UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

TESIS DOCTORAL

Mejora de la Calidad en Imagen Estática y Vídeo Basada en Súper-Resolución con Prestaciones de Tiempo Real y Bajo Coste mediante Codificación Híbrida

Autor: D. GUSTAVO IVÁN MARRERO CALLICÓ
Director: DR. D. ANTONIO NÚÑEZ ORDÓÑEZ
Co-Director: DR. D. RAFAEL PESET LLOPIS

Las Palmas de Gran Canaria, Julio de 2003

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
DOCTORADO EN INGENIERÍA DE TELECOMUNICACIÓN

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA
PROGRAMA DE INGENIERÍA ELECTRÓNICA

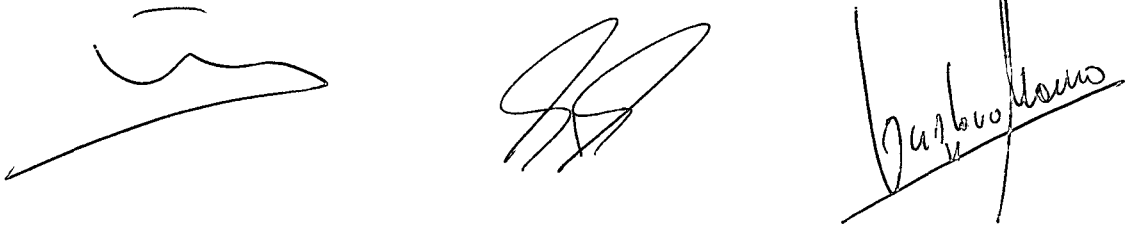
Mejora de la Calidad en Imagen Estática y Vídeo Basada en Súper-Resolución con Prestaciones de Tiempo Real y Bajo Coste mediante Codificación Híbrida

TESIS DOCTORAL PRESENTADA POR:
GUSTAVO IVÁN MARRERO CALLICÓ

DIRIGIDA POR:
DR. D. ANTONIO NÚÑEZ ORDÓÑEZ

CO-DIRIGIDA POR:
DR. D. RAFAEL PESET LLOPIS

El Director, El Co-Director El Doctorando,



Las Palmas de Gran Canaria, Julio de 2003

A mi esposa Emi y a mi hijo Julio.
Ellos han sido la fuente a la que he ido a beber
cuando he sentido que mis fuerzas flaqueaban.

A Emi con todo mi amor

Si pudiera describir la belleza de tus ojos
Y enumerar con números nuevos todas tus gracias,
El futuro diría: - Este poeta miente;
Tales rasgos celestiales nunca fueron de rostros terrenales.

William Shakespeare (1564-1616)
Escritor y poeta inglés.

A mi hijo Julio con todo mi corazón

Es tu risa la espada
más victoriosa.
Vencedor de las flores
y las alondras.
Rival del sol,
porvenir de mis huesos
y de mi amor.
...
Desperté de ser niño.
Nunca despiertes.
Triste llevo la boca.
Ríete siempre.
Siempre en la cuna,
defendiendo la risa
pluma por pluma.

Miguel Hernández (1910-1942)
"Nanas de la cebolla". Poeta español.

A mis directores Antonio y Rafa con mucho cariño,

«Cantemos a los grandes hombres
- que no se envanecen-
por su trabajo seguido;
y su trabajo conseguido,
ancho y hondo proseguido,
¡los años lo acrecen!
...
Allá estaban grandes hombres
para vigilarnos;
nos trataban a baqueta
- podéis creerlo, a baqueta-
maltratados a baqueta.
Decían amarnos.
...
Juntos cantemos los Grandes
son nuestros mayores;
pues nos dieron buen juicio
- nos mostraron qué es juicio -
pues de Dios viene el Juicio;
¡vale más que honores!
...
Y de ellos lo aprendimos,
no sé como se usa,
mostrando, con su trabajo,
que hay que acabar el trabajo
- bien o mal, cualquier trabajo -
sin ninguna excusa.
...
Y de ellos lo aprendimos,
casi sin saberlo.
Pero al transcurrir los años
- sólo al transcurrir los años -
sin ayuda, al pasar los años,
vino el comprenderlo.
...
Cantemos a los grandes hombres
- que no se envanecen-
por su trabajo seguido;
y su trabajo conseguido,
ancho y hondo proseguido,
¡los años lo acrecen!»

Rudyard Kipling (1865-1936)
"Stalky & Cia". Premio Nóbel de Literatura en 1907.

El viento azotaba la bandera del templo, y dos monjes disputaban sobre la cuestión. Uno de ellos decía que la bandera se movía, el otro que se movía el viento. Argumentaban sin cesar. Eno el Patriarca dijo: «No es que el viento se mueva; no es que la bandera se mueva; es que vuestras honorables mentes se mueven».

Doctrina Sutra.

No nos hace falta valor para emprender ciertas cosas porque sean difíciles, sino que son difíciles porque nos falta valor para emprenderlas

Lucio Anneo Séneca (4 a.C.- 65 d.C.)
Filósofo, orador, escritor y político hispano-romano.

Agradecimientos

Cosa dulce es un amigo verdadero; bucea en el fondo de nuestro corazón inquiriendo nuestras necesidades. Nos ahorra el tener que descubrirlas por nosotros mismos.

Jean de la Fontaine (1621-1695).
Escritor francés.

Ningún trabajo se realiza en solitario, y este no es una excepción. Son muchas las personas que han contribuido directa o indirectamente a que esta tesis haya por fin visto la luz, posiblemente más de las que citaré en este escueto apartado. Siempre que se realiza una relación de personas se corre el riesgo de dejar fuera algún nombre. Ya de antemano pido perdón por si cometo la torpeza de dejar de nombrar a alguien que se sienta implicado en este trabajo. Sin embargo, existen algunos nombres claves que bajo ninguna circunstancia podrían dejar de aparecer.

Cuando en una ocasión le preguntaron a ese gran escritor y divulgador científico que fue Isaac Asimov cuál era el mayor científico de todos los tiempos respondió que eso no era difícil de contestar: Sir Isaac Newton, y añadió que lo difícil hubiese sido preguntarle cuál era el segundo mayor científico, en cuyo caso se lo ocurrían múltiples respuestas: Albert Einstein, Marx Plank, Niels Bohr, Pascal, etc. En mi caso sucede algo parecido: si tengo que decir quién es la persona a la que le estoy más agradecido por la elaboración de esta tesis doctoral, no duraría ni un momento en decir que a mi adorada esposa Emi. Ella ha sido la verdadera artífice de este trabajo. Sin su ayuda (y enorme comprensión) no hubiese podido realizar la estancia en los laboratorios de Philips en Eindhoven sobre la que se fundamenta el grueso del trabajo de esta tesis. Creo que si hubiese sido ella la que me hubiese pedido marcharse a más de tres mil kilómetros de distancia a trabajar durante un año, dejándome solo, con un bebe de un año y con un trabajo de gran responsabilidad y estrés, no hubiese dudado ni por un instante en darle mi respuesta: No. Sin embargo, y para mi enorme sorpresa, ella no sólo me dio su beneplácito, sino que encima me animó, me apoyó en todo momento y me ayudó para que la empresa pudiese llegar a buen puerto. Creo que nunca le podré estar lo bastante agradecido.

El segundo lugar en mi lista de agradecimientos lo ocupan por igual dos nombres claves en mi corta vida como investigador: Antonio Núñez Ordóñez y Rafael Peset Llopis. A Rafael Peset tengo que agradecerle las interminables horas encerrados en el laboratorio que me dedico como supervisor en Philips, su enorme paciencia conmigo para conseguir entender el funcionamiento de las miles de líneas de código que forman el codificador *Picasso*, sus valiosísimas ideas y contribuciones en el desarrollo del algoritmo, su confianza en que el trabajo podría realizarse (muchas veces en contra de mis propias opiniones) y sobre todo y antes que nada, su amistad. Sin él, mi estancia en Los Países Bajos no sólo hubiese resultado estéril, sino además un auténtico infierno. A través de su ejemplo pude aprender lo que constituyen las bases de las tareas de investigación: el seguir una metodología estructurada, el diseño adecuado y cuidadoso de los experimentos y el análisis concienzudo de los resultados. A Antonio Núñez tengo tanto que agradecerle que no sé realmente por donde empezar. En primer lugar, deseo agradecerle su confianza en mí. Su confianza en que podría llegar a ser un profesor de esta universidad, su confianza en proponerme para ir a Eindhoven a investigar, y su confianza en que esta tesis vería algún día la luz. En segundo lugar desearía agradecerle su ayuda y apoyo incondicional en todo momento, incluso en momentos en los que yo mismo dudaba que las cosas fuesen a salir bien (es obvio que no soy una persona demasiado optimista). Y en tercer lugar quería agradecerle también su valiosa amistad. Creo que he tenido muchísima suerte por haber tenido el privilegio de poder trabajar con ambos. Contraigo con ambos una deuda de gratitud que difícilmente podré pagar, pero que sin duda lo intentare el resto de mi vida. Muchas gracias a los dos de todo corazón.

Como ya apuntaba al principio, una tesis Doctoral es el resultado de un trabajo en equipo y en ésta, han sido dos equipos humanos los que han colaborado de forma decisiva: el formado por el Instituto Universitario de Microelectrónica Aplicada (IUMA) y el formado por los laboratorios de Philips Research en Eindhoven (Nat.Lab). Dentro del primer equipo, tengo que destacar la ayuda de Pedro Pérez Carballo, que ha actuado como guía y amigo desde que empecé esta increíble aventura. Asimismo, no puedo dejar de agradecer las innumerables ayudas prestadas por ese gran amigo que es Félix Tobajas. De no ser por él y por su incesable apoyo, es seguro que esta tesis no hubiese visto nunca la luz. Sería también injusto no mencionar a José Ramón Sendra y su inestimable apoyo logístico. No estaría donde estoy de no ser por ellos. Dentro de este grupo de buenos amigos creo que tampoco podría agradecer todos los favores que me han hecho Margarita Marrero y Alfonso Medina. Este es otro caso donde mi deuda de gratitud excede con mucho el breve espacio dedicado a este menester. Gracias también a Valentín de Armas, a Roberto Esper-Chaín, a José López, a Roberto Sarmiento y a Juan Antonio Montiel por los muchos ánimos que me han prestado durante la escritura de la tesis y por su pródiga amistad.

Mención aparte merece el agradecimiento a Sebastián López (Chano para los amigos), en quien he encontrado no sólo un valioso e inteligente compañero de investigación sino también un gran amigo y una excelente persona. A el le debo muchas horas de discusión sobre innumerables temas multimedia, un gran numero de horas dedicadas a la minuciosa corrección de este documento y un tan ingente número de favores que no sé si sólo en esta corta vida tendré tiempo de poder devolverle aunque tan sólo sea una pequeña parte.

No puedo dejar de agradecer el valioso apoyo técnico de Enrique Mostesdeoca y Agustín Quintana, gracias a cuyo trabajo las cosas van saliendo en este Instituto. Y por supuesto a Elvira Martín, gran amiga que no sólo me escucha y me da sabios consejos, sino que también me resuelve la inmensa mayoría de mis problemas.

Tampoco puedo olvidar a los amigos y compañeros de fatigas, como Javier García, Javier del Pino y Juan Cerezo. Muchos ánimos de mi parte y muchas gracias por los ánimos recibidos. Gracias también a Antonio Hernández por sus siempre sabios y bien ponderados consejos. Él no lo sabe, pero para mí siempre ha constituido, junto con Antonio Núñez, el ideal de docencia al que me gustaría parecerme.

Y hablando de compañeros de fatigas, tampoco puedo dejar de agradecerles a Tomás Bautista y a Nieves Hernández lo mucho que me aguantaron y ayudaron en Eindhoven. Fue un tiempo duro, pero gracias a ustedes más soportable.

Otro grupo al que quisiera dar las gracias es al formado por Aurelio Vega, Manolo Chaves y Pepe Cabrera. Espero poder seguir contando siempre con su amistad.

Dentro del segundo equipo, liderado por Rafael Peset en Los Países Bajos, también quería agradecer a Ramanathan Saturaman su inagotable paciencia y doctos consejos, así como todo el tiempo que me dedicó a realizar un análisis crítico y detallado de los resultados. A Richard Kleihorse por las indicaciones a seguir en mis primeros pasos sobre ese para mí desconocido mundo de la súper-resolución. Y sobre todo, quería agradecer a Marc op de Beack, padre intelectual del primer algoritmo iterativo de súper-resolución sobre el que se basa esta tesis, su infinita paciencia conmigo, las muchas horas dedicadas a este trabajo, y su gran valor como persona. De no ser por sus claras y amenas explicaciones todavía estaría perdido en ese intrincado mundo del procesamiento de imágenes.

En mi larga estancia en Eindhoven, no sólo me traje el grueso del trabajo de la tesis, también me traje otra cosa mucho más importante: grandes y muy buenos amigos: Leandro, Andréu, Carles, Pili, Piluca, Ana, Noemí, Lourdes, David, Estephanie, Lee Cum, Eeleon, ShewYen, Twan, Ghias,... Ellos son realmente lo mejor que me pasó en Eindhoven.

Como leí hace mucho tiempo, si uno de los objetivos de la tesis doctoral es el de hacer y reforzar lazos de amistad, creo que este objetivo se ha visto sobradamente cumplido, y es sin duda alguna el logro más valioso que tiene para mi la conclusión de este trabajo. Muchas gracias a todos.

Índice General

Cuanto más grande es el caos, más cerca está la solución.

Proverbio Chino

Índice

Índice de figuras

A los doce años dibujaba como Rafael, pero necesite toda una vida para dibujar como un niño.

Pablo Ruiz Picasso (1881-1973)
Pintor y escultor español.

Índice de Tablas

Si uno no puede explicar lo que ha estado haciendo, su trabajo carecerá de valor.

Erwin Schrödinger (1887-1961)
Físico austriaco Premio Nóbel en 1933

Abreviaturas y acrónimos

Un matemático que no es también un poco poeta no será jamás un matemático completo.

Karl Weierstrass (1815-1897)
Matemático alemán.

2D-DCT : *2 Dimensional Discrete Cosine Transform*

2D-IDCT : *2 Dimensional Inverse Discrete Cosine Transform*

3-DRS : *3 – Dimensional Recursive Search*

ALU : *Arithmetic Logic Unit*

AMBA : *Advanced Microcontroller Bus Architecture*

APM : *Advanced Prediction Modem*

ARM : *Advanced RISC Machines*

ASIC : *Application Specific Integrated Circuits*

ASIP : *Application Specific Integrated Processor*

ASISP : *Application Specific Instruction-Set Processor*

ASR : Algoritmo de Súper-Resolución

ASSP : *Application Specific Signal Processor*

AVGA : *Adapted Video Gate Array*

BCU : *Bus Control Unit*

bpf : *bits per frame*

bps : *bits per second*

CAM : *Content Addressable Memory*

CAT : *Computer Aided Tommography*

CCD : *Charge-Coupled Device*

C-HEAP : CPU-*controlled Heterogeneous Embedded Architectures for signal Processing*

CIF : *Common Intermediate Format*

CISC : *Complex Instruction Set Computer*

CMOS : *Complementary Metal Oxide Semiconductor*

CPU : *Central Processing Unit*
DCT : *Discrete Cosine Transform*
DEC : *Decoder unit*
DISP : *Displacement unit*
DSP : *Digital Signal Processor*
EM : *Expectation Maximization*
ENC : *Encoder Unit*
FIR : *Finite Impulse Response*
FPGA : *Field Programmable Gate Array*
fps : *frames per second*
HAVGA : *Half Adapted Video Gate Array*
HDTV : *High Definition Television*
HMRF : *Huber-Markov Random Field*
HR : *High Resolution*
IBP : *Iterative Back-Projection*
IC : *Integrated Circuit*
IC : *Integrated Circuit*
IDCT : *Inverse Discrete Cosine Transform*
iid : *independent and identically distributed*
IIR : *Infinite Impulse Response*
IMC : *Inverse Motion Compensator unit*
IP : *Intellectual Property*
JPEG : *Joint Pictures Expert Group*
Kbytes : *Kilo bytes (1024 bytes)*
LR : *Low Resolution*
LSI : *Linear Space Invariant*
MAP : *Maximum A-posteriori Probability*
MB : Macro-bloque
Mbytes : *Mega bytes (1024·1024 bytes)*
MC : *Motion Compensator unit*
MCM : *Multi Chip Module*
ME : *Motion Estimator unit*

MIMO : *Multi Input Multi Output*
MLE : *Maximum Likelihood Estimator*
MMSE : *Minimum Mean Square Error*
MP@ML : *Main Profile and Main Level*
MPEG : *Motion Pictures Expert Group*
MRF : *Markov Random Field*
MRI : *Magnetic Resonance Images*
NSP : *Native Signal Processor*
PCB : *Printed Circuit Board*
OTF : *Optical Transfer Function*
POCS : *Projection Onto Convex Sets*
PSNR : *Peak Signal to Noise Ratio*
Q : *Quantifier unit*
QAVGA : *Quarter Adapted Video Gate Array*
QCIF : *Quarter CIF*
RAM : *Random Access Memory*
RASSP : *Rapid Prototyping of Application Specific Signal Processor*
RISC : *Reduced Instruction Set Computer*
RLE : *Run Length Encoder*
ROI : *Region of Interest*
ROM : *Read Only Memory*
RTOS : *Real Time Operating System*
SAD : *Sum of Absolute Differences*
SBC : *Single Board Computer*
SCC : *Spectral Correlation Coefficient*
SNR : *Signal to Noise Ratio*
SOC : *System On Chip*
SP@ML : *Single Profile and Main Level*
SPARC : *Scalable Processor Architecture*
SR : Súper-Resolución
SRA : *Super Resolution Algorithm*
UMV : *Unrestricted Motion Vectors*

VDSP : *Video Digital Signal Processor*

VGA : *Video Gate Array*

VHDL : *VHSIC Hardware Description Language*

VHR : *Very High Resolution*

VHSIC : *Very High Speed Integrated Circuits*

VIS : *Visual Instruction Set*

VLE : *Variable Length Encoder*

VLIW : *Very Long Instruction Word*

VLSI : *Very Large Scale Integration*

VOD : *Video On Demand*

VP : *Vision Processor*

ZZ : *Zig-Zag encoder unit*

TAC : Tomografía Asistida por Computador

Capítulo 1

No son, desde luego, los frutos de la investigación científica los que elevan al hombre y enriquecen su personalidad, sino el deseo de comprender, el trabajo intelectual, creador o receptivo.

Albert Einstein (1879-1955).
Físico alemán. Premio Nóbel de física en 1921.

En el curso de esos experimentos, ¿cuántos bellos sistemas construimos que pronto nos vemos obligados a destruir?

Benjamin Franklin (1706-1790).
Filósofo, político y científico estadounidense.

Introducción

1.1 Introducción

Es patente la importancia que están cobrando hoy día los medios audiovisuales. Desde que en 1885 se realizara la primera proyección pública de imágenes en movimiento previamente grabadas, la adquisición, almacenamiento, transmisión y reproducción de imágenes en movimiento y en general la industria cinematográfica y audio visual se ha convertido en uno de los sectores económicos más importantes de la actualidad. Los campos de aplicación de la tecnología audiovisual se adivinan inagotables: videotelefonía, videoconferencia, almacenamiento de vídeo digital, televisión digital, vídeo bajo demanda (*Video On Demand*, VOD), Televisión de alta definición (*High Definition Televisión,* HDTV*)*, sistemas multimedia, y muchas otras aplicaciones entre las cuales incluimos los sistemas de mejora de calidad de imágenes y vídeo, área en la que se enmarca esta tesis.

La súper-resolución es el proceso de obtener imágenes con una resolución mayor que la del sensor utilizado para capturar dichas imágenes. Debido a que la mayoría de las imágenes contiene bordes abruptos, no podemos considerar que estén estrictamente limitadas en banda, por lo que el proceso de muestreo acarreará indefectiblemente cierta cantidad de *aliasing* (FIGURA 1.1) que se manifestará en general como distorsiones en el dominio espacial,

concretamente como la pérdida de muchos de los detalles de la imagen. La súper-resolución conlleva una conversión desde una rejilla de muestreo de menor densidad a otra de mayor densidad [Tek95].

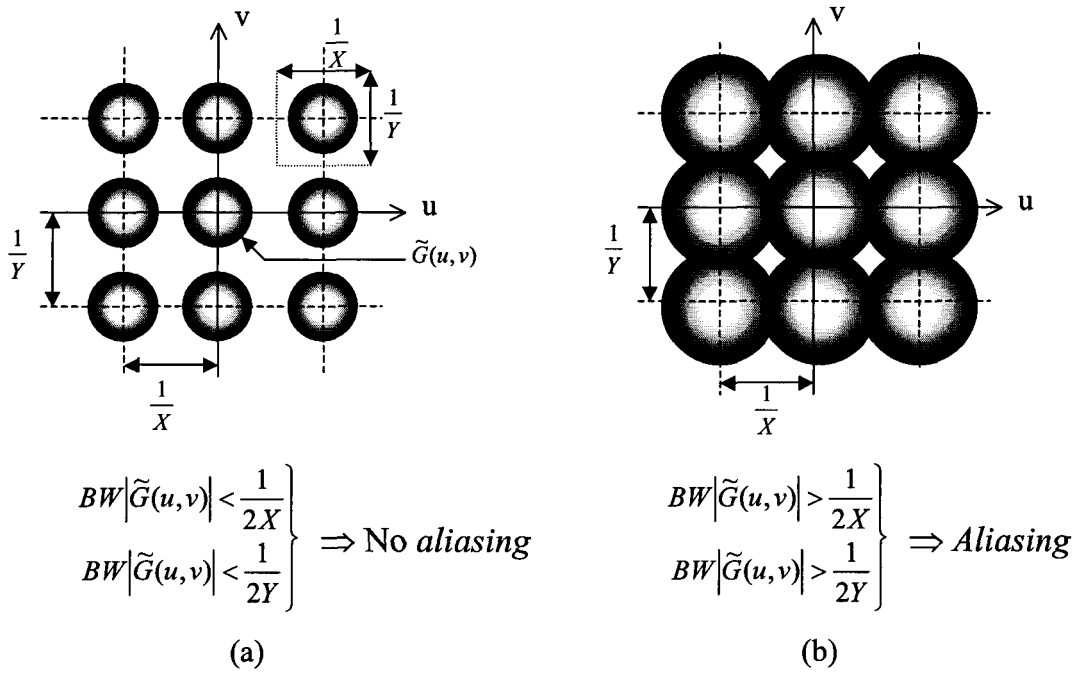


FIGURA 1.1. Espectro de una imagen sin *aliasing* (a) y con *aliasing* (b) junto con las condiciones que han de cumplir las frecuencias de muestreo horizontal y vertical con respecto al ancho de banda de la imagen en cada caso.

Si deseamos aumentar la resolución de una imagen, la forma más inmediata de hacerlo consistiría en usar sensores de mayor resolución, es decir, con mayor cantidad de fotosensores. El problema es que para aumentar la densidad (número de sensores por μm^2) nos vemos forzados a disminuir el tamaño del fotosensor y en consecuencia el área activa de píxel, que es donde se realiza la integración lumínica. Al llegar menos cantidad de luz al fotosensor, éste verá decrementada su sensibilidad y además será mucho más sensible al ruido generado por las fluctuaciones aleatorias de los portadores de carga en sus desplazamientos (*shot noise*). Se estima que el tamaño mínimo de los fotodetectores es de aproximadamente $50\mu m^2$ [KIA93], límite que ya ha sido alcanzado por la tecnología CCD (*Charge-Coupled Device*). Una solución a este problema es la de realizar el aumento de resolución ideando algoritmos destinados a ese propósito como es el caso de los algoritmos de súper-resolución. Además, el uso de este tipo de estrategias permite obtener imágenes con una resolución equivalente a la de un sensor de alta calidad, y que sin embargo han sido capturadas con un sensor de baja calidad y por lo tanto de coste muy inferior.

El problema que tradicionalmente han planteado estos algoritmos es que su elevada complejidad acarrea una elevada carga computacional. Por lo tanto su implementación en tiempo real solo es posible usando plataformas de alta velocidad masivamente paralelas. En este caso el ahorro experimentado en el sensor desaparecería al tener que recurrir a este tipo de subsistema de elevado coste.

Por otra parte, el constante aumento de la densidad de transistores que ofrece hoy la electrónica integrada en un circuito monolítico ha abierto la investigación de nuevos paradigmas de diseño que permiten hacer un uso eficiente de esta mayor densidad. Estos paradigmas incluyen el diseño de grandes circuitos de aplicación específica sintetizados con modernas herramientas de diseño; el codiseño, que parte de la descomposición del problema en tareas que se ejecutan por circuitería específica y en tareas que se ejecutan por programas sobre procesadores sintetizados *had hoc*; o el diseño basado sobre programas ejecutados por procesadores de juego de instrucciones específicos. Simplificamos estos grandes enfoques con los términos ASIC (*Application Specific Integrated Circuits*), codiseño, SOC (*System on Chip*) y ASIP (*Application Specific Integrated Processor*) o ASISP (*Application Specific Instruction-Set Processor*).

Con todos estos enfoques pueden realizarse grandes sistemas empotrados o embebidos [Ern98], [Ern97], [GVNG94], [LSVH96] y [GZ97], que con tecnología convencional se realizaban hasta el presente mediante una integración de componentes sobre placas PCB o en módulos MCM. Sin embargo, el enfoque que supone un traslado más directo de la integración PCB a la integración sobre un chip es el que se ha denominado SOC [Fos99].

La tecnología SOC sólo puede rentabilizarse con producción del circuito en volumen para lo que debe orientarse a grandes mercados de consumo. En otro caso debe organizarse de tal modo que su arquitectura pueda dar soporte a varias aplicaciones. Los módulos componentes de la arquitectura deben ser reutilizables en otros productos y aplicaciones, bien como modelos sintetizables IP (*Intellectual Property*) o bien mediante cierta capacidad de reconfiguración. Esta estrategia pretende crear plataformas arquitecturales multipropósito y los sistemas integrados así concebidos se denominan diseños basados en plataformas.

Esta tesis persigue implementar algoritmos de súper-resolución sin tener que recurrir a *hardware* específico, usando (reutilizando) los recursos ya existentes en una plataforma SOC de codificador de vídeo híbrido. Este codificador ha sido desarrollado por Philips Research y recibe el nombre de *Picasso* [PKL+99]. Desde este punto de vista la súper-resolución es un valor añadido a la plataforma de codificación de vídeo, pudiendo usarse para aumentar la

resolución de las imágenes grabadas, para mejorar la calidad de las imágenes decodificadas o bien como una forma de realizar un zoom digital que ofrece calidades de imagen superiores a las de la mera interpolación.

1.2 Objeto de la tesis

El objetivo de esta tesis es el análisis, desarrollo e implementación de diferentes tipos de algoritmos de súper-resolución, tratando de alcanzar prestaciones de tiempo real y bajo coste. Para ello se ha tenido que:

- Romper el carácter iterativo de los algoritmos existentes en la literatura, ya que es imposible que con los recursos disponibles y con la alta carga computacional que conlleva cada paso de iteración se puedan alcanzar los objetivos trazados de ejecución en tiempo real.
- Eliminar las restricciones que tradicionalmente se imponen a las imágenes de entrada, tales como movimientos suaves y pequeños en comparación al tamaño de la imagen, desplazamientos perfectamente conocidos, ausencia de ruido y/o de difuminado (*blur*), etc. Los algoritmos deben tener la robustez necesaria para enfrentarse a cualquier tipo de imagen de entrada, como ocurre cuando se toman secuencias de imágenes reales.
- Restringirnos al uso de los recursos existentes en la plataforma híbrida de codificación de vídeo *Picasso*. Si los recursos existentes mostrasen ser insuficientes o inadecuados el objetivo es estudiar la posibilidad de añadir nuevos coprocesadores o modificar los ya existentes, de forma siempre compatible con el resto de aplicaciones soportadas por *Picasso*.

Estos objetivos generales pueden ser desglosados en objetivos más concretos de la siguiente forma:

1. Análisis y adaptación de los algoritmos iterativos de súper-resolución a los recursos proporcionados por el codificador de vídeo. Una vez adaptados a la plataforma *Picasso*, todas las modificaciones y nuevas versiones generadas deben estar soportadas por la misma plataforma.
2. Creación de un nuevo algoritmo no iterativo de súper-resolución. La ruptura de la iteratividad es el primer paso importante hacia el objetivo final de esta tesis, y tal vez sea su principal aportación.

3. Adaptación de los algoritmos no iterativos de súper-resolución. Hasta este punto, los algoritmos siempre generan una sola imagen de súper-resolución por cada N imágenes de entrada, lo que supe que una secuencia de entrada de tamaño M fotogramas (*frames*) es diezmada a un tamaño de salida de M/N fotogramas. Es necesario afrontar importantes cambios en el algoritmo para pasar de un esquema de súper-resolución estática (generación de una sola imagen) a un esquema de súper-resolución dinámica (generación de secuencias de imágenes y vídeo).
4. Independencia del algoritmo con respecto a la imagen de entrada. Dado que normalmente es difícil predecir el tipo de imágenes que van a ser capturadas por el sistema de adquisición, y en consecuencia tratadas para incrementar su calidad, el algoritmo debe ser lo suficientemente robusto como para adaptarse dinámicamente a las condiciones presentes en las imágenes de entrada, procurando aumentar su resolución a la salida siempre que sea posible.
5. Finalmente el algoritmo debe ser modificado para evitar usar más memoria de la que puede disponerse internamente en el chip. De esta forma tratamos de evitar la inclusión de memoria externa, con los consiguientes aumentos de los retardos en los accesos a los datos, de la potencia disipada y del coste del circuito final.

1.3 Resumen de resultados

Esta Tesis Doctoral se ha centrado en la mejora de la calidad de imágenes, aprovechando la información contenida en otras imágenes relacionadas entre si, como puede ser el caso de una secuencia de vídeo o la toma sucesiva de varias fotografías. De esta forma se consigue un aumento significativo de la calidad de la imagen resultante por encima de la resolución del sensor con que dichas imágenes fueron muestreadas. El principio matemático que sustenta este tipo de algoritmos es una generalización del teorema de muestro de *Nyquist*. Esta generalización establece que es posible reconstruir una señal a partir de varias series de muestras en presencia de *aliasing* siempre y cuando se pueda asegurar que los periodos de muestreo sean diferentes para cada serie de muestras. En la literatura científica este tipo de técnicas son conocidas como técnicas de súper-resolución.

Cuando se trata de secuencias de imágenes reales, no es posible asegurar que los desplazamientos existentes entre las imágenes de baja resolución ofrezcan un conjunto suficiente de muestras que permitan la perfecta reconstrucción de la imagen de alta resolución. Por este motivo se ha dotado al algoritmo final desarrollado con la posibilidad de realizar una interpolación de los datos ausentes cuando la falta de nuevos datos impida

aumentar la calidad de la imagen resultante. De esta forma el algoritmo desarrollado ve incrementada notablemente su robustez, estableciendo como límite inferior a la calidad obtenida la calidad de interpolación, que es por otra parte la calidad que ofrecen normalmente la mayoría de los sistemas comerciales usados para aumentar el tamaño de las imágenes.

Aunque existen actualmente diferentes algoritmos capaces de obtener mejoras de súper-resolución, todos ellos presentan una o más de las siguientes desventajas:

- Son algoritmos normalmente iterativos, incapaces de trabajar en tiempo real con la tecnología disponible actualmente e incluso dentro de algunos años.
- Los altos requerimientos de memoria encarecen enormemente su coste, y en ocasiones impiden una implementación viable.
- Con la intención de simplificar, muchas veces separan el factor clave de la estimación del movimiento (*registration*) del resto del algoritmo (*restauration*), asumiendo normalmente un perfecto conocimiento del movimiento antes de iniciar el proceso. Esto no refleja una solución completa ni realista al problema planteado, aunque sin duda aporte otro tipo de información de gran interés.
- Por motivos análogos de simplicidad, imponen varias restricciones al tipo de imágenes a tratar, restringiendo su aplicación sólo a este limitado subconjunto. Nuevamente, esto no supone una solución realista al amplio tipo de imágenes que podemos adquirir y tratar en el mundo real.

Tras realizar un estudio de diferentes algoritmos de súper-resolución (capítulo 2) se ha optado por implementar el algoritmo esbozado en [BK99] por investigadores de Philips Research, adaptándolo a la plataforma para la codificación híbrida de vídeo *Picasso*. Los algoritmos propuestos en [BK99] son de tipo iterativo, y las versiones v1.0, v1.1, v1.2 y v1.3 de los algoritmos de súper-resolución desarrollados en esta tesis están fuertemente basados en ellos, aunque adaptados para su ejecución sobre *Picasso*. Las versiones mencionadas pertenecen a la categoría de algoritmos iterativos para súper-resolución estática o para imagen fija, ya que el resultado es una sola imagen de salida. Las calidades medias de luminancia obtenidas, medidas como la relación señal-ruido de pico, para las secuencias de prueba usadas fueron de 23.19 dB para la versión v1.2 y de 28.24 dB para la versión v1.3, que son las versiones iterativas para súper-resolución estática más representativas.

Como los algoritmos iterativos no son capaces de ofrecer prestaciones de funcionamiento en tiempo real se ha tenido que idear un nuevo algoritmo no iterativo basado en interpolar las imágenes de baja resolución a una rejilla de mayor densidad, pero dejando huecos los píxeles no cubiertos por la imagen de baja resolución. Estos huecos son rellenados

por las sucesivas imágenes entrantes compensadas en movimiento y combinados usando el concepto de "peso contributivo" creado para este fin. Además, la combinación de imágenes ponderadas usando este concepto de contribuciones elimina de por sí los problemas aparecidos en los bordes de las imágenes. Este primer conjunto de algoritmos son válidos para lo que denominamos súper-resolución estática, es decir, la generación de una sola imagen de alta resolución mediante la combinación de varias imágenes de baja resolución. Las versiones v2.0 y v2.1 pertenecen a la categoría de algoritmos no iterativos para súper-resolución estática, y las calidades medias obtenidas para ambas versiones fueron de 30.4701 dB y 34.6331 dB respectivamente.

El siguiente paso ha consistido en abordar el problema de la súper-resolución dinámica, para lo cual se ha modificado la versión v2.1 adaptándola a un flujo continuo de imágenes de entrada para obtener un flujo de salida de igual tamaño al de entrada. Esto ha dado lugar a la versión v3. Sin embargo estas versiones tienen el problema de requerir gran cantidad de memoria, lo que se ha solucionado mediante la realimentación de los datos de súper-resolución obtenidos para la imagen anterior. De esta forma conseguimos disminuir los requerimientos de memoria en un factor 50 veces inferior a los de las versiones v2.0 y v2.1, y 35 veces inferior a los de las versiones v1.2 y v1.3. La reutilización de los datos de la imagen anterior supone una dilución de los valores de los píxeles, aunque experimentalmente la pérdida de calidad con respecto a las versiones anteriores fue inferior a 2 dB. La calidad media de la luminancia, medida como relación señal-ruido de pico fue de 28.6415 dB.

Todos los algoritmos presentados se implementan sobre la plataforma integrada *Picasso*, habiéndose tenido que modificar algunos de sus componentes, pero reutilizando su inmensa mayoría. De esta forma se ha añadido la capacidad de realizar mejoras de súper-resolución a la ya existente codificación de vídeo con un coste prácticamente nulo.

La implementación final ha supuesto un caso típico de codiseño, donde se han llevado a *hardware* aquellas tareas con un procesamiento intensivo de datos, y que por lo tanto comprometen de forma importante su funcionamiento en tiempo real, y se han llevado a *software* las tareas más vinculadas a la toma de decisiones y al control e intercambio de datos entre los coprocesadores *hardware*.

Se ha demostrado cualitativa y cuantitativamente la mejora de la calidad en imagen estática y vídeo mediante técnicas de súper-resolución con prestaciones de tiempo real y bajo coste, usando un codificador híbrido de vídeo.

1.4 Organización de la memoria

Esta memoria está organizada en seis capítulos que a su vez se pueden agrupar en tres bloques temáticos: un primer bloque introductorio compuesto por los capítulos 1, 2 y 3, un segundo bloque que recoge las descripciones de los algoritmos de súper-resolución y sus resultados, en los capítulos 4 y 5, y un tercer y último bloque de conclusiones y líneas futuras de trabajo, en el capítulo 6. A continuación se detalla en mayor profundidad el contenido de cada capítulo.

Capítulo 1. En este primer capítulo se da una visión general del contenido de la tesis, de los objetivos que se persiguen, así como de la motivación que la sustenta. Se presenta un primer resumen de resultados que recoge de forma global los hitos más importantes de la tesis.

Capítulo 2. Este capítulo presenta el estado del arte en el objeto científico de la tesis, centrado en las innovaciones en algoritmos de súper-resolución, en arquitecturas de codificación de vídeo basados en plataforma SOC con organización de procesador y coprocesadores heterogéneos y en los métodos de transformaciones algorítmicas y arquitecturales para el mutuo mapeado de funciones. También se realiza una clasificación general en algoritmos de súper-resolución estáticos y dinámicos, estando ambos tipos abarcados en esta tesis.

Capítulo 3. El último capítulo introductorio realiza una descripción de la arquitectura para codificación híbrida de vídeo *Picasso*, desarrollado por la sección ESAS (*Embedded System Architectures over Silicon*) perteneciente al grupo IST (*Information and Software Technologies*) de los laboratorios de investigación de Philips en Eindhoven (*Nat.Lab*) [PKL+99]. Esta arquitectura está a su vez basada en la arquitectura C-HEAP (CPU-*controlled Heterogeneous Embedded Architectures for signal Processing*) para la descripción de arquitecturas empotradas heterogéneas [Lip97]. Los algoritmos de súper-resolución están implementados usando los recursos disponibles en la plataforma descrita.

Capítulo 4. Se expone la metodología seguida en la elaboración de los experimentos conducentes a determinar las métricas de calidad de las imágenes obtenidas. Seguidamente se presentan las primeras versiones de algoritmos iterativos para súper-resolución estática, y se muestran los resultados preliminares junto con la explicación de porqué en determinados casos se produce una disminución importante de la calidad de imagen obtenida. Finalmente se

exponen medidas cualitativas y cuantitativas de calidad de imagen que demuestran las mejoras de resolución experimentadas por las imágenes resultantes.

Capítulo 5. Con el objetivo de poder alcanzar prestaciones de tiempo real y bajo coste se idea un nuevo algoritmo no iterativo para súper-resolución estática. En este sentido se muestran las modificaciones que conducen a una nueva versión no iterativa de súper-resolución. Haciendo uso de los conceptos de pesos contributivos y realimentación de la imagen anterior, se crea una versión no iterativa del algoritmo para súper-resolución dinámica lográndose, gracias a la contención de la memoria a valores que permiten su implementación *on-chip*, prestaciones de tiempo real y bajo coste. Junto con la descripción del algoritmo y las estimaciones de uso de memoria se muestran también las métricas de calidad de imagen obtenidas para cada caso.

Capítulo 6. El capítulo final recoge las conclusiones alcanzadas en el desarrollo de esta tesis y plantea diversas líneas de investigación basadas en este trabajo.

Capítulo 2

Somos una civilización científica. Eso significa una civilización en la que el saber y su integridad son factores cruciales. Ciencia no es más que una palabra latina que significa conocimiento... Nuestro destino es el conocimiento.

Jacob Bronowski (1908-1974)
Matemático e historiador inglés.

¿Qué es el tiempo? Un misterio sin realidad propia y omnipotente. Es una condición del mundo de los fenómenos, un movimiento mezclado y unido a la existencia de los cuerpos en el espacio y a su movimiento. Pero ¿habría tiempo si no hubiese movimiento? ¿Habría movimiento si no hubiese tiempo? ¡Es inútil preguntar! ¿Es el tiempo función del espacio? ¿O es lo contrario? ¿Son ambos una misma cosa? ¡Es inútil continuar preguntando!

Thomas Mann (1875-1955)
"La Montaña Mágica", 1924.
Escritor alemán. Premio Nóbel de Literatura de 1929.

Antecedentes

2.1 Introducción

La posibilidad de reconstruir una imagen de "Súper-resolución" a partir de un grupo de imágenes fue propuesta inicialmente por Huang y Tsay en 1984 [HT84], aunque los teoremas de muestreo generales formulados con anterioridad por Yen en 1956 [Yen56] y Papoulis en 1977 [Pap77] sugerían exactamente el mismo concepto (desde un punto de vista teórico). Desde la propuesta de Huang y Tsay hasta la actualidad, varios grupos de investigación han desarrollado diferentes algoritmos para esta tarea de reconstrucción, obtenidos a partir de diferentes estrategias o enfoques del problema.

La teoría clásica de restauración de una imagen a partir de imágenes borrosas y con ruido ha atraído la atención de muchos investigadores durante las últimas tres décadas. En la literatura científica se han propuesto muchos algoritmos para este problema clásico y los problemas con él relacionados, contribuyendo a la construcción de una teoría unificada que

engloba muchos de los métodos de restauración existentes [LB91]. En la teoría de restauración de una imagen existen principalmente tres aproximaciones distintas que son ampliamente usadas para obtener algoritmos de restauración prácticos: estimador de máxima similitud (MLE, *Maximum Likelihood Estimator*) [GW87], [Jai89], [Pra91] y [LB91], estimador de probabilidad máxima a posteriori (MAP, *Maximum A-posteriori Probability*) [GW87], [Jai89], [Pra91] y [LB91] y la proyección sobre conjuntos convexos (POCS, *Projection Onto Convex Sets*) [You78].

El gran avance experimentado por la tecnología de computadores en los últimos años ha dado lugar a un renovado y creciente interés en la teoría de restauración de imágenes. Los principales enfoques se basan en tratamientos no tradicionales del problema de restauración clásico, se orientan a nuevos problemas de restauración de segunda generación, y utilizan algoritmos que son más complejos y más intensivos computacionalmente. Estos nuevos problemas de segunda generación pueden clasificarse en problemas de restauración de una imagen [Hua83], [HK84], [HT84], [Sor93], [Zer92] y [Kat90] , restauración de una secuencia de imágenes [KDE+89], [KKE+91], [KL91] y [PTS93], y reconstrucción de imagen mejorada con súper-resolución [PST94], [EF95], [SS95], [PST95], [SS96], [EF97] y [PRK98]. Esta tesis se encuadra dentro del último enfoque mencionado, la reconstrucción de imagen estática y de secuencias de imagen con mejora por súper-resolución.

2.2 Reconstrucción de imagen mediante súper-resolución

El problema de reconstrucción usando súper-resolución se podría definir como el objetivo de reconstruir una imagen o toma de una escena, con mayor calidad o resolución a partir de un conjunto finito de imágenes de inferior resolución tomadas de la misma escena [Ela96] y [EF97], tal y como se muestra en la FIGURA 2.1. Este conjunto de imágenes de baja resolución deben haber sido adquiridas a partir de diferentes condiciones de captura de la imagen, de diferentes posiciones espaciales y/o de diferentes cámaras. Este problema de reconstrucción es un aspecto del problema general de fusión de sensores.

Entre las múltiples aplicaciones de los métodos de reconstrucción usando súper-resolución podemos destacar las siguientes:

(i) Adquisición remota [CKK+84], [Ghi84], [KIA+93] y [KW93], donde se obtienen varias imágenes de la misma escena y se busca una imagen de mayor resolución.

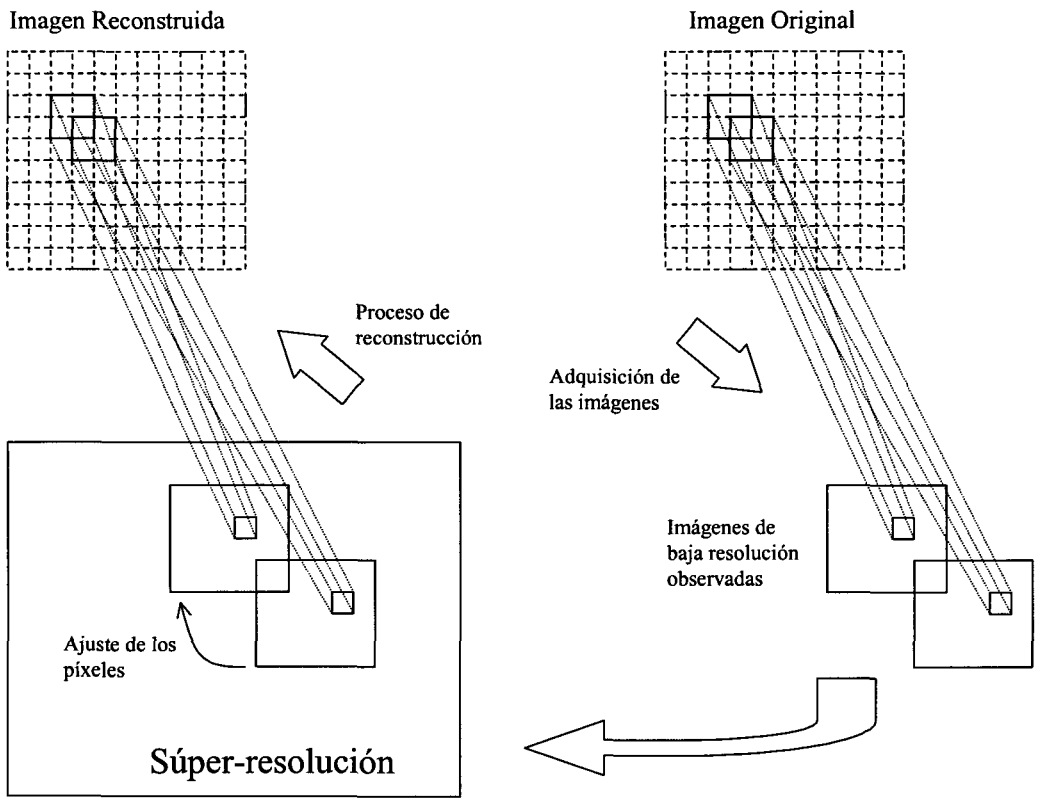


FIGURA 2.1. Modelo del proceso de reconstrucción mediante súper-resolución.

(ii) Mejora de fotogramas de vídeo [BBZ96], [HKK97a], [HKK97b], [AD97] y [Egg00]. Debido a que normalmente los fotogramas de vídeo son de baja calidad, no suelen ser adecuados para realizar directamente una copia impresa tipo fotografía digital. Se puede mejorar la calidad de la imagen usando varias imágenes consecutivas combinadas entre ellas por una algoritmo de súper-resolución (SRA, *Super Resolution Algorithm*).

(iii) Toma de imágenes médicas [GLM+94]. Muchos equipos médicos como la Tomografía asistida por computador (CAT, *Computer Aided Tommography*), las imágenes de resonancia magnética (MRI, *Magnetic Resonance Images*) o las imágenes de ecografía o ultra-sonidos, permiten la obtención de varias imágenes, que podrían ser usadas para combinarlas y generar nuevas imágenes de mayor resolución.

(iv) Cámaras inteligentes (*smart cameras*), de forma que se aprovechen los movimientos naturales de la cámara y los de la escena, para disponer de varias tomas que puedan componerse con súper-resolución.

(v) Mejora de imágenes de vídeo comprimido [KG98], donde se procura recuperar zonas de alta frecuencia de la imagen perdidas en el momento de la compresión.

(vi) Mejora en imágenes de radar [Can98] y [CP98]. Lo que permite observar con mayor claridad detalles en ocasiones críticos relacionados con seguridad aérea o para aplicaciones de observaciones científicas.

(vii) Mejora de calidad de imágenes obtenidas desde el espacio exterior. Un ejemplo de ello son los resultados expuestos en [CKK+94] de imágenes tomadas desde el satélite Viking o el aumento de resolución de las imágenes transmitidas desde el *Mars Pathfinder* (FIGURA 2.2).

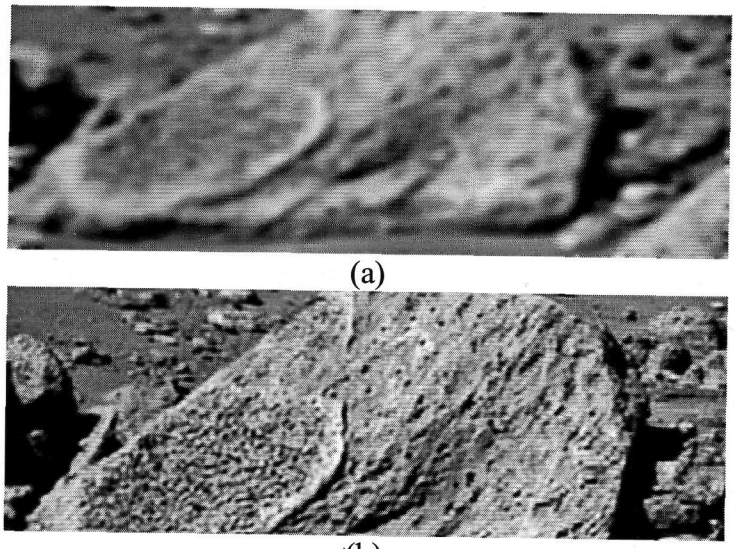


FIGURA 2.2. Imágenes de baja resolución transmitidas desde el *Mars Pathfinder* (a) y resultados obtenidos por la NASA después de aplicar mejoras de súper-resolución.

Al problema de reconstrucción de una determinada imagen a partir de un conjunto de imágenes de menor calidad con cierto movimiento relativo entre ellas se le conoce como problema de súper-resolución estático. En contraposición, tenemos el problema de súper-resolución dinámico, donde se trata de obtener una secuencia de mayor calidad a partir de otra secuencia de imágenes de baja calidad, procurando que ambas secuencias tengan la misma longitud. Estos dos problemas también pueden ser denominados como problema de súper-resolución para imágenes estáticas y problema de súper-resolución para vídeo, respectivamente [Cha01].

2.2.1 Reconstrucción de imagen mediante súper-resolución estática

La idea de la reconstrucción con súper-resolución fue propuesta inicialmente por Huang y Tsay [HT84]. Ellos enfocaron el problema desde el dominio de la frecuencia para demostrar la posibilidad de reconstruir una imagen de resolución mejorada a partir de varias imágenes sin ruido y sub-muestreadas de la misma escena, basándose en el efecto de *aliasing* espacial. Asumían movimiento puramente traslacional y resolvían al mismo tiempo el problema de alineación (*registration*) y de restauración (la alineación conlleva la estimación de los desplazamientos relativos entre las observaciones y la alineación implica la estimación de muestras en una rejilla o *grid* uniforme de mayor resolución). La etapa de restauración se enfoca en la actualidad principalmente como un problema de interpolación con muestreo no uniforme. Otros resultados sugirieren una generalización simple de esta idea para imágenes de contorno borroso y con ruido. Así, en [KBV90], [BKV93] y [Kim94] se propone un algoritmo recursivo en el dominio de la frecuencia para la reconstrucción de una imagen de súper-resolución a partir de tomas borrosas y con ruido basado en una teoría de mínimos cuadrados recursiva ponderada. Este método es luego refinado por Kim y Su, quienes consideraron el caso de difuminados diferentes en cada observación de baja resolución, usando la regularización de Tikhonov para determinar la solución de un conjunto inconsistente de ecuaciones lineales [KW93].

Ur y Gross [UG92] proponen una alternativa en el dominio espacial, basada en los teoremas de muestreo generalizados multi-canal de Papoulis [Pap77], Yen [Yen56] y Brown [Bro81] para realizar una interpolación no uniforme de un conjunto de imágenes de baja resolución espacialmente desplazadas. Este paso es seguido por otro de corrección del difuminado (*deblurring*). Se asume en todo momento un conocimiento preciso de los desplazamientos relativos entre las imágenes de entrada. Otro método alineamiento-interpolación para súper-resolución partiendo de observaciones con traslaciones sub-píxel es el descrito en [VH99]. Srinivas y Srinath [SS90] proponen un algoritmo de reconstrucción de súper-resolución basado en el enfoque MMSE (*Minimum Mean Square Error*) para el problema de restauración de múltiples imágenes y de interpolación de las imágenes restauradas en una sola. Todos estos métodos de restauración por súper-resolución están limitados al desplazamiento de traslación uniforme y global entre las diferentes tomas de la misma escena, a un movimiento borroso o difuminado (*blur*) lineal invariante en el espacio (LSI, *Linear Space Invariant*) y a ruido aditivo homogéneo.

Peleg [PKS87] y [IP91] sugiere un enfoque diferente al problema de reconstrucción de súper-resolución, basándose en el método de retro-proyección iterativo (IBP, *Iterative Back-Projection*) adoptado de la Tomografía Asistida por Computador (TAC). Este método empieza con una predicción inicial de la imagen de salida, proyecta el resultado temporal a las tomas de baja resolución (mediante simulación), y actualiza la predicción temporal según el error de esta simulación. Este método no está limitado como los anteriores a características específicas del movimiento y permite flujos de movimiento suave y arbitrario. En este caso, el proceso de alineación usa el procedimiento descrito en [KPB88] seguido de un algoritmo de súper-resolución iterativo en el que se minimiza el error entre el conjunto de imágenes de baja resolución observadas y las obtenidas mediante la simulación de las imágenes de baja resolución a partir de las imágenes de alta resolución. Dado que la alineación se realiza de forma independiente a la restauración, la precisión del método depende fundamentalmente de la precisión de los desplazamientos estimados. El método de alineación sub-píxel descrito en [KPB88] considera dos imágenes como dos funciones relacionadas entre si mediante desplazamientos horizontales y verticales y un ángulo de rotación. Se realiza una expansión en series de Taylor de la imagen original en términos de parámetros de movimiento y se trata de minimizar una función de error mediante el cálculo de sus derivadas con respecto a sus parámetros de movimiento. Sin embargo la convergencia del algoritmo propuesto sólo ha sido comprobada para deformaciones geométricas afines entre las imágenes tomadas [IP93].

Tom y Katsaggelos [TK95] han tenido en cuenta la interdependencia de las etapas de alineación, interpolación y restauración, planteando el problema como una estimación de máxima verosimilitud (*Maximum Likelihood Estimator, MLE*) que es resuelto con un algoritmo de maximización de la expectativa (*Expectation Maximization*, EM). El problema se enmarca en un entorno multi-canal en el que la ecuación que describe la formación de las imágenes de baja resolución contiene variables de desplazamiento, de difuminado y de ruido. La estructura de las matrices implicadas en la función objetivo permiten realizar los cálculos de forma eficiente en el domino de la frecuencia. El problema MLE resuelve el cálculo de los desplazamientos sub-píxel, de las variaciones de ruido de cada imagen y de la imagen de alta resolución. En [KIA+93] se utiliza un teorema de muestreo no uniforme propuesto en [CPL85] para transformar muestras espaciadas de forma no uniforme adquiridas por múltiples cámaras en una sola rejilla de muestreo uniforme. No obstante, si las cámaras tienen la misma apertura aparecerán severas limitaciones tanto en la disposición como en la configuración de la escena. Esta dificultad es solventada usando múltiples cámaras con diferentes aperturas. En [CB00] se describe un método de súper-resolución usando deformaciones de imágenes. Las características de deformación de las lentes reales son aproximadas mediante el acoplamiento

del modelo de degradación del sistema de adquisición de las imágenes con la etapa de integración del nuevo muestreo [Fan86].

Otro enfoque al problema de reconstrucción de súper-resolución es el propuesto por Schultz y Stevenson [SS95] y [SS96]. Su enfoque usa el estimador de máximo a posteriori (*Maximum A-posteriori Probability*, MAP), usando previamente el campo aleatorio de Huber-Markov (*Huber-Markov Random Field,* HMRF). Se supone que la distorsión de las imágenes tomadas es debida simplemente al promediado, y que el ruido aditivo de las tomas es un vector Gausiano independiente e idénticamente distribuido (*independent and identically distributed, iid*). Esta elección provoca que la complejidad algorítmica del problema sea no cuadrática, con la consiguiente complicación del problema de minimización resultante.

En [WDM99] se propone un algoritmo de restauración ciego de imágenes de alta resolución multi-canal usando múltiples filtros de respuesta al impulso finita (*Finite Impulse Response*, FIR). Este proceso consta de dos etapas, una primera de deconvolución a ciegas desde múltiples entradas y múltiples salidas (*multi-input-multi-output*, MIMO) usando filtros FIR y una segunda que consiste en una separación a ciegas de los componentes poli-fase combinados. Debido al proceso de sub-muestreo, cada imagen de baja resolución es una combinación lineal de los componentes poli-fase de la imagen de entrada de alta resolución, ponderados con los componentes poli-fase de las respuestas al impulso de los canales individuales. En consecuencia, se presenta el problema como la deconvolución ciega en dos dimensiones de un sistema MIMO dirigido por los componentes poli-fase de una señal limitada en banda. Dado que la deconvolución MIMO a ciegas basada en estadísticos de segundo orden contiene cierta interdependencia coherente, los componentes poli-fase necesitan ser separados tras la deconvolución.

Stark y Oskoui sugieren por primera vez un conjunto de estimaciones teóricas de imágenes de alta resolución [SO89] usando la formulación de proyecciones en conjuntos convexos (*Projection Onto Convex Sets*, POCS). El método fue extendido por Tekalp [TOS92] para incluir el ruido observado. Además, observaron que la formulación POCS también podía ser usada como un método nuevo para la restauración de imágenes con difuminado variable en el espacio. Se mostraba que tanto el problema de la reconstrucción de imágenes de alta resolución como el de la restauración variable en el espacio podían ser reducidos al problema de resolver un conjunto de ecuaciones lineales simultaneas. Calle y Montanvert afrontan el problema de incrementar la resolución como un problema inverso de la reducción de imágenes [CM98]. La imagen de alta resolución debe pertenecer al conjunto de imágenes que mejor se aproxima a la estimación reducida. La proyección de una imagen

en este conjunto proporciona una de las posibles imágenes aumentadas y a este proceso se le ha denominado inducción. Por lo tanto, el problema de súper-resolución se plantea mediante la elaboración de un modelo regularizado que restaura los datos perdidos durante el proceso de aumento de tamaño. Otros enfoques teóricos alrededor del concepto POCS son los recogidos en [SO89], [TO92] , [PST94] y [PST95]. El principal resultado obtenido es la posibilidad de definir conjuntos convexos que representan fuertes restricciones en la imagen que se desea obtener. El procedimiento de reconstrucción basado en POCS obtiene los mismos beneficios que el método IBP anteriormente mencionado: movimiento arbitrario suave, distorsión lineal variable en espacio y ruido aditivo no homogéneo. De hecho, POCS puede ser mejor que IBP, dado que las restricciones no lineales pueden ser fácilmente combinadas con el proceso de reconstrucción. No obstante, la aplicación en la práctica de las proyecciones que usan el método POCS es computacionalmente intensiva, lo que limita su campo de aplicación.

Schultz y Stevenson describen en [SS94] un método para aumentar la definición de una sola imagen observada usando súper-resolución o interpolación. Ambos proponen un método de expansión no lineal de la imagen que preserva la discontinuidad, donde las técnicas de estimación MAP optimizan una función convexa. Aunque consideran tanto imágenes con y sin ruido, excluyen cualquier tipo de difuminado en su modelo. En [HBA97] se presenta un entorno para estimar conjuntamente los parámetros de alineamiento de las imágenes y la propia imagen de alta resolución. Los parámetros de alineamiento, desplazamientos horizontales y verticales en este caso, son actualizados iterativamente junto con la imagen de alta resolución mediante un procedimiento de optimización cíclico. En [HBB+98] se expone un proceso de dos etapas para estimar los parámetros de alineación seguido de la reconstrucción de la imagen de alta resolución partiendo del conocimiento del sistema óptico y de la matriz de sensores usada. La imagen de alta resolución estimada se obtiene mediante la minimización de una función de coste regularizada basada en el modelo de la observación. También se muestra como con la elección correcta de los parámetros de ajuste, el algoritmo exhibe gran robustez en presencia de ruido. Para minimizar la función de coste se utilizan tanto el procedimiento de optimización de gradiente descendiente como el de gradiente descendiente conjugado. En [BK00] Baker y Kanade proponen un algoritmo que aprende en base a reconocimientos anteriores para algunos tipos específicos de escenarios y lo aplican al reconocimiento de caras y de texto. También muestran que para grandes factores de ampliación, las restricciones de reconstrucción de súper-resolución no permiten obtener más información útil a medida que la ampliación aumenta.

Elad y Feuer proponen una metodología unificada que combina las tres herramientas de estimación principales en la restauración de imágenes: MLE, estimador MAP y la aproximación teórica usando POCS. El procedimiento de restauración propuesto es general, pero asume un conocimiento explícito del difuminado e impone restricciones de movimiento suave. También proponen un algoritmo híbrido que combina los beneficios de sencillez del MLE y la habilidad de POCS para incorporar restricciones no elipsoidales. Este algoritmo híbrido resuelve un problema de minimización convexa con restricciones, combinando todo el conocimiento que *a priori* se obtiene del resultado requerido en el proceso de restauración. Cheeseman aplica en [CKK+94] estimación Bayesiana con un modelo Gausiano previo al problema de integrar múltiples imágenes de satélites observadas por la cápsula espacial Viking. También se presentan algunas extensiones de este método incluyendo reconstrucción en tres dimensiones.

La mayoría de los algoritmos de súper-resolución propuestos en la literatura están confinados a aplicaciones en dos dimensiones. En [SBZ+96] se propone una versión en tres dimensiones donde se estima el albedo de alta resolución de una superficie Lambertiana partiendo del conocimiento de la altura de alta resolución y viceversa. El problema de reconstrucción de superficies ha sido formulado como uno de maximización anticipativa y ha sido afrontado dentro de un marco probabilístico usando modelos de campos aleatorios de Markov (*Markov Random Field*, MRF). La idea ha sido extendida al problema inverso de reconstrucción simultanea del albedo y la altura en [SBZ95] usando la extensión del teorema de muestro generalizado de Papoulis para casos de dimensión N.

Dentro de la comunidad óptica, la resolución se describe en términos de función de transferencia óptica[1] (*Optical Transfer Function*, OTF) . Esto ha dado lugar a una definición ligeramente diferente de súper-resolución. En [Hun95] la súper-resolución se define como el procesado de una imagen que permite recuperar información del objeto más allá del ancho de banda de la frecuencia espacial del sistema óptico que forma la imagen. El tamaño físico de la imagen permanece constante. Esto puede verse como un método equivalente a la extrapolación en el dominio de la frecuencia [Jai89]. El algoritmo de Gerchberg es uno de los primeros algoritmos de súper-resolución [Ger74]. Aquí las restricciones que existen en el objeto son impuestas por la imagen en el dominio espacial. La imagen modificada se transforma al dominio de Fourier y a continuación se imponen restricciones en el mismo dominio de Fourier. Esta restricciones vienen típicamente del conocimiento de la

[1] La OTF se define como la autocorrelación de la función de transferencia que limita físicamente la energía lumínica que entra en el sistema óptico.

transformada de Fourier bajo los límites de difracción. Los datos de la transformada de Fourier modificados son luego inversamente transformados al dominio espacial. Walsh y Delaney presentan una modificación del algoritmo de Gerchberg que calcula directamente los componentes de la frecuencia espacial por encima del límite de difracción [WN94]. Shepp y Vardi usan una técnica iterativa basada en una estimación ML de los estadísticos de Poisson de la emisión de positrones para la tomografía por emisión de positrones. Hunt y Seminelli proponen un algoritmo similar en el cual los estadísticos de Poisson se dan por sentados y se reconstruye una estimación MAP de forma iterativa. Las prestaciones de estos algoritmos de súper-resolución han sido estudiadas en [SHN93].

En [Raj01] el autor expone una aplicación interesante de las técnicas de súper-resolución. El difuminado por desenfoque debido a las diferentes profundidades de los objetos en una imagen con apertura real se usa como una pista natural en el proceso de súper-resolución. El concepto de profundidad por desenfoque [CR99] se ha incorporado en este esquema para recuperar el difuminado por desenfoque variable en el espacio y desconocido. Dado que la profundidad está relacionada con el difuminado relativo entre dos o más observaciones de la misma escena, se puede recuperar un mapa de profundidades de gran densidad. El autor propone un método para estimaciones MAP de súper-resolución simultaneas tanto para la imagen como para los campos de profundidad.

En los resultados anteriormente mencionados el problema de súper-resolución ha sido definido como el objetivo de crear una imagen particular de calidad mejorada a partir de un conjunto finito de imágenes de menor calidad. Todos estos resultados se basan en una estimación del movimiento (parámetros de alineación) entre las imágenes de partida. Sin embargo existen muchas circunstancias en las que tal estimación es muy difícil de generar (por ejemplo movimientos bruscos y cambios de planos). En estos casos la imagen reconstruida de SR que se obtiene es de muy baja calidad y tiene poca utilidad práctica. Por otro lado, la mayoría de los métodos propuestos carecen de implementaciones prácticas, dejando a un lado en muchos casos el problema de las arquitecturas de proceso más adecuadas y el del tipo de prestaciones deseadas en velocidad, precisión o coste de los sistemas de proceso.

2.2.2 Reconstrucción de imagen mediante súper-resolución dinámica

Una generalización importante y directa de la aplicación de restauración de imágenes específicas es la restauración de secuencias de imágenes continuas [KDE+89], [KKE+91], [KL91] y [PTS93]. El término "continuas" se refiere a la presunción básica de que la secuencia de imágenes contiene una escena filmada, con vectores de movimiento muy pequeños en comparación con el tamaño de las imágenes. Suponemos una cámara de video estándar como la fuente primaria de tales señales. El interés teórico de este problema reside en el hecho de que la solución que se requiere implica generalizar de algún modo los resultados conocidos de la teoría de restauración de una sola imagen. Además, desde el punto de vista práctico, la reconstrucción por súper-resolución dinámica permite suprimir el ruido aditivo en una secuencia continua de imagen lo que es una fase de preproceso importante en muchas aplicaciones como en codificación de secuencias de imagen y en visión por computador [Lee80], [KSS+85] y [ML85]. El aclarado de imágenes de este tipo es una herramienta importante para la mejora de los datos visuales antes de presentárselos al observador humano o al sistema de decisión, y es una fase clave del preprocesado. A pesar de su importancia, este problema de restauración de una secuencia de imágenes no ha sido tan ampliamente tratado como el problema homólogo de restauración de una sola imagen. Esto puede deberse en parte a la cantidad de memoria y carga computacional que se requiere cuando se aplica incluso el más simple de los algoritmos de restauración a tales imágenes. Los métodos de restauración existentes para secuencias de imagen continuas [KDE+89], [KKE+91], [KL91] y [PTS93] son sólo aquellos cuya simplicidad reduce adecuadamente los requisitos computacionales, proporcionando no obstante alguna solución de compromiso aceptable en los resultados. La simplicidad se alcanza típicamente haciendo suposiciones en el modelo, tales como inmovilidad, homogeneidad, causalidad en la superficie de la imagen, y localidad en el filtro de aclarado (*deblurring*).

