13,40534
Rango	1,0426	0,83672	0,61762	0,49273	0,37703	0,41074	0,31456	0,18881
Mediana	27,96842	23,72885	20,92051	18,82537	17,12237	15,67447	14,41919	13,30288
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.28: Estadistica bicúbica block 1x1; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	26,15993	22,24527	19,81949	17,92413	16,42119	15,13585	14,03682	13,05394
Varianza	0,06428	0,03553	0,02554	0,01722	0,0085	0,00684	0,00371	0,00229
Error Est·ndar (de la Media)	0,01793	0,01333	0,0113	0,00928	0,00652	0,00585	0,00431	0,00338
MÌnimo	25,43236	21,7832	19,38645	17,52458	16,20478	14,92229	13,85505	12,92764
M·ximo	26,90522	22,77737	20,25914	18,25387	16,69735	15,42058	14,21008	13,17722
Rango	1,47286	0,99417	0,8727	0,72929	0,49257	0,49829	0,35503	0,24958
Mediana	26,16033	22,23845	19,82392	17,91508	16,41361	15,13142	14,03437	13,05093
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.29: Estadistica bicúbica block 2x2; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,47819	23,22667	20,4988	18,45757	16,82602	15,43329	14,22475	13,17244
Varianza	0,12117	0,06877	0,03956	0,02547	0,01731	0,01219	0,00736	0,0037
Error Est·ndar (de la Media)	0,02461	0,01854	0,01406	0,01129	0,0093	0,00781	0,00607	0,0043
MÌnimo	26,46797	22,61517	19,94288	17,99634	16,42399	15,16049	14,01536	13,02416
M·ximo	28,33064	24,00908	20,97318	18,98918	17,1976	15,73338	14,49911	13,3315
Rango	1,86267	1,39391	1,03029	0,99284	0,77361	0,57289	0,48375	0,30734
Mediana	27,49624	23,19824	20,48218	18,46002	16,82759	15,42499	14,21571	13,17098
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.30: Estadistica bicúbica block 8x8; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,16727	26,11079	22,85644	20,36338	18,29085	16,59855	15,07502	13,72348
Varianza	0,2115	0,11492	0,05977	0,04259	0,02812	0,01663	0,01128	0,00536
Error Est·ndar (de la Media)	0,03252	0,02397	0,01729	0,01459	0,01186	0,00912	0,00751	0,00518
MÌnimo	29,62974	25,28924	22,09916	19,77556	17,93686	16,27434	14,76643	13,5437
M·ximo	32,12501	27,00806	23,5599	20,86883	18,88258	17,01418	15,35953	13,95265
Rango	2,49527	1,71882	1,46074	1,09327	0,94572	0,73984	0,5931	0,40895
Mediana	31,21836	26,14769	22,84495	20,39296	18,27338	16,61603	15,0793	13,7233
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.31: Estadistica bilineal block 1x1; pkg 27 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	23,45048	19,12704	16,24169	14,05251	12,20254	10,62848	9,22251	7,95487
Varianza	0,0479	0,02415	0,01565	0,00927	0,00766	0,00503	0,00286	0,00144
Error Est·ndar (de la Media)	0,01548	0,01099	0,00885	0,00681	0,00619	0,00501	0,00378	0,00268
MÌnimo	22,95382	18,64542	15,88884	13,82824	11,9411	10,44722	9,04764	7,85096
M·ximo	24,06853	19,47576	16,58629	14,47318	12,39616	10,80902	9,36021	8,07322
Rango	1,11471	0,83035	0,69746	0,64494	0,45506	0,3618	0,31256	0,22226
Mediana	23,43391	19,12386	16,2359	14,05008	12,20659	10,63085	9,22401	7,95269
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.32: Estadistica bilineal block 2x2; pkg 27 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	24,22494	19,80496	16,90384	14,59981	12,66867	10,99939	9,49746	8,12656
Varianza	0,03445	0,02327	0,0134	0,00712	0,00651	0,0042	0,00342	0,00153
Error Est·ndar (de la Media)	0,01312	0,01079	0,00819	0,00597	0,0057	0,00458	0,00414	0,00277
MÌnimo	23,73981	19,23667	16,52589	14,36699	12,45346	10,83309	9,32489	8,02215
M·ximo	24,69528	20,13918	17,29077	14,87326	12,90476	11,15214	9,63102	8,23778
Rango	0,95547	0,90252	0,76489	0,50626	0,4513	0,31904	0,30613	0,21564
Mediana	24,22106	19,80657	16,90096	14,60565	12,6636	10,99817	9,49535	8,12833
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.33: Estadistica bilineal block 1x1; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	23,28349	18,95101	16,13637	13,95247	12,11235	10,55048	9,16572	7,90878
Varianza	0,11354	0,06125	0,03399	0,02345	0,0136	0,00814	0,00551	0,00306
Error Est·ndar (de la Media)	0,02383	0,0175	0,01304	0,01083	0,00825	0,00638	0,00525	0,00391
MÌnimo	22,52032	18,2846	15,67527	13,5632	11,81601	10,32112	8,92046	7,75231
M·ximo	24,14459	19,66209	16,64534	14,46026	12,43483	10,76529	9,39439	8,10908
Rango	1,62427	1,37749	0,97008	0,89706	0,61882	0,44417	0,47392	0,35676
Mediana	23,27836	18,94321	16,14671	13,94268	12,11336	10,54556	9,16563	7,90863
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.34: Estadistica bilineal block 2x2; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	23,33554	19,04413	16,19977	14,02424	12,19236	10,60707	9,21781	7,94968
Varianza	0,12657	0,05601	0,04101	0,02372	0,0178	0,01456	0,00871	0,00444
Error Est·ndar (de la Media)	0,02516	0,01673	0,01432	0,01089	0,00943	0,00853	0,0066	0,00471
MÌnimo	22,47506	18,37637	15,64745	13,60765	11,86354	10,31201	8,98914	7,75473
M·ximo	24,11717	19,76808	16,7431	14,48376	12,59548	10,90201	9,50413	8,09736
Rango	1,64211	1,39171	1,09565	0,87611	0,73194	0,59001	0,51499	0,34263
Mediana	23,34128	19,05728	16,19255	14,03327	12,18909	10,6039	9,21565	7,94907
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.35: Estadistica bilineal block 8x8; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	30,00384	24,6355	20,86081	17,9195	15,4121	13,14899	11,10361	9,20501
Varianza	0,28237	0,13241	0,0766	0,06159	0,04411	0,02261	0,02069	0,00997
Error Est·ndar (de la Media)	0,03757	0,02573	0,01957	0,01755	0,01485	0,01063	0,01017	0,00706
MÌnimo	28,25575	23,79124	19,90326	17,15551	14,87384	12,79096	10,77681	8,91123
M·ximo	31,34709	25,71814	21,54308	18,52396	15,88906	13,66979	11,5075	9,4701
Rango	3,09134	1,9269	1,63981	1,36845	1,01523	0,87882	0,73069	0,55887
Mediana	30,01344	24,66392	20,86259	17,92275	15,41201	13,14184	11,09684	9,20134
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.36: Estadistica cdd block 1x1; pkg 27 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	36,76664	33,42299	31,24073	29,47224	27,84209	25,97562	22,69842	16,29269
Varianza	0,14103	0,06916	0,04169	0,04013	0,04828	0,06073	0,05747	0,17113
Error Est·ndar (de la Media)	0,02897	0,02343	0,02152	0,02504	0,02963	0,05653	0,05651	0,10342
MÌnimo	35,72917	32,7806	30,68835	29,05746	27,33559	25,37538	22,28268	15,73389
M·ximo	37,58532	34,06606	31,68047	29,94106	28,39651	26,40001	23,04993	16,93783
Rango	1,85615	1,28546	0,99212	0,88361	1,06092	1,02463	0,76725	1,20394
Mediana	36,76714	33,42677	31,24436	29,48314	27,84569	26,02324	22,61404	16,29056
% de Èxito	84	63	45	32	27,5	9,5	9	8

Cuadro A.37: Estadistica cdd block 2x2; pkg 27 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	35,56237	32,22483	29,99836	28,22005	26,5478	24,67073	21,26423	15,27469
Varianza	0,12483	0,06486	0,03419	0,02781	0,03138	0,06012	0,10457	0,07458
Error Est·ndar (de la Media)	0,02875	0,0226	0,0196	0,01965	0,02457	0,04205	0,08643	0,09655
MÌnimo	34,33089	31,60763	29,58664	27,78284	26,19106	24,1853	20,72762	14,83116
M·ximo	36,3517	33,01695	30,49421	28,62928	26,93205	25,25207	21,91383	15,62789
Rango	2,02081	1,40932	0,90757	0,84643	0,74099	1,06677	1,18621	0,79673
Mediana	35,55831	32,24911	30,00653	28,22554	26,58343	24,64303	21,32397	15,33051
% de Èxito	75,5	63,5	44,5	36	26	17	7	4

Cuadro A.38: Estadistica cdd block 1x1; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	36,57621	33,2183	30,91936	28,62363	25,57159	21,24733	16,51452	11,71709
Varianza	0,34465	0,15988	0,16304	0,32335	0,55824	0,69187	0,38926	0,08624
Error Est·ndar (de la Media)	0,044	0,03222	0,03501	0,05235	0,07748	0,09418	0,08337	0,04112
MÌnimo	34,61275	32,34417	29,65577	26,25636	23,01592	19,09558	15,29717	11,06374
M·ximo	37,82937	34,11294	31,88841	29,9758	27,23428	23,03985	17,99534	12,32178
Rango	3,21662	1,76877	2,23264	3,71944	4,21837	3,94427	2,69817	1,25805
Mediana	36,68894	33,22824	30,94348	28,69131	25,58006	21,28456	16,46464	11,70796
% de Èxito	89	77	66,5	59	46,5	39	28	25,5

Cuadro A.39: Estadistica cdd block 2x2; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	35,13841	31,57505	28,79449	25,63971	21,62712	17,15756	12,9384	9,1661
Varianza	0,28002	0,17282	0,29234	0,36793	0,56749	0,21722	0,15551	0,0266
Error Est·ndar (de la Media)	0,04278	0,03689	0,0538	0,07466	0,10549	0,09514	0,07887	0,06658
MÌnimo	33,76576	30,30696	27,07103	24,14411	19,69532	15,93961	11,95676	8,9801
M·ximo	36,26374	32,61496	30,09544	26,90944	23,3954	18,14583	13,5997	9,3547
Rango	2,49798	2,308	3,02441	2,76533	3,70008	2,20622	1,64294	0,3746
Mediana	35,15009	31,6458	28,8295	25,62259	21,55797	17,08214	13,07915	9,16867
% de Èxito	76,5	63,5	50,5	33	25,5	12	12,5	3

Cuadro A.40: Estadistica cdd block 8x8; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	32,84062	29,26323	26,96789	24,97733	22,85335	20,05158	15,79763	10,4717
Varianza	0,20513	0,14034	0,15442	0,08485	0,11507	0,25668	0,13418	0,26446
Error Est·ndar (de la Media)	0,03569	0,03392	0,0393	0,03409	0,04846	0,0925	0,0705	0,22998
MÌnimo	31,45782	28,18734	25,67378	24,33873	22,15729	18,78182	15,03049	9,8479
M·ximo	33,92193	30,247	27,84886	25,66588	23,72284	21,13079	16,6794	11,04914
Rango	2,46411	2,05966	2,17508	1,32715	1,56555	2,34896	1,64891	1,20124
Mediana	32,89807	29,2501	27,03407	24,99148	22,7862	20,08023	15,734	10,67696
% de Èxito	80,5	61	50	36,5	24,5	15	13,5	2,5

Cuadro A.41: Estadistica msr block 1x1; pkg 27 bytes; img2

1	**10 %**	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	17,58062	13,00912	9,29263	6,40539	4,33617	3,17588	2,56026	0,
Varianza	12,44758	10,83061	10,31678	7,40588	4,97894	2,39639	2,1799	NaN
Error Est·ndar (de la Media)	0,24948	0,23271	0,22712	0,20117	0,18099	0,18244	0,36911	NaN
MÌnimo	6,42771	2,80017	1,50361	1,11433	1,00837	1,00774	1,05935	0,
M·ximo	28,9436	21,65157	16,82406	13,23748	11,07875	8,67525	6,8743	0,
Rango	22,51589	18,8514	15,32045	12,12315	10,07038	7,66751	5,81496	0,
Mediana	18,12154	13,13035	9,10614	6,24461	4,21556	3,10105	2,2561	0,
% de Èxito	100	100	100	91,5	76	36	8	0,5

Cuadro A.42: Estadistica msr block 2x2; pkg 27 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,66034	23,569	20,76246	18,60359	16,81069	15,24291	13,81097	12,48925
Varianza	0,29034	0,12919	0,07926	0,05387	0,03021	0,02227	0,01449	0,00796
Error Est·ndar (de la Media)	0,0381	0,02542	0,01991	0,01641	0,01229	0,01055	0,00851	0,00631
MÌnimo	25,91404	22,61402	19,80983	18,00533	16,37005	14,85048	13,34401	12,24032
M·ximo	29,10049	24,64556	21,45583	19,30001	17,34039	15,71871	14,16675	12,70331
Rango	3,18645	2,03154	1,64599	1,29468	0,97034	0,86823	0,82274	0,46299
Mediana	27,68338	23,58093	20,77167	18,59817	16,82385	15,25443	13,81924	12,48574
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.43: Estadistica msr block 1x1; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	14,82243	8,57811	5,51397	4,165	3,33575	1,6867	1,52687	1,72183
Varianza	111,61102	44,80309	22,96567	7,44809	5,35719	2,18734	1,5964	NaN
Error Est·ndar (de la Media)	0,77673	0,53938	0,43747	0,33342	0,34503	0,49299	0,44671	NaN
MÌnimo	0,14594	0,07396	0,06148	0,00676	0,00559	0,13986	0,13087	1,72183
M·ximo	29,65679	24,68969	21,73063	19,46262	11,69415	3,88859	3,77386	1,72183
Rango	29,51085	24,61573	21,66915	19,45586	11,68856	3,74873	3,64299	0,
Mediana	7,58753	7,13509	4,28006	4,13222	3,36098	1,2609	1,41953	1,72183
% de Èxito	92,5	77	60	33,5	22,5	4,5	4	0,5

Cuadro A.44: Estadistica msr block 2x2; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	26,37256	22,54997	19,83322	17,8848	16,2692	14,81852	13,51959	12,30404
Varianza	1,50785	0,58941	0,27133	0,14284	0,09207	0,04352	0,02267	0,01878
Error Est·ndar (de la Media)	0,08683	0,05429	0,03683	0,02672	0,02146	0,01475	0,01065	0,00969
MÌnimo	23,49691	20,62993	18,62037	16,8057	15,42547	14,26559	13,14207	11,93686
M·ximo	29,19579	24,35619	21,0805	18,86754	17,17776	15,38513	13,95856	12,68343
Rango	5,69888	3,72625	2,46013	2,06185	1,75229	1,11955	0,81649	0,74657
Mediana	26,24713	22,49468	19,84945	17,84977	16,26299	14,81643	13,50529	12,30131
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.45: Estadistica msr block 8x8; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,86096	27,07627	23,82022	21,23398	19,02314	17,16987	15,46407	13,92236
Varianza	0,22333	0,13738	0,08028	0,05948	0,04158	0,02538	0,01722	0,009
Error Est·ndar (de la Media)	0,03342	0,02621	0,02003	0,01725	0,01442	0,01127	0,00928	0,00671
MÌnimo	30,54671	25,96987	22,89719	20,55333	18,55754	16,7761	15,14735	13,67672
M·ximo	33,02477	28,01517	24,55744	21,78072	19,73995	17,66852	15,79014	14,25219
Rango	2,47806	2,04529	1,66025	1,22739	1,18241	0,89242	0,64279	0,57546
Mediana	31,85825	27,09978	23,81598	21,24249	19,02197	17,17918	15,46694	13,92432
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.46: Estadistica tv block 1x1; pkg 27 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	36,08328	32,54009	29,34128	27,83736	27,04722	24,70816	20,39446	16,03305
Varianza	7,21193	7,77926	22,2139	18,87923	3,80201	8,8922	14,82757	3,88055
Error Est·ndar (de la Media)	0,21165	0,25896	0,50823	0,50172	0,23646	0,49023	0,70303	0,37228
MÌnimo	10,19897	15,0999	10,20306	10,30444	12,23272	11,48533	10,85206	10,34882
M·ximo	37,36696	33,8805	31,68594	29,85033	27,99125	25,89148	22,99095	17,75855
Rango	27,16799	18,7806	21,48288	19,54588	15,75853	14,40615	12,13888	7,40973
Mediana	36,54352	33,17891	30,96815	29,20468	27,38472	25,39618	22,09306	16,79819
% de Èxito	80,5	58	43	37,5	34	18,5	15	14

Cuadro A.47: Estadistica tv block 2x2; pkg 27 bytes; img2

1	**10 %**	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	35,56391	31,95339	29,30608	27,71474	25,47936	23,54305	19,57541	14,98104
Varianza	5,10551	5,40129	8,62129	4,69755	7,69513	5,22908	15,67895	5,10847
Error Est·ndar (de la Media)	0,19234	0,23358	0,33461	0,30056	0,45604	0,57168	1,39995	1,30492
MÌnimo	10,17004	12,71516	12,23509	13,19802	13,95421	16,54749	11,86411	12,41091
M·ximo	36,56676	32,9586	30,70544	28,73538	26,92145	24,99589	22,1071	16,65887
Rango	26,39672	20,24344	18,47035	15,53735	12,96725	8,4484	10,24299	4,24797
Mediana	35,81734	32,35701	30,07465	28,15368	26,31916	24,42726	21,6362	15,87335
% de Èxito	69	49,5	38,5	26	18,5	8	4	1,5

Cuadro A.48: Estadistica tv block 1x1; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	35,76913	32,28739	29,7207	27,42893	24,65926	20,46271	15,9001	12,48936
Varianza	7,14073	8,30793	8,35345	3,70336	3,60438	6,67798	3,28422	0,56285
Error Est·ndar (de la Media)	0,21126	0,24807	0,28341	0,21651	0,23919	0,34533	0,31547	0,13475
MÌnimo	14,95205	11,34185	15,7791	16,00147	15,83143	10,03681	10,23454	10,01243
M·ximo	37,71044	33,93492	31,69701	29,10645	26,98368	22,37843	17,73469	13,3627
Rango	22,75839	22,59307	15,91791	13,10498	11,15224	12,34161	7,50015	3,35027
Mediana	36,26744	32,75268	30,41315	27,83636	25,04582	21,05354	16,60687	12,73386
% de Èxito	80	67,5	52	39,5	31,5	28	16,5	15,5

Cuadro A.49: Estadistica tv block 2x2; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	34,37325	29,82372	27,25282	23,61965	20,51932	16,80376	13,71682	10,81862
Varianza	11,72335	16,3038	12,10181	14,56827	9,26923	6,20047	1,53482	0,08832
Error Est·ndar (de la Media)	0,26338	0,34497	0,323	0,3916	0,33828	0,32696	0,19116	0,06066
MÌnimo	12,39158	14,43101	10,2064	10,67339	10,16354	10,2423	10,24499	10,15314
M·ximo	36,52644	32,45418	29,66367	26,51913	23,63577	19,07804	14,97374	11,33496
Rango	24,13486	18,02317	19,45727	15,84574	13,47224	8,83575	4,72874	1,18182
Mediana	35,08232	31,20089	28,23309	25,03016	21,62941	17,83795	14,10677	10,84716
% de Èxito	84,5	68,5	58	47,5	40,5	29	21	12

Cuadro A.50: Estadistica tv block 8x8; pkg 100 bytes; img2

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,99262	27,53663	26,23862	23,04619	22,74092	20,38969	17,26585	13,75205
Varianza	14,33255	21,43618	9,79071	19,16748	1,73334	2,1747	1,74204	0,03778
Error Est·ndar (de la Media)	0,32111	0,46769	0,40395	0,76212	0,26874	0,30749	0,49886	0,11221
MÌnimo	10,76961	11,87392	10,90108	10,06418	16,72935	15,57085	14,53999	13,53182
M·ximo	33,97216	30,18982	27,77678	25,80971	23,57744	21,44806	18,44408	13,89956
Rango	23,20255	18,3159	16,8757	15,74553	6,84809	5,87721	3,90409	0,36775
Mediana	32,83813	29,20372	26,93461	24,82653	23,0685	20,82759	17,51366	13,82477
% de Èxito	69,5	49	30	16,5	12	11,5	3,5	1,5