Como ya se comentó, la mayoría de los algoritmos de súper-resolución aplicados a vídeo son extensiones de sus equivalentes para una sola imagen. Irani y Peleg minimizan el error cuadrático medio entre las imágenes observadas y simuladas usando un método de retro-proyección similar al usado en la tomografía asistida por computador [IP93]. Este método es el mismo que el usado por ellos en la súper-resolución de una sola imagen a partir de observaciones desplazadas. No obstante, aquí el factor clave es el cálculo preciso del movimiento de la imagen. Después de calcular el movimiento para diferentes regiones de la imagen, dichas regiones son mejoradas mediante la fusión de fotogramas (*frames*) sucesivos

que convergen en la misma región. Posibles mejoras incluyen un aumento de la resolución, el relleno de las regiones con oclusiones y la reconstrucción de objetos transparentes. Anteriormente, en [KPB88] se minimizaba la misma diferencia, pero el método de minimización era relativamente simple: cada píxel era examinado y a su vez su valor era incrementado en una unidad, mantenido constante o decrementado en una unidad, de tal forma que la función de coste decreciese. En [BBZ96] se optimiza un función de coste que, además de las diferencias cuadráticas entre las imágenes de baja resolución observadas y simuladas, contiene restricciones de continuidad de segundo orden de la imagen reconstruida. Asimismo, la imagen de baja resolución simulada tiene en cuenta el difuminado debido al movimiento, el difuminado óptico y el promediado de la señal en cada celda CCD debido al muestreo espacial.

Schultz y Stevenson usan el algoritmo modificado de ajuste de bloques jerárquico para estimar los vectores de desplazamiento sub-píxel y luego resolver el problema de estimar el fotograma de alta resolución dada una secuencia de baja resolución y lo formulan usando la estimación MAP. Esto da lugar a un problema de optimización con restricciones con un mínimo único [SS96]. Este método es similar a su propio método de expansión de imágenes estáticas descrito en [SS94]. En [PST97] se propone un modelo completo de adquisición de vídeo con rejilla de muestreo arbitraria y tiempo de apertura distinto de cero. Se propone un algoritmo basado en este modelo usando la teoría de POCS para reconstruir vídeo de alta resolución a partir de secuencias de imágenes de baja resolución. No obstante, las prestaciones del algoritmo de súper-resolución basado en POCS estarán limitadas en última instancia por la efectividad del estimador de movimiento y del modelo usado. Por supuesto, este hecho es aplicable a cualquier algoritmo de súper-resolución basado en estimación del movimiento. En [EST97] se extiende la técnica introducida en [PST97] a escenas con múltiples objetos en movimiento introduciendo los conceptos de mapas de validación y de mapas de segmentación. Los mapas de validación fueron introducidos para permitir reconstrucciones robustas ante la presencia de errores en la estimación del movimiento. Los mapas de segmentación permiten procesamiento basado en objetos. Además, el método propuesto es capaz de manejar casos de oclusiones. El algoritmo de mejora de vídeo de súper-resolución propuesto por Shah y Zakhor [SZ99] también considera el hecho de que la estimación de movimiento usada en el proceso de reconstrucción puede ser imprecisa. Con este propósito, su algoritmo busca un conjunto de estimaciones de movimiento candidatas en lugar de un solo vector de movimiento para cada píxel, para a continuación calcular un denso mapa de movimientos con precisión sub-píxel usando tanto los valores de crominancia como los de luminancia. La estimación del fotograma de alta resolución es subsecuentemente generada mediante un método basado en el algoritmo de Landweber. En [HKK97] se define

una función convexa suave de múltiples entradas y se usa para obtener una secuencia de vídeo de alta resolución globalmente óptima. Baker y Kanade proponen un algoritmo para la estimación simultanea de vídeo de súper-resolución y flujo óptico [Sin92a] y [BFB94], tomando como entradas secuencias de vídeo convencional. Este algoritmo resulta ser especialmente útil en la súper-resolución de secuencias de caras.

2.2.3 Estimación del movimiento

Un ingrediente importante en el proceso recursivo de reconstrucción de súper-resolución es el movimiento de la escena, el cual debe ser estimado a partir de las imágenes tomadas. Existen muchos enfoques y métodos para la tarea de la estimación de movimiento entre dos imágenes dadas [LK81], [Ter86], [Chi92] y [CKM+93]. Un grupo de estos métodos ampliamente analizados es el encuadre diferencial, propuesto inicialmente por Horn y Schunck [HS81]. En su enfoque la tarea de estimación de movimiento se convierte en un problema de estimación lineal, donde el movimiento es definido con vectores de movimiento para cada píxel, llamado flujo óptico. El principal inconveniente del enfoque de Horn y Schunck es que su algoritmo olvida por completo los resultados de estimación previos disponibles en la secuencia de imágenes, además de ser un método iterativo no apto para trabajar con prestaciones de tiempo real. El método separa el problema de los cambios de plano en la escena y da por hecho que el flujo del movimiento es suave en el tiempo.

Chin y Willsky [CKM+93] y [Chi92] proponen una generalización del algoritmo de Horn y Schunck, con el propósito de superar el inconveniente de no usar los resultados de estimación previos. Su algoritmo es una aproximación muy compleja al perfectamente conocido Filtro Kalman [CC90], aplicado a la tarea de estimación recursiva por flujo óptico. Otros autores han intentado hacer uso de la suavidad temporal del flujo óptico, tal como se presenta por ejemplo en [Sin92b] y [FL95].

La estimación de movimiento juega un papel importante en cualquier aplicación de súper-resolución [Bov00]. De hecho, no es posible ninguna reconstrucción de súper-resolución sin una correcta estimación del movimiento entre imágenes. Este problema está en la base de nuestro interés en el campo de la estimación de movimiento. Otra motivación importante para fijar nuestra atención en la estimación de movimiento es el hecho de que para los diferentes enfoques de la estimación de movimiento, el modelo de representación obtenido está relacionado muy de cerca y es muy similar al modelo obtenido para el problema de

reconstrucción de súper-resolución dinámica, lo que significa que las soluciones propuestas para la solución de un problema pueden servir también para la solución del otro.

Aunque los métodos de estimación de movimiento anteriormente expuestos resultan ser muy precisos, y además los algoritmos de súper-resolución resultan ser muy sensibles a la precisión alcanzada en la estimación del movimiento, en esta tesis se ha optado por otro método alternativo de estimación del movimiento ampliamente usado en los sistemas de codificación de vídeo: el método de ajuste por bloques (*block matching*) [BK96], [HB98], [Ol98] y [Bot00]. La menor precisión que ofrece este método se compensa con su gran velocidad de ejecución, alcanzando sin mayores problemas prestaciones de tiempo real en la mayoría de los casos. El principal inconveniente del ajuste por bloques es que se obtiene un vector de movimiento que es compartido por todo un bloque de píxeles, en contraste con los otros métodos capaces de estimar el movimiento individual de cada píxel. En contrapartida, los métodos de ajuste por bloques se ven menos afectados por el ruido de fondo, aumentando la convergencia de las soluciones. Adicionalmente, la resolución del estimador se ha incrementado a valores sub-píxel, aumentando de esta forma la sensibilidad del sistema ante pequeños desplazamientos. Existen también estimadores de movimiento por ajuste de bloques en el dominio de la DCT [KC98] muy adecuados para ser usados en codificadores de vídeo, pero su elevada complejidad los aleja igualmente de las prestaciones deseadas de bajo coste y tiempo real.

2.3 Arquitecturas para el procesamiento de imágenes y vídeo

Una vez desarrollado y verificado el funcionamiento a nivel de comportamiento de un algoritmo para el procesamiento de imágenes, y si se desean alcanzar prestaciones de tiempo real, será necesario implementarlo, en el sentido más genérico, sobre alguna plataforma *hardware/software*. Es difícil que los algoritmos dedicados al procesamiento de imágenes puedan ofrecer prestaciones de tiempo real en una implementación totalmente *software*, ya que suelen incluir tareas muy intensivas computacionalmente. La forma clásica de resolver este problema es mediante el particionado del algoritmo, separando las tareas computacionalmente intensivas, que serán mapeadas en coprocesadores más rápidos (aceleradores *hardware*), de las tareas más intensivas en control, que serán mapeadas en unidades que ofrezcan mayor flexibilidad (procesadores programables). De esta forma conseguimos una solución de compromiso, en la cual el sistema final verá incrementada sus prestaciones manteniendo un grado aceptable de configurabilidad. A continuación

estudiaremos diferentes posibilidades usadas a la hora de seleccionar unidades de procesamiento de imágenes.

2.3.1 Procesadores estándares adaptados con soporte de vídeo

El tiempo empleado en ejecutar aplicaciones de procesado de vídeo en los procesadores convencionales supera con mucho los límites razonables. Por esta razón han comenzado a emplearse otros enfoques en la construcción de sistemas multimedia.

Los diseñadores de arquitecturas han observado que el tamaño de los tipos de datos que se proporcionan en los procesadores resulta excesivo para alojar la estructura y el rango dinámico de la información de tipo multimedia con la que se necesita operar. Al almacenar una información de 8 bits en las palabras disponibles de 32 ó 64 bits se desperdicia espacio. Sin embargo, en una palabra de información de 32 bits se pueden ubicar simultáneamente 4 datos de 8 bits en el mismo espacio. En los procesadores que trabajan con datos de 64 bits en este espacio se pueden alojar 8 datos de 8 bits ó 4 de 16. De esta forma, en un número menor de unidades de almacenamiento se puede almacenar mayor información. Si se tratan estos formatos de datos como válidos y se modifica la ruta de datos para realizar operaciones sobre cada uno de los distintos campos de forma concurrente, entonces se puede acelerar la ejecución de las aplicaciones de forma significativa. Estas nuevas operaciones emplean las mismas técnicas SIMD que se emplean en algunos multiprocesadores, donde una misma instrucción se ejecuta sobre datos diversos. La aceleración lograda permite la codificación y decodificación de vídeo en aplicaciones de bajo régimen binario, y puede lograr igualmente decodificación MPEG de moderada definición en tiempo real.

Inspirados en estas ideas y objetivos, Hewlett-Packard diseñó el PA-7100LC, que incorpora un pequeño juego de instrucciones multimedia denominado MAX-1 [Lee95]. Tras esto Sun Microsystems introdujo este tipo de extensiones especiales para aplicaciones multimedia en sus microprocesadores UltraSPARC. A esta extensión del juego de instrucciones se le denominó Visual Instruction Set, o VIS [KMT+95] y [TONH96].

Posteriormente Hewlett-Packard desarrolló el MAX-2 [Lee96], que consistía en el MAX-1 con instrucciones añadidas para alineamiento de datos y mayor paralelismo en las subpalabras. Asimismo Intel incorporó el juego de instrucciones MMX en sus procesadores

Pentium [PW96]. Estas extensiones eran muy similares a las anteriores, incorporando además instrucciones de multiplicación de subpalabras en paralelo.

Otros fabricantes que han empleado técnicas similares han sido Silicon Graphics, que introdujo MDMX [MIP97], y Compaq con su juego de instrucciones MVI para el Procesador Alpha 21264 especialmente ideado para acelerar el algoritmo MPEG-2.

En este sentido el autor desarrolló un procesador SPARC v.8 en VHDL [Mar95] al que luego se le añadieron extensiones basadas en el VIS [VIS] de los procesadores UltraSPARC [BMCN96]. Además, estos trabajos permitieron establecer una metodología de prototipado rápido de núcleos de procesamiento RISC de altas prestaciones orientados a multimedia usando VHDL [BMCN97]

La práctica totalidad de procesadores estándares nuevos se están desarrollando con ciertas capacidades de procesado de vídeo en tiempo real. Al haber superado ampliamente los 600 MHz de reloj, muchos procesadores estándares pueden procesar vídeo de bajas prestaciones. A estos procesadores con extensiones en su juego de instrucciones a veces se les denomina como NSP (*Native Signal Processors*) [LBSL97] y [Lap95].

2.3.2 DSPs y Procesadores de Señales de Aplicación Específica ASSP

En muchas circunstancias debido a las características de la aplicación, la naturaleza de las tareas que van a coexistir e interrelacionarse son diversas. Para una mayoría de las tareas podría convenir que el procesador aún siendo de propósito general en su juego de instrucciones, tuviera cierto tipo de estructuras especializadas que facilitase la ejecución de determinados métodos de procesado. En este sentido se dice que el procesador está orientado a un dominio específico de aplicación [BCF+97]. Ejemplos típicos de dominios específicos son los orientados al proceso de señal, ya sean flujos de datos (*streams*) de audio o de vídeo. Esto supone un paso más en el camino de las arquitecturas especializadas respecto a la categoría expuesta en la sección 2.3.1 basada en extensiones del juego de instrucciones.

En este tipo de procesadores la arquitectura puede tener ciertas estructuras que permitan una ejecución acelerada de determinados cálculos necesarios en las aplicaciones del dominio en cuestión. Esta alternativa es la que ha dado origen, entre otros, a los procesadores para el tratamiento digital de señales, o DSPs. A modo de ejemplo, en los DSPs normalmente

se dispone de un multiplicador para acelerar las multiplicaciones y, en algunos casos, *hardware* específico para dar soporte a la indexación de datos consecutivos. La disposición consecutiva de datos es una organización típicamente utilizada para almacenar tablas de datos (que normalmente proceden de señales que evolucionan continuamente con el tiempo). De este modo, la ejecución de estas subtareas se realiza en menor tiempo, y además, en determinados casos, se prescinde de destinar circuitería específica externa para realizarlas, con lo que se evita el aumento del coste del sistema, y se logra un mejor rendimiento global.

En el grupo de investigación dentro del IUMA se ha usado profusamente la familia TMS320 de Texas Instruments [DSP]. En concreto, el autor desarrolló un algoritmo paralelo para el bucle de codificación MPEG capaz de distribuir de forma dinámica la carga computacional entre los cuatro procesadores VLIW disponibles en el TMS320C80 [GMC+98].

Un problema importante en este ámbito es la decisión sobre la longitud de palabra del procesador. Existen herramientas [fro] y métodos [SK95], [SVR+97] y [CRS+98] para obtener y refinar implementaciones en punto fijo desde análisis de alto nivel en MATLAB o C++ en coma flotante.

Una amplia documentación sobre núcleos DSP disponibles para integración en sistemas en un chip puede consultarse en [Bie95]. Es recomendable la consulta de guías de vendedores de núcleos DSP, por ejemplo, en [ISD]. Ejemplos de incorporación de núcleos DSP modificados para arquitecturas de procesadores de señal de aplicación específica (ASSP) se dan en [SM95] y [Mad95]. El programa RASSP de ARPA mantiene información de sus resultados en [RAS].

2.3.3 Procesadores específicos de vídeo

La carga computacional del procesamiento multimedia está dominada por las tareas de procesado de vídeo [Pir98], [Ack93], [Ack94], [Ack95] y [Nuñ95]. Estas tareas requieren realizar operaciones complejas sobre grandes volúmenes de datos a elevadas velocidades de muestreo. Aparecen requisitos de tiempo real para satisfacer las exigencias de la percepción visual humana. Además, se necesita manejar flujos (cada vez con más frecuencia denominados *streams*, y *stream processors* los procesadores que operan con ellos) de diferentes tipos de datos. La viabilidad de muchas aplicaciones multimedia depende de esquemas de compresión que faciliten la transmisión y almacenamiento de datos multimedia.

Entre los estándares de compresión existentes pueden mencionarse los H.261, H.263, H.263++ y H.26L del ITU-T, que cubren aplicaciones de comunicaciones como videotelefonía; el MPEG-1 de ISO, que se emplea en almacenamiento y reproducción CD-ROM; y el estándar más genérico MPEG-2, que se dirige a aplicaciones como difusión de TV o como vídeo bajo demanda. El estándar MPEG-4 de ISO, utiliza una codificación más eficiente, y ofrece mayor funcionalidad como la integración de fuentes sintéticas y naturales o la interacción del usuario basada (o guiada) en los contenidos.

La literatura científica sobre arquitecturas de procesadores específicos de vídeo es muy amplia. En el breve resumen que hacemos seguimos la clasificación de [Fer98] y de [SPM98] para algunos de los aspectos VLIW más importantes.

Al ser los algoritmos multimedia notablemente sofisticados, el éxito comercial de las aplicaciones descansa en su eficiente ejecución en procesadores estándares de altas prestaciones o en su implementación VLSI. Los procesadores estándares actuales (RISCs, superescalares, etc.) no pueden procesar aplicaciones multimedia de tipo general sin ser adaptados. Estos procesadores no están orientados al procesado de señal, al procesado de *streams*, y además son demasiado caros y con demasiado consumo para el mercado de aplicaciones aisladas multimedia, el campo de mayor crecimiento en los próximos años. A su vez, los DSPs convencionales, aunque están orientados y optimizados para el procesado de audio y voz, no llegan a las prestaciones que requiere el vídeo. En consecuencia, se están desarrollando arquitecturas adaptadas a vídeo derivadas de procesadores y DSPs estándares, y arquitecturas específicas, en ambos casos manteniendo objetivos de bajo coste. Existe un proceso de convergencia entre ellas. Las primeras suelen orientarse a aplicaciones de bajo régimen binario, o bien de decodificación en tiempo real. Las segundas suelen afrontar el reto de codificar MPEG-2 en tiempo real, así como los perfiles y niveles superiores del estándar. La integración monolítica de un decodificador MPEG MP@ML data de 1994 [Tho94]. Hoy en día muchas empresas como Thomson, ST Microelectronics, Intel, LSI Logic, C-Cube, IIT o IBM disponen de estos decodificadores, cuyo mercado principal son las cajas de usuario (*set-top box*) de los equipos conectados al aparato de TV. También se dispone ya de chips codificadores MPEG-2 MP@ML, como los DV^X5110 y DV^X6210 [Ccu97], pero el problema del diseño de codificadores integrados en un solo chip para niveles y perfiles superiores de MPEG-2 continua siendo un reto mucho más exigente que la decodificación. Revisamos a continuación la evolución ocurrida en este tipo de arquitecturas en la última década.

Las arquitecturas específicas se clasifican en fijas y programables. Las fijas suelen estar formadas por un elemento de control y varias unidades específicas dedicadas a diferentes

partes del algoritmo, y adoptan una arquitectura tipo ASIP o ASSP. Las específicas programables, denominadas VDSP o VP, están formadas por una o varias rutas de datos programables con estructuras menos específicas programables, configurables o parametrizables. Presentan mayor flexibilidad y utilidad conforme evolucionan los estándares. Los estándares varían en resolución de imagen, tasa binaria a la salida, sintaxis de la trama, márgenes de variación de diferentes parámetros, algoritmos de interpretación, carga computacional, etc., pero muchas estructuras de cálculo son comunes. La progresiva transparencia en la visibilidad de las estructuras de cálculo al nivel del formato de instrucción ha conducido a una variante de las arquitecturas específicas programables denominadas VLIW (*Very Long Instruction Word*).

a) Arquitecturas fijas ASIP

El VRP (Video RISC Processor) CL4000 [Bur93] es un ASIP escalable. Con dos VRPs se puede codificar MPEG-1 en formato SIF. Con 10 VRPs se codifica MPEG-2 MP@ML [BK96]. Un sólo VRP2 CL4100 codifica MPEG-1 en tiempo real.

El VideoFlow [Lee94] codifica con dos chips H.261 en CIF o MPEG en SIF. Los Enc-C y Enc-M de NTT [Kon96, Ike96] codifican MPEG-2 SP@ML. Todas estas arquitecturas son ASIP.

b) Arquitecturas programables V-DSP

Entre las arquitecturas específicas heterogéneas V-DSP se encuentran los VC (Vision Controller), VP (Video Processor) [Bai92] y VCP [IIT93] de IIT, el AVP1300E para H.261 ó los AVP1400E y AVP4310E [Ack93] para mayores niveles, de Lucent Technologies, los VDSP [Aon92] y VDSP2 [Ara94], [Toy94] y [Aki94] de Matsushita, para MPEG o el chip-set VISC de LSI Logíc [LSI96] para MPEG-2.

Entre las arquitecturas específicas programables homogéneas V-DSP, se encuentra el VSP3 [Ino93] y [Eno93] de NEC, que codifica H.261 en CIF, o el HiPAR-USP [RK96] que puede codificar MPEG-2 MP@ML sin estimación de movimiento. Estas arquitecturas son homogéneas; en ellas los coprocesadores trabajan sobre una parte de la imagen.

c) Arquitecturas programables VLIW

La tendencia arquitectural a acoplar cada vez más fuertemente los coprocesadores o unidades funcionales con la misma unidad de control ha llevado a las arquitecturas monoprocesador de palabra larga (LIW) y muy larga (VLIW) [NC89] y [FDF98]. Estos procesadores son idóneos para aplicaciones multimedia, específicamente para procesado de imagen y gráficos, y, en menor medida, vídeo [DWWO96] y [MT97].

Los tres representantes VLIW más señalados son, quizás, la arquitectura VelociTI, de Texas Instruments [Ses98], la arquitectura Mpact2 de Chromatic Research [Pur98] y la arquitectura Trimedia de Philips Semiconductors [RS98]. Trimedia ha sido especialmente concebido para la decodificación de MPEG-2 en tiempo real, y probablemente ofrece las mejores prestaciones.

d) Arquitecturas programables para plataformas SOC

La creciente importancia del diseño basado en plataformas SOC conduce al predominio de un esquema arquitectural heterogéneo resultado de combinar las categorías b) y c) anteriores. Las arquitecturas de Philips C-HEAP [Lip97] y *Picasso* [PKL+99] pertenecen a esta categoría.

El presente trabajo de tesis se centra en el estudio de algoritmos iterativos de superresolución y su modificación para lograr prestaciones de tiempo real. Estos algoritmos se han desarrollado de tal forma que su ejecución se realiza usando los recursos proporcionados por el codificador de vídeo híbrido de la plataforma arquitectural *Picasso*, asegurando de esta forma una implementación de bajo coste. El problema de transformación mutua y adaptada entre algoritmos y arquitecturas suele denominarse problema de mapeado. Las aportaciones a este problema a partir de un algoritmo base y una arquitectura base son parte central de esta tesis.

2.4 Conclusiones

La riqueza de la señal con la que tratamos – secuencias de imágenes – depara una posibilidad nueva: reconstruir las imágenes de la secuencia con una resolución mejorada. Esta idea se basa en el hecho de que los datos filmados son tomados para cada imagen en una posición ligeramente diferente (debido tanto al movimiento de la cámara como al movimiento

de los objetos en la escena), de tal manera que pueden ser combinados para crear una imagen de salida con mayor resolución. Las secuencias de imágenes continuas contienen básicamente varias imágenes tomadas de cada objeto filmado, por lo que pueden ser combinadas para crear una descripción de súper-resolución del mismo. Mientras que en la aplicación de súper-resolución estática se creaba una imagen a partir de un conjunto de imágenes dado, en la súper-resolución dinámica aspiramos a crear una secuencia de imágenes de súper-resolución de la misma longitud que la secuencia de imágenes de peor calidad de la que partimos. Por tanto, la aplicación es una combinación de dos ideas: la reconstrucción de una imagen generando súper-resolución estática a partir de varias imágenes y la restauración de una secuencia de imágenes continuas. Cada uno de estos problemas es abordado en los capítulos 4 y 5 respectivamente de esta tesis. La conjunción de ambas ideas es una aportación novedosa que además puede plantearse con el objeto de obtener una secuencia con mejora de resolución en la secuencia bajo ciertas condiciones. En efecto, como se ha expuesto existen varios métodos conocidos para el problema de la reconstrucción de súper-resolución, pero ninguno da una solución al problema de la estimación progresiva en tiempo real de una secuencia de imágenes que proporcione en el tiempo súper-resolución espacial, logrando al mismo tiempo prestaciones de tiempo real y bajo coste. Los métodos existentes para la restauración de secuencias de imágenes continuas quedan todavía lejos de ofrecer prestaciones en tiempo real para secuencias de vídeo reales, al usar complejos métodos iterativos o métodos POCS, o necesitar partir de un perfecto conocimiento del movimiento y del difuminado, o realizar una conjunto de presunciones tales como suavidad en el movimiento, inexistencia de oclusiones o globalidad del movimiento. La aplicación que se estudia en esta tesis toma una secuencia continua de imágenes de baja calidad sin restricciones y su objetivo es reconstruir una secuencia continua de imágenes de calidad mejorada mediante súper-resolución, ofreciendo prestaciones de tiempo real y soluciones de bajo coste. Esta aplicación no sería posible sin los recursos computacionales existentes hoy en día. El reto es fundamentalmente obtener nuevos algoritmos y optimizarlos para su implementación en arquitecturas de computación de bajo coste, para lo que es necesario concebir la solución del problema en base a los potentes núcleos de computación de video destinados a los mercados de consumo, así como introducir nuevos núcleos computacionales flexibles que permitan dar soporte a súper-resolución.

Capítulo 3

Lo esencial es invisible a los ojos.

Antoine de Saint-Exupéry (1900-1944).
"*El principito*". Escritor y aviador francés.

El cambio es mi tema. Vosotros los dioses, cuyo poder ha creado
Todas las transformaciones, ayudad al pensamiento del poeta,
Y haced que la ininterrumpida sucesión de mi canto fluya
Desde los primeros comienzos hasta los días que conocemos.

Ovidio (43 a.C.-17 d.C.)
Poeta latino.

Codificación híbrida de vídeo

3.1 Introducción

El gran auge que han tenido en los últimos años los sistemas multimedia portátiles ha incentivado recientemente la investigación de nuevos diseños de bajo consumo, impulsada por las innovaciones tecnológicas en el diseño de circuitos y en la concepción de nuevos algoritmos y arquitecturas. Junto con el consumo, el coste de este tipo de sistemas está dominando cada vez más las tendencias del mercado. Estos dos factores están condicionando de forma importante la trayectoria seguida en la investigación de sistemas multimedia móviles.

La codificación híbrida de vídeo es un campo específico del área de tratamiento de señales multidimensionales, gráficos, imágenes y vídeo, área en la que existe una literatura científica muy abundante una visión general puede obtenerse en [BK96], [Kou95] y [Das95]. En este capítulo revisamos brevemente el estado el arte en codificación híbrida de vídeo, como área clave para la introducción masiva de sistemas multimedia portátiles y sólo en aquellos aspectos pertinentes a esta tesis, y en particular discutimos más extensamente a continuación la arquitectura *Picasso*, desarrollada en los Laboratorios Nat.Lab de Philips Research en Eindhoven, a la que ha contribuido el autor en el marco de colaboración

establecido entre el Nat.Lab y el IUMA, y que se encuentra en fase de transferencia a Philips Semiconductors para su entrada en producción.

La característica principal de *Picasso* es su flexibilidad para codificar vídeo para múltiples formatos estándares. Esta flexibilidad original se ha modificado para dar soporte a un rango más amplio de aplicaciones, en concreto las aplicaciones de súper-resolución tratadas en esta tesis. El proceso de modificación arquitectural y de diseño de la plataforma *Picasso* es un bucle que trata de iterar transformaciones e innovaciones algorítmicas con cambios e innovaciones en los bloques de proceso y en la arquitectura de coprocesadores, memorias y buses, mediante técnicas de diseño encuadradas en los conceptos de codiseño y *mapping* arquitectural de algoritmos.

La compresión de vídeo [BK96] juega un papel vital a la hora de hacer posible el procesamiento de secuencias de vídeo en sistemas multimedia portátiles (por ejemplo, en vídeo-teléfonos móviles, terminales IP móviles, etc.), lo que ha estimulado la evolución de estándares de compresión de vídeo como H.261 [H261] [Lio91], H.263 [H263] o MPEG-1 [Gal91], MPEG-2 [H262] y MPEG-4 [MPEG4], [Per96] y [MPF+96]. La arquitectura *Picasso* está pensada originalmente como un codificador de vídeo de bajo coste y bajo consumo basado en el estándar H.263, aunque diseñada siguiendo una filosofía de gran flexibilidad que permita incluir nuevos estándares de vídeo e imágenes estáticas con facilidad. Las restricciones de bajo coste y bajo consumo impuestas al codificador han sido alcanzadas mediante la creación de algoritmos novedosos y la concepción de arquitecturas altamente eficientes.

Las implementaciones del codificador H.263 han dado lugar hasta el presente a soluciones multi-chip [Ha+99], [Ta+98] y [BK96], debido a la presencia de grandes bucles de memoria (por ejemplo, 3.5 Mbits para imágenes VGA). No obstante, la compresión de las imágenes almacenadas en el bucle de memoria puede dar lugar a una solución de un solo chip. Esta compresión se ha logrado en *Picasso* mediante el uso de un esquema de codificación por zonas (escalable) basado en los planos de bits, tal y como se describe en [KVP00] (patente de Philips), donde la memoria de bucle está en el domino de la DCT. Además, el esquema de codificación escalable soporta compresión con pérdidas y sin pérdidas. Esta importante transformación algorítmica realizada para reducir el tamaño de la memoria de bucle tiene como contra partida el aumento de la cantidad de operaciones a realizar (*computational overhead*), al añadir cálculos adicionales de DCT/IDCT y de codificaciones/decodificaciones escalables hacia y desde la memoria de bucle. No obstante, este efecto puede ser minimizado encontrando un diseño arquitectural eficiente [POR+01] [PKL+99]. Este ha sido el reto de investigación encontrado y resuelto por *Picasso*.

La partición *hardware/software* del codificador H.263 de *Picasso* se ha efectuado siguiendo el criterio bien establecido de ubicar las tareas intensivas en cálculos (*compute-intensive*) en *hardware* (mínima flexibilidad – máximas prestaciones) y las tareas intensivas en control (*control-intensive*) en *software* (máxima flexibilidad – mínimas prestaciones). El codificador H.263 usa un modelo arquitectural basado en C-HEAP (CPU-*controlled Heterogeneous Embedded Architectures for signal Processing* [Lip97]) donde procesadores con diferentes niveles de flexibilidad ejecutan tareas concurrentes. A su vez, el protocolo de comunicaciones usado por C-HEAP esta basado en el modelo de Khan para redes de procesos [LP95].

3.2 Algoritmos de compresión de vídeo

3.2.1 Codificación genérica estándar

La compresión produce una representación compacta de una señal. Las señales de vídeo son buenas candidatas a la compresión debido a la alta correlación espacial y temporal que poseen. La mayoría de los estándares de codificación de vídeo (como H.261, H263, MPEG-1, MPEG-2 y MPEG-4) consiguen una alta correlación espacial y temporal. La FIGURA 3.1 muestra el diagrama de bloques del codificador de vídeo híbrido genérico. Un codificador híbrido combina la compresión en el dominio espacial con la compresión en el dominio temporal. La compresión en el dominio espacial se consigue mediante el uso de una transformación de descorrelación (la transformada discreta del coseno bidimensional) y la compresión en el dominio temporal se consigue usando predicción compensada del movimiento [Bh+96].

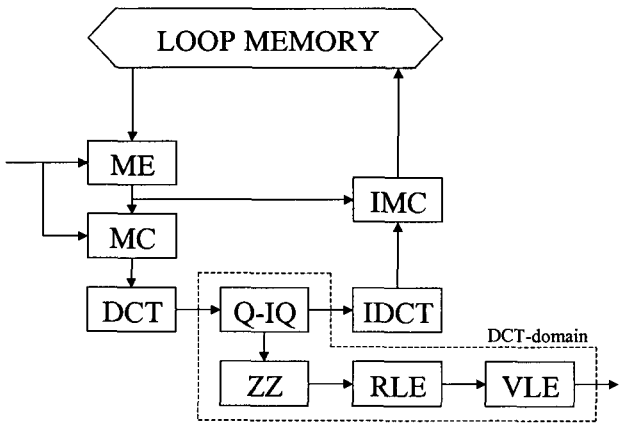


FIGURA 3.1. Diagrama de bloques del flujo de datos de un codificador de vídeo híbrido genérico.

Los bloques de proceso que componen el codificador híbrido son los siguientes: el estimador de movimiento (ME, *Motion Estimator*), el compensador de movimiento (MC, *Motion Compensation*) , el compensador de movimiento inverso (IMC, *Inverse Motion Compensation*), transformada discreta del coseno (DCT, *Discrete Cosine Transform*), transformada discreta del coseno inversa (IDCT, *Inverse Discrete Cosine Transform*), cuantificador (Q, *Quantizer*), cuantificador inverso (IQ, Inverse *Quantizer*), transformación en zig-zag (ZZ, *Zigzag transform*), codificador de longitud codificada (RLE, *Run Length Encoder*), codificador de longitud variable (VLE, *Variable Length Encoder*) y la memoria de bucle (LM, *Loop Memory*). Los bloques ME, MC e IMC junto con la memoria de bucle realizan la compresión en el dominio temporal. Los bloques DCT, Q, IQ, IDCT realizan la compresión en el dominio espacial. Los bloques RLE y VLE realizan la codificación aritmética para el canal.

El estimador de movimiento determina el mejor ajuste del macro-bloque actual (16x16 píxeles) en la imagen reconstruida almacenada en la memoria de bucle. Existen diversos criterios para obtener los vectores de movimiento. El criterio utilizado comúnmente para determinar el mejor ajuste es buscar el macro-bloque con menor suma de diferencias absolutas (SAD, *Sum of Absolute Differences*), expresado como recoge la ecuación (3.1), donde *A(i,j)* es la luminancia del macro-bloque actual y *S(i,j)* es la luminancia de un macro-bloque del área de búsqueda (N×P) definido por $(n,p) / n,p \in (N,P)$.

$$SAD(n,p) = \sum_{i=0}^{15}\sum_{j=0}^{15}\left|A(i,j) - S(i+n, j+p)\right| \tag{3.1}$$

El mejor ajuste dará asimismo el vector de movimiento del macro-bloque, es decir, las coordenadas $(n,p)_{SAD,min}$ dentro del área de búsqueda del bloque con menor SAD. Se calcula la diferencia entre el macro-bloque actual, y el macro-bloque con menor función de coste (MC), se transforma al dominio de la frecuencia (DCT), se cuantifica (Q), se codifica (ZZ, RLE y VLE) y se transmite al canal y al extremo decodificador junto con el vector de movimiento. Los datos transmitidos se procesan además localmente: se cuantifican inversamente (IQ), se les aplica la DCT inversa (IDCT) y se reconstruyen (IMC) para ser almacenados en la memoria de bucle y ser usados en la codificación de la siguiente imagen.

El procesador para la estimación del movimiento es el más intensivo en cálculos. En el caso de *Picasso* con precisión de ½ píxel, se deben evaluar 23 vectores de movimiento. Esto significa 23 accesos a la memoria de bucle, seguidos de una sola ejecución del resto del diagrama de flujo de datos para cada macro-bloque de entrada. El proceso se itera con cada nuevo macro-bloque hasta completar todo el cuadro de imagen.

Desde un punto de vista energético, es también esencial controlar minuciosamente la potencia disipada en el ME [PG98].

Se han incluido funciones de pre-filtrado y filtrado de bucle para reducir la cantidad de ruido de la señal, mejorando la relación señal-ruido de la salida. El pre-filtrado está basado en un algoritmo de filtrado IIR espacio-temporal de promedio adaptativo [Cam99]. El filtro de bucle esta basado en un algoritmo de filtrado temporal con compensación de movimiento [WWV97].

3.2.2 Codificación escalable

El codificador escalable [KVP00] usado es un esquema de codificación zonal basado en planos de bits que convierte los coeficientes de los bloques DCT en una cadena de bits. En el dominio temporal cada píxel se almacena usando 8 bits. En el dominio de la DCT cada coeficiente debe tener 12 bits. Por lo tanto, reducir el tamaño de la memoria en un factor de 4, en comparación con un codificador convencional, requiere un factor de compresión de 6 para los coeficientes DCT. Esto daría una media de 2 bits por coeficiente. El objetivo de la codificación escalable es el de comprimir los bloques DCT antes de almacenarlos en memoria. Debido al tamaño fijo de la memoria de bucle y según sea el tamaño disponible, pueden acontecer dos casos:

1. Que los coeficientes quepan en memoria. En este caso la compresión es sin pérdidas y no hay pérdida de información.
2. Que los coeficientes no quepan en memoria. Los coeficientes codificados tienen que ser reducidos para ser adaptados al espacio disponible. Esto implica una compresión con pérdidas con la consiguiente pérdida de información.

Por lo tanto, podemos extraer las siguientes propiedades del sistema de codificación escalable:

- **Requiere una alta relación de compresión**: Cuanto mayor sea la relación de compresión del codificador escalable, mayor probabilidad habrá de que los coeficientes quepan en la memoria de bucle disponible. Esto generará una compresión con menos pérdidas. Recíprocamente, cuanto mayor sea la compresión menor es el tamaño de memoria necesario para las mismas pérdidas.
- **Requiere buena escalabilidad en la precisión de los datos**: Debido al tamaño fijo de la memoria de bucle es muy poco probable que los coeficientes codificados tengan todos cabida en su interior. El tamaño de los coeficientes codificados se puede reducir más truncando la información que va a ser almacenada, truncando su

precisión. En este caso, es importante truncar la información menos importante de los coeficientes DCT (los bits menos significativos).

- **Requiere facilidad de implementación:** Debido a que el codificador escalable va a ser implementado en *hardware*, es muy importante velar por la sencillez de su implementación. Desgraciadamente, las técnicas de codificación sin pérdidas con altas relaciones de compresión son a menudo difíciles de implementar en *hardware*.

Estas restricciones han llevado a Kleihorst y colaboradores [KVP00] a implementar una codificación zonal basada en planos de bits de los coeficientes de la DCT. El codificador convierte los coeficientes de los bloques DCT en cadenas de bits. Para lograr escalabilidad, los planos de bits son codificados de mayor a menor peso. En el caso de ser necesario truncar datos, sólo se perderá la información de los planos inferiores. La FIGURA 3.2 muestra cómo funciona el codificador escalable. Antes de codificar, todos los coeficientes AC son transformados a magnitudes sin signo, dado que el codificador ha sido diseñado para números positivos. La cadena de bits comienza con el coeficiente DC de la transformada, con la intención de evitar modificarlo, ya que el deterioro de este coeficiente conllevaría perdidas adicionales de calidad en el decodificador.

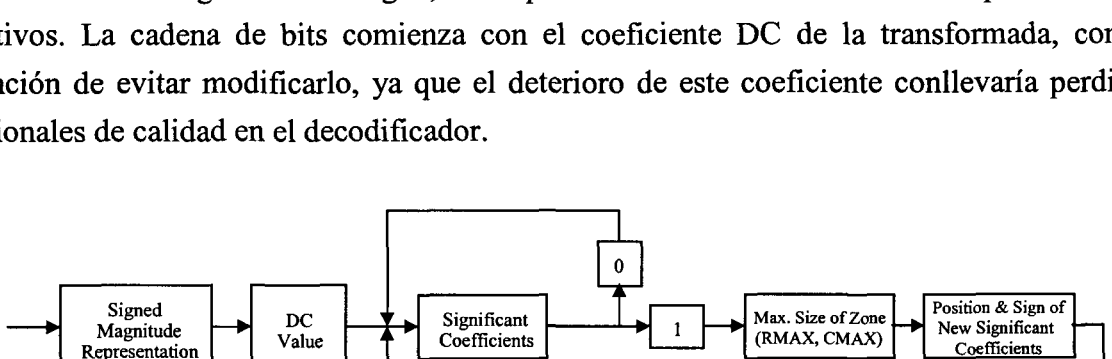


FIGURA 3.2. Funcionamiento del codificador escalable.

A continuación se codifican los coeficientes AC por planos de bits, empezando por el plano de bits de mayor orden. Mientras se realiza la codificación de cada plano de bits, se distingue entre coeficientes relevantes y coeficientes irrelevantes. Un coeficiente relevante es un coeficiente para el cual se ha codificado un uno en un plano de bit superior, mientras que un coeficiente irrelevante es un coeficiente para el cual no se ha codificado todavía ningún bit (debido a que eran ceros).

Inicialmente se marcan todos los coeficientes AC como irrelevantes. A continuación se codifican los planos de bits comenzando por el plano de bit de mayor orden. En primer lugar, para cada coeficiente significante se añade el bit en el plano de bit actual a la cadena de salida. Luego sigue un bit indicando si hay algún coeficiente irrelevante que se haya convertido en significante en el plano de bit actual. Si éste es el caso, entonces se codifica la posición de estos nuevos coeficientes relevantes seguida por el bit significativo para cada uno

de estos coeficientes. El bit del plano de bit actual para los nuevos coeficientes relevantes no es almacenado dado que es siempre uno. Este procedimiento se repite para todos los planos de bit.

La codificación de la posición de los nuevos coeficientes relevantes está basada en el hecho de que normalmente los coeficientes con mayor magnitud tienden a estar localizados cerca del coeficiente DC. Por lo tanto, primero se determina la pequeña zona rectangular alrededor del coeficiente DC que contiene los coeficientes relevantes nuevos. Se codifica el tamaño de esta zona, determinada por RMAX y CMAX como se ve en la FIGURA 3.3, en la cadena de bits (tres bits para cada una). A continuación, para cada coeficiente irrelevante se codifica un bit para indicar si ese coeficiente se ha transformado en relevante (1) o no (0), seguido del bit de signo en el caso de que se haya convertido en relevante. Dado que no se transmiten bits para los coeficientes que ya eran relevantes, ni para el coeficiente DC, la codificación por posición es muy eficiente. De esta partición del área en zonas deriva el nombre de codificación zonal dado a este esquema.

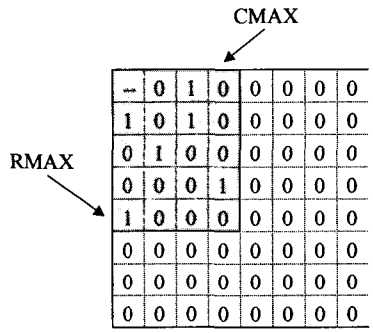


FIGURA 3.3. Ejemplo de plano de bit con CMAX=3 y RMAX=4.

La cadena binaria resultante puede ser truncada antes de ser almacenada en la memoria de bucle. Dado que la longitud de la cadena binaria varía entre un bloque DCT y otro, la memoria de bucle se organiza en dos zonas: una zona de punteros y otra zona de datos. Para cada bloque DCT se almacena un puntero en la zona de punteros, indicando el comienzo de la cadena binaria en la zona de datos. Se usa el puntero de siguiente bloque DCT para indicar el final de la cadena binaria del bloque DCT actual.

El esquema de compresión descrito proporciona una compresión sin pérdidas y ofrece una compresión media de 4 bits por coeficiente. Esto supone una reducción del doble y del triple en comparación con los píxeles sin comprimir (8 bits) y con los coeficientes DCT (12 bits), respectivamente. Se puede conseguir una compresión mayor que el doble truncando la cadena binaria, lo que supondría usar una media de 2 bits por coeficiente. Cabe destacar que cuando se trunca la cadena binaria se pierden parte de los planos de bit inferiores. Dado que la mayoría del ruido se concentra en los planos de bit inferiores, el truncado de la cadena binaria

supone disminuir este tipo de ruido, pero también información de la imagen. Con respecto al uso de memorias comprimidas en el dominio de la DCT en sistemas de restauración de imágenes, debemos ser cautelosos en su uso. Si el sistema de compresión es sin pérdidas y dispone de memorias grandes, obviamente no existirá ningún tipo de problema, pero si se necesita truncar los planos de bit inferiores, posiblemente se estén perdiendo también detalles de la imagen que a veces han sido costosamente recuperados gracias al uso de algoritmos complejos que muchas veces tienen como fin precisamente el ajuste de esos pequeños detalles o zonas de altas frecuencias. La ventaja de la codificación escalable zonal es que la decisión sobre la truncatura de los planos de bits es ajustable

La propiedad de escalabilidad proviene de escalar desde el plano de bits de mayor orden al plano de bits de menor orden. Esto asegura que al truncar para lograr mayor compresión sólo se pierdan los bits menos significativos (información del plano de bits de menor orden). La cadena de bits codificada comienza con el coeficiente DC seguidos de los coeficientes AC codificados en cada plano de bits.

El esquema de decodificación es simplemente la aplicación inversa del esquema de codificación antes visto. La única dificultad surge en el momento de decodificar una cadena binaria truncada. En este caso se pueden seguir varias estrategias: añadir ceros o añadir bits de forma aleatoria. Dado que el efecto de truncar es la eliminación de información de alta frecuencia y del ruido de los planos de bits inferiores, la presunción de que el ruido tiene una distribución aleatoria y de que no es relevante la alta frecuencia, conduciría a añadir bits aleatorios al final de la cadena binaria. No obstante, dado que el *hardware* para generar bits aleatorios es mucho más complejo que añadir ceros se ha optado por esta última solución.

3.2.3 Codificador escalable

En la FIGURA 3.4 se muestra un codificador escalable de vídeo, donde se han resaltado los bloques de compresión empotrados para la memoria de bucle. Dado que la memoria de bucle está organizada en bloques DCT los accesos a un macro-bloque (MB) desde memoria a una posición desplazada (FIGURA 3.5) por el estimador de movimiento requieren la lectura de 9 bloques DCT (en el peor de los casos) desde memoria. Esto significa 9 operaciones de decodificación escalables (DEC) y 9 operaciones de DCT inversas. El bloque para el cálculo de los desplazamientos (DISP) calcula el bloque desplazado en el dominio del tiempo basándose en los bloques DCT obtenidos de la memoria de bucle.

Se puede minimizar el número de bloques DCT traídos desde la memoria de bucle aprovechando la propiedad de localidad del bloque de referencia mediante el uso de memorias caché (de tamaño 4x4 bloques DCT), reduciendo por lo tanto el coste computacional extra provocado por las operaciones de decodificación escalables y de transformada inversa del coseno.

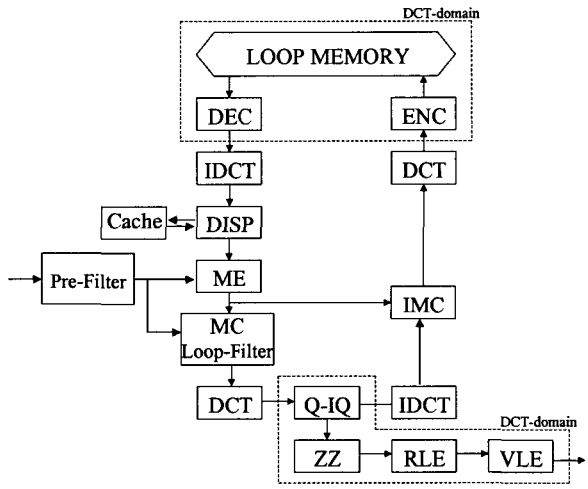


FIGURA 3.4. Diagrama de bloques de un codificador de vídeo híbrido modificado para usar memorias comprimidas en el dominio de la DCT.

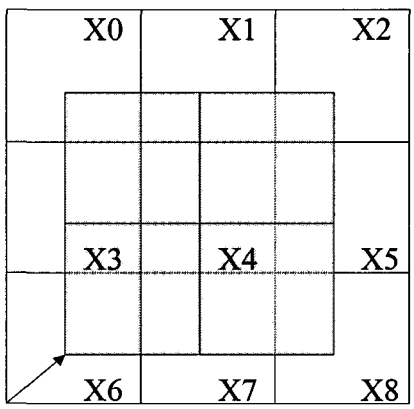


FIGURA 3.5. Lectura de un bloque DCT desplazado desde una memoria de bucle orientada a bloques DCT.

3.2.4 Estimación de movimiento

La estimación del movimiento es la operación encargada de determinar el mejor ajuste del macro-bloque actual con la imagen almacenada en la memoria de bucle. El algoritmo 3DRS usado es una modificación del descrito en [HB98], [HBH+93] y [WWV+97]. El algoritmo original está basado en un conjunto de cinco vectores candidatos cuidadosamente elegido en las proximidades del macro-bloque actual, tal y como se muestra en la FIGURA 3.6.

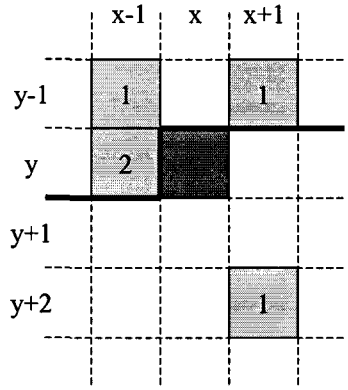


FIGURA 3.6. Vectores candidatos en el algoritmo 3DRS.

Si (x,y) son las coordenadas del macro-bloque actual los siguientes vectores serían usados como candidatos:

1. Vector de movimiento del macro-bloque situado en (x-1,y-1).
2. Vector de movimiento del macro-bloque situado en (x+1,y-1).
3. Vector de movimiento del macro-bloque situado en (x-1,y) con una actualización aleatoria en el rango (-1,1).
4. Vector de movimiento del macro-bloque situado en (x-1,y) con una actualización aleatoria en el rango (-6,6).
5. Vector de movimiento del macro-bloque situado en (x+1,y+2).

Los primeros cuatro candidatos son de la imagen actual, mientras que el último candidato proviene de la imagen anterior.

El algoritmo 3DRS original ha sido modificado de la siguiente forma:

- Se han añadido dos vectores adicionales al conjunto de candidatos, el vector cero y el vector en la posición (x,y), este último de la imagen anterior.
- Todos los vectores candidatos están expresados en precisión de medio píxel. Dado que la memoria de bucle está en el dominio de la DCT no hay coste adicional al recuperar un macro-bloque desplazado medio píxel en comparación a un macro-bloque desplazado una cantidad entera de píxel. Esto contrasta con los codificadores convencionales en el dominio espacial, donde los desplazamientos de medio píxel requerían normalmente mayor cantidad de accesos a memoria e interpolaciones adicionales.
- Se pueden especificar penalizaciones (positivas o negativas) adicionales a cada vector candidato para discriminarlos. Veremos que esta opción va ha resultar especialmente útil en súper-resolución para permitir disminuir la preponderancia del vector cero.

- Se realizan dos mejoras, consistentes en una nueva búsqueda sobre el vector que ha resultado ser mejor candidato, la primera con precisión de píxel y la segunda con precisión de medio píxel. Estas mejoras son necesarias para incrementar la convergencia del algoritmo 3DRS en el caso de bajas velocidades de *frame* (por ejemplo, 10 *frames* por segundo) [Ol98].
- Los vectores mejorados son almacenados en la memoria de vectores. Esto es ligeramente diferente al método seguido en [Ol98], donde el vector era almacenado en la memoria de vectores después de la mejora en precisión de píxel entero y antes de la mejora a medio píxel. Esto es debido a que no existe ninguna penalización adicional en realizar la estimación de movimiento completa con precisión de medio píxel, como ya se ha indicado.
- Los macro-bloques en los bordes de la imagen tienen menos macro-bloques alrededor. Con el objetivo de mantener un conjunto de siete vectores candidatos, los macro-bloques no existentes son aproximados a los macro-bloques existentes más próximos. Así por ejemplo, en el caso del borde izquierdo se usaría el vector de movimiento del macro-bloque (0,1) en lugar del (-1,1).
- Los vectores que apunten fuera de la imagen son recortados a la posición del borde más próximo.
- Después de determinar el mejor vector de movimiento para un macro-bloque, debe determinarse el tipo de codificación para este macro-bloque:
  1. Omitirlo.
  2. Modo *'intra'*, o de imagen inicial de sub-secuencia , cuadro no interpolado o predecido.
  3. Modo *'inter'* con vector de movimiento cero, correspondiente a una secuencia con predicción o interpolación.
  4. Modo *'inter'* con vector de movimiento distinto de cero
  5. Modo *'inter'* con cuatro vectores de movimiento 8x8

  No obstante, esta elección es exterior al bloque y no tiene impacto en el diseño del estimador de movimiento propiamente dicho. El estimador de movimiento siempre está activo, incluso si los vectores de movimiento resultantes no son usados debido al tipo de codificación usada. Esto se hace así para incrementar la convergencia del algoritmo 3DRS, y para permitir filtrado de ruido con compensación de movimiento temporal.

3.3 Arquitectura de la plataforma de codificación: de C_HEAP a *Picasso*

3.3.1 Particionado hardware/software

La implementación del codificador de vídeo usa un modelo arquitectural basado en C-HEAP (CPU-*controlled Heterogeneous Embedded Architectures for signal Processing*) [Lip97]. C-HEAP es una arquitectura de coprocesadores heterogéneos especializados, con dependencia jerárquica de un solo procesador central que despacha las tareas a ls unidades y funciones. Las tareas se sincronizan con la disponibilidad del espacio disponible en los *buffers* y con la disponibilidad de los datos y puede usar *buffers on-chip* para reducir el ancho de banda con la memoria externa. Los procesadores se escogen de tal forma que se consiga un buen compromiso entre flexibilidad y eficiencia. Así por ejemplo, las funciones cuyas características son bien conocidas y no están sujetas a cambio pueden ser implementadas de forma eficiente en *hardware* (con flexibilidad mínima). Por otra parte, las funciones que necesitan ser adaptadas al contexto en el que operan, deben ser implementadas con flexibilidad máxima (aunque con menores prestaciones). El modelo arquitectural de C-HEAP soporta tanto núcleos flexibles (procesos en procesador central) como núcleos altamente eficientes (procesos en coprocesadores especializados o "cores").

El protocolo de comunicaciones entre procesos de C-HEAP puede manejar cadenas de datos de longitud teóricamente infinita. El protocolo está basado en el modelo de redes de procesos [LP95], donde la función global se descompone en cierto número de procesos paralelos que se comunican a través de canales punto a punto, comportándose como memorias FIFO. Un proceso puede bloquearse debido tanto a la no-disponibilidad de los datos en el canal de entrada como a la no disponibilidad de espacio libre en el *buffer* de salida. La sincronización de los procesos se obtiene del estado de los canales de comunicación, FIFOs y se realiza según un mecanismo basado en 'testigos' o '*tokens*'. Cabe destacar que la cantidad de datos asociados a un *token* puede variar desde cero hasta una imagen completa de una secuencia de vídeo. Sin embargo, en el contexto de implementaciones de la arquitectura C-HEAP con memoria compartida no se requieren transferencias físicas de datos. En este caso sólo se necesitan las primitivas de sincronización con los punteros de los datos. Además, debido a factores de coste y prestaciones, no se realiza reasignación dinámica de memoria para los multiprocesadores. Esto significa que durante la inicialización todos los recursos de memoria quedan ya asignados para poder ser usados durante la ejecución de las operaciones (procesos).

La partición se ha realizado siguiendo los siguientes pasos:

- El primer paso es determinar las partes del algoritmo más adecuadas para que sean implementadas en *hardware*. Una descripción del algoritmo de codificación está dada por el diagrama de bloques del flujo de datos de la FIGURA 3.4. Los bloques que requieren mayores prestaciones y demandan menor flexibilidad, son: (I)DCT, (I)MC, (I)Q, parte del DISP, el ZZ, el RLE, el DEC y el ENC. El bloque ME se ha dividido en dos partes: evaluación de los vectores y selección de los vectores. Además de estos bloques se han implementado en *hardware* dos bloques adicionales: el generador de bordes y el empaquetador. El bloque de generación de bordes es el encargado de generar un borde artificial alrededor de la imagen mediante la simple copia de los píxeles de los márgenes. Este bloque es necesario para implementar dos opciones del estándar H.263. El empaquetador es el encargado de concatenar los bits de salida del VLE dado que su funcionalidad es compleja de implementar en *software*.
- El siguiente paso es trasladar cualquier flexibilidad requerida de los bloques *hardware* al *software* (por ejemplo, los parámetros dependientes del estándar de codificación). Un buen ejemplo es el recorte de los vectores. Dado que el tamaño máximo de los vectores de movimiento depende del estándar de codificación, el tamaño del vector máximo se determina por *software* y se pasa al *hardware* como un parámetro. Los vectores se recortan luego en *hardware* en base a este parámetro.
- El siguiente paso es agrupar estos bloques combinando los bloques con mayor relación entre sí en un co-procesador de mayor tamaño. Esto ha dado lugar a tres co-procesadores:
  1. El estimador de movimiento (DEC, DISP y evaluación de vectores del ME).
  2. El compresor (DEC, ENC, DISP, (I)MC, (I)DCT, (I)Q, ZZ, RLE y generador de bordes).
  3. El empaquetador.
- La descripción en código C se ha dividido en procesos, un proceso por co-procesador, y un proceso para el *software* que se ejecuta en un procesador ARM (*Advanced RISC Machine*) [Fur96].
- El modelo resultante se ha simulado extensamente para verificar que se alcanzan las prestaciones deseadas. Estas simulaciones muestran que implementar el VLE de los coeficientes en *software* (lo que es bueno desde el punto de vista de la flexibilidad, dado que cada estándar de codificación tiene diferentes tablas VLE) da como resultado unas prestaciones bajas. Por lo tanto, se ha adoptado la siguiente decisión: la VLE de los coeficientes tiene que ser realizada en *hardware*, y por su dependencia en el flujo de datos ha sido incorporada al coprocesador

compresor, con lo cual se han conseguido las prestaciones requeridas de poder codificar VGA a 30 *frames* por segundo. La inicialización de la parte *hardware* del VLE debe ser tan flexible como sea posible (por ejemplo descargando las tablas VLE en una memoria direccionable [KWS+99]). No obstante, la parte *hardware* de los coeficientes del VLE puede ser desactivada, para ser realizada en *software* si así se desea. En este sentido es posible realizar un VLE completamente diferente, a expensas de bajar las prestaciones.

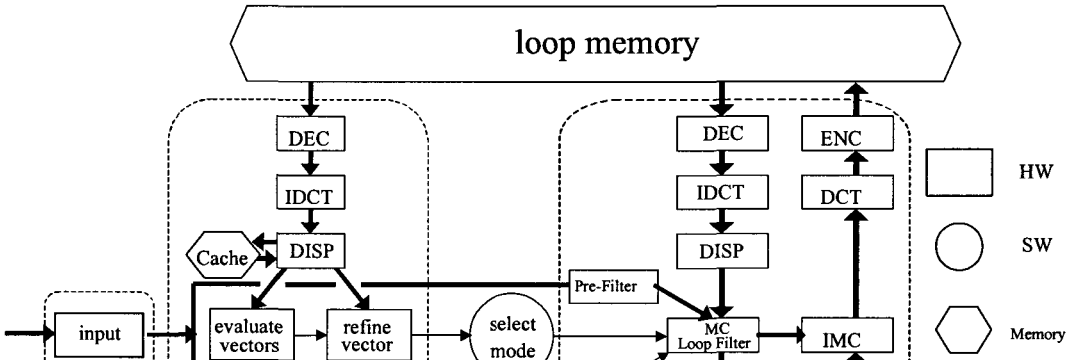


FIGURA 3.7. Partición *hardware/software* del codificador H.263.

El resultado final es la partición mostrada en la FIGURA 3.7, donde los bloques de *hardware* se han representado como rectángulos, las tareas de *software* como círculos, y con líneas más anchas se indica el flujo de datos.

3.3.2 Particionado de la ruta de datos: procesadores, memorias y buses

La arquitectura heterogénea resultante se muestra en la FIGURA 3.8. Además de los co-procesadores anteriormente mencionados, se ha añadido un co-procesador de entrada. El microprocesador central usado es un ARM-7. Debido al amplio ancho de banda de la memoria de bucle del algoritmo de codificación, ha sido necesario utilizar dos memorias. Una memoria para el microprocesador (código, datos, etc.), y la otra dedicada exclusivamente a

almacenar datos en formato de píxeles. Los vectores de movimiento son almacenados en la memoria el ARM.

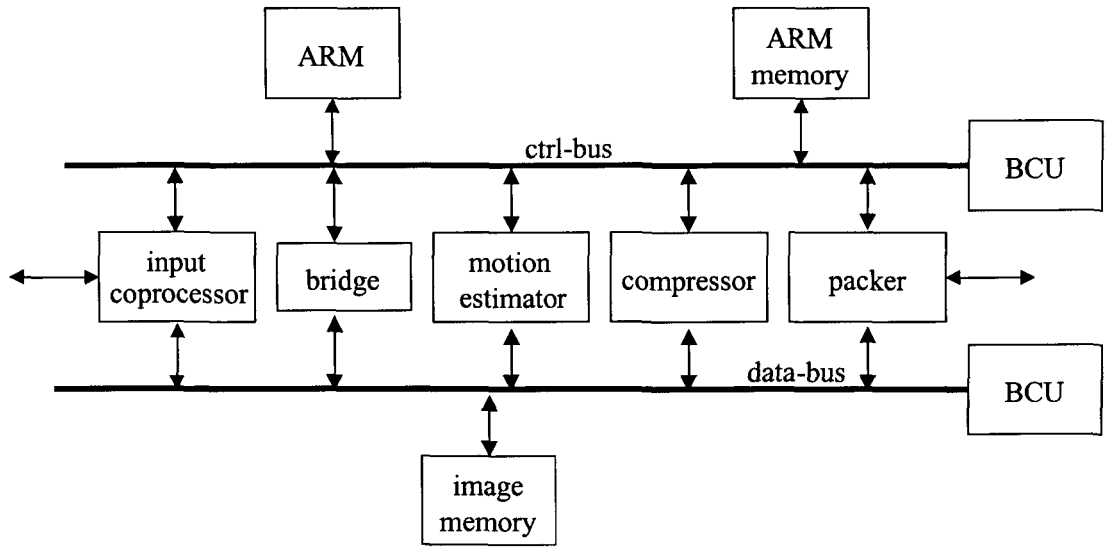


FIGURA 3.8. Arquitectura C-HEAP del codificador H.263.

Para acelerar las transferencias de datos se adoptó una arquitectura de dos buses, cada uno basado en el protocolo de bus AMBA, y dos unidades de control de bus (BCU, *Bus Control Unit*) necesarias para controlarlos. El bus superior 'ctrl-bus' se usa para la comunicación desde y hacia el ARM (datos de programa, variables, vectores de movimiento y protocolo de comunicación con los co-procesadores). La memoria ARM esta dedicada al sistema operativo, las aplicaciones *software* y las estructuras de datos. La memoria de imagen consta del *buffer* de entrada, memoria de bucle y buffer de salida. El ARM está configurado como el controlador por defecto de ese bus. El bus inferior 'data-bus' se usa para la comunicación desde y hacia la memoria de imágenes (*buffer* de entrada, memoria de bucle y buffer de salida). Debido a los grandes requerimientos de ancho de banda del estimador de movimiento, este co-procesador se ha definido como el controlador por defecto de ese bus. Además, se ha añadido un puente (*bridge*) entre ambos buses con la idea de permitir que el ARM acceda a la imagen.

Se han añadido dos buses adicionales a la arquitectura, uno conectado al co-procesador de entrada y el otro al empaquetador. Estos buses son los responsables de las conexiones al mundo exterior. Sólo se ha usado un reloj para toda la arquitectura.

3.3.3 Co-procesador de entrada

El co-procesador de entrada es el responsable de las comunicaciones con el subsistema de captura de imágenes (subsistemas relacionados con los sensores de imágenes) realizando la conversión de línea a franja (*stripe*) de píxeles. Se realizan lecturas de líneas de píxeles que son almacenadas en una pequeña memoria *buffer*. Después de leer 16 líneas se le pasa un puntero al *software* para indicarle donde puede encontrar la franja. Se usa este para comenzar el procesamiento de los macro-bloques de la franja.

Mientras se procesa la franja actual se va leyendo la siguiente, lo que requiere un tamaño del *buffer* de entrada igual a dos franjas. Dado que la entrada a los datos es progresiva este tamaño de *buffer* es suficiente para el procesador de entrada.

3.3.4 Estimador de movimiento

El co-procesador para la estimación del movimiento, ME, evalúa un conjunto de vectores candidatos recibidos desde el *software* y selecciona el mejor vector en precisiones de píxel y de ½ píxel. Cada vector candidato consiste en las coordenadas del vector un valor de penalización y el tamaño de penalización aleatoria. El co-procesador de ME también calcula el coste de codificar un macro-bloque en modo '*intra*' o en modo '*inter*'. La salida del co-procesador de ME consiste en un conjunto de vectores de movimiento, un conjunto de sumas de diferencias absolutas (SAD, *Sum of Absolute Diferences*) y de métricas para el modo '*intra*'. Esta información se usa en el *software* para determinar el modo de codificación del macro-bloque actual (saltarlo, codificación '*intra*', codificación '*inter*' con vector de movimiento cero, codificación '*inter*' con vector de movimiento, o codificación '*inter*' con cuatro vectores de movimiento de 8x8).

Una propiedad del algoritmo 3DRS es que usa el vector de movimiento del macro-bloque (x-1,y) como vector candidato (con una actualización aleatoria adicional) para el macro-bloque (x,y). Esto requiere que se haya completado la estimación de movimiento del macro-bloque anterior y que el vector de movimiento resultante haya sido transmitido al *software* antes de que éste pueda empezar a generar el conjunto de vectores candidatos para el siguiente macro-bloque. Esto implicaría que no se podrían usar técnicas de segmentación para acelerar los cálculos. El almacenamiento en *hardware* del vector de movimiento resultante, de forma local al coprocesador, ha eliminado esta dependencia. Tan solo la actualización es generada por el *software*, que es añadida por el *hardware* al vector almacenado.

3.3.5 Compresor

El co-procesador compresor (*compressor*) efectúa la codificación y decodificación de los macro-bloques filtrados y almacena los macro-bloques decodificados en la memoria de bucle. La salida del compresor consiste en códigos VLE de los coeficientes DCT del macro-bloque que se esté tratando (si la VLE de los coeficientes se realiza en *software*, entonces la salida consiste en una lista de valores RLE). Se utiliza una memoria direccionable por contenido (CAM, *Content, Addressable M*emory) para transformar los valores RLE en valores VLE [KWS+99]. El estimador de movimiento va dos macro-bloques por delante con respecto al compresor, dado que el compresor precisa (para la opción de modo de predicción en avance) de los vectores de movimiento de los macro-bloques situados en (x-1,y), (x,y), (x+1,y) y (x,y-1) para codificar el MB (x,y). Además, el co-procesador compresor va por delante del *software* en un macro-bloque, dado que el *software* necesita el resultado del compresor para generar los códigos VLE para las cabeceras y los vectores de movimiento. Esta temporización se expone en la sección 3.3.8 de planificación.

La presencia de una caché de tamaño 4x4 bloques DCT en el dominio espacial reduce el número de accesos a la memoria de bucle, y por lo tanto, reduce la cantidad de cálculos extra en las operaciones DEC e IDCT necesarias. La organización de la caché es asociativa de un solo nivel.

3.3.6 Empaquetador

El co-procesador 'empaquetador' (*packer*) concatena los códigos VLE generados por el compresor y por el *software* y los transfiere a la salida del codificador (para su almacenamiento o para su conexión a algún otro sistema). Debido al paralelismo entre el compresor y las tareas *software* es necesario multiplexar para entrelazar los códigos VLE de las cabeceras y los vectores de movimiento con los códigos VLE de los coeficientes. Esta multiplexación se expone en la sección de planificación de tareas (sección 3.3.8).

3.3.7 Memorias empotradas

La arquitectura *Picasso* contiene dos memorias empotradas: la memoria del ARM y la memoria de imagen.

3.3.7.1 Memoria del ARM

La memoria del ARM se conecta al ARM a través del bus 'ctrl-bus', y es usada para almacenar todos los datos relacionados con el ARM, como son:

- Sistema operativo y programa del ARM.
- Variables del ARM incluyendo los vectores de movimiento.
- Datos de comunicación para las comunicaciones C-HEAP con los co-procesadores.

El tamaño de esta memoria depende del tipo de aplicación final, y para aplicaciones estándares es aproximadamente 500 Kbits.

3.3.7.2 Memoria de imagen

La memoria de imagen se conecta a través del bus 'data-bus' a todos los co-procesadores, en indirectamente a través del puente al bus 'crtl-bus'. En su interior se almacenan los siguientes datos:

- El *buffer* de entrada consistente en dos franjas. Dado que el tamaño de la memoria para almacenar estas franjas es bastante grande (240 Kbits y 132 Kbits para VGA y CIF, respectivamente) se usará compresión empotrada en esta franja.
- La memoria de bucle que contiene las imágenes reconstruidas previa y actual. El tamaño de la imagen reconstruida (estimando 2 bits por píxel) es de 1008 Kbits y 360Kbits para VGA y CIF, respectivamente. Se tienen que almacenar dos imágenes: la anterior para la estimación de movimiento y la actual que esta siendo codificada. No obstante, dado que se codifican las imágenes de forma progresiva, es posible unir ambas imágenes como se muestra en la FIGURA 3.9. Desde que una zona de la imagen reconstruida anterior deja de ser utilizada, ésta es sobrescrita por la imagen actual. Esto requiere un solape de dos franjas. Sin embargo, con el objetivo de soportar la opción de vectores de movimiento sin restricción (UMV, Unrestricted Motion Vector) del estándar H.263 se necesita un solape de tres franjas. Esto arroja una memoria de 990 Kbits para VGA y de 346.5 Kbits para el formato CIF. Como ya se ha comentado, esta memoria está organizada en dos partes: una zona de punteros y una zona de datos. Ambas zonas están organizadas en bloques DCT.
- Cabe destacar que el *buffer* de salida no es parte de la arquitectura *Picasso*. Se utiliza normalmente un *buffer* de salida para ajustar cambios en la tasa binaria. No obstante, el tamaño de este *buffer* depende de la aplicación, y más en concreto del

ancho de banda del canal usado para transmitir el flujo (*stream*) de vídeo comprimido.

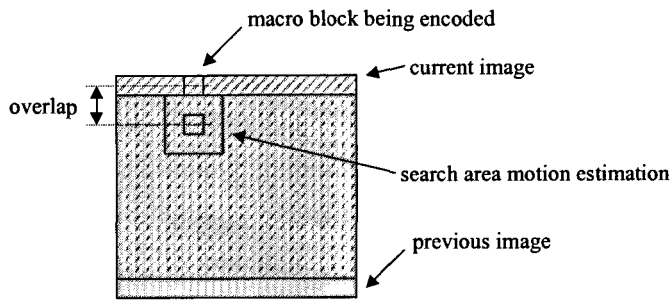


FIGURA 3.9. Combinación de la imagen reconstruida actual y anterior.

3.3.8 Software

Las tareas *software* ejecutadas sobre el procesador ARM consisten en:

- Acciones dependientes del control y la inicialización.
- Generación de los vectores de movimiento candidatos para su evaluación en el estimador de movimiento.
- Selección de modo de codificación del macro-bloque.
- Generación de los códigos VLE para las cabeceras y los vectores de movimiento.
- Control de la tasa binaria (*bit rate control*) de salida, acción sobre el cuantificador, y protocolos de despacho y comunicaciones con los co-procesadores.

La tarea más importante del ARM es realizar el protocolo de comunicación con los co-procesadores *hardware*. El aspecto más importante de este protocolo es que ha sido diseñado de tal forma que se pueden ejecutar en paralelo la mayor cantidad posible de tareas *hardware* y *software* como veremos a continuación.

3.3.9 Planificación y segmentación del flujo de datos

En la FIGURA 3.10 se muestra la segmentación realizada para el caso de una cadena de vídeo de tamaño SQCIF (8x6 macro-bloques por imagen).

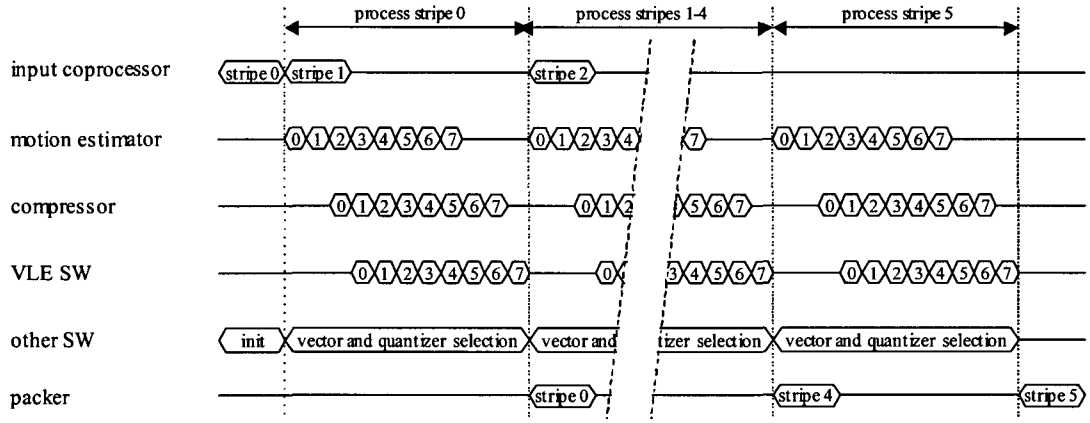


FIGURA 3.10. Planificación y segmentación de las tareas *hardware* y *software* del codificador H.263.

El co-procesador de entrada siempre va una franja por delante. La estimación de movimiento, la compresión y la VLE en *software* son realizadas en paralelo. Como ya se comentó anteriormente el estimador de movimiento va dos macro-bloques por delante del compresor quien a su vez va un macro-bloque por delante del VLE en *software*. Como se observa en la figura, el pipeline de estos tres bloques es rellenado y vaciado el principio y al final de cada franja respectivamente. Esto permite realizar los cálculos para el control de la tasa binaria antes de empezar una nueva franja. Además el empaquetador recoge los códigos VLE de varios macro-bloques en un *buffer*, y los escribe a la salida cada vez que el *buffer* está lleno o al final de una franja. Finalmente, el compresor aguarda hasta que se haya codificado completamente una imagen, antes de generar los bordes artificiales requeridos para los vectores que apuntan fuera de la imagen.

En la FIGURA 3.10 puede observarse que durante algunos periodos de tiempo el estimador de movimiento, el compresor y el VLE en *software* se están ejecutando en paralelo. Durante este tiempo el pipeline está lleno y los bloques *hardware/software* son usados de forma muy eficiente. Fuera de este periodo las operaciones se realizan de forma menos eficiente dado que los tres bloques no se están ejecutando en paralelo. No obstante, para tamaños de imágenes grande (como por ejemplo VGA) el número total de macro-bloques por franja de macro-bloque se incrementa, originando un aumento del periodo de eficiencia. Dado que la longitud del periodo ineficiente permanece inalterada la eficiencia global se ve incrementada.

3.4 Prestaciones del codificador *Picasso*

El uso de compresión empotrada para las memorias y en el dominio de la frecuencia ha permitido una solución de bajo coste en un solo chip. Se han evaluado las prestaciones de la compresión empotrada para las secuencias de vídeo 'SUZIE' y 'CARDPHONE', ambas en formato CIF a 10 fps y los resultados pueden observarse en la FIGURA 3.11 y la FIGURA 3.12 respectivamente para distintas tasas binarias por *frame*.

Se aprecia que la inclusión de compresión empotrada afecta de forma más notable a las secuencias con mayor cantidad de detalles y mayor cantidad de movimiento (como es CARPHONE). Sin embargo, las diferencias de calidad sólo son apreciables a altas tasas binarias, lo que tampoco es deseable en compresión de vídeo. También se observa que las diferencias entre ambas posibilidades (con y sin compresión) oscila entre 0.13 dB y 2.1 dB, lo que es perfectamente tolerable en este tipo de aplicaciones ya que las calidades son siempre superiores a los 35 decibelios. Además, al observar las secuencias no se aprecia ninguna degradación perceptual entre ambas (con y sin compresión de memoria). El factor de compresión empotrado es de 2.7 (factor de compresión medio de la memoria de bucle gracias al uso de compresión en el dominio de la DCT) y el factor de compresión de codificación varía entre 15 y 50 (factor de compresión de la cadena binaria de salida en relación con el tamaño de las imágenes de entrada).

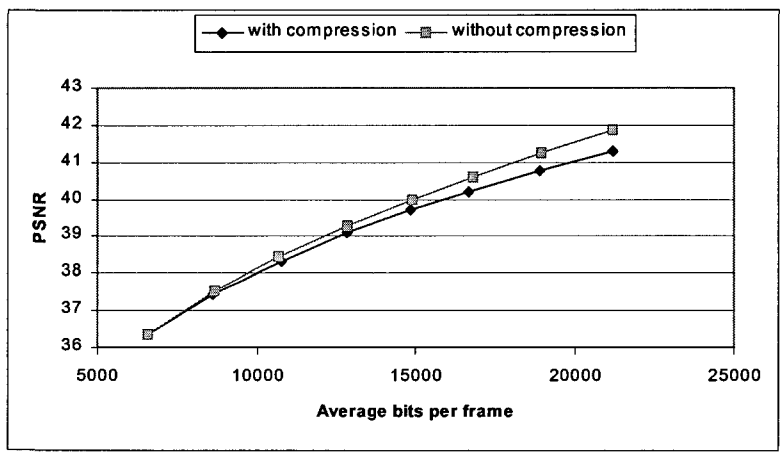


FIGURA 3.11. Curva de distorsión en función de la tasa binaria para la secuencia 'SUZIE' con y sin compresión empotrada.

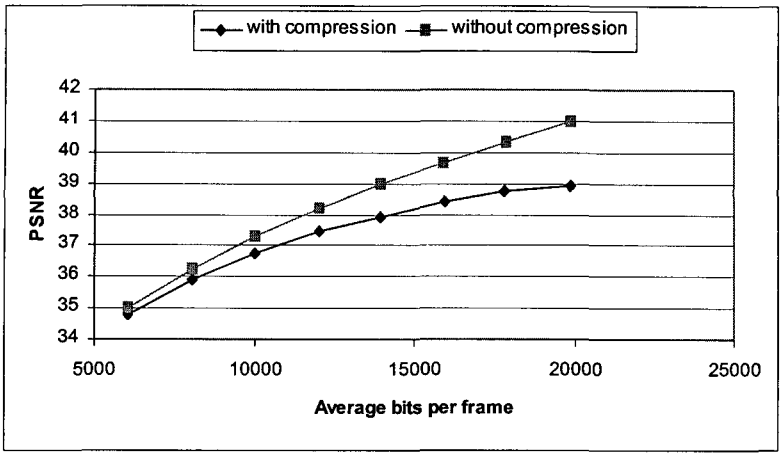


FIGURA 3.12. Curva de distorsión en función de la tasa binaria para la secuencia 'CARDPHONE' con y sin compresión empotrada.

La TABLA 3.1 proporciona algunos detalles del procesador ARM usado como CPU. Las estimaciones realizadas del factor de utilización de la CPU se obtienen de procesar secuencias de vídeo CIF a 30 fps.

Frecuencia de reloj	110 MHz
Área	0.8 mm^2
Potencia	0.3 mW/MHz
Factor de utilización	54 %

TABLA 3.1. Características del procesador ARM usado.

La TABLA 3.2 muestra las estimaciones de área y disipación de potencia para distintos componentes del codificador H.263 mapeados en tecnología CMOS de 0.18μm.

	Área (mm^2)	Potencia (mW)
CPU	0.8	18
Memoria CPU	3.1	15
Co-procesadores	6.0	50
Memoria de Imagen	3.7	2.5
Otros	1.0	10
Total	14.6	95.5

TABLA 3.2. Estimaciones de área y potencia del codificador para una tecnología CMOS 0.18.μm.

Las estimaciones del caso peor indican una implementación con 100mW de disipación de potencia y 15mm^2 de área. La FIGURA 3.13 compara la energía por macro-bloque de memoria para varias implementaciones que se encuentran en la literatura, escaladas a la misma tecnología de 0.18 μm. Las prestaciones de *Picasso* recogidas en la TABLA 3.2 y

FIGURA 3.13 han sido obtenidas a partir de estimaciones realizadas a varios niveles de abstracción de la aplicación [POR+00].

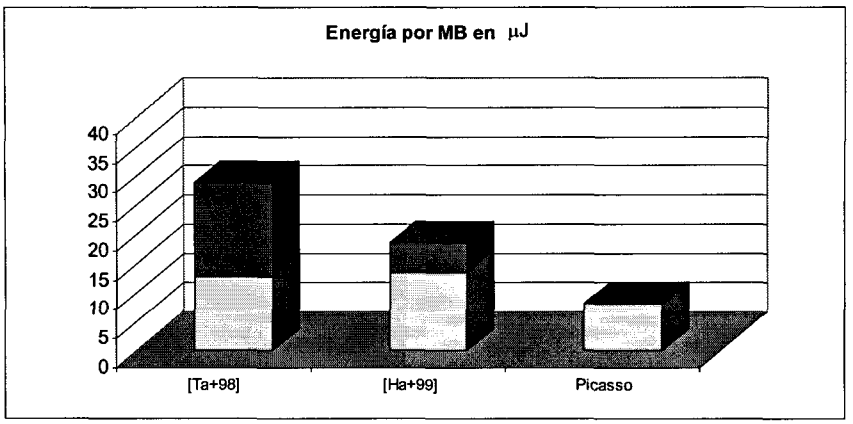


FIGURA 3.13. Comparación de la energía disipada por MB con respecto a las referencias [Ta+98] y [Ha+99]. La parte superior del diagrama de barras corresponde a la disipación de potencia fuera del chip debido al uso de memoria de bucle externa. Además, todas las implementaciones están normalizadas a la misma tecnología (0.18µm CMOS). La disipación de potencia de 60mW suministrada en [Ta+98] excluye la potencia disipada debido a la memoria de bucle externa.

3.5 Conclusiones

La plataforma *Picasso* supone un potente sistema de codificación híbrida de vídeo con las siguientes características:

- Implementación del núcleo básico de compresión para algoritmos basados en la transformada discreta del coseno.
- Sistema abierto a la implementación de nuevos estándares de compresión de vídeo e imagen estática.
- Particionado realizado de forma que se obtienen muy altas prestaciones.
- La compresión de las memorias en el dominio de la DCT permite usar sólo memorias empotradas *on-chip*. Al evitar el uso de memorias externas se consigue una importante disminución de la potencia disipada al tiempo que se minimizan los costes a un sólo circuito integrado.
- La metodología de diseño permite implementar y evaluar con sencillez nuevas aplicaciones sobre esta plataforma.

La plataforma *Picasso* es capaz de comprimir secuencias de vídeo siguiendo el estándar H.263 e implementando prácticamente todas de las opciones del estándar H.263+. Así mismo, está preparado para dar soporte al núcleo principal de compresión del estándar MPEG-4.

También puede comprimir imágenes estáticas en formato JPEG y se han desarrollado librerías que permiten su funcionamiento multi-hilo, lo que permite ejecutar varias de las aplicaciones anteriormente citadas al mismo tiempo siguiendo un esquema de compartición de recursos en el dominio del tiempo. Finalmente, como veremos en los próximos capítulos, una modificación adicional de *Picasso* permite mantener las prestaciones para las aplicaciones descritas en este párrafo y además ejecutar las nuevas aplicaciones de súper-resolución.

Capítulo 4

El pensamiento lógico puro no pude brindarnos ningún conocimiento del mundo empírico; todo conocimiento de la realidad empieza en la experiencia y termina en ella.

Albert Einstein (1879-1955).
Físico alemán. Premio Nóbel de física en 1921.

Es mucho más fácil presentar de modo atractivo la sabiduría destilada durante siglos de interrogación paciente y colectiva sobre la naturaleza que detallar el complicado aparato de destilación. El método, aunque sea indigesto y espeso, es mucho más importante que los descubrimientos de la ciencia.

Carl Sagan (1934-1996)
Escritor y astrónomo estadounidense.

Algoritmos de súper-resolución iterativos sobre la plataforma *Picasso*

4.1 Introducción

En este capítulo y en el siguiente se mostrarán los diferentes algoritmos de súper-resolución desarrollados sobre la plataforma de compresión híbrida de vídeo *Picasso* y la calidad de imagen que se logra con cada uno de ellos así como sus entornos de aplicación. Se mostrarán también diversos parámetros que caracterizan a este tipo de algoritmos y cómo han influido estos datos en la toma de decisiones a la hora de realizar las nuevas versiones de los algoritmos de súper-resolución. Los algoritmos se encuentran divididos en tres grandes apartados:

- *Algoritmos de súper-resolución iterativos*. Se engloban aquí los primeros algoritmos desarrollados en base a procesos iterativos. Aunque su funcionamiento reporta imágenes de buena calidad, en determinadas circunstancias esta calidad puede verse drásticamente disminuida. En todo caso, su funcionamiento en tiempo real con los recursos disponibles es imposible. Tienen el problema adicional de que entregan una imagen de súper-resolución por cada cuatro imágenes de entrada de baja resolución, por lo que su aplicación se restringe básicamente a imágenes estáticas. Estos algoritmos se trataran en este capítulo.
- *Algoritmos de súper-resolución no iterativos para imagen estática.* Este grupo de algoritmos consigue vencer el inconveniente de usar procesos iterativos, siendo mucho más adecuado para trabajar en tiempo real. Además, una de las versiones permite combinar un número variable de imágenes de entrada para conseguir imágenes de súper-resolución de mayor calidad. Este tipo de algoritmos supone una muy buena opción para cámaras digitales en las que en el momento de tomar una fotografía, la cámara realmente tomase varias desplazadas aprovechando el temblor de la mano para combinarlas en una fotografía de mucha mayor calidad que la que podría ofrecer el sensor. Sin embargo, el alto uso de memoria requerido sigue haciéndolos inviables para su uso en aplicaciones de vídeo. Estos algoritmos se desarrollarán en el capítulo 5.
- *Algoritmos de súper-resolución no iterativos para vídeo*. Los algoritmos aquí incluidos son ya aptos para vídeo, ya que al ser la última imagen adquirida la referencia de *movimiento*, el algoritmo sigue perfectamente y sin ningún retraso la secuencia de entrada. La primera versión de este grupo adolece no obstante del mismo problema que las versiones para imágenes estáticas y es que no entrega el mismo número de imágenes de salida que el que recibe a su entrada. Si combinamos N *frames* de entrada por cada *frame* de salida, quiere decir que si a la entrada tenemos M *frames*, a la salida tendríamos M/N *frames*. Por ejemplo, combinado 4 *frames*, una película de 40 minutos quedaría reducida a tan solo 10 minutos. La última versión usa un método recursivo que permite reutilizar los últimos resultados de súper-resolución y combinarlos con las nuevas imágenes de entrada a medida que estas vayan entrando. Además incorpora un sistema para la toma de decisiones que permite hacer el sistema muy robusto frente al ruido, frente a movimientos rápidos, frente a oclusiones totales o parciales y frente a cambios de contexto. Debido a su naturaleza realimentada este último algoritmo utiliza muy poca memoria lo que lo convierte en un firme candidato a ser implementado sobre la plataforma *Picasso*. Estos algoritmos se desarrollarán también en el capítulo 5.

Con el objeto de poder entender mejor los resultados y el proceso de diseño se ha incluido también un apartado introductorio sobre el método de preparación de las secuencias de prueba y la forma de contrastar la calidad de las imágenes obtenidas.

4.2 Sistemas de experimentación y prueba

Con la intención de poder contrastar en todo momento la calidad de las imágenes obtenidas se ha desarrollado un sistema de experimentación basado en la generación de imágenes sintéticas de baja resolución con las que serán alimentados los diferentes algoritmos de súper-resolución. Estas imágenes de baja resolución se obtienen a partir de imágenes de prueba de mayor resolución mediante la aplicación de desplazamientos controlados y posteriormente de diezmado o sub-muestreo. Los desplazamientos pretenden simular el movimiento que experimentaría el sistema de adquisición de imágenes en condiciones reales, y el diezmado persigue emular la adquisición de las imágenes usando un sensor de inferior calidad y un sistema de lentes que permitiese la presencia de *aliasing* en la secuencia de entrada. Los desplazamientos producidos pueden ser con precisión de ½ píxel o de ¼ de píxel, en función de la precisión que el estimador de movimiento pueda resolver. En dicho sentido, las primeras versiones usan desplazamientos de ½ píxel y las últimas desplazamientos de ¼ de píxel. Otro aspecto importante ha sido la obtención de un modelo de los desplazamientos. En este sentido se han habilitado las siguientes posibilidades:

a. *Desplazamientos establecidos manualmente por el experimentador.* Ésta es una forma de analizar meticulosamente el correcto funcionamiento del estimador de movimiento y de controlar cómo afectan los desplazamientos a la calidad de las imágenes obtenidas.

b. *Desplazamientos aleatorios.* Este tipo de desplazamientos tienen como objetivo determinar el comportamiento del sistema frente a desplazamientos desconocidos, incorrelados entre sí y erráticos. No es el tipo de desplazamientos que se suelen encontrar en los sistemas de adquisición reales pero refleja un caso extremo de funcionamiento del sistema.

c. *Desplazamientos aleatorios acotados de media cero.* Este pretende ser un modelo del tipo de movimiento que se generaría en una videocámara sostenida manualmente por un operario. Normalmente el ser humano intentará mantener la videocámara centrada sobre el objetivo a pesar del movimiento involuntario producido en el intento de sostener la cámara a mano sin el uso de un trípode o

algún otro método de sujeción. Este intento de mantener el objetivo centrado provocará desplazamientos relativos con media cero.

Para la generación de las imágenes de prueba podemos seguir cualquiera de los dos esquemas propuestos en la FIGURA 4.1. El esquema de la FIGURA 4.1 (a) sitúa primero el desplazador y luego el diezmador, mientras que el esquema de la FIGURA 4.1 (b) lo hace a la inversa. En nuestro caso se ha preferido usar el primer esquema ya que de esa forma podemos obtener con mayor facilidad desplazamientos sub-píxel sin tener que recurrir a interpoladores en el diezmador. Así en el primer esquema se introducen los desplazamientos *dx* y *dy* en unidades de píxel y se obtienen las fracciones gracias al diezmado.

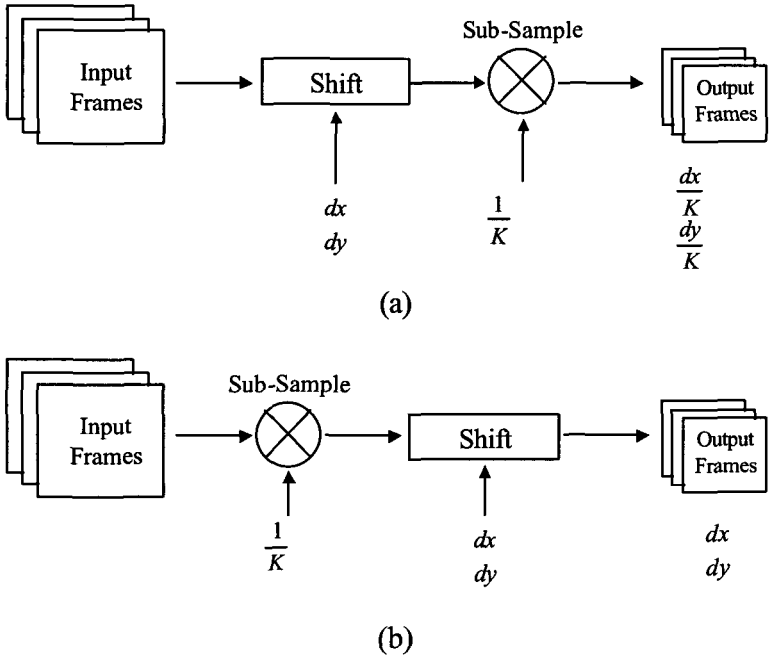


FIGURA 4.1. Esquemas para la generación de secuencias de imágenes de prueba: (a) situando primero el desplazador y (b) situando primero el diezmador.

Inicialmente se ha seguido el esquema de la FIGURA 4.1 (a) con K=2 y valores de desplazamiento generados manualmente que cubren las cuatro posiciones de píxel para facilitar al máximo la reconstrucción de la imagen de súper-resolución al estar presentes todos los datos de entrada. Los valores de los desplazamientos generados son los recogidos en la TABLA 4.1 referidos tal y como muestra la FIGURA 4.2. A este conjunto de vectores que cubren todos los píxeles de una celda base de 2×2 píxeles les llamaremos vectores canónicos.

dx	dy	Posición respecto a una celda base
0	0	Píxel superior izquierdo
1	0	Píxel superior derecho
0	1	Píxel inferior izquierdo
1	1	Píxel inferior derecho

TABLA 4.1. Desplazamientos generados a nivel de píxel para cubrir todos los píxeles de una celda base de 2×2 píxeles o vectores canónicos.

El desplazamiento de un píxel en la imagen de alta resolución conlleva el desplazamiento de medio píxel en la imagen de baja resolución después del diezmado, que es la precisión de la que inicialmente disponía el estimador de movimiento.

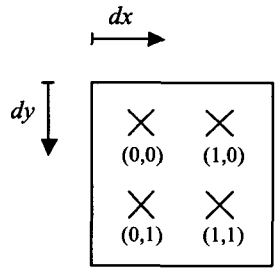


FIGURA 4.2. Celda base de 2×2 píxeles.

El proceso de diezmado ha consistido en la selección alternativa de cada uno de los cuatro píxeles de la celda base, generando una imagen de salida de baja resolución conteniendo *aliasing* espacial por cada uno de los píxeles de la celda base.

4.2.1 Sistema de generación de secuencias de prueba para los algoritmos iterativos

Así pues, los algoritmos iterativos diseñados inicialmente usan el esquema más detallado que se muestra en la FIGURA 4.3 para la generación de imágenes de prueba. Cada imagen de alta resolución de tamaño M×N se desplaza según el conjunto de vectores canónicos y cada una de las imágenes desplazadas se diezma para obtener un conjunto de cuatro imágenes de baja resolución con *aliasing* de tamaño mitad que la imagen original. Este conjunto de imágenes de baja resolución se introduce en el algoritmo de súper-resolución que debe obtener una imagen de salida de alta resolución (tamaño doble que el del conjunto de entrada, es decir M×N) de calidad similar a la imagen original.

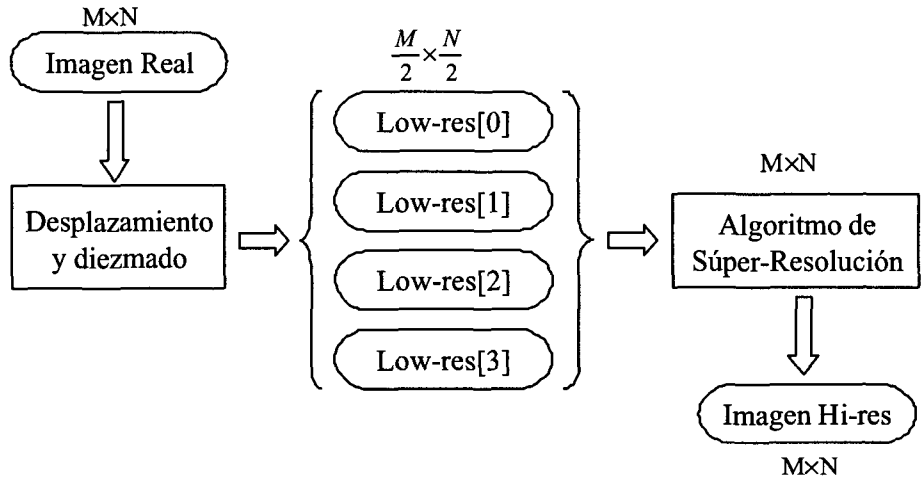


FIGURA 4.3. Esquema seguido en los algoritmos iterativos para la generación de imágenes de prueba.

La figura de mérito que nos permite medir el mejor o peor funcionamiento del algoritmo es la mayor o menor similitud entre la imagen resultante y la imagen original, llamada aquí imagen real. Esto se refleja en la FIGURA 4.4 a nivel de píxel. La imagen real o de referencia se desplaza y diezma según se ha visto anteriormente, produciendo cuatro imágenes de baja resolución al aplicar el conjunto de vectores canónicos. Por ejemplo, al aplicar el vector de desplazamiento (0,0) y posteriormente diezmar, la imagen de baja resolución correspondiente contiene todos los píxeles representados en la FIGURA 4.4 como rombos. El vector (1,0) da como resultado una imagen de baja resolución formada por los píxeles en forma de triangulo, el vector (0,1) da como resultado la imagen de baja resolución formada por los píxeles en forma de hexágono y finalmente el vector (1,1) da la imagen de baja resolución formada por los píxeles en forma de cruz.

Como se aprecia en la FIGURA 4.4 (a) el algoritmo de súper-resolución da una imagen de alta resolución formada por los píxeles $x_{1,2,3,4}$ en la celda básica, que idealmente y en el mejor de los casos deberían ser igual a los de la celda básica de la imagen real. La FIGURA 4.4 (b) muestra el proceso de ubicación de los píxeles a partir de una celda básica en una imagen usando el conjunto de vectores descritos.

Este método de generación de imágenes de prueba presenta un grave problema en los bordes de la imagen, que provoca una importante caída de la relación señal-ruido (SNR, *Signal to Noise Ratio*). Dependiendo del vector generado, es casi imposible evitar la perdida de datos en los bordes de la imagen al desplazar la imagen de referencia. Un desplazamiento

usando el vector (-2,5) de píxeles de alta resolución, provoca la pérdida de datos en los bordes superior y derecho, como se aprecia en la FIGURA 4.5, donde se ha exagerado el efecto para una mayor claridad. Al desplazar a la izquierda y abajo el autorretrato de *Picasso*, la única solución para evitar saltos bruscos es la de replicar los bordes de la imagen. Estos datos replicados van a entorpecer el proceso de reconstrucción de los bordes de la imagen original. Como medida inicial, la primera solución ha sido no tener en cuenta los bordes en el cálculo de la relación señal-ruido y del coeficiente de correlación espectral. El tamaño de los bordes que no participan en el cálculo de las figuras de mérito es de un macro-bloque, ya que los vectores de movimiento generados estan acotados a un valor inferior.

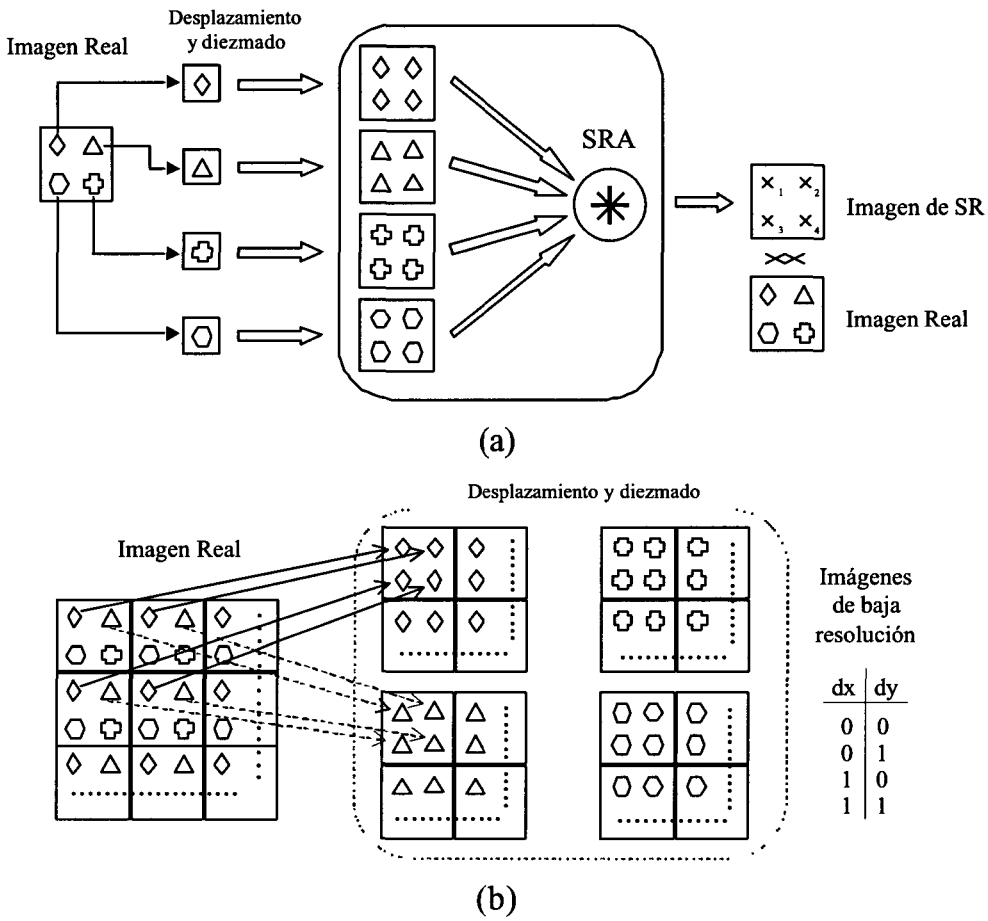


FIGURA 4.4. Esquema seguido en los algoritmos iterativos para la generación de imágenes de prueba a nivel de píxel.

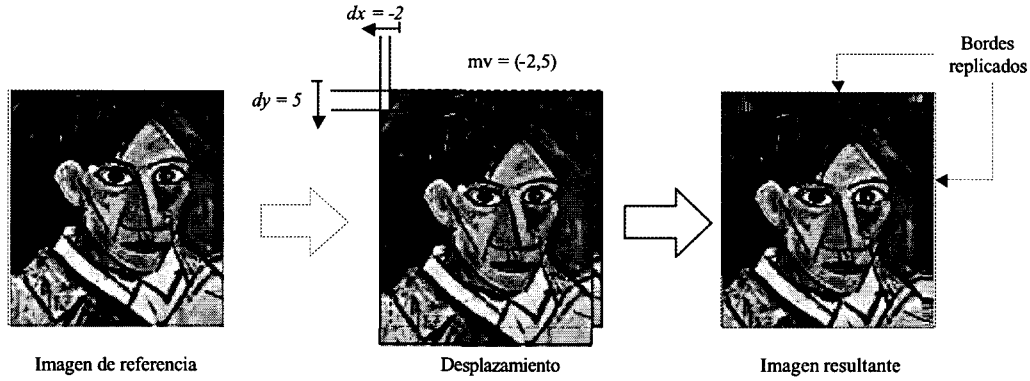


FIGURA 4.5. Efecto del desplazamiento en los bordes de la imagen.

4.2.2Sistema de generación de secuencias de prueba para los algoritmos no iterativos

Sin embargo, la mejor solución para el problema de los bordes es alterar el proceso de generación de imágenes de prueba. Es obvio que la solución de replicar los bordes no modela en absoluto el proceso real de adquisición de imágenes. En efecto, si un sistema real de adquisición de imágenes se mueve recogerá nueva información del mundo real que es, a este nivel, continuo. Por ello lo que se ha hecho es partir de imágenes de mayor tamaño, desplazarlas como ya se ha visto y luego recortarlas para quedarnos sólo con la parte central. De esta forma se evita el indeseado efecto de borde. La FIGURA 4.6 muestra el esquema modificado para la generación de imágenes de prueba. A diferencia del sistema anterior que engloba todo el proceso en un solo programa, el nuevo sistema es totalmente modular y esta formado por cinco módulos para la transformación de las imágenes. Estos son: `make_sequence`, `move`, `crop`, `subsample` y `downsample` y se muestran como rectángulos sombreados. El módulo `make_sequence` selecciona los *frames* de la secuencia de entrada que van a formar parte de la secuencia de prueba. Es necesario incorporar un módulo de este tipo, ya que algunas de las imágenes se obtienen usando una videocámara en los laboratorios de Philips Research y no se dispone de ninguna herramienta que trate las imágenes en crudo (sin comprimir) y que use el formato YCbCr para seleccionar las secuencias adecuadas para los experimentos. Además, al grabar imágenes de tamaño muy grande, en ocasiones el sistema se satura (especialmente el *bus*) dando como resultado un *frame_skip* que debe eliminarse luego con esta utilidad.

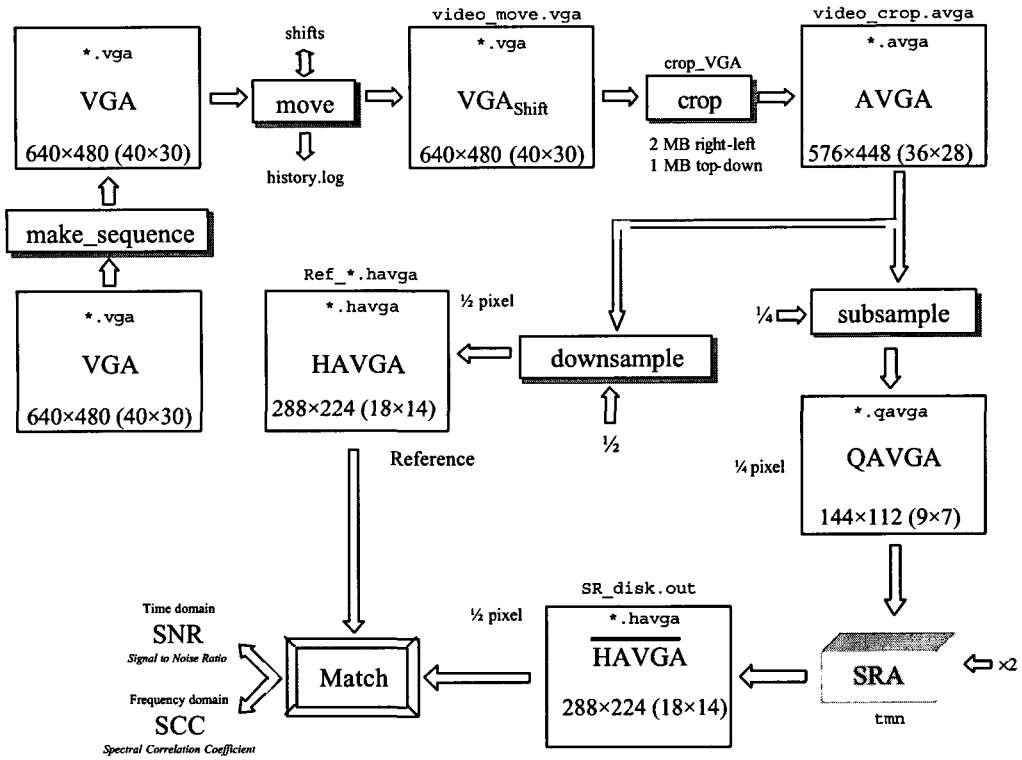


FIGURA 4.6. Esquema modificado para la generación de imágenes de prueba.

El módulo `move` desplaza cada fotograma de entrada, bien usando los vectores proporcionados en el fichero de texto `shifts`, o bien generándolos él de forma aleatoria (con o sin media cero). A su vez, genera un fichero histórico con los desplazamientos aplicados realmente.

Como ya se comentó, interesa usar un formato de imagen lo bastante grande como para luego recortarlo y evitar así el efecto de borde. Los formatos más comúnmente utilizados son los recogidos en la TABLA 4.2. De entre estos se ha optado por el formato VGA ya que es lo suficientemente grande para nuestros propósitos y es el mayor formato de imagen que la cámara es capaz de entregar.

	SQCIF	QCIF	CIF	VGA	4CIF	16CIF
Píxeles	128×96 (=12288)	176×144 (=25344)	352×288 (=101376)	640×480 (=307200)	704×576 (=405504)	1408×1152 (=1622016)
Macro Bloques	8×6 (=48)	11×9 (=99)	22×18 (=396)	40×30 (=1200)	44×36 (=1584)	88×72 (=6336)

TABLA 4.2. Tamaños de imágenes más usados y su equivalencia en macro-bloques.

No obstante, el tamaño VGA tiene un importante problema, puesto que si bien puede ser dividido por dos conservando un número entero de macro-bloques (640÷2=320 píxeles, es decir 20 macro-bloques y 480÷2=240 píxeles, es decir 15 macro-bloques) , no puede ser dividido por cuatro manteniendo un número entero de macro-bloques (640÷4=160 píxeles, es decir 10 macro-bloques y 480÷4=120 píxeles, es decir 7.5 macro-bloques). Este problema se ha solventado creando un nuevo formato de imagen bautizado como AVGA (*Adapted VGA*) de tamaño (576×448). Este nuevo formato sí puede ser dividido por 2 y por 4 conservando un número entero de macro-bloques, tal y como se muestra en la TABLA 4.3.

	VGA	AVGA	HAVGA	QAVGA
Píxeles	640×480 (=307200)	576×448 (=258048)	288×224 (=64512)	144×112 (=16128)
Macro Bloques	40×30 (=1200)	36×28 (=1008)	18×14 (=252)	9×7 (=63)

TABLA 4.3. Nuevos tamaños de imágenes basados en el formato VGA.

Así pues, el procedimiento seguido consiste primero en desplazar la imagen VGA sin preocuparnos por lo que pueda ocurrir en los bordes, para a continuación recortarle los bordes y dejarla en el tamaño AVGA. Esta nueva imagen ya no está afectada por el efecto de borde y además es posible dividirla por dos y por cuatro. El requerimiento de poder dividir la imagen por cuatro viene impuesto por la necesidad de poder tener imágenes con desplazamientos de ¼ de píxel en baja resolución. Es decir, desplazando la imagen cantidades de píxel en el tamaño VGA, luego tendremos desplazamientos de ¼ de píxel en el tamaño QAVGA. El módulo encargado de recortar los bordes de la imagen y adaptar su tamaño a AVGA es el `crop`. Este módulo elimina 2 macro-bloques de los bordes derecho e izquierdo de la imagen (pasando de 40 a 36 macro-bloques de anchura) y 1 macro-bloque de los bordes superior e inferior (pasando de 30 a 28 macro-bloques de altura).

A partir de la secuencia de vídeo ya desplazada y adaptada en tamaño se realiza un diezmado de factor 4 con frecuencia de muestreo inferior a la frecuencia de Nyquist para producir una nueva secuencia de video con *aliasing* de tamaño QAVGA. Esta secuencia es la entrada al algoritmo de súper-resolución y se genera a partir del módulo `subsample`.

También a partir de la secuencia de tamaño AVGA se realizado un diezmado de factor 2, pero esta vez con una frecuencia de muestreo que cumpla en criterio de muestreo de Nyquist para evitar *aliasing*. Esta nueva secuencia de video de tamaño HAVGA es la

secuencia de referencia, ya que el algoritmo de súper-resolución genera imágenes de tamaño doble a las de entrada -es decir de QAVGA generará HAVGA- que no pueden ser comparadas con las originales de tamaño AVGA. El proceso de diezmado de factor 2 para obtener la secuencia de referencia se lleva a cabo por el módulo `downsample`.

El algoritmo de súper-resolución (SRA) entrega como resultado una secuencia reconstruida de tamaño HAVGA, que debe compararse con la secuencia de referencia para obtener los diferentes parámetros de calidad que nos permitan evaluar el funcionamiento del algoritmo. El propio algoritmo realiza una comparativa con la secuencia de referencia en el dominio del tiempo, dándonos la relación señal al ruido entre cada imagen y su correspondiente en la secuencia de referencia. Sin embargo, para realizar el análisis en el dominio de la frecuencia se ha optado por usar el paquete matemático MATLAB. Para ello se ha desarrollado una serie de funciones que obtiene diferentes figuras de mérito de forma automática. Estas figuras de mérito son: la relación señal-ruido, el coeficiente de correlación espectral y el coeficiente de correlación espectral enventanado.

4.3 Algoritmos de súper-resolución iterativos

Este trabajo se basa inicialmente en el algoritmo de súper-resolución iterativo propuesto por Marc op de Beeck y Richard Kleihorst [Bee97] y [BK99] en los laboratorios centrales de investigación de Philips en Eindhoven. Este algoritmo no usa todavía los recursos ofrecidos por un codificador híbrido de vídeo, ni es capaz de trabajar en tiempo real, pero se ha tomado como punto de partida para su implementación en el codificador *Picasso*, para a partir de esta implementación estudiar sus características y realizar las modificaciones oportunas hasta llegar a la implementación final capaz de tratar secuencias de vídeo en tiempo real. Aunque los algoritmos iterativos son poco aptos para trabajar en tiempo real, tienen la ventaja de ser muy robustos frente al ruido y de dar buenos resultados cuando se conocen de forma imprecisa los desplazamientos entre las imágenes.

4.3.1El algoritmo de súper-resolución iterativo básico

El primer algoritmo de súper-resolución implementado intenta seguir en lo posible la versión original basándose en el principio de que los únicos datos disponibles son varios conjuntos de secuencias de imágenes sub-muestreadas. En la primera iteración, las posiciones de muestreo de los conjuntos de imágenes son inciertas. Es más, debido a la presencia de

aliasing en los conjuntos de imágenes de baja resolución, es previsible que se cometan errores al estimar los desplazamientos entre imágenes, dado que las técnicas de correlación por bloques no funcionan correctamente ante la presencia de imágenes con *aliasing*. Partiendo de secuencias de datos ruidosas, el método es capaz de iterar hasta lograr un resultado interpolado de mayor resolución que se adecua de forma óptima a todas las imágenes de baja resolución presentes.

El proceso iterativo consta de los siguientes pasos:

1. Determinar los desplazamientos relativos del conjunto de imágenes con respecto a una imagen de referencia. Debido a el *aliasing*, estos resultados pueden no ser precisos, y tendrán que ser optimizados en pasos posteriores.
2. Realizar una interpolación lineal de las imágenes con *aliasing* por separado, creando un conjunto de imágenes sobre una rejilla de alta resolución. Frente a la curva de interpolación exacta, deben esperarse grandes desviaciones. No obstante, este procedimiento asegura que el resultado de la primera iteración será de baja resolución pero liberado de gran cantidad de ruido. En iteraciones sucesivas se irá añadiendo información complementaria de alta resolución.
3. Sumar todas las imágenes interpoladas. Esta suma asegura una mayor reducción del ruido, mientras que el resultado de la primera iteración se desvía del conjunto original de imágenes en casi todas las posiciones de muestreo.
4. Re-alinear las imágenes de baja resolución con la interpolación recién creada. Dado que esta aproximación de alta resolución ya no contiene prácticamente *aliasing*, se pueden usar de nuevo técnicas estándares para generar vectores de desplazamiento sub-píxel.
5. Determinar por sub-muestreo la diferencia entre la estimación de alta resolución y las diferentes imágenes de baja resolución. Sumar todas las diferencias y realimentar esta suma como una actualización a la aproximación de alta resolución.
6. Iterar los pasos 4 y 5 hasta lograr convergencia.

Si se dispone de conjuntos de datos redundantes, este esquema iterativo se adecua muy bien a técnicas de reducción de ruido, dado que se puede incorporar con facilidad información estadística en el bucle de realimentación. En el caso de que sólo se disponga de datos suficientes, el proceso de iteración puede ser detenido antes de la convergencia total, dando lugar a una versión de menor calidad que es óptimamente compatible con el conjunto de datos disponible.

4.3.2Base teórica del algoritmo de súper-resolución iterativo básico

Comenzaremos realizando una descripción analítica del proceso, para lo que se usará la siguiente simbología:

- x, y : coordenadas de baja resolución.
- $\bar{x}, \bar{y}$: coordenadas de alta resolución.
- p: número de imágenes de baja resolución a combinar.
- $\Delta\delta_l(x,y)_{(fr2ref)}$: Desplazamiento horizontal del píxel x,y en el instante t_l del *frame l* respecto de la referencia.
- $\Delta\delta_l(x,y)_{(ref2fr)}$: Desplazamiento horizontal del píxel x,y en el instante t_l de la referencia respecto del *frame l.*
- $\Delta\lambda_l(x,y)_{(fr2ref)}$: Desplazamiento vertical del píxel x,y en el instante t_l del *frame l* respecto de la referencia.
- $\Delta\lambda_l(x,y)_{(ref2fr)}$: Desplazamiento vertical del píxel x,y en el instante t_l de la referencia respecto del *frame l.*
- Usaremos el superíndice $^{(n)} / n \in \mathbb{N}$ para indicar que se refieren a datos de la *n-sima* iteración.

Si llamamos $f(x,y,t)$ a la imagen de entrada, podemos considerar todos los efectos del sub-sistema de adquisición de la imagen (filtrado de la lente, aberraciones cromáticas, distorsiones por muestreo, pérdida de información por conversión de formato, etc.) incluidos en la función de respuesta $h(x,y)$, por lo que a la entrada del algoritmo tendríamos $g(x,y,t)$ expresada como la convolución bidimensional de $f(x,y,t)$ y $h(x,y)$ tal y como se refleja en la ecuación (4.1). En la FIGURA 4.7 puede verse la posición de las señales en el sistema real (a) y en el esquema simplificado (b).

$$g(x,y,t) = f(x,y,t) ** h(x,y) \tag{4.1}$$

A su vez, llamaremos $S(\bar{x},\bar{y},t)$ a la imagen obtenida después de aplicar el algoritmo de súper-resolución, tal y como se refleja en la ecuación (4.2).

$$S(\bar{x},\bar{y},t) = g(x,y,t) ** SR(\bar{x},\bar{y}) \tag{4.2}$$

Y la imagen de súper-resolución de salida quedaría como indica la ecuación (4.3), en relación a las funciones de transferencia del subsistema de entrada y del codificador de vídeo híbrido funcionando en forma de súper-resolución.

$$S(\bar{x},\bar{y},t) = h(x,y) ** SR(\bar{x},\bar{y}) ** f(x,y,t) \tag{4.3}$$

El algoritmo comienza suponiendo que se encuentran disponibles un número *'p'* de imágenes de baja resolución $g(x,y,t_i)$, donde *'t_i'* indica el instante de tiempo en el que la imagen fue adquirida. Hay que tener en cuenta que el algoritmo se restringe a tratar *'p'* imágenes de entrada, por lo que sólo resulta útil referenciar estas *'p'* últimas imágenes. Por ello, usaremos el subíndice *'l'*, definido como $l = i \ mod \ p$, para hacer referencia a variables internas al algoritmo. Para mayor claridad, usaremos la memoria de imagen $g'_l(x,y)$ para almacenar la imagen de entrada que el algoritmo se dispone a usar, tal y como se indica en (4.4) y se muestra en la FIGURA 4.8. Llamando $\overline{g}'(x,y)$ a la imagen media de entrada, tal y como vienen dada por (4.5), el error promedio para la primera iteración se obtiene calculando primero las diferencias entre esta imagen media y cada una de las imágenes de entrada, como se recoge en la ecuación (4.6).

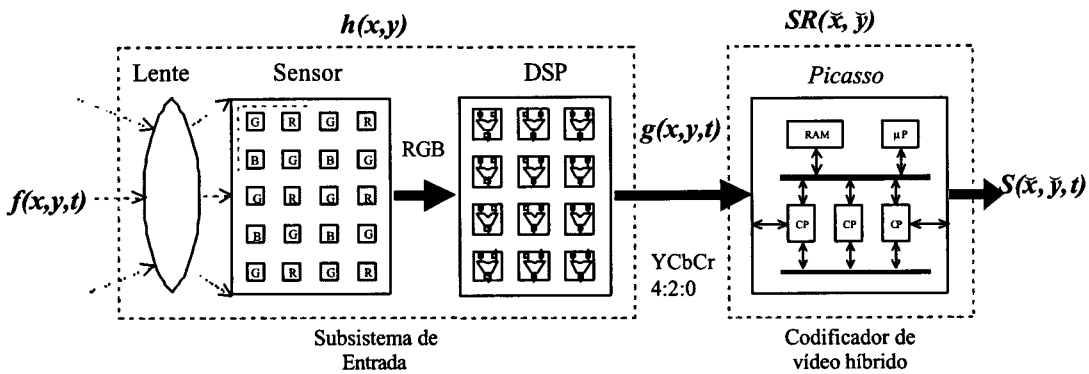
$$g'_l(x,y) = g(x,y,t_i) \quad / \quad l = i \bmod p \tag{4.4}$$

$$\overline{g}'(x,y) = \frac{1}{p}\sum_{l=0}^{p-1}\left(\frac{1}{N\cdot M}\cdot\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} g'_l(i,j)\right) \quad , \quad \forall\, x,y \tag{4.5}$$

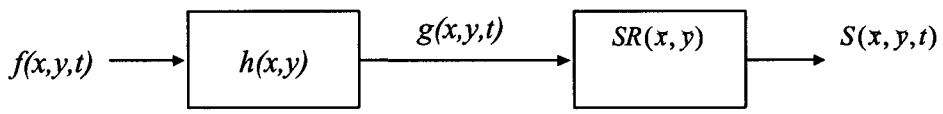
$$e_l(x,y)^{(1)} = g'_l(x,y) - \overline{g}'(x,y) \quad , \quad l = 0..(p-1) \tag{4.6}$$

Este error hay que llevarlo a alta resolución mediante, por ejemplo, un interpolador de orden cero, obteniéndose el error en alta resolución como se refleja en (4.7). Dado que los píxeles de alta resolución ausentes van a ser restaurados usando técnicas de súper-resolución, no merece la pena invertir en un interpolador de mayor calidad, que resultaría más caro y lento.

$$e_l(\bar{x},\bar{y})^{(1)} = \textbf{\textit{upsample}}\,(e_l(x,y)^{(1)}) \quad , \quad l = 0..(p-1) \tag{4.7}$$



(a)



(b)

FIGURA 4.7. Señales que intervienen en el sistema global. (a) Con respecto al sistema real y (b) esquema simplificado.

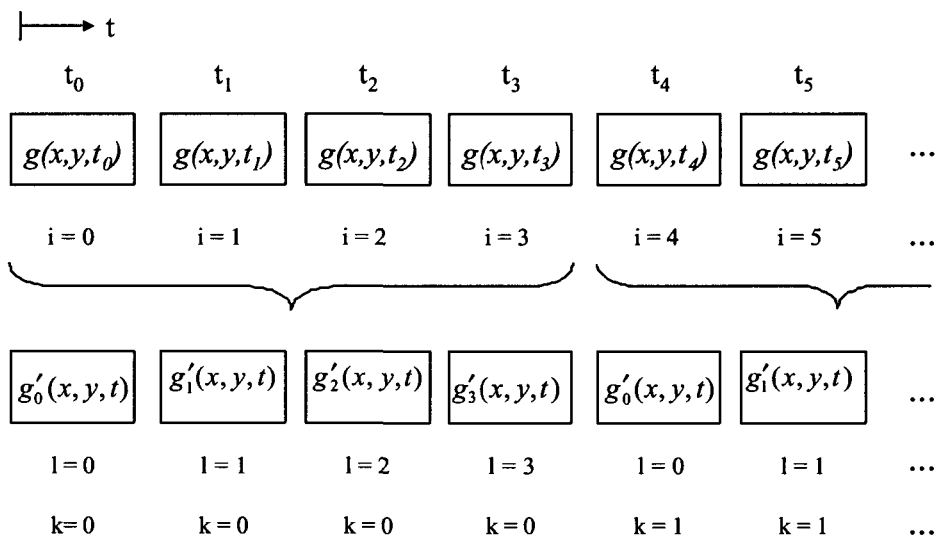


FIGURA 4.8. Nomenclatura y numeración usada para referenciar las imágenes de entrada.

Una vez en alta resolución hay que ajustar el error a la trama de referencia, desplazando la imagen $\Delta\delta_l(x,y)^{(l)}_{(fr2ref)}$ y $\Delta\lambda_l(x,y)^{(l)}_{(fr2ref)}$ en las coordenadas horizontales y verticales respectivamente. En principio, estos desplazamientos se aplican a cada píxel de

forma individual, aunque esto dependerá de la técnica que se use para la estimación y la compensación del movimiento. Una vez ajustados todos los errores a la referencia, se promedian y ese promedio pasa a ser la primera actualización de la imagen de súper-resolución, como muestra la ecuación (4.8).

$$S_0(\breve{x},\breve{y})^{(1)} = \frac{1}{p}\cdot\sum_{l=0}^{p-1} e_l\left(\breve{x}+\Delta\delta_l(x,y)^{(1)}_{(fr2ref)}, \breve{y}+\Delta\lambda_l(x,y)^{(1)}_{(fr2ref)}\right)^{(1)} \tag{4.8}$$

La ecuación (4.8) refleja el resultado de la primera iteración. Así por ejemplo $S_0(\breve{x},\breve{y})^{(1)}$ es la primera versión de la imagen de súper-resolución, correspondiente a $t=t_0$, y que tendrá que irse mejorando en las sucesivas iteraciones. La iteración *n-sima* parte de esta imagen y comienza por obtener mediante diezmado una versión en baja resolución para a continuación calcular los desplazamientos entre cada una de las imágenes de entrada y esta imagen diezmada y viceversa, es decir, los desplazamientos entre la imagen diezmada y las imágenes de entrada. De esta forma tendríamos disponibles los desplazamientos de la iteración *n-sima*: $\Delta\delta_l(x,y)^{(n)}_{(fr2ref)}$, $\Delta\lambda_l(x,y)^{(n)}_{(fr2ref)}$, $\Delta\delta_l(x,y)^{(n)}_{(ref2ref)}$ y $\Delta\lambda_l(x,y)^{(n)}_{(ref2fr)}$. La versión en baja resolución de la imagen obtenida en alta resolución en este momento del proceso se obtienen de la ecuación (4.9).

$$S_0(x,y)^{(n)} = \textbf{\textit{downsample}}\left(S_0(\breve{x},\breve{y})^{(n-1)}\right) \tag{4.9}$$

El siguiente paso es compensar el movimiento de la imagen de alta resolución hacía los diferentes *frames* de entrada usando los desplazamientos $\Delta\delta_l(x,y)^{(n)}_{(ref2ref)}$ y $\Delta\lambda_l(x,y)^{(n)}_{(ref2fr)}$ para a continuación llevarlos a baja resolución y obtener el error con respecto a cada una de las imágenes de entrada, como puede apreciarse en (4.10).

$$e_l(x,y)^{(n)} = g_l'(x,y) - S_0\left(x+\Delta\delta_l(x,y)^{(n)}_{(ref2fr)}, y+\Delta\lambda_l(x,y)^{(n)}_{ref2fr}\right)^{(n)} \qquad l = 0\,..\,(p-1) \tag{4.10}$$

Este error de baja resolución tiene que llevarse nuevamente a alta resolución mediante interpolación y compensar su movimiento de nuevo hacia la referencia. El promedio de estos

'p' errores constituyen la actualización incremental n-esima de la imagen de alta resolución, tal y como se muestra en (4.11).

$$S_0(\breve{x},\breve{y})^{(n)} = S_0(\breve{x},\breve{y})^{(n-1)} + \frac{1}{p}\cdot\sum_{l=0}^{p-1} e_l\left(\breve{x}+\Delta\delta_l(x,y)^{(n)}_{(fr2ref)}, \breve{y}+\Delta\lambda_l(x,y)^{(n)}_{(fr2ref)}\right)^{(n)} \quad (4.11)$$

Consideramos que se alcanza la convergencia cuando los cambios en el error promedio son mínimos, es decir, cuando la varianza del error promedio está por debajo de un cierto umbral establecido de forma empírica.

Una vez obtenida la imagen de súper-resolución para el instante t_0 con las primeras 'p' imágenes, tenemos que repetir el proceso con la siguientes 'p' imágenes para obtener la siguiente imagen de súper-resolución usando un número máximo prefijado de iteraciones o iterando hasta que se alcance convergencia. La obtención de cada imagen de súper-resolución implica el empleo de 'p' imágenes de baja resolución, por lo que en el instante t_i, estaríamos generando la imagen de súper-resolución $k = entero\left(\frac{i}{p}\right)$. En este caso, la ecuación (4.11) pasaría a adoptar la forma más genérica recogida en (4.12).

$$S_k(\breve{x},\breve{y})^{(n)} = S_k(\breve{x},\breve{y})^{(n-1)} + \frac{1}{p}\cdot\sum_{l=0}^{p-1} e_l\left(\breve{x}+\Delta\delta_l(x,y)^{(n)}_{(fr2ref)}, \breve{y}+\Delta\lambda_l(x,y)^{(n)}_{(fr2ref)}\right)^{(n)} \quad (4.12)$$

4.3.3Descripción del algoritmo iterativo básico en Pseudo-código

El siguiente paso ha sido realizar una descripción del algoritmo en pseudo-código, con el objetivo de facilitar su posterior implementación. Esta primera versión del algoritmo es la mostrada en la FIGURA 4.9.

Esta versión usa tres memorias de alta resolución (con prefijo HR, *High Resolution*) y una cantidad variable de memorias de baja resolución (con prefijo LR, *Low Resolution*) igual al número de imágenes de entrada que se desee almacenar para iniciar el algoritmo, más una memoria adicional que recoja datos intermedios de baja resolución (LR_B).

Analizando de forma conceptual el algoritmo, vemos que el proceso se inicia calculando la diferencia entre cada una de las imágenes de entrada (sólo en luminancia) y una

imagen con un valor promedio de la equivalente de alta resolución. Cada imagen error se desplaza para ajustarse a la imagen de referencia y se calcula el promedio de todos los errores. Este error promedio desplazado constituye la primera actualización de la imagen final de súper-resolución, que ha sido previamente inicializada a cero. Las siguientes iteraciones repiten en esencia el mismo proceso, pero partiendo siempre de la ultima imagen de súper-resolución obtenida, por lo que hay que recalcular el movimiento. Es decir, se comienza por estimar los desplazamientos entre la imagen de entrada y la de súper-resolución llevada a baja resolución y viceversa. A continuación se desplaza la imagen de súper-resolución nuevamente a la imagen de entrada. Esta imagen desplazada se lleva a baja resolución para calcular el error con respecto a la imagen de entrada. El error se lleva de nuevo a alta resolución y allí se desplaza a la referencia. Este proceso tiene que repetirse, dentro de cada iteración, para cada una de las imágenes de entrada. Los errores obtenidos son promediados y este promedio constituye la actualización de la imagen de súper-resolución. La calidad de la imagen de súper-resolución irá aumentando con el número de iteraciones, aunque esta mejora será cada vez menor a medida que la imagen de súper-resolución se vaya adecuando a todas las imágenes de entrada disponibles.

4.3.4Crítica del algoritmo iterativo básico

El algoritmo comienza almacenando una secuencia de imágenes de entrada en la memoria LR_I. En nuestro caso se ha optado por comenzar con 4 imágenes de baja resolución ($p = 4$) , ya que ese es el incremento en área que la imagen de alta resolución va a experimentar. Evidentemente, mientras mayor sea el número de imágenes disponibles, mayor será la probabilidad de disponer de todos los datos necesarios para una reconstrucción óptima. Sin embargo, este aumento del número de datos de entrada supone también un aumento considerable de los requisitos de memoria y de tiempo de cálculo en cada iteración. Siempre existe un compromiso cuantificable entre la calidad final y los requerimientos de cálculo (tiempo y recursos). La versión de vídeo propuesta en el capítulo 5 supone posiblemente como veremos la mejor combinación entre estos dos factores clave.

Como puede apreciarse, esta temprana versión no asume todavía ningún esquema real para el cálculo de los desplazamientos, suponiendo sencillamente que los desplazamientos entre cada imagen y la referencia son objeto de otro problema separado y en este caso son perfectamente conocidos. En esta tesis se pretende aportar un enfoque más completo y realista que incluya, entre otras cosas, el cálculo de estos desplazamientos.

```
Create HR image HR_B, HR_B_Shift, HR_B_Average
Create LR image LR_B
Create series of LR images LR_I[#images]

Read an aligned non-interlaced series to LR_I[]
Pixel align LR_I[] with LR_I[0], remember sub-pix shift

Fill HR_B with zero
Fill LR_B with average

First Iteration
    LR_B = LR_I[] - LR_B
    Upsample LR_B to HR_B_Shift
    Back shift HR_B_Shift and average in HR_B_Average
    HR_B = HR_B + HR_B_Average
Next Iterations
    Shift HR_B to HR_B_Shift
    Downsample HR_B_Shift to LR_B
    LR_B = LR_I[] - LR_B
    Upsample LR_B to HR_B_Shift
    Back shift HR_B_Shift and average in HR_B_Average
    HR_B = HR_B + HR_B_Average
```

FIGURA 4.9. Pseudo-código de la primera versión del primer algoritmo iterativo.