A.3. Estadísticas Peppers

Cuadro A.51: Estadistica bicúbica block 1x1; pkg 27 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,39771	23,99375	21,87205	20,29889	19,00865	17,9346	16,98648	16,14777
Varianza	0,0204	0,00962	0,00528	0,00371	0,00283	0,00191	0,00119	0,00069
Error Est·ndar (de la Media)	0,0101	0,00693	0,00514	0,00431	0,00376	0,00309	0,00244	0,00186
MÌnimo	27,00007	23,64083	21,61729	20,16498	18,86394	17,82105	16,89172	16,06476
M·ximo	27,76311	24,29127	22,07371	20,45195	19,16784	18,04145	17,07602	16,23023
Rango	0,76304	0,65044	0,45642	0,28697	0,30391	0,2204	0,1843	0,16547
Mediana	27,41256	24,00672	21,87135	20,2934	19,00519	17,93682	16,98627	16,14908
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.52: Estadistica bicúbica block 2x2; pkg 27 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	28,0592	24,60687	22,40952	20,75134	19,38959	18,2326	17,21278	16,29697
Varianza	0,01511	0,00813	0,00479	0,00405	0,00243	0,00224	0,00155	0,00102
Error Est·ndar (de la Media)	0,00869	0,00638	0,00489	0,0045	0,00349	0,00335	0,00279	0,00225
MÌnimo	27,74601	24,34517	22,22261	20,57048	19,26824	18,10834	17,07737	16,22696
M·ximo	28,34199	24,84099	22,60471	20,9135	19,51584	18,40213	17,32764	16,38561
Rango	0,59598	0,49582	0,3821	0,34302	0,2476	0,29379	0,25027	0,15865
Mediana	28,06375	24,60301	22,4118	20,75307	19,38784	18,2388	17,21443	16,29721
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.53: Estadistica bicúbica block 1x1; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,30117	23,92784	21,82369	20,25271	18,98215	17,90048	16,96743	16,13389
Varianza	0,06405	0,02201	0,01351	0,00971	0,0062	0,00373	0,00226	0,00171
Error Est·ndar (de la Media)	0,0179	0,01049	0,00822	0,00697	0,00557	0,00432	0,00336	0,00292
MÌnimo	26,45797	23,4705	21,51191	20,01009	18,70742	17,73651	16,82901	16,02143
M·ximo	27,845	24,3177	22,08745	20,50571	19,15518	18,03614	17,08487	16,24488
Rango	1,38703	0,8472	0,57554	0,49562	0,44776	0,29963	0,25586	0,22344
Mediana	27,31552	23,92688	21,82342	20,25095	18,98309	17,89823	16,9683	16,13461
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.54: Estadistica bicúbica block 2x2; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,76084	24,30544	22,11799	20,51066	19,18724	18,05488	17,08141	16,20462
Varianza	0,05801	0,02029	0,0149	0,01162	0,00697	0,00464	0,00396	0,00196
Error Est·ndar (de la Media)	0,01703	0,01007	0,00863	0,00762	0,0059	0,00481	0,00445	0,00313
MÌnimo	26,99085	23,98443	21,82157	20,2555	18,95635	17,8513	16,90434	16,08304
M·ximo	28,51892	24,66202	22,46674	20,7793	19,35243	18,30048	17,23904	16,32746
Rango	1,52808	0,67758	0,64517	0,5238	0,39607	0,44917	0,3347	0,24442
Mediana	27,75909	24,31571	22,11525	20,50766	19,19291	18,05999	17,08583	16,20292
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.55: Estadistica bicúbica block 8x8; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	28,50556	25,08978	22,92438	21,2395	19,84907	18,62874	17,51707	16,5175
Varianza	0,0316	0,01581	0,00821	0,00813	0,00679	0,00558	0,00428	0,00318
Error Est·ndar (de la Media)	0,01257	0,00889	0,00641	0,00637	0,00583	0,00528	0,00462	0,00399
MÌnimo	27,94382	24,76843	22,68245	20,96276	19,55513	18,43675	17,36196	16,35895
M·ximo	28,92585	25,4732	23,13619	21,44048	20,09789	18,82846	17,69115	16,66363
Rango	0,98203	0,70477	0,45374	0,47772	0,54276	0,39171	0,32918	0,30468
Mediana	28,50079	25,0977	22,92079	21,2387	19,84754	18,62461	17,51471	16,51457
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.56: Estadistica bilineal block 1x1; pkg 27 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	21,93903	17,73974	14,94267	12,78758	10,98819	9,44316	8,05976	6,80331
Varianza	0,01916	0,01194	0,00798	0,00596	0,00338	0,00217	0,00129	0,00055
Error Est·ndar (de la Media)	0,00979	0,00773	0,00632	0,00546	0,00411	0,0033	0,00254	0,00166
MÌnimo	21,47938	17,30697	14,66778	12,57715	10,82896	9,31211	7,95871	6,74379
M·ximo	22,30305	18,00295	15,1935	12,97583	11,11269	9,62782	8,15473	6,86515
Rango	0,82367	0,69598	0,52572	0,39868	0,28372	0,31571	0,19602	0,12136
Mediana	21,93346	17,73565	14,94845	12,78774	10,98546	9,44505	8,05785	6,80366
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.57: Estadistica bilineal block 2x2; pkg 27 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	22,56092	18,33173	15,51346	13,3005	11,42883	9,79311	8,32226	6,97833
Varianza	0,01586	0,00896	0,00618	0,00356	0,00207	0,00206	0,00111	0,00061
Error Est·ndar (de la Media)	0,0089	0,00669	0,00556	0,00422	0,00321	0,00321	0,00236	0,00175
MÌnimo	22,17693	18,12457	15,29963	13,12974	11,26412	9,66062	8,23277	6,91203
M·ximo	22,92187	18,60607	15,75018	13,43608	11,53484	9,9148	8,40218	7,03362
Rango	0,74493	0,48149	0,45055	0,30634	0,27072	0,25417	0,1694	0,12159
Mediana	22,55636	18,32873	15,50853	13,29951	11,42909	9,79401	8,32508	6,97818
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.58: Estadistica bilineal block 1x1; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	21,79194	17,56307	14,84568	12,69697	10,91832	9,36998	8,00725	6,76678
Varianza	0,05291	0,0295	0,01935	0,01242	0,00901	0,00533	0,00292	0,00135
Error Est·ndar (de la Media)	0,01626	0,01215	0,00983	0,00788	0,00671	0,00516	0,00382	0,0026
MÌnimo	21,04056	17,14728	14,43984	12,33666	10,63403	9,16638	7,88558	6,68484
M·ximo	22,3968	18,02812	15,27245	12,97218	11,17177	9,56784	8,15545	6,85671
Rango	1,35624	0,88084	0,83261	0,63552	0,53774	0,40145	0,26987	0,17187
Mediana	21,7939	17,55956	14,83306	12,6899	10,91406	9,37035	8,00723	6,76923
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.59: Estadistica bilineal block 2x2; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	21,79037	17,61202	14,85803	12,76383	10,97916	9,418	8,04959	6,80506
Varianza	0,05363	0,02696	0,01899	0,01176	0,00817	0,0052	0,00331	0,00168
Error Est·ndar (de la Media)	0,01638	0,01161	0,00974	0,00767	0,00639	0,0051	0,00407	0,0029
MÌnimo	21,102	17,12262	14,44204	12,43154	10,7174	9,24866	7,87987	6,69425
M·ximo	22,30668	18,11898	15,12822	13,07057	11,18569	9,60189	8,18943	6,96013
Rango	1,20468	0,99636	0,68617	0,63904	0,46829	0,35323	0,30956	0,26588
Mediana	21,81709	17,61589	14,85996	12,75842	10,97612	9,42004	8,05005	6,80032
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.60: Estadistica bilineal block 8x8; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	26,97057	22,42965	19,16878	16,44569	14,09627	11,95096	9,95826	8,08202
Varianza	0,05303	0,03871	0,03542	0,02102	0,01627	0,00956	0,00576	0,00408
Error Est·ndar (de la Media)	0,01628	0,01391	0,01331	0,01025	0,00902	0,00691	0,00537	0,00452
MÌnimo	26,34788	21,84825	18,68531	16,00805	13,65213	11,6836	9,75737	7,89895
M·ximo	27,54315	22,86968	19,63636	16,83889	14,41318	12,24147	10,19286	8,25133
Rango	1,19527	1,02143	0,95105	0,83084	0,76106	0,55787	0,43549	0,35238
Mediana	26,95715	22,42807	19,17344	16,46584	14,10455	11,95661	9,96401	8,08391
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.61: Estadistica cdd block 1x1; pkg 27 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,74182	26,58843	24,64862	23,21075	22,07239	21,04664	19,8176	17,01893
Varianza	0,05041	0,0207	0,01511	0,00683	0,00496	0,00167	0,00604	0,08561
Error Est·ndar (de la Media)	0,01722	0,01287	0,01303	0,01033	0,00941	0,00937	0,01831	0,07554
MÌnimo	29,08656	26,2369	24,3641	22,99798	21,88947	20,98181	19,67147	16,37572
M·ximo	30,21154	26,9021	24,90412	23,43781	22,22096	21,1268	19,94639	17,40029
Rango	1,12498	0,6652	0,54002	0,43983	0,33149	0,14499	0,27492	1,02457
Mediana	29,77092	26,58799	24,63418	23,21931	22,07889	21,04859	19,83236	17,06543
% de Èxito	85	62,5	44,5	32	28	9,5	9	7,5

Cuadro A.62: Estadistica cdd block 2x2; pkg 27 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,37045	26,23607	24,33029	22,96538	21,826	20,81867	19,47184	16,35516
Varianza	0,03207	0,01341	0,0083	0,00492	0,00478	0,00352	0,00952	0,05525
Error Est·ndar (de la Media)	0,01453	0,01044	0,0096	0,00816	0,00933	0,01032	0,02439	0,07433
MÌnimo	28,9191	25,96393	24,11623	22,82701	21,67233	20,65045	19,34347	15,95362
M·ximo	29,75923	26,53011	24,57707	23,1264	21,95052	20,93017	19,67611	16,77413
Rango	0,84013	0,56617	0,46084	0,2994	0,2782	0,27972	0,33264	0,82051
Mediana	29,373	26,24304	24,33448	22,96563	21,8238	20,82328	19,46423	16,38372
% de Èxito	76	61,5	45	37	27,5	16,5	8	5

Cuadro A.63: Estadistica cdd block 1x1; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,65056	26,48267	24,54858	23,08058	21,7344	20,03915	17,452	13,84905
Varianza	0,16778	0,05935	0,03105	0,02322	0,03001	0,07226	0,15611	0,0843
Error Est·ndar (de la Media)	0,0307	0,01951	0,01528	0,01397	0,01777	0,03005	0,05233	0,03988
MÌnimo	28,45127	25,89585	24,16198	22,55806	21,19573	19,34102	16,55892	13,08262
M·ximo	30,79063	27,10186	24,9176	23,42003	22,1092	20,64014	18,20695	14,42639
Rango	2,33936	1,20601	0,75562	0,86196	0,91347	1,29912	1,64803	1,34376
Mediana	29,67176	26,47954	24,54963	23,09377	21,7527	20,04252	17,50174	13,85753
% de Èxito	89	78	66,5	59,5	47,5	40	28,5	26,5

Cuadro A.64: Estadistica cdd block 2x2; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,00153	25,91353	23,95623	22,37955	20,60045	18,06737	14,75142	11,21606
Varianza	0,10319	0,05332	0,03436	0,04039	0,06848	0,13143	0,06681	0,04686
Error Est·ndar (de la Media)	0,02589	0,02025	0,01854	0,02437	0,03701	0,07251	0,05276	0,09681
MÌnimo	28,28431	25,41434	23,37028	21,94367	19,87762	17,26526	14,02755	10,90446
M·ximo	29,78147	26,46582	24,48037	22,88772	21,22479	18,65197	15,10803	11,46561
Rango	1,49715	1,05148	1,11009	0,94405	1,34717	1,38671	1,08049	0,56115
Mediana	29,03573	25,88852	23,95347	22,37942	20,63249	18,13175	14,77414	11,23216
% de Èxito	77	65	50	34	25	12,5	12	2,5

Cuadro A.65: Estadistica cdd block 8x8; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	28,99929	25,86216	23,9447	22,51135	21,20061	19,49244	16,70122	12,28087
Varianza	0,039	0,01623	0,01369	0,00991	0,00807	0,08135	0,05227	0,14385
Error Est·ndar (de la Media)	0,01556	0,01149	0,01182	0,01142	0,0127	0,05296	0,04321	0,16962
MÌnimo	28,37458	25,48403	23,65242	22,23117	20,98386	18,78374	16,30896	11,60991
M·ximo	29,51356	26,18569	24,29614	22,79338	21,42802	20,05906	17,24299	12,4868
Rango	1,13898	0,70165	0,64372	0,56221	0,44416	1,27532	0,93403	0,87689
Mediana	28,99892	25,86136	23,94331	22,51227	21,18615	19,50451	16,71015	12,47001
% de Èxito	80,5	61,5	49	38	25	14,5	14	2,5

Cuadro A.66: Estadistica msr block 1x1; pkg 27 bytes; img3

1	**10 %**	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	17,55719	13,04857	9,3672	6,48911	4,43257	3,18436	2,59408	0,
Varianza	12,18487	11,00025	10,55809	7,59911	5,16834	2,5892	2,34564	NaN
Error Est·ndar (de la Media)	0,24683	0,23452	0,22976	0,20378	0,1844	0,1858	0,37145	NaN
MÌnimo	6,45566	2,81378	1,54609	1,18287	1,11701	1,01547	1,06069	0,
M·ximo	28,72688	21,86611	17,08214	13,62549	11,41914	8,9124	7,23307	0,
Rango	22,27122	19,05233	15,53604	12,44262	10,30213	7,89693	6,17238	0,
Mediana	18,10664	13,17373	9,18572	6,3304	4,26376	3,0286	2,32246	0,
% de Èxito	100	100	100	91,5	76	37,5	8,5	0,5

Cuadro A.67: Estadistica msr block 2x2; pkg 27 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	26,95204	23,49952	21,19246	19,42155	17,95285	16,67432	15,48906	14,37137
Varianza	0,26495	0,13818	0,09805	0,08067	0,05615	0,04482	0,0289	0,01729
Error Est·ndar (de la Media)	0,0364	0,02628	0,02214	0,02008	0,01676	0,01497	0,01202	0,0093
MÌnimo	25,35257	22,64406	20,30612	18,74414	17,35736	16,05868	14,86201	14,0049
M·ximo	28,03259	24,48661	21,91456	20,15025	18,61679	17,2029	15,99959	14,68581
Rango	2,68002	1,84255	1,60844	1,40611	1,25944	1,14421	1,13758	0,68091
Mediana	27,00397	23,4852	21,22016	19,40889	17,95293	16,6788	15,49599	14,37643
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.68: Estadistica msr block 1x1; pkg 100 bytes; img3

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	14,76884	8,71639	5,66755	4,33005	3,49966	1,79854	1,67473	1,86362
Varianza	108,95938	46,42368	24,75121	8,02076	5,1707	2,22389	1,729	NaN
Error Est·ndar (de la Media)	0,76744	0,55084	0,45606	0,34861	0,34281	0,49709	0,46489	NaN
MÌnimo	0,03281	0,01874	0,01328	0,2937	0,30102	0,24568	0,22419	1,86362
M·ximo	29,04743	25,27563	22,80794	20,80606	11,7197	4,07334	4,01191	1,86362
Rango	29,01462	25,25689	22,79466	20,51236	11,41869	3,82766	3,78772	0,
Mediana	7,61843	7,21174	4,40278	4,26063	3,43583	1,41805	1,48319	1,86362
% de Èxito	92,5	76,5	59,5	33	22	4,5	4	0,5

Cuadro A.69: Estadistica msr block 2x2; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	25,84996	22,59254	20,31534	18,72662	17,43046	16,23505	15,20201	14,18969
Varianza	1,63182	0,74219	0,38934	0,23418	0,19296	0,08946	0,0491	0,0373
Error Est·ndar (de la Media)	0,09033	0,06092	0,04412	0,03422	0,03106	0,02115	0,01567	0,01366
MÌnimo	23,28017	20,66337	18,94724	17,56443	16,18161	15,35781	14,52809	13,69128
M·ximo	28,46209	24,79883	22,2027	20,36522	18,73631	17,11864	15,94032	14,73362
Rango	5,18192	4,13546	3,25547	2,80079	2,55469	1,76083	1,41223	1,04235
Mediana	25,74979	22,56604	20,33084	18,69255	17,41145	16,22884	15,17835	14,1916
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.70: Estadistica msr block 8x8; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	28,45472	25,1091	22,99077	21,32687	19,91949	18,64276	17,46256	16,37785
Varianza	0,03911	0,02352	0,01322	0,00932	0,00969	0,00826	0,00686	0,00515
Error Est·ndar (de la Media)	0,01398	0,01084	0,00813	0,00683	0,00696	0,00643	0,00586	0,00507
MÌnimo	27,80223	24,71937	22,72184	21,03376	19,6494	18,39105	17,22886	16,18492
M·ximo	28,94992	25,54057	23,3403	21,57735	20,14164	18,86131	17,68844	16,54977
Rango	1,14769	0,8212	0,61846	0,54359	0,49225	0,47027	0,45959	0,36485
Mediana	28,44832	25,11114	22,99447	21,32734	19,92067	18,64812	17,45931	16,37446
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.71: Estadistica tv block 1x1; pkg 27 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,58476	26,30344	24,48167	23,08922	21,52235	20,14056	18,13298	14,84829
Varianza	2,44006	3,9461	1,67028	0,7499	4,52086	5,57507	6,86306	3,45526
Error Est·ndar (de la Media)	0,12273	0,17697	0,14272	0,09622	0,27224	0,43109	0,42498	0,3286
MÌnimo	12,83106	10,07855	13,1186	15,7409	10,54575	11,12657	10,18852	10,25851
M·ximo	30,19864	26,91453	24,95409	23,44635	22,31236	21,15879	19,68781	16,21014
Rango	17,36758	16,83598	11,83549	7,70545	11,76661	10,03222	9,49929	5,95163
Mediana	29,77234	26,63572	24,64574	23,2015	21,98932	20,90809	19,35619	15,73675
% de Èxito	81	63	41	40,5	30,5	15	19	16

Cuadro A.72: Estadistica tv block 2x2; pkg 27 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,3219	25,94862	24,26318	22,75627	21,24393	20,77596	17,92225	14,6316
Varianza	1,9314	4,6563	1,17986	2,09916	3,69051	0,00438	8,23526	5,09342
Error Est·ndar (de la Media)	0,11746	0,22023	0,12543	0,20092	0,32018	0,0156	0,95657	1,0093
MÌnimo	13,1376	10,45198	15,01953	12,54788	11,36966	20,60905	11,5076	10,61774
M·ximo	29,84347	26,58736	24,63897	23,09275	22,00005	20,90292	19,49368	15,93599
Rango	16,70588	16,13538	9,61944	10,54487	10,6304	0,29387	7,98608	5,31825
Mediana	29,43583	26,29643	24,3938	22,97845	21,79937	20,77784	19,21018	15,65785
% de Èxito	70	48	37,5	26	18	9	4,5	2,5

Cuadro A.73: Estadistica tv block 1x1; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,36359	26,29431	24,2377	22,53816	20,85822	18,08746	14,70606	11,51592
Varianza	4,41726	1,78755	2,20342	1,94363	3,906	4,52708	2,26159	0,16524
Error Est·ndar (de la Media)	0,16827	0,11911	0,14844	0,15303	0,27147	0,30711	0,24081	0,06971
MÌnimo	11,45605	11,786	11,25841	14,31872	10,10821	11,13307	10,1366	10,16883
M·ximo	30,75827	27,06283	24,89279	23,29678	21,82467	19,86733	15,87911	12,06941
Rango	19,30222	15,27683	13,63438	8,97807	11,71646	8,73426	5,74251	1,90059
Mediana	29,64211	26,44896	24,46585	22,8477	21,2718	18,89067	15,31157	11,5661
% de Èxito	78	63	50	41,5	26,5	24	19,5	17

Cuadro A.74: Estadistica tv block 2x2; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	28,4207	25,29484	22,84868	21,09185	18,46503	16,28953	13,02099	10,05712
Varianza	9,96804	7,30499	9,14193	8,65234	9,05761	1,63818	0,83199	0,00483
Error Est·ndar (de la Media)	0,24358	0,23525	0,28698	0,29866	0,35225	0,17581	0,13597	0,04915
MÌnimo	11,34186	11,23995	11,51834	11,05399	10,86956	11,41992	10,08762	10,00798
M·ximo	29,80479	26,49042	24,29781	22,87565	20,5591	17,56711	13,96124	10,10627
Rango	18,46293	15,25047	12,77947	11,82165	9,68954	6,14719	3,87362	0,09829
Mediana	29,02626	25,85565	23,80367	22,11499	19,85084	16,77565	13,37805	10,05712
% de Èxito	84	66	55,5	48,5	36,5	26,5	22,5	1