4.3.5 Primera versión del algoritmo iterativo básico sobre la plataforma *Picasso*

Si ahora nos restringimos al uso de los recursos proporcionados por la arquitectura de codificación híbrida de vídeo "*Picasso*", podemos escribir de nuevo el algoritmo tal y como se muestra en la FIGURA 4.10. Esta primera versión, etiquetada como v1.0, supone la primera implementación real del algoritmo de súper-resolución. El algoritmo está estructurado en tres partes: inicialización, primera iteración y bucle para el resto de las iteraciones. El algoritmo termina al completar el número de iteraciones fijado o cuando la varianza del error es inferior a 0.5. La inicialización parte de un valor llamado *scale* (en nuestro caso igual a 2) que será el incremento de tamaño horizontal y vertical. A partir del valor de *scale*, se establece *nr_frames* (número de *frames* de entrada) como $scale^2$ (en este caso 4), ya que este será el incremento de área de la imagen. A continuación se declaran las memoria que se va a usar y se inicializan algunos valores. En concreto:

- HR_B: Memoria *(buffe)r* acumulativa en alta resolución, donde se almacenará el resultado del proceso de súper-resolución.

- HR_A: Memoria (*average*) donde se realizará el cálculo del valor medio en alta resolución.
- HR_S: Memoria (*shift*) de alta resolución para almacenar el resultado de los desplazamientos.
- HR_T: Memoria temporal de alta resolución.
- LR_B: Versión en baja resolución de HR_B obtenida mediante diezmado.
- LR_I[]: Vector de memorias para almacenar las imágenes de entrada. Sus índices van de 0 a *nr_frames*-1.
- MV_fr2ref[] y MV_ref2fr[]: Vectores de memorias destinadas a albergar los vectores de desplazamiento estimados.

Uno de los principales problemas a resolver fue el de la estimación del movimiento, necesario para compensar el movimiento en las sucesivas mejoras incrementales. Este es un aspecto clave en los algoritmos de súper-resolución y su influencia en la calidad de la imagen resultante es muy alta, como ya veremos. En la literatura sobre el tema [EF97], [HKK87a] es habitual hacer una separación teórica entre este problema y el propio de súper-resolución, limitándose mayoritariamente a suponer los desplazamientos perfectamente conocidos en el momento de aplicar el algoritmo. Aún en el caso de incluir la determinación del movimiento, ocurre que los autores se remiten a procedimientos a nivel de píxel, muy precisos, pero muy costosos desde el punto de vista computacional, y por supuesto, en su inmensa mayoría, con escasas o ninguna posibilidad de ser implementados en tiempo real con la tecnología existente hoy en día, e incluso con las previsiones de tecnología disponible dentro de algunos años. El objetivo de esta tesis es encontrar una solución completa y viable en tiempo real. Para ello es absolutamente necesario incorporar la estimación del movimiento dentro del algoritmo de súper-resolución. Como ya se ha comentado en el capítulo 3, el camino escogido para minimizar los costes de implementación ha sido el de reutilizar la arquitectura para la codificación híbrida de vídeo *Picasso*. Desde este punto de vista, la súper-resolución es una característica de valor añadido, incorporada a *Picasso* con un coste industrial prácticamente nulo. Recíprocamente, *Picasso* pude dar una solución al problema de súper-resolución en tiempo real, si se encuentra un mapeo adecuado del algoritmo a los recursos de la arquitectura.

A la luz de los recursos disponibles y de las prestaciones de tiempo real y bajo coste deseadas, la mejor forma de realizar la estimación del movimiento es usar el estimador de movimiento ('*motion estimator*') que incluido en el módulo de compresión de vídeo de *Picasso*, Esto supone la revisión de los siguientes aspectos:

```
* Block align super-resolution algorithm v1.0

Set the value of the improvement: scale
nr_frames = scale*scale
If LR_I[] is a MxN matrix, then LR_B is the same size: LR_B[M][N]
HR_B[scale*M][scale*N], HR_S[scale*M][scale*N], HR_A[scale*M][scale*N]
Read a set of aliased Low-Resolution images in LR_I[#nr_frames]
Set the value of the maximum number of iterations: nr_iterations
The size of motion vector matrixes depends on the block size of the Motion Estimation

MV_fr2ref[0] = 0

FOR fr = 1 .. nr_frames-1
        MV_fr2ref[fr] = Calc_Motion_Estimation (LR_I[fr], LR_I[0])
            MV_fr2ref[fr] = 2 .* MV_fr2ref[fr]
END FOR

HR_A = 0
HR_B = 0

// First Iteration
FOR fr = 0 .. nr_frames-1
        LR_B = 128
        LR_B = LR_I[fr] – LR_B
        HR_S = Upsample (LR_B, scale)
        HR_T = Motion_Compensation (HR_S, MV_fr2ref[fr])
        HR_S = HR_T
        HR_A = HR_A + HR_S
END FOR

HR_A = HR_A ./ nr_frames
HR_B = HR_B + HR_A
Contrast_Clip (HR_B, 0, 255)

// Next Iterations
FOR it = 1 .. nr_iterations
        LR_B = Downsample(HR_B, scale)
            FOR fr = 0 .. nr_frames-1
                MV_fr2ref[fr] = Calc_Motion_Estimation (LR_I[fr], LR_B)
                MV_ref2fr[fr] = Calc_Motion_Estimation (LR_B, LR_I[fr])
                MV_fr2ref[fr] = scale * MV_fr2ref[fr]
                 MV_ref2fr[fr] = scale * MV_ref2fr[fr]
             END FOR
            HR_A = 0
            FOR fr = 0 .. nr_frames-1
                HR_S = Motion_Compensation (HR_B, MV_ref2fr[fr])
                LR_B = Downsample(HR_S, scale)
                LR_B = LR_I[fr] – LR_B
            HR_S = Upsample (LR_B, scale)
                HR_T = Motion_Compensation (HR_S, MV_fr2ref[fr])
                HR_S = HR_T
                HR_A = HR_A + HR_S
            END FOR
            HR_A = HR_A ./ nr_frames
            variance = Variance(HR_A)
            HR_B = HR_B + HR_A
    Contrast_Clip (HR_B, 0, 255)
    IF (variance < 0.5)  break;
END FOR
```

FIGURA 4.10. Pseudo-código del primer algoritmo iterativo usando los recursos de un codificador de vídeo híbrido.

- La estimación de movimiento debe realizarse, no a nivel de píxel, como sería deseable desde el punto de vista algorítmico, sino a nivel de macro-bloque, es decir, de agrupaciones de 8×8 píxeles, tal y como exige el módulo de compensación de vídeo.
- Debe analizarse la influencia del método de estimación (*full-search, modified 3 dimensional recursive search 3DRS, etc.*) y sus parámetros (área de búsqueda, tamaño máximo de los vectores de movimiento, precisión, etc.) en la calidad de los vectores de movimiento obtenidos y sobre todo en la imagen final.
- Debe determinarse el tamaño de macro-bloque óptimo para el problema planteado. Un tamaño pequeño nos acercaría al ideal de desplazamiento por píxel, pero estará más afectado por el ruido, mientras que un bloque mayor ofrecerá, en general, un vector de movimiento global de mayor calidad, pero de peor calidad cuando se trata de detectar pequeños movimientos locales.
- Debe estudiarse la influencia de la presencia de *aliasing* en la estimación de movimiento, ya que el estimador de movimiento está optimizado para compresión de vídeo sin *aliasing*.

El estimador de movimiento del compresor de vídeo usado en *Picasso* tiene originalmente una precisión de ½ píxel, por lo que la primera modificación introducida ha sido la de introducir una etapa más de refinamiento en el calculo de los vectores para llevar la precisión a ¼ de píxel. No obstante, esta característica se ha dejado como un elemento configurable, con el objeto de poder comparar experimentalmente como afecta la precisión de los vectores de movimiento en la calidad final de la imagen.

Al realizar la estimación de movimiento en baja resolución, los vectores obtenidos son también de baja resolución, por lo que para realizar la compensación de movimiento en alta resolución previamente hay que escalar convenientemente los vectores, multiplicándolos por el factor de escala usado.

En principio, el algoritmo de súper-resolución está pensado para cortas secuencias de imágenes (`nr_frames`) donde cada una de ellas supone una versión desplazada de la anterior. Nosotros llamaremos a este tipo de secuencias "imágenes afectadas por desplazamiento global", ya que toda la imagen está afectada por el mismo vector de movimiento, que es único y global para cada imagen de la secuencia. Como el estimador de movimiento está preparado para calcular el vector de movimiento de cada macro-bloque de luminancia (cuatro bloques DCT de luminancia, es decir un vector de movimiento cada 16x16 píxeles) lo que haremos

será calcular a partir de este conjunto de vectores el vector global. Para ello se podrían seguir tres filosofías:

1. Calcular la media de todos los vectores de desplazamiento horizontales y verticales (*mv.x* y *mv.y*) y asignar este valor al vector global (*MV.x* y *MV.y*). Esta operación es la recogida en (4.13) para un número MxN de macro-bloques de luminancia.

$$MV.x = \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} mv(i,j).x$$
$$MV.y = \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} mv(i,j).y \qquad (4.13)$$

2. Calcular el vector mas frecuente y asignar este vector al vector global.

3. Calcular la componente del vector más frecuente en el sentido horizontal y en el sentido vertical de forma independiente y asignar estas dos componentes al vector global.

Las pruebas que hemos realizado en este sentido arrojan una gran convergencia entre las tres opciones, observándose tan solo ocasionalmente pequeñas desviaciones, normalmente de valor inferior a ¼ de píxel. Por ejemplo, en la TABLA 4.4 pueden verse los vectores calculados siguiendo los tres procedimientos y el error cometido en cada caso. En este caso el método de estimación ha sido una búsqueda recursiva en tres dimensiones modificada (3DRS), con precisión de ¼ de píxel, y la secuencia de prueba usada fueron 12 *frames* artificialmente desplazados de la secuencia *KANTOOR,* donde el *frame* 0 se usó como referencia.

En lugar de calcular inicialmente la imagen media $g'(x,y)$ como se expresaba en (4.5), resulta más rápido e igualmente preciso el partir del valor medio de los posibles valores de píxel. Como cada píxel se recoge como un número de 8 bits, y en el caso de la luminancia sus valores son siempre positivos [0..255], el valor medio es 128. Aunque partiendo del valor medio real se alcanza antes la convergencia, seguiríamos estando muy lejos del funcionamiento en tiempo real, por lo que no tiene demasiado sentido incrementar la carga computacional del algoritmo en la etapa inicial que, en todo caso, puede ahorrar como mucho una iteración. Aunque la carga computacional de esta iteración extra sea sin duda mayor que la del cálculo del valor medio su efecto no es significativo para el objeto de alcanzar las prestaciones deseadas y de esta otra forma el algoritmo queda más simplificado.

Vectores reales		Vector Medio				Vector más frecuente				Componente más frecuente			
		valores		Error		valores		error		valores		error	
2	-2	2	-2	0	0	2	-2	0	0	2	-2	0	0
3	-1	3	0	0	-1	3	-1	0	0	3	-1	0	0
3	-3	3	-3	0	0	3	-3	0	0	3	-3	0	0
2	0	2	0	0	0	2	0	0	0	2	0	0	0
0	-3	0	-3	0	0	0	-3	0	0	0	-3	0	0
0	-1	0	-1	0	0	0	-1	0	0	0	-1	0	0
-2	-1	-2	0	0	-1	-2	-1	0	0	-2	-1	0	0
1	0	1	0	0	0	1	0	0	0	1	0	0	0
0	-2	0	-2	0	0	0	-2	0	0	0	-2	0	0
2	0	2	0	0	0	2	0	0	0	2	0	0	0
0	2	0	2	0	0	0	2	0	0	0	2	0	0

TABLA 4.4. Errores en la estimación del vector global para la secuencia de prueba *KANTOOR* de 12 *frames*. El *frame* 0 es la referencia.

Después de realizar el cálculo del valor medio debemos tener en cuenta que toda aquella memoria que vaya a contener una imagen no puede tener valores que no puedan ser albergados en 8 bits, por lo que antes de almacenar el resultado de una operación aritmética en una memoria, debe realizarse un recorte de los valores de la imagen a 8 bits. Es lo que en el algoritmo se recoge como '**Contrast_Clip**'. De hecho, este es uno de los primeros problemas con los que nos encontramos por el hecho de querer mapear aplicaciones de procesamiento de imágenes reutilizar una arquitectura pensada para la compresión de imágenes: los problemas aritméticos.

Es un principio básico de la aritmética binaria que la suma de dos números de N bits da como resultado un número de N+1 bits. Este principio tan básico ha supuesto una fuente importante de problemas en el desarrollo del algoritmo. Como ya se vio en el capitulo 3, dentro de la arquitectura *Picasso* las imágenes se leen en tiras de macro-bloques dentro de los co-procesadores. En ese momento, cada píxel está representado en una cifra digital de 8 bits. Una vez dentro del coprocesador, los valores de la imagen se procesan en una arquitectura de ancho de palabra de 16 bits (procesamiento a nivel de bloque), pero el resultado tiene que ser de nuevo almacenado en una memoria de imagen de 8 bits, ésto se ha esquematizado en la FIGURA 4.11. Para la compresión de imágenes esto no resulta ser especialmente un problema, pero al intentar reutilizar la misma arquitectura para procesos con operaciones aritméticas sobre la imagen nos encontramos con la limitación de que cualquier resultado intermedio que se desee almacenar tendrá que limitarse a 8 bits. Ante este problema hemos adoptado las siguientes soluciones:

- Intentar realizar todas las operaciones aritméticas a nivel de bloque.

- Reorganizar las operaciones aritméticas de forma que al guardar las imágenes intermedias éstas estén limitadas dentro de lo posible a 8 bits.

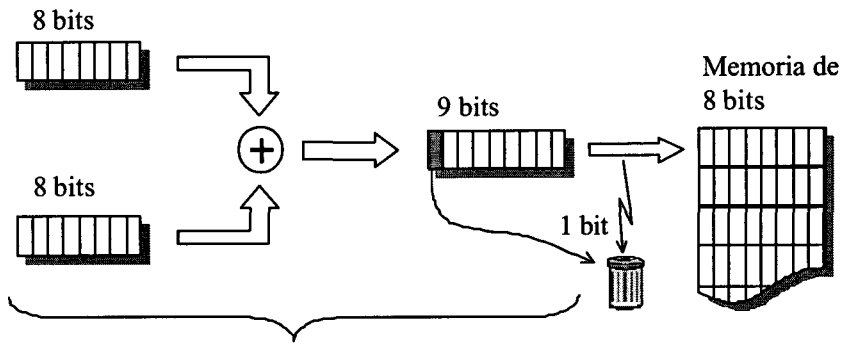


FIGURA 4.11. Efectos de la pérdida de precisión al realizar operaciones aritméticas sobre imágenes de 8 bits.

Dentro del aspecto de la reorganización de las operaciones aritméticas, el cambio más inmediato es el de distribuir la división implicada por el cálculo de los valores medios en divisiones dentro del bucle. Para ello tenemos dos alternativas, tal y como se muestra también en la FIGURA 4.12:

- La primera es dividir cada una de las imágenes por el total (4 en nuestro caso) antes de sumarlas, pero eso conlleva un importante recorte del margen dinámico que al final afecta negativamente a la calidad de las imágenes.
- La segunda es distribuir la división entre la operación de resta y el acumulador que soporta la suma de todos los errores. De esta forma el recorte del margen dinámico queda dividido entre la memoria que almacena la resta y la memoria que almacena la suma acumulada.

Esta segunda opción es la que ofrece mejores resultados y ha sido por lo tanto la seleccionada. Aunque con este arreglo se logra una solución satisfactoria, a efectos de simulación se ha optado por incrementar el tamaño de la memoria acumuladora HR_A a 16 bits, con la intención de poder cuantificar de forma más precisa los desbordamientos producidos.

Otro importante problema es al hecho de que la arquitectura *Picasso* no está preparada para soportar simultáneamente dos tamaños diferentes de memoria. En este caso se ha optado por la solución de asumir que el sensor sólo entrega al compresor la región de interés que se

desea mejorar (*ROI, Region of Interest*) lo que equivale a usar sólo imágenes de alta resolución.

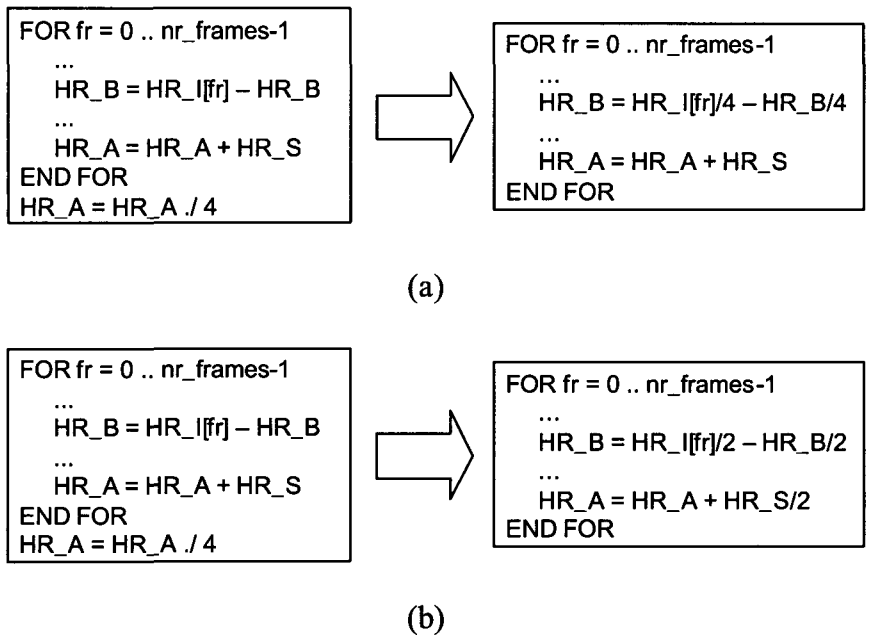
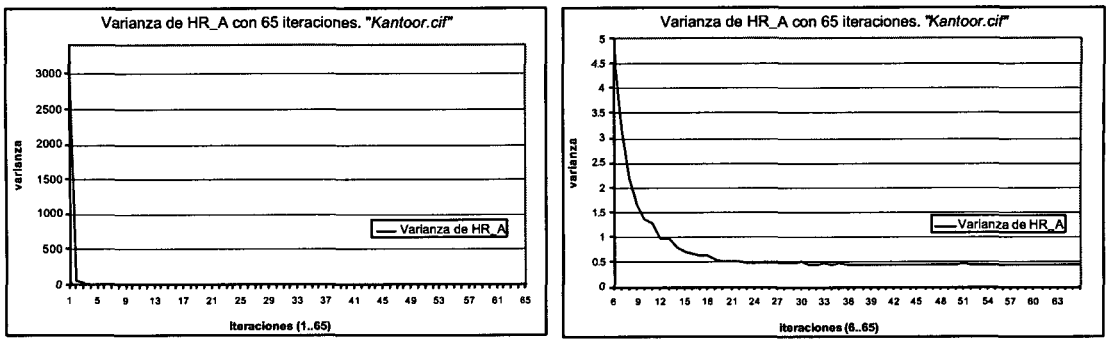


FIGURA 4.12. Estrategias de distribución de código para el cálculo del valor medio. (a) Agrupando la división en unas sola operación. (b) Distribuyendo la división.

Esta solución de usar un solo tamaño de memorias supone además la ventaja adicional de evitar las operaciones de interpolado y diezmado para aumentar y reducir el tamaño de las imágenes, evitando los errores que generan estas operaciones. Asimismo, en tanto en cuanto la estimación de movimiento se hace también en alta resolución, el escalado de los vectores es nuevamente innecesario.

En la FIGURA 4.13 (a) puede verse el valor de la varianza de HR_A para la imagen de prueba *KANTOOR* de tamaño original QCIF a medida que aumenta el número de iteraciones hasta 65. En la FIGURA 4.13 (b) puede observarse dicho valor a partir la iteración número seis, con la intención de resaltar la tendencia de la varianza hacía el valor 0.439. Por este motivo, se selecciona de forma empírica la varianza 0.5 como valor de corte para detener el proceso iterativo. En este caso el valor de corte se alcanzaría en la iteración número 21.

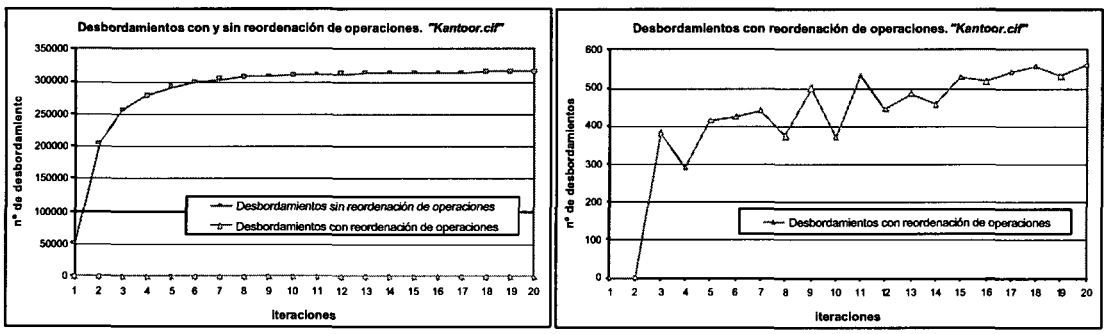


(a) (b)

FIGURA 4.13. Comportamiento de la varianza de HR_A durante 65 iteraciones (a) y detalle que muestra la varianza a partir de la iteración número 6 (b).

4.3.6Mejoras al algoritmo de súper-resolución iterativo básico para mapeado en codificador híbrido *Picasso* original

Bajo todas estas premisas, se ha desarrollado la segunda versión del algoritmo iterativo de súper-resolución, mostrado en la FIGURA 4.15 y etiquetada como v1.1. El algoritmo ha quedado notablemente simplificado al haberse eliminado las operaciones de cambio de tamaño entre alta y baja resolución. La memoria HR_A aparece en negrita por tratarse de una memoria especial de 9 bits. Las etiquetas que aparecen a la derecha entre corchetes son las acciones emitidas al co-procesador y reflejan el conjunto de operaciones realizadas por el coprocesador a nivel de bloque, es decir, con una precisión interna de 16 bits.



(a) (b)

FIGURA 4.14. Desbordamientos aritméticos producidos durante 20 iteraciones con y sin reordenación de operaciones (a) y detalle que muestra sólo los desbordamientos con reordenación de operaciones aritméticas (b).

```
* Iterative super-resolution algorithm v1.1

Set the value of the improvement: scale
nr_frames = scale*scale, M'= scale*M, N'= scale*N
HR_B[M'][N'], HR_S[M'][N'], HR_T[M'][N'] all of 8 bits. HR_A[M'][N'] is 9 bits,
Read a set of aliased Low-Resolution images in LR_I[#nr_frames][M][N]
Set the value of the maximum number of iterations: nr_iterations
The size of motion vector matrixes depends on the block size of the Motion Estimation

MV_fr2ref[0] = 0

FOR fr = 1 .. nr_frames-1
    HR_I[fr] = Upsample (LR_I[fr], scale)                          [SR_STORE]
    MV_fr2ref[fr] = Calc_Motion_Estimation (HR_I[fr], HR_I[0])
END FOR

HR_A = 0                                                           [SR_INIT_A_B]
HR_B = 0

// First Iteration
FOR fr = 0 .. nr_frames-1
        HR_S = 128                                                 [SR_ACT1]
        HR_S = HR_I[fr]/2 – HR_S/2
        HR_T = Motion_Compensation (HR_S, MV_fr2ref[fr])           [SR_ACT2]
        HR_S = HR_T
        HR_A = HR_A + HR_S/2
END FOR

HR_B = HR_B + (HR_A)                                               [SR_UPDATE]

// Next Iterations
FOR it = 1 .. nr_iterations
        FOR fr = 0 .. nr_frames-1
            MV_fr2ref[fr] = Calc_Motion_Estimation (HR_I[fr], HR_B)
            MV_ref2fr[fr] = Calc_Motion_Estimation (HR_B, HR_I[fr])
        END FOR

        HR_A = 0                                                   [SR_INIT_A]

        FOR fr = 0 .. nr_frames-1
            HR_S = Motion_Compensation (HR_B, MV_ref2fr[fr])       [SR_ACT3]
            HR_S = HR_I[fr]/2 – HR_S/2
            HR_T = Motion_Compensation (HR_S, MV_fr2ref[fr])       [SR_ACT2]
            HR_S = HR_T
            HR_A = HR_A + HR_S/2
        END FOR
        HR_B = HR_B + (HR_A)                                       [SR_UPDATE]
        variance = variance(HR_A)                                  [SR_STAT]
        If (variance < 0.5)  break
END FOR
```

FIGURA 4.15. Pseudo-código del algoritmo iterativo modificado v1.1.

La redistribución de las operaciones aritméticas consigue disminuir de forma muy importante el número de desbordamientos producidos en la ejecución del código. La FIGURA 4.14 (a) muestra el número de desbordamientos producidos en la versión v.1.0 sin reordenación de operaciones y en la versión v1.1 con reordenación de operaciones. Debe tenerse en cuenta que el número máximo de desbordamientos por iteración sería el tamaño de la imagen de alta resolución (352×288) multiplicado por las tres componentes (luminancia, crominancia roja y crominancia azul) y multiplicado por el número de operaciones que pueden provocar desbordamiento, en nuestro caso tres (SR_ACT1 o SR_ACT3, SR_ACT2 y SR_UPDATE), es decir 352×288×3×3 = 912384. Puede apreciarse que sin reordenación de las operaciones el número de desbordamientos alcanza los 298196 en la 6ª iteración, es decir un 32.68% de las operaciones aritméticas realizadas producen desbordamiento. Con reordenamiento, para el mismo número de iteraciones, los desbordamientos son 424, que representa el 0.046% de las operaciones realizadas. En la TABLA 4.5 puede observarse el número exacto de desbordamientos producidos en ambas versiones y el porcentaje que esto representa sobre el total de operaciones aritméticas realizadas.

Iteración	Versión v1.0		Versión 1.1	
	total	%	total	%
1	50688	5.56	0	0.000
2	202960	22.25	0	0.000
3	255064	27.96	380	0.042
4	277290	30.39	289	0.032
5	290483	31.84	413	0.045
6	298196	32.68	424	0.046
7	303131	33.22	440	0.048
8	306110	33.55	372	0.041
9	308189	33.78	499	0.055
10	309610	33.93	370	0.041
11	310888	34.07	533	0.058
12	311791	34.17	444	0.049
13	312464	34.25	484	0.053
14	313181	34.33	456	0.050
15	313600	34.37	529	0.058
16	314040	34.42	515	0.056
17	314402	34.46	543	0.060
18	314683	34.49	556	0.061
19	315017	34.53	530	0.058
20	315118	34.54	561	0.061

TABLA 4.5. Desbordamientos aritméticos producidos en las versiones v1.0 y versión v1.1 y porcentaje sobre el total de operaciones aritméticas realizadas a nivel de bloque.

Los desbordamientos provocan saturaciones en la imagen. Si el desbordamiento es provocado por una operación cuyo resultado es mayor que 256, este se cortará a 256 (*overflow*) y si el resultado es inferior a cero, se recortará a cero (*underflow*). Si el número de

desbordamientos es elevado se apreciará una degradación importante en la calidad de la imagen. Sin embargo, un número pequeño de desbordamientos es admisible, ya que tan solo supone un pequeño efecto de cuantificación, imperceptible en la mayor parte de los casos. No obstante, aunque visualmente este efecto sea inapreciable, se reflejará de forma importante como una disminución en la relación señal al ruido y en el coeficiente de correlación espectral.

4.3.7 Modificación de la arquitectura *Picasso* y transformaciones del algoritmo iterativo con referencia a la imagen promedio

Con el propósito de disminuir la cantidad de operaciones a realizar se ha modificado la arquitectura *Picasso* y se ha desarrollado la siguiente versión del algoritmo de súper-resolución etiquetada como v.1.2, se retoma también la idea de usar dos tamaños de imágenes (y por lo tanto de memorias) de forma simultánea. La nueva versión del algoritmo, mostrada en la FIGURA 4.16, recoge además algunas modificaciones adicionales para su adecuado mapeo en la nueva arquitectura.

Se ha decidido homogeneizar el bucle iterativo, eliminando la primera e irregular iteración. Se ha optado por usar como primera propuesta de imagen de súper-resolución la media de las imágenes de entrada, disminuyendo así además el ruido promedio en la imagen resultante. La instrucción SR_AVERAGE se encarga de leer cuatro imágenes de entrada de baja resolución, llevarlas a alta resolución mediante interpolación de orden cero y obtener la media, que pasará a formar la primera propuesta de imagen de alta resolución.

Puede observarse en el algoritmo el uso simultáneo de dos tamaños de memoria, como se detalla en la sección 4.3.7.1.

Se ha introducido, como en la versión v1.1, la memoria temporal HR_T de alta resolución para evitar que, al mismo tiempo que se lee la imagen HR_S para desplazarla, esta se sobrescriba con nuevos datos. A la hora de su implementación, esta memoria no será necesaria, ya que debido al solape existente en la arquitectura *Picasso* entre la memoria actual y la reconstruida de dos filas de macro-bloques, este problema de dependencia de datos queda automáticamente solventado por la estructura del propio codificador híbrido.

```
* Iterative super-resolution algorithm v1.2

    Set the value of the improvement: scale
    nr_frames = scale*scale, M'= scale*M, N'= scale*N
    HR_B[M'][N'], HR_S[M'][N'] , HR_T[M'][N'] all of 8 bits. HR_A[M'][N'] is 9 bits
    LR_B[M][N] for the motion estimation
    Read a set of aliased Low-Resolution images in LR_I[#nr_frames][M][N]
    Set the value of the maximum number of iterations: nr_iterations

    // Starting with the Average Image
    IF(frame_no==3)
        HR_B = Upsample(LR_I[0]+LR_I[1]+LR_I[2])+LR_I[3])      [SR_AVERAGE]
        HR_B = HR_B/4
    END IF

    // Iterations
    FOR it = 1 .. nr_iterations
        LR_B = Downsample(HR_B)                                [SR_DOWNSAMPLE]
        FOR fr = 0 .. nr_frames-1
            MV_fr2ref[fr] = Calc_Motion_Estimation (LR_I[fr], LR_B)
            Select_global_motion_vector()
            MV_ref2fr[fr] = - MV_fr2ref[fr]                    INVERT_MV()
            MV_fr2ref[fr] = 2 .* MV_fr2ref[fr]
            MV_ref2fr[fr] = 2 .* MV_ref2fr[fr]
        END FOR
        HR_A = 0                                               [SR_INIT_A]
        FOR fr = 0 .. nr_frames-1
            HR_S = Motion_Compensation (HR_B, MV_ref2fr[fr])   [SR_MOT_COMP1]
            HR_S = Upsample(LR_I[fr])/2 – HR_S/2               [SR_UPSAMPLE]
            HR_T = Motion_Compensation (HR_S, MV_fr2ref[fr])   [SR_MOT_COMP2]
            SR_S = HR_T
            HR_A = HR_A + HR_S/2                               [SR_ADD]
        END FOR
        HR_B = HR_B + HR_A                                     [SR_UPDATE]
        variance = variance(HR_A)                              [SR_STAT]
        If (variance < 0.5)  break
    END FOR
```

FIGURA 4.16. Pseudo-código del algoritmo iterativo v1.2, modificado usando dos tamaños de memorias de forma simultanea.

La memoria LR_B contendrá en todo momento una versión de baja resolución de la imagen de súper-resolución, obtenida por diezmado (*Downsample*) y su uso se restringe al cálculo de los vectores de movimiento en baja resolución. Debe tenerse en cuenta que la estimación de movimiento es una de las tareas más intensivas en cálculo, por lo que realizarla en baja resolución significa un ahorro importante de tiempo y consumo de potencia. Otra forma de reducir los requerimientos del algoritmo ha consistido en eliminar la segunda estimación de movimiento.

Hasta el momento siempre hemos realizado dos estimaciones de movimiento por iteración y *frame* de entrada: una de la imagen de súper-resolución respecto del *frame* de entrada y otra del *frame* de entrada respecto de la imagen de súper-resolución. Sin embargo, debido al tipo de estimación de movimiento que estamos realizando, con vectores expresados de forma cartesiana, supone una buena aproximación considerar que los segundos vectores de movimiento se pueden obtener de los primeros mediante una simple inversión de sentido. Aunque conceptualmente esto es correcto, en la practica no se obtienen los mismos vectores de movimiento invertidos si se realizan ambas estimaciones y se comparan, debido a que el proceso de estimación de movimiento es realmente un simple *block matching*, es decir, una comparación de las sumas de las diferencias absolutas (SAD, *Sum of absolute differences*) acumuladas obtenidas al comparar cada bloque DCT con otros bloques DCT de la otra imagen dentro de un cierto área de búsqueda. Así pues es posible que al ser diferentes las áreas de búsqueda pueda existir otro bloque DCT en la segunda imagen que arroje un menor SAD, alterando así el vector de movimiento homólogo, aunque normalmente de forma muy leve. En el algoritmo propuesto de la versión v1.2 esto se recoge como una instrucción que es en realidad una función (por eso no se ha encerrado entre corchetes) denominada INVERT_MV().

4.3.7.1 Diagrama de bloques y requerimientos de memoria

En la FIGURA 4.17 se muestra el diagrama de bloques del algoritmo. Los bloques funcionales se han sombreado para poder distinguirlos de las memorias. Todas las memorias de imágenes usadas son de 8 bits por componente, excepto HR_A que es de 9 bits, y por ese motivo se ha distinguido usando doble rayado en los bordes. Los bloques funcionales *Motion Estimation* y *Motion Compensation* ya están incluidos en el codificador híbrido de vídeo. El resto de los bloques tienen que ser añadido sobre alguno de los coprocesadores, y se ha optado finalmente por el *compressor*.

El diagrama de bloques de la FIGURA 4.17 también supone una ayuda importante para poder evaluar los recursos necesarios para una ejecución secuencial del algoritmo sobre el flujo de píxeles en formato de macro-bloques. En la TABLA 4.6 se resumen los requisitos de memoria que demandaría la implementación de esta versión del algoritmo en función del número de macro-bloques de la imagen de baja resolución, es decir, en función del número de macro-bloques de entrada. Estos datos pueden obtenerse de la TABLA 4.2 y de la TABLA 4.3. Al número de macro-bloques de las columnas se le ha denominado *mb_x* y al número de macro-bloques de las filas *mb_y*. Así por ejemplo, la memoria HR_A tendría un número de macro-

bloques *(2· mb_x) ·(2· mb_y)*, ya que al tratarse que una imagen de alta resolución su tamaño es doble en ambas direcciones. Como cada macro-bloque tiene 16×16 píxeles en luminancia y 8×8 píxeles en crominancia y además existen dos crominancias, la roja y la azul, esto supondría que el número de píxeles total es de *(2·mb_x·2·mb_y·16·16)* de luminancia y *(2·mb_x·2·mb_y·8·8·2)* de crominancia. No obstante, debe tenerse en cuenta que la memoria HR_A es de 9 bits, por lo que para obtener el número total de bits hay que multiplicar por los 9 bits que ocupa cada píxel. En el resto de los casos hay que multiplicar por 8 bits por píxel.

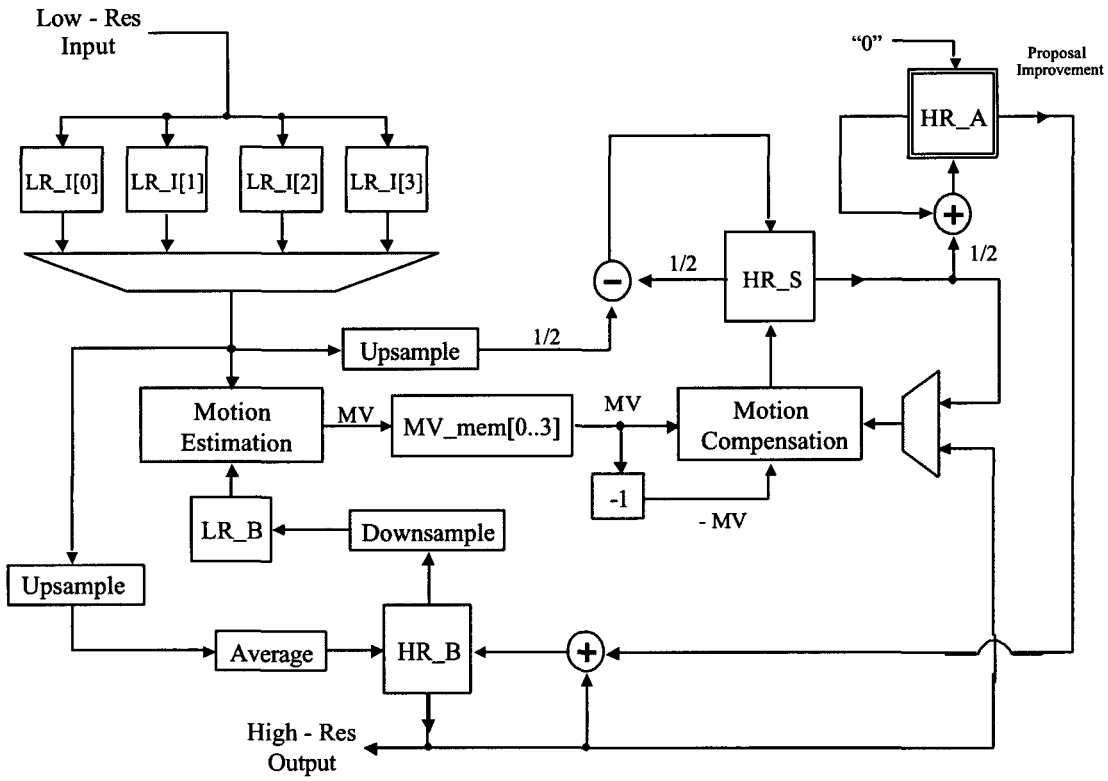


FIGURA 4.17. Diagrama de bloques del flujo de datos del algoritmo de súper-resolución v1.2.

La TABLA 4.7 muestra los requerimientos de memoria del algoritmo para los tamaños de imágenes anteriormente comentados expresados en *Kbytes* y en *Mbytes*, y en la FIGURA 4.18 se muestran estos datos de forma gráfica expresados en *Kbytes*. Puede apreciarse en la gráfica el importante aumento en los requerimientos de memoria a medida que se incrementa el tamaño de la imagen, debido a que las ecuaciones de la TABLA 4.6 varían de forma no lineal. La memoria HR_T no se ha incorporado a la lista por los motivos expuestos en la sección 4.3.7, y en su lugar se ha incluido el tamaño de las tres franjas de macro-bloques adicionales incluidas en la arquitectura *Picasso* para evitar el solapamiento de datos a la hora de realizar la compensación de movimiento.

Denominación	Memoria		
	Luminancia (bits)	Crominancia (bits)	Total (bits)
HR_A	*(2·mb_x·2·mb_y·16·16·9)*	*(2·mb_x·2·mb_y·8·8·2·9)*	*13,824·mb_x·mb_y*
HR_B	*(2·mb_x·2·mb_y·16·16·8)*	*(2·mb_x·2·mb_y·8·8·2·8)*	12,288·*mb_x·mb_y*
HR_S	*(2·mb_x·2·mb_y·16·16·8)*	*(2·mb_x·2·mb_y·8·8·2·8)*	12,288·*mb_x·mb_y*
3 Stripes HR	*(2·3·2·mb_y·16·16·8)*	*(2·3·2·mb_y·8·8·2·8)*	36,864·*mb_y*
LR_B	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
LR_I[0]	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
LR_I[1]	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
LR_I[2]	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
LR_I[3]	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
MV_mem[0]	(mb_x·mb_y·8)	0	8· *mb_x·mb_y*
MV_mem[1]	(mb_x·mb_y·8)	0	8· *mb_x·mb_y*
MV_mem[2]	(mb_x·mb_y·8)	0	8· *mb_x·mb_y*
MV_mem[3]	(mb_x·mb_y·8)	0	8· *mb_x·mb_y*
Total (bits)	mb_y· (35,872·mb_x + 24,576)	mb_y· (17,920· mb_x + 12288)	mb_y · (53,792 mb_x + 36,864)

TABLA 4.6. Resumen de la memoria utilizada por la versión v1.2 del algoritmo de súper-resolución en función del número de macro-bloques.

No obstante, debe tenerse en cuenta que el tamaño expresado en el eje de abscisas es el de la imagen de entrada en baja resolución y que la salida tiene un tamaño doble.

Tamaño	mb_x	mb_y	Memoria (Kbytes)	Memoria (Mbytes)
SQCIF	8	6	342.19	0.33
QAVGA	9	7	445.18	0.43
QCIF	11	9	690.57	0.67
HAVGA	18	14	1,717.73	1.68
CIF	22	18	2,681.30	2.62
AVGA	36	28	6,744.94	6.59
VGA	40	30	8,014.69	7.83
4CIF	44	36	10,563.19	10.32
16CIF	88	72	41,928.75	40.95

TABLA 4.7. Memoria utilizada por la versión v1.2 del algoritmo de súper-resolución para diferentes tamaños de imágenes.

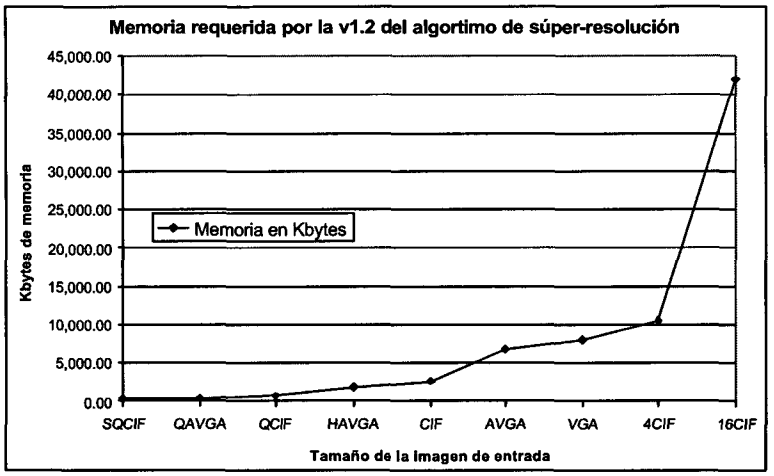


FIGURA 4.18. Memoria usada por el algoritmo de súper-resolución v1.2 para los tamaños de imágenes más comunes.

4.3.7.2 Resultados de simulación y análisis de calidad y comportamiento de la versión v1.2 del algoritmo iterativo

Aunque la versión v1.2 del algoritmo iterativo ofrece muy buena calidad perceptual de la imagen, presenta un importante problema referente a la obtención de la relación señal-ruido. En concreto, el método de obtención de la primera propuesta de imagen de súper-resolución como la media entre las imágenes de entrada provoca un desplazamiento de esta primera imagen respecto a la referencia, tal y como se muestra en la FIGURA 4.19. El proceso de sub-muestreo toma diferentes píxeles de la imagen real para formar nuevas imágenes de baja resolución. El problema es que al realizar la media de estas imágenes de baja resolución, los píxeles quedan desplazados ½ píxel respecto a la imagen real o de referencia. Este desplazamiento se mantiene a través de todo el proceso de reconstrucción lo que provoca que la relación señal-ruido caiga de forma drástica al no estar la imagen de referencia y la obtenida de súper-resolución sobre la misma rejilla.

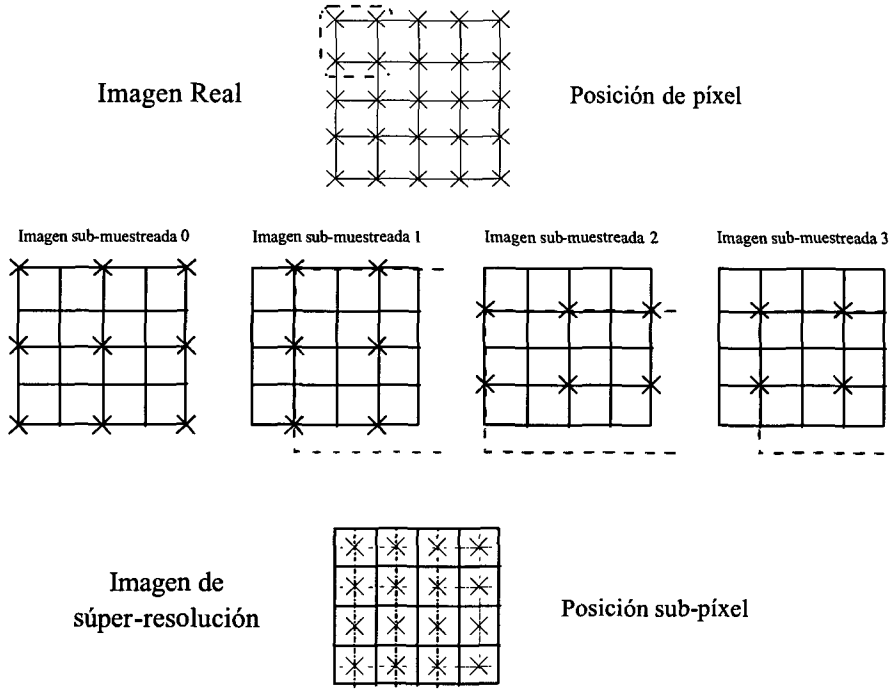


FIGURA 4.19. Efecto de desplazamiento sub-píxel al usar como primera propuesta de imagen de súper-resolución la media de las imágenes de entrada.

La solución encontrada para poder realizar medidas entre la imagen de súper-resolución y la referencia ha sido asegurar que el conjunto de vectores generado tiene media cero. De esta forma la imagen de súper-resolución resultante está ajustada a la posición (0,0) que coincide con la posición de los píxeles de la imagen de referencia. Además, esta solución ofrece la ventaja de modelar el movimiento producido durante la grabación manual y sin sistema de sujeción y/o apoyo de una secuencia de video usando una videocámara doméstica. Normalmente el ser humano intentará mantener la videocámara centrada sobre el objetivo a pesar del movimiento involuntario producido por el intento de sostener la cámara a mano. Este intento de mantener el objetivo centrado puede provocar desplazamientos relativos con media próxima a cero si el intervalo entre imágenes de entrada no es demasiado breve. Para lograr este tipo de vectores de movimiento simulado se ha desarrollado un programa que genera series de cuatro vectores aleatorios de media cero. Un ejemplo de este tipo de series es el recogido en la TABLA 4.8 donde se ha generado una secuencia de 40 vectores de movimiento que da como resultado de su aplicación 10 imágenes de salida de súper-resolución.

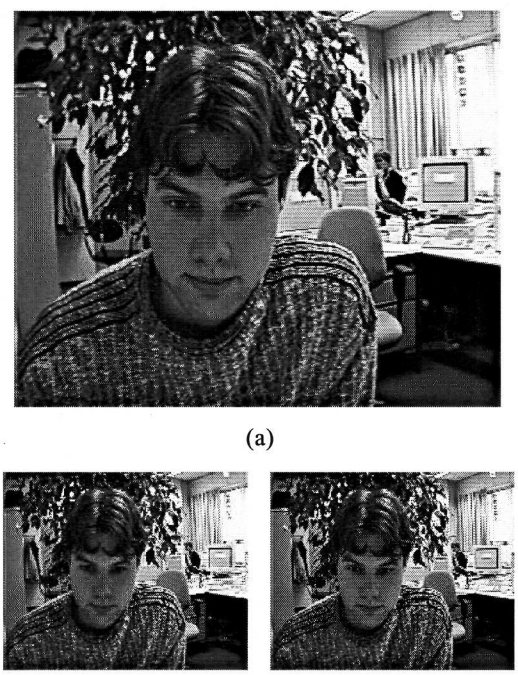
En la FIGURA 4.20 (a) se muestra la imagen de referencia en formato CIF (352×288) etiquetada como *KANTOOR*. Resulta de gran interés la existencia de un primer plano sin

demasiados detalles y un segundo plano con abundancia de detalles que se perderán en el proceso de sub-muestreo para a continuación ser recuperados gracias al proceso de súper-resolución. El primer experimento ha consistido en generar una secuencia de vectores de movimiento aleatorios con distancia de dos píxeles en la misma imagen de referencia repetida diez veces, para que tras el diezmado estas distancias quedasen reducidas a un píxel. A partir de cada imagen de referencia desplazada se obtienen por diezmado cuatro imágenes de baja resolución de tamaño QCIF (176×144). Para el caso de la primera imagen se ha generado la secuencia (b), (c), (d) y (e) de la FIGURA 4.20. Estas imágenes han sido sub-muestreadas por debajo de la frecuencia de Nyquist, por lo que contienen cierta cantidad de *aliasing*, que en el dominio del tiempo se manifiesta como una perdida de información, especialmente en los bordes (zonas de alta frecuencia). Por ejemplo, puede observarse como en las imágenes de baja resolución (b) y (c) el borde inferior del recuadro interior del monitor ha desaparecido y en todas ellas los detalles de los objetos sobre la mesa son prácticamente irreconocibles. También se pude apreciar que la calidad de las líneas rectas ha quedado degradada constatándose un fuerte efecto de pixelización en casi todos los bordes de los objetos que aparecen en la escena.

frame	Vectores de alta resolución		Vectores de baja resolución		Reducción a vectores canónicos		*frame*	Vectores de alta resolución		Vectores de baja resolución		Reducción a vectores canónicos	
0	0	-2	0	-1	0	1	5	-2	-4	-1	-2	1	0
	-2	0	-1	0	1	0		0	2	0	1	0	1
	2	0	1	0	1	0		4	0	2	0	0	0
	0	2	0	1	0	1		-2	2	-1	1	1	1
1	-2	0	-1	0	1	0	6	2	0	1	0	1	0
	0	2	0	1	0	1		0	-4	0	-2	0	0
	-2	-4	-1	-2	1	0		-2	2	-1	1	1	1
	4	2	2	1	0	1		0	2	0	1	0	1
2	-2	0	-1	0	1	0	7	0	2	0	1	0	1
	-2	-2	-1	-1	1	1		-2	-2	-1	-1	1	1
	0	-2	0	-1	0	1		2	2	1	1	1	1
	4	4	2	2	0	0		0	-2	0	-1	0	1
3	2	2	1	1	1	1	8	2	2	1	1	1	1
	2	0	1	0	1	0		-2	-2	-1	-1	1	1
	0	-2	0	-1	0	1		0	0	0	0	0	0
	-4	0	-2	0	0	0		0	0	0	0	0	0
4	0	0	0	0	0	0	9	2	-2	1	-1	1	1
	2	2	1	1	1	1		2	2	1	1	1	1
	0	-2	0	-1	0	1		-2	0	-1	0	1	0
	-2	0	-1	0	1	0		-2	0	-1	0	1	0

TABLA 4.8. Vectores de movimiento aleatoriamente generados con media cero, en distancias de dos píxeles para alta resolución, de un píxel para baja resolución y su reducción a vectores canónicos.

El conjunto de vectores de movimiento globales aplicado es el mostrado en la TABLA 4.8. En la primera columna se muestra el número de *frame* generado en el proceso de súper-resolución. En la segunda columna se muestra el conjunto de vectores que, aplicados sobre el *frame* de referencia, han servido para realizar la reconstrucción de la imagen de salida. La tercera columna muestra los vectores de baja resolución, es decir, dividiendo por dos sus componentes x e y. La cuarta columna muestra la reducción de los vectores al conjunto de vectores canónicos antes definidos. El significado de esta reducción se dará más adelante cuando se realice el análisis de los resultados, en especial el análisis de la PSNR.



(a)

(b) (c) (d) (e)

FIGURA 4.20. Imágenes usadas para el algoritmo iterativo. La imagen (a) de tamaño CIF es la referencia. A partir de ella se obtienen por diezmado las imágenes (b)-(e) de baja resolución y tamaño QCIF, que constituyen el primer conjunto de entrada al algoritmo de súper-resolución.

4.3.7.3 Análisis de la calidad en el dominio espacial

En la Figura 4.21, FIGURA 4.22 y FIGURA 4.23 se muestra el resultado de ejecutar la versión v1.2 del algoritmo de súper-resolución sobre los datos de entrada comentados (40 imágenes de baja resolución) después de efectuar tan solo dos iteraciones en el bucle del algoritmo. La salida son diez imágenes de súper-resolución de tamaño doble que las de

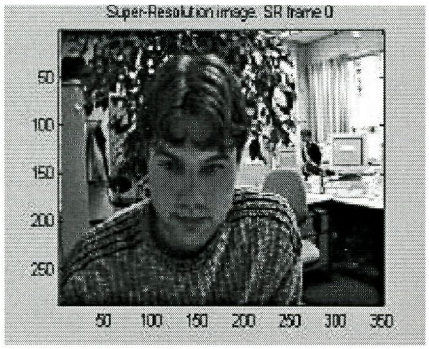
entrada. Asimismo se muestra junto con las imágenes su error asociado en el dominio del tiempo. Este error se obtiene como la resta entre la imagen de referencia (antes de realizar el sub-muestreo) y cada una de las imágenes de súper-resolución de salida. El nivel cero se ha desplazado al gris, lo que significa que cuanto mayor cantidad de color gris contenga la imagen, tanto mejor será la imagen de súper-resolución, en tanto en cuanto ambas imágenes, la de referencia y la de súper-resolución, serán más parecidas.

Lo primero que puede apreciarse a simple vista es una importante degradación de la imagen de salida (h) correspondiente al *frame* número 7. Es deseable cuantificar el grado de similitud y por tanto de calidad de la imagen de súper-resolución obtenida. Para ello, en el dominio del tiempo, usaremos la relación señal-ruido de pico (PSNR, *Peak Signal to Noise Ratio*), definida como indica la ecuación (4.14), donde *R(i,j)* es la imagen de referencia y *SR(i,j)* es la imagen de súper-resolución, teniendo en cuanta que el tamaño de la imagen es de MxN píxeles.

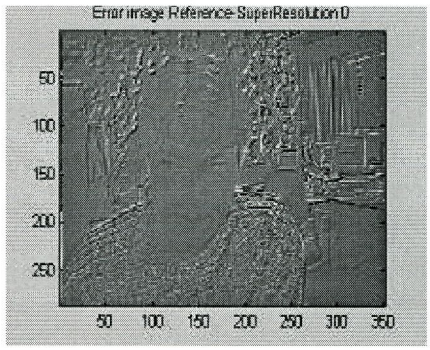
$$PSNR = \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left(\frac{255^2}{(R(i,j) - SR(i,j))^2} \right) \tag{4.14}$$

El valor 255 proviene del máximo valor que puede tomar un píxel cuando éste es almacenado en palabras de un *byte*, es decir 2^8-1. La relación señal-ruido es una de las medidas más usuales cuando se trata de medir la calidad de una señal y por este motivo ha sido la métrica adoptada para imágenes en el dominio temporal.

Cuando obtenemos una imagen de mayor tamaño a partir de una imagen menor, como estamos haciendo al aplicar súper-resolución, se podría presentar la objeción de que se podría conseguir el mismo objetivo realizando una simple interpolación. Entre los diferentes métodos de interpolación existentes [PKT83],[Jai89],[Pra91] hemos escogido el de la copia del píxel vecino más próximo (*nearest neighbour interpolation*) o interpolación de orden cero, por tratarse del método más sencillo y rápido en aplicaciones de tiempo real y el método de interpolación bilineal por tratarse de un método que con poca complejidad adicional relativa consigue mejoras sustanciales en la calidad de la imagen interpolada. A modo de referencia las imágenes interpoladas usando cada uno de los métodos mencionados así como sus errores asociados se muestran en la FIGURA 4.24.

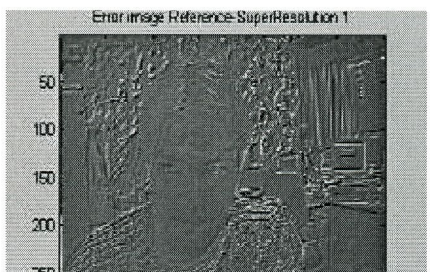


(a.1)

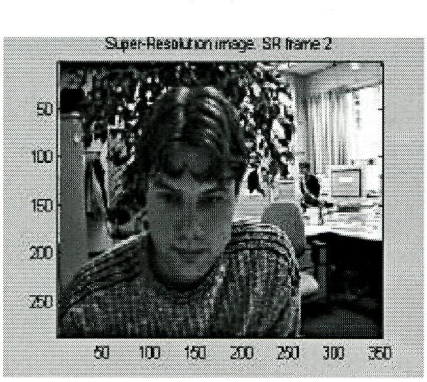


(a.2)

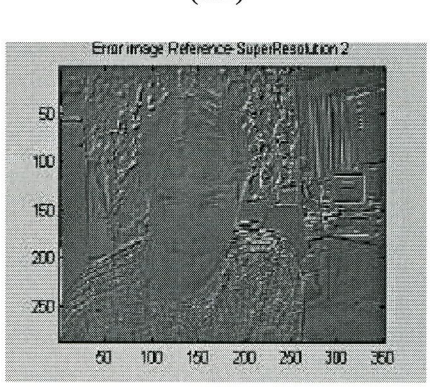
(b.1)



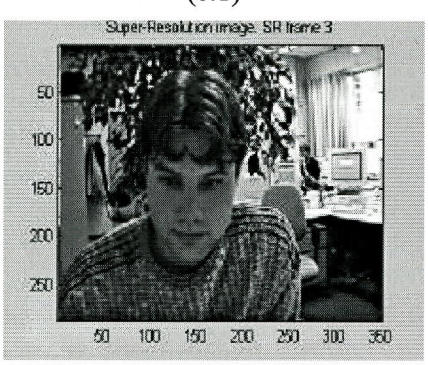
(b.2)



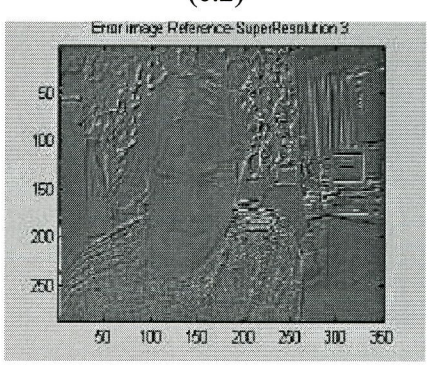
(c.1)



(c.2)

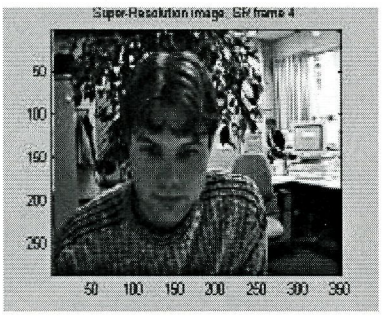


(d.1)

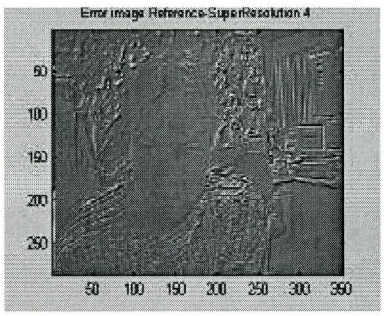


(d.2)

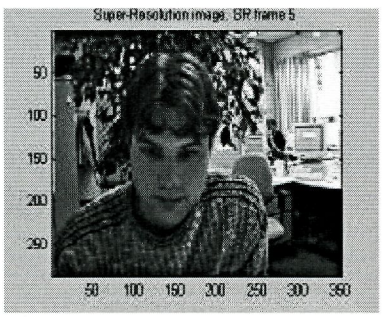
Figura 4.21. Imágenes de tamaño CIF obtenidas al aplicar el algoritmo de súper-resolución. La secuencia (a)-(d) corresponde a los *frames* 0 al 3 respectivamente.



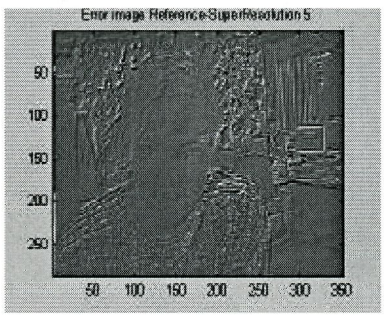
(e.1)



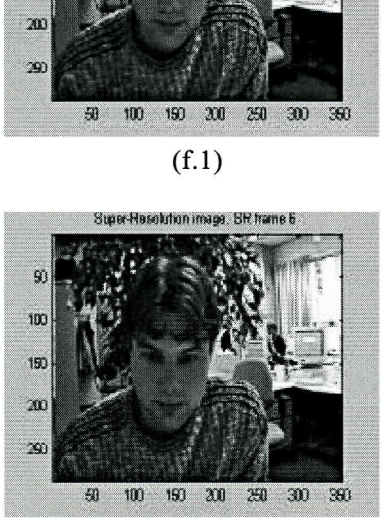
(e.2)



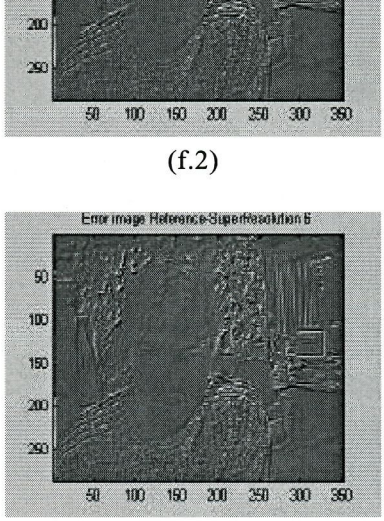
(f.1)



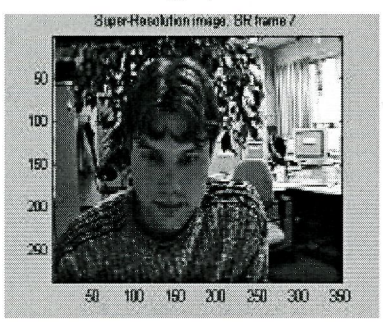
(f.2)



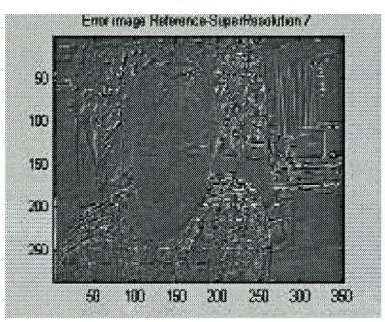
(g.1)



(g.2)

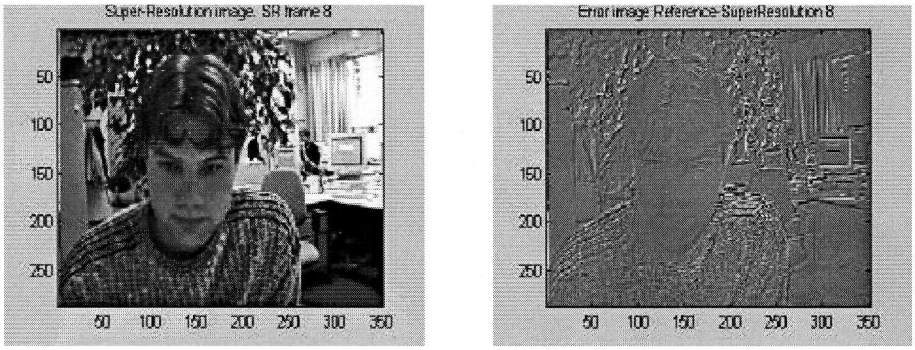


(h.1)



(h.2)

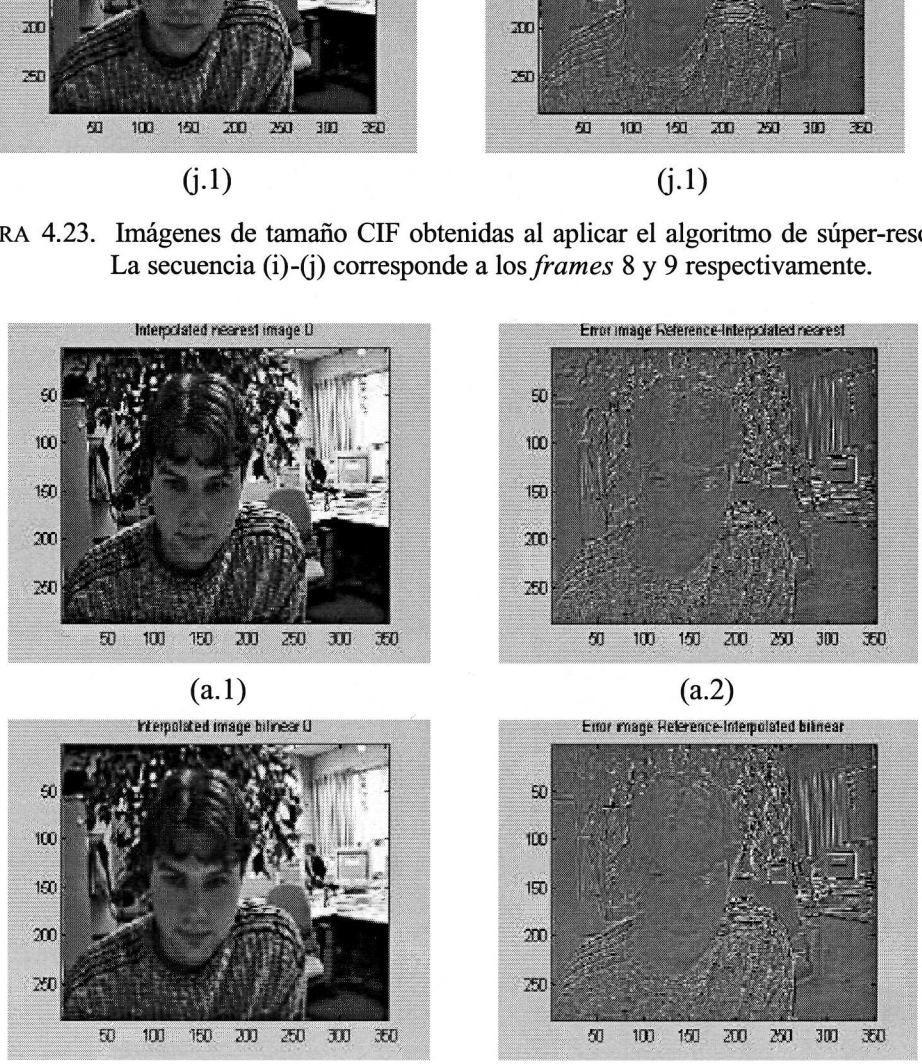
FIGURA 4.22. Imágenes de tamaño CIF obtenidas al aplicar el algoritmo de súper-resolución. La secuencia (e)-(h) corresponde a los *frames* 4 al 7 respectivamente.