Cuadro A.75: Estadistica tv block 8x8; pkg 100 bytes; img3

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,87923	25,22954	22,90998	21,81445	20,99864	19,53677	15,67131	13,13013
Varianza	13,77963	5,99205	8,94828	8,20953	1,09046	1,15711	8,22094	0,06192
Error Est·ndar (de la Media)	0,31715	0,26396	0,36545	0,47754	0,20097	0,23473	0,79522	0,09405
MÌnimo	10,49005	10,9781	10,2429	10,10353	15,93596	16,2331	10,49916	12,90642
M·ximo	29,46147	26,30045	24,37181	22,72547	21,40536	20,12149	18,26384	13,6156
Rango	18,97142	15,32234	14,12891	12,62194	5,4694	3,88839	7,76468	0,70918
Mediana	28,96162	25,86167	23,93107	22,48551	21,24796	19,86543	17,42485	13,09431
% de Èxito	68,5	43	33,5	18	13,5	10,5	6,5	3,5

A.4. Estadísticas Bird

Cuadro A.76: Estadistica bicúbica block 1x1; pkg 27 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,62406	25,9509	23,62936	21,9823	20,52294	19,36871	18,33692	17,42234
Varianza	1,53558	0,75118	0,44016	0,32874	0,20897	0,13883	0,09139	0,04589
Error Est·ndar (de la Media)	0,08762	0,06129	0,04691	0,04054	0,03232	0,02635	0,02138	0,01515
MÌnimo	26,69809	23,61953	21,53641	20,62136	19,30318	18,05271	17,50531	16,9252
M·ximo	32,1717	28,02741	25,05756	23,46119	22,09163	20,30134	19,21	18,05509
Rango	5,47361	4,40788	3,52115	2,83983	2,78844	2,24863	1,70468	1,1299
Mediana	29,67154	25,95399	23,60332	22,00495	20,48911	19,39721	18,34722	17,41313
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.77: Estadistica bicúbica block 2x2; pkg 27 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,35828	27,4027	24,91725	22,98842	21,38912	20,02507	18,81278	17,71594
Varianza	0,85348	0,52746	0,43998	0,36069	0,22779	0,12643	0,10458	0,06574
Error Est·ndar (de la Media)	0,06533	0,05135	0,0469	0,04247	0,03375	0,02514	0,02287	0,01813
MÌnimo	28,72162	25,45215	22,77056	21,3696	20,11496	19,08763	18,08935	17,08047
M·ximo	33,30926	29,10432	26,23982	24,46365	22,48811	21,06394	19,68214	18,58376
Rango	4,58764	3,65218	3,46926	3,09405	2,37314	1,97632	1,59279	1,50329
Mediana	31,46074	27,39339	24,96706	22,95325	21,3993	20,03386	18,77118	17,69717
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.78: Estadistica bicúbica block 1x1; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	30,03342	26,23072	23,80247	21,98549	20,7634	19,46379	18,48292	17,52395
Varianza	3,8516	2,19741	1,65102	1,08258	0,73897	0,52371	0,37972	0,21214
Error Est·ndar (de la Media)	0,13877	0,10482	0,09086	0,07357	0,06079	0,05117	0,04357	0,03257
MÌnimo	24,75383	21,62845	20,29289	19,62404	18,67944	17,77497	17,31822	16,79488
M·ximo	32,96328	28,77904	26,09167	24,32373	22,82256	21,44061	20,35033	19,05501
Rango	8,20945	7,15059	5,79878	4,69969	4,14311	3,66564	3,03211	2,26012
Mediana	30,49298	26,61156	23,84875	21,90315	20,7957	19,4218	18,41297	17,48811
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.79: Estadistica bicúbica block 2x2; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,14844	27,39128	24,73977	22,9987	21,32983	19,97902	18,84618	17,90011
Varianza	2,85275	1,63893	1,76194	1,05971	0,82119	0,5167	0,38794	0,24913
Error Est·ndar (de la Media)	0,11943	0,09052	0,09386	0,07279	0,06408	0,05083	0,04404	0,03529
MÌnimo	24,86035	23,55425	20,57725	20,27273	19,36349	18,30284	17,48924	16,93866
M·ximo	33,71065	29,79755	27,1954	24,98879	23,19038	21,81944	20,50589	19,26171
Rango	8,8503	6,24329	6,61814	4,71606	3,82689	3,51661	3,01665	2,32305
Mediana	31,38108	27,68973	25,0512	23,03193	21,32923	19,94033	18,83239	17,86765
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.80: Estadistica bicúbica block 8x8; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	32,7718	28,72128	25,9985	23,96716	22,18866	20,78848	19,40512	18,09132
Varianza	1,86296	1,37929	1,10433	0,69215	0,47942	0,37459	0,26724	0,19003
Error Est·ndar (de la Media)	0,09651	0,08304	0,07431	0,05883	0,04896	0,04328	0,03655	0,03082
MÌnimo	27,67663	24,3446	22,00407	21,46516	20,12097	19,03347	17,8816	17,07458
M·ximo	35,27365	31,34057	27,90143	26,00832	24,11477	22,55735	20,81995	19,1277
Rango	7,59702	6,99597	5,89736	4,54316	3,9938	3,52389	2,93835	2,05312
Mediana	33,03138	28,85215	26,13337	23,97743	22,29172	20,76446	19,38517	18,0384
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.81: Estadistica bilineal block 1x1; pkg 27 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	21,51097	17,17091	14,37191	12,17347	10,34123	8,75786	7,3453	6,08311
Varianza	0,13396	0,05171	0,03624	0,02138	0,01252	0,0086	0,0051	0,00312
Error Est·ndar (de la Media)	0,02588	0,01608	0,01346	0,01034	0,00791	0,00656	0,00505	0,00395
MÌnimo	20,32841	16,44759	13,842	11,73458	9,99799	8,4954	7,18675	5,92233
M·ximo	22,30629	17,99469	14,89649	12,63469	10,67654	9,0174	7,59802	6,22744
Rango	1,97788	1,54711	1,05449	0,90011	0,67855	0,522	0,41127	0,30511
Mediana	21,51232	17,16728	14,36544	12,17087	10,33125	8,75234	7,3454	6,08412
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.82: Estadistica bilineal block 2x2; pkg 27 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	22,3292	17,95427	15,02221	12,73434	10,80993	9,12709	7,63861	6,27012
Varianza	0,13255	0,07248	0,03768	0,02048	0,0163	0,00851	0,0055	0,00275
Error Est·ndar (de la Media)	0,02574	0,01904	0,01373	0,01012	0,00903	0,00652	0,00525	0,00371
MÌnimo	21,50796	17,15242	14,60728	12,31075	10,50671	8,82092	7,38319	6,12818
M·ximo	23,37717	18,64725	15,6171	13,12675	11,14074	9,36664	7,82898	6,42158
Rango	1,86921	1,49483	1,00982	0,81601	0,63403	0,54572	0,44579	0,2934
Mediana	22,34222	17,96072	15,01498	12,7265	10,81741	9,12148	7,64116	6,26928
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.83: Estadistica bilineal block 1x1; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	21,31053	17,01175	14,26009	12,05299	10,24703	8,68946	7,29927	6,04535
Varianza	0,69336	0,27771	0,17194	0,08704	0,06379	0,03679	0,01852	0,00842
Error Est·ndar (de la Media)	0,05888	0,03726	0,02932	0,02086	0,01786	0,01356	0,00962	0,00649
MÌnimo	19,0707	15,72266	13,25709	11,25377	9,20641	8,23853	7,02619	5,79567
M·ximo	23,21411	18,42934	15,34538	12,8302	10,90957	9,2406	7,67795	6,34411
Rango	4,14341	2,70668	2,08829	1,57643	1,70316	1,00207	0,65177	0,54844
Mediana	21,35622	16,96574	14,2806	12,04857	10,24313	8,68807	7,30649	6,04263
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.84: Estadistica bilineal block 2x2; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	21,45871	17,0531	14,31403	12,13889	10,29506	8,72638	7,35127	6,07182
Varianza	0,74556	0,36346	0,17552	0,10725	0,05802	0,03662	0,02325	0,01069
Error Est·ndar (de la Media)	0,06106	0,04263	0,02962	0,02316	0,01703	0,01353	0,01078	0,00731
MÌnimo	19,04179	15,71644	13,23138	11,09166	9,60823	8,19398	7,01067	5,72456
M·ximo	23,58486	18,71932	15,51012	13,02881	10,8376	9,38414	7,72147	6,41435
Rango	4,54307	3,00288	2,27874	1,93715	1,22937	1,19016	0,7108	0,68978
Mediana	21,50571	17,00049	14,32224	12,17053	10,30521	8,72797	7,3354	6,06644
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.85: Estadistica bilineal block 8x8; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	29,76015	23,61248	19,48599	16,30943	13,7349	11,47456	9,36076	7,41295
Varianza	1,88189	0,78167	0,34546	0,24652	0,14266	0,09047	0,04982	0,01936
Error Est·ndar (de la Media)	0,097	0,06252	0,04156	0,03511	0,02671	0,02127	0,01578	0,00984
MÌnimo	26,13513	20,38622	17,74549	14,57945	12,78879	10,72701	8,77197	7,04963
M·ximo	32,7841	26,05779	20,90897	17,5276	15,0343	12,24296	10,06654	7,76911
Rango	6,64896	5,67157	3,16348	2,94815	2,24551	1,51595	1,29457	0,71948
Mediana	29,65207	23,70405	19,45396	16,30366	13,7554	11,4489	9,35013	7,41929
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.86: Estadistica cdd block 1x1; pkg 27 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	35,83044	31,53162	29,04231	27,43497	25,97694	24,41367	22,52093	19,65948
Varianza	0,71555	2,49683	2,23127	1,25285	0,98347	1,35489	1,76059	0,71425
Error Est·ndar (de la Media)	0,15986	0,49968	0,41429	0,23339	0,16763	0,21252	0,28955	0,34502
MÌnimo	33,74812	29,13649	26,06142	25,21814	24,20409	22,3701	19,65541	18,56055
M·ximo	36,84104	33,53346	30,98334	28,96359	28,66418	27,43304	24,92757	20,68604
Rango	3,09292	4,39697	4,92192	3,74545	4,46009	5,06294	5,27217	2,12549
Mediana	36,03635	31,55526	29,41608	27,57359	25,91319	24,50981	22,52558	19,62931
% de Èxito	14	5	6,5	11,5	17,5	15	10,5	3

Cuadro A.87: Estadistica cdd block 2x2; pkg 27 bytes; img4

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	35,96613	32,24473	29,73394	27,42095	25,12334	23,41591	21,52275	18,30499
Varianza	0,7064	0,64209	1,38217	2,12113	0,95347	0,72362	1,47177	0,43252
Error Est·ndar (de la Media)	0,10761	0,12364	0,16969	0,20394	0,17538	0,16683	0,28595	0,20797
MÌnimo	32,92548	29,97311	26,68362	23,59271	23,37674	21,90513	20,0178	17,36276
M·ximo	37,00661	33,55805	31,49403	29,39996	27,01186	25,3324	23,93057	19,37477
Rango	4,08113	3,58494	4,81041	5,80725	3,63512	3,42727	3,91277	2,01202
Mediana	36,28734	32,41678	29,80489	27,64821	25,11491	23,27891	21,07121	18,21837
% de Èxito	30,5	21	24	25,5	15,5	13	9	5

Cuadro A.88: Estadistica cdd block 1x1; pkg 100 bytes; img4

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	34,99353	30,71471	27,58867	25,12048	24,02631	22,36129	19,69768	14,9911
Varianza	7,70061	5,01702	6,54695	3,40529	2,66678	3,41256	2,81849	1,50489
Error Est·ndar (de la Media)	0,29251	0,34158	0,47514	0,38478	0,34051	0,34304	0,27234	0,24535
MÌnimo	28,71748	26,21852	21,1881	21,43618	19,79215	18,18917	16,35035	11,85361
M·ximo	37,32101	34,06852	32,15278	28,60027	26,85003	25,733	22,70765	17,09394
Rango	8,60352	7,85	10,96468	7,16408	7,05788	7,54383	6,3573	5,24032
Mediana	36,871	30,82412	27,47549	24,81157	24,33984	22,25455	19,96136	15,18175
% de Èxito	45	21,5	14,5	11,5	11,5	14,5	19	12,5

Cuadro A.89: Estadistica cdd block 2x2; pkg 100 bytes; img4

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	33,33044	29,32974	26,73347	25,208	23,27565	20,67871	16,57195	11,37202
Varianza	6,88214	4,3338	4,66277	2,39143	2,07401	2,19885	2,00522	0,5895
Error Est·ndar (de la Media)	0,23559	0,21137	0,21381	0,16301	0,15621	0,18393	0,18594	0,1214
MÌnimo	25,93458	23,45888	20,8256	20,85209	19,83133	17,00664	13,46511	9,4851
M·ximo	37,08154	33,4729	31,16318	28,04629	26,44814	23,71056	20,61487	12,91518
Rango	11,14696	10,01401	10,33759	7,1942	6,61681	6,70392	7,14976	3,43009
Mediana	33,28045	29,27803	26,97671	25,65807	23,61627	20,72085	16,50309	11,36241
% de Èxito	62	48,5	51	45	42,5	32,5	29	20

Cuadro A.90: Estadistica cdd block 8x8; pkg 100 bytes; img4

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	31,61574	27,81009	25,47409	23,66855	22,22295	20,731	17,96699	12,50817
Varianza	3,8325	3,37284	2,19693	1,13745	0,9654	0,50954	0,51588	0,26663
Error Est·ndar (de la Media)	0,15879	0,15466	0,14822	0,12657	0,13758	0,1158	0,15313	0,15569
MÌnimo	26,69871	23,69568	21,98495	21,49058	20,49161	19,31462	16,37266	11,47151
M·ximo	35,59864	31,19505	28,50433	26,8751	25,33335	22,00263	19,39536	13,05397
Rango	8,89993	7,49937	6,51938	5,38452	4,84174	2,68801	3,0227	1,58245
Mediana	31,67074	27,47789	25,40383	23,53193	22,07652	20,69987	18,09595	12,62163
% de Èxito	76	70,5	50	35,5	25,5	19	11	5,5

Cuadro A.91: Estadistica msr block 1x1; pkg 27 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	17,46927	11,5871	8,98389	7,00102	6,52272	6,60318	3,5806	1,92297
Varianza	57,26504	16,34307	12,84007	6,71056	3,96638	0,37173	0,02653	NaN
Error Est·ndar (de la Media)	0,57204	0,33343	0,32987	0,27306	0,34155	0,23044	0,09405	NaN
MÌnimo	5,89881	5,33401	4,73309	4,30063	4,34729	5,83086	3,47699	1,92297
M·ximo	31,39035	26,92618	22,20427	14,05865	10,55573	7,38674	3,76836	1,92297
Rango	25,49154	21,59218	17,47118	9,75802	6,20845	1,55588	0,29137	0,
Mediana	14,67853	11,51212	8,64683	6,60571	5,8358	6,43908	3,49646	1,92297
% de Èxito	87,5	73,5	59	45	17	3,5	1,5	0,5

Cuadro A.92: Estadistica msr block 2x2; pkg 27 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	25,50204	21,50132	19,17997	17,26593	15,5182	14,05839	12,76871	11,46834
Varianza	4,72991	2,5386	1,63826	1,65076	0,83653	0,62882	0,46636	0,27613
Error Est·ndar (de la Media)	0,15378	0,11266	0,09051	0,09085	0,06467	0,05607	0,04829	0,03716
MÌnimo	21,23791	16,50002	16,02748	14,37017	13,16004	12,40041	11,0694	10,28402
M·ximo	30,80158	25,40195	22,24167	20,54369	18,06214	16,01195	14,75145	13,13764
Rango	9,56367	8,90193	6,21419	6,17352	4,9021	3,61154	3,68205	2,85362
Mediana	25,36468	21,55142	19,16691	17,27022	15,47357	14,06006	12,66998	11,44845
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.93: Estadistica msr block 1x1; pkg 100 bytes; img4

1	**10 %**	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	30,2971	26,14885	23,67783	21,39204	19,29119	17,07993	14,90629	10,85536
Varianza	1,52573	0,9769	0,74615	0,48385	0,44281	0,58235	0,38807	1,01487
Error Est·ndar (de la Media)	0,10085	0,09646	0,10324	0,09937	0,13309	0,21165	0,27859	0,41127
MÌnimo	26,17493	22,47267	21,02677	19,72952	18,36515	15,29926	14,12945	8,92462
M·ximo	32,08245	27,89711	25,08771	22,89859	20,35291	17,92023	15,79751	11,70413
Rango	5,90752	5,42444	4,06095	3,16908	1,98776	2,62098	1,66805	2,77951
Mediana	30,5683	26,35256	23,90063	21,51339	19,0659	17,14884	14,78033	10,98144
% de Èxito	75	52,5	35	24,5	12,5	6,5	2,5	3

Cuadro A.94: Estadistica msr block 2x2; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	25,01805	20,63438	18,4707	16,43577	14,86762	13,87287	12,7745	11,6284
Varianza	27,06086	11,51767	7,32988	3,67144	2,44104	1,619	1,14741	0,5234
Error Est·ndar (de la Media)	0,36784	0,23998	0,19144	0,13549	0,11048	0,08997	0,07574	0,05116
MÌnimo	15,73137	14,63318	13,01873	12,25209	12,10765	11,37023	11,27845	10,58215
M·ximo	32,20465	27,86018	25,01136	23,16572	21,20623	18,05757	16,94744	14,11461
Rango	16,47328	13,227	11,99263	10,91363	9,09857	6,68734	5,669	3,53245
Mediana	23,60277	19,72006	17,92175	16,27673	14,74246	13,83922	12,67759	11,44589
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.95: Estadistica msr block 8x8; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,91171	27,95433	25,22001	23,21668	21,38117	19,88631	18,3568	16,89141
Varianza	1,69684	1,57168	1,59331	1,1505	0,9275	0,78316	0,62263	0,45334
Error Est·ndar (de la Media)	0,09211	0,08865	0,08926	0,07585	0,0681	0,06258	0,0558	0,04761
MÌnimo	26,52658	22,93288	19,86013	19,73544	18,46289	17,22205	16,14864	15,4155
M·ximo	34,3369	30,31161	27,49497	25,41712	23,81509	21,84579	20,42102	18,85924
Rango	7,81032	7,37873	7,63484	5,68168	5,3522	4,62373	4,27239	3,44374
Mediana	32,15264	28,1181	25,59923	23,3006	21,43484	19,82197	18,33759	16,77265
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.96: Estadistica tv block 1x1; pkg 27 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	35,83768	32,05062	30,06157	28,09545	25,98495	24,40773	21,4967	17,05769
Varianza	1,97085	3,85193	4,00924	1,7552	4,93529	1,7734	3,57909	2,6537
Error Est·ndar (de la Media)	0,11099	0,1728	0,19541	0,14455	0,26365	0,21893	0,36409	0,34731
MÌnimo	31,30772	21,92497	13,50492	24,59421	10,72368	21,77258	14,44165	11,58549
M·ximo	37,51531	34,02861	32,14344	30,58504	28,83424	27,38287	23,87708	19,15103
Rango	6,2076	12,10364	18,63852	5,99083	18,11056	5,61029	9,43543	7,56554
Mediana	36,28149	32,66261	30,38799	28,25909	26,27582	24,515	21,907	17,63552
% de Èxito	80	64,5	52,5	42	35,5	18,5	13,5	11

Cuadro A.97: Estadistica tv block 2x2; pkg 27 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	35,65202	32,18413	29,58377	27,9543	25,44967	23,75063	21,87712	16,19983
Varianza	7,94324	8,4202	8,61097	5,46934	6,33246	5,33843	0,49074	1,09509
Error Est·ndar (de la Media)	0,23325	0,25851	0,33884	0,32748	0,43806	0,56038	0,28599	0,60418
MÌnimo	11,92238	14,0893	14,01934	13,18912	14,08833	15,35092	20,89763	15,1003
M·ximo	37,30039	33,9163	31,82517	30,16269	27,77963	25,51505	22,9812	17,1836
Rango	25,37802	19,827	17,80583	16,97357	13,6913	10,16414	2,08356	2,0833
Mediana	36,45605	32,70471	30,34556	28,45906	26,13246	24,28897	21,72839	16,31558
% de Èxito	73	63	37,5	25,5	16,5	8,5	3	1,5

Cuadro A.98: Estadistica tv block 1x1; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	35,15495	30,81186	29,19332	26,99469	24,39397	21,08204	17,51793	12,39414
Varianza	12,73668	11,71278	6,52219	6,95518	7,46481	4,39224	3,11709	0,9006
Error Est·ndar (de la Media)	0,27292	0,28324	0,23712	0,26777	0,31548	0,28006	0,30279	0,1898
MÌnimo	16,49296	12,5211	22,87515	20,42458	14,52133	15,89665	14,16863	10,80074
M·ximo	37,63948	34,21544	32,32998	30,71434	29,37348	26,6108	20,35747	14,2988
Rango	21,14652	21,69435	9,45483	10,28976	14,85215	10,71416	6,18884	3,49806
Mediana	36,91181	31,68654	28,48245	27,31083	24,54809	21,10302	17,44528	12,26911
% de Èxito	85,5	73	58	48,5	37,5	28	17	12,5

Cuadro A.99: Estadistica tv block 2x2; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	33,85774	30,07066	27,48067	24,922	22,14384	19,08551	14,13611	10,08481
Varianza	7,78154	8,93226	6,66764	5,61325	4,43657	2,34949	0,71179	0,17676
Error Est·ndar (de la Media)	0,21208	0,2652	0,23572	0,25114	0,25175	0,226	0,14061	0,10856
MÌnimo	26,57212	17,22327	11,29056	12,02321	12,10246	16,47449	12,44528	9,56165
M·ximo	37,23195	33,83477	31,37213	29,25112	27,62886	22,52444	16,38725	10,92037
Rango	10,65984	16,6115	20,08157	17,22791	15,52641	6,04996	3,94197	1,35872
Mediana	33,5121	30,60454	27,6933	25,1768	22,30496	18,92444	14,12497	9,98669
% de Èxito	86,5	63,5	60	44,5	35	23	18	7,5

Cuadro A.100: Estadistica tv block 8x8; pkg 100 bytes; img4

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	32,89717	28,5585	26,2962	24,75775	23,10435	21,47229	17,72075	13,16128
Varianza	7,65516	13,95562	4,68833	3,14167	1,33303	1,65812	5,95901	0,73956
Error Est·ndar (de la Media)	0,2282	0,36632	0,24361	0,22883	0,20098	0,27453	0,63029	0,42999
MÌnimo	10,86564	10,15745	11,53008	15,55909	17,79145	16,72239	10,2094	12,46136
M·ximo	35,34847	31,70573	28,841	27,57406	24,73393	23,20791	19,82583	14,23053
Rango	24,48283	21,54828	17,31092	12,01497	6,94248	6,48553	9,61644	1,76917
Mediana	33,46674	29,20944	26,60667	24,86623	23,25913	21,74494	17,94259	12,97662
% de Èxito	73,5	52	39,5	30	16,5	11	7,5	2

A.5. Estadísticas Corridor

Cuadro A.101: Estadistica bicúbica block 1x1; pkg 27 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,92621	24,05992	21,54359	19,76527	18,21808	16,97738	15,883	14,90205
Varianza	0,28532	0,13874	0,11327	0,06715	0,04143	0,03349	0,02005	0,00902
Error Est·ndar (de la Media)	0,03777	0,02634	0,0238	0,01832	0,01439	0,01294	0,01001	0,00672
MÌnimo	26,19056	22,82208	20,61939	18,84781	17,68614	16,3933	15,47078	14,61153
M·ximo	29,57444	25,04455	22,29716	20,40863	18,77009	17,36851	16,31812	15,10189
Rango	3,38388	2,22247	1,67777	1,56082	1,08396	0,9752	0,84734	0,49036
Mediana	27,90868	24,06505	21,53511	19,77711	18,23203	16,98453	15,89184	14,90756
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.102: Estadistica bicúbica block 2x2; pkg 27 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,92621	24,05992	21,54359	19,76527	18,21808	16,97738	15,883	14,90205
Varianza	0,28532	0,13874	0,11327	0,06715	0,04143	0,03349	0,02005	0,00902
Error Est·ndar (de la Media)	0,03777	0,02634	0,0238	0,01832	0,01439	0,01294	0,01001	0,00672
MÌnimo	26,19056	22,82208	20,61939	18,84781	17,68614	16,3933	15,47078	14,61153
M·ximo	29,57444	25,04455	22,29716	20,40863	18,77009	17,36851	16,31812	15,10189
Rango	3,38388	2,22247	1,67777	1,56082	1,08396	0,9752	0,84734	0,49036
Mediana	27,90868	24,06505	21,53511	19,77711	18,23203	16,98453	15,89184	14,90756
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.103: Estadistica bicúbica block 1x1; pkg 100 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,71935	23,97644	21,49898	19,65364	18,2221	16,89568	15,86399	14,86854
Varianza	0,6329	0,34504	0,15486	0,11484	0,088	0,03957	0,02842	0,01321
Error Est·ndar (de la Media)	0,05625	0,04154	0,02783	0,02396	0,02098	0,01407	0,01192	0,00813
MÌnimo	25,07696	22,40389	20,0444	18,87668	17,16686	16,36509	15,37074	14,57861
M·ximo	29,92962	25,51844	22,51256	20,62383	18,93061	17,55507	16,3418	15,14808
Rango	4,85266	3,11455	2,46816	1,74716	1,76375	1,18998	0,97106	0,56947
Mediana	27,81238	23,96143	21,50632	19,64026	18,25297	16,90027	15,87215	14,86698
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.104: Estadistica bicúbica block 2x2; pkg 100 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	28,89993	24,97119	22,20635	20,21175	18,66499	17,22971	16,07176	14,91522
Varianza	0,88566	0,62818	0,29993	0,18188	0,10977	0,06933	0,03809	1,14518
Error Est·ndar (de la Media)	0,06655	0,05604	0,03873	0,03016	0,02343	0,01862	0,0138	0,07567
MÌnimo	26,14187	22,33951	20,33942	19,08719	17,97486	16,47084	15,59458	0,
M·ximo	31,50352	26,92154	23,39741	21,16916	19,68194	18,05603	16,63941	15,40223
Rango	5,36165	4,58202	3,05799	2,08197	1,70709	1,5852	1,04483	15,40223
Mediana	29,05446	24,95083	22,21108	20,22936	18,66447	17,23613	16,08845	14,98632
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.105: Estadistica bicúbica block 8x8; pkg 100 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,8371	27,14899	24,37798	21,88437	19,97854	18,31002	16,83181	15,5761
Varianza	1,59541	0,71583	0,55266	0,36138	0,24727	0,19442	0,12673	0,06784
Error Est·ndar (de la Media)	0,08931	0,05983	0,05257	0,04251	0,03516	0,03118	0,02517	0,01842
MÌnimo	28,66364	25,09902	22,65869	20,31757	18,69001	16,65149	15,89599	14,86039
M·ximo	35,24341	29,49182	26,23652	23,63658	21,16092	19,63985	17,70291	16,24606
Rango	6,57977	4,3928	3,57784	3,31902	2,47092	2,98836	1,80692	1,38567
Mediana	31,9265	27,07108	24,4223	21,80454	19,97292	18,33313	16,81808	15,55891
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.106: Estadistica bilineal block 1x1; pkg 27 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	20,89783	16,56738	13,65998	11,51389	9,65997	8,0809	6,68173	5,40076
Varianza	0,33482	0,12844	0,06593	0,04664	0,02642	0,01775	0,00973	0,00477
Error Est·ndar (de la Media)	0,04092	0,02534	0,01816	0,01527	0,01149	0,00942	0,00697	0,00489
MÌnimo	19,27492	15,5898	12,83489	10,95714	9,26515	7,64912	6,3914	5,189
M·ximo	22,06462	17,72758	14,28596	12,05751	10,02677	8,36071	6,97646	5,58069
Rango	2,7897	2,13778	1,45107	1,10038	0,76162	0,71159	0,58506	0,39169
Mediana	20,90317	16,54739	13,67983	11,49681	9,64989	8,07822	6,68591	5,39643
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.107: Estadistica bilineal block 2x2; pkg 27 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	21,62915	17,24003	14,33193	12,01388	10,09956	8,43555	6,94484	5,57371
Varianza	0,29485	0,12286	0,0663	0,05829	0,03907	0,02054	0,01345	0,00747
Error Est·ndar (de la Media)	0,0384	0,02479	0,01821	0,01707	0,01398	0,01013	0,0082	0,00611
MÌnimo	20,18987	16,33837	13,6506	11,4104	9,70062	7,98927	6,70715	5,334
M·ximo	23,12575	18,351	14,94402	12,78093	10,66435	8,8559	7,26058	5,8094
Rango	2,93588	2,01263	1,29343	1,37053	0,96373	0,86662	0,55343	0,4754
Mediana	21,65385	17,25139	14,3224	12,00739	10,10164	8,43817	6,94206	5,57467
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.108: Estadistica bilineal block 1x1; pkg 100 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	20,67977	16,49449	13,55453	11,40665	9,66173	8,01055	6,65317	5,36141
Varianza	0,81006	0,41568	0,23294	0,14257	0,09288	0,04801	0,02467	0,01102
Error Est·ndar (de la Media)	0,06364	0,04559	0,03413	0,0267	0,02155	0,01549	0,01111	0,00742
MÌnimo	18,19847	14,51012	12,20059	10,45177	8,47725	7,36454	6,20718	5,0153
M·ximo	22,46936	18,50868	14,98172	12,24279	10,44632	8,66181	7,04088	5,63607
Rango	4,27088	3,99856	2,78113	1,79102	1,96907	1,29727	0,8337	0,62076
Mediana	20,71981	16,46823	13,57664	11,45043	9,67292	7,99776	6,6597	5,3642
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.109: Estadistica bilineal block 2x2; pkg 100 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	20,59893	16,58045	13,64752	11,4976	9,74837	8,09098	6,71752	5,35623
Varianza	0,83551	0,39748	0,19054	0,12224	0,06459	0,04523	0,02413	0,15751
Error Est·ndar (de la Media)	0,06463	0,04458	0,03087	0,02472	0,01797	0,01504	0,01098	0,02806
MÌnimo	18,24648	14,96551	12,52172	10,51001	9,16804	7,50562	6,22338	0,
M·ximo	22,64611	18,51056	15,00889	12,29184	10,43482	8,61752	7,09284	5,70477
Rango	4,39963	3,54505	2,48716	1,78182	1,26678	1,1119	0,86946	5,70477
Mediana	20,57664	16,62301	13,65751	11,51036	9,74112	8,09328	6,721	5,38902
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.110: Estadistica bilineal block 8x8; pkg 100 bytes; img5

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	28,76044	22,48418	18,6261	15,44837	12,87498	10,61158	8,54028	6,7073
Varianza	3,21459	1,53995	0,74649	0,42383	0,22268	0,16839	0,08448	0,03869
Error Est·ndar (de la Media)	0,12678	0,08775	0,06109	0,04603	0,03337	0,02902	0,02055	0,01391
MÌnimo	24,7289	18,54041	16,54904	13,7287	11,59009	8,96376	7,82044	6,25982
M·ximo	34,7905	25,88494	20,80523	17,54269	13,90113	11,99416	9,47266	7,31555
Rango	10,0616	7,34453	4,25619	3,81399	2,31105	3,0304	1,65221	1,05573
Mediana	28,53447	22,32668	18,59334	15,4241	12,82536	10,61924	8,54021	6,71311
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.111: Estadistica cdd block 1x1; pkg 27 bytes; img5

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	35,85934	32,57469	30,45573	28,83391	27,39911	25,85997	23,88632	20,65301
Varianza	0,4128	0,18768	0,14463	0,17925	0,13675	0,22589	0,18243	0,40554
Error Est·ndar (de la Media)	0,04987	0,03922	0,03902	0,05097	0,05513	0,08677	0,09106	0,22515
MÌnimo	33,89605	31,43622	29,34313	27,99762	26,42502	24,93446	23,21526	19,72664
M·ximo	37,36995	33,54399	31,39508	29,6693	28,29652	26,68718	24,6199	21,63729
Rango	3,4739	2,10778	2,05195	1,67168	1,8715	1,75272	1,40464	1,91066
Mediana	35,84473	32,62376	30,41897	28,89429	27,38947	25,90971	23,89788	20,64772
% de Èxito	83	61	47,5	34,5	22,5	15	11	4

Cuadro A.112: Estadistica cdd block 2x2; pkg 27 bytes; img5

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	34,91281	31,51277	29,44008	27,59889	26,03893	24,35753	22,60405	19,7082
Varianza	0,44643	0,30311	0,21402	0,14097	0,23924	0,16116	0,22538	0,5279
Error Est·ndar (de la Media)	0,0525	0,05026	0,05078	0,04488	0,06988	0,0721	0,11514	0,20974
MÌnimo	33,26875	30,28091	28,04185	26,72142	24,57846	23,61279	21,56735	18,59029
M·ximo	36,57684	32,95428	30,33724	28,49181	27,25874	25,15466	23,33762	20,98762
Rango	3,30809	2,67337	2,29539	1,77039	2,68028	1,54187	1,77028	2,39734
Mediana	34,88563	31,49138	29,49999	27,62712	26,07764	24,29359	22,66092	19,63816
% de Èxito	81	60	41,5	35	24,5	15,5	8,5	6

Cuadro A.113: Estadistica cdd block 1x1; pkg 100 bytes; img5

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	35,65273	32,42908	30,24056	28,65543	27,0244	25,16631	22,6806	18,79132
Varianza	0,71791	0,45521	0,29768	0,43715	0,46646	1,44602	1,75471	2,24211
Error Est·ndar (de la Media)	0,06298	0,05385	0,04455	0,06011	0,06697	0,13043	0,1696	0,23977
MÌnimo	33,30952	30,5713	28,62012	25,50739	22,94247	21,37867	18,69417	15,20099
M·ximo	37,74641	34,09001	31,44437	29,70979	28,239	26,74851	25,08313	21,26986
Rango	4,43689	3,51871	2,82425	4,2024	5,29654	5,36985	6,38896	6,06887
Mediana	35,72536	32,4848	30,22661	28,77946	27,10347	25,59752	22,97567	18,67235
% de Èxito	90,5	78,5	75	60,5	52	42,5	30,5	19,5

Cuadro A.114: Estadistica cdd block 2x2; pkg 100 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	34,63673	31,22316	28,86125	26,94057	24,90077	22,67927	19,94001	16,15939
Varianza	0,99719	0,66726	0,37826	0,85575	1,30351	0,62724	1,22519	0,88304
Error Est·ndar (de la Media)	0,07798	0,07137	0,06002	0,12474	0,16146	0,15242	0,29583	0,38363
MÌnimo	32,35521	28,18465	27,40165	24,72524	21,51366	21,40959	17,74301	14,55908
M·ximo	36,7072	33,30814	30,60875	28,28651	26,58709	24,30323	21,86101	17,18297
Rango	4,352	5,12348	3,2071	3,56128	5,07343	2,89364	4,118	2,62388
Mediana	34,65581	31,15014	28,92371	27,13329	25,11588	22,75771	19,95284	16,19374
% de Èxito	82	65,5	52,5	27,5	25	13,5	7	3

Cuadro A.115: Estadistica cdd block 8x8; pkg 100 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,76392	28,30307	26,16447	24,42891	22,77487	21,29476	18,68756	16,09159
Varianza	1,17592	0,70184	0,45133	0,4175	0,27074	0,19002	0,89165	0,77572
Error Est·ndar (de la Media)	0,08682	0,07493	0,07203	0,07615	0,07844	0,07829	0,23607	0,35956
MÌnimo	28,78391	26,34925	24,04463	22,36381	21,67734	20,30491	16,21733	14,97773
M·ximo	34,58912	30,2785	28,23213	25,66266	24,27245	22,1337	20,01549	17,04374
Rango	5,8052	3,92925	4,1875	3,29885	2,59511	1,82879	3,79816	2,06601
Mediana	31,70993	28,29706	26,29548	24,4859	22,76589	21,34384	18,4496	16,34391
% de Èxito	78	62,5	43,5	36	22	15,5	8	3

Cuadro A.116: Estadistica msr block 1x1; pkg 27 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	20,72709	13,98543	11,39526	10,07898	8,85642	8,44093	5,8181	3,90614
Varianza	65,7701	29,87526	11,56293	4,64349	1,46486	1,42487	0,00011	1,04089
Error Est·ndar (de la Media)	0,61658	0,46031	0,32721	0,27149	0,21738	0,29842	0,00733	0,72142
MÌnimo	10,09672	8,08027	7,48635	7,4218	7,09848	7,13954	5,81077	3,18472
M·ximo	30,37329	25,4858	22,23446	19,30014	11,52996	10,98261	5,82543	4,62756
Rango	20,27656	17,40554	14,7481	11,87835	4,43148	3,84307	0,01466	1,44284
Mediana	17,09881	12,43889	10,75158	9,61648	8,3257	8,07647	5,8181	3,90614
% de Èxito	86,5	70,5	54	31,5	15,5	8	1	1

Cuadro A.117: Estadistica msr block 2x2; pkg 27 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	26,19873	22,03159	19,5323	17,546	15,92179	14,49542	13,15484	11,81773
Varianza	3,29251	1,36964	0,95004	0,52544	0,40039	0,26959	0,21775	0,11003
Error Est·ndar (de la Media)	0,12831	0,08275	0,06892	0,05126	0,04474	0,03671	0,033	0,02346
MÌnimo	20,35072	19,20812	16,59295	15,88241	14,40212	12,91909	12,12414	10,94333
M·ximo	30,33536	24,78434	23,00437	19,39668	17,59882	15,84657	14,70643	12,63544
Rango	9,98463	5,57623	6,41142	3,51427	3,19671	2,92748	2,5823	1,69211
Mediana	26,2802	22,0514	19,54666	17,52253	15,90568	14,49601	13,12486	11,8
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.118: Estadistica msr block 1x1; pkg 100 bytes; img5

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	25,8807	18,62117	15,30323	12,50477	10,34886	8,33899	8,15669	7,66656
Varianza	50,54632	61,89424	45,77217	33,48514	22,18843	13,24233	9,81037	2,10646
Error Est·ndar (de la Media)	0,54528	0,64023	0,58228	0,50948	0,50214	0,40685	0,4833	0,4376
MÌnimo	5,87353	5,61158	5,42021	5,40444	5,00246	4,60486	5,02772	5,12646
M·ximo	30,33037	26,03192	23,00025	20,38994	18,32337	16,43452	14,3833	10,31512
Rango	24,45684	20,42034	17,58004	14,9855	13,32091	11,82966	9,35558	5,18866
Mediana	28,68208	24,10752	12,21198	11,75648	10,86026	6,30539	6,24148	7,72551
% de Èxito	85	75,5	67,5	64,5	44	40	21	5,5

Cuadro A.119: Estadistica msr block 2x2; pkg 100 bytes; img5

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	25,62532	21,79427	19,002	17,03044	15,7364	14,25523	13,11228	11,89705
Varianza	12,06515	5,75088	2,85447	1,68699	1,16573	0,50845	0,33302	0,19727
Error Est·ndar (de la Media)	0,24561	0,16957	0,11947	0,09184	0,07635	0,05042	0,04081	0,03141
MÌnimo	17,51414	16,15144	14,84181	14,22179	13,73515	12,84988	12,003	11,09183
M·ximo	30,22861	26,15102	22,67642	20,36547	18,58763	15,96917	14,78151	13,11827
Rango	12,71447	9,99959	7,83461	6,14367	4,85249	3,11929	2,77851	2,02644
Mediana	27,51963	21,47214	18,90555	16,93813	15,78414	14,19826	13,07415	11,79931
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.120: Estadistica msr block 8x8; pkg 100 bytes; img5

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	30,64084	26,68005	24,23823	22,06094	20,2302	18,62823	17,11956	15,8186
Varianza	0,89049	0,50021	0,4597	0,28514	0,20599	0,19042	0,10899	0,07069
Error Est·ndar (de la Media)	0,06673	0,05001	0,04794	0,03776	0,03209	0,03086	0,02334	0,0188
MÌnimo	28,00345	25,0856	22,33566	20,61753	18,82462	17,30736	16,31822	15,01736
M·ximo	32,76684	28,49246	26,73189	23,57163	21,28581	19,7835	18,26249	16,50066
Rango	4,76339	3,40686	4,39623	2,9541	2,46119	2,47614	1,94427	1,4833
Mediana	30,6281	26,65483	24,23986	22,09126	20,20969	18,6265	17,1071	15,81674
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.121: Estadistica tv block 1x1; pkg 27 bytes; img5

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %
Media	35,35688	31,93878	29,45208	27,73165	26,13006	24,34068	20,28341	14,63331
Varianza	1,26831	1,11755	7,10342	5,52106	4,48811	1,6385	2,58615	0,97525
Error Est·ndar (de la Media)	0,08513	0,08575	0,2433	0,22203	0,22456	0,14982	0,20761	0,16936
MÌnimo	28,45702	27,65351	11,73089	12,65425	11,06276	20,58828	16,4636	12,61901
M·ximo	36,949	33,43141	31,27024	29,60853	28,07408	26,22091	23,60106	16,91774
Rango	8,49197	5,7779	19,53935	16,95428	17,01132	5,63263	7,13747	4,29872
Mediana	35,55336	32,26227	30,06258	28,31454	26,65835	24,64017	20,30528	14,52679
% de Èxito	87,5	76	60	56	44,5	36,5	30	17