(i.1) (i.1)

(j.1) (j.1)

FIGURA 4.23. Imágenes de tamaño CIF obtenidas al aplicar el algoritmo de súper-resolución. La secuencia (i)-(j) corresponde a los *frames* 8 y 9 respectivamente.



(a.1) (a.2)

(b.1) (b.2)

FIGURA 4.24. Imágenes de tamaño CIF obtenidas mediante interpolación por copia del píxel vecino más próximo (a.1) y por interpolación bilineal (b.1) así como sus errores asociados.

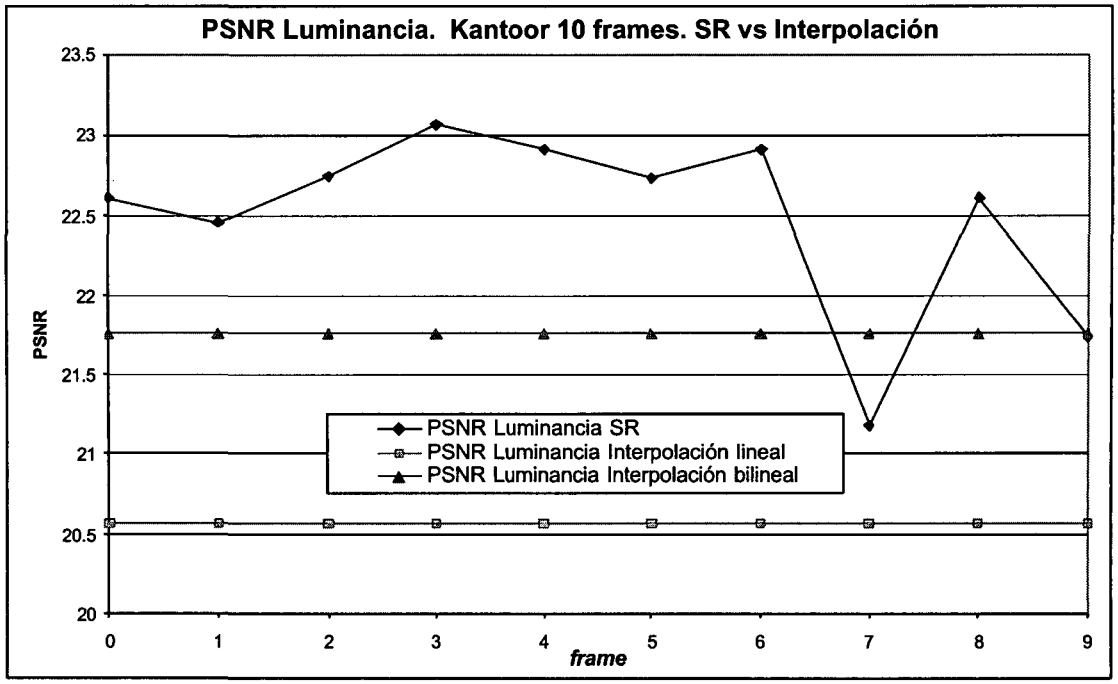


FIGURA 4.25. PSNR de la secuencia de salida de súper-resolución frente a las imágenes interpoladas usando interpolación lineal de orden cero e interpolación bilineal.

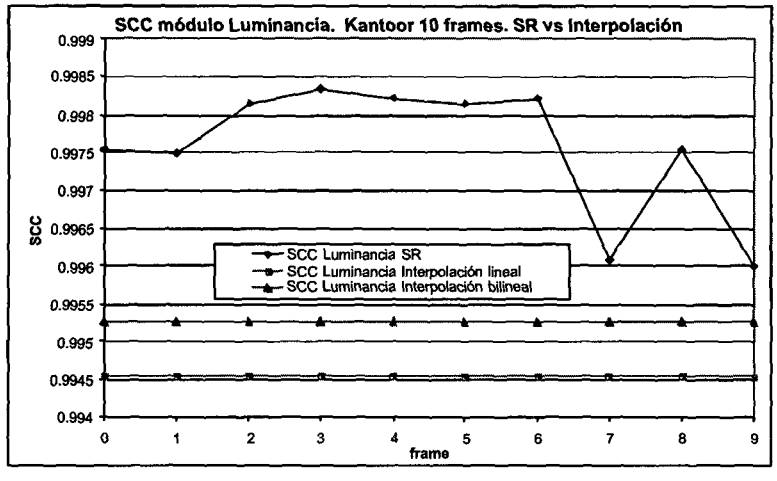
En la FIGURA 4.25 se muestra la PSNR obtenida para cada imagen de la secuencia de salida junto con la PSNR obtenida para las imágenes interpoladas. A la PSNR obtenida para la imagen con interpolación bilineal le llamaremos 'nivel de interpolación' y resulta de gran ayuda ya que establece el nivel por encima del cual la súper-resolución es realmente útil. Si la imagen de salida está por debajo del nivel de interpolación, quiere decir que con una simple interpolación se hubiese logrado mayor calidad que aplicando súper-resolución, no mereciendo la pena ésta última. En todo caso, la obtención de PSNR inferiores al nivel de interpolación suelen ser síntomas de algún tipo de error, bien en el proceso de generación o en el de análisis de los resultados. La diferencia entre la PSNR obtenida usando súper-resolución y la PSNR lograda por interpolación nos da la ganancia de calidad sobre el uso de interpoladores.

4.3.7.4 Análisis de calidad en el dominio de la frecuencia espacial

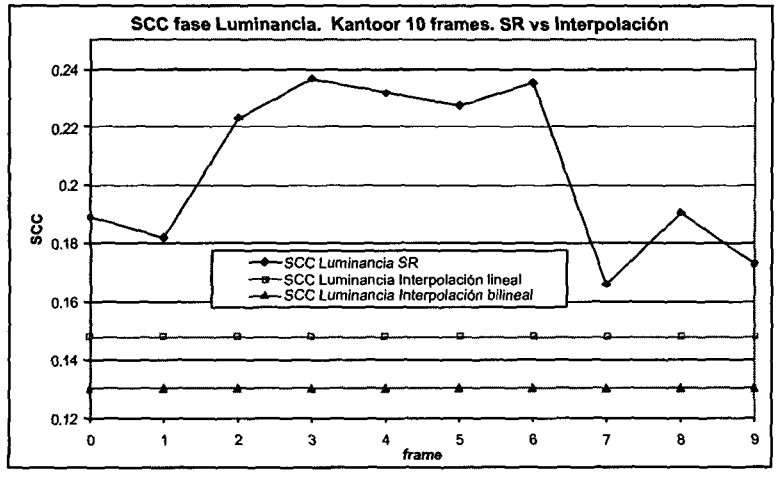
La primera métrica usada para cuantificar el grado de similitud entre la imagen de referencia y la obtenida por el proceso de súper-resolución es la relación señal-ruido de pico, cálculo que se realiza en el dominio espacial de la señal. Sin embargo, también aporta valiosa

información el grado de similitud en el dominio de la frecuencia (frecuencia espacial). Por ello y con el objetivo de medir el grado de similitud entre los espectros de las imágenes, calculamos el coeficiente de correlación bidimensional entre la imagen de referencia y la imagen resultante del proceso de súper-resolución. Debido a que las transformadas bidimensionales de Fourier son valores complejos, ésto nos dará una correlación para el módulo y otra para la fase. En la FIGURA 4.35 se muestran los valores de los coeficientes de correlación espectral (SCC, *Spectral Correlation Coefficient*) bidimensionales del módulo y de la fase para la secuencia de salida que estamos tratando.

Como se ve en la figura, la correlación espectral en módulo es sustancialmente mayor que en fase, debido fundamentalmente a que la fase recoge más cantidad de información sobre el desplazamiento relativo entre las imágenes, y al ser la imagen de súper-resolución una composición de diferentes imágenes desplazadas entre si, la composición final queda



(a)



(b)

FIGURA 4.26. Coeficientes de correlación espectrales en módulo (a) y fase (b).

ligeramente desplazada respecto a la referencia. Por este mismo motivo la correlación espectral en módulo de la interpolación bilineal es mayor que la correlación de la interpolación lineal. Sin embargo ocurre todo lo contrario con la correlación espectral en fase. El rellenar los valores intermedios de la imagen obtenida por interpolación bilineal con valores promedio, equivale a rellenarlos con versiones desplazadas de la imagen, lo que se refleja en una menor correlación espectral en fase. Sin embargo, la correlación espectral en módulo, a la que el ojo humano es mucho más sensible al estar relacionada con variaciones de luminosidad, obtiene valores de correlación muy altos, en todos los casos por encima del 0.996, llegando a 0.998 en algunos casos.

Otro aspecto que merece la pena destacar es el comportamiento similar que siguen las métricas de calidad usadas: obsérvese que los dos primeros *frames* (0 y 1) muestran calidades similares en los tres casos (PSNR, SCC en módulo y SCC en fase), y siempre inferiores a las cinco siguientes *frames* (2, 3, 4, 5 y 6). El *frame* 8 muestra una calidad similar a los *frames* iniciales y en los *frames* 7 y 9 la calidad baja considerablemente.

Si bien la información que aportan las transformadas bidimensionales de Fourier en módulo y fase sobre la calidad de la imagen resultante es difícil de evaluar a simple vista, sí resulta mucho más enriquecedora la observación de las imágenes de error entre los espectros de las imágenes de referencia y los espectros de las señales interpoladas y de súper-resolución. En la FIGURA 4.27 se muestran las transformadas de Fourier bidimensionales en módulo (a) y fase (b) de la imagen de referencia.

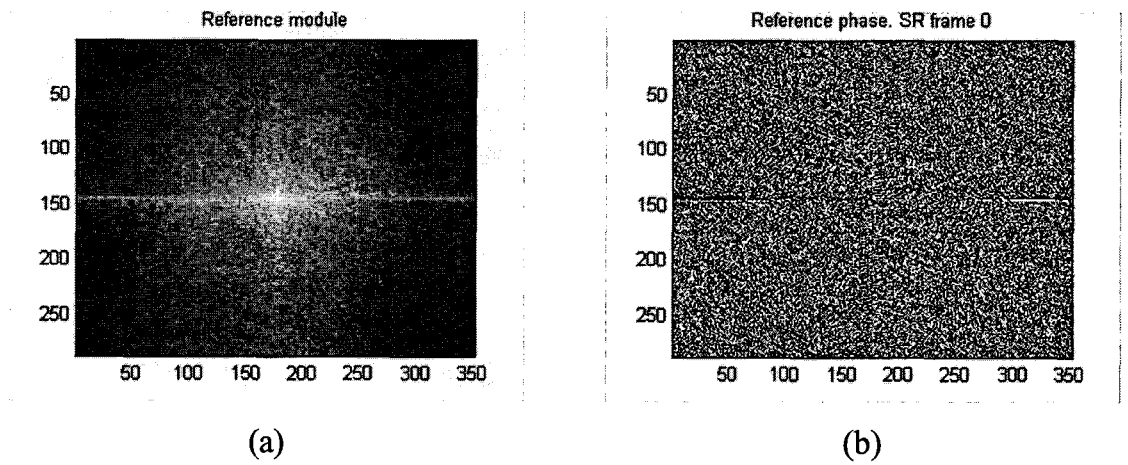


FIGURA 4.27. Transformadas de Fourier bidimensionales en módulo (a) y fase (b) de la imagen de referencia.

Estas imágenes son las que sirven de referencia para mostrar el error cometido en módulo y fase en las imágenes interpoladas y de súper-resolución. En la FIGURA 4.28 se muestra el módulo y el error en módulo de las imágenes interpoladas y en la FIGURA 4.29 se muestra la fase y el error en fase de estas mismas imágenes interpoladas.

En el caso de la FIGURA 4.28 puede apreciarse cómo en ambos casos el error cometido es menor en la zona de las bajas frecuencias, donde se observa una zona central bastante homogénea de bajo error. En el caso de las fases representadas en la FIGURA 4.29 es claro que esta zona homogénea es de menor extensión., lo que es coherente con la menor correlación exhibida. Como es normal en las transformadas de Fourier de las imágenes, las fases aportan poca información visual, y aunque normalmente no son tenidas en cuenta, aquí se ha decidido mantenerlas por dos razones: en primer lugar, por completitud y en segundo lugar porque evaluando el error existente obtenemos información sobre el grado de similitud entre las fases de las imágenes, lo que resulta importante en este tipo de aplicaciones.

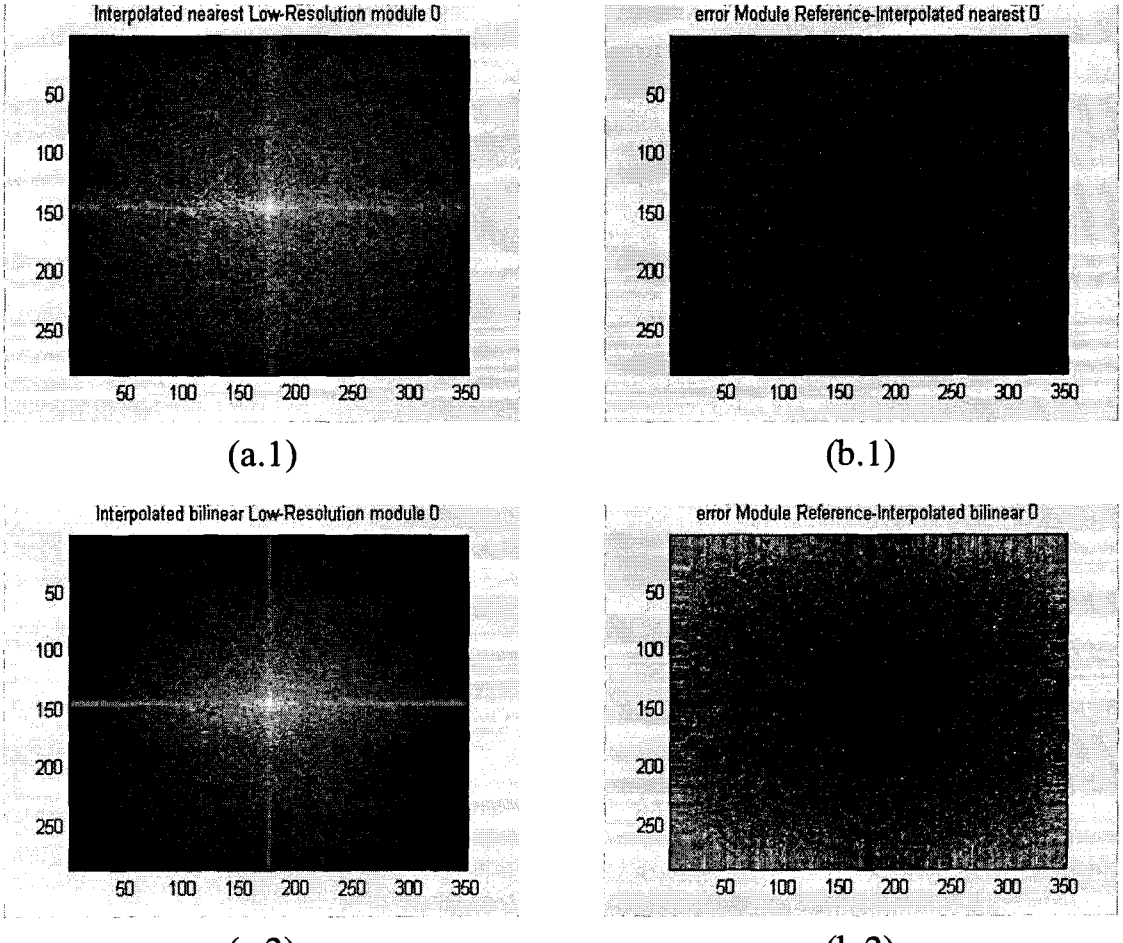


FIGURA 4.28. Transformadas de Fourier bidimensionales en módulo (a) y su error asociado (b) para las imágenes interpoladas.

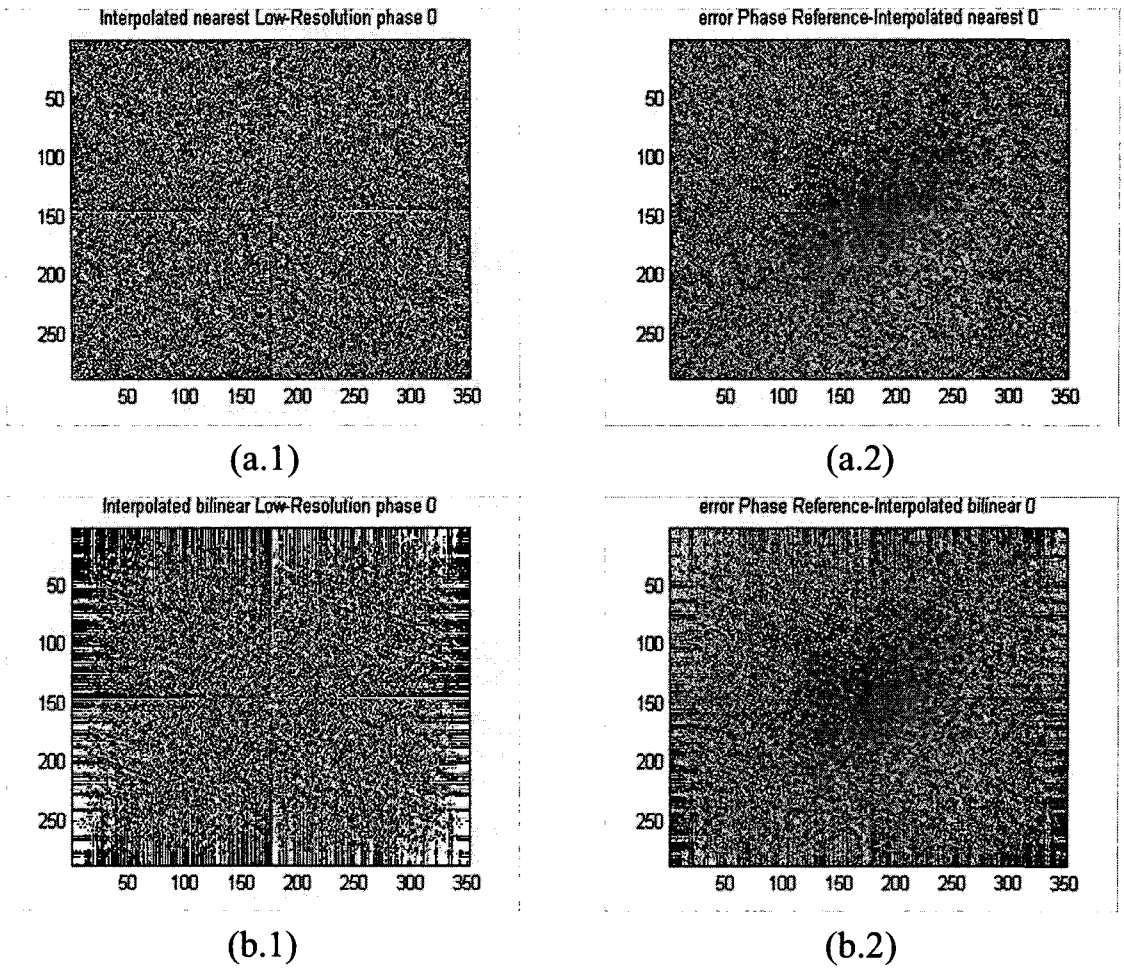
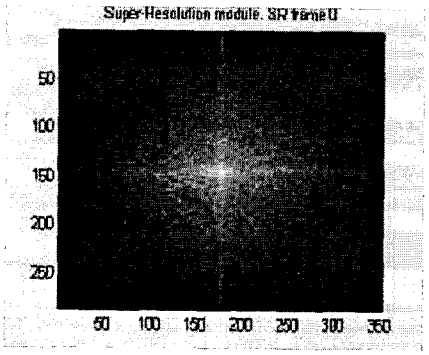
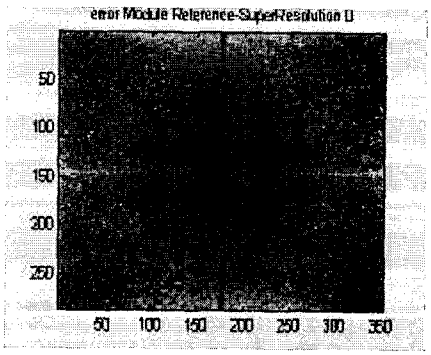


FIGURA 4.29. Transformadas de Fourier bidimensionales en fase (a) y su error asociado (b) para las imágenes interpoladas.

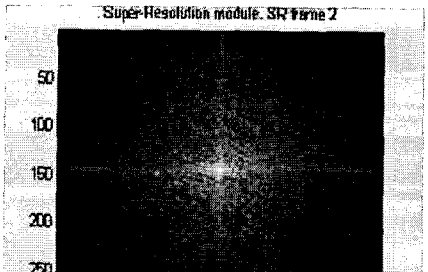
En la FIGURA 4.30 se muestran las transformadas de Fourier bidimensionales en fase y su error asociado para algunos *frames* de la secuencia de súper-resolución. En concreto, se muestran los *frames* 0, 2, 7 y 9 por contener las variaciones más bruscas tanto en la PSNR como en las correlaciones espectrales. Puede verse que, en la zona de bajas frecuencias, el *frame* 2 tiene un error menor que el resto, lo que es coherente con todas las métricas obtenidas. Resulta muy curioso constatar que el *frame* 7 tiene menor error en la zona de frecuencias horizontales, mientras que el 9 presenta un comportamiento similar, pero en la zona de frecuencias verticales. El análisis de la fase de estas mismas imágenes, mostrado en la FIGURA 4.31, ofrece el mismo comportamiento que el módulo. Este fenómeno nos ha dado una pista importante para el desarrollo de las consideraciones que se hacen en las secciones siguientes sobre las variaciones de la calidad de las imágenes de súper-resolución.



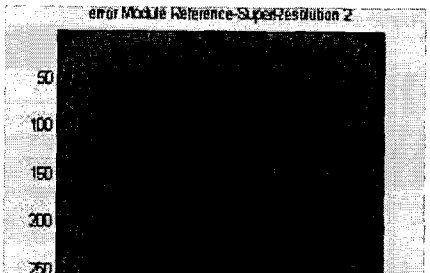
(a.1)



(a.2)



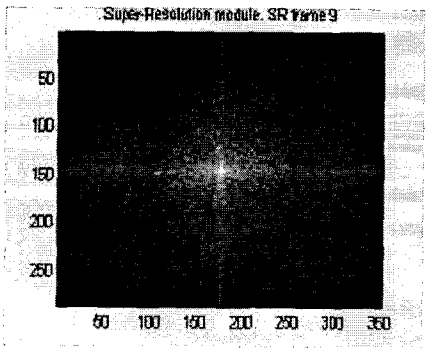
(b.1)



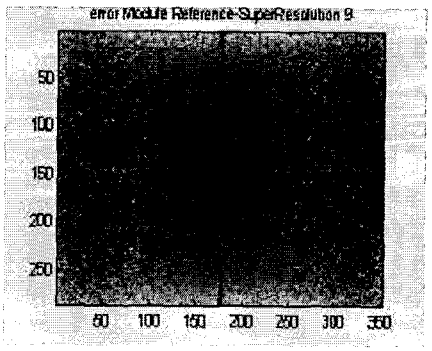
(b.2)

(c.1)

(c.2)



(d.1)



(d.2)

FIGURA 4.30. Transformadas de Fourier bidimensionales en módulo y su error asociado para las imágenes de súper-resolución 0 (a) , 2 (b) , 7 (c) y 9 (d).

4.3.7.5 Efectos de los bordes

Como ya se había observado con anterioridad, el *frame* 7 experimenta un drástico descenso en su calidad, bajando incluso por debajo del nivel de interpolación. Otro hecho importante es que, tratándose siempre de la misma imagen de referencia de entrada, las calidades obtenidas son muy diferentes, en contra de lo esperado. Dado qué el *frame* de entrada es siempre el mismo, cabía esperar en todos los casos una calidad de salida similar, pero al no ser así, este hecho nos da una indicación de que los movimientos de la imagen influyen de forma importante en la calidad final resultante. Un análisis más detallado de las imágenes de error nos muestra que una de las principales fuentes de error reside en los bordes de la imagen. Ya se ha comentado que el efecto de borde podía hacer caer de forma drástica la PSNR. AL hacer estas medidas y análisis cuantitativo este efecto se hace muy patente patente. Una forma sencilla de eliminar este problema del proceso de medición consistió en redefinir el cálculo de la relación señal-ruido de pico, excluyendo los bordes de la imagen en un cierto número de píxeles 'q'. Inicialmente se establece este factor de recorte 'q' igual a 16 píxeles, es decir, un macro-bloque. La nueva relación señal-ruido de pico, que denominaremos PSNR' o PSNR sin bordes, viene dada por (4.15).

$$PSNR' = \frac{1}{(M-2\cdot q)\cdot(N-2\cdot q)} \sum_{i=q}^{M-1-q} \sum_{j=q}^{N-1-q} \left(\frac{255^2}{\left(R(i,j)-SR(i,j)\right)^2} \right) \tag{4.15}$$

Si representamos ahora la PSNR' para la misma secuencia, se obtiene la gráfica mostrada en la FIGURA 4.32, donde se aprecia tal y como se esperaba una menor variación de la PSNR'. Sin embargo, también podemos observar cómo varios *frames* comparten la misma PSNR', como es el caso de los *frames* 0,1 y el de los *frames* 2,3,4,5 y 6. La explicación de este fenómeno originará una de las principales limitaciones físicas en la mejora de la calidad de imágenes usando técnicas de súper-resolución.

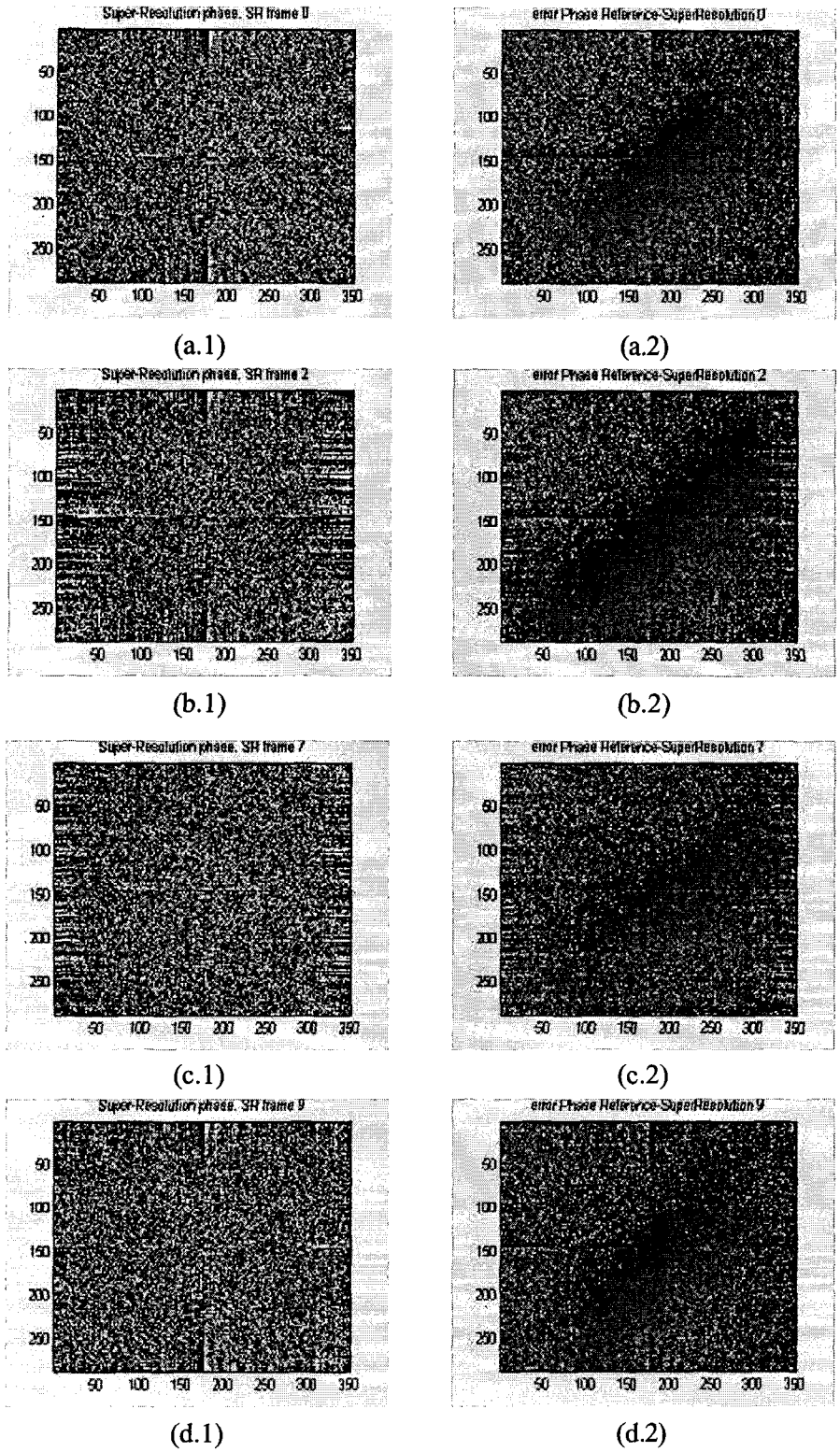


FIGURA 4.31. Transformadas de Fourier bidimensionales en fase y su error asociado para las imágenes de súper-resolución 0 (a) , 2 (b) , 7 (c) y 9 (d).1.

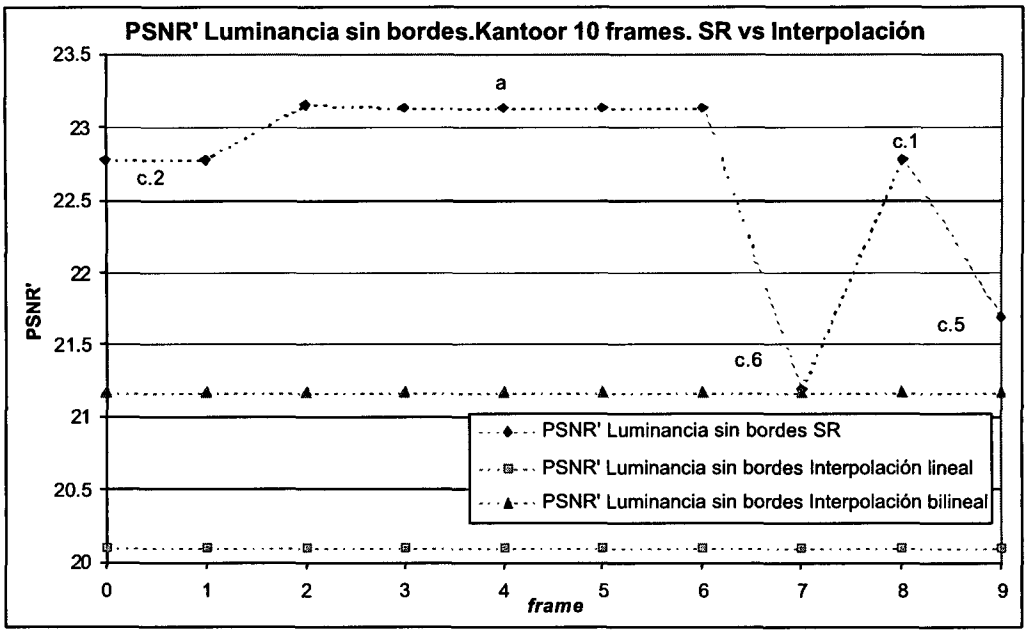


FIGURA 4.32. PSNR' sin bordes de la secuencia de salida de súper-resolución frente a las imágenes interpoladas usando interpolación lineal de orden cero y bilineal.

4.3.7.6 Efecto de los vectores de desplazamiento en la calidad

Obviamente, las diferencias de calidad tienen que residir en las diferencias de los desplazamientos entre las imágenes de entrada, ya que el resto de los factores permanece igual. Si analizamos en detalle el proceso de sub-muestreo, observamos, como se muestra en la FIGURA 4.33, que vectores de desplazamiento en apariencia totalmente diferentes pueden generar muestreos equivalentes en el cuerpo principal de la imagen, es decir, excluyendo los efectos de borde.

En dicha figura puede observarse cómo una misma imagen es desplazada usando los vectores de desplazamiento (0,1) y (2,-3) y luego es sub-muestreada. El sub-muestreo consiste en seleccionar los píxeles (representados como cruces) encerrados en un circulo. Se ha señalado con líneas pespunteadas lo que serían los bordes de la nueva imagen sub-muestreada. Como puede apreciarse, los píxeles seleccionados (excepto en los bordes) son exactamente los mismos, por lo que el segundo desplazamientos no aportaría nueva información con respecto al primero. Aunque partamos siempre de cuatro desplazamientos,

diferentes en apariencia, es obvio que si existen desplazamientos equivalentes entre ellos, la calidad de la imagen se verá deteriorada respecto a otro conjunto de desplazamientos en el que cada uno de ellos aporte información nueva que permita mejorar la calidad de los detalles de la imagen original.

Debido a que el sistema de sub-muestreo que estamos utilizando consiste en tomar alternativamente píxeles de la imagen original, partiendo del desplazamiento generado, es claro que cualquier desplazamiento se podrá reducir a la celda base que se describió en la FIGURA 4.2, o lo que es lo mismo, todo vector de desplazamiento podrá ser reducido a su vector canónico. Si dos vectores de desplazamiento pueden ser reducidos al mismo vector canónico, entonces la información que aportan es la misma (excepto en los bordes). El vector canónicos $(\tilde{x},\tilde{y})$, del vector de desplazamiento de coordenadas *(X,Y)* se obtiene como indica la ecuación (4.16).

$$\begin{aligned} \tilde{x} &= |X \bmod 2| \\ \tilde{y} &= |Y \bmod 2| \end{aligned} \tag{4.16}$$

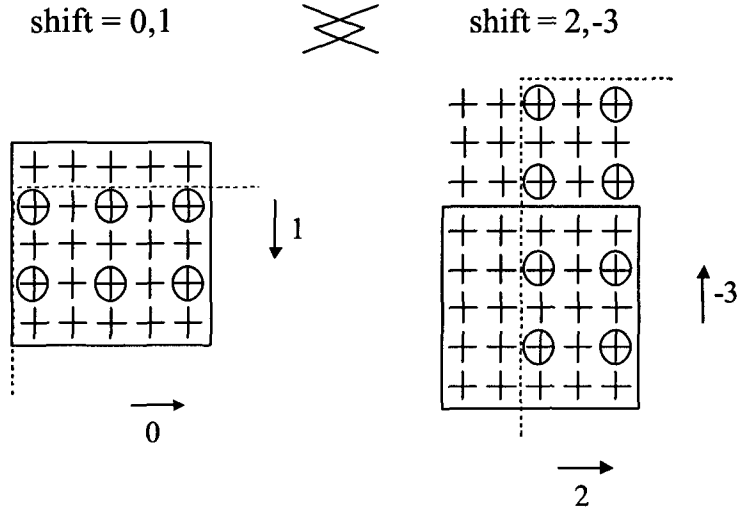


FIGURA 4.33. Muestra del efecto de muestreo equivalente usando vectores de desplazamiento equivalentes.

En la TABLA 4.8 se muestra la reducción a vectores canónicos del conjunto de vectores aleatoriamente generado. Analizando nuevamente la TABLA 4.8 con los vectores de desplazamiento generados, donde se incluyeron asimismo la reducción a vectores canónicos, podemos constatar cómo los *frames* 0 y 1 tienen los mismos vectores canónicos [(0,0),(1,1)] y por lo tanto la misma PSNR sin bordes: 22.78 dB. Lo mismo ocurre en el caso de los *frames*

2,3,4,5 y 6, que comparten los vectores canónicos [(0,0),(0,1),(1,0),(1,1)], que completan toda las posiciones de la celda base y dan por lo tanto la máxima PSNR, en este caso 23.13 dB.

En base al número y posición de los píxeles de la celda base presentes tras el proceso de sub-muestreo, o lo que es lo mismo, en función de los vectores canónicos a partir de los cuales fue reconstruida, establecemos la clasificación que se ilustra en la FIGURA 4.34. Como puede apreciarse, la primera clasificación, etiquetada con una letra minúscula, atiende al número de píxeles diferentes que fueron muestreados de la imagen original, y la segunda sub-clasificación, atiende a la posición de estos píxeles en la celda base. Así por ejemplo, una imagen tipo 'a' esta formada a partir de los cuatro píxeles de la imagen original, por lo que siempre tendrá la mayor SNR sin bordes. En el caso opuesto, las imágenes de tipo 'd' han sido reconstruidas usando un solo píxel de la imagen original, y su calidad será evidentemente menor. Retomando el problema del *frame* 7, vemos que, según esta clasificación, sería una imagen de tipo 'c.6', es decir, ha sido generada a partir de tan solo dos píxeles de la imagen original, pero de los dos píxeles inferiores, por lo que se ha perdido toda la información horizontal de algunas líneas, dando una calidad inferior a la de las imágenes tipo 'c.1' (*frame* 8) o tipo 'c.2' (*frames* 0 y 1) que, aunque también han sido generadas sólo a partir de dos píxeles, estos corresponden a posiciones alternas en los bordes de la celda base, lo que permite recoger más información de la imagen original. El caso del *frame* 9, de tipo 'c.5', es

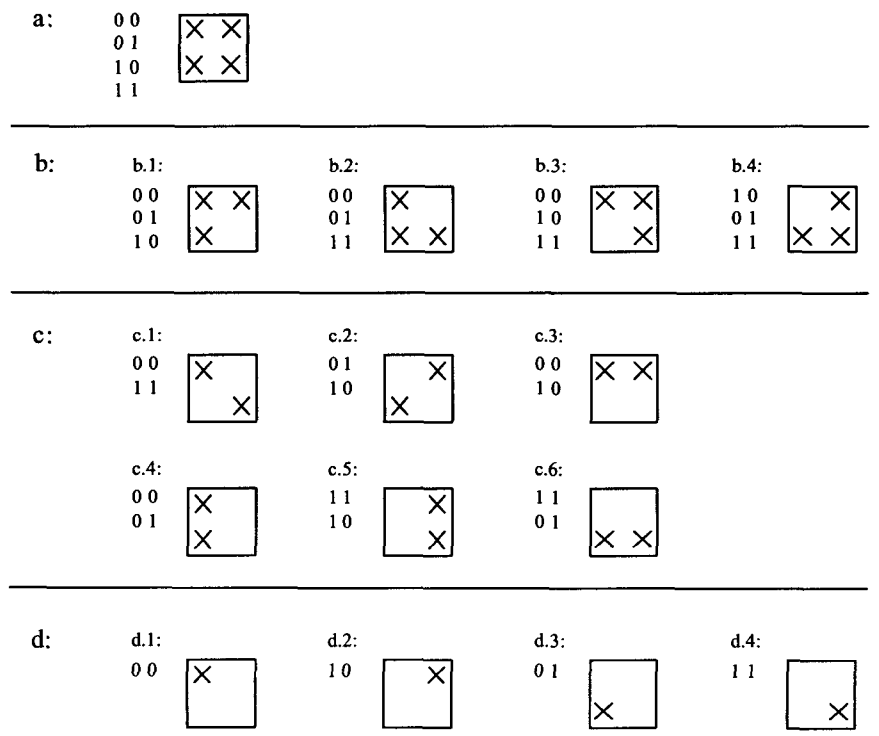


FIGURA 4.34. Clasificación de las imágenes de súper-resolución en función de los vectores canónicos a partir de los cuales fue reconstruida.

similar al del *frame* 7, pero en este caso los píxeles están en posición vertical, originando una nueva caída de la PSNR'. En la FIGURA 4.32 se ha marcado cada *frame* con la etiqueta correspondiente a la clasificación establecida.

4.3.7.7 Evaluación con el número de iteraciones en el bucle del algoritmo

La PSNR alcanzada por el *frame* 7 coincide prácticamente con el nivel de interpolación bilineal, debido a que se ha realizado un número de iteraciones reducido. Si el número de iteraciones se elevase, el algoritmo trataría de asemejar el resultado al conjunto de muestras disponibles, y al no estar algunas de las muestras disponibles, estos datos dejarían progresivamente de ajustarse, ya que no influyen directamente en la minimización del error entre la imagen de súper-resolución y cada una de las imágenes de baja resolución. Esto se refleja cualitativamente como la aparición de rayas o puntos distorsionados en la imagen, y cuantitativamente como un drástico descenso de la PSNR.

En la FIGURA 4.35 puede verse la evolución de la PSNR para el mismo conjunto de vectores de desplazamiento durante 80 iteraciones por imagen de salida. Hasta ahora sólo se había representado la PSNR final de la imagen de súper-resolución obtenida, sin embargo, en esta figura se representa la PSNR obtenida en cada iteración. Los resultados obtenidos se ajustan perfectamente a los resultados esperados en el sentido de que las imágenes tipo 'a' tienden a aumentar su calidad a medida que aumenta el número de iteraciones, mientras que el resto de las imágenes disminuirá su calidad con las iteraciones debido al fenómeno antes comentado de ajuste a los datos ausentes. También era de esperar la menor PSNR de las imágenes del tipo 'c.5' y 'c.6' frente a las del tipo 'c.1' y 'c.2' debido a la distribución irregular del muestreo.

En la FIGURA 4.37 y en la FIGURA 4.36 se muestran los resultados de las imágenes de súper-resolución obtenidas después de 80 iteraciones. Como se hizo anteriormente, sólo mostraremos aquellos *frames* que resultan de interés, es decir, los *frames* 0 (tipo c.2) , 4 (tipo a) , 7 (tipo c.6) y 9 (tipo c.5).

Obsérvese, sobre todo en las imágenes de error, cómo éste es menor en las imágenes tipo 'a' que en el resto, al estar presentes en el proceso de reconstrucción todas las muestras originales de la imagen de referencia. Incluso ante la ausencia de la mitad de las muestras originales, como es el caso del *frame* 1, de tipo 'c', el proceso de reconstrucción ofrece

buenos resultados si se dispone de al menos una muestra de cada línea. Sin embargo, si ambas muestras pertenecen a la misma línea, la calidad de la imagen se verá seriamente disminuida, apareciendo como puede apreciarse en la FIGURA 4.36 líneas horizontales o verticales en función de los datos ausentes.

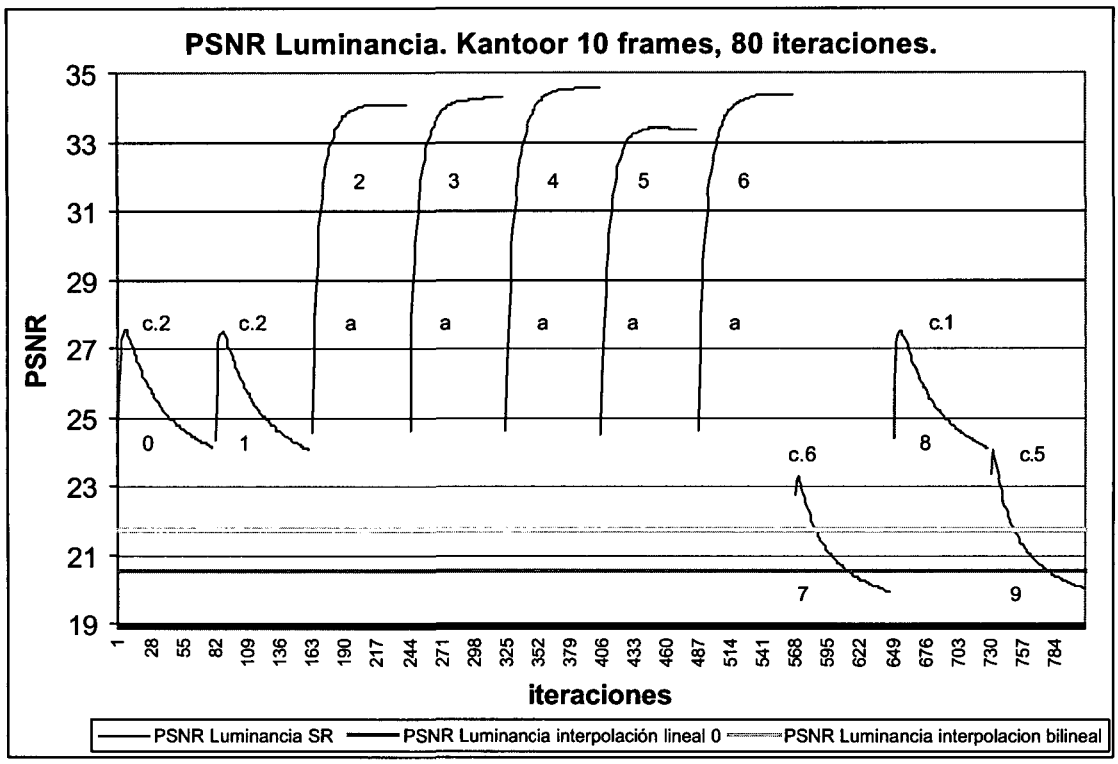


FIGURA 4.35. Evolución de la PSNR durante 80 iteraciones de la secuencia de salida de súper-resolución frente a las imágenes interpoladas usando interpolación lineal de orden cero y bilineal.

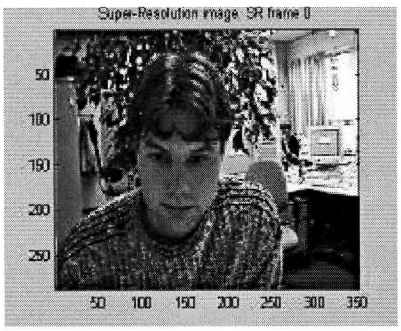
Es interesante resaltar que la calidad de las imágenes hubiese resultado mejor en el caso de haber interrumpido el proceso iterativo en las primeras etapas, tal y como puede verse en la FIGURA 4.35, donde se aprecia que, excepto para el caso de las imágenes tipo 'a', el resto alcanza un máximo en la PSNR en las primeras iteraciones para luego ir disminuyendo. Esto se debe a que en las primeras iteraciones los valores intermedios se rellenan con valores de las imágenes promedio aumentando la PSNR, pero a medida que aumenta el número de iteraciones, y con el objetivo de disminuir el error entre la imagen de súper-resolución y el conjunto de imágenes de entrada disponibles, las muestras ausentes no participan de igual forma en el proceso de súper-resolución, quedando mal ajustadas. En la FIGURA 4.38 y en la FIGURA 4.39 se muestran el módulo y la fase así como sus errores asociados respecto a la referencia.

En la FIGURA 4.38 se puede constatar que en el caso de las figuras tipo 'a' (b.2), el error espectral en módulo es muy bajo para casi todo el rango de frecuencias, especialmente para las bajas frecuencias. Para las imágenes tipo 'c.1' y 'c.2' (a.2) el error es también muy bajo, aunque no llega a los niveles de penetración en las altas frecuencias del caso anterior. Tal y como cabía esperar, los espectros en módulo de las imágenes tipo 'c.5' y 'c.6' son peores que los casos anteriores, y el error mínimo está claramente orientado en el sentido horizontal (c.2) y vertical (d.2) , dependiendo de las muestras ausentes en la fase de muestreo. Las mismas conclusiones son aplicables a las fases de la FIGURA 4.39, aunque con un nivel menor de similitud.

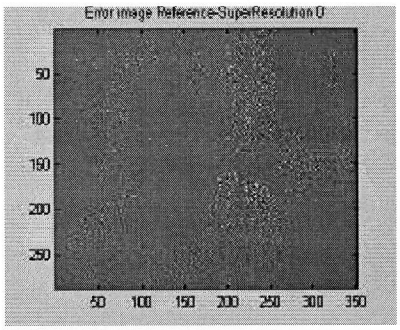
4.3.7.8 Implicaciones en el proceso real de muestreo

Las primeras simulaciones realizadas usando el algoritmo de súper-resolución iterativo inicialmente descrito no presentaban el problema de disminución de la calidad de las imágenes resultantes debido a la coincidencia de los muestreos por equivalencia en los vectores de desplazamiento, debido a que usaban siempre combinaciones de imágenes donde estaban presentes todas las muestras de la imagen original de alta resolución. La constatación de este problema surge en el momento de generar vectores de desplazamiento aleatorios, en un intento de modelar la captura de imágenes en un sistema de adquisición real y gracias a realizar todos los experimentos sobre un sistema donde cada parámetro está perfectamente controlado. Cabe ahora preguntarnos si este problema podría darse en un sistema real o si es meramente un artefacto del sistema de experimentación.

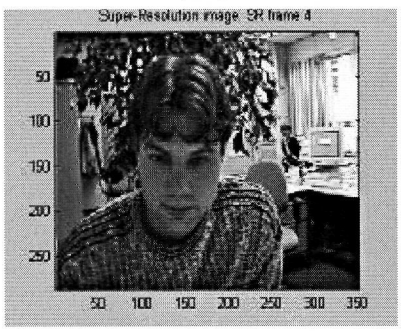
Reflexionando sobre el sistema de muestreo, debemos tener en cuenta que es imposible asegurar que las cuatro imágenes que se muestreen de forma consecutiva contendrán toda la información necesaria para incrementar significativamente la resolución de la imagen. Tampoco se puede asegurar que no vayan a existir casos de muestrear varias veces la misma imagen (movimiento lento y bajo tiempo de captura) o de que las imágenes muestreadas no tendrán muestras redundantes (movimientos equivalentes). Si esto ocurre, ya hemos visto los efectos que tendría en la calidad de la imagen final, pero de forma estadística es poco probable que, de entre un conjunto de movimientos aleatorios, se obtuviesen de forma repetitiva imágenes que no aportasen nueva información que permita incrementar la resolución de la imagen final, siempre que se pueda asegurar una distribución en cierto grado aleatoria de los movimientos. Si el sistema de adquisición estuviese limitado por algún



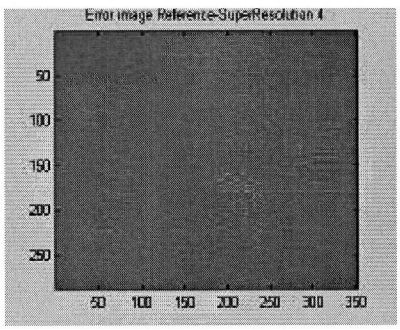
(a.1)



(a.2)



(b.1)

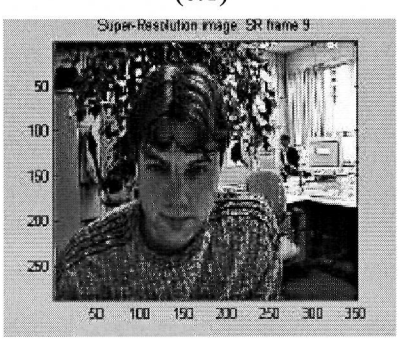


(b.2)

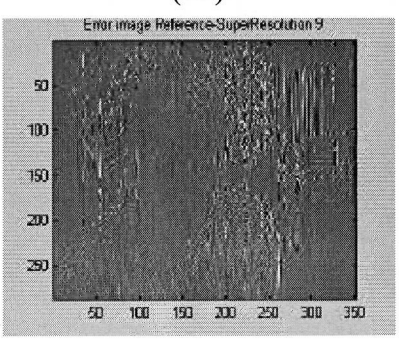
FIGURA 4.37. Imágenes de súper-resolución (1) y errores asociados (2) de los *frames* 0 (a) y 1 (b) obtenidos usando 80 iteraciones.

(c.1)

(c.2)



(d.1)



(d.2)

FIGURA 4.36. Imágenes de súper-resolución (1) y errores asociados (2) de los *frames* 7 (c) y 8 (d) obtenidos usando 80 iteraciones.

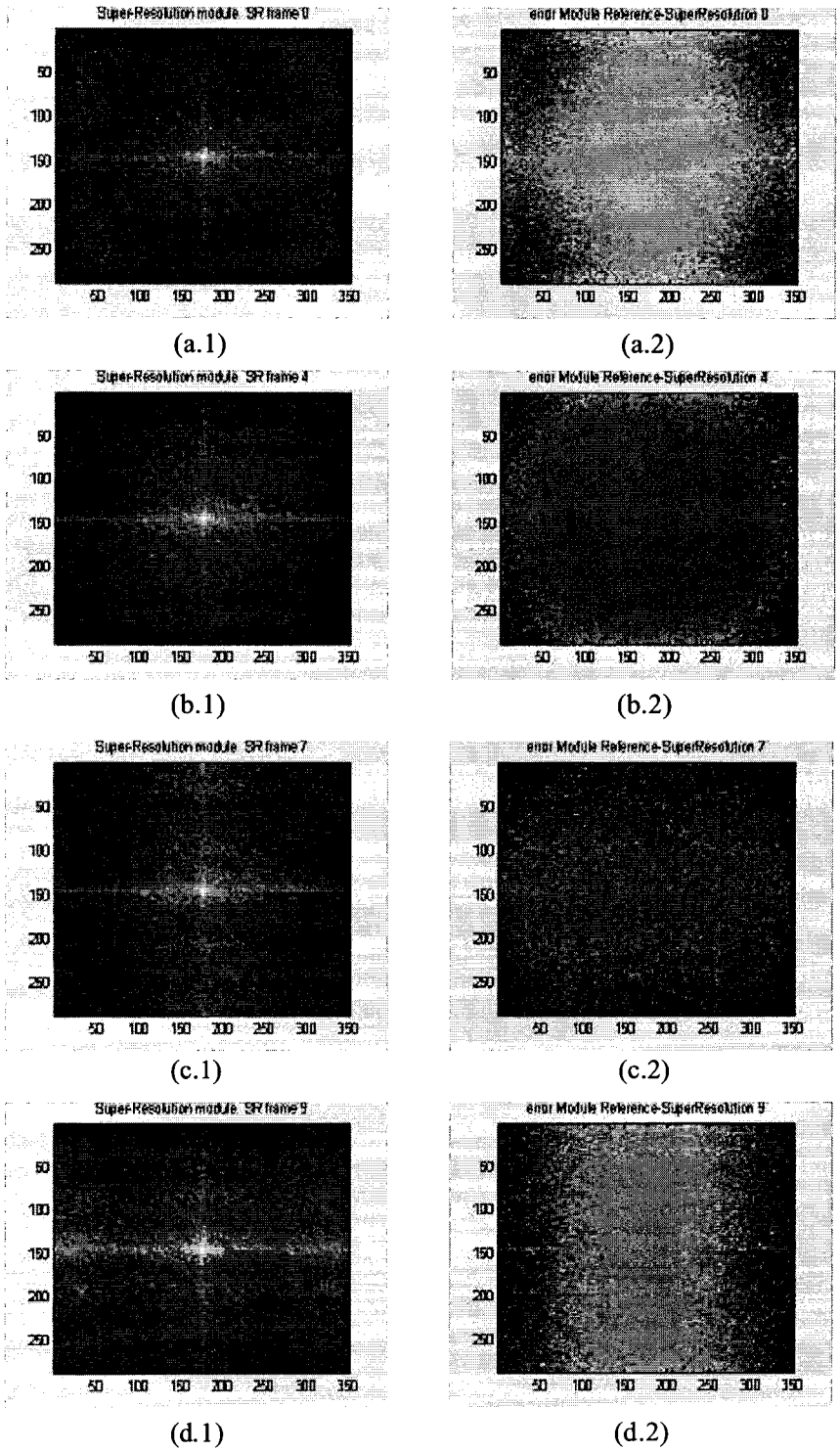


FIGURA 4.38. Módulos de la transformadas de Fourier bidimensionales (1) y errores asociados (2) de los *frames* 0 (a), 4 (b), 7 (c) y 8 (d) obtenidos usando 80 iteraciones.

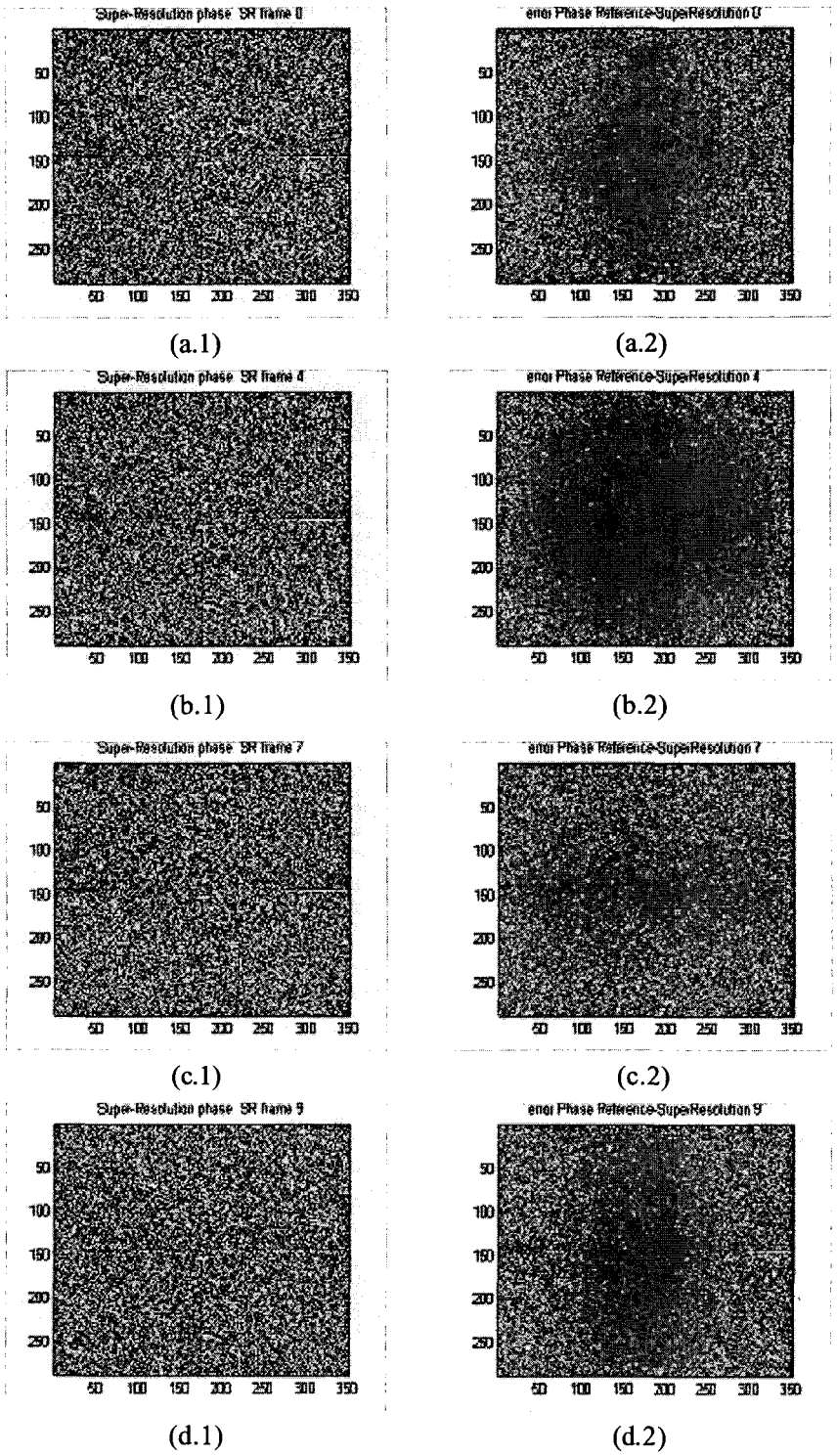


FIGURA 4.39. Fases de la transformadas de Fourier bidimensionales (1) y errores asociados (2) de los *frames* 0 (a), 4 (b), 7 (c) y 8 (d) obtenidos usando 80 iteraciones.

sistema de sujeción sobre guías físicas que no permitan todos los movimientos (trípodes, carros de sujeción y transporte, etc.) entonces los grados de libertad quedan limitados y es posible que la calidad se vea así limitada. En todo caso, sólo se verían limitadas las mejoras de la imagen debidas al movimiento de la cámara, pero no las mejoras de la imagen debidas a movimientos propias de la escena.

Otra forma de minimizar la influencia de este efecto es aumentar el número de *frames* a combinar o aumentar la resolución del estimador de movimiento, para aumentar el número de píxeles de la celda base. Por ejemplo, la celda base estudiada hasta ahora se basa en el uso de un estimador de movimiento de ½ píxel en baja resolución, que al pasar a alta resolución nos daría una capacidad de separación de 1 píxel. Si aumentamos la resolución del estimador de movimiento a ¼ de píxel en baja resolución, entonces en alta resolución tendríamos una capacidad de separación de ½ píxel, lo que modifica la celda base tal y como se muestra en FIGURA 4.40.

En este caso, las posiciones vienen indicadas en unidades de ½ píxel. Así por ejemplo, el vector (2,2) indica un desplazamiento de 2 medios píxeles (es decir 1 píxel) en ambos sentidos. Se ha marcado en negrita las posiciones de píxel, ya que un muestreo en dichas posiciones origina un aumento sustancial de la calidad de la imagen resultante, porque las posiciones sub-píxel aportan información que comparten entre si los píxeles circundantes. Con esta nueva celda base, las posibilidades de muestreo equivalente (mismos vectores canónicos) se reducen del 25% al 6.25%.

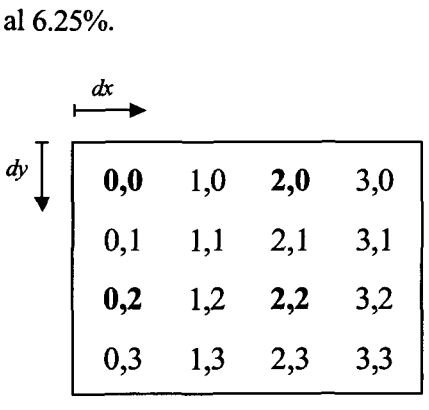


FIGURA 4.40. Celda base usando una estimador de movimiento de ¼ de píxel en baja resolución (unidades en ½ píxel).

De todas formas, a la hora de evaluar un sistema real de súper-resolución, que como tal no tiene la posibilidad de medir los vectores reales de desplazamiento, debe tenerse en cuenta un cierto factor de reducción de la calidad esperada debido a posibles ocurrencias de muestreo equivalente.

4.3.7.9 Mejoras de súper-resolución en la crominancia

Hasta este punto sólo se han mencionado las mejoras de luminancia obtenidas en las imágenes de súper-resolución. De hecho, muy pocos sistemas de súper-resolución tienen en cuenta los valores de crominancia. Debido a la baja sensibilidad del ojo humano ante la crominancia y al incremento de cálculo que su inclusión implicaría, típicamente los algoritmos de súper-resolución mejoran la luminancia pero usan una simple versión interpolada de las crominancias. En nuestro caso, se ha optado por incluir la crominancia dentro del proceso de mejora de la imagen, teniendo en cuenta que debido al tipo de muestreo YCbCr 4:2:0 las dos crominancias tienen la mitad de tamaño que la luminancia, ya que cada valor de crominancia afecta a cuatro píxeles de crominancia. Este tipo de formato es muy usado en los sistema de compresión de imágenes [BK96] debido a que el propio formato de muestreo supone un ahorro del 50% en el espacio de almacenamiento de la memoria.

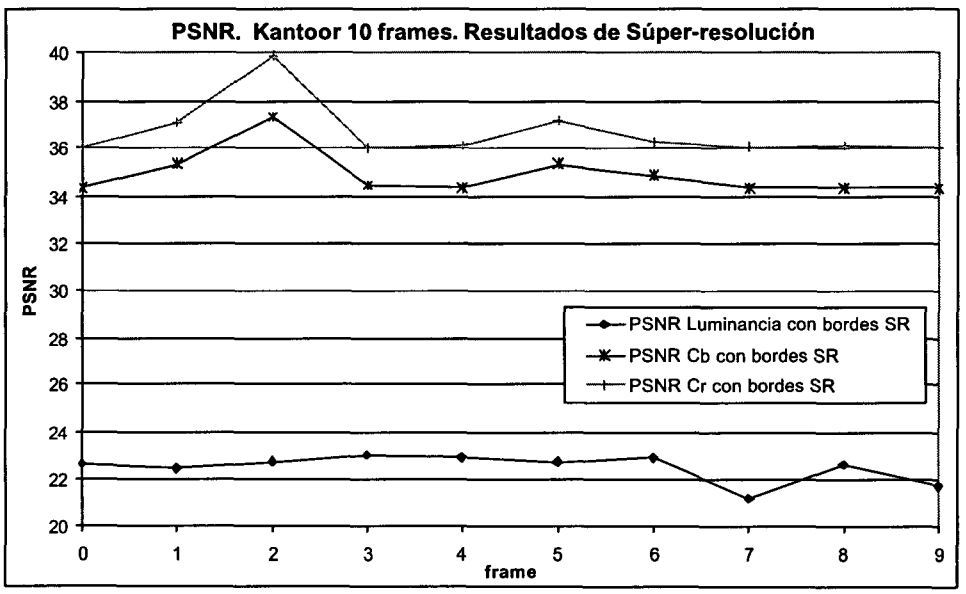
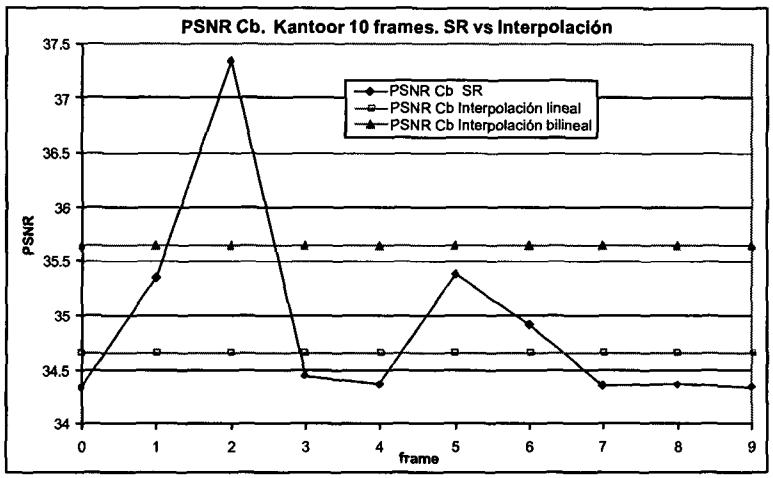


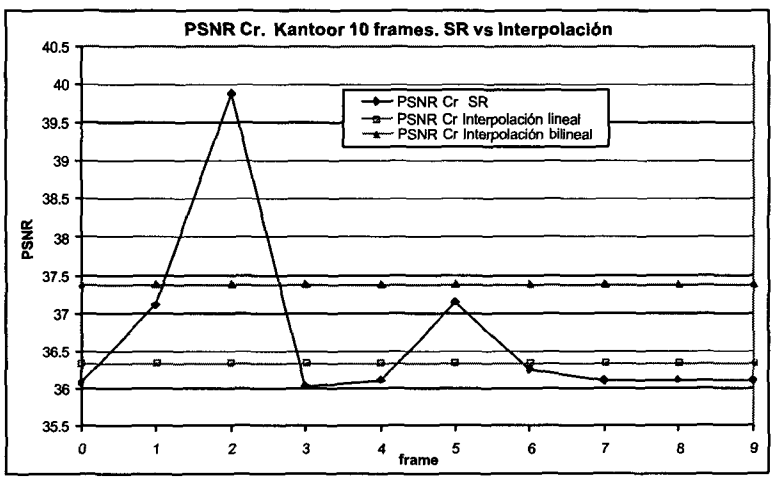
FIGURA 4.41. PSNR Crominancias azul y roja (Cb y Cr) de la secuencia de súper-resolución comparada con la PSNR de la luminancia (ver FIGURA 4.25).

En la FIGURA 4.41 se muestra la PSNR de las crominancias junto con la luminancia para la secuencia de súper-resolución mostrada en la Figura 4.21, la FIGURA 4.22 y la FIGURA 4.23. Como puede verse, las crominancias tienen una mayor PSNR que la luminancia debido a que tienen una entropía mucho menor, lo que origina que en promedio los errores entre las crominancias de referencia y la crominancia de súper-resolución sean mucho menores, lo que

a su vez provoca un incremento notable en la PSNR. Esta menor entropía de las crominancias puede verse en la FIGURA 4.43.



(a)



(b)

FIGURA 4.42. PSNR de la crominancia azul (a) y de la crominancia roja (b) comparada con la PSNR de las crominancias interpoladas.

Sin embargo, aunque la PSNR de las crominancias sea mucho más alta que la de la luminancia, en un principio llaman la atención que esté muchas veces por debajo de los niveles de interpolación, tal y como se muestra en la FIGURA 4.42. La explicación de este fenómeno reside en la aplicación de principios de compresión de imágenes en aplicaciones de procesamiento de imágenes. Cuando se realiza la estimación de movimiento en un codificador de imágenes, sólo se calculan los vectores de movimiento de la luminancia, y a la hora de realizar la compensación del movimiento, se aplican los mismos vectores tanto a la luminancia como a las crominancias, con la intención de ahorrar tiempo de cálculo y bajo la

suposición de que las tres imágenes son muy similares. Aunque los vectores de la crominancia no sean iguales a los de la luminancia, esto no redundará negativamente en la calidad de las imágenes transmitidas, ya que junto con los vectores se transmite el error. Sí afectará negativamente a la compresión, ya que con vectores mejores se lograrían imágenes con menor entropía que serían luego más eficientemente comprimidas por la transformación discreta del coseno (DCT, *Discrete Cosine Transform*) y el cuantificador. Pero como en el caso de las crominancias las imágenes ya tienen baja entropía de por si, este efecto se hace apenas patente. Sin embargo, un simple vistazo a las imágenes de luminancia y crominancias por separado, tal y como se recoge en la FIGURA 4.42, hace patente que, a efectos de restauración y teniendo en cuenta lo sensible que son los SRA a los vectores de movimiento, la aplicación de unos vectores de movimiento que no son los específicos de la imagen, apenas mejora la calidad por encima de los niveles de interpolación. La única forma de obtener ventajas, sería hacer una estimación de movimiento adicional para cada una de las dos imágenes de crominancia, lo que aumentaría enormemente la cantidad de tiempo requerida para realizar dichos cálculos comprometiendo su ejecución en tiempo real.

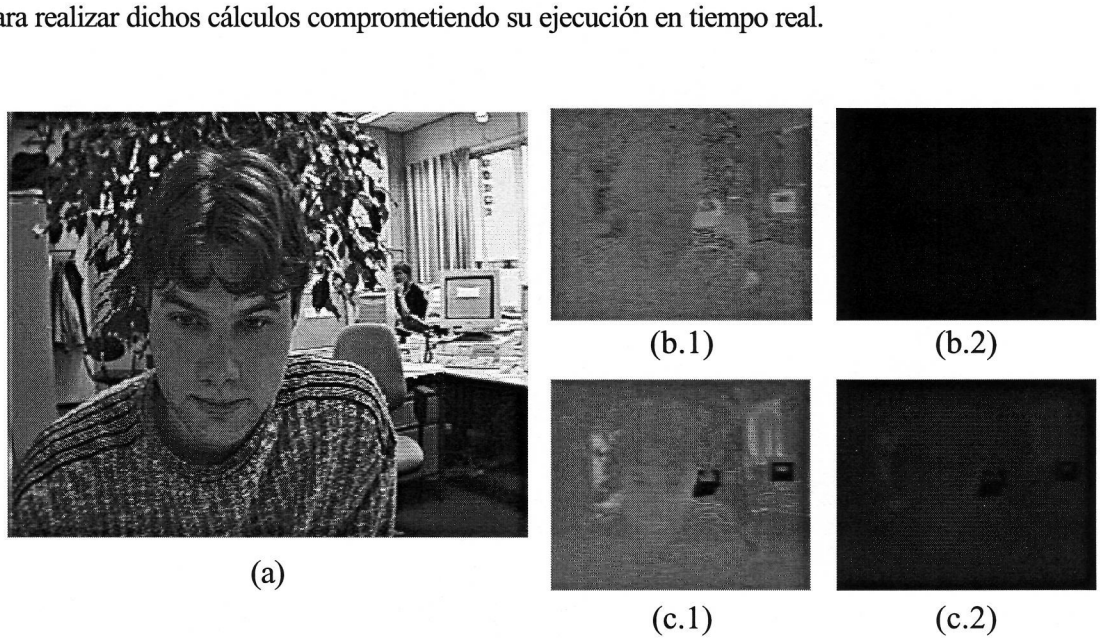


FIGURA 4.43. Imagen de referencia en formato YCbCr 4:2:0 y tamaño CIF, descompuesta en luminancia (a), crominancia azul (b) y crominancia roja (c). Las crominancias se muestran en variaciones de luminosidad (1) y en color (2).

A nivel espectral las conclusiones son similares: los coeficientes de correlación espectrales de las crominancias son mayores que el coeficiente de la luminancia y los coeficientes de correlación en fase pocas veces supera los niveles de interpolación, aunque el coeficiente de correlación en módulo está con mayor frecuencia por encima de los niveles de interpolación, como se aprecia en la FIGURA 4.44.

A raíz de estos resultados concluimos que, de no poder realizar dos estimaciones adicionales de movimiento sobre las crominancias sin que estas operaciones adicionales comprometan las restricciones de funcionamiento en tiempo real, no merece la pena aplicar el algoritmo de súper-resolución sobre las crominancias, puesto que se consigue un resultado similar a partir de la interpolación de las crominancias.

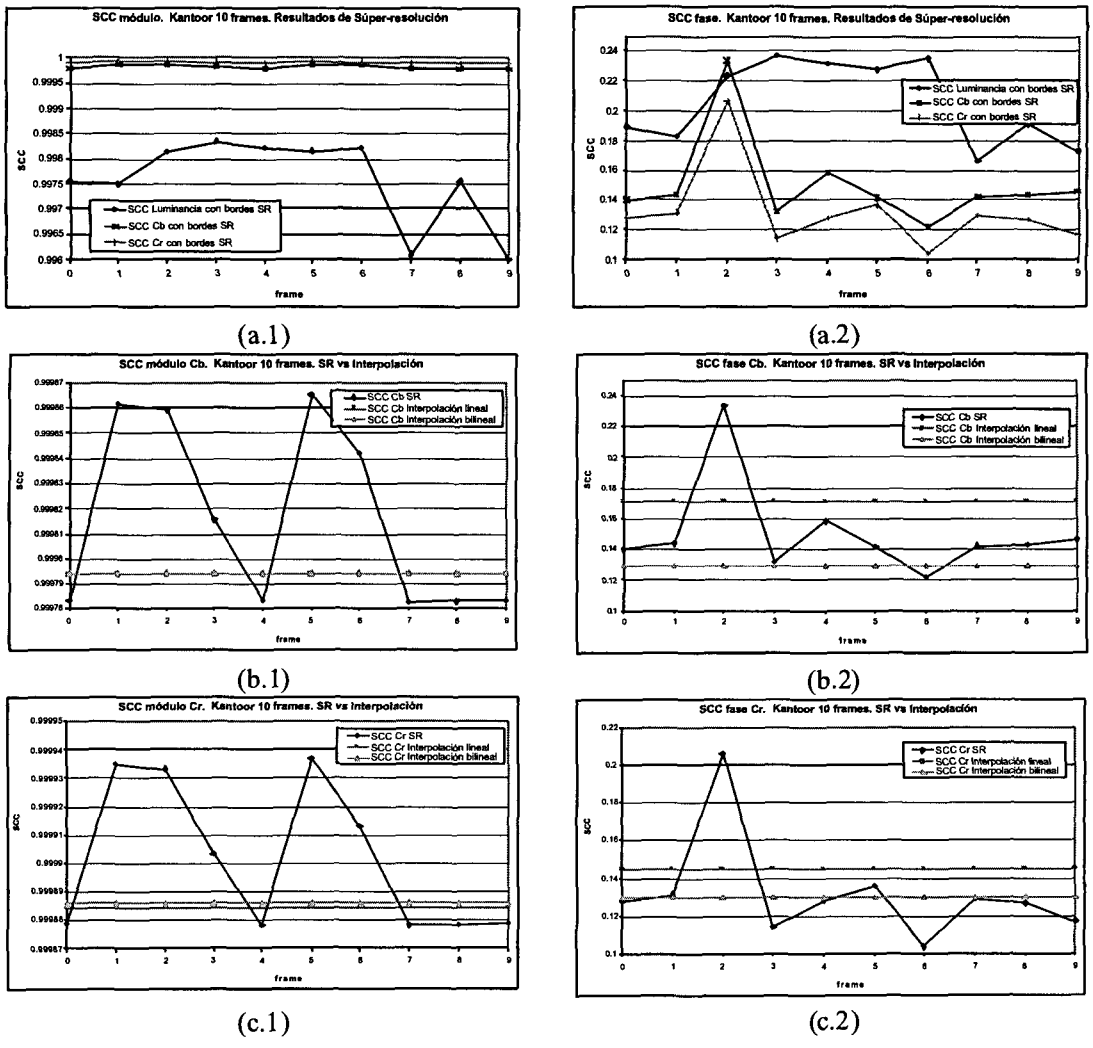


FIGURA 4.44. SCC en módulo (1) y en fase (2) de las crominancias. En (a) se comparan las crominancias con la luminancia, en (b) se muestra la crominancia azul frente a sus niveles de interpolación y en (c) se muestra la crominancia roja frente a sus niveles de interpolación. Nótese que la escala de las ordenadas para el módulo está muy ampliada.

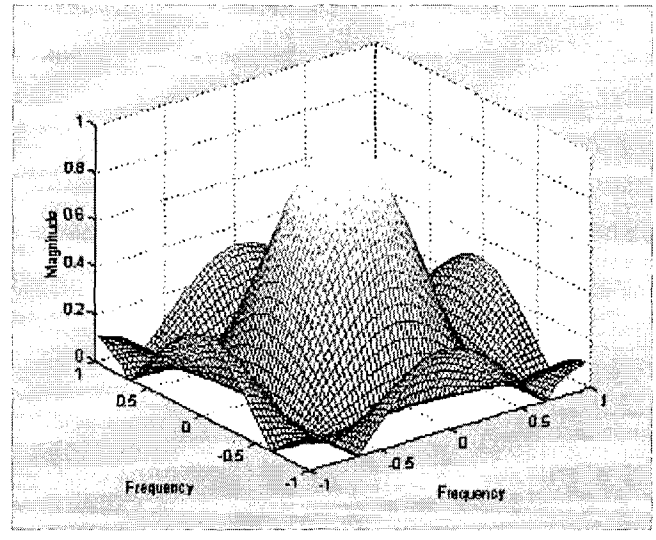
4.3.8Algoritmo iterativo con referencia a la primera imagen

La solución adoptada en la versión 1.2 para poder realizar las diferentes medidas de calidad tiene el problema de que el conjunto de vectores usados para la generación de la imagen de súper-resolución ha de tener forzosamente media cero. Si el proceso de generación de vectores es realmente aleatorio errático, esta condición es imposible de cumplir, especialmente si el número de imágenes a combinar tampoco fuese un número fijo. Por está razón se ha desarrollado una nueva versión del algoritmo donde se cambia la primera propuesta de imagen de súper-resolución por la primera imagen interpolada, en lugar de la imagen promedio. De esta forma se asegura que los píxeles de la imagen final ocuparán las mismas posiciones que la de referencia permitiendo el cálculo directo de la relación señal-ruido, siempre y cuando la primera imagen generada por sub-muestreo use el vector de desplazamiento (0,0). Desde el punto de vista de la mejora de imágenes estáticas esto no supone ninguna limitación al proceso, más bien al contrario supone una mejoría en el modelo de adquisición de imágenes estáticas. Cuando tomamos una fotografía, lo normal es que deseemos que la primera imagen tomada sea la que sirva de referencia en el proceso de mejora de la calidad. Este es el significado del vector (0,0): que la primera imagen tomada será la referencia y que el resto de imágenes se usarán para mejorar la calidad de la imagen ya existente, ajustándolas a la primera imagen. Desde el punto de vista de la mejora de imágenes para vídeo, tan solo supone facilitar el proceso de medida. De no forzar el primero vector de la serie al vector (0,0) tendríamos que realizar un desplazamiento de la imagen resultante antes de calcular la PSNR, bien a la primera imagen o bien a la imagen promedio si se hubiese seguido con la estrategia anterior. La nueva versión (y última entre las versiones iterativas) del algoritmo se ha etiquetado como v1.3 y se muestra en la FIGURA 4.46.

Esta versión aporta además algunas novedades respecto a las anteriores que pasaremos a comentar. Como ya se ha mencionado, en este caso la primera propuesta de imagen de súper-resolución se obtiene mediante una interpolación de orden cero de la primera imagen de baja resolución de la serie de entrada. Como en las versiones anteriores, esta imagen se almacena en la memoria HR_B, que contendrá en todo momento la imagen de súper-resolución. Si no es el caso de la primera iteración, LR_B se obtendrá mediante diezmado de HR_B, en caso contrario (que sea la primera iteración) LR_B es la primera imagen de entrada de baja resolución. El siguiente paso dentro del bucle iterativo es la obtención de los vectores de movimiento. Si se trata del primer *frame*, su vector de movimiento no hace falta que sea calculado, ya que es el vector cero. Para el resto de los casos, se calculan los vectores de movimiento entre el resto de las imágenes de entrada y la versión en baja resolución de la

imagen de súper-resolución, es decir LR_B. La novedad en este caso es que antes de realizar la estimación de movimiento se realiza un filtrado paso bajo de orden tres de LR_B usando la función de transferencia normalizada que se muestra en la FIGURA 4.45. Este filtrado previo tiene por objeto disminuir la cantidad inicial de *aliasing* y mejorar por lo tanto la calidad de los vectores de movimiento.

$$m = \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$



$$\begin{aligned} O(x,y) = {} & m(0,0)\, I(x-1,y-1) + m(0,1)\, I(x,y-1) + m(0,2)\, I(x+1,y-1) + \\ + {} & m(1,0)\, I(x-1,y) + m(1,1)\, I(x,y) + m(1,2)\, I(x+1,y) + \\ + {} & m(2,0)\, I(x-1,y+1) + m(2,1)\, I(x,y+1) + m(2,2)\, I(x+1,y+1) \end{aligned}$$

FIGURA 4.45. Respuesta al impulso normalizada del filtro paso bajo de orden tres usado para el filtrado de las imágenes.

```
* Iterative super-resolution algorithm v1.3

    Set the value of the improvement: scale
    nr_frames = scale*scale, M'= scale*M, N'= scale*N
    HR_B[M'][N'], HR_S[M'][N'], HR_T[M'][N'] all of 8 bits. HR_A[M'][N'] is 9 bits
    LR_B[M][N] for the motion estimation
    Read a set of aliased Low-Resolution images in LR_I[#nr_frames][M][N]
    Set the value of the maximum number of iterations: nr_iterations

    // Starting with the First Image

    LR_B = LR_I[0]                                                    [SR_INIT_B]
    HR_B = Upsample(LR_I[0])

    // Iterations
    FOR it = 0 .. nr_iterations-1

       IF (it ≠ 0)   LR_B = Downsample(HR_B)                          [SR_DOWNSAMPLE]

       FOR fr = 0 .. nr_frames-1
            IF (fr==0)
               MV_fr2ref[fr] = 0
            ELSE
               IF (it ≠ 0)    LR_B = Low_Pass_Filter(LR_B)            [SR_FILTER]
               MV_fr2ref[fr] = Calc_Motion_Estimation (LR_I[fr], LR_B)
            END IF
            Select_global_motion_vector()
            MV_ref2fr[fr] = -MV_fr2ref[fr]                            INVERT_MV()
            MV_fr2ref[fr] = 2 .* MV_fr2ref[fr]
            MV_ref2fr[fr] = 2 .* MV_ref2fr[fr]
       END FOR
       HR_A = 0                                                       [SR_INIT_A]
       FOR fr = 0 .. nr_frames-1
            HR_S = Motion_Compensation (HR_B, MV_ref2fr[fr])          [SR_MOT_COMP1]
            HR_S = Upsample(LR_I[fr])/2 – HR_S/2                      [SR_UPSAMPLE]
            HR_T = Motion_Compensation (HR_S, MV_fr2ref[fr])          [SR_MOT_COMP2]
            HR_A = HR_A + HR_T/2                                      [SR_ADD]
       END FOR
       HR_B = HR_B + HR_A                                             [SR_UPDATE]
       variance = variance(HR_A)                                      [SR_STAT]
       If (variance < 0.5)  break
    END FOR
```

FIGURA 4.46. Pseudo-código del algoritmo iterativo v1.3, modificado para usar la primera imagen de baja resolución como referencia.

4.3.8.1 Diagrama de bloques y requerimientos de memoria

El diagrama de bloques del flujo de datos de esta última versión del algoritmo es el mostrado en la FIGURA 4.47. Puede compararse con el de la FIGURA 4.17 de la versión v1.2 para observar la desaparición de la operación Average, la aparición del filtrado y variaciones en el flujo de datos y en las memorias. Los requerimientos de memoria son los mismos que los de la versión anterior v1.2, por lo que son aplicables aquí también los resultados de la TABLA 4.6 y de la TABLA 4.7 así como los de la FIGURA 4.18. La memoria HR_A se ha señalizado usando doble rayado para indicar que es de 9 bits y los operadores se han sombreado. El resto de las memorias se ha recuadrado usando líneas simples.

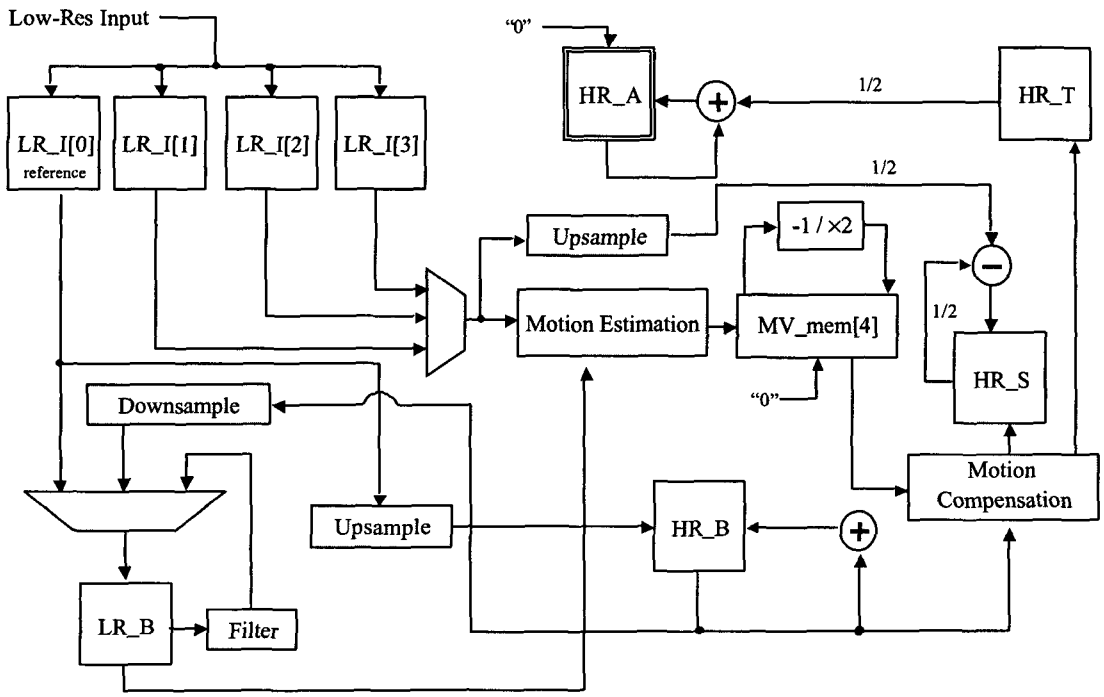


FIGURA 4.47. Diagrama de bloques del flujo de datos del algoritmo de súper-resolución v1.3.

En la TABLA 4.9 se muestran los vectores de desplazamiento aplicados al primer *frame* de entrada para obtener la secuencia de video de prueba. La primera columna muestra el número de orden del *frame* de salida, la segunda y la tercera columnas muestran los vectores de desplazamiento de alta resolución y de baja resolución respectivamente, en unidades de ½ píxel. La última columna muestra la reducción a vectores canónicos. En función de los vectores canónicos, podemos hacer la misma clasificación de imágenes que se recoge en la FIGURA 4.34. Esta clasificación se muestra en la TABLA 4.10 y supone ya de por si una

anticipación de la calidad que se obtendrá al finalizar el proceso de súper-resolución. Debido a que el vector (0,0) siempre estará presente, los tipos b.4, c.2, c.5, c.6, d.2, d.3 y d.4 no son posibles en esta versión del algoritmo. Debemos recordar, como se ha explicado en la sección 4.3.7.6 que los tipos c y d limitan la calidad de imagen de súper-resolución obtenida.

4.3.8.2 Resultados de simulación y análisis de calidad y comportamiento de la versión v1.3 del algoritmo iterativo.

En la FIGURA 4.48 se muestra la PSNR obtenida tras aplicar este algoritmo a la primera imagen de la secuencia KANTOOR, de tamaño CIF (352×288), sub-muestreada y desplazada las cantidades recogidas en la TABLA 4.9. Al igual que en el caso anterior, también se han incluido las PSNRs de las imágenes interpoladas para establecer los niveles de interpolación.

frame	Vectores de alta resolución		Vectores de baja resolución		Reducción a vectores canónicos		*frame*	Vectores de alta resolución		Vectores de baja resolución		Reducción a vectores canónicos	
0	0	0	0	0	0	0	5	0	0	0	0	0	0
	-2	2	-1	1	1	1		2	6	1	3	1	1
	2	2	1	1	1	1		6	4	3	2	1	0
	0	4	0	2	0	0		0	6	0	3	0	1
1	0	0	0	0	0	0	6	0	0	0	0	0	0
	2	2	1	1	1	1		-2	-4	-1	-2	1	0
	0	-4	0	-2	0	0		-4	2	-2	1	0	1
	6	2	3	1	1	1		-2	2	-1	1	1	1
2	0	0	0	0	0	0	7	0	0	0	0	0	0
	0	-2	0	-1	0	1		-2	-4	-1	-2	1	0
	2	-2	1	-1	1	1		2	0	1	0	1	0
	6	4	3	2	1	0		0	-4	0	-2	0	0
3	0	0	0	0	0	0	8	0	0	0	0	0	0
	0	-2	0	-1	0	1		-4	-4	-2	-2	0	0
	-2	-4	-1	-2	1	0		-2	-2	-1	-1	1	1
	-6	-2	-3	-1	1	1		-2	-2	-1	-1	1	1
4	0	0	0	0	0	0	9	0	0	0	0	0	0
	2	2	1	1	1	1		0	4	0	2	0	0
	0	-2	0	-1	0	1		-4	2	-2	1	0	1
	-2	0	-1	0	1	0		-4	2	-2	1	0	1

TABLA 4.9. Vectores de movimiento aleatoriamente generados en distancias de dos píxeles para alta resolución, de un píxel para baja resolución y su reducción a vectores canónicos.

frame	Tipo de imagen de salida	*frame*	Tipo de imagen de salida
0	c.1	5	a
1	c.1	6	a
2	a	7	c.3
3	a	8	c.1
4	a	9	c.4

TABLA 4.10. Tipos de imágenes de salida en función de las muestras presentes en la celda base de 4 píxeles.

De forma similar al caso anterior, observamos inicialmente grandes variaciones en la calidad de las imágenes, provocadas básicamente por el conocido efecto de borde. Para sustraer este efecto de la PSNR, en la FIGURA 4.49 se muestra la PSNR de la misma secuencia, ó PSNR', una vez que se ha eliminado un borde de 16 píxeles alrededor de las imágenes.

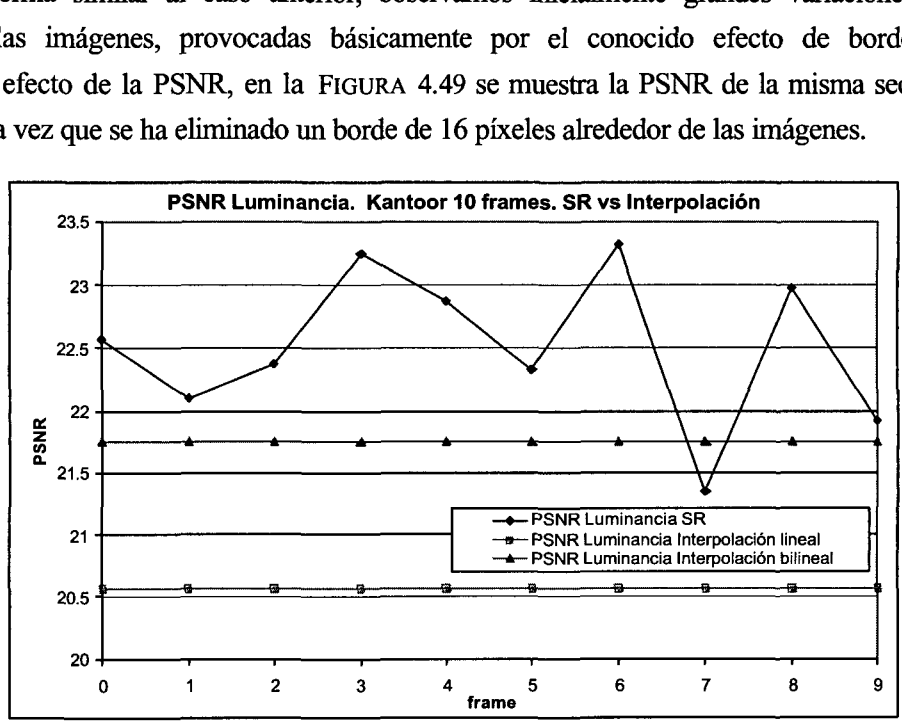


FIGURA 4.48. PSNR de la secuencia de salida usando la versión v1.3 del ASR frente a las imágenes interpoladas usando interpolación lineal de orden cero y bilineal.

En la gráfica se aprecia nuevamente una distribución de la PSNR que depende básicamente del tipo de muestreo. Para reducir el número de grados de libertad y facilitar por lo tanto el análisis de los resultados, se ha mantenido inalterable la semilla de generación de los números aleatorios que componen los desplazamientos. Por este motivo, los tipos de imágenes de salida son prácticamente los mismos que los obtenidos en la versión anterior del ASR. Las diferencias residen en que para asegurar media cero, el último vector de

desplazamiento de cada grupo de cuatro vectores se ha ajustado en consecuencia. Este ajuste busca establecer una base común o más similar para la comparación entre los algoritmos, pero obviamente no afecta –no existe– en el comportamiento real del sistema.

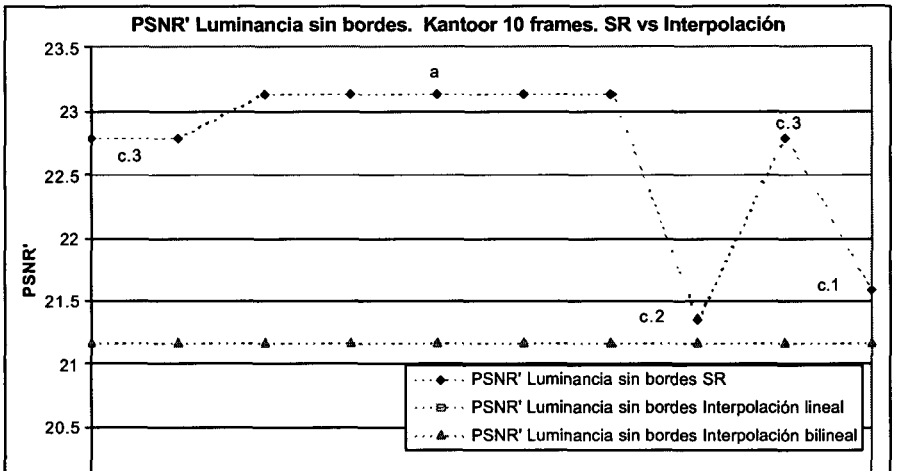


FIGURA 4.49. PSNR' de la secuencia de salida de la versión v1.3 del ASR eliminando un borde de 16 píxeles alrededor de la imagen.

Los resultados obtenidos para este algoritmo son muy similares en calidad y comportamiento en ambas versiones del algoritmo iterativo. En la FIGURA 4.50 se muestra una comparativa de la PSNR de la versión v1.2 y la versión v1.3 usando la imagen completa (a) y sin bordes (b), pudiéndose comprobar que en este último caso las calidades obtenidas son casi las mismas. Sin embargo, cuando se tiene la imagen completa, parece que la versión que utiliza el *frame* cero de referencia obtiene puntualmente mejores resultados (*frames* 3, 6 y 8) si bien en el resto de los casos se obtienen calidades ligeramente inferiores. En cualquier caso, esta figura proporciona indicaciones de que las mejoras obtenidas mediante variantes del algoritmo de súper-resolución iterativo parecen haber alcanzado un límite. En el capítulo 5 analizaremos transformaciones del algoritmo hacía esquemas no iterativos y los nuevos problemas de mapeo en la arquitectura *Picasso* adecuadamente modificada. Sin embargo, la influencia de los vectores de movimiento en la calidad final merece un ulterior análisis que hacemos en la próxima sección.

4.3.9Consideraciones sobre la búsqueda del vector de movimiento

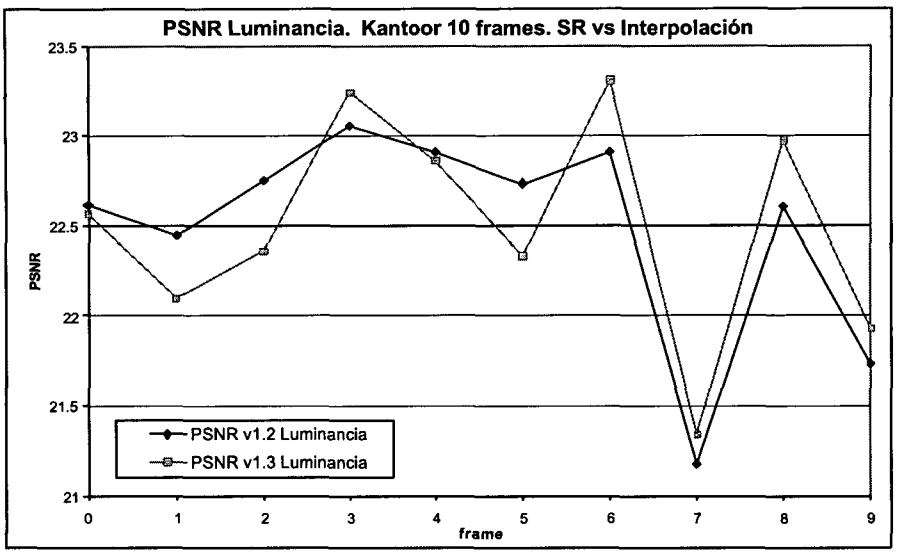
Como ya se ha comentado, la calidad de las imágenes obtenidas por los algoritmos de súper-resolución depende de forma muy importante de la precisión de los vectores de movimiento. La estimación del movimiento es una de las funciones más costosas en cantidad de cálculo y tiempo de procesamiento, por lo que con la intención de mantenernos en condiciones de bajo coste y prestaciones de tiempo real, hemos hecho uso del *'Motion Estimator'* que proporciona la arquitectura de *Picasso*. En el capítulo 3, donde se describió la arquitectura *Picasso* que soporta la ejecución de los ASR aquí desarrollados, se muestran los dos tipos de estimación de movimiento implementados: la búsqueda exhaustiva o *Full-Search* y la búsqueda recursiva en tres dimensiones o 3DRS (*3 Dimensional Recursive Search*). Mientras que el primer tipo de búsqueda tiene la ventaja de completitud en la búsqueda y la desventaja del alto coste computacional, el segundo tipo presenta la ventaja de la rapidez frente a la desventaja de evaluar sólo ciertos vectores, aunque los resultados obtenidos sean razonablemente precisos.

El uso de una u otra estrategia de búsqueda compromete la calidad de las imágenes obtenidas. En la FIGURA 4.51 se muestra la PSNR de la secuencia de salida KANTOOR vista hasta ahora, usando la versión v1.2 (a) y la versión v1.3 (b) para ambos métodos de búsqueda de los vectores de movimiento.

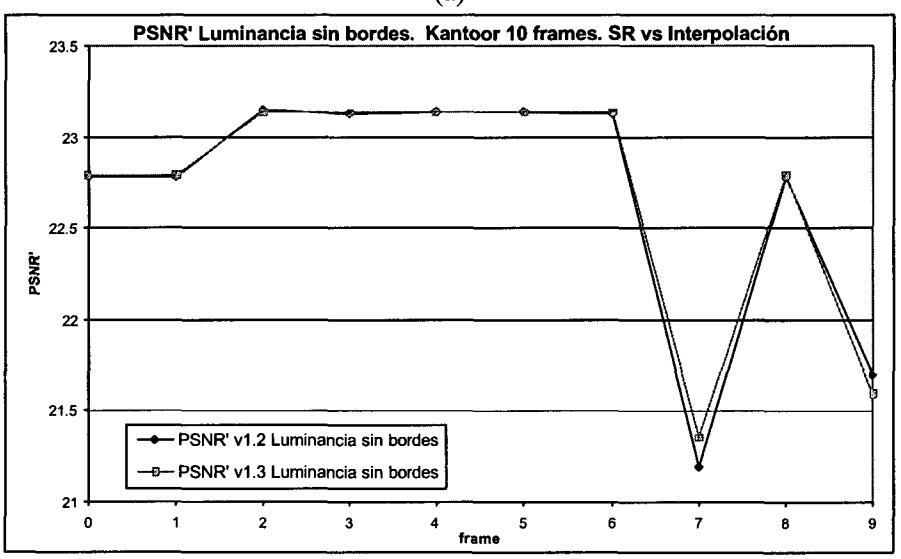
Un aspecto muy interesante a resaltar de estos resultados es que claramente la versión v1.3 exhibe resultados mucho mejores que la versión v1.2, estando casi todas las PSNR por encima del nivel de interpolación (*excepto* el *frame* 2). Esto es debido a la inclusión del filtro paso bajo antes de realizar la estimación del movimiento. En efecto, la eliminación del *aliasing* de la imagen de entrada antes de realizar la estimación de movimiento provoca una mejoría considerable en la calidad de los vectores de movimiento y ésta repercute directamente en la calidad de la imagen final. Esta diferencia se hace aún más patente cuando eliminamos los bordes de las imágenes, como se muestra en la FIGURA 4.52. En este caso, todos los *frames* de salida de la versión v1.3 tienen calidades superiores al nivel de interpolación.

Además, la versión v1.3 muestra una mayor estabilidad en la calidad de las imágenes, lo que repercute en una mejor visión de la imagen final. Una secuencia de video con saltos

bruscos de calidad resulta más molesta en el momento de ser visionada que una secuencia de inferior calidad pero más estable.



(a)

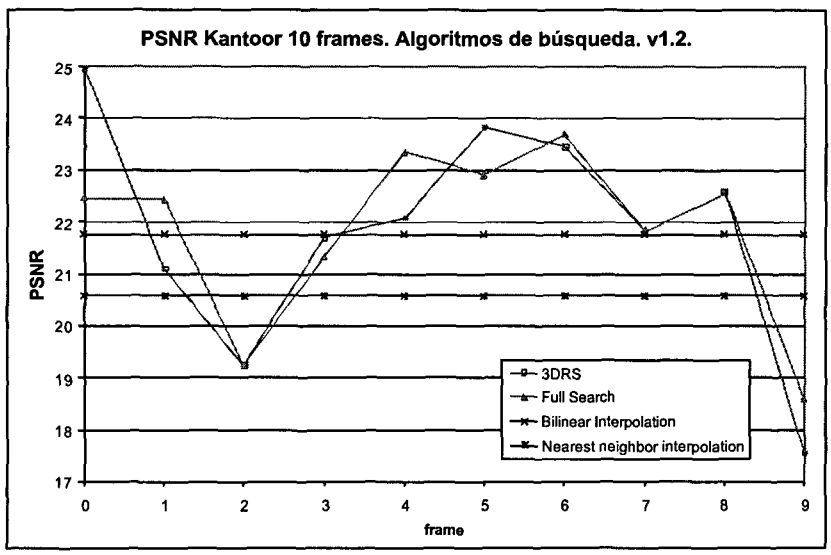


(b)

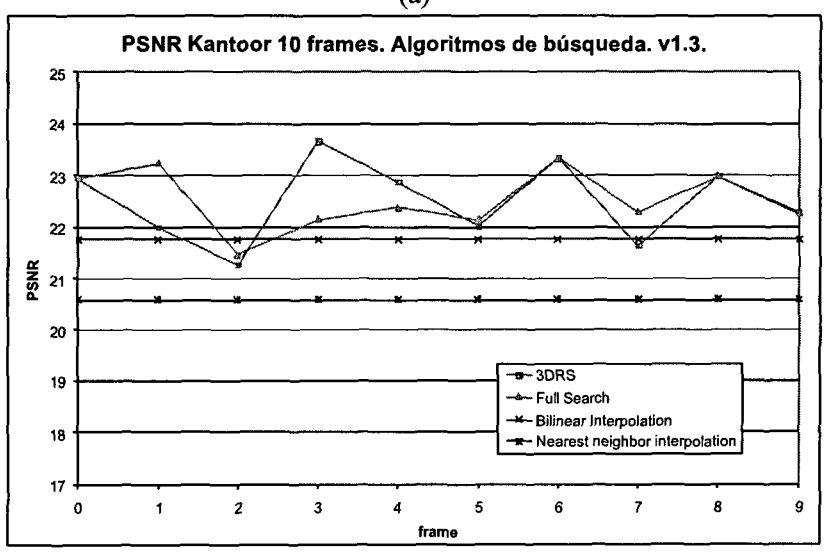
FIGURA 4.50. Comparación de la PSNR obtenida en las versiones v1.2 y v1.3 de los ASR, usando la imagen completa (a) y eliminando los bordes del cálculo (b).

A pesar de que el análisis de la calidad de los vectores de movimiento nos ha conducido a observar más claramente la superioridad de la versión v1.3 del algoritmo de súper-resolución sin embargo, no podemos concluir nada sobre qué método es mejor para la estimación del movimiento, tal y como se desprende de la observación de las calidades

obtenidas en la FIGURA 4.52, donde ya se ha eliminado el efecto de borde. Puede comprobarse cómo el método que 'a priori' es el mejor candidato para obtener vectores de movimiento buenos, el *Full-Search*, no llega a prevalecer sobre los otros métodos. Esto es debido a que en realidad el proceso de estimación de movimiento usado no refleja el movimiento real de la imagen, sino que tan sólo trata de minimizar una función de coste a nivel de bloque, en este caso el error absoluto medio, por lo que la calidad de la imagen variará en función del parecido, o la falta de parecido, entre el vector real y el vector que minimice el coste, el error absoluto medio, en cada momento.

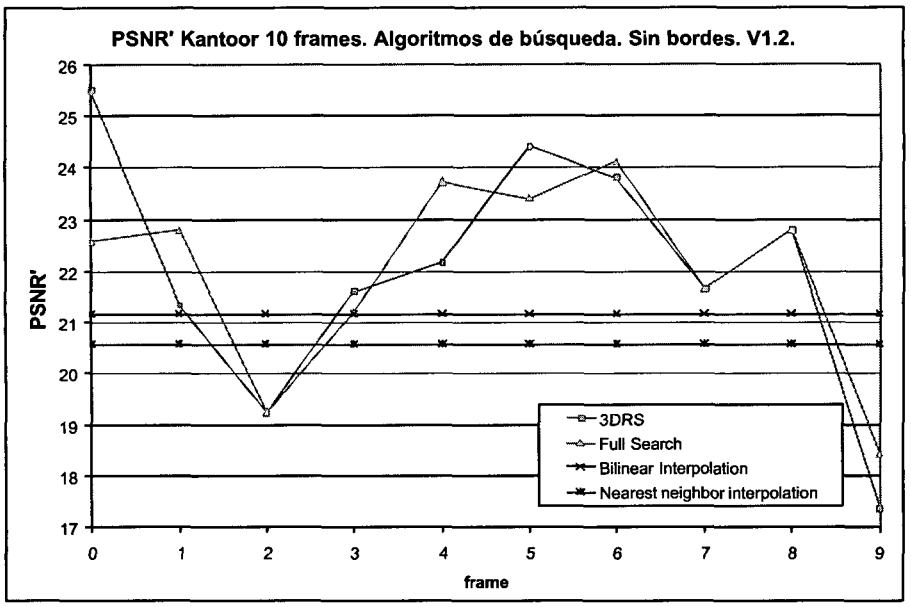


(a)

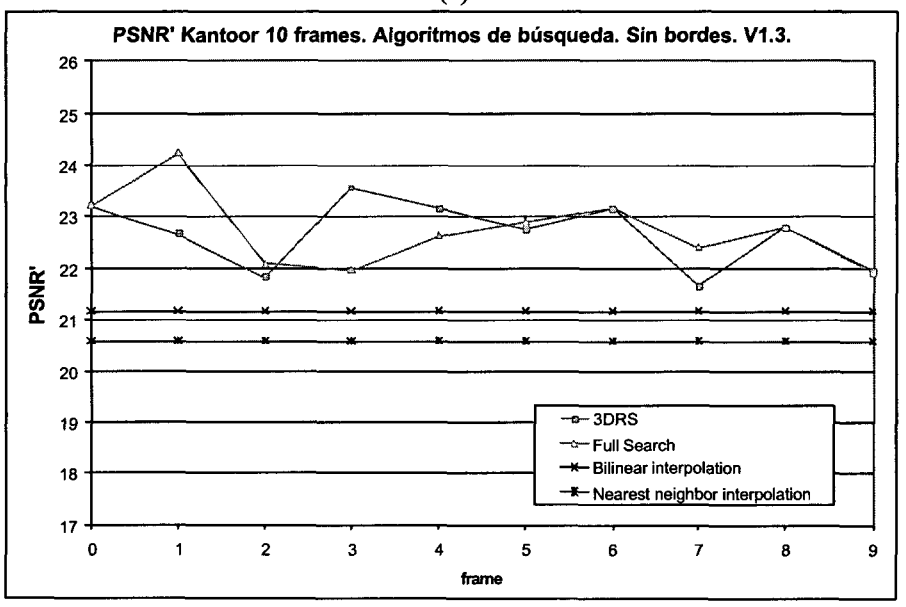


(b)

FIGURA 4.51. PSNR de las versiones v1.2 (a) y v1.3 (b) de los ASR, usando diferentes estrategias de búsqueda de los vectores de movimiento.



(a)



(b)

FIGURA 4.52. PSNR' de las versiones v1.2 (a) y v1.3 (b) de los ASR, usando diferentes estrategias de búsqueda de los vectores de movimiento, eliminando los bordes de las imágenes.

4.4 Conclusiones

En general, a la hora de calcular las figuras de mérito de cada algoritmo, debe tenerse en cuenta el tratamiento que se hace de los bordes de la imagen. Los bordes de las imágenes suponen el tener que tratar casos particulares en función del borde, lo que hace aún más engorrosa la descripción de los algoritmos. Sin embargo, esta dificultad ya es tenida en cuenta en las unidades de procesamiento de *Picasso*, por lo que se intenta hacer uso ventajoso de estas características. Por otra parte, el proceso de generación de imágenes de prueba debe tener especial cuidado con los bordes de las imágenes. Como el tratamiento de estos bordes disminuye de forma drástica los valores de las figuras de mérito, dependiendo de los vectores de movimiento aleatoriamente generados, una solución consiste en no tener en cuenta los bordes de las imágenes a la hora de calcular las figuras de mérito, permitiéndonos de esta forma alcanzar conclusiones genéricas sobre el funcionamiento del algoritmo. Este problema ha sido detectado y resuelto durante el desarrollo y evaluación de los algoritmos iterativos, pero se va a solventar desde el principio en el proceso de generación de las imágenes de prueba para los algoritmos no iterativos.

La calidad de las imágenes de súper-resolución depende mucho del tipo de muestreo realizado sobre las imágenes de entrada. Si el conjunto de imágenes usadas para generar la imagen de súper-resolución tiene muestras comunes, el proceso de súper-resolución se ve privado de información nueva, generando imágenes de calidad menor a las que se podrían obtener si se dispusiese de imágenes donde cada una aporta información diferente.

De forma general, podemos concluir que sólo merece la pena aplicar técnicas de súper-resolución en las imágenes de crominancia cuando podemos permitirnos realizar estimaciones de movimiento especificas para este tipo de imágenes. En caso contrario, es preferible usar versiones interpoladas de las crominancias.

El método usado para la estimación de movimiento no tiene una influencia apreciable sobre la calidad de la imagen final en este tipo de algoritmos, debido a que los vectores de movimiento obtenidos son sólo una aproximación a los movimientos reales de la imagen. Así pues, no resulta en una mejor calidad el usar un método de estimación de movimiento de búsqueda exhaustiva de alto coste computacional frente a un método mucho más rápido como es la búsqueda jerárquica en tres dimensiones, ya implementada en la arquitectura *Picasso* para la compresión de imágenes.

Sin embargo, sí resulta decisivo el uso de filtros paso bajo en la imagen de entrada antes de la estimación de movimiento, alcanzando de esta forma mejoras apreciables en la calidad y en la estabilidad de la calidad, lo que mejora la visión de la secuencia en el momento de ser visionada.

Aunque ya se han presentado las gráficas con las métricas de calidad obtenidas para los dos algoritmos iterativos de súper-resolución principales, las versiones v1.2 y v1.3, resulta útil, a efectos de comparativa con las versiones siguientes, resumir las calidades obtenidas en una sola fgura de mérito, como puede ser la PSNR media de toda la serie de imágenes. Estos datos se muestran en la TABLA 4.11 para 8 iteraciones, ya que es ese el momento aproximado donde la PSNR empieza a bajar para las imágenes que no son del tipo 'a'. Esto supone una solución de compromiso, ya que las imágenes tipo 'a' realmente no alcanzan la PSNR máxima hasta pasadas 15 ó 20 iteraciones. Como se desprende de estos resultados, en promedio las crominancias muestran mayor PSNR debido a su menor entropía y la versión v1.3 (referencia al primer *frame*) muestra 5 dB de mejoría en luminancia respecto a la v1.2 (referencia al *frame* promedio). De todas formas, hay que tener en cuenta que estos resultados dependen mucho del número de iteraciones y de las posiciones de las muestras presentes en el momento de la adquisición, por lo que su validez debe ser sopesada con precaución.

	PSNR media luminancia	PSNR media crominancia roja	PSNR media crominancia azul
V1.2	23.19478 dB	38.45123 dB	40.79584 dB
V1.3	28.2419 dB	38.24526 dB	39.88109 dB

TABLA 4.11. PSNR media para las versiones v1.2 y v1.3 de 10 *frames* usando 8 iteraciones.

Los algoritmos iterativos constituyen un primer paso hacia el objetivo trazado de conseguir algoritmos de súper-resolución de bajo coste y tiempo real. Al haber conseguido implementar este tipo de algoritmos sobre la plataforma *Picasso* que soporta para la compresión híbrida de video y está destinada a su producción en volumen por Philips, y al haber realizado solamente modificaciones pequeñas en los componentes de la arquitectura, podemos considerar alcanzado los objetivos de bajo coste, pero no así los de tiempo real. Para que un algoritmo iterativo alcance prestaciones de tiempo real, es necesario que el tiempo requerido para realizar el número total de iteraciones necesarias, la latencia del algoritmo, se mantenga por debajo del limite máximo establecido como tiempo real. En nuestro caso, ese límite viene impuesto por la cadencia de entrada de las imágenes de la secuencia de video. Para poder considerar que el algoritmo trabaja en tiempo real, debemos entregar una imagen

de salida por cada nueva imagen de entrada, y en *Picasso*, el tiempo de adquisición y entrega de una nueva imagen de entrada es aproximadamente equiparable al de la realización de una estimación de movimiento. Dado que cada iteración de los algoritmos desarrollados conlleva una estimación de movimiento, concluimos que sólo es posible el funcionamiento en tiempo real si ,siguiendo el esquema algorítmico propuesto y manteniendo la aplicación dentro de los límites de bajo coste, se encuentra un algoritmo de súper-resolución de tipo **no iterativo**. Obviamente, una solución para mantener este tipo de algoritmos sería el replicar las unidades de procesamiento, segmentando además el flujo de datos en el tiempo. De esta forma conseguiríamos prestaciones de tiempo real, pero nos alejaríamos definitivamente de las restricciones de bajo coste. Así pues, la única posibilidad es la de estudiar agresivamente modificaciones algorítmicas que permitan alcanzar al mismo tiempo ambos objetivos.

Por tanto, el siguiente paso, cubierto en el próximo capítulo, es realizar las modificaciones algorítmicas necesarias para conseguir la ejecución en tiempo real del algoritmo, al tiempo que se mantienen las condiciones de bajo coste.

Capítulo 5

Una señal inequívoca de amor a la verdad es no mantener ninguna proposición con mayor seguridad de la que garantizan las pruebas en las que se basa.

John Locke (1632-1704).
Filósofo y escritor inglés.

Lo opuesto de una formula correcta es una formula falsa. Pero lo opuesto de una verdad profunda puede ser muy bien una verdad profunda.

Niels Henrik David Bohr (1885-1962)
Físico danés. Premio Nóbel de física en 1922.

Algoritmos de súper-resolución no iterativos sobre la plataforma *Picasso*

5.1 Introducción

En este capítulo se mostrarán los diferentes algoritmos de súper-resolución no iterativos desarrollados sobre la plataforma de compresión híbrida de vídeo *Picasso* y la calidad de imagen que se logra con cada uno de ellos así como sus campos de aplicación. En contraposición con los algoritmos desarrollados en el capítulo anterior, que necesitaban varias iteraciones para alcanzar un resultado de alta calidad, en este capítulo se presentarán algoritmos no iterativos, mucho más aptos para su funcionamiento en tiempo real.

Tal vez la principal aportación de esta tesis descanse precisamente en el novedoso método ideado para poder conseguir mejoras de súper-resolución sin tener que recurrir a iteraciones. El capítulo se ha dividido en dos grande apartados. Un primer apartado dedicado

a los algoritmos no iterativos de súper-resolución para imágenes estáticas y un segundo apartado dedicado a algoritmos no iterativos de súper-resolución para vídeo. A su vez, el primer apartado se ha subdividido en una primera parte dedicada a describir el algoritmo base (v2.0) y una segunda que aborda la posibilidad de combinar un número variable de imágenes de entrada por cada imagen de súper-resolución de salida (v2.1). Esta última adaptación supone ya de por sí una versión operativa para video, pero adolece de requerir todavía una cantidad elevada de memoria, lo que compromete los requisitos de bajo coste. Al estar la memoria comprimida en el dominio de la DCT [KVP00], es posible incorporar la memoria del bucle de codificación dentro del circuito integrado (memorias *on-chip*), con todas las ventajas que ello reporta: disminución del coste al usar un solo integrado, disminución del consumo de potencia y aumento de la velocidad, al eliminar las comunicaciones fuera del integrado. Sin embargo, si el algoritmo de súper-resolución superara la memoria disponible *on-chip*, entonces tendríamos que añadir memoria externa, perdiendo las ventajas anteriormente enumeradas.

Por esa razón, el siguiente apartado de este capítulo aporta una solución al problema de las memorias mediante una reutilización de la última imagen de súper-resolución obtenida. Esta última imagen está almacenada en memoria interna y se utiliza para aumentar la resolución, de la nueva imagen de baja resolución que entra al sistema, formando una nueva imagen de súper-resolución que servirá a su vez para mejorar la siguiente imagen de baja resolución. Como puede apreciarse, se trata de un sistema realimentado que va aumentando la calidad de las imágenes a medida que entra nueva información de entrada. La primera imagen se obtiene como una mera interpolación de la entrada, ya que es toda la información disponible en ese momento. El bucle de realimentación tendrá que romperse cuando exista un cambio de contexto, ya que no podemos aumentar la calidad de una imagen basándonos en una anterior que no está relacionada con esta última. Asimismo, surge un problema importante cuando existen movimientos locales bruscos de objetos de la escena. En este caso, localmente ocurre un caso similar al anterior de cambio de contexto: el objeto desplazado no coincide con posiciones anteriores y no puede mejorarse su calidad. Además, causa una oclusión sobre posiciones anteriores que tampoco pueden mejorar su resolución. Éste es un problema complicado y raras veces tratado en el campo de la súper-resolución. En nuestro caso veremos cómo lo hemos resuelto estableciendo un sistema de umbrales dinámico. La versión del ASR para video (v3) incorpora un mecanismo eficiente para el cambio automático de contexto. Uno de los criterios de decisión ha sido actualizar cada píxel de la imagen de alta resolución en función de las diferencias entre los píxeles de la imagen actual y la de súper-resolución. Además se ha incluido un filtrado de las imágenes antes de realizar la estimación de movimiento, para aumentar la consistencia de los vectores de movimiento obtenidos. La

versión v3 recopila las mejores opciones de todas las versiones anteriores para constituir la propuesta de algoritmo final de súper-resolución, en tiempo real y bajo coste para secuencias de video.

5.2 Algoritmos de súper-resolución no iterativos para imagen estática

Aunque las versiones iterativas ofrecen buena calidad de imagen mapeadas dentro de un codificador híbrido de video, se hace patente la necesidad de crear un nuevo algoritmo que usando los mismos recursos sea capaz de operar en un solo ciclo, es decir, un algoritmo no iterativo. La idea principal se basa en las siguientes consideraciones:

- Cada imagen aporta nueva información que debe ser combinada en una nueva rejilla de forma adecuada.
- Es imposible saber *'a priori'* (para el campo de aplicaciones del algoritmo de súper-resolución) la posición de los nuevos datos y si van o no a constituir realmente información nueva, por lo que hay que ponderar el impacto de la nueva información en la nueva imagen.

En base a estas consideraciones, se ha creado el siguiente algoritmo:

1. Inicialmente se toma la primera imagen de baja resolución y se sitúan sus píxeles en una rejilla de alta resolución, dejando a cero los píxeles no cubiertos. A este proceso se le ha denominado **'upsample_holes'**. Dado que el incremento de tamaño es de un factor de dos en ambas direcciones, en la FIGURA 5.1 puede

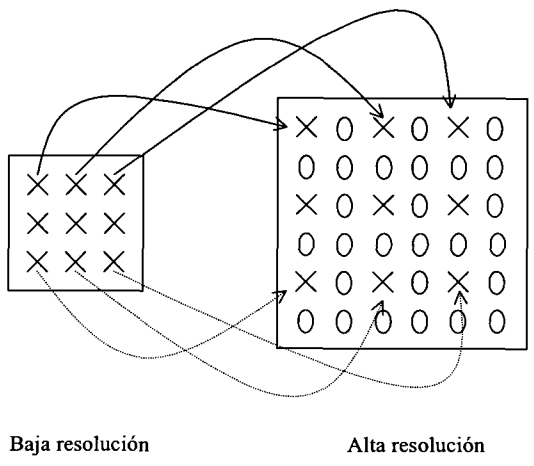


FIGURA 5.1. Mapeo de los píxeles de la imagen de baja resolución en la rejilla de alta resolución, dejando huecos en los píxeles ausentes.

verse la ubicación de los píxeles en las rejillas de alta y baja resolución.

2. A continuación, se generan las 'contribuciones' de los píxeles. Las contribuciones son pesos que se le dan a cada píxel para indicar la cantidad de información que aporta en cada posición. Debido a que el incremento de tamaño es de 2^2, el valor inicial de la contribución será de 4. Estas contribuciones están expresadas sobre la rejilla de alta resolución. Así, las contribuciones de la imagen de la FIGURA 5.1 se muestran en la FIGURA 5.2, indicando que los píxeles añadidos tienen la contribución máxima y el resto están a cero.
3. Ahora se calculan los movimientos relativos entre la siguiente imagen leída y la primera imagen o referencia. Estos desplazamientos se almacenan en memoria ya que serán usados a continuación.
4. La nueva imagen es sometida al mismo procedimiento que el indicado en los pasos 1 y 2, es decir se adapta a la rejilla de alta resolución, dejando a cero los píxeles ausentes y se generan las contribuciones iniciales.

4	0	4	0	4	0
0	0	0	0	0	0
4	0	4	0	4	0
0	0	0	0	0	0
4	0	4	0	4	0
0	0	0	0	0	0

FIGURA 5.2. Contribuciones de la imagen inicial.

5. En este paso, tanto la nueva imagen sobre la rejilla de alta resolución como sus contribuciones asociadas se compensan en movimiento hacia la imagen de referencia. En las contribuciones compensadas quedará reflejada la contribución real de los nuevos píxeles a la imagen de referencia de alta resolución.
6. Se realiza ahora la suma lineal de las imágenes y las contribuciones iniciales con las compensadas en movimiento. Este efecto de promedio reducirá además el ruido de la imagen resultante.
7. Se repiten los pasos 3 a 6 para las siguientes imágenes de entrada.

8. Una vez acabado el paso 7, tenemos una imagen de alta resolución con la suma de todas los píxeles compensados y una memoria con la suma de todas las compensaciones. A continuación ajustamos la imagen de alta resolución en función de las compensaciones, tal y como indica la ecuación (5.1) .

$$SR(i,j) = 4 \cdot \frac{\sum_{fr=1}^{4} Motion\_Compesate(Upsample\_Holes(LR\_I[fr]))}{\sum_{fr=1}^{4} Motion\_Compesate(contributions[fr])} \qquad (5.1)$$

9. Después del ajuste, es posible que algunas posiciones de píxeles hayan quedado vacías, es decir, que en el conjunto de imágenes leídas dicha posición haya quedado sin aportación de nueva información. Este caso se reflejará con un cero tanto en la imagen de alta resolución como en sus contribuciones y la única solución es interpolar los ceros resultantes usando los datos de alrededor distintos de cero. Sin embargo, no podemos sacar la conclusión de que un cero

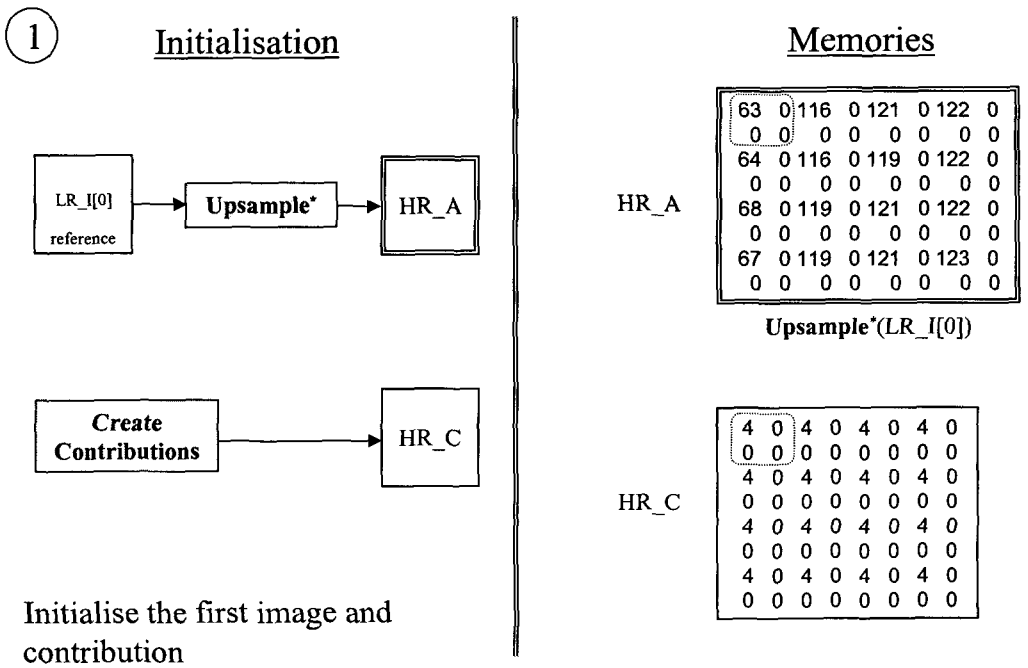
```
nr_frames = scale*scale, M'= scale*M, N'= scale*N
HR_B, HR_S, HR_T, HR_T2, HR_C and HR_S2 all of 8 bits and size [M'][N'].
HR_A is 10 bits and size [M'][N'].
LR_I[M][N] for the motion estimation
Read a set of aliased Low-Resolution  images in LR_I[#nr_frames][M][N]
MV_ref2fr[0] = 0,0

FOR fr = 1 .. nr_frames-1
   MV_ref2fr[fr] = Motion_Estimation( LR_I[fr], LR_I[0])
   MV_ref2fr[fr] = 2 .* MV_ref2fr[fr]
END FOR

HR_A  = Upsample_Holes(LR_I[0])
HR_C  = Create_image_contributions()

FOR fr = 1 .. nr_frames-1
    HR_S  = Upsample_Holes(LR_I[fr])
    HR_S2 = Create_image_contributions()

    HR_T  = Motion_Compensation(HR_S, MV_ref2fr[fr])
    HR_T2= Motion_Compensation(HR_S2, MV_ref2fr[fr])
    HR_A  = HR_A + HR_T
    HR_C  = HR_C + HR_T2
END FOR

HR_A = 4*HR_A/HR_C

IF  (HR_C(i)==0)  THEN
   HR_B.lum = Interpolate(HR_A.lum)
ELSE
    HR_B.lum = HR_A.lum
END IF
HR_B.chrom = HR_A.chrom
```

de la imagen implica que dicho valor debe ser interpolado, ya que el cero es un valor posible y válido de la imagen. Un píxel será interpolado sólo si su contribución final, "su peso", es cero.

En la FIGURA 5.3 se muestra este algoritmo en pseudo-código, usando las memorias y recursos del codificador híbrido de video *Picasso*.

Obsérvese que, con respecto a las versiones iterativas del capítulo anterior, no sólo se ha eliminado el carácter iterativo, sino que además se ha eliminado una de las estimaciones de movimiento necesarias. Dado que la estimación de movimiento se sigue haciendo en baja resolución, y la compensación de movimiento en alta resolución, sigue siendo necesario multiplicar por dos los vectores obtenidos. Inicialmente el número de *frames* se ha establecido en cuatro, siguiendo con la filosofía de los algoritmos iterativos, debido a que éste es el incremento total de tamaño de la imagen de súper-resolución.

Para entender mejor el funcionamiento del algoritmo, mostraremos su funcionamiento paso a paso, usando el macro-bloque (0,2) del *frame* 0 de la luminancia de la secuencia de prueba *KRANT*. Además, con la intención de cubrir todas las contingencias, se han escogido unos vectores de movimiento que generen huecos en la imagen final, aunque hay que resaltar que no es el caso usual. La FIGURA 5.4 muestra la etapa de inicialización de la imagen de súper-resolución y de la memoria de contribuciones. Las memorias se muestran en recuadros de doble línea para diferenciarlas de los operadores, que se muestran con línea simple. La primera imagen de baja resolución LR_I[0] será la referencia de movimiento y se almacenará



Upsample* ≡ Upsample_Holes

en la memoria HR_A. A su vez las contribuciones se almacenarán en HR_C.