Cuadro A.122: Estadistica tv block 2x2; pkg 27 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	34,79763	31,05695	28,88969	27,03796	25,31831	23,02807	19,91494	14,53174
Varianza	1,72115	2,11668	2,21182	0,61379	0,48591	6,34177	1,11484	1,12242
Error Est·ndar (de la Media)	0,10538	0,13172	0,16037	0,09571	0,11022	0,49388	0,3048	0,74914
MÌnimo	25,58447	24,66277	17,27533	23,639	22,91676	11,01854	18,55962	13,7826
M·ximo	36,52222	33,02384	30,25147	28,03343	26,48968	24,63629	22,10258	15,28088
Rango	10,93775	8,36108	12,97615	4,39443	3,57292	13,61775	3,54296	1,49828
Mediana	34,97443	31,33775	29,13723	27,26742	25,41014	23,49453	19,86383	14,53174
% de Èxito	77,5	61	43	33,5	20	13	6	1

Cuadro A.123: Estadistica tv block 1x1; pkg 100 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	34,99145	31,50913	28,54498	25,70968	21,42614	17,02064	12,84176	10,4265
Varianza	4,67685	4,31448	5,29522	7,96967	9,3794	4,33172	1,70605	0,1277
Error Est·ndar (de la Media)	0,16255	0,16848	0,19176	0,26211	0,32282	0,24876	0,18659	0,20632
MÌnimo	27,6996	25,92829	21,92219	18,06618	11,17473	13,43131	10,30721	10,03803
M·ximo	37,5622	33,98076	31,31675	29,58046	27,20681	21,43743	16,52627	10,74124
Rango	9,86261	8,05248	9,39456	11,51428	16,03209	8,00611	6,21906	0,70321
Mediana	35,53283	32,25293	29,52221	25,99667	21,57204	16,96007	12,65671	10,50023
% de Èxito	88,5	76	72	58	45	35	24,5	1,5

Cuadro A.124: Estadistica tv block 2x2; pkg 100 bytes; img5

1	**10 %**	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	34,37256	30,77189	27,79458	24,61187	20,65743	16,14839	12,43615	8,53032
Varianza	3,06288	2,33724	2,56386	6,16065	5,72845	3,27511	1,69641	0,0494
Error Est·ndar (de la Media)	0,13708	0,13408	0,15701	0,34094	0,34912	0,36941	0,3481	0,15716
MÌnimo	29,26791	23,46372	18,99423	16,81792	14,03714	11,69905	10,52544	8,37316
M·ximo	37,01505	33,39001	30,48163	27,80336	25,08449	19,5858	15,16677	8,68748
Rango	7,74714	9,92629	11,4874	10,98545	11,04735	7,88675	4,64133	0,31432
Mediana	34,71646	31,02136	28,14169	25,03081	20,56029	16,38835	12,20926	8,53032
% de Èxito	81,5	65	52	26,5	23,5	12	7	1

Cuadro A.125: Estadistica tv block 8x8; pkg 100 bytes; img5

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,21993	27,56389	25,55754	23,54776	22,1356	19,56449	16,19489	12,30628
Varianza	4,62205	5,82895	1,36654	1,62034	0,49745	1,74087	3,02869	0,16236
Error Est·ndar (de la Media)	0,17732	0,23232	0,13409	0,17485	0,11755	0,24935	0,39926	0,1802
MÌnimo	14,26231	11,44074	17,62985	15,97415	20,29065	15,9558	12,31133	11,72756
M·ximo	34,54672	30,15356	27,47035	25,01227	23,23241	21,08706	19,02783	12,85467
Rango	20,28441	18,71282	9,8405	9,03812	2,94175	5,13127	6,7165	1,12712
Mediana	31,39034	27,97818	25,72695	23,71868	22,08415	19,74368	16,30921	12,35193
% de Èxito	73,5	54	38	26,5	18	14	9,5	2,5

A.6. Estadísticas Motes

Cuadro A.126: Estadistica bicúbica block 1x1; pkg 27 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	26,45781	22,70217	20,22853	18,40915	16,93194	15,68703	14,5909	13,61946
Varianza	0,54516	0,29969	0,22504	0,16971	0,1136	0,09527	0,04932	0,03263
Error Est·ndar (de la Media)	0,05221	0,03871	0,03354	0,02913	0,02383	0,02183	0,0157	0,01277
MÌnimo	24,55221	21,30942	18,88641	17,3552	16,20947	14,82899	14,12397	13,10565
M·ximo	28,39955	24,0884	21,24235	19,37412	17,79209	16,7959	15,20331	14,07555
Rango	3,84733	2,77897	2,35594	2,01891	1,58262	1,9669	1,07934	0,9699
Mediana	26,52028	22,71768	20,25723	18,37063	16,88163	15,64358	14,56691	13,62318
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.127: Estadistica bicúbica block 2x2; pkg 27 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	26,45781	22,70217	20,22853	18,40915	16,93194	15,68703	14,5909	13,61946
Varianza	0,54516	0,29969	0,22504	0,16971	0,1136	0,09527	0,04932	0,03263
Error Est·ndar (de la Media)	0,05221	0,03871	0,03354	0,02913	0,02383	0,02183	0,0157	0,01277
MÌnimo	24,55221	21,30942	18,88641	17,3552	16,20947	14,82899	14,12397	13,10565
M·ximo	28,39955	24,0884	21,24235	19,37412	17,79209	16,7959	15,20331	14,07555
Rango	3,84733	2,77897	2,35594	2,01891	1,58262	1,9669	1,07934	0,9699
Mediana	26,52028	22,71768	20,25723	18,37063	16,88163	15,64358	14,56691	13,62318
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.128: Estadistica bicúbica block 1x1; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	26,41149	22,65335	20,08678	18,33925	16,88678	15,60439	14,55585	13,59366
Varianza	1,11499	0,66298	0,44869	0,4002	0,23519	0,15688	0,10377	0,0538
Error Est·ndar (de la Media)	0,07467	0,05758	0,04737	0,04473	0,03429	0,02801	0,02278	0,0164
MÌnimo	24,03637	20,57286	18,67125	16,34582	15,39799	14,64842	13,70651	12,94986
M·ximo	29,2711	24,65876	22,43342	19,97889	18,26572	16,85161	15,47703	14,3065
Rango	5,23473	4,0859	3,76217	3,63307	2,86773	2,20319	1,77052	1,35665
Mediana	26,41649	22,69665	20,11313	18,32033	16,84335	15,58301	14,54436	13,6013
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.129: Estadistica bicúbica block 2x2; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,46444	23,43931	20,80422	18,8455	17,31835	15,9229	14,79621	13,65931
Varianza	2,08183	1,13949	0,74721	0,49769	0,3685	0,22938	0,15457	1,03951
Error Est·ndar (de la Media)	0,10203	0,07548	0,06112	0,04988	0,04292	0,03387	0,0278	0,07209
MÌnimo	22,9607	19,81083	18,35536	17,25339	15,80241	14,92507	13,86898	0,
M·ximo	30,6734	26,25376	22,92048	21,76713	19,00326	17,22611	16,01323	14,45659
Rango	7,7127	6,44293	4,56512	4,51374	3,20085	2,30104	2,14425	14,45659
Mediana	27,6057	23,48681	20,7447	18,81615	17,23905	15,88983	14,75389	13,76097
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.130: Estadistica bicúbica block 8x8; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	30,16585	25,69853	22,82194	20,49834	18,58324	17,01831	15,49755	14,30532
Varianza	2,61842	1,4742	1,05067	0,66434	0,54845	0,40682	0,22519	0,15683
Error Est·ndar (de la Media)	0,11442	0,08585	0,07248	0,05763	0,05237	0,0451	0,03356	0,028
MÌnimo	25,40614	22,28557	20,20021	18,1922	16,26386	15,6119	14,29653	13,35863
M·ximo	34,2481	28,61549	25,00789	22,42994	20,68024	18,78213	16,6923	15,52159
Rango	8,84196	6,32992	4,80768	4,23774	4,41638	3,17023	2,39577	2,16296
Mediana	30,03219	25,76893	22,82976	20,52542	18,58874	16,94834	15,49877	14,29651
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.131: Estadistica bilineal block 1x1; pkg 27 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	20,44932	16,14242	13,30991	11,18835	9,36177	7,79437	6,42821	5,14375
Varianza	0,50537	0,16412	0,09417	0,07203	0,05036	0,03285	0,02077	0,01303
Error Est·ndar (de la Media)	0,05027	0,02865	0,0217	0,01898	0,01587	0,01282	0,01019	0,00807
MÌnimo	18,25257	14,99002	12,37569	10,4883	8,68652	7,28609	5,93634	4,89316
M·ximo	22,17034	17,24136	14,34283	11,97329	9,99622	8,26476	6,78024	5,56558
Rango	3,91778	2,25134	1,96714	1,48499	1,3097	0,97867	0,8439	0,67242
Mediana	20,47586	16,11428	13,31527	11,19485	9,37059	7,80501	6,42657	5,1507
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.132: Estadistica bilineal block 2x2; pkg 27 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	21,08411	16,78151	13,90204	11,63891	9,78344	8,15197	6,66449	5,30514
Varianza	0,43762	0,1942	0,10233	0,08275	0,04668	0,03341	0,02349	0,01177
Error Est·ndar (de la Media)	0,04678	0,03116	0,02262	0,02034	0,01528	0,01292	0,01084	0,00767
MÌnimo	19,29934	15,34187	12,96078	10,90586	9,14205	7,72712	6,23247	4,94134
M·ximo	22,60496	17,89032	14,72245	12,37948	10,37455	8,60946	7,00174	5,57305
Rango	3,30562	2,54845	1,76167	1,47362	1,2325	0,88234	0,76927	0,63171
Mediana	21,08228	16,7855	13,89033	11,63886	9,81238	8,14154	6,65826	5,30716
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.133: Estadistica bilineal block 1x1; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	20,27508	16,16796	13,28346	11,14501	9,41853	7,77154	6,41109	5,1203
Varianza	1,16831	0,52533	0,43179	0,24612	0,15892	0,09603	0,0518	0,02931
Error Est·ndar (de la Media)	0,07643	0,05125	0,04646	0,03508	0,02819	0,02191	0,01609	0,01211
MÌnimo	17,59993	13,79924	11,5809	9,88753	8,22945	6,92812	5,52988	4,55614
M·ximo	22,64832	18,17836	14,76367	12,58722	10,58863	8,64143	7,06979	5,55045
Rango	5,04839	4,37912	3,18277	2,69969	2,35918	1,71331	1,53991	0,99431
Mediana	20,37311	16,15412	13,29606	11,14534	9,4389	7,77514	6,41735	5,117
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.134: Estadistica bilineal block 2x2; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	20,12501	16,20021	13,28243	11,17564	9,44664	7,808	6,44631	5,13506
Varianza	1,46505	0,65289	0,38644	0,18277	0,13659	0,08212	0,06323	0,03833
Error Est·ndar (de la Media)	0,08559	0,05714	0,04396	0,03023	0,02613	0,02026	0,01778	0,01384
MÌnimo	16,75371	14,05242	11,52553	10,20454	8,53933	7,1072	5,71	4,65513
M·ximo	22,68999	18,87817	15,06728	12,48401	10,28464	8,49679	7,14727	5,56224
Rango	5,93629	4,82575	3,54175	2,27947	1,74531	1,38959	1,43726	0,90711
Mediana	20,10067	16,16748	13,34392	11,18297	9,48371	7,81889	6,44078	5,12977
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.135: Estadistica bilineal block 8x8; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	27,50995	21,66168	17,97586	14,97478	12,41713	10,22892	8,20868	6,34376
Varianza	2,67571	1,7038	0,99506	0,64517	0,38331	0,29699	0,17766	0,07137
Error Est·ndar (de la Media)	0,11567	0,0923	0,07054	0,0568	0,04378	0,03854	0,0298	0,01889
MÌnimo	23,05896	17,59077	15,52835	12,91458	10,98249	8,95	7,14131	5,59975
M·ximo	33,13185	24,82797	20,95954	16,99756	13,88684	11,61585	9,20356	7,03512
Rango	10,07289	7,2372	5,43119	4,08298	2,90435	2,66585	2,06226	1,43537
Mediana	27,42551	21,61672	17,95709	14,95424	12,44871	10,24438	8,23147	6,34342
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.136: Estadistica cdd block 1x1; pkg 27 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	34,13015	30,86041	28,84955	27,2235	25,85151	24,50519	21,90813	16,60331
Varianza	0,38998	0,26108	0,16715	0,13715	0,11723	0,09049	1,47556	6,23492
Error Est·ndar (de la Media)	0,04862	0,04607	0,04239	0,04364	0,05162	0,05586	0,27162	1,01939
MÌnimo	32,5696	29,83757	28,07286	26,61489	25,15276	23,78917	18,8537	13,16789
M·ximo	35,55199	32,02611	29,76174	27,93267	26,43924	24,97107	23,03579	18,66623
Rango	2,98239	2,18855	1,68889	1,31779	1,28647	1,18191	4,1821	5,49834
Mediana	34,26518	30,87656	28,8223	27,15757	25,90938	24,50673	22,24734	18,03454
% de Èxito	82,5	61,5	46,5	36	22	14,5	10	3

Cuadro A.137: Estadistica cdd block 2x2; pkg 27 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	33,28638	29,96408	27,91787	26,20414	24,76133	23,33696	21,56268	17,63292
Varianza	0,66878	0,33989	0,24264	0,17382	0,2552	0,23824	0,42752	1,27129
Error Est·ndar (de la Media)	0,06445	0,05344	0,05375	0,04983	0,07291	0,08767	0,15858	0,32549
MÌnimo	31,41194	28,25968	26,52601	25,33545	23,48554	22,18361	20,22417	15,35927
M·ximo	36,33659	31,0949	28,88002	27,21627	25,96462	24,22119	22,57367	18,99185
Rango	4,92465	2,83522	2,35402	1,88082	2,47908	2,03758	2,3495	3,63259
Mediana	33,29181	29,98436	27,91084	26,20969	24,74801	23,28466	21,70978	17,69541
% de Èxito	80,5	59,5	42	35	24	15,5	8,5	6

Cuadro A.138: Estadistica cdd block 1x1; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	34,00604	30,81156	28,6679	26,90394	25,07419	22,49467	17,98188	12,29782
Varianza	0,67332	0,41873	0,29537	1,60184	2,31536	6,98621	7,16453	3,53453
Error Est·ndar (de la Media)	0,06099	0,05164	0,04437	0,11506	0,14921	0,28669	0,34271	0,30105
MÌnimo	31,43288	29,2128	27,52486	17,87721	16,43811	14,13034	11,02139	8,15542
M·ximo	36,65033	32,2047	30,10884	28,27128	26,70291	25,19239	23,47862	16,53057
Rango	5,21745	2,99189	2,58398	10,39407	10,26481	11,06205	12,45723	8,37515
Mediana	34,05927	30,84632	28,65331	27,11887	25,48027	23,33822	17,35152	12,10895
% de Èxito	90,5	78,5	75	60,5	52	42,5	30,5	19,5

Cuadro A.139: Estadistica cdd block 2x2; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	80 %
Media	32,98799	29,72805	27,5297	25,33594	23,03101	19,43548	15,18228	10,36264
Varianza	1,26824	0,68019	0,50277	1,9801	3,99458	7,44657	5,24273	3,82059
Error Est·ndar (de la Media)	0,08794	0,07206	0,06953	0,18974	0,28265	0,52517	0,61195	0,79798
MÌnimo	30,63872	27,50806	25,01478	21,07859	16,19063	14,35521	11,31744	8,62291
M·ximo	36,66456	31,4365	28,98843	27,9971	25,52384	22,92676	19,25007	12,81246
Rango	6,02584	3,92844	3,97365	6,91851	9,33321	8,57155	7,93263	4,18954
Mediana	32,99588	29,71873	27,60614	25,59111	23,6504	19,50613	14,89608	9,56486
% de Èxito	82	65,5	52	27,5	25	13,5	7	3

Cuadro A.140: Estadistica cdd block 8x8; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,48156	28,19457	26,00428	24,18028	22,79119	20,54017	16,69583	11,62148
Varianza	1,29804	0,78177	0,45333	0,49387	0,4974	1,45754	3,61902	3,06696
Error Est·ndar (de la Media)	0,09122	0,08005	0,07219	0,08225	0,10632	0,21683	0,47559	0,71496
MÌnimo	28,97144	25,92734	24,12291	22,22541	20,87221	17,53837	13,23321	9,0613
M·ximo	34,7593	30,41775	27,74046	25,57287	23,99602	22,09332	19,48455	13,42803
Rango	5,78786	4,49041	3,61755	3,34746	3,12382	4,55494	6,25134	4,36673
Mediana	31,41901	28,17635	26,00369	24,13679	22,89765	20,72461	16,53314	11,85072
% de Èxito	78	61	43,5	36,5	22	15,5	8	3

Cuadro A.141: Estadistica msr block 1x1; pkg 27 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	18,10238	12,86228	10,34722	9,0588	7,80212	8,00687	3,37383	2,12454
Varianza	75,34415	34,26453	13,47673	6,93006	0,23096	0,9485	NaN	NaN
Error Est·ndar (de la Media)	0,64878	0,5556	0,43878	0,48884	0,15197	0,56229	NaN	NaN
MÌnimo	7,06602	7,08619	7,03138	7,12974	7,32852	6,88597	3,37383	2,12454
M·ximo	28,69715	23,95849	20,94093	17,8107	8,82421	8,64601	3,37383	2,12454
Rango	21,63114	16,87229	13,90954	10,68096	1,49569	1,76004	0,	0,
Mediana	13,02069	9,96527	9,23203	8,06605	7,75626	8,48862	3,37383	2,12454
% de Èxito	89,5	55,5	35	14,5	5	1,5	0,5	0,5

Cuadro A.142: Estadistica msr block 2x2; pkg 27 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	23,97575	19,83661	17,38708	15,40891	13,75702	12,34317	11,00016	9,61898
Varianza	4,0113	1,69146	1,21925	0,67494	0,48684	0,34747	0,30148	0,14977
Error Est·ndar (de la Media)	0,14162	0,09196	0,07808	0,05809	0,04934	0,04168	0,03882	0,02736
MÌnimo	18,35209	16,66232	13,97442	13,52754	12,12218	10,68491	9,85784	8,44747
M·ximo	28,96336	23,26118	20,95514	17,45414	15,68043	14,01068	12,78167	10,66976
Rango	10,61126	6,59886	6,98072	3,9266	3,55825	3,32577	2,92383	2,22228
Mediana	23,84339	19,71349	17,4063	15,30961	13,7756	12,33601	10,95251	9,593
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.143: Estadistica msr block 1x1; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	24,32601	17,9367	14,80171	13,39854	9,79224	8,49601	8,80226	6,11835
Varianza	51,218	57,55445	41,49763	24,67024	20,93111	12,93829	4,75739	0,79477
Error Est·ndar (de la Media)	0,55215	0,64816	0,60071	0,52947	0,56747	0,54853	0,48772	0,33695
MÌnimo	4,2087	4,16747	4,11013	4,05909	3,52672	3,6518	6,70163	5,3416
M·ximo	28,80949	24,62069	21,55503	18,85753	16,51327	14,53266	12,36452	7,9114
Rango	24,60079	20,45323	17,4449	14,79844	12,98655	10,88086	5,66288	2,5698
Mediana	27,09388	22,55476	19,4582	16,70301	8,92367	8,13049	7,54157	5,92944
% de Èxito	84	68,5	57,5	44	32,5	21,5	10	3,5