Una vez inicializadas las memorias, nos internamos en el bucle para tratar las tres imágenes restantes, comenzando por la estimación del movimiento, como se muestra en la FIGURA 5.5, donde también se muestran los vectores de movimiento usados en alta y en baja resolución. Estos vectores se almacenan en la memoria MV_ref2fr[0..3] para usarlos posteriormente.

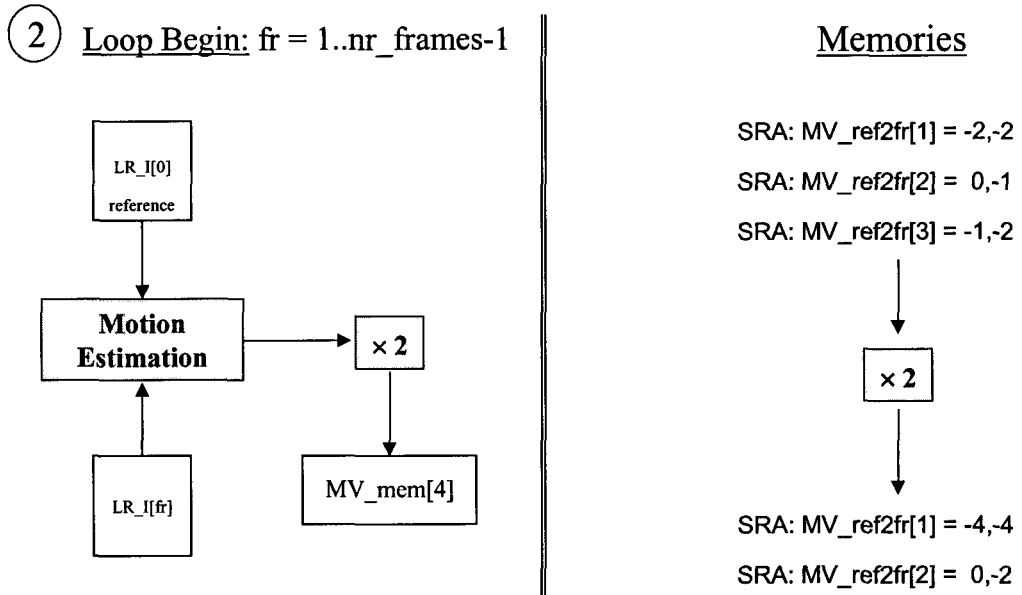


FIGURA 5.5. Estimación del movimiento del resto de las imágenes con referencia a la primera imagen.

El siguiente paso (FIGURA 5.6) es inicializar el resto de las memorias con las imágenes de entrada llevadas a alta resolución y sus contribuciones correspondientes. Debido a que lo que interesa es la suma de todas las imágenes, en realidad las memorias usadas (HR_S para las imágenes y HR_S2 para las contribuciones) son reutilizadas para cada nueva imagen de entrada. En la FIGURA 5.6 se muestran además los diferentes valores que tomará HR_S para cada imagen de entrada, indexadas como [1], [2] y [3]. Nótese sin embargo que las contribuciones iniciales son las mismas para las cuatro imágenes.

Una vez creadas las imágenes de alta resolución, debemos compensarlas en movimiento hacia la imagen de referencia, y lo mismo hacemos con las contribuciones, pero con un objetivo diferente: determinar la contribución de cada píxel a la imagen de referencia. Los resultados son almacenados en HR_T y HR_T2.

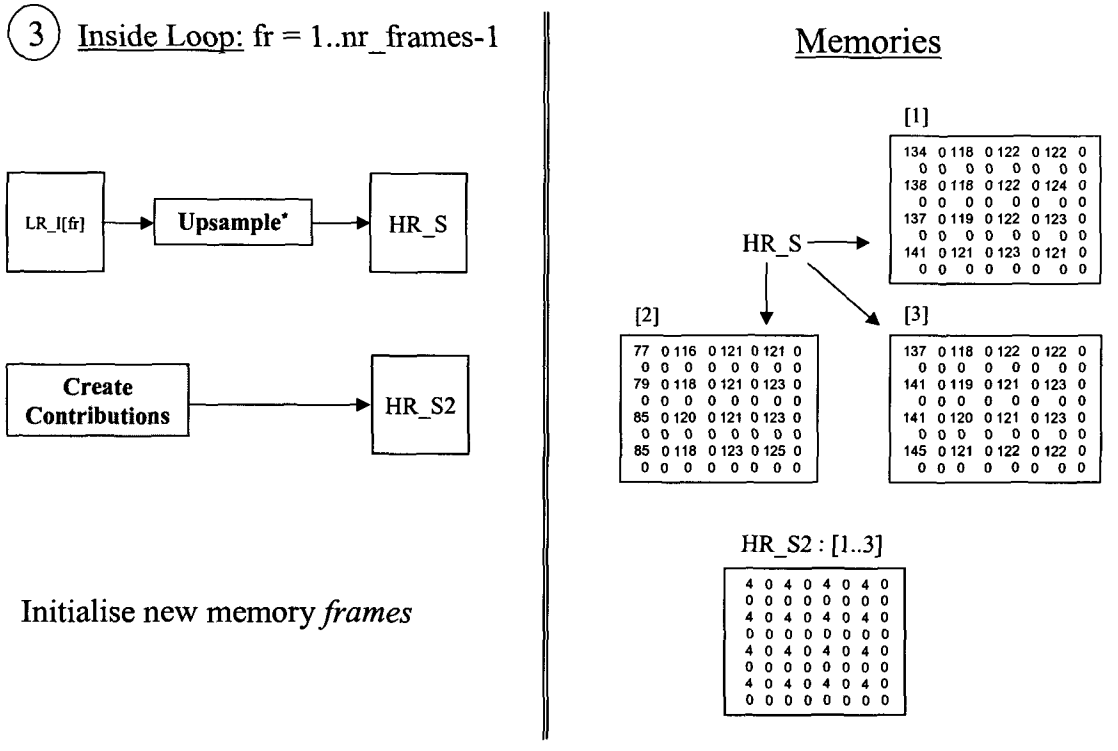


FIGURA 5.6. Inicialización de las imágenes de alta resolución y sus contribuciones dentro del bucle principal de procesamiento.

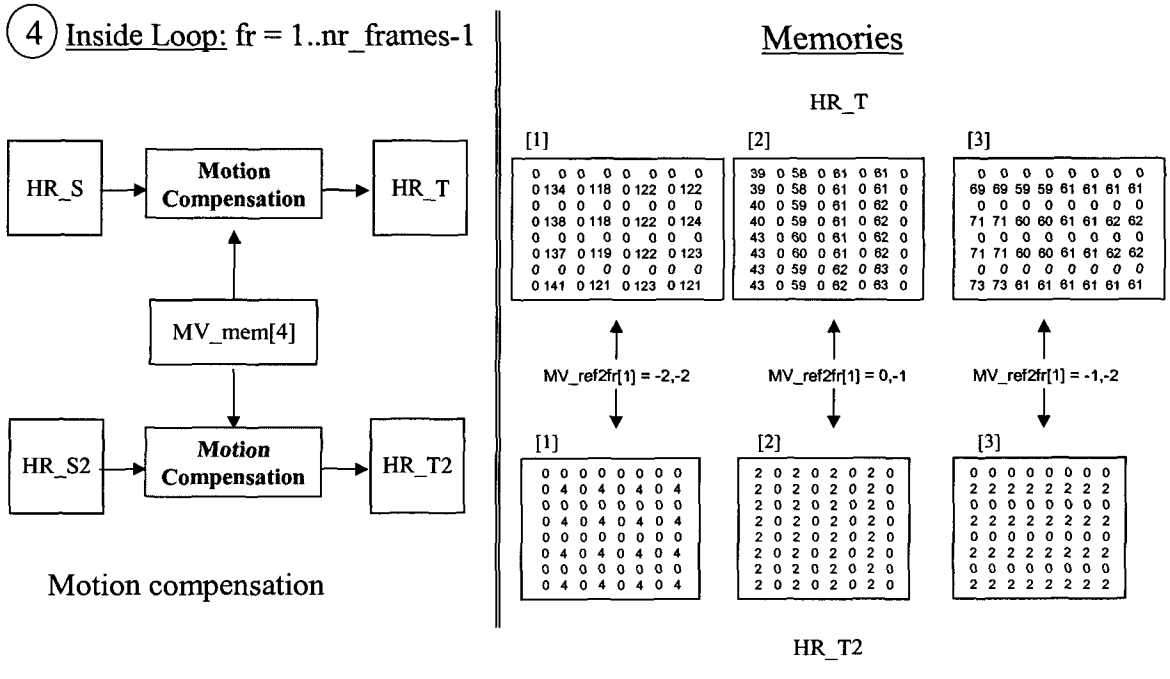


FIGURA 5.7. Compensación de movimiento de las imágenes de entrada en alta resolución y de sus contribuciones dentro del bucle principal de procesamiento.

Una vez compensadas las imágenes y sus contribuciones, realizamos la suma de todas ellas. Desde el punto de vista de la imágenes, supone la superposición de los datos nuevos y el promedio de aquellos datos que estén repetidos. Desde el punto de vista de las contribuciones, supone totalizar el peso de cada píxel sobre la imagen final. En la FIGURA 5.8 se muestra cómo las imágenes son acumuladas en HR_A y las contribuciones en HR_C. En el último paso del bucle, en HR_A y HR_C quedarán acumulados los valores finales, donde cabe resaltar el valor cero que queda en el segundo píxel superior de la celda base. Esto quiere decir que las imágenes de entrada usadas no han sido capaces de aportar nueva información que encaje en dichas posiciones, ya que se ha producido un muestreo repetitivo en las mismas posiciones de píxel.

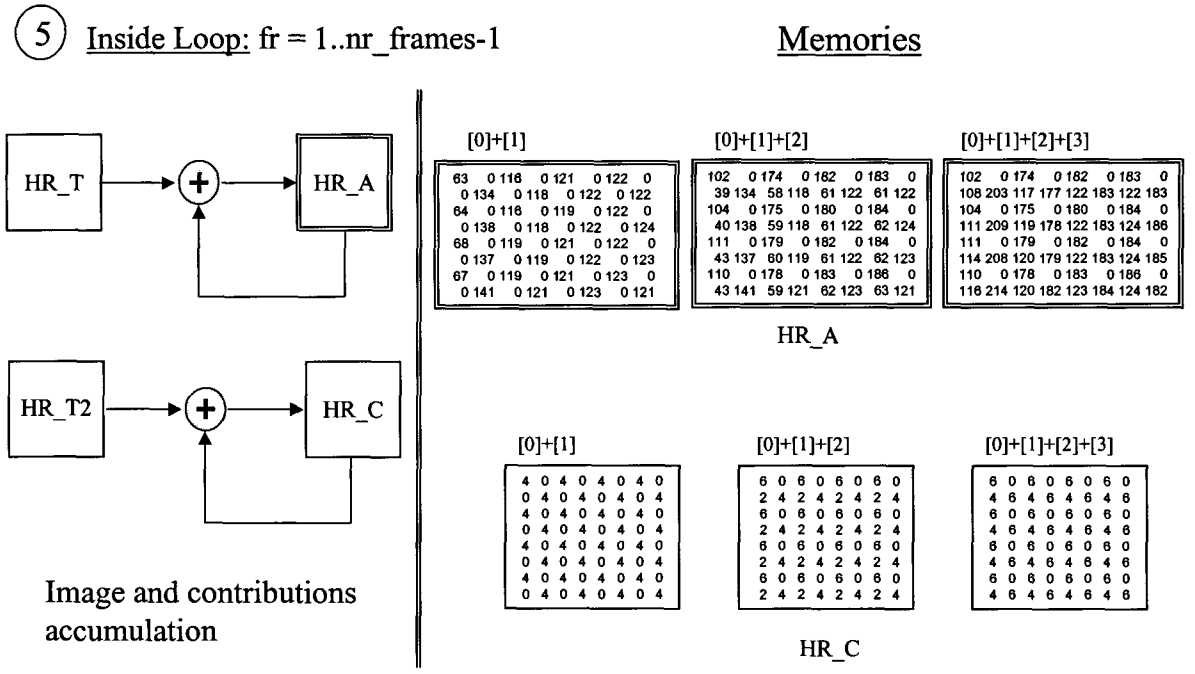


FIGURA 5.8. Compensación de movimiento de las imágenes de entrada en alta resolución y de sus contribuciones dentro del bucle principal de procesamiento.

Una vez terminado el bucle, es necesario ajustar los valores acumulados en función de los pesos de cada píxel. Esto se hace aplicando la ecuación (5.1) a la memoria acumulativa HR_A usando HR_C como contribuciones, como se ve en la FIGURA 5.9. Si la suma de las contribuciones es cero, evidentemente no podremos dividir por su valor, en cuyo caso mantenemos el valor de cero en la imagen.

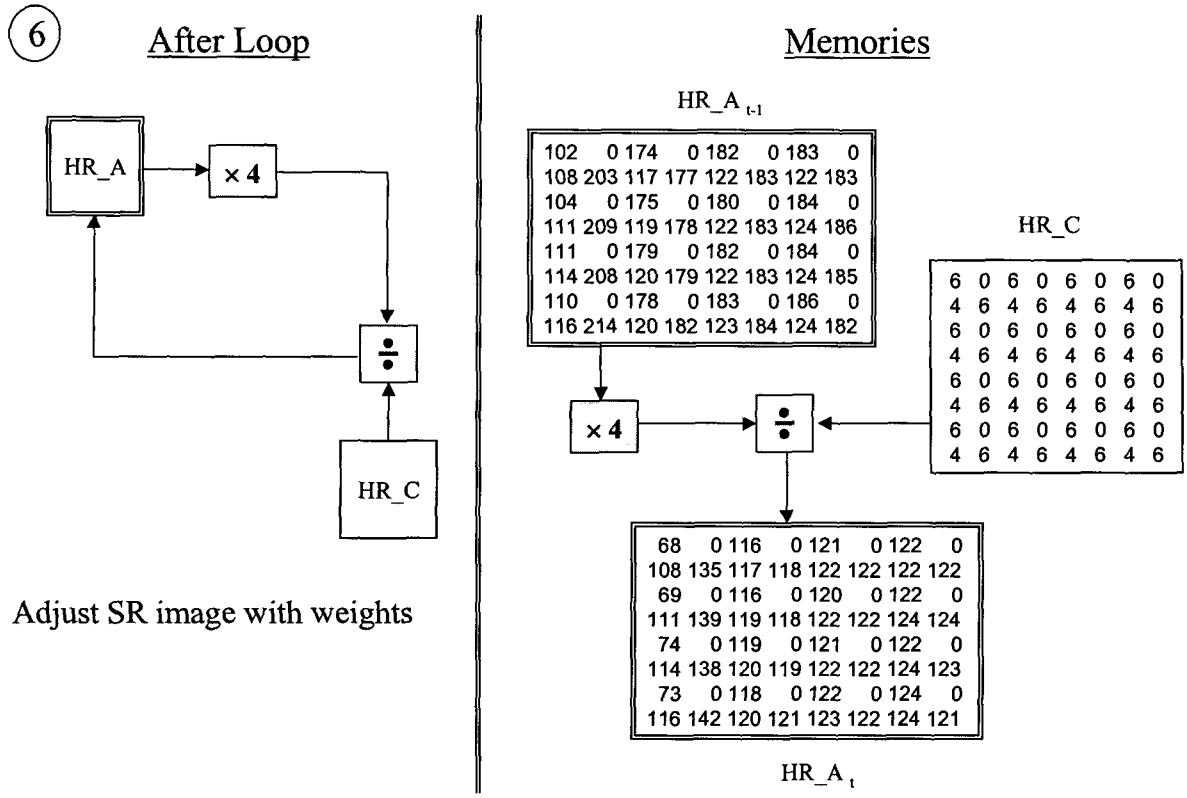


FIGURA 5.9. Compensación de movimiento de las imágenes de entrada en alta resolución y de sus contribuciones dentro del bucle principal de procesamiento.

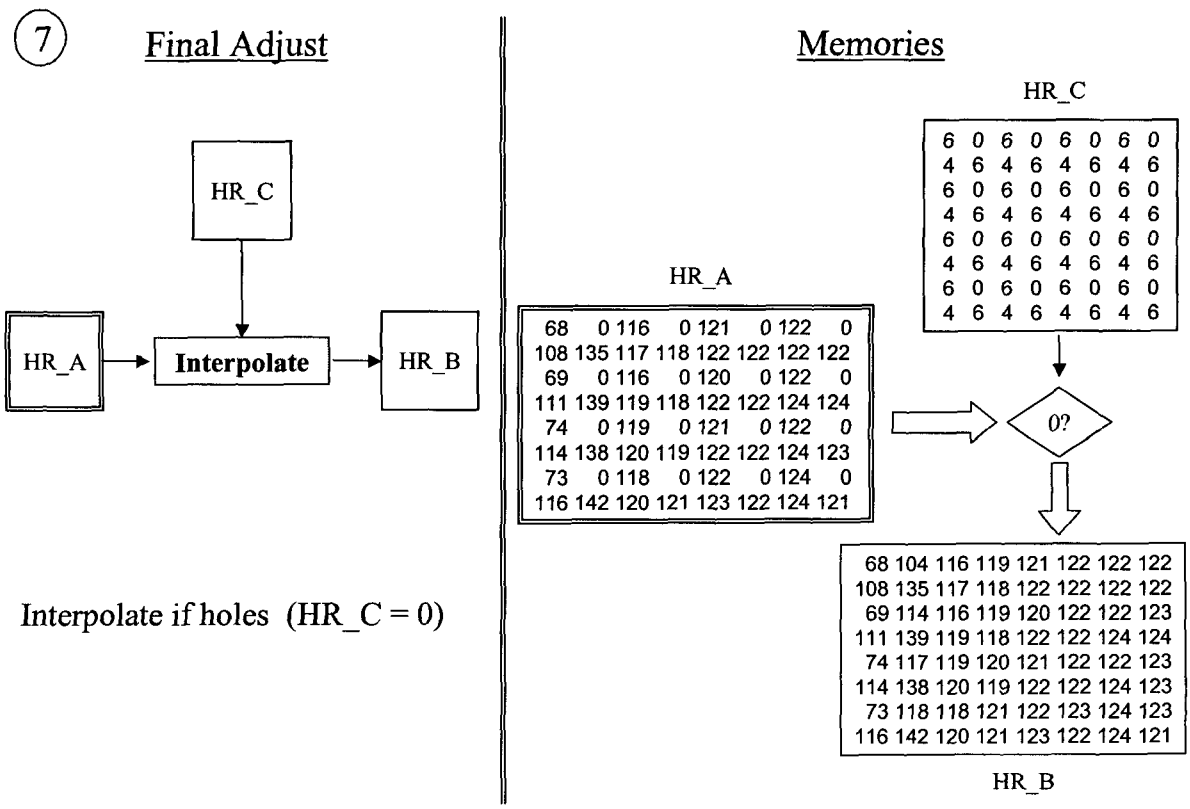


FIGURA 5.10. Ajuste final por interpolación sólo en los ceros de imagen con contribución cero.

Los valores que hayan quedado a cero por no haber información disponible, serán interpolados en la última etapa del algoritmo (FIGURA 5.10), consultando los valores de las contribuciones para evitar interpolar valores ceros propios de la imagen. Al igual que en los algoritmos iterativos, la imagen de súper-resolución obtenida se almacena en HR_B. Este principio de usar HR_B para dicho fin será mantenido para todas las versiones de todos los ASR, por claridad.

5.2.1 Modificaciones en la arquitectura Picasso

La arquitectura *Picasso* es una arquitectura usada para compresión/descompresión de datos que afecta a otras aplicaciones.

El compensador de movimiento implementado en la arquitectura *Picasso* está pensado para que las imágenes resultantes no presenten distorsiones apreciables a la hora de mostrar las imágenes descomprimidas, y en este sentido, cuando desplaza una imagen fuera de sus límites físicos, rellena la parte vacía replicando los bordes. Debido a que los vectores de movimiento no suelen ser de valores elevados comparados con el tamaño de las imágenes y a la menor atención que le presta el ojo humano a los bordes frente al centro de las imágenes, este efecto apenas se aprecia. Sin embargo, cuando se desean realizar mejoras de imagen aplicando súper-resolución, la introducción artificial de datos inexistentes provoca alteraciones de calidad importantes en los bordes. Como resulta obvio, lo deseable en estos casos es que el estimador de movimiento rellenase los valores vacío con ceros para de esta forma dar una oportunidad al algoritmo de rellenar dichos huecos con valores válidos procedentes de otras imágenes, sin alterar por otra parte el funcionamiento del algoritmo, ya que este no espera la generación espontánea de "nuevos" datos generados por el compensador de movimiento, que además no han sido tenidos en cuenta en las etapas anteriores del algoritmo.

Por lo tanto, la arquitectura *Picasso* debe ser modificada de forma que el estimador de movimiento replique los bordes cuando trabaje en modo de compresión / descompresión de imágenes y en cambio inyecte ceros cuando trabaje en modo de súper-resolución. Debe tenerse en cuenta asimismo la forma especial de *Upsample* requerida por la luminancia, que inyecta tres ceros por cada nuevo píxel.

5.2.2 Ajustes para la aplicación del ASR en la crominancia

Debido al diferente formato de muestreo usado para la luminancia y para las crominancias es necesario realizar algunas modificaciones en el algoritmo propuesto para poderlo aplicar también a las crominancias.

En primer lugar, la forma de realizar el paso a las rejillas de alta resolución o *upsample*, no puede ser la misma que la de la luminancia. Este hecho se refleja en la FIGURA 5.11, donde se constata cómo cada píxel de crominancia afecta a cuatro píxeles de luminancia, y por lo tanto, al ser llevados a la rejilla de alta resolución de crominancia, cada píxel ha de replicarse cuatro veces, para mantener la coherencia cromática, como se muestra en la FIGURA 5.12 (a).

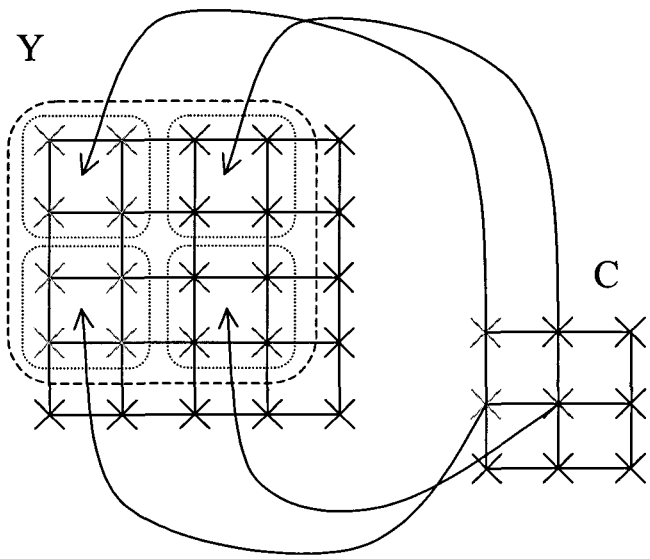
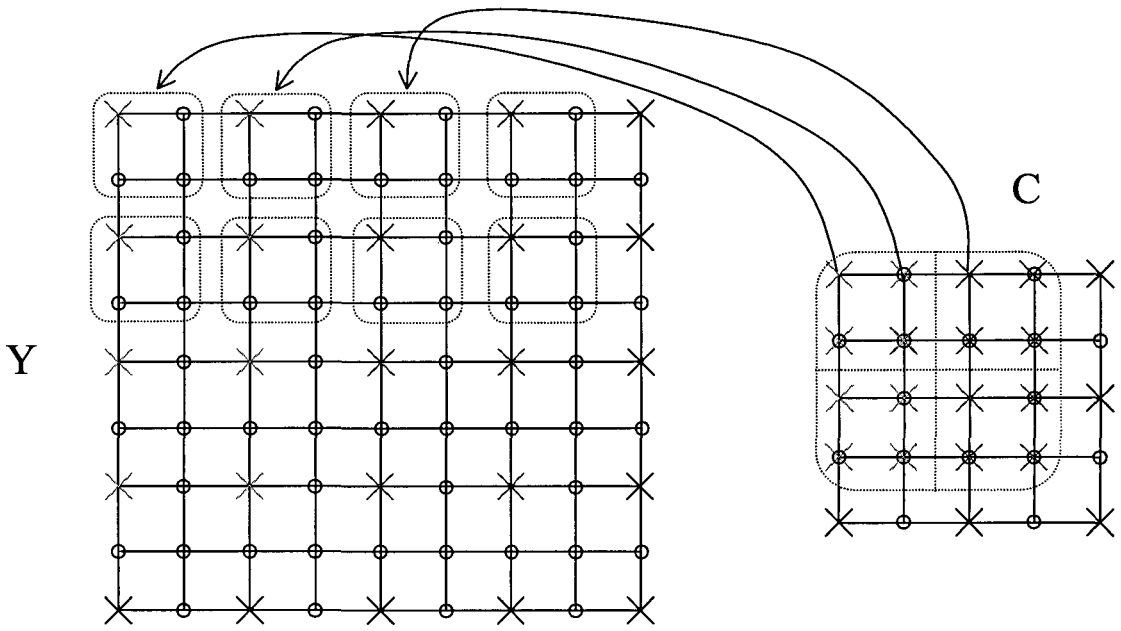


FIGURA 5.11. Relación entre los píxeles de crominancia y luminancia en el formato de muestreo YCbCr 4:2:0.

Por este mismo motivo, las contribuciones iniciales no pueden ser las mismas que las de la luminancia. Al no haber un relleno con ceros, todos los píxeles de crominancia han de tener inicialmente el mismo peso contributivo. En la FIGURA 5.12 (b) se muestran las contribuciones iniciales para la imagen de luminancia y para las imágenes de crominancia.

Al ser las matrices de las contribuciones de las crominancias homogéneas, es patente que no serán afectadas por la compensación de movimiento, por lo que en apariencia podría

ahorrarse este paso. Sin embargo, no podemos olvidar que la compensación de movimiento modificada para súper-resolución inyecta ceros en los bordes, tanto para la luminancia como para las crominancias, por lo que el mantener la misma estructura de compensación de las contribuciones que en las luminancias, nos permitirá corregir los efectos de borde en las crominancias.



(a)

Contribuciones luminancia:	4 0 0 0	
Contribuciones crominancia:	4 4 4 4	

(b)

FIGURA 5.12. Mapeo de la crominancia C a la rejilla de alta resolución mediante replicado de los píxeles y su relación con la luminancia Y (a). Contribuciones iniciales de las imágenes de luminancia y crominancias (b).

5.2.3 Algoritmo de súper-resolución no iterativo básico (v2.0)

Todas las ideas expuestas hasta ahora son recogidas en la versión v2.0 del ASR, que se muestra en la FIGURA 5.13.

En esta versión, las memorias son todas de 8 bits, excepto HR_A que es de 10 bits, al tratarse de una memoria acumulativa. Dado que el número mayor que puede alcanzar un píxel

es 2^8-1 = 255, el mayor número que podrá ser acumulado tras sumar cuatro imágenes es 255·4 = 1020, para el que hacen falta 10 bits (2^{10}-1 = 1023). La memoria de contribuciones HR_Cont no es necesario hacerla de 10 bits, ya que los valores de las contribuciones son pequeños. El valor máximo después de acumular cuatro contribuciones de tamaño 4 es 16, que cabe perfectamente en 8 bits.

```
* Non iterative super-resolution algorithm  v2.0

  Set the value of the improvement: scale
  nr_frames = scale*scale, M'= scale*M, N'= scale*N
  HR_B, HR_S, HR_T, HR_T2, HR_Cont and HR_S2 all of 8 bits and size [M'][N']
  HR_A is 10 bits and size [M'][N']
  LR_I[M][N] for the motion estimation
  Read a set of aliased Low-Resolution  images in LR_I[#nr_frames][M][N]

// First Image is the reference
MV_ref2fr[0] = 0,0

FOR fr = 1 .. 3
   MV_ref2fr[fr] = Calc_Motion_Estimation ( LR_I[fr], LR_I[0])
   IF Global_Motion THEN Select_global_motion_vector()
   MV_ref2fr[fr] = 2 .* MV_ref2fr[fr]
END FOR

HR_A.lum    = Upsample_Holes(LR_I[0].lum)                    [SR_INIT_A]
HR_A.chrom = Upsample_Neighbours(LR_I[0].chrom)

HR_Cont     = Create_image_contributions()                   [SR_INIT_CONT]

FOR fr = 1 .. 3
   HR_S.lum    = Upsample_Holes(LR_I[fr].lum)                [SR_UPSAMPLE]
   HR_S.chrom = Upsample_ Neighbours(LR_I[fr].chrom)

   HR_S2 = Create_image_contributions()

   HR_T    = Motion_Compensation*(HR_S, MV_ref2fr[fr])       [SR_MOT_COMP]
   HR_T2   = Motion_Compensation*(HR_S2, MV_ref2fr[fr])      [SR_MOT_COMP_CONT]
   HR_A    = HR_A + HR_T                                     [SR_ADD]
   HR_Cont = HR_Cont + HR_T2                                 [SR_ADD_CONT]
END FOR

HR_A = 4*HR_A/HR_Cont
                                                             [SR_ADJUST_A]
IF (Hole)   HR_B = Interpolate(HR_A)                         [SR_UPDATE]
Else        HR_B = HR_A

Clip(HR_B, 0, 255)

High Resolution result image in HR_B
```

FIGURA 5.13. Pseudo-código del algoritmo no iterativo v2.0.

Se le ha puesto un asterisco al '*motion compensation*' para indicar que es diferente al usado en las operaciones de compresión de imágenes, en el sentido de que inyecta ceros en los bordes en lugar de replicarlos.

5.2.3.1 Diagrama de bloques y requerimientos de memoria de la versión v2.0

En la FIGURA 5.14 se muestra el diagrama de bloques del flujo de datos de la versión v2.0 del ASR mostrado en la FIGURA 5.13.

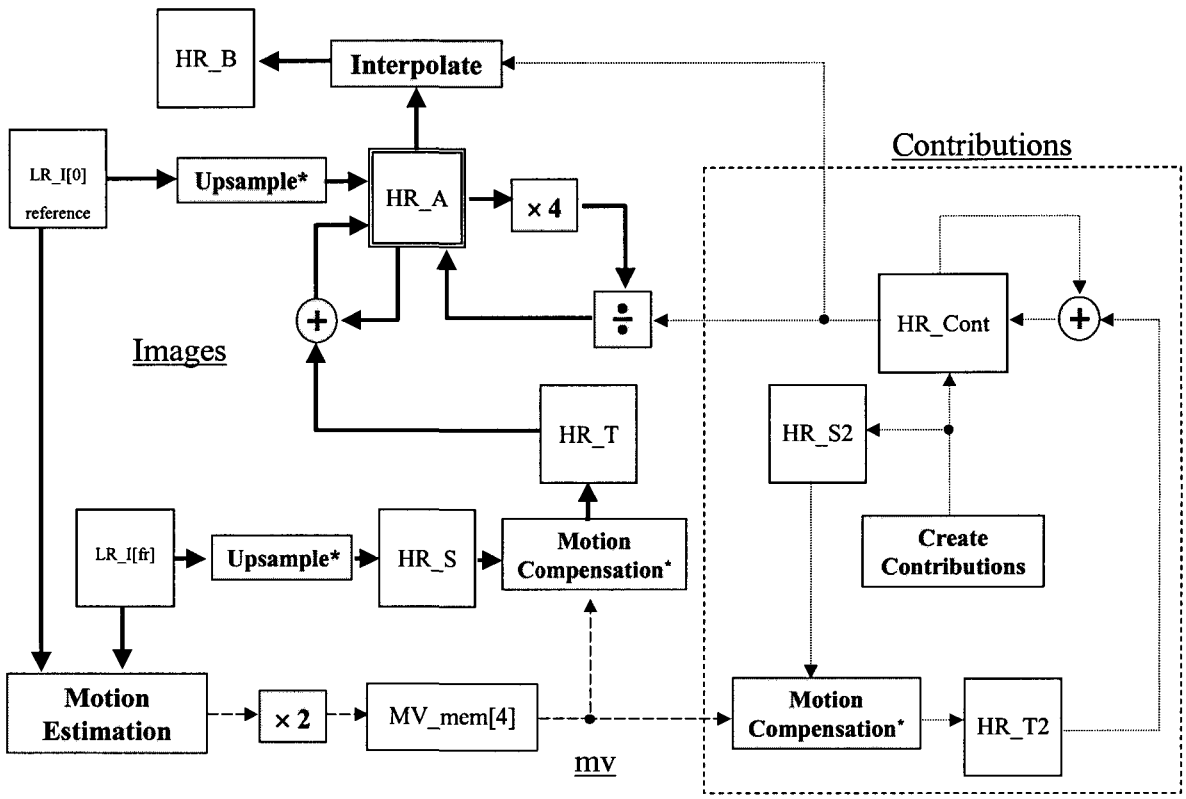


FIGURA 5.14. Diagrama de bloques del algoritmo de súper-resolución v2.0.

El diagrama de bloques se ha dividido, por un lado en la zona ocupada del procesamiento de imágenes, que hace uso de las memorias HR_A, HR_T, HR_S y LR_I[0..3], además de guardar los vectores de movimiento en MV_mem[0..3] y por otro lado, en una zona para el procesamiento de las contribuciones, que hace uso de las memorias HR_S2, HR_T2 y HR_Cont. Para resaltar las relaciones entre ellas, se han puesto en línea continua los flujos que llevan imágenes, en línea punteada los flujos que llevan contribuciones

y en línea discontinua los flujos que llevan los vectores de movimiento. Además, las funciones de *'upsample'* y *'motion compensation'*, se han marcado con un asterisco para remarcar su funcionamiento diferente cuando se usan en súper-resolución.

En la TABLA 5.1 se muestra un resumen de los requerimientos de memoria de la versión 2.0 del ASR, tal y como se recoge en la FIGURA 5.14. Debe recordarse que todas las memorias son de 8 bits, excepto HR_A que es de 10 bits.

Denominación	Memoria		
	Luminancia (bits)	Crominancia (bits)	Total (bits)
HR_A	*(2·mb_x·2·mb_y·16·16·10)*	*(2·mb_x·2·mb_y·8·8·2·10)*	*15,360·mb_x·mb_y*
HR_B	*(2·mb_x·2·mb_y·16·16·8)*	*(2·mb_x·2·mb_y·8·8·2·8)*	12,288·*mb_x·mb_y*
HR_S	*(2·mb_x·2·mb_y·16·16·8)*	*(2·mb_x·2·mb_y·8·8·2·8)*	12,288·*mb_x·mb_y*
HR_S2	*(2·mb_x·2·mb_y·16·16·8)*	*(2·mb_x·2·mb_y·8·8·2·8)*	12,288·*mb_x·mb_y*
HR_Cont	*(2·mb_x·2·mb_y·16·16·8)*	*(2·mb_x·2·mb_y·8·8·2·8)*	12,288·*mb_x·mb_y*
3 Stripes HR	(2·3·2·mb_y·16·16·8)	(2·3·2·mb_y·8·8·2·8)	36,864·mb_y
LR_I[0]	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
LR_I[1]	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
LR_I[2]	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
LR_I[3]	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
MV_mem[0]	(mb_x·mb_y·8)	0	8· *mb_x·mb_y*
MV_mem[1]	(mb_x·mb_y·8)	0	8· *mb_x·mb_y*
MV_mem[2]	(mb_x·mb_y·8)	0	8· *mb_x·mb_y*
MV_mem[3]	(mb_x·mb_y·8)	0	8· *mb_x·mb_y*
Total (bits)	mb_y · (51,232· mb_x + 24,576)	mb_y · (25,600· mb_x + 12,288)	mb_y · (76.832·mb_x+ 36,864)

TABLA 5.1. Resumen de la memoria utilizada por la versión v2.0 del algoritmo de súper-resolución

Las memorias HR_T y HR_T2 no se incluyen en estas medidas ya que se usan en esta versión del algoritmo para evitar el solapamiento de datos al realizar la compensación de movimiento, pero este problema no existirá en la versión *hardware* que se implementa finalmente sobre *Picasso*, al existir de hecho un *buffer* de 3 filas de macro-bloques que permite evitar precisamente este problema. El la TABLA 5.2 se muestran los requerimientos totales de memoria para diferentes tamaños de imágenes.

En la FIGURA 5.15 se muestran los datos de la TABLA 5.2 expresados en kilo-bytes. Los requerimientos de memoria son mayores que en el caso de los algoritmos iterativos debido a que estamos reservando una memoria entera para las contribuciones. Más adelante veremos

que el ajuste de las contribuciones se puede hacer a nivel de macro-bloques, lo que nos ahorrará dos memorias de alta resolución.

Tamaño	mb_x	mb_y	Memoria (Kbytes)	Memoria (Mbytes)
SQCIF	8	6	477.19	0.47
QAVGA	9	7	622.37	0.61
QCIF	11	9	969.01	0.95
HAVGA	18	14	2,426.48	2.37
CIF	22	18	3,795.05	3.71
AVGA	36	28	9,579.94	9.36
VGA	40	30	11,389.69	11.12
4CIF	44	36	15,018.19	14.67
16CIF	88	72	59,748.75	58.35

TABLA 5.2. Memoria utilizada por la versión v2.0 del algoritmo de súper-resolución para diferentes tamaños de imágenes.

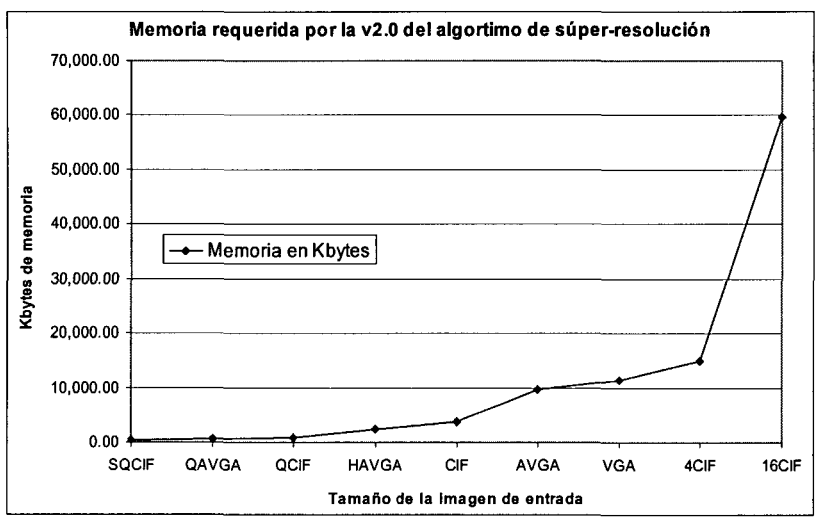


FIGURA 5.15. Memoria usada por el algoritmo de súper-resolución v2.0 para los tamaños de imágenes más usuales.

5.2.3.2 Resultados de simulación y análisis de calidad y comportamiento de la versión v2.0 usando precisión de ½ píxel

Con la intención de poder realizar una comparativa con las versiones iterativas anteriores, las primeras simulaciones se han realizado usando las mismas imágenes de prueba (KANTOOR, formato CIF (352×288), que se sub-muestrea a QCIF (176×144) como entrada del ASR para luego ampliarlo a formato CIF otra vez), el mismo *frame* inicial (*frame* 0) y los mismos desplazamientos. Por lo tanto, cualquier cambio en la calidad de las imágenes resultantes se deberá sólo a modificaciones algorítmicas, y en ningún caso a disponer de mayor cantidad o mejor calidad de las muestras de entrada. El conjunto de vectores usados es el de la Tabla 4.9 y por lo tanto su tipo es el indicado en la Tabla 4.10.

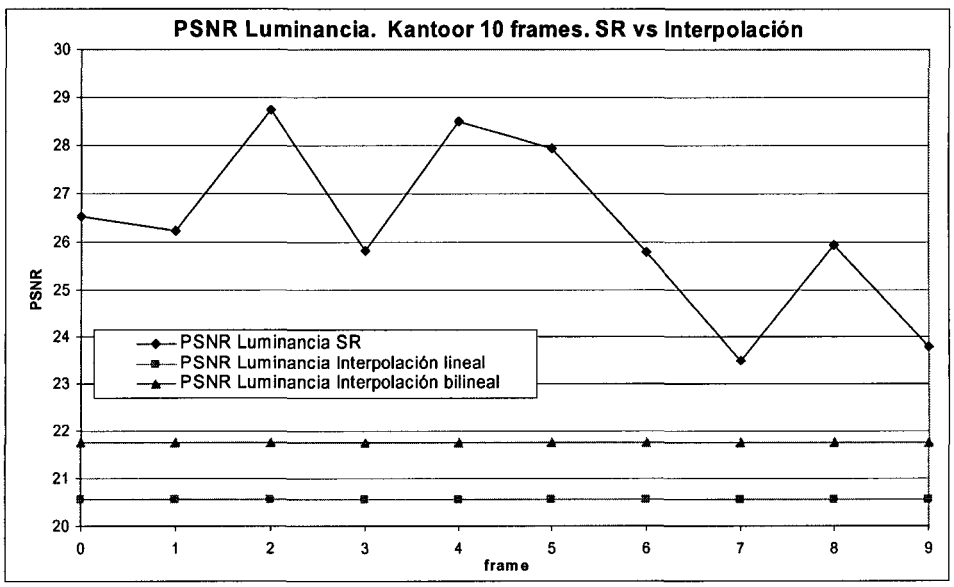


FIGURA 5.16. PSNR de la secuencia de salida usando la versión v2.0 del ASR frente a las imágenes interpoladas usando interpolación lineal de orden cero y bilineal.

Si comparamos estos resultados con los obtenidos en las versiones v1.2 y v1.3 (FIGURA 5.17) , vemos que la relación señal-ruido de pico PSNR de la imagen completa (con bordes) es significativamente más alta. Si además sustraemos los bordes del cálculo de la PSNR, obtenemos la gráfica de la FIGURA 5.18. En este segundo caso, los resultados para las imágenes tipo 'a' son verdaderamente espectaculares. La relación señal a ruido está limitada a 99.9 dB, que sería el valor que toma la PSNR cuando ambas señales son iguales, como ocurre en este caso. Esta asombrosa precisión es debida a que en el proceso de reconstrucción de las

imágenes se está en realidad siguiendo exactamente el proceso inverso al de generación de imágenes de prueba. No es de extrañar por lo tanto un resultado tan preciso. De hecho, el algoritmo de súper-resolución propuesto está inspirado en esta filosofía, que ha sido necesario luego validar usando imágenes reales e implementarlo con los recursos disponibles.

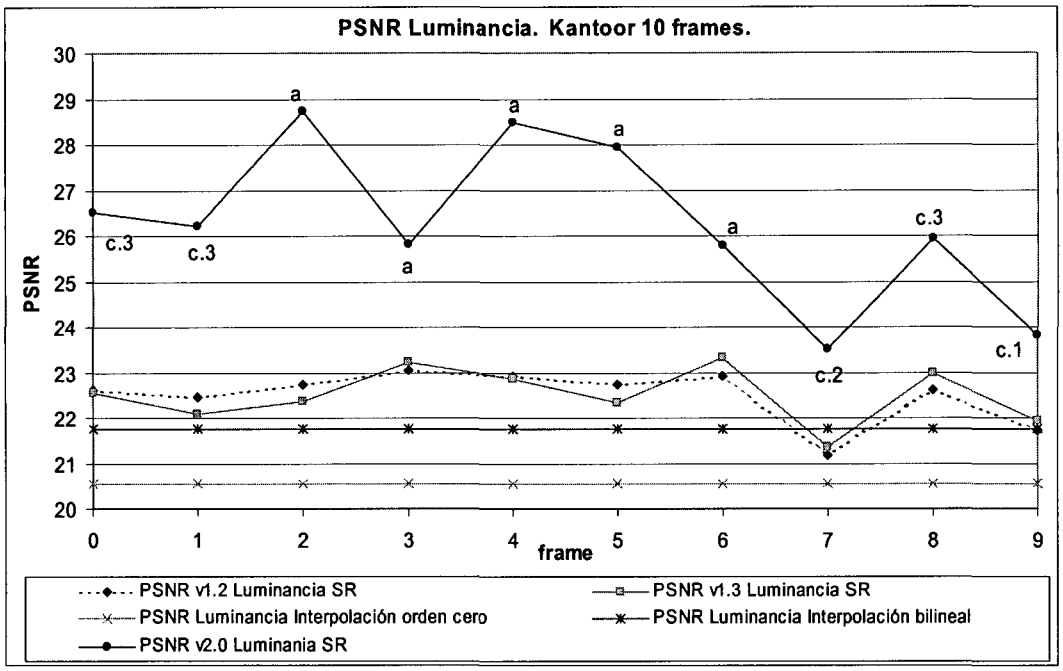


FIGURA 5.17. Comparación de la PSNR de la secuencia Kantoor de salida para la versión v2.0 del ASR frente a las imágenes interpoladas y a las obtenidas en los algoritmos iterativos, versiones v1.2 y v1.3.

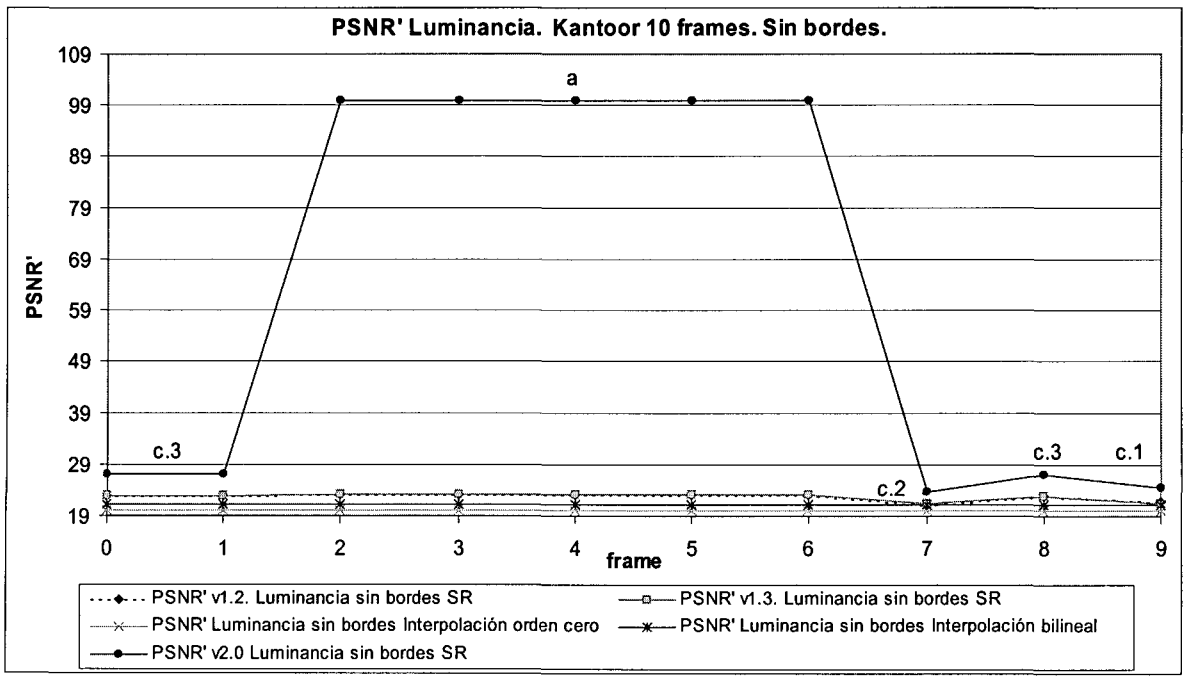


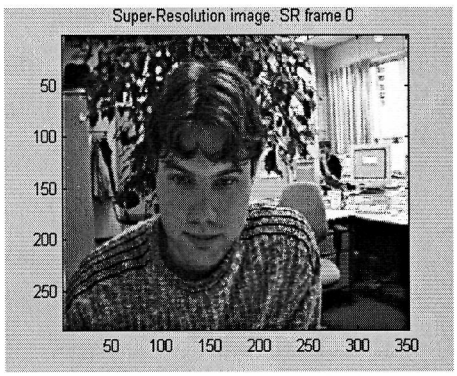
FIGURA 5.18. Comparación de la PSNR' sin bordes de versión v2.0 del ASR frente a las imágenes interpoladas y a las obtenidas en los algoritmos iterativos, versiones v1.2 y v1.3.

Al igual que en el capítulo anterior, en lugar de mostrar toda la secuencia de imágenes, mostraremos tan solo una de cada uno de los tipos principales: del tipo 'c.3' mostraremos el *frame* 0, del tipo 'a', mostraremos el *frame* 4, del tipo 'c.2' mostraremos el *frame* 7 y del tipo 'c.1' mostraremos el *frame* 9.

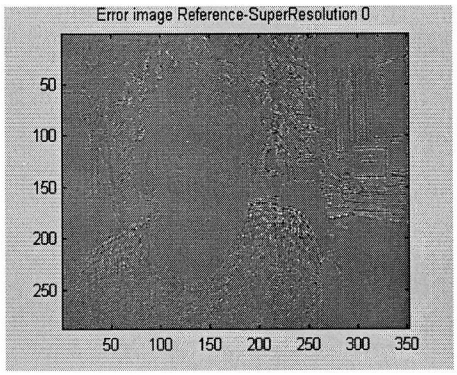
Las imágenes obtenidas de resolución son las mostradas en la FIGURA 5.19. A la izquierda, etiquetadas con extensión '1', se muestran las imágenes de tamaño CIF y a la derecha, etiquetadas con extensión '2' se muestran los errores asociados al compararlas con la imagen original. Al igual que en los casos anteriores, las imágenes de tipo 'a', (FIGURA 5.19 b) son las que muestran mayor calidad y en consecuencia menor imagen de error, que de hecho es cero en la parte central de la imagen. Las divergencias que aparecen se deben a la crominancia, cuya calidad no llega a la de la luminancia por los motivos ya comentados de divergencia entre los vectores de movimiento de la luminancia usados y los reales de la crominancia. Las imágenes de tipo 'c.1' (FIGURA 5.19 d) y 'c.2' (FIGURA 5.19 c) muestran un predominio de errores en sentido vertical y horizontal respectivamente. Este hecho también se constata en sus transformadas de Fourier bidimensionales, tanto en módulo (FIGURA 5.20) como en fase (FIGURA 5.21), donde las componentes espectrales horizontales son casi nulas en módulo y fase para la imagen tipo 'c.2' (FIGURA 5.20 c.2 y FIGURA 5.21 c.2) y casi nulas en el sentido vertical (FIGURA 5.20 d.2 y FIGURA 5.21 d.2) para la imagen tipo 'c.1'.

Para la imagen tipo 'a', se puede constatar el bajísimo error exhibido en el dominio temporal y en una muy amplia zona de bajas y medias frecuencias en el dominio de la frecuencia. Este bajo error en frecuencias altas es un claro indicativo de recuperación de la información en zonas abruptas como pueden ser los bordes de los objetos presentes en las imágenes o de los pequeños detalles del fondo.

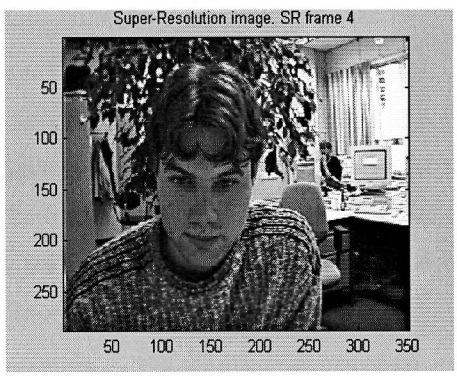
Aunque en general las crominancias suelen tener una mayor PSNR que las luminancias debido a su menor entropía, como efectivamente ocurre cuando se tienen en cuenta los bordes en los cálculos (FIGURA 5.22.a), no pasa lo mismo cuando se eliminan los bordes de las imágenes (FIGURA 5.22.b). Debido a que el proceso está mejor ajustado para la luminancia que para las crominancias, cuando se dispone de todas las muestras de entrada (imágenes tipo 'a') la PSNR de la luminancia sin bordes es muy superior a la de las crominancias.



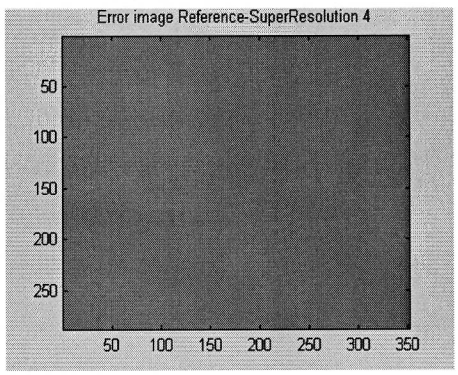
(a.1)



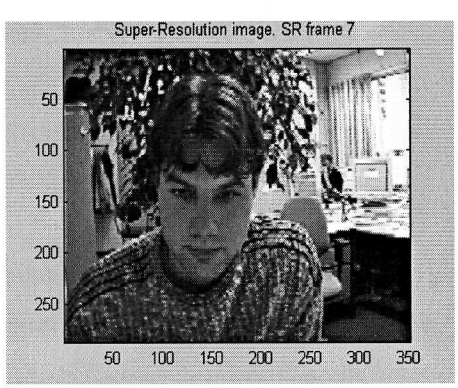
(a.2)



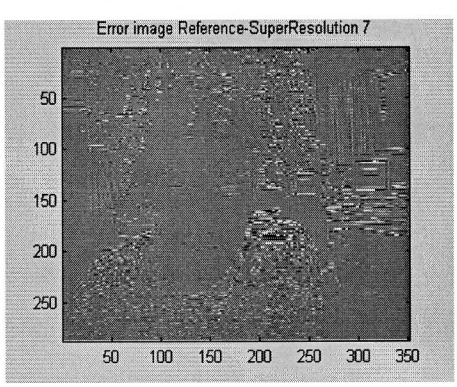
(b.1)



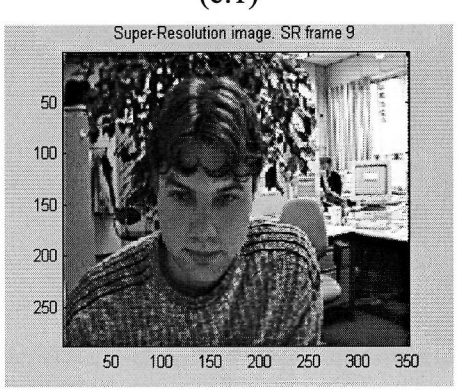
(b.2)



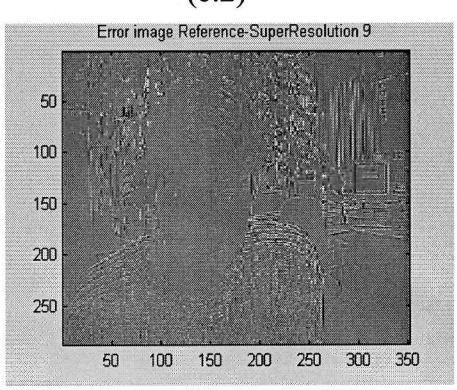
(c.1)



(c.2)



(d.1)



(d.2)

FIGURA 5.19. Imágenes de súper-resolución (1) y sus errores asociados (2), del *frame* 0 (a), del *frame* 4 (b), del *frame* 7 (c) y del *frame* 9 (d).

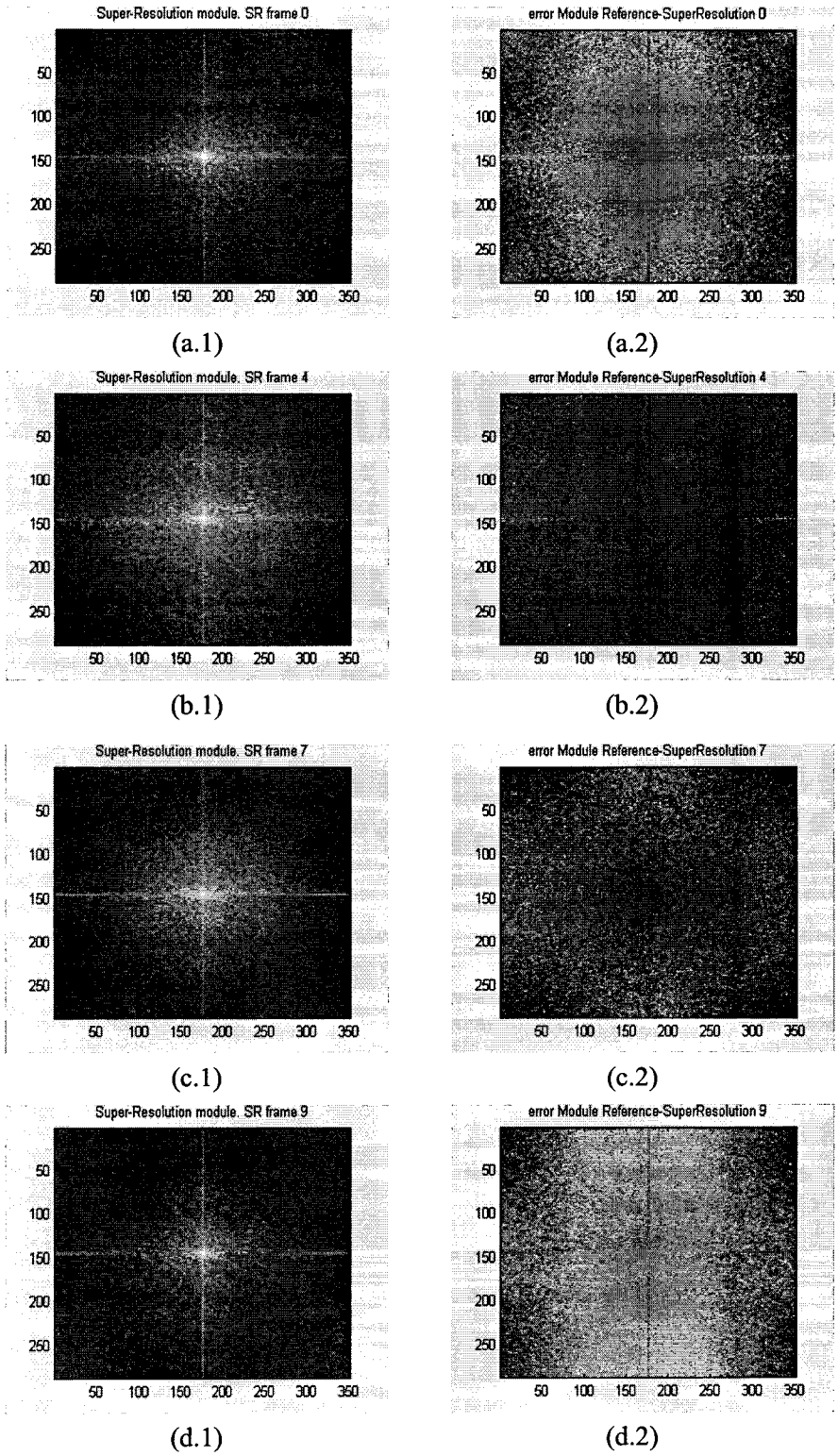


FIGURA 5.20. Transformadas de Fourier bidimensionales en módulo de las imágenes de súper-resolución en (1) y sus errores asociados (2), del *frame* 0 (a), del *frame* 4 (b) , del *frame* 7 (c) y del *frame* 9 (d).

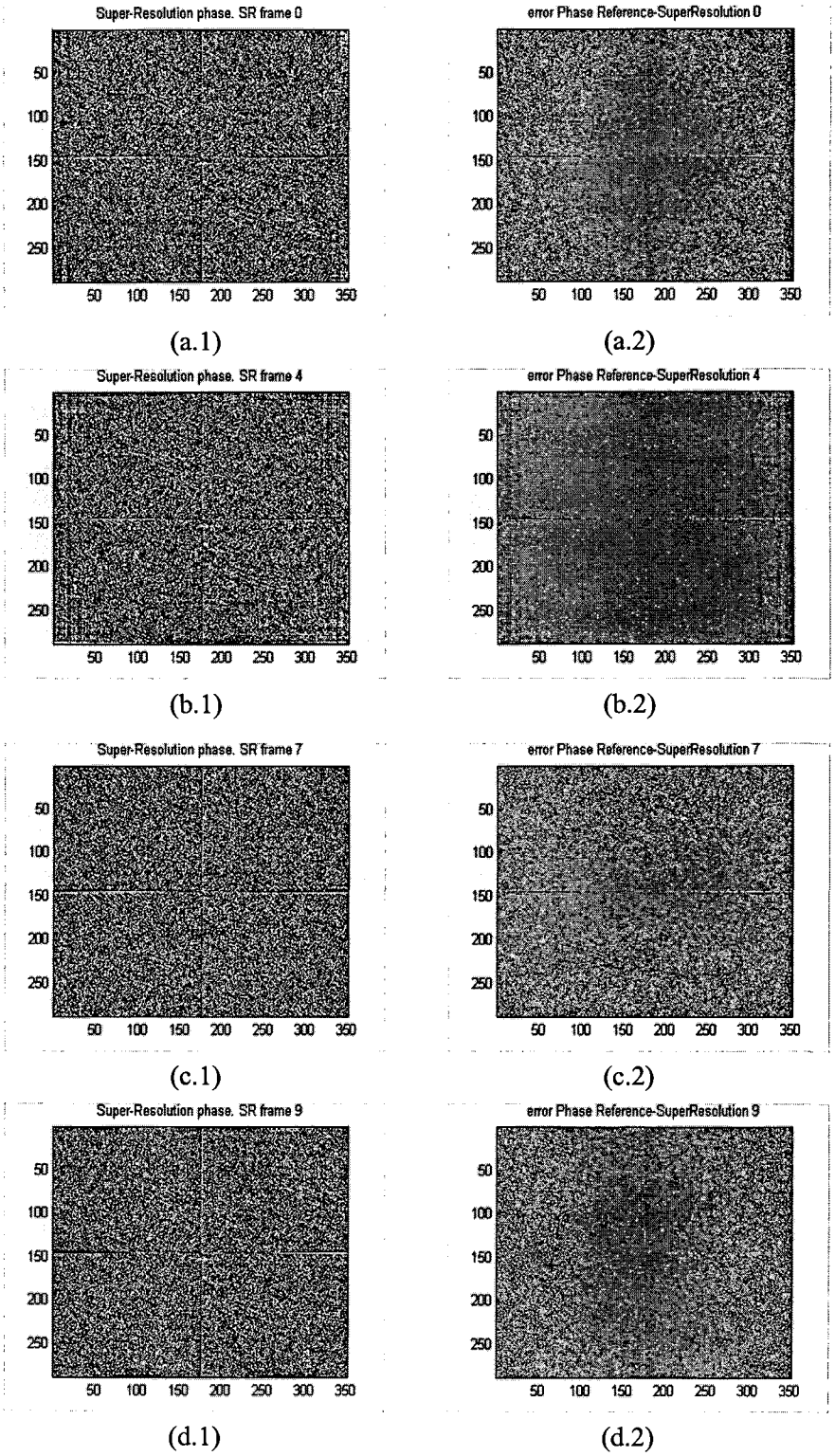


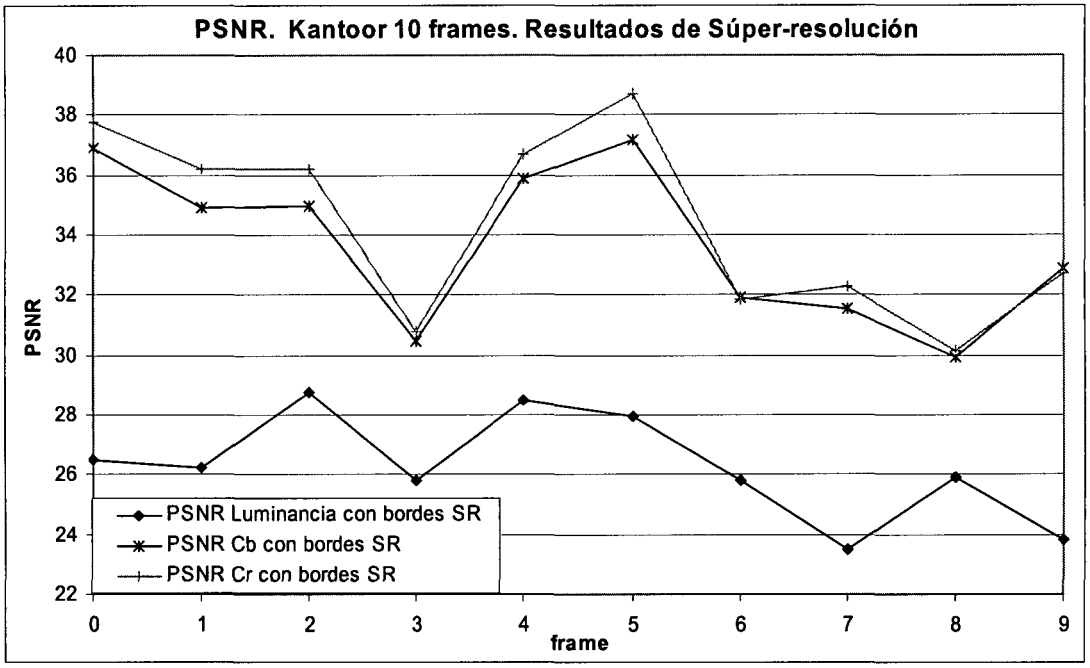
FIGURA 5.21. Transformadas de Fourier bidimensionales en fase de las imágenes de súper-resolución en (1) y sus errores asociados (2), del *frame* 0 (a), del *frame* 4 (b) , del *frame* 7 (c) y del *frame* 9 (d).

Sin embargo, resulta muy interesante realizar una comparativa entre los valores de PSNR de las crominancias (por ejemplo la crominancia roja que exhibe una mayor PSNR media) obtenidos por esta versión del algoritmo y las versiones iterativas. Si realizamos un representación de las PSNR de las imágenes completas (FIGURA 5.23 a) no se observa gran diferencia de calidad entre ellas, pero si eliminamos los bordes del cálculo de la PSNR (FIGURA 5.23 b), entonces podemos apreciar una mejoría apreciable en las crominancias obtenidas con esta nueva versión del ASR, observándose una mejora promedio de 1.25 decibelios frente a la interpolación bilineal, de 1.82 decibelios frente a la versión v1.2 y de 1.09 decibelios frente a la versión v1.3.