Cuadro A.144: Estadistica msr block 2x2; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	23,60973	19,68081	16,85772	14,86577	13,59746	12,10866	10,962	9,73145
Varianza	15,48342	7,92074	4,00955	2,3577	1,6367	0,7295	0,4453	0,25734
Error Est·ndar (de la Media)	0,27824	0,19901	0,14159	0,10857	0,09046	0,06039	0,04719	0,03587
MÌnimo	15,15437	13,65551	12,28657	11,57817	11,05515	10,62659	9,68466	8,69723
M·ximo	29,62815	24,3063	21,57726	19,04341	17,05857	14,5877	12,71981	11,05822
Rango	14,47379	10,65079	9,29069	7,46524	6,00343	3,96111	3,03516	2,36099
Mediana	25,6056	19,18198	16,7319	14,77752	13,57634	12,04218	10,9348	9,6434
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.145: Estadistica msr block 8x8; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	28,39924	24,73397	22,20749	20,22588	18,4096	16,89702	15,36902	14,12852
Varianza	1,14437	0,78607	0,57764	0,42865	0,43276	0,37168	0,23859	0,17327
Error Est·ndar (de la Media)	0,07564	0,06269	0,05374	0,0463	0,04652	0,04311	0,03454	0,02943
MÌnimo	25,04129	22,11597	20,26954	17,61031	16,81821	15,49502	14,03336	13,05674
M·ximo	31,17121	27,20933	24,08871	21,68854	20,55936	18,55058	16,74362	15,33715
Rango	6,12992	5,09336	3,81916	4,07823	3,74115	3,05556	2,71027	2,2804
Mediana	28,36046	24,84717	22,28625	20,21288	18,47071	16,87858	15,35609	14,07868
% de Èxito	100	100	100	100	100	100	100	100

Cuadro A.146: Estadistica tv block 1x1; pkg 27 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	33,58248	30,02588	28,21327	26,11509	24,61676	22,91904	20,09882	14,90357
Varianza	5,6606	5,73472	1,33765	1,63608	1,48168	2,0128	2,00375	1,41493
Error Est·ndar (de la Media)	0,17883	0,20096	0,10602	0,12086	0,13525	0,16274	0,19087	0,21028
MÌnimo	13,816	11,84042	23,12229	22,17092	21,31364	18,82475	15,327	12,01586
M·ximo	35,54125	31,92246	29,89353	27,71862	26,33013	24,74415	22,88762	17,31429
Rango	21,72525	20,08205	6,77125	5,5477	5,01648	5,9194	7,56061	5,29843
Mediana	34,11634	30,56064	28,51809	26,43849	24,95981	23,38495	20,01987	14,91768
% de Èxito	88,5	71	59,5	56	40,5	38	27,5	16

Cuadro A.147: Estadistica tv block 2x2; pkg 27 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	32,85636	29,67703	27,35816	25,5837	24,25738	22,52753	19,95752	15,65052
Varianza	6,15868	1,96563	4,71533	2,00528	1,00528	0,66119	2,13577	0,94334
Error Est·ndar (de la Media)	0,19998	0,12641	0,23018	0,17172	0,15853	0,16598	0,39058	0,48563
MÌnimo	16,30439	16,54821	12,34125	16,06874	19,81609	20,65679	17,10067	15,04684
M·ximo	36,68031	31,21624	28,87741	27,07446	25,74244	23,92983	21,71255	17,08254
Rango	20,37592	14,66803	16,53616	11,00573	5,92635	3,27304	4,61189	2,0357
Mediana	33,16345	29,84128	27,76911	25,9149	24,3416	22,52885	20,23724	15,23634
% de Èxito	77	61,5	44,5	34	20	12	7	2

Cuadro A.148: Estadistica tv block 1x1; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	33,36475	30,00549	27,52294	25,57136	22,41578	18,24768	13,81467	10,59179
Varianza	6,00862	4,59741	3,96987	4,89739	6,61596	7,9687	3,04853	0,40271
Error Est·ndar (de la Media)	0,18321	0,17507	0,16604	0,20636	0,25983	0,31561	0,24213	0,18319
MÌnimo	25,14636	24,33156	20,41138	18,23061	13,0827	12,76815	10,18607	10,1378
M·ximo	36,77118	32,17006	29,99104	27,94231	26,15389	23,72411	18,04404	12,17982
Rango	11,62482	7,8385	9,57966	9,71171	13,0712	10,95596	7,85798	2,04202
Mediana	34,06056	30,73606	28,33729	26,32973	22,37329	18,54343	13,82295	10,29578
% de Èxito	89,5	75	72	57,5	49	40	26	6

Cuadro A.149: Estadistica tv block 2x2; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	32,40254	29,36317	26,65784	24,40873	21,43623	17,21856	13,4364	8,50689
Varianza	4,12836	1,83082	2,29275	3,41215	4,1177	3,62564	1,78192	0,04119
Error Est·ndar (de la Media)	0,15866	0,11913	0,15218	0,24908	0,29919	0,38867	0,40248	0,11718
MÌnimo	26,29328	24,55665	20,96636	17,1913	16,2458	12,63996	11,63063	8,27271
M·ximo	36,65065	31,47199	28,81008	26,83849	25,21841	19,83659	15,96604	8,63176
Rango	10,35737	6,91534	7,84372	9,64719	8,97261	7,19663	4,33542	0,35905
Mediana	32,88547	29,58971	27,11795	24,81858	21,71403	17,63781	13,13908	8,6162
% de Èxito	82	64,5	49,5	27,5	23	12	5,5	1,5

Cuadro A.150: Estadistica tv block 8x8; pkg 100 bytes; img6

	10 %	**20 %**	**30 %**	**40 %**	**50 %**	**60 %**	**70 %**	**80 %**
Media	31,13881	27,92128	25,52508	23,60996	22,03792	19,50522	15,76111	11,22011
Varianza	9,90648	4,14217	3,95515	5,48263	1,79449	4,17417	3,35721	1,20244
Error Est·ndar (de la Media)	0,2596	0,19768	0,22097	0,33114	0,2406	0,41704	0,44439	0,44767
MÌnimo	10,5825	10,523	10,48351	12,89404	16,64826	12,53399	11,95234	10,01394
M·ximo	35,21151	31,49752	27,52619	26,7253	23,7825	21,47618	18,19826	12,55795
Rango	24,62902	20,97453	17,04268	13,83126	7,13424	8,94219	6,24593	2,544
Mediana	31,45326	28,06873	25,94961	24,06975	22,23347	20,18201	15,90117	11,14122
% de Èxito	73,5	53	40,5	25	15,5	12	8,5	3

Apéndice B

Algoritmos en lenguaje C++

B.1. RebuilderTV.h

```
#ifndef CLASS_REBUILDERTV
#define CLASS_REBUILDERTV

#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))

#include "Rebuilder.h"
#include <iostream>
#include <vector>

using namespace std;

class RebuilderTV: public Rebuilder{

public:

RebuilderTV(){};

~RebuilderTV(){};

virtual void hidden(Images* img, bool show_data, bool export_images, HEAD *
    header){
        /**
        * @brief Metodo que oculta los pixeles dados y los reemplaza por
            el promedio de sus vecinos mas cercanos
```

```
        * @param img Imagen sobre la que se le aplicar  el m todo de
            ocultamiento de errores
        *
        */
img->clearBlocks(header);
DataBlock** matrix = img->getMatrix();
int amount_channels = img->getType();
int w = img->getWidth()/img->getWidthBlock();
int h = img->getHeight()/img->getHeightBlock();
    // estas son las ecuaciones de TV inpainting
     float V[img->getWidth()][img->getHeight()][amount_channels];
     float U[img->getWidth()][img->getHeight()][amount_channels];

float gridUw2,gridUe2,gridUs2,gridUn2,w1,w2,w3,w4;
    int n=1;
    // la cantidad de iteraciones para realizar el inpainting
    int iteraciones=500;
    int umbral=252;
    //binarizando imagen
for(int i = 0; i < h; i++){
for(int j = 0; j < w; j++){
vector<int> add(amount_channels); //
for(int x=0;x<amount_channels;x++) add[x]=0;//se inicializa el acomulador
for(int iBlock = max((int)i-1,0); iBlock <= min(i+1,h-1) ; iBlock++)   /**/
for(int jBlock = max((int)j-1,0); jBlock <= min(j+1,w-1); jBlock++){ /** 8
    conectados */
vector<Pixel> *e = (vector<Pixel>*)matrix[iBlock][jBlock].getExtras();//
    contiene todo los pixeles de un bloque expecificio
                    for(int r=0;r<img->getHeightBlock();r++){
                        for(int c=0;c<img->getWidthBlock();c++){
                            vector<int> chnls = e->at((r*img->getWidthBlock())
                                +c).getChannels();//varible con los pixeles de
                                 cada canal
                            for(int x=0;x<amount_channels;x++){
                               V[i][j][x]=chnls[x];
                               U[i][j][x]=chnls[x];
                            if (chnls[x]>=umbral){
                                 add[x]=0;
                            }
                            else {
                                 add[x]=255;
                            }
```

```
DataBlock* px = new DataBlock(amount_channels,add,img->getWidthBlock(),img->
    getHeightBlock());
                                matrix[i][j]= *px;
                               }
                           }
                       }
                   }
               }

 //*********************************************************************

    //ahora se aplica el inpainting a la imagen binaria
    while (n<=iteraciones) {
    for(int i = 0; i < h; i++){
        for(int j = 0; j < w; j++){
    //    if((!matrix[i][j].isValid()) && amountNeighbors(img,i,j)>=neighbors)
        {// si el bloque no es v lido
            vector<int> add(amount_channels); //
            for(int x=0;x<amount_channels;x++) add[x]=0;//se inicializa el
                acomulador
            for(int iBlock = max((int)i-1,0); iBlock <= min(i+1,h-1) ; iBlock
                ++)   /**/
                for(int jBlock = max((int)j-1,0); jBlock <= min(j+1,w-1);
                    jBlock++){ /** 8 conectados */
                    vector<Pixel> *e = (vector<Pixel>*)matrix[iBlock][jBlock].
                        getExtras();//contiene todo los pixeles de un bloque
                        expecificio
                    for(int r=0;r<img->getHeightBlock();r++){
                        for(int c=0;c<img->getWidthBlock();c++){
                            vector<int> chnls = e->at((r*img->getWidthBlock())
                                +c).getChannels();//varible con los pixeles de
                                cada canal
                            for(int x=0;x<amount_channels;x++){
                            if (chnls[x]==0){
                                // se generan los vectores de inpainting al
                                    rededor del pixel en cuestion
gridUw2=pow(V[i][j][x]-V[i-1][j][x], 2)+pow((V[i-1][j+1][x]+V[i][j+1][x]-V[i
    -1][j-1][x]-V[i][j-1][x])/2, 2);
gridUe2=pow(V[i][j][x]-V[i+1][j][x], 2)+pow((V[i][j+1][x]+V[i+1][j+1][x]-V[i][
    j-1][x]-V[i+1][j-1][x])/2, 2);
```

```
gridUs2=pow(V[i][j][x]-V[i][j-1][x], 2)+pow((V[i+1][j][x]+V[i+1][j-1][x]-V[i
    -1][j][x]-V[i-1][j-1][x])/2, 2);
gridUn2=pow(V[i][j][x]-V[i][j+1][x], 2)+pow((V[i+1][j][x]+V[i+1][j+1][x]-V[i
    -1][j][x]-V[i-1][j+1][x])/2, 2);
w1 = 1/sqrt(1+gridUw2);
w2 = 1/sqrt(1+gridUe2);
w3 = 1/sqrt(1+gridUs2);
w4 = 1/sqrt(1+gridUn2);
//Aqui se van guardando los pixeles regenerados a partir de TV inpainting
U[i][j][x] =(w1*V[i-1][j][x]+w2*V[i+1][j][x]+w3*V[i][j+1][x]+w4*V[i][j-1][x])
    /(w1+w2+w3+w4);

                                }

                                }
                            }
                        }
                    }
                }
            }
        n++;
        for(int i = 0; i < h; i++){
            for(int j = 0; j < w; j++){
                for(int x=0;x<amount_channels;x++){
                    V[i][j][x]=U[i][j][x];
                }
            }
        }
        }
        for(int i = 0; i < h; i++){
            for(int j = 0; j < w; j++){
                vector<int> add(amount_channels); //
                for(int x=0;x<amount_channels;x++){
                    add[x]=floor(V[i][j][x]);//antes era U
                }
                DataBlock* px = new DataBlock(amount_channels,add,img->
                    getWidthBlock(),img->getHeightBlock());
                matrix[i][j]= *px;
            }
        }
        if(export_images){
                    string path = header->folder+"/image_restored_tv.bmp";
```

```
            img->save(path.c_str());
            }
            if(show_data)
                cout << "The Image has been restored por TV\n";
        }

private:

        int amountNeighbors(Images* img, int i, int j){
        /**
        * @brief Metodo que calcula la cantidad de vecinos vlidos que tiene
            un pixel de la imagen img en la posicin (i,j)
        * @param img Imagen que contiene el pixel en la posicin (i,j)
        * @param i fila en la que est ubicada el pixel
        * @param j columna en la que est ubicada el pixel
        * @return neighbors
        */
                int w = img->getWidth()/img->getWidthBlock();
                int h = img->getHeight()/img->getHeightBlock();
                DataBlock** matrix=img->getMatrix();
                int neighbors=0;
                for(int iBlock = max((int)i-1,0); iBlock <= min(i+1,h-1) ;
                    iBlock++)   /**/
                for(int jBlock = max((int)j-1,0); jBlock <= min(j+1,w-1);
                    jBlock++) /** 8 conectados */
                    if((i != iBlock) || (j != jBlock))
                        if(matrix[iBlock][jBlock].isValid()) // si el bloque
                            vecino existe
                        {
                            neighbors++;
                        }
            //cout << "(" << i << "," << j << "):" << neighbors << "--\n";
            return neighbors;
        }

        bool isLossBlock(Images* img){
        /**
        * @brief Metodo que valida si existen pixeles perdidos en la imagen
        * @param img Imagen sobre la que se buscan los pixeles perdidos
        * @return true si hay pixeles perdidos en la imagen
        * @return false si no hay pixeles perdidos en la imagen
        */
```

```
                DataBlock** matrix = img->getMatrix();
                int w=img->getWidth()/img->getWidthBlock();
                int h=img->getHeight()/img->getHeightBlock();
                for (int i = 0; i < h; ++i)
                {
                        for (int j = 0; j < w; ++j)
                        {
                                if(!matrix[i][j].isValid())
                                        return true;
                        }
                }
                return false;
        }

};
#endif
```

B.2. RebuilderCDD.h

```
#ifndef CLASS_REBUILDERCDD
#define CLASS_REBUILDERCDD

#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))

#include "Rebuilder.h"
#include <iostream>
#include <vector>

using namespace std;

class RebuilderCDD: public Rebuilder{

public:

    RebuilderCDD(){};

    ~RebuilderCDD(){};
```

```
virtual void hidden(Images* img, bool show_data, bool export_images, HEAD
    *header){
     /**
      * @brief M todo que oculta los pixeles da ados
      * @param img Imagen sobre la que se le aplicar  el m todo de
          ocultamiento de errores
      *
      */
     img->clearBlocks(header);
     DataBlock** matrix = img->getMatrix();
     int amount_channels = img->getType();
     int w = img->getWidth()/img->getWidthBlock();
     int h = img->getHeight()/img->getHeightBlock();
     // matrices auxiliares para CDD
     float V[img->getWidth()][img->getHeight()][amount_channels];
     float U[img->getWidth()][img->getHeight()][amount_channels];
     float R[img->getWidth()][img->getHeight()][amount_channels];

     float gridUw2,gridUe2,gridUs2,gridUn2,w1,w2,w3,w4;
     float gridw2, gride2, grids2, gridn2;
     float a1,a2,a3,a4,k;
     int a=5;
     int n=1;
     //    // la cantidad de iteraciones para realizar el inpainting
     int iteraciones=400;
for(int i = 0; i < h; i++){
     for(int j = 0; j < w; j++){
    if((!matrix[i][j].isValid())){// si el bloque no es v lido
                for(int r=0;r<img->getHeightBlock();r++){
                    for(int c=0;c<img->getWidthBlock();c++){
                        for(int x=0;x<amount_channels;x++){
                            V[i][j][x]=255;
                            U[i][j][x]=255;
                            R[i][j][x]=0;
                        }
                    }
                }
    }//comentar para quitar validacion de perdida
    else{
        vector<Pixel> *e = (vector<Pixel>*)matrix[i][j].getExtras();//
            contiene todo los pixeles de un bloque expecificio
             for(int r=0;r<img->getHeightBlock();r++){
```

```
                    for(int c=0;c<img->getWidthBlock();c++){
                        vector<int> chnls = e->at((r*img->getWidthBlock())+c).
                            getChannels();//varible con los pixeles de cada
                            canal
                        for(int x=0;x<amount_channels;x++){
                           V[i][j][x]=chnls[x];
                           U[i][j][x]=chnls[x];
                           R[i][j][x]=255;
                        }
                    }
                }
            }
         }
        }
      cout << "imagen binarizada\n";

    //ahora se aplica el inpainting a la imagen binaria
    while (n<=iteraciones) {
    for(int i = 0; i < h; i++){
        for(int j = 0; j < w; j++){
            for(int x=0;x<amount_channels;x++){
                if (R[i][j][x]==0){
//                  if(i>0&&i<h-1&&j>0&&j<w-1){
gridw2=pow(V[i][j][x]-V[i-1][j][x], 2)+pow((V[i-1][j-1][x]-V[i-1][j+1][x])/2,
    2);
gride2=pow(V[i][j][x]-V[i+1][j][x], 2)+pow((V[i+1][j-1][x]-V[i+1][j+1][x])/2,
    2);
grids2=pow(V[i][j][x]-V[i][j-1][x], 2)+pow((V[i-1][j-1][x]-V[i+1][j-1][x])/2,
    2);
gridn2=pow(V[i][j][x]-V[i][j+1][x], 2)+pow((V[i-1][j+1][x]-V[i+1][j+1][x])/2,
    2);
a1 = 1/sqrt(a+gridw2);
a2 = 1/sqrt(a+gride2);
a3 = 1/sqrt(a+grids2);
a4 = 1/sqrt(a+gridn2);
k =(a1*V[i-1][j][x]+a2*V[i+1][j][x]+a3*V[i][j+1][x]+a4*V[i][j-1][x])/(a1+a2+a3
    +a4);

gridUw2=pow(V[i][j][x]-V[i-1][j][x], 2)+pow((V[i-1][j-1][x]-V[i-1][j+1][x])/2,
     2);
gridUe2=pow(V[i][j][x]-V[i+1][j][x], 2)+pow((V[i+1][j-1][x]-V[i+1][j+1][x])/2,
     2);
```

```
gridUs2=pow(V[i][j][x]-V[i][j-1][x], 2)+pow((V[i-1][j-1][x]-V[i+1][j-1][x])/2,
    2);
gridUn2=pow(V[i][j][x]-V[i][j+1][x], 2)+pow((V[i-1][j+1][x]-V[i+1][j+1][x])/2,
    2);
w1 = k/sqrt(gridUw2+a);
w2 = k/sqrt(gridUe2+a);
w3 = k/sqrt(gridUs2+a);
w4 = k/sqrt(gridUn2+a);
//Aqui se van guardando los pixeles regenerados a partir de CDD inpainting
U[i][j][x]=(w1*V[i-1][j][x]+w2*V[i+1][j][x]+w3*V[i][j+1][x]+w4*V[i][j-1][x])/(
    w1+w2+w3+w4);
//                  }
//                  else{
//                      cout<<"pixel de borde\n";
//                  }
                 }
                 }
             }
         }
     n++;
     for(int i = 0; i < h; i++){
         for(int j = 0; j < w; j++){
             for(int x=0;x<amount_channels;x++){
                 V[i][j][x]=U[i][j][x];
             }
         }
     }
         }
     for(int i = 0; i < h; i++){
         for(int j = 0; j < w; j++){
             vector<int> add(amount_channels); //
             for(int x=0;x<amount_channels;x++){
                 add[x]=floor(V[i][j][x]);//
             }
             DataBlock* px = new DataBlock(amount_channels,add,img->
                 getWidthBlock(),img->getHeightBlock());
             matrix[i][j]= *px;
            }
        }
         if(export_images){
             string path = header->folder+"/image_restored_by_CDD.bmp";
             img->save(path.c_str());
```

```
        }
        if(show_data)
            cout << "The Image has been restored by CDD\n";
    }

private:

    int amountNeighbors(Images* img, int i, int j){
        /**
         * @brief Metodo que calcula la cantidad de vecinos vlidos que
             tiene un pixel de la imagen img en la posicin (i,j)
         * @param img Imagen que contiene el pixel en la posicin (i,j)
         * @param i fila en la que est ubicada el pixel
         * @param j columna en la que est ubicada el pixel
         * @return neighbors
         */
        int w = img->getWidth()/img->getWidthBlock();
        int h = img->getHeight()/img->getHeightBlock();
        DataBlock** matrix=img->getMatrix();
        int neighbors=0;
        for(int iBlock = max((int)i-1,0); iBlock <= min(i+1,h-1) ; iBlock++)
             /**/
            for(int jBlock = max((int)j-1,0); jBlock <= min(j+1,w-1); jBlock
               ++) /** 8 conectados */
                if((i != iBlock) || (j != jBlock))
                    if(matrix[iBlock][jBlock].isValid()) // si el bloque
                        vecino existe
                    {
                        neighbors++;
                    }
        //cout << "(" << i << "," << j << "):" << neighbors << "--\n";
        return neighbors;
    }

    bool isLossBlock(Images* img){
        /**
         * @brief Metodo que valida si existen pixeles perdidos en la imagen
         * @param img Imagen sobre la que se buscan los pixeles perdidos
         * @return true si hay pixeles perdidos en la imagen
         * @return false si no hay pixeles perdidos en la imagen
         */
        DataBlock** matrix = img->getMatrix();
```

```
        int w=img->getWidth()/img->getWidthBlock();
        int h=img->getHeight()/img->getHeightBlock();
        for (int i = 0; i < h; ++i)
        {
            for (int j = 0; j < w; ++j)
            {
                if(!matrix[i][j].isValid())
                    return true;
            }
        }
        return false;
    }

};
#endif
```

B.3. RebuilderBilineal.h

```
#ifndef CLASS_REBUILDERBILINEAL
#define CLASS_REBUILDERBILINEAL

#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))

#include "Rebuilder.h"
#include <iostream>
#include <vector>

using namespace std;

class RebuilderBilineal: public Rebuilder{

public:

        RebuilderBilineal(){};

        ~RebuilderBilineal(){};
```

```
        virtual void hidden(Images* img, bool show_data, bool export_images,
           HEAD *header){
        /**
        * @brief Metodo que oculta los pixeles dados y los reemplaza por
            el promedio de sus vecinos mas cercanos
        * @param img Imagen sobre la que se le aplicar  el metodo de
            ocultamiento de errores
        *
        */
//          img->clearBlocks(header);
        DataBlock** matrix = img->getMatrix();
        int amount_channels = img->getType();
        int w = img->getWidth()/img->getWidthBlock();
        int h = img->getHeight()/img->getHeightBlock();
        float reparador;
        do{
            for(int i = 0; i < h; i++){
                for(int j = 0; j < w; j=j+1){
                    if(!matrix[i][j].isValid()){
                        vector<int> add(amount_channels);//vector donde se
                            guardaran los valores de las bandas(o canales) de
                            un pixel
                        reparador=0;//valor de cada banda o canal
                        vector<Pixel> punto;//vector de pixeles para construir
                             el bloque a reconstruir

                        for(int r=0;r<img->getHeightBlock();r++){
                            for(int c=0;c<img->getWidthBlock();c++){
                                for(int x=0;x<amount_channels;x++){
                                    if((i>0) & (i<h-1) & (j>0) & (j<w-1)){//
                                        bloques interiores
                                        //cout << "reparando bloques
                                            interiores\n";
                                        reparador=(nn(img,i,j,c,x)+(ss(img,i,j
                                            ,c,x))+(ee(img,i,j,r,x))+(oo(img,i,
                                            j,r,x)))/4;
                                    }
                                    else{
                                        if((i>0&&i<h-1) && (j==0 | j==w-1)){//
                                            primera y ultima columna sin
                                            conciderar las esquinas
                                             if(j==0){
```

```
                //cout << " reparando primera
                    columna\n";
                reparador=(ss(img,i,j,c,x)+nn(img,
                    i,j,c,x)+ee(img,i,j,r,x))/3;
                }
                else{
                //cout << " reparando ultima
                    columna\n";
                reparador=(ss(img,i,j,c,x)+nn(img,
                    i,j,c,x)+oo(img,i,j,r,x))/3;
                }
            }
            else{
                if((i==0 | i==h-1) && (j>0&&j<w-1)
                    ){//primera y ultima fila sin
                    conciderar las esquinas
                    if(i==0){
                    //cout << "reparando primera
                        fila\n";
                    reparador=(oo(img,i,j,r,x)+ee(
                        img,i,j,r,x)+ss(img,i,j,c,x
                        ))/3;
                    }
                    else{
                    //cout << "reparando ultima
                        fila\n";
                    reparador=(oo(img,i,j,r,x)+ee(
                        img,i,j,r,x)+nn(img,i,j,c,x
                        ))/3;
                    }
                }
                else{
                    if(i==0 & j==0){//esquina
                        superios izquierda
                      //  cout << "reparando
                            esquina superios
                            izquierda\n";
                         reparador=((ss(img,i,j,c,x
                             ))+(ee(img,i,j,r,x)))
                             /2;
                    }
                    else{
```

```
                    if(i==0&j==w-1){//esquina
                        superior derecha
                    //cout << "reparando
                        esquina superior
                        derecha\n";
                        reparador=((ss(img,i,j
                            ,c,x))+(oo(img,i,j,
                            r,x)))/2;
                    }
                    else{
                        if(i==h-1 & j==0){//
                            esquina inferiro
                            izquierda
                    //cout << "reparando
                        esquina inferiro
                        izquierda\n";
                            reparador=((nn(img
                                ,i,j,c,x))+(ee(
                                img,i,j,r,x)))
                                /2;
                        }
                        else{//esquina
                            inferior derecha
                    //cout << "reparando
                        esquina inferior
                        derecha\n";
                          reparador=((nn(img,i,
                              j,c,x))+(oo(img,i,
                              j,r,x)))/2;
                        }
                    }

                }

            }

        }
    add[x]=floor(reparador);//actualizando el
        vector con el valor de la banda o canal
        "x"
```

```
                        }

                            Pixel aux(add,amount_channels); //creando el pixel
                                en su respectivo formato
                            punto.push_back(aux);//creando el vector de pixel
                               que con el cual se construira el bloque
                               recuperado
                            }
                        }//aqui se pasa el vector de pixel al constructor
                           DataBlock en que debuelve un bloque reparado
                        DataBlock* db = new DataBlock(punto,img->getWidthBlock
                           (),img->getHeightBlock(),amount_channels);

//              vector<Pixel> *e = (vector<Pixel>*)db->getExtras();
//              for(int r=0;r<img->getHeightBlock();r++){
//                                                  for(int c=0;c<img->
    getWidthBlock();c++){
//                                                      vector<int>
    chnls = e->at((r*img->getWidthBlock())+c).getChannels();
//                                                      for(int x=0;x<
    amount_channels;x++){
//
                    cout<<"pixel desde bloque creado"<<chnls[x]<<"\n";
//
            }
//                                              }
//              }

                            matrix[i][j]= *db;//actualizando la matrix de bloques
                               con el nuevo bloque reconstruido
                    }
                        }
                    }
            }while(isLossBlock(img));
            if(export_images){
                string path = header->folder+"/image_received_bilineal.bmp";
            img->save(path.c_str());
            }
            if(show_data)
                cout << "The Image has been restored by bilineal\n";
        }
```

```
private:

        bool isLossBlock(Images* img){
        /**
        * @brief Metodo que valida si existen pixeles perdidos en la imagen
        * @param img Imagen sobre la que se buscan los pixeles perdidos
        * @return true si hay pixeles perdidos en la imagen
        * @return false si no hay pixeles perdidos en la imagen
        */
                DataBlock** matrix = img->getMatrix();
                int w=img->getWidth()/img->getWidthBlock();
                int h=img->getHeight()/img->getHeightBlock();
                for (int i = 0; i < h; ++i)
                {
                        for (int j = 0; j < w; ++j)
                        {
                                if(!matrix[i][j].isValid())
                                        return true;
                        }
                }
                return false;
        }
        int nn(Images* img, int i, int j,int jj, int x){
        /**
        * @brief Metodo que devuelve el pixel inferior del bloque superior
            contiguo al pixel perdido
        * @param img Imagen que contiene el pixel en la posici n (i,j)
        * @param i fila en la que est ubicada el bloque afectado
        * @param j columna en la que est ubicada el el bloque afectado
        * @param jj ubicacion del pixel en el bloque con perdida
        * @param x banda de interes
        * @return pixel
        */
         //   cout << "bloque superior \n";
            DataBlock** matrix=img->getMatrix();
            int pixel;
            if(!matrix[i-1][j].isValid()){
                pixel=127;
            }
            else{
            vector<Pixel> *e = (vector<Pixel>*)matrix[i-1][j].getExtras();
```

```
            vector<int> chnls = e->at(((img->getHeightBlock()-1)*img->
                getWidthBlock())+jj).getChannels();
            pixel=chnls[x];
            }
//            cout << "pixel superior: " << pixel << "\n";
            return pixel;
        }
        int oo(Images* img, int i, int j,int ii, int x){
        /**
        * @brief Metodo que devuelve el pixel derecho del bloque izquierdo
            contiguo al pixel perdido
        * @param img Imagen que contiene el pixel en la posicion (i,j)
        * @param i fila en la que esta ubicada el bloque afectado
        * @param j columna en la que esta ubicada el el bloque afectado
        * @param ii ubicacion del pixel en el bloque con perdida
        * @param x banda de interes
        * @return pixel
        */
            //cout << "bloque izquierdo \n";
            DataBlock** matrix=img->getMatrix();
            int pixel;
            if(!matrix[i][j-1].isValid()){
                pixel=127;
            }
            else{
            vector<Pixel> *e = (vector<Pixel>*)matrix[i][j-1].getExtras();
            vector<int> chnls = e->at(((ii)*img->getWidthBlock())+(img->
                getWidthBlock()-1)).getChannels();
            pixel=chnls[x];
            }
            //cout << "pixel izquierdo: " << pixel << "\n";
            return pixel;
        }
        int ee(Images* img, int i, int j,int ii, int x){
        /**
        * @brief Metodo que devuelve el pixel derecho del bloque izquierdo
            contiguo al pixel perdido
        * @param img Imagen que contiene el pixel en la posicion (i,j)
        * @param i fila en la que esta ubicada el bloque afectado
        * @param j columna en la que esta ubicada el el bloque afectado
        * @param ii ubicacion del pixel en el bloque con perdida
        * @param x banda de interes
```

```
* @return pixel
*/
    //cout << "bloque derecho \n";
    DataBlock** matrix=img->getMatrix();
    int pixel;
    if(!matrix[i][j+1].isValid()){
       pixel=127;
    }
    else{
    vector<Pixel> *e = (vector<Pixel>*)matrix[i][j+1].getExtras();
    vector<int> chnls = e->at(((ii)*img->getWidthBlock())).getChannels
        ();
    pixel=chnls[x];
    }
    //cout << "pixel derecho: " << pixel << "\n";
    return pixel;
}
int ss(Images* img, int i, int j,int jj, int x){
/**
* @brief M todo que devuelve el pixel superior  del bloque inferior
    contiguo al pixel perdido
* @param img Imagen que contiene el pixel en la posici n (i,j)
* @param i fila en la que est  ubicada el bloque afectado
* @param j columna en la que est  ubicada el el bloque afectado
* @param jj ubicacion del pixel en el bloque con perdida
* @param x banda de interes
* @return pixel
*/
    //cout << "bloque inferior\n";
    DataBlock** matrix=img->getMatrix();
    int pixel;
    if(!matrix[i+1][j].isValid()){//si no existe el bloque se devuelve
        0
        pixel=127;
    }
    else{
    vector<Pixel> *e = (vector<Pixel>*)matrix[i+1][j].getExtras();//
       los elementos del bloque se recuperan en un vector
    vector<int> chnls = e->at(jj).getChannels();//se busca el elemento
        de interes
    pixel=chnls[x];//el correspondiente canal del elemento de interes
    }
```

```
            //cout << "pixel inferior: " << pixel << "\n";
            return pixel;
        }

};
#endif
```

B.4. RebuilderBicubico.h

```
#ifndef CLASS_REBUILDERBICUBICO
#define CLASS_REBUILDERBICUBICO

#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))

#include "Rebuilder.h"
#include <iostream>
#include <vector>

using namespace std;

class RebuilderBicubico: public Rebuilder{

public:

        RebuilderBicubico(){};

        ~RebuilderBicubico(){};

        virtual void hidden(Images* img, bool show_data, bool export_images,
           HEAD *header){
        /**
        * @brief Metodo que oculta los pixeles dados y los reemplaza por
            el promedio de sus vecinos mas cercanos
        * @param img Imagen sobre la que se le aplicar  el metodo de
            ocultamiento de errores
        *
        */
//        img->clearBlocks(header);
```

```
        DataBlock** matrix = img->getMatrix();
        int amount_channels = img->getType();
        int w = img->getWidth()/img->getWidthBlock();
        int h = img->getHeight()/img->getHeightBlock();
        float reparador,f_i_j,df_dc,df_dr,d2f_dcdr;

        do{
            for(int i = 0; i < h; i++){
                for(int j = 0; j < w; j=j+1){
                    if(!matrix[i][j].isValid()){
                        vector<int> add(amount_channels);//vector donde se
                            guardaran los valores de las bandas(o canales) de
                            un pixel
                        reparador=0;//valor de cada banda o canal
                        vector<Pixel> punto;//vector de pixeles para construir
                             el bloque a reconstruir

                        for(int r=0;r<img->getHeightBlock();r++){
                            for(int c=0;c<img->getWidthBlock();c++){
                                for(int x=0;x<amount_channels;x++){
                                    if((i>0) & (i<h-1) & (j>0) & (j<w-1)){//
                                        bloques interiores
                                        //cout << "reparando bloques
                                            interiores\n";
                                        if ((r>0)&&(r<img->getWidthBlock()-1)
                                            &&(c>0)&&(c<img->getWidthBlock()-1)
                                            ) {
//cout << "entre al centro del bloque del bloque\n";



f_i_j=(nn(img,i,j,c-1,x)+nn(img,i,j,c+1,x)+(ss(img,i,j,c-1,x))+(ss(img,i,j,c
    +1,x))+(ee(img,i,j,r-1,x))+(ee(img,i,j,r+1,x))+(oo(img,i,j,r-1,x))+(oo(img,
    i,j,r+1,x)));

df_dc= (((nn(img,i,j,c-1,x)-nn(img,i,j,c+1,x))/2)+((ss(img,i,j,c-1,x)-ss(img,i
    ,j,c+1,x))/2));

df_dr= (((nn(img,i,j,c-1,x)-ss(img,i,j,c-1,x))/img->getWidthBlock()+2)+((nn(
    img,i,j,c+1,x)-ss(img,i,j,c+1,x))/img->getWidthBlock()+2));
```

```
d2f_dcdr= (ss(img,i,j,c+1,x)-nn(img,i,j,c+1,x)-ss(img,i,j,c-1,x)+nn(img,i,j,c
    -1,x));
                // reparador=(nn(img,i,j,c,x)*((img->getWidthBlock()+2)-c)  +(
                    ss(img,i,j,c,x)*(c-1))+(ee(img,i,j,r,x)*(r-1))+(oo(img,i,j,
                    r,x)*((img->getWidthBlock()+2)-r)))/((img->getWidthBlock()
                    *2)+2);
reparador=  ((f_i_j)/8)+ ((df_dc)/16)+ ((df_dr)/16)+ ((d2f_dcdr)/64);
                                        }
                                        if ((r==0) && (c==0)) {
                                        //cout << "entre a la esquina del
                                            bloque\n";
                                            reparador= (nn(img,i,j,c,x)+oo(img
                                                ,i,j,r,x))/2;

                                        }
                                        if ((r==0)&&(c==img->getWidthBlock()
                                            -1)) {
                                            reparador= ((nn(img,i,j,c,x)+ee(
                                                img,i,j,r,x))/2);

                                        }
                                        if ((r==img->getWidthBlock()-1)&&(c
                                            ==0)) {
                                            reparador= ((ss(img,i,j,c,x)+oo(
                                                img,i,j,r,x))/2);

                                        }
                                        if ((r==img->getWidthBlock()-1)&&(c==
                                            img->getWidthBlock()-1)) {
                                            reparador= ((ss(img,i,j,c,x)+ee(
                                                img,i,j,r,x))/2);

                                        }
                                         if ((r==0)&&(c<img->getWidthBlock()
                                            -1)&&(c>0)) {

                                            reparador= (nn(img,i,j,c-1,x)+nn(
                                                img,i,j,c,x)+nn(img,i,j,c+1,x))
                                                /3;

                                        }
```

```
if ((r==img->getWidthBlock()-1)&&(c<
    img->getWidthBlock()-1)&&(c>0)) {

     reparador= (ss(img,i,j,c-1,x)+ss(
         img,i,j,c,x)+ss(img,i,j,c+1,x))
         /3;

}
if ((c==0)&&(r<img->getWidthBlock()-1)
    &&(r>0)) {

     reparador= (oo(img,i,j,r-1,x)+oo(
         img,i,j,r,x)+oo(img,i,j,r+1,x))
         /3;

}
if ((c==img->getWidthBlock()-1)&&(r<
    img->getWidthBlock()-1)&&(r>0)) {

     reparador= (ee(img,i,j,r-1,x)+ee(
         img,i,j,r,x)+ee(img,i,j,r+1,x))
         /3;

}


//reparador=(nn(img,i,j,c,x)+(ss(img,i
    ,j,c,x))+(ee(img,i,j,r,x))+(oo(img,
    i,j,r,x)))/4;//i,j coordenadas del
    bloque, c=columna pixel dentro del
    bloque
```

```
}
else{
    if((i>0&&i<h-1) && (j==0 | j==w-1)){//
        primera y ultima columna sin
        conciderar las esquinas
        if(j==0){
        //cout << " reparando primera
            columna\n";
        reparador=(ss(img,i,j,c,x)+nn(img,
            i,j,c,x)+ee(img,i,j,r,x))/3;
        }
        else{
        //cout << " reparando ultima
            columna\n";
        reparador=(ss(img,i,j,c,x)+nn(img,
            i,j,c,x)+oo(img,i,j,r,x))/3;
        }
    }
```

```
                else{
                    if((i==0 | i==h-1) && (j>0&&j<w-1)
                        ){//primera y ultima fila sin
                        conciderar las esquinas
                        if(i==0){
                        //cout << "reparando primera
                            fila\n";
                        reparador=(oo(img,i,j,r,x)+ee(
                            img,i,j,r,x)+ss(img,i,j,c,x
                            ))/3;
                        }
                        else{
                        //cout << "reparando ultima
                            fila\n";
                        reparador=(oo(img,i,j,r,x)+ee(
                            img,i,j,r,x)+nn(img,i,j,c,x
                            ))/3;
                        }
                    }
                    else{
                        if(i==0 & j==0){//esquina
                            superios izquierda
                            //cout << "reparando
                                esquina superios
                                izquierda\n";
                            reparador=((ss(img,i,j,c,x
                                ))+(ee(img,i,j,r,x)))
                                /2;
                        }
                        else{
                            if(i==0&j==w-1){//esquina
                                superior derecha
                            //cout << "reparando
                                esquina superior
                                derecha\n";
                                reparador=((ss(img,i,j
                                    ,c,x))+(oo(img,i,j,
                                    r,x)))/2;
                            }
                            else{
                                if(i==h-1 & j==0){//
                                    esquina inferiro
```

```
                                        izquierda
                                    //cout << "reparando
                                        esquina inferiro
                                        izquierda\n";
                                            reparador=((nn(img
                                                ,i,j,c,x))+(ee(
                                                img,i,j,r,x)))
                                                /2;
                                        }
                                        else{//esquina
                                            inferior derecha
                                    //cout << "reparando
                                        esquina inferior
                                        derecha\n";
                                          reparador=((nn(img,i,
                                            j,c,x))+(oo(img,i,
                                            j,r,x)))/2;
                                        }
                                    }

                                }

                            }

                        }
                        add[x]=floor(reparador);//actualizando el
                            vector con el valor de la banda o canal
                             "x"
    }

        Pixel aux(add,amount_channels); //creando el pixel
            en su respectivo formato
        punto.push_back(aux);//creando el vector de pixel
            que con el cual se construira el bloque
            recuperado
        }
    }//aqui se pasa el vector de pixel al constructor
        DataBlock en que debuelve un bloque reparado
    DataBlock* db = new DataBlock(punto,img->getWidthBlock
        (),img->getHeightBlock(),amount_channels);
```

```
//              vector<Pixel> *e = (vector<Pixel>*)db->getExtras();
//              for(int r=0;r<img->getHeightBlock();r++){
//                                                  for(int c=0;c<img->
    getWidthBlock();c++){
//                                                                  vector<int>
    chnls = e->at((r*img->getWidthBlock())+c).getChannels();
//                                                                  for(int x=0;x<
    amount_channels;x++){
//
                    cout<<"pixel desde bloque creado"<<chnls[x]<<"\n";
//
            }
//                                                  }
//              }

                        matrix[i][j]= *db;//actualizando la matrix de bloques
                            con el nuevo bloque reconstruido
                    }
                        }
                    }
            }while(isLossBlock(img));
            if(export_images){
                string path = header->folder+"/image_restored_bicubica.bmp";
            img->save(path.c_str());
            }
            if(show_data)
                cout << "The Image has been restored by bicubic\n";
        }

private:

        bool isLossBlock(Images* img){
        /**
        * @brief Metodo que valida si existen pixeles perdidos en la imagen
        * @param img Imagen sobre la que se buscan los pixeles perdidos
        * @return true si hay pixeles perdidos en la imagen
        * @return false si no hay pixeles perdidos en la imagen
        */
                DataBlock** matrix = img->getMatrix();
                int w=img->getWidth()/img->getWidthBlock();
                int h=img->getHeight()/img->getHeightBlock();
```

```
        for (int i = 0; i < h; ++i)
        {
                for (int j = 0; j < w; ++j)
                {
                        if(!matrix[i][j].isValid())
                                return true;
                }
        }
        return false;
}
int nn(Images* img, int i, int j,int jj, int x){
/**
* @brief Metodo que devuelve el pixel inferior del bloque superior
    contiguo al pixel perdido
* @param img Imagen que contiene el pixel en la posicion (i,j)
* @param i fila en la que esta ubicada el bloque afectado
* @param j columna en la que esta ubicada el el bloque afectado
* @param jj ubicacion del pixel en el bloque con perdida
* @param x banda de interes
* @return pixel
*/
    //cout << "bloque superior \n";
    DataBlock** matrix=img->getMatrix();
    int pixel;
    if(!matrix[i-1][j].isValid()){
        pixel=127;
    }
    else{
    vector<Pixel> *e = (vector<Pixel>*)matrix[i-1][j].getExtras();
    vector<int> chnls = e->at(((img->getHeightBlock()-1)*img->
        getWidthBlock())+jj).getChannels();
    pixel=chnls[x];
    }
    //cout << "pixel superior: " << pixel << "\n";
    return pixel;
}
int oo(Images* img, int i, int j,int ii, int x){
/**
* @brief Metodo que devuelve el pixel derecho del bloque izquierdo
    contiguo al pixel perdido
* @param img Imagen que contiene el pixel en la posicion (i,j)
* @param i fila en la que esta ubicada el bloque afectado
```

```
* @param j columna en la que est  ubicada el el bloque afectado
* @param ii ubicacion del pixel en el bloque con perdida
* @param x banda de interes
* @return pixel
*/
    //cout << "bloque izquierdo \n";
    DataBlock** matrix=img->getMatrix();
    int pixel;
    if(!matrix[i][j-1].isValid()){
        pixel=127;
    }
    else{
    vector<Pixel> *e = (vector<Pixel>*)matrix[i][j-1].getExtras();
    vector<int> chnls = e->at(((ii)*img->getWidthBlock())+(img->
        getWidthBlock()-1)).getChannels();
    pixel=chnls[x];
    }
    //cout << "pixel izquierdo: " << pixel << "\n";
    return pixel;
}
int ee(Images* img, int i, int j,int ii, int x){
/**
* @brief M todo que devuelve el pixel derecho  del bloque izquierdo
    contiguo al pixel perdido
* @param img Imagen que contiene el pixel en la posici n (i,j)
* @param i fila en la que est  ubicada el bloque afectado
* @param j columna en la que est  ubicada el el bloque afectado
* @param ii ubicacion del pixel en el bloque con perdida
* @param x banda de interes
* @return pixel
*/
    //cout << "bloque derecho \n";
    DataBlock** matrix=img->getMatrix();
    int pixel;
    if(!matrix[i][j+1].isValid()){
       pixel=127;
    }
    else{
    vector<Pixel> *e = (vector<Pixel>*)matrix[i][j+1].getExtras();
    vector<int> chnls = e->at(((ii)*img->getWidthBlock())).getChannels
        ();
    pixel=chnls[x];
```

```
            }
            //cout << "pixel derecho: " << pixel << "\n";
            return pixel;
        }
        int ss(Images* img, int i, int j,int jj, int x){
        /**
        * @brief Metodo que devuelve el pixel superior del bloque inferior
            contiguo al pixel perdido
        * @param img Imagen que contiene el pixel en la posicion (i,j)
        * @param i fila en la que esta ubicada el bloque afectado
        * @param j columna en la que esta ubicada el el bloque afectado
        * @param jj ubicacion del pixel en el bloque con perdida
        * @param x banda de interes
        * @return pixel
        */
            //cout << "bloque inferior\n";
            DataBlock** matrix=img->getMatrix();
            int pixel;
            if(!matrix[i+1][j].isValid()){//si no existe el bloque se devuelve
                0
                pixel=127;
            }
            else{
            vector<Pixel> *e = (vector<Pixel>*)matrix[i+1][j].getExtras();//
                los elementos del bloque se recuperan en un vector
            vector<int> chnls = e->at(jj).getChannels();//se busca el elemento
                de interes
            pixel=chnls[x];//el correspondiente canal del elemento de interes
            }
            //cout << "pixel inferior: " << pixel << "\n";
            return pixel;
        }

};
#endif
```

B.5. RebuilderMSR.h

```
#ifndef CLASS_REBUILDERMSR
```

```
#define CLASS_REBUILDERMSR

#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))

#include "Rebuilder.h"
#include <iostream>
#include <vector>

using namespace std;

class Rebuildermsr: public Rebuilder{

public:

        RebuilderMSR(){};

        ~RebuilderMSR(){};

        virtual void hidden(Images* img, bool show_data, bool export_images,
           HEAD *header){
        /**
        * @brief Metodo que oculta los pixeles dados y los reemplaza por
            el promedio de sus vecinos mas cercanos
        * @param img Imagen sobre la que se le aplicar  el metodo de
            ocultamiento de errores
        *
        */
//        img->clearBlocks(header);
        DataBlock** matrix = img->getMatrix();
        int amount_channels = img->getType();
        int w = img->getWidth()/img->getWidthBlock();
        int h = img->getHeight()/img->getHeightBlock();
        float reparador;

        do{
            for(int i = 0; i < h; i++){
                for(int j = 0; j < w; j=j+1){
                    if(!matrix[i][j].isValid()){
                        vector<int> add(amount_channels);//vector donde se
                            guardaran los valores de las bandas(o canales) de
                            un pixel
```

```
reparador=0;//valor de cada banda o canal
vector<Pixel> punto;//vector de pixeles para construir
     el bloque a reconstruir

for(int r=0;r<img->getHeightBlock();r++){
     for(int c=0;c<img->getWidthBlock();c++){
          for(int x=0;x<amount_channels;x++){
               if((i>0) & (i<h-1) & (j>0) & (j<w-1)){//
                    bloques interiores
                    //cout << "reparando bloques
                         interiores\n";
                    reparador=(nn(img,i,j,c,x)+(ss(img,i,j
                         ,c,x))+(ee(img,i,j,r,x))+(oo(img,i,
                         j,r,x)))/4;

               }
               else{
                    if((i>0&&i<h-1) && (j==0 | j==w-1)){//
                         primera y ultima columna sin
                         conciderar las esquinas
                         //cout << " reparando primera o
                              ultima columna\n";
                         reparador=((nn(img,i,j,c,x)*(img->
                              getHeightBlock()+2-r)+(ss(img,i
                              ,j,c,x))*(2+r))/(img->
                              getHeightBlock()+2));
                    }
                    else{
                         if((i==0 | i==h-1) && (j>0&&j<w-1)
                              ){//primera y ultima fila sin
                              conciderar las esquinas
                         //cout << "reparando primera o
                              ultima fila\n";
                              reparador=((ee(img,i,j,r,x))
                                   *(2+c)+(oo(img,i,j,r,x))*(
                                   img->getWidthBlock()+2-c))
                                   /(img->getWidthBlock()+2);
                         }
                         else{
                              if(i==0 & j==0){//esquina
                                   superios izquierda
```

```
            //cout << "reparando
                esquina superios
                izquierda\n";
            reparador=((ss(img,i,j,c,x
                ))+(ee(img,i,j,r,x)))
                /2;
        }
        else{
            if(i==0&j==w-1){//esquina
                superior derecha
            //cout << "reparando
                esquina superior
                derecha\n";
                 reparador=((ss(img,i,j
                     ,c,x))+(oo(img,i,j,
                     r,x)))/2;
            }
            else{
                 if(i==h-1 & j==0){//
                     esquina inferiro
                     izquierda
            //cout << "reparando
                esquina inferiro
                izquierda\n";
                      reparador=((nn(img
                          ,i,j,c,x))+(ee(
                          img,i,j,r,x)))
                          /2;
                 }
                 else{//esquina
                     inferior derecha
            //cout << "reparando
                esquina inferior
                derecha\n";
                   reparador=((nn(img,i,
                       j,c,x))+(oo(img,i,
                       j,r,x)))/2;
                 }
            }

        }
```

```
                        }

                    }

                }
                add[x]=floor(reparador);//actualizando el
                    vector con el valor de la banda o canal
                    "x"
}

    Pixel aux(add,amount_channels); //creando el pixel
        en su respectivo formato
    punto.push_back(aux);//creando el vector de pixel
        que con el cual se construira el bloque
        recuperado
    }
}
//hasta aqui todo bien
if(img->getHeightBlock()>1&&img->getWidthBlock()>1){
if((i>0) & (i<h-1) & (j>0) & (j<w-1)){
    for(int r=1;r<img->getHeightBlock()-1;r++){//
        columna 0
            vector<int> chnls = punto[(r*img->
                getWidthBlock())].getChannels();
            for(int x=0;x<amount_channels;x++){
                add[x]=(oo(img,i,j,r,x)+chnls[x])/2;
            }
                punto[(r*img->getWidthBlock())].
                    setChannels(add);
        }

    for(int r=1;r<img->getHeightBlock()-1;r++){//
        columna ultima
            vector<int> chnls = punto[(r*img->
                getWidthBlock())+img->getWidthBlock()
                -1].getChannels();
            for(int x=0;x<amount_channels;x++){
                add[x]=(ee(img,i,j,r,x)+chnls[x])/2;
            }
                punto[(r*img->getWidthBlock())+img->
                    getWidthBlock()-1].setChannels(add)
                    ;
```

```
            }
        for(int c=1;c<img->getWidthBlock()-1;c++){//fila 0
                vector<int> chnls = punto[c].getChannels()
                    ;
                for(int x=0;x<amount_channels;x++){
                    add[x]=(nn(img,i,j,c,x)+chnls[x])/2;
                }
                    punto[c].setChannels(add);
            }
        for(int c=1;c<img->getWidthBlock()-1;c++){//fila
            ultima
                vector<int> chnls = punto[(img->
                    getHeightBlock()-1)*img->getWidthBlock
                    ()+c].getChannels();
                for(int x=0;x<amount_channels;x++){
                    add[x]=(ss(img,i,j,c,x)+chnls[x])/2;
                }
                    punto[(img->getHeightBlock()-1)*img->
                        getWidthBlock()+c].setChannels(add)
                        ;
            }
    vector<int> chnls = punto[0].getChannels();//esquina 1
    for(int x=0;x<amount_channels;x++){
                    add[x]=(oo(img,i,j,0,x)+chnls[x]+nn(
                        img,i,j,0,x))/3;
                }
    punto[0].setChannels(add);
     chnls = punto[img->getWidthBlock()-1].getChannels();
        //esquina 2
    for(int x=0;x<amount_channels;x++){
                    add[x]=(ee(img,i,j,img->getWidthBlock
                        ()-1,x)+chnls[x]+nn(img,i,j,img->
                        getWidthBlock()-1,x))/3;
                }
    punto[img->getWidthBlock()-1].setChannels(add);
     chnls = punto[(img->getHeightBlock()-1)*img->
        getWidthBlock()].getChannels();//esquina 3
     for(int x=0;x<amount_channels;x++){
                    add[x]=(oo(img,i,j,img->getHeightBlock
                        ()-1,x)+chnls[x]+ss(img,i,j,img->
                        getWidthBlock()-1,x))/3;
                }
```

```
punto[(img->getHeightBlock()-1)*img->getWidthBlock()].
    setChannels(add);
 chnls = punto[(img->getHeightBlock()-1)*img->
     getWidthBlock()+img->getWidthBlock()-1].
     getChannels();//esquina 4
 for(int x=0;x<amount_channels;x++){
                add[x]=(ee(img,i,j,img->getHeightBlock
                    ()-1,x)+chnls[x]+ss(img,i,j,img->
                    getHeightBlock()-1,x))/3;
            }
punto[(img->getHeightBlock()-1)*img->getWidthBlock()+
    img->getWidthBlock()-1].setChannels(add);
for(int r=1;r<=floor(img->getHeightBlock()/2);r++){//
    segundo cuadrante
    for(int c=1;c<=floor(img->getWidthBlock()/2);c++){
        vector<int> chnls = punto[(r*img->
            getWidthBlock())+c].getChannels();
        for(int x=0;x<amount_channels;x++){
            add[x]=(chnls[x]+punto[((r-1)*img->
                getWidthBlock())+c].getChannels()[x]+
                punto[((r)*img->getWidthBlock())+c-1].
                getChannels()[x])/3;
        }
        punto[(r*img->getWidthBlock())+c].setChannels(
            add);
    }
}
for(int r=1;r<=floor(img->getHeightBlock()/2);r++){//
    primer cuadrante
    for(int c=img->getWidthBlock()-2;c>floor(img->
        getWidthBlock()/2);c--){
        vector<int> chnls = punto[(r*img->
            getWidthBlock())+c].getChannels();
        for(int x=0;x<amount_channels;x++){
            add[x]=(chnls[x]+punto[((r-1)*img->
                getWidthBlock())+c].getChannels()[x]+
                punto[((r)*img->getWidthBlock())+c+1].
                getChannels()[x])/3;
        }
        punto[(r*img->getWidthBlock())+c].setChannels(
            add);
    }
```

```
                }
                for(int r=img->getHeightBlock()-2;r>floor(img->
                    getHeightBlock()/2);r--){//cuarto cuadrante
                    for(int c=img->getWidthBlock()-2;c>floor(img->
                        getWidthBlock()/2);c--){
                        vector<int> chnls = punto[(r*img->
                            getWidthBlock())+c].getChannels();
                        for(int x=0;x<amount_channels;x++){
                            add[x]=(chnls[x]+punto[((r+1)*img->
                                getWidthBlock())+c].getChannels()[x]+
                                punto[((r)*img->getWidthBlock())+c+1].
                                getChannels()[x])/3;
                        }
                        punto[(r*img->getWidthBlock())+c].setChannels(
                            add);
                    }
                }
                for(int r=img->getHeightBlock()-2;r>floor(img->
                    getHeightBlock()/2);r--){//tercer cuadrante
                    for(int c=1;c<=floor(img->getWidthBlock()/2);c++){
                        vector<int> chnls = punto[(r*img->
                            getWidthBlock())+c].getChannels();
                        for(int x=0;x<amount_channels;x++){
                            add[x]=(chnls[x]+punto[((r+1)*img->
                                getWidthBlock())+c].getChannels()[x]+
                                punto[((r)*img->getWidthBlock())+c-1].
                                getChannels()[x])/3;
                        }
                        punto[(r*img->getWidthBlock())+c].setChannels(
                            add);
                    }
                }
                }
            }
                else{
                    //cout<<"no se puede aplicar bilineal ponderado
                        mejorado\n"
                }

                //aqui se pasa el vector de pixel al constructor
                    DataBlock en que debuelve un bloque reparado
```

```
                        DataBlock* db = new DataBlock(punto,img->getWidthBlock
                            (),img->getHeightBlock(),amount_channels);


//              vector<Pixel> *e = (vector<Pixel>*)db->getExtras();
//              for(int r=0;r<img->getHeightBlock();r++){
//                                                  for(int c=0;c<img->
    getWidthBlock();c++){
//                                                                  vector<int>
    chnls = e->at((r*img->getWidthBlock())+c).getChannels();
//                                                                  for(int x=0;x<
    amount_channels;x++){
//
                    cout<<"pixel desde bloque creado"<<chnls[x]<<"\n";
//
            }
//                                                  }
//              }

                        matrix[i][j]= *db;//actualizando la matrix de bloques
                            con el nuevo bloque reconstruido
                    }
                        }
                    }
            }while(isLossBlock(img));
            if(export_images){
                string path = header->folder+"/image_restaured_MSR.bmp";
            img->save(path.c_str());
            }
            if(show_data)
                cout << "The Image has been restored by MSR\n";
        }

private:

        bool isLossBlock(Images* img){
        /**
        * @brief Metodo que valida si existen pixeles perdidos en la imagen
        * @param img Imagen sobre la que se buscan los pixeles perdidos
        * @return true si hay pixeles perdidos en la imagen
        * @return false si no hay pixeles perdidos en la imagen
        */
```

```
        DataBlock** matrix = img->getMatrix();
        int w=img->getWidth()/img->getWidthBlock();
        int h=img->getHeight()/img->getHeightBlock();
        for (int i = 0; i < h; ++i)
        {
                for (int j = 0; j < w; ++j)
                {
                        if(!matrix[i][j].isValid())
                                return true;
                }
        }
        return false;
}
int nn(Images* img, int i, int j,int jj, int x){
/**
* @brief Metodo que devuelve el pixel inferior del bloque superior
    contiguo al pixel perdido
* @param img Imagen que contiene el pixel en la posicion (i,j)
* @param i fila en la que est ubicada el bloque afectado
* @param j columna en la que est ubicada el el bloque afectado
* @param jj ubicacion del pixel en el bloque con perdida
* @param x banda de interes
* @return pixel
*/
  //  cout << "bloque superior \n";
    DataBlock** matrix=img->getMatrix();
    int pixel;
    if(!matrix[i-1][j].isValid()){
        pixel=127;
    }
    else{
    vector<Pixel> *e = (vector<Pixel>*)matrix[i-1][j].getExtras();
    vector<int> chnls = e->at(((img->getHeightBlock()-1)*img->
        getWidthBlock())+jj).getChannels();
    pixel=chnls[x];
    }
//    cout << "pixel superior: " << pixel << "\n";
    return pixel;
}
int oo(Images* img, int i, int j,int ii, int x){
/**
```

```
     * @brief M todo que devuelve el pixel derecho  del bloque izquierdo
        contiguo al pixel perdido
     * @param img Imagen que contiene el pixel en la posici n (i,j)
     * @param i fila en la que est  ubicada el bloque afectado
     * @param j columna en la que est  ubicada el el bloque afectado
     * @param ii ubicacion del pixel en el bloque con perdida
     * @param x banda de interes
     * @return pixel
     */
   //       cout << "bloque izquierdo \n";
          DataBlock** matrix=img->getMatrix();
          int pixel;
          if(!matrix[i][j-1].isValid()){
              pixel=127;
          }
          else{
          vector<Pixel> *e = (vector<Pixel>*)matrix[i][j-1].getExtras();
          vector<int> chnls = e->at(((ii)*img->getWidthBlock())+(img->
              getWidthBlock()-1)).getChannels();
          pixel=chnls[x];
          }
  //        cout << "pixel izquierdo: " << pixel << "\n";
          return pixel;
      }
      int ee(Images* img, int i, int j,int ii, int x){
      /**
      * @brief M todo que devuelve el pixel derecho  del bloque izquierdo
         contiguo al pixel perdido
      * @param img Imagen que contiene el pixel en la posici n (i,j)
      * @param i fila en la que est  ubicada el bloque afectado
      * @param j columna en la que est  ubicada el el bloque afectado
      * @param ii ubicacion del pixel en el bloque con perdida
      * @param x banda de interes
      * @return pixel
      */
          DataBlock** matrix=img->getMatrix();
          int pixel;
          if(!matrix[i][j+1].isValid()){
             pixel=127;
          }
          else{
          vector<Pixel> *e = (vector<Pixel>*)matrix[i][j+1].getExtras();
```

```
            vector<int> chnls = e->at(((ii)*img->getWidthBlock())).getChannels
                ();
            pixel=chnls[x];
            }
  //          cout << "pixel derecho: " << pixel << "\n";
            return pixel;
        }
        int ss(Images* img, int i, int j,int jj, int x){
        /**
        * @brief Metodo que devuelve el pixel superior del bloque inferior
            contiguo al pixel perdido
        * @param img Imagen que contiene el pixel en la posicion (i,j)
        * @param i fila en la que esta ubicada el bloque afectado
        * @param j columna en la que esta ubicada el el bloque afectado
        * @param jj ubicacion del pixel en el bloque con perdida
        * @param x banda de interes
        * @return pixel
        */
//            cout << "bloque inferior\n";
            DataBlock** matrix=img->getMatrix();
            int pixel;
            if(!matrix[i+1][j].isValid()){//si no existe el bloque se devuelve
                0
                pixel=127;
            }
            else{
            vector<Pixel> *e = (vector<Pixel>*)matrix[i+1][j].getExtras();//
                los elementos del bloque se recuperan en un vector
            vector<int> chnls = e->at(jj).getChannels();//se busca el elemento
                de interes
            pixel=chnls[x];//el correspondiente canal del elemento de interes
            }
            //cout << "pixel inferior: " << pixel << "\n";
            return pixel;
        }

};
#endif
```