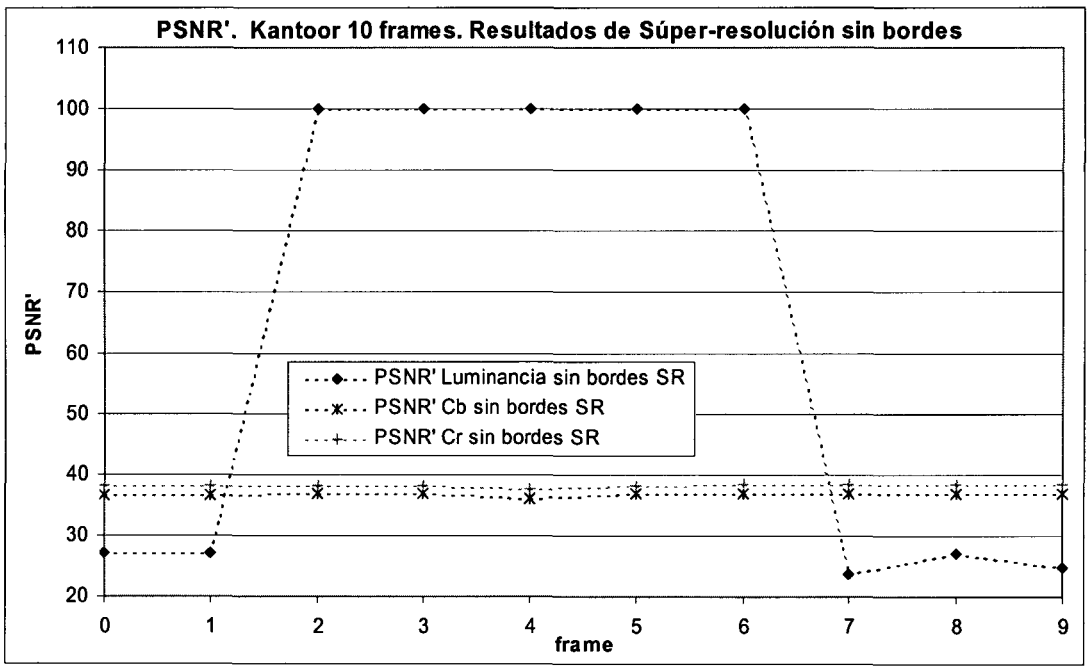
Las correlaciones espectrales en módulo y fase, con y sin bordes (FIGURA 5.24) muestran el comportamiento esperado, coherente con las variaciones de la PSNR. Cabe destacar la alta correlación espectral en fase obtenida, muy superior a la alcanzada por los algoritmos iterativos, incluso en el caso de incluir los bordes de las imágenes en el cálculo. Tan solo en el caso del *frame* 9 se produce un descenso de la correlación espectral por debajo del nivel de interpolación, que se recupera al eliminar los bordes.

Es también interesante destacar una mayor independencia de los bordes de las imágenes en el nuevo algoritmo. Esto se constata en una mayor similitud de las curvas con y sin bordes. El motivo es que este algoritmo intenta recuperar información de los bordes de la imagen, interpolando los datos ausentes, pero nunca replicándolos, gracias a las modificaciones introducidas en el compensador de movimiento.

Como era de esperar, la eliminación de los bordes de las imágenes produce unas correlaciones espectrales unidad, que evidencian la total igualdad entre las imágenes tipo 'a' sin bordes obtenidas y las imágenes de referencia.

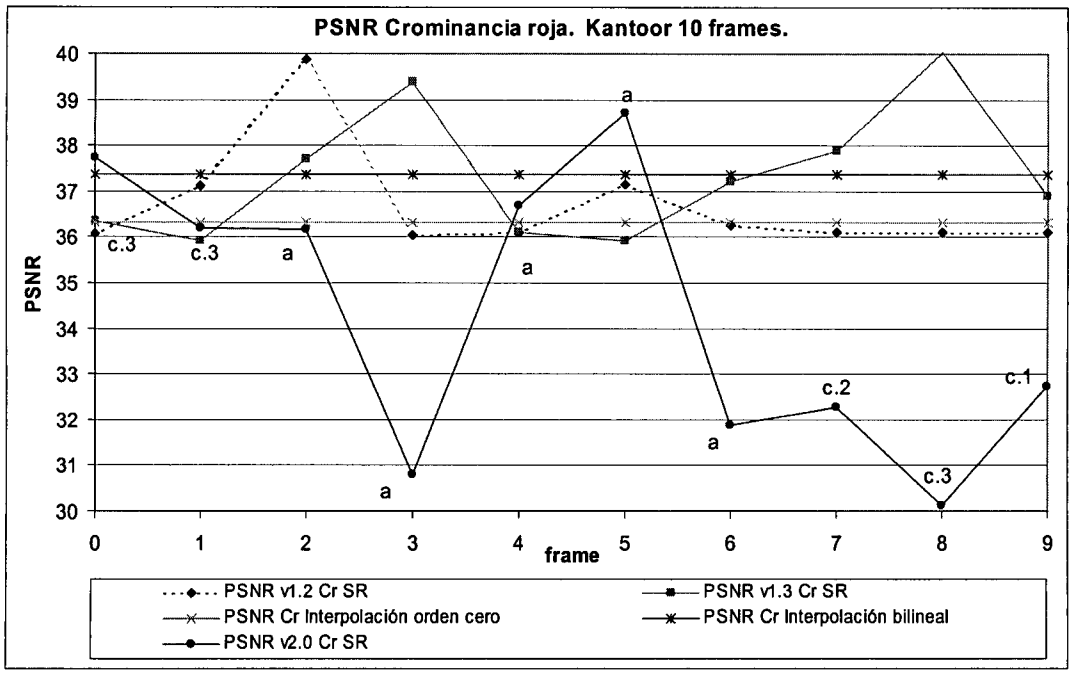


(a)

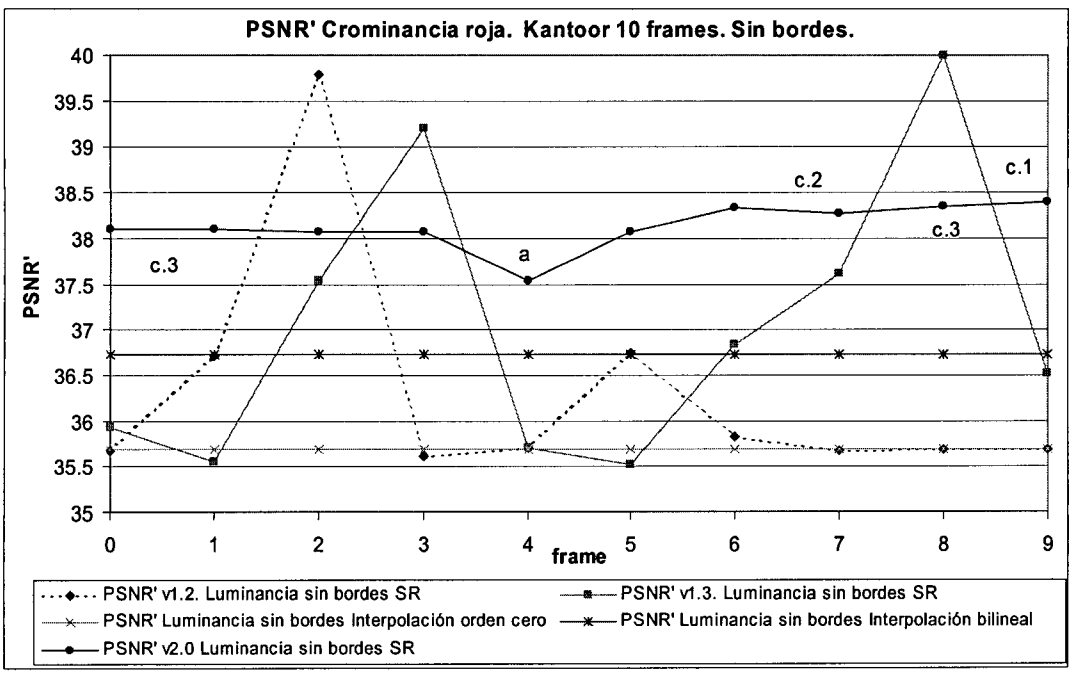


(b)

FIGURA 5.22. PSNR' de la versión v2.0 de la luminancia y de las crominancias usando las imágenes completas (a) y sin bordes (b).



(a)



(b)

FIGURA 5.23. PSNR' compara de las versiones v1.2, v1.3 y v2.0 y de las imágenes interpoladas de la crominancia roja usando las imágenes completas (a) y sin bordes (b).

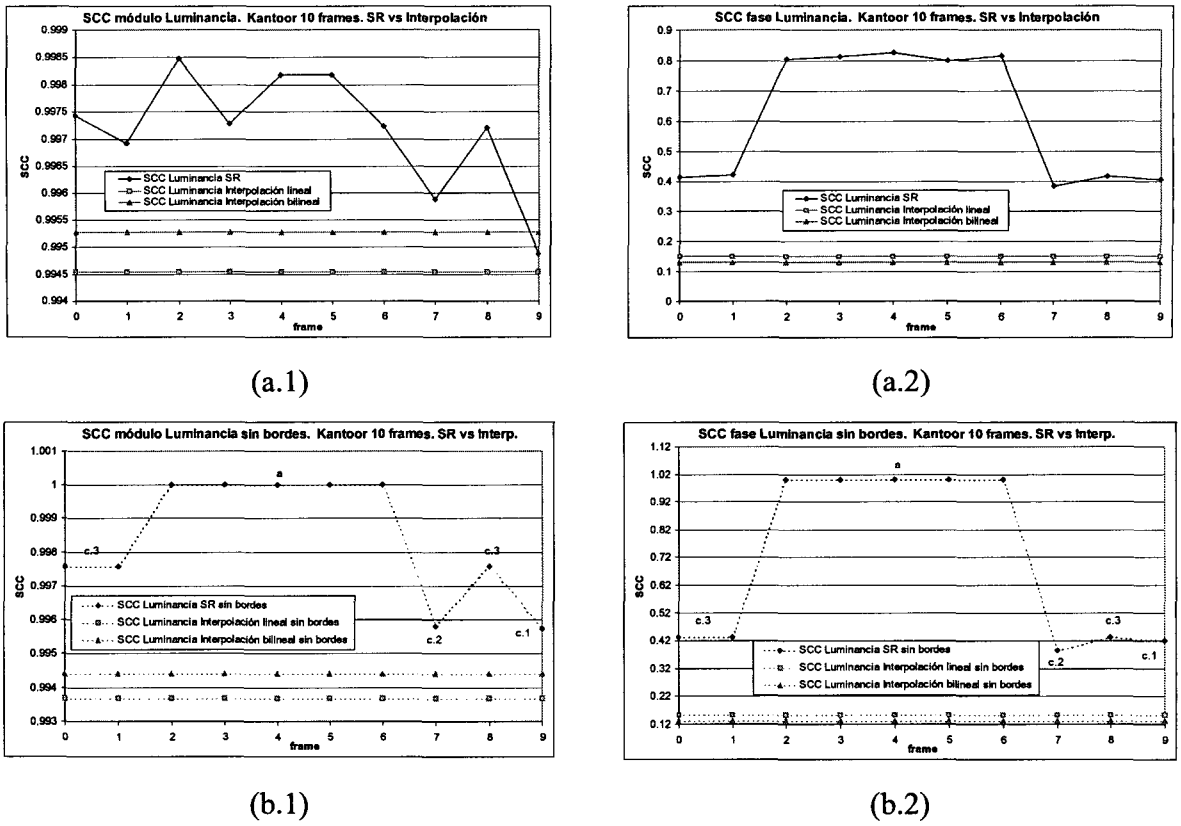


FIGURA 5.24. Correlaciones espectrales en módulo (1) y en fase (2) de las imágenes de luminancia completas (a) y sin bordes (b).

5.2.3.3 Resultados de simulación y análisis de calidad y comportamiento de la versión v2.0 usando precisión de ¼ de píxel

Todos los resultados mostrados hasta ahora han sido obtenidos usando una estimación de movimiento en baja resolución de ½ píxel. Sin embargo, es claramente deseable aumentar la precisión del estimador de movimiento a ¼ de píxel, y en ese sentido se ha modificado el '*motion estimator*'. También se ha alterado la forma de obtener las secuencias de prueba, como se ha comentado en el apartado 4.2.2 de esta tesis doctoral. Por ello se ha querido probar este mismo algoritmo usando una mayor precisión. En consecuencia el proceso de generación de imágenes de prueba ha sido alterado como ya se comentó, optándose además por cambiar a la secuencia de prueba etiquetada como 'KRANT[1]' de tamaño VGA, que tiene

[1] Krant significa 'periódico' en holandés.

una marcada distribución espacial de los objetos que en ella aparecen, y que da lugar a un espectro con componentes orientadas, mucho menos homogéneo que en el caso de 'KANTOOR[2]'.

Al cambiar la precisión de la estimación de movimiento, también cambia la celda base, debiendo usarse en este caso la celda base mostrada en la FIGURA 4.40. Sin embargo, mientras que en el caso de la estimación de movimiento si es posible realizar una clasificación abarcable de las imágenes de súper-resolución en función de las posiciones de las muestras disponibles, en este caso el número total de posibilidades es demasiado alto como, para establecer una clasificación. En concreto, el número total de combinaciones, recogiendo las muestras de cuatro en cuatro, y siendo N el número total de píxeles en la celda base, es el expresado en (5.2), que para N=4 (celda base de ½ píxel) arroja una cifra de 15 (estas son las clasificaciones realizadas) y para N=16 (celda base de ¼ de píxel), arroja una cifra mucho mayor de 2516.

$$\sum_{k=1}^{4} \frac{N!}{k! \cdot (N-k)!} \tag{5.2}$$

Dada la enorme cantidad de posibilidades de muestreo, desistimos de hacer una clasificación exhaustiva, pero sí hay que destacar que sigue rigiendo el principio general de que las muestras seleccionadas por los desplazamientos afectarán de forma crítica a la calidad de las imágenes resultantes de súper-resolución, alcanzándose el máximo valor de calidad cuando los desplazamientos incluyan las cuatro posiciones enteras de píxel, que corresponde a los vectores de entrada (0,0), (0,2), (2,0) y (2,2), expresados en unidades de píxel de muy alta resolución (VHR, *Very High Resolution*). En la FIGURA 5.25 pueden verse las relaciones entre los cambios de escala y unidades de los desplazamientos originales a través del proceso de generación de imágenes de prueba y súper-resolución. En la parte inferior de la imagen se ha puesto el ejemplo del desplazamiento (2,2) en unidades de píxel de muy alta resolución, nombre que le damos a la imagen en un formato original grande, como puede ser VGA o AVGA para luego recortarla y evitar el indeseable efecto de borde. Partiendo de esta imagen recortada y desplazada, se diezma y filtra en un factor ½ para obtener la imagen de alta resolución (HR, *High Resolution*) que servirá de referencia en la obtención de las medidas de calidad. Al mismo tiempo, la imagen se sub-muestrea en un factor ¼ para generar *aliasing*, lo que constituirá la simulación de la entrada de baja resolución (LR, *Low Resolution*) al ASR. Una vez dentro del sistema desarrollado, las unidades usadas son de ¼ de píxel, que se pueden

[2] *Kantoor* significa 'oficina' en holandés.

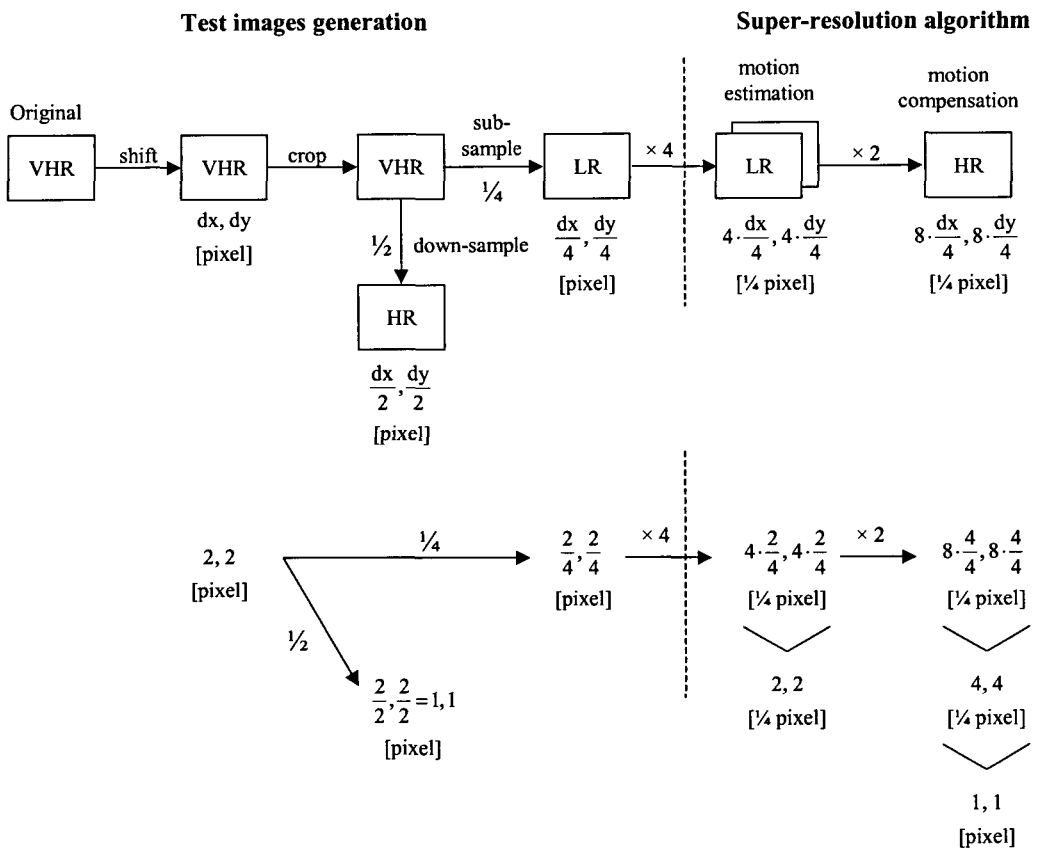


FIGURA 5.25. Relaciones de unidades y cambios de escala sufridos por los desplazamientos durante el proceso de generación de imágenes de prueba y súper-resolución.

obtener multiplicando por cuatro los desplazamientos en unidades de píxel. Al pasar de baja a alta resolución, tanto las imágenes como los vectores de movimiento son multiplicados por dos, y se puede comprobar cómo el desplazamiento generado en muy ata resolución de (2,2) se corresponde con un desplazamiento en la imagen de alta resolución de (1,1) píxel. Aunque la imagen de muy alta resolución se desplaza realmente (2,2) píxeles, es obvio que la relación entre las imágenes de VHR y las de HR es de un factor de escala de dos. Por lo tanto, podemos ver esos desplazamientos en unidades de ½ píxel de alta resolución. Así, dos medios píxeles equivaldrían a un píxel, y se puede ver con antelación que el desplazamiento (2,2) en muy alta resolución equivale al (1,1) en alta resolución.

Para la secuencia de prueba KRANT se selecciona el primer *frame*, y se le aplicaron los vectores de movimiento (expresados en píxeles de muy alta resolución) recogidos en la TABLA 5.3, generando 17 *frames* de salida. Todos los vectores son generados de forma aleatoria, excepto el primer conjunto de cuatro, que se genera de forma artificial en las posiciones enteras de píxel en alta resolución para ver la máxima calidad que puede alcanzar el algoritmo.

frame	Vectores de muy alta resolución		Vectores de alta resolución		Vectores de baja resolución	
0	0	0	0	0	0	0
	2	0	1	0	0.5	0
	0	2	0	1	0	0.5
	2	2	1	1	0.5	0.5
1	0	0	0	0	0	0
	-1	-3	-0.5	-1.5	-0.25	-0.75
	-1	-3	-0.5	-1.5	-0.25	-0.75
	-1	-2	-0.5	-1	-0.25	-0.5
2	0	0	0	0	0	0
	3	2	1.5	1	0.75	0.5
	3	1	1.5	0.5	0.75	0.25
	2	0	1	0	0.5	0
3	0	0	0	0	0	0
	3	-3	1.5	-1.5	0.75	-0.75
	2	-1	1	-0.5	0.5	-0.25
	1	0	0.5	0	0.25	0
4	0	0	0	0	0	0
	3	2	1.5	1	0.75	0.5
	1	3	0.5	1.5	0.25	0.75
	2	2	1	1	0.5	0.5
5	0	0	0	0	0	0
	-2	2	-1	1	-0.5	0.5
	-2	3	-1	1.5	-0.5	0.75
	-1	1	-0.5	0.5	-0.25	0.25
6	0	0	0	0	0	0
	-2	-3	-1	-1.5	-0.5	-0.75
	0	-1	0	-0.5	0	-0.25
	-1	-2	-0.5	-1	-0.25	-0.5
7	0	0	0	0	0	0
	2	2	1	1	0.5	0.5
	0	1	0	0.5	0	0.25
	1	2	0.5	1	0.25	0.5
8	0	0	0	0	0	0
	0	3	0	1.5	0	0.75
	-3	1	-1.5	0.5	-0.75	0.25
	0	1	0	0.5	0	0.25
9	0	0	0	0	0	0
	3	-1	1.5	-0.5	0.75	-0.25
	2	-3	1	-1.5	0.5	-0.75
	0	-1	0	-0.5	0	-0.25
10	0	0	0	0	0	0
	2	-3	1	-1.5	0.5	-0.75
	2	-1	1	-0.5	0.5	-0.25
	2	-2	1	-1	0.5	-0.5
11	0	0	0	0	0	0
	-1	2	-0.5	1	-0.25	0.5
	0	2	0	1	0	0.5
	0	1	0	0.5	0	0.25
12	0	0	0	0	0	0
	1	2	0.5	1	0.25	0.5
	3	3	1.5	1.5	0.75	0.75
	2	1	1	0.5	0.5	0.25
13	0	0	0	0	0	0
	0	1	0	0.5	0	0.25
	0	0	0	0	0	0
	-3	0	-1.5	0	-0.75	0
14	0	0	0	0	0	0
	-2	-3	-1	-1.5	-0.5	-0.75
	0	-1	0	-0.5	0	-0.25
	-3	-1	-1.5	-0.5	-0.75	-0.25
15	0	0	0	0	0	0
	-1	0	-0.5	0	-0.25	0
	-1	-1	-0.5	-0.5	-0.25	-0.25
	-1	-1	-0.5	-0.5	-0.25	-0.25
16	0	0	0	0	0	0
	-2	-3	-1	-1.5	-0.5	-0.75
	-2	-2	-1	-1	-0.5	-0.5
	0	0	0	0	0	0

TABLA 5.3. Vectores de movimiento aleatoriamente generados expresados en píxeles de muy alta resolución, en píxeles de alta resolución y su reducción a vectores canónicos en la celda base de ¼ de píxel.

En la FIGURA 5.26 se muestra el *frame* de referencia (a.1) junto a su transformada de Fourier bidimensional en módulo (a.2) y la secuencia de entrada de baja resolución (b.1-e.1) junto con sus correspondientes transformadas de Fourier bidimensionales en módulo (b.2-e.2).

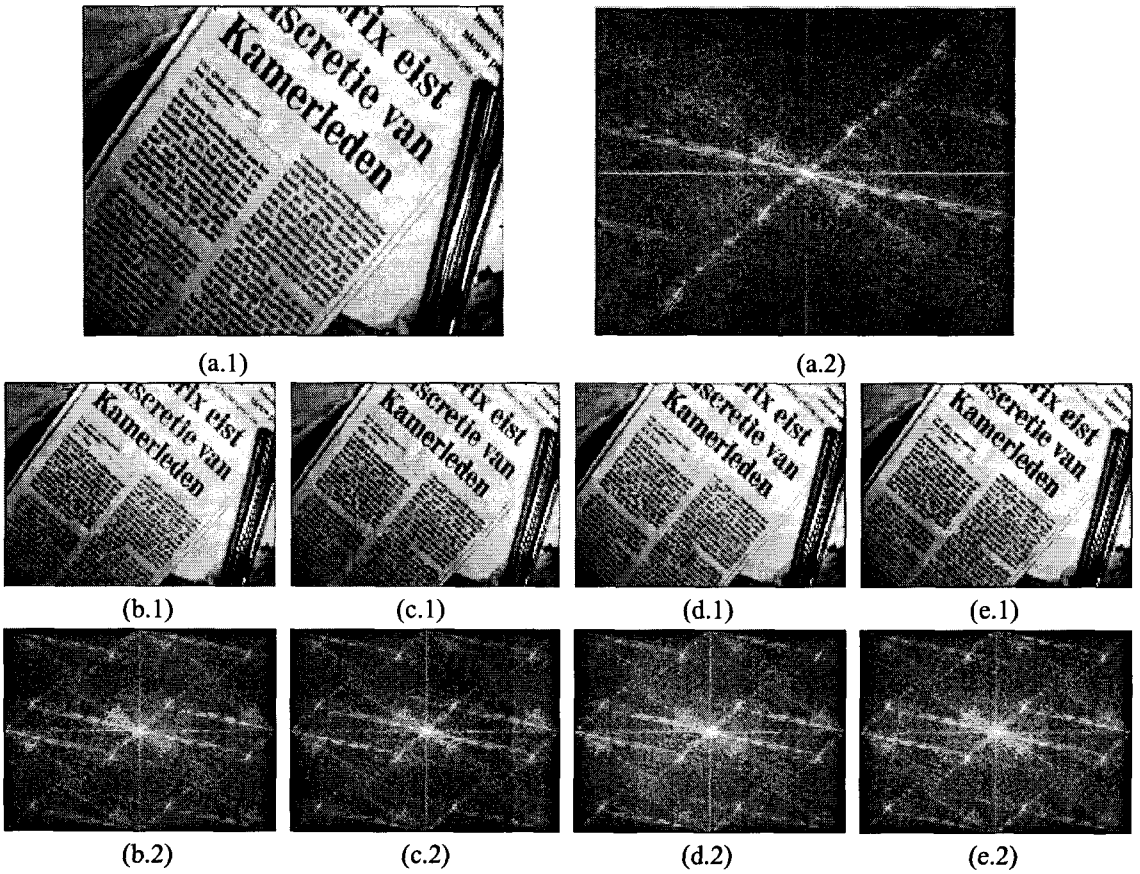
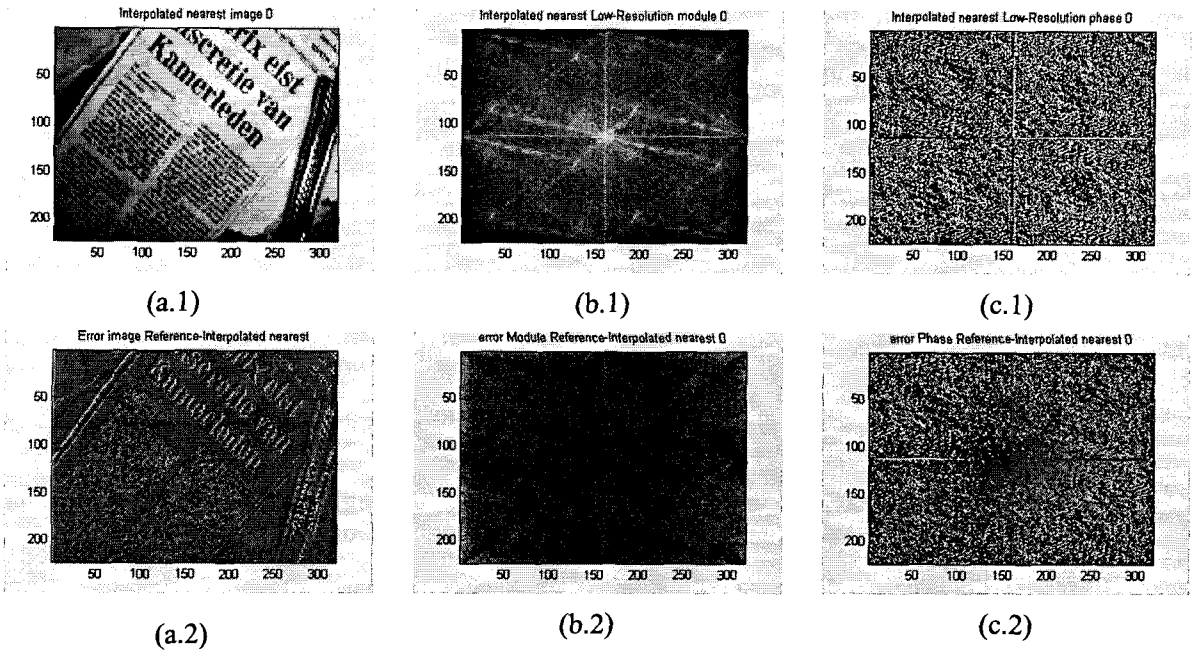


FIGURA 5.26. *Frame* cero de la secuencia de prueba KRANT (a.1) junto con su transformada de Fourier bidimensional en módulo (a.2) y la secuencia de entrada de baja resolución (b.1-e.1) junto con sus correspondientes transformadas de Fourier bidimensionales en módulo (b.2-e.2).

Dado que la imagen tiene una clara orientación de los objetos que aparecen, su transformada de Fourier bidimensional en módulo tiene componentes espectrales de alta energía muy direccionales. Esta característica permite apreciar con claridad la gran cantidad de *aliasing* presente en la secuencia de entrada. Dado que el *aliasing* está presente dentro de la banda de paso de la señal, en principio no se puede separar usando técnicas de filtrado convencionales.

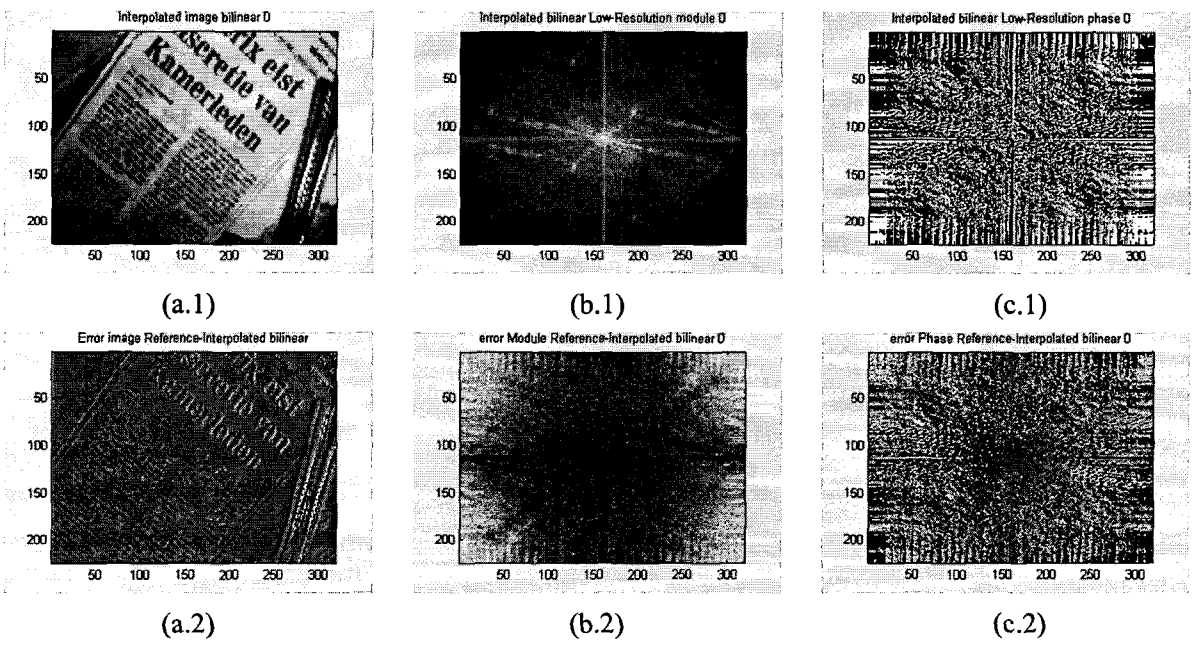
También para referencia se muestran las imágenes interpoladas, en tiempo (a), módulo (b) y fase (c) , tanto para la interpolación lineal de orden cero FIGURA 5.28, como para la bilineal FIGURA 5.27, junto con sus errores asociados (2).



(a.1) (b.1) (c.1)

(a.2) (b.2) (c.2)

FIGURA 5.27. Imagen interpolada usando interpolación lineal de orden cero, en el dominio del tiempo (a), de la frecuencia en módulo (b) y de la frecuencia en fase (c), así como sus errores asociados (2).



(a.1) (b.1) (c.1)

(a.2) (b.2) (c.2)

FIGURA 5.28. Imagen interpolada usando interpolación bilineal, en el dominio del tiempo (a), de la frecuencia en módulo (b) y de la frecuencia en fase (c), así como sus errores asociados (2).

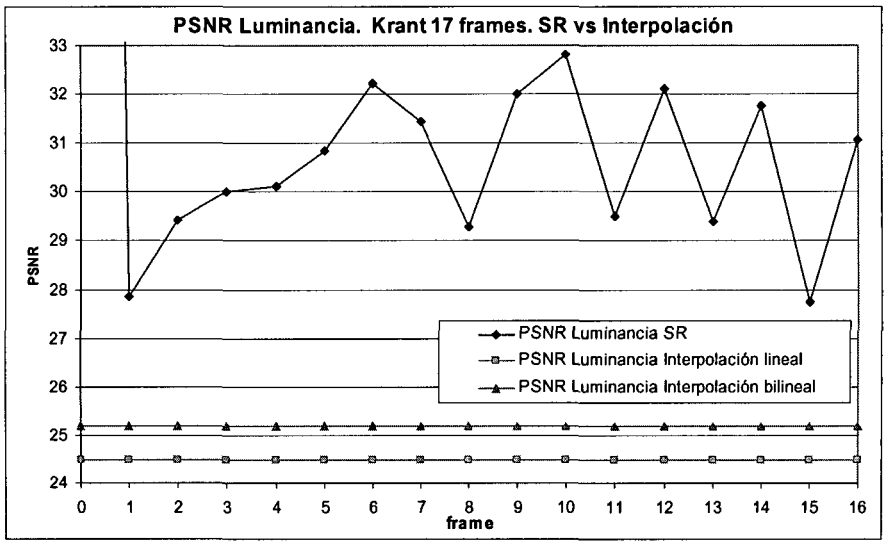


FIGURA 5.29. PSNR de la luminancia de la secuencia KRANT y de las imágenes interpoladas.

Tras la ejecución del algoritmo sobre 17 *frames* de la imagen mostrada en la FIGURA 5.26 (a.1), sub-muestreada en un factor de ¼ tras haber sido desplazada las cantidades y direcciones dadas por los vectores de desplazamiento de la TABLA 5.3, se han obtenido las relaciones señal-ruido de pico de la FIGURA 5.29. El *frame* cero presenta una PSNR máxima de 99.99, tal y como era previsible al contar con todas sus muestras de entrada en posiciones enteras de píxel. La PSNR media de la luminancia (sin contar el *frame* 0 de salida) es de 30.47 dB, que sobrepasa en 7.28 dB al promedio obtenido para la versión v1.2 y en 2.23 dB al promedio obtenido para la versión v1.3.

Si mostramos en una misma gráfica las PSNR de la secuencia con y sin bordes (FIGURA 5.30) vemos que la calidad es muy similar (el error medio es de 0.35 dB). Debido a que la única diferencia con los experimentos del apartado anterior reside en la diferencia de precisión del estimador de movimiento, podemos afirmar que este aumento de precisión en la estimación del movimiento insensibiliza de forma importante esta versión del algoritmo de los indeseados efectos de borde.

En lugar de mostrar todos los *frames*, prestaremos especial atención a las esquinas del proceso, es decir, al mejor y al peor caso. El *frame* con mayor PSNR es el *frame* 10 y el peor es el *frame* 15. En la FIGURA 5.31se muestra el *frame* 10 en el dominio del tiempo (a), de la frecuencia en módulo (b) y de la frecuencia en fase (c), así como sus errores asociados (2).

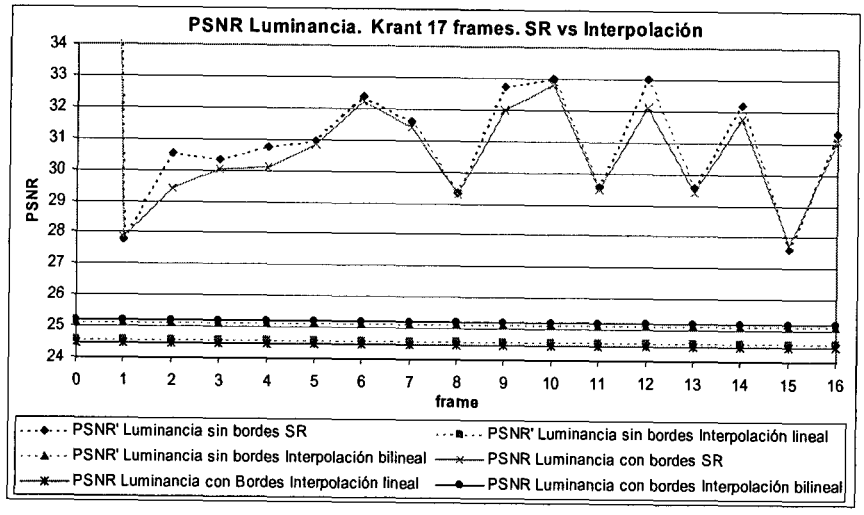


FIGURA 5.30. PSNR de la luminancia de la secuencia KRANT con y sin bordes tras aplicar la versión v2.0 con precisión de ¼ de píxel.

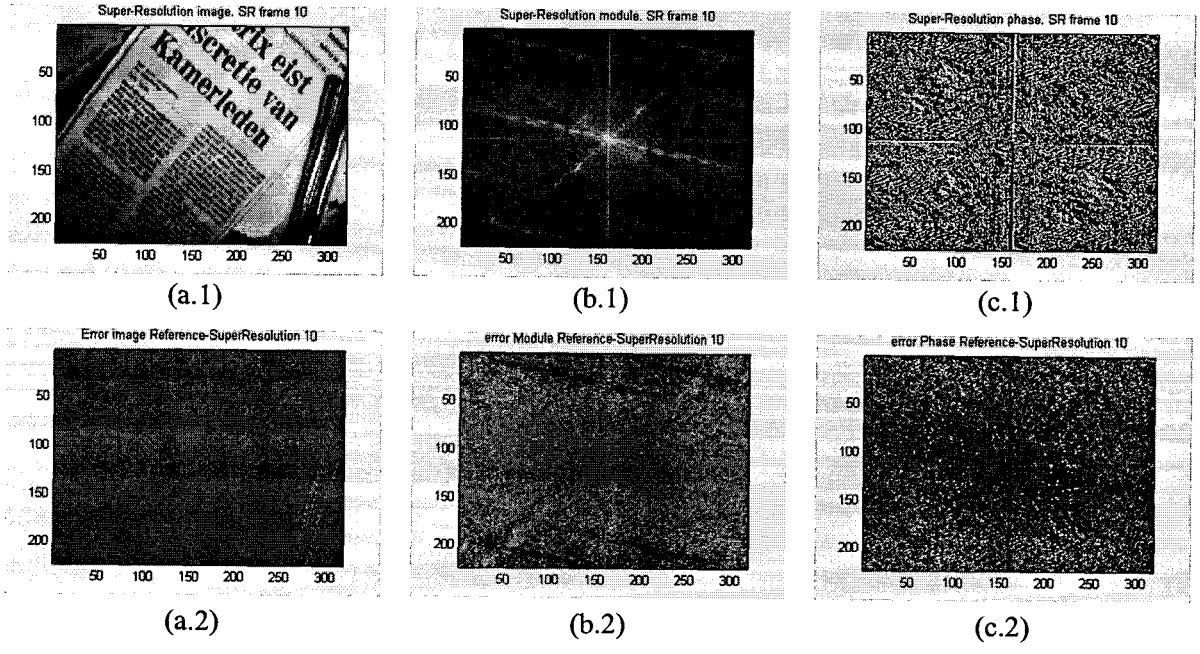


FIGURA 5.31. *Frame* 10 de súper-resolución en el dominio del espacio (a), de la frecuencia espacial en módulo (b) y de la frecuencia espacial en fase (c), así como sus errores asociados (2).

Si representamos conjuntamente a la las transformadas de Fourier bidimensionales en módulo mostradas en las FIGURA 5.26 (a.2), en la FIGURA 5.26 (b.2) y en la FIGURA 5.31(b.1) en la FIGURA 5.32, podemos apreciar claramente que se ha disminuido de forma considerable la cantidad de *aliasing* presente en el espectro de la imagen de súper-resolución (c) frente a la contenida en las imágenes de entrada (b), comparadas ambas con el espectro de la señal original (a). Sin embargo, todavía se aprecian restos de *aliasing*, especialmente de las bandas

más energéticas diagonales de la señal original en la zona de las altas frecuencias, tal y como se señala en la figura.

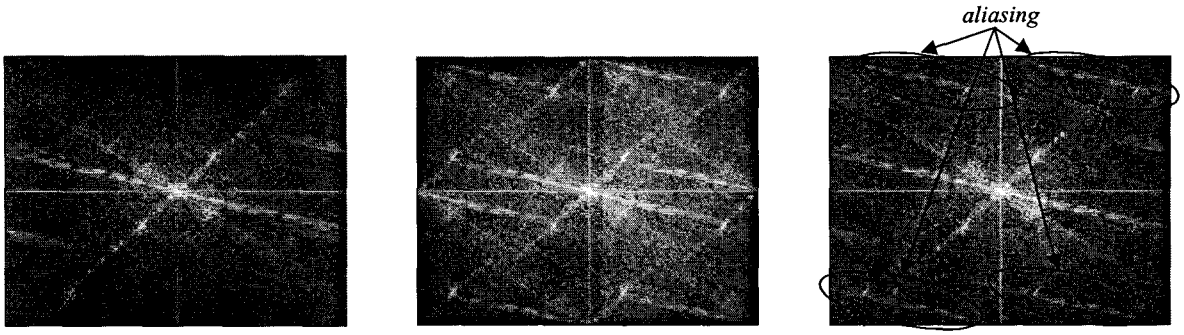


FIGURA 5.32. Transformadas de Fourier bidimensionales en módulo de la luminancia de la imagen original (a), de la primera imagen de entrada (b) y de la imagen de salida número 10 de la secuencia de súper-resolución (c).

El *aliasing* también es apreciable en el dominio del espacio como una pérdida de calidad motivada por la ausencia de datos. Esto se ve con claridad en los pequeños detalles, como las letras, y en la pérdida de continuidad de los bordes, como los de los bolígrafos de la derecha. El proceso de súper-resolución permite incorporar nueva información proveniente de otras imágenes cercanas, aumentando considerablemente la calidad de la imagen resultante. Por ejemplo, en la FIGURA 5.33 se muestra de forma ampliada la esquina superior derecha de la primera imagen de entrada de la secuencia KRANT (a) y la misma zona mejorada con el proceso de súper-resolución (b). Nótese que antes de aplicar el algoritmo era prácticamente imposible leer la palabra *'nieuw'* (nuevo en holandés) que aparece en la esquina superior derecha. Los bordes de los bolígrafos y de las letras de los titulares del periódico también han sido claramente mejorados, haciéndose patente el aumento de la resolución como una

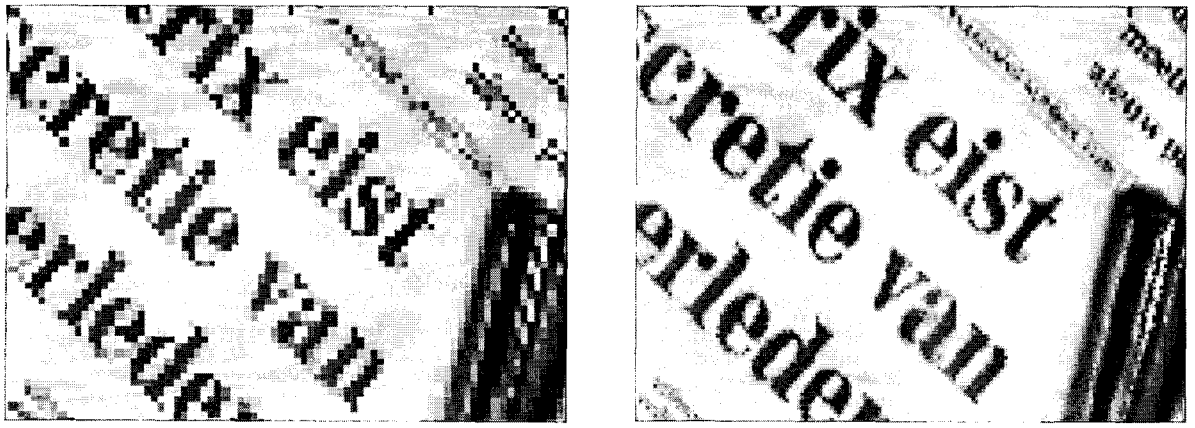


FIGURA 5.33. Detalle del primer *frame* de la secuencia KRANT, antes (a) y después de aplicar el algoritmo de súper-resolución v2.0 (b).

disminución del tamaño medio aparente de los píxeles.

Si observamos la imagen de peor calidad, la que ofrece el *frame* 15 (FIGURA 5.34), vemos que la cantidad de *aliasing* presente es mayor, aunque se haya eliminado cierta cantidad frente a la contenida en las imágenes de entrada.

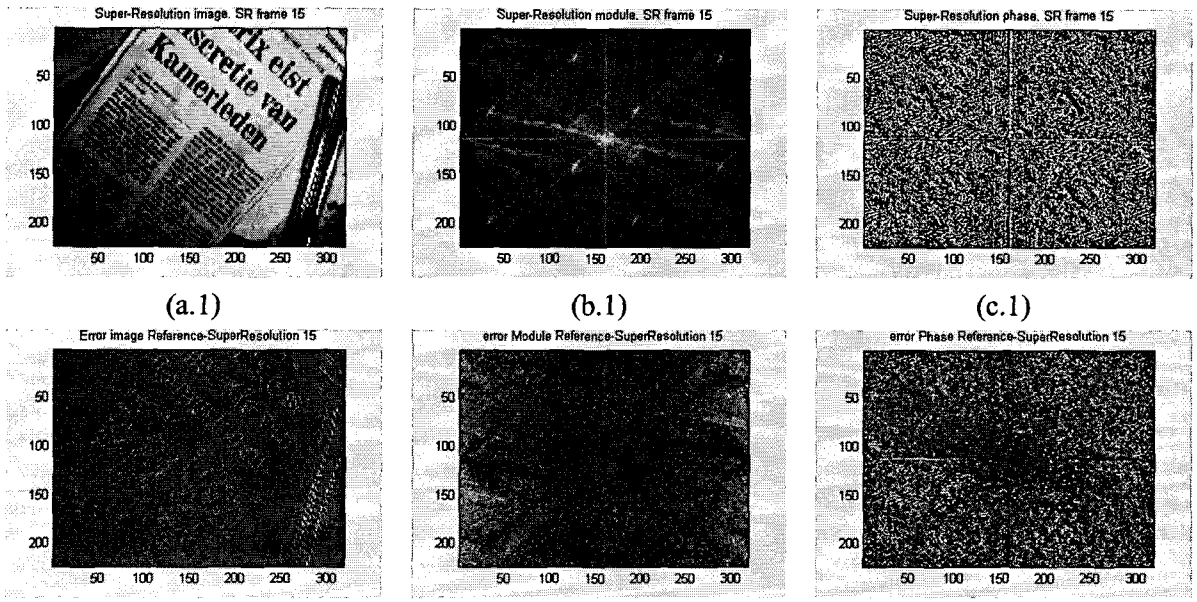
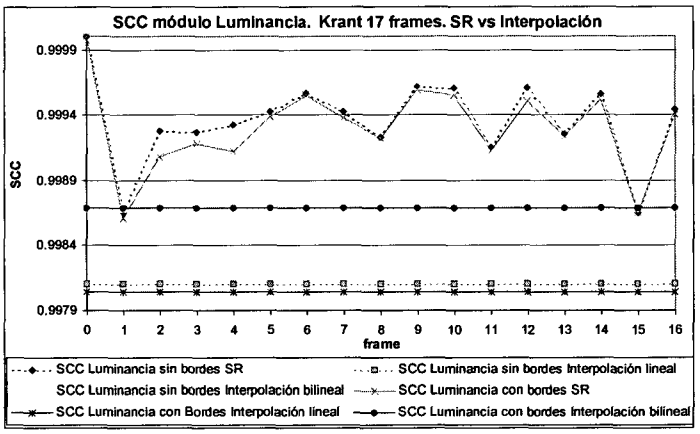
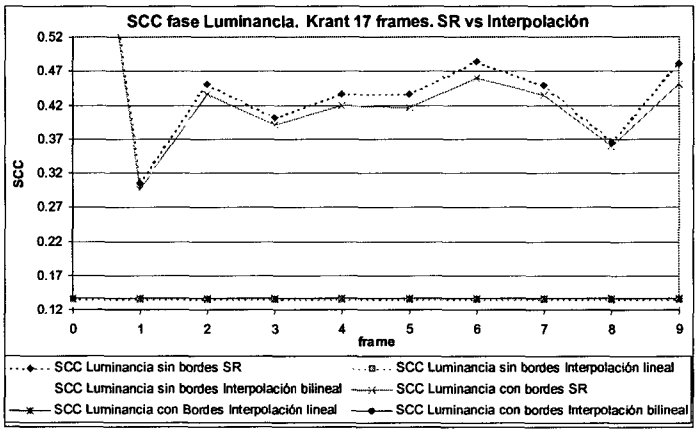


FIGURA 5.34. *Frame* 15 de súper-resolución en el dominio del espacio (a), de la frecuencia en módulo (b) y de la frecuencia en fase (c), así como sus errores asociados (2).

Las correlaciones espectrales también demuestran una menor diferencia entre las imágenes con bordes y sin bordes (FIGURA 5.35). El error absoluto medio en módulo entre ambas es de 5.8648E-05 y en fase es de 0.01568. Sin embargo, aunque el error medio del módulo sea menor, su distancia a los niveles de interpolación es menor que en el caso de la fase, llegando a ponerse al nivel de interpolación en los *frames* 1 y 15, cosa que nunca llega a suceder en fase. La mayor correlación y menor error entre imágenes con y sin borde en módulo indica un mejor funcionamiento del algoritmo en módulo, aspecto muy positivo en tanto en cuanto el ojo humano es más sensible al módulo que a la fase de las señales. La mayor distancia de la correlación en fase del nivel de interpolación se debe, por un lado a la mejoría en la resolución del estimador de movimiento, que permite ajustar mejor los desplazamientos entre las imágenes de entrada, y por otro lado al desplazamiento de medio píxel que introduce la interpolación, lo que disminuye la correlación en fase de las imágenes interpoladas.



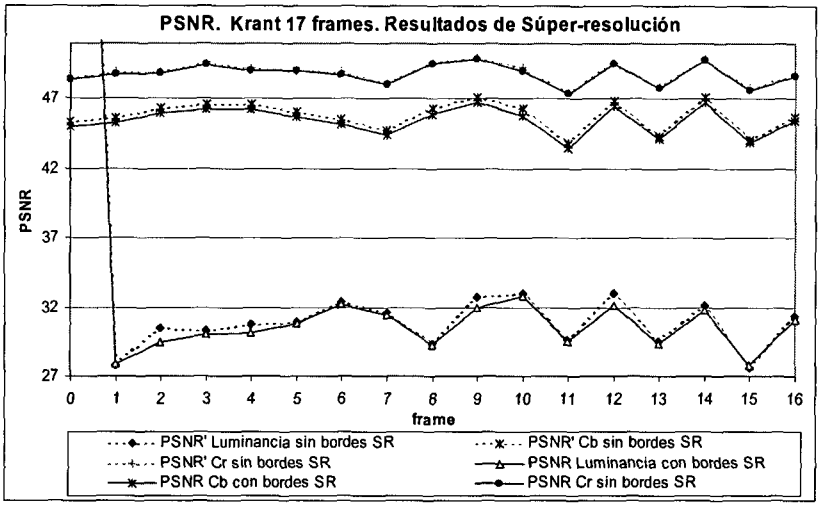
(a)



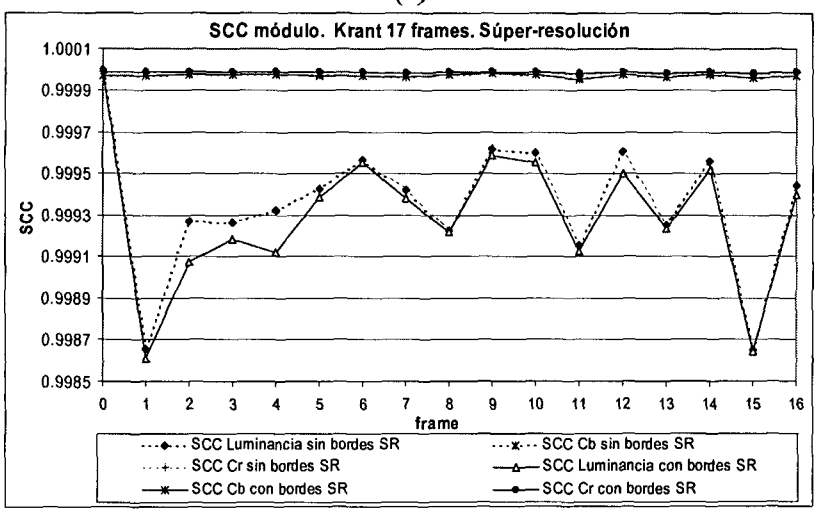
(b)

FIGURA 5.35. Coeficientes de correlación espectral en módulo (a) y fase (b) de la secuencia KRANT de 17 *frames*.

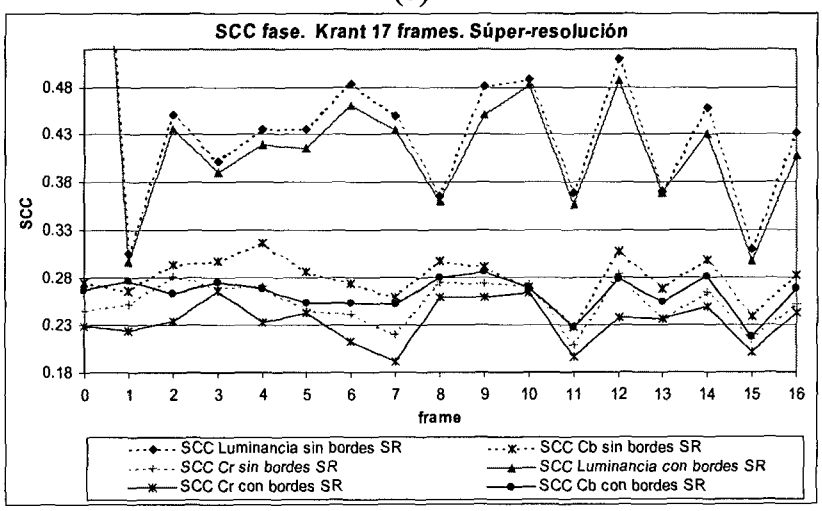
La PSNR y la correlación espectral en módulo de las crominancias es mayor que la de la luminancia (FIGURA 5.36), debido a la menor entropía de las primeras, pero sin embargo, la correlación espectral en fase de la luminancia es mayor que la de las crominancias. Esto es lógico si pensamos que los vectores de movimiento se han calculado sólo para la luminancia, por lo que es lógico que su fase se ajuste mejor a la original que la fase de las crominancias, para las que se han usado los vectores de movimiento de la luminancia. También puede apreciarse una gran similitud entre las curvas con y sin bordes, aunque las figuras de mérito sin bordes muestran un valor ligeramente superior a sus homólogas con bordes.



(a)



(b)



(c)

FIGURA 5.36. PSNR (a), coeficientes de correlación espectral en módulo(b) y fase (c) de la secuencia Krant con bordes (líneas continuas) y sin bordes (líneas discontinuas).

5.2.4 Algoritmo de súper-resolución no iterativo incremental (v2.1)

Un análisis detallado del pseudo-código de la versión v2.0 del algoritmo de súper-resolución de la FIGURA 5.13, refleja que en realidad no existe ningún impedimento para ampliar el número de imágenes de baja resolución a combinar. Basta con modificar dos aspectos:

1. Aumentar el número de imágenes de entrada en los bucles del algoritmo.
2. Aumentar el número de bits de la memoria acumulativa HR_A para poder albergar la suma de varias aportaciones provenientes de más imágenes.
3. Aumentar el número de bits de la memoria las contribuciones, aunque debido a su menor valor, es previsible que el ancho de esas memorias sea menor que en el caso de la memoria destinada a albergar imágenes.

Parece razonable suponer que si los desplazamientos a los que son sometidas las imágenes de entrada son aleatorios, mientras mayor sea el número de *frames* combinados mayor probabilidad habrá de tener una cantidad mayor de información nueva, lo que redundará en una mayor calidad de la imagen de súper-resolución. Esta suposición se ha verificado en las simulaciones de esta versión del algoritmo.

Para dimensionar la memoria HR_A se procede de la siguiente forma:

1. Se establece el número máximo de *frames* a acumular igual a 12. Por lo tanto, en el peor de los casos, tenemos que ser capaces de almacenar el número $(2^8-1)\cdot 12=3060$, ya que las imágenes de entrada tienen 8 bits.
2. Para almacenar el número 3060 se necesitan $\log_2(3060)=11.57\,\text{bits}$, que redondeamos por arriba a 12 bits.
3. Al efectuar el redondeo del número de bits, en realidad hemos aumentado el número de imágenes que pueden ser almacenadas en HR_A. El mayor número que ahora podemos almacenar con 12 bits es el $2^{12}-1=4095$.
4. En 4095 caben $\frac{4095}{255}=16.05$ imágenes, que redondeando por debajo determina que podemos almacenar, en el peor de los casos, 16 imágenes de entrada de 8 bits cada una.

```
* Non iterative super-resolution algorithm v2.1

  Set the value of the improvement: scale
  nr_frames = scale*scale, M'= scale*M, N'= scale*N
  HR_B, HR_S, HR_T, HR_T2, HR_Cont and HR_S2 all of 8 bits and size [M'][N'].
  HR_A is 12 bits and size [M'][N'].
  LR_I[M][N] for the motion estimation
  Read the first frame in LR_I_0[M][N] and the rest frames in LR_I[M][N]

// First Image is the reference
  nr_frames = cal_nr_frames();

  HR_A.lum    = Upsample_Holes(LR_I_0.lum)                    | [SR_INIT_A]
  HR_A.chrom = Upsample_Neighbours(LR_I_0.chrom)              |

  HR_Cont     = Create_image_contributions                    [SR_INIT_CONT]

  FOR fr = 1 .. nr_frames-1
      MV_ref2fr = Calc_Motion_Estimation ( LR_I, LR_I_0)
      IF Global_Motion THEN Select_global_motion_vector()
      MV_ref2fr = 2 .* MV_ref2fr

      HR_S.lum    = Upsample_Holes(LR_I.lum)                  | [SR_UPSAMPLE]
      HR_S.chrom = Upsample_ Neighbours(LR_I.chrom)           |

      HR_S2 = Create_image_contributions                      [SR_INIT_CONT]

      HR_T  = Motion_Compensation(HR_S, MV_ref2fr)            [SR_MOT_COMP]
      HR_T2 = Motion_Compensation(HR_S2, MV_ref2fr)           [SR_MOT_COMP_CONT]

      HR_A    = HR_A + HR_T                                   [SR_ADD]
      HR_Cont = HR_Cont + HR_T2                               [SR_ADD_CONT]
  END FOR

  HR_A = 4*HR_A /HR_Cont                                      [SR_ADJUST_A]

  If (HR_Cont(i,j)==0) THEN HR_B = Interpolate(HR_A(i,j))     | [SR_UPDATE]
  ELSE  HR_B = HR_A                                           |

  Clip(HR_B, 0, 255)

High Resolution result image in HR_B
```

FIGURA 5.37. Pseudo-código del algoritmo no iterativo v2.1.

5.2.4.1 Resultados de simulación y análisis de calidad y comportamiento de la versión v2.1 usando precisión de ¼ de píxel

Para comprobar que un aumento en el número de *frames* combinados se corresponde con un aumento de la calidad de la imagen resultante se ha diseñado el siguiente experimento: se generan 12 vectores de desplazamiento (TABLA 5.4), siendo el primero el vector (0,0) y siendo los otros 11 vectores aleatorios. El primer vector de desplazamiento es el cero, para que la imagen resultante (ajustada siempre al primer *frame* de la secuencia) quede con desplazamiento nulo, permitiendo la comparación directa con la imagen de referencia, que no ha sido desplazada. De este conjunto de vectores se toman los tres primeros se aplican al primer *frame* de la secuencia KRANT. Luego se toman los cuatro siguientes y se aplican al mismo *frame*, y así sucesivamente hasta llegar a generar 12 imágenes de baja resolución. En total se generan 3+4+5+6+7+8+9+10+11+12=75 *frames* de baja resolución para servir como entrada al ASR versión v2.1. Esta secuencia de entrada genera 10 *frames* de salida, cuyas calidades se muestran en la FIGURA 5.38.

Frame	Vectores de VHR	Vectores de HR	*Frame*	Vectores de VHR	Vectores de HR
0	0 , 0	0 , 0	6	0 , 1	0 , 0.5
1	2 , 2	1 , 1	7	-2 , 1	-1 , 0.5
2	3 , 1	1.5 , 0.5	8	1 , 0	0.5 , 0
3	3 , 3	1.5 , 1.5	9	0 , 2	0 , 1
4	2 , 0	1 , 0	10	2 , 0	1 , 0
5	0 , 3	0 , 1.5	11	0 , -2	0 , -1

TABLA 5.4. Vectores de movimiento generados para una reconstrucción incremental de la imagen de súper-resolución.

Tal y como se esperaba, a medida que aumenta el número de *frames* de entrada combinados, la PSNR va aumentando hasta llegar a 38.44 dB en el *frame* 9 de salida, que proviene de la combinación de 12 *frames* de entrada de baja resolución. De forma empírica se comprueba que tras combinar 6 *frames* de entrada (*frame* 3 de salida), es decir, cuando se alcanzan los 36.55 dB, el ojo humano apenas percibe mejoría en la calidad de la imagen. Además, se observa que el mayor incremento de calidad (mayor pendiente de la PSNR) tiene lugar en los 4 primeros *frames* de salida. Por todo esto, un sistema puede limitarse a combinar 5 ó 6 *frames* de entrada, dependiendo de los recursos disponibles y de la calidad deseada de salida.

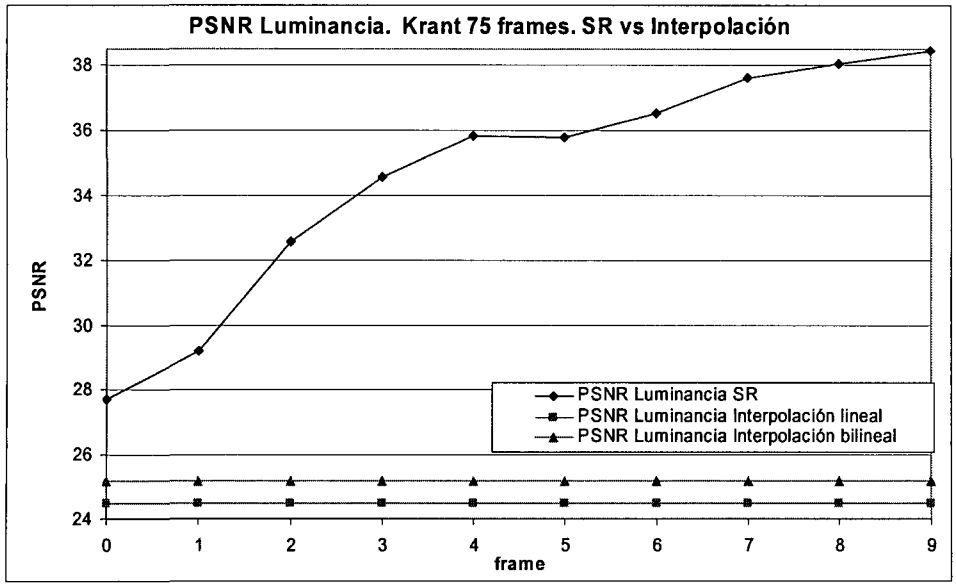


FIGURA 5.38. PSNR de la secuencia KRANT con 10 *frames* de salida incrementales.

Debe tenerse en cuenta que la calidad máxima obtenida en los algoritmos de súper-resolución iterativos es de 34.5635 dB para la luminancia, mientras que en esta versión no iterativa del algoritmo ese valor se alcanza al combinar 6 *frames* de entrada (34.5698 dB) y se supera en más de 3 dB combinando algunos *frames* adicionales. Con respecto al nivel de interpolación bilineal (más alto que el lineal) estamos en todo momento por encima de él (distancia mínima de 2.51 dB y máxima de 11.37 dB).

En la FIGURA 5.39 puede verse el pequeño aumento en la PSNR producido al eliminar los bordes de las imágenes. Sin embargo, debe tenerse en cuenta que la disminución del número de píxeles de una imagen hace aumentar sensiblemente la SNR, y en este caso, el aumento de 1.029 dB de promedio es debido a esta causa, y no a diferencias en los bordes. Esta hipótesis se verifica mediante una inspección de la imagen error (FIGURA 5.40) del *frame* 9. La PSNR de la imagen sin bordes de este *frame* es 1.32 dB superior a la imagen con bordes, pero en los bordes de la imagen de error no se observa ninguna discontinuidad apreciable. Concluimos pues que el aumento de la PSNR de estas imágenes es debido a la disminución del número de píxeles.

En la FIGURA 5.40 se aprecia que la imagen de error del *frame* 9 en el dominio del tiempo (a.2) es altamente uniforme, indicando un error muy pequeño respecto a la imagen de referencia. La transformada de Fourier bidimensional en módulo muestra una eliminación casi total del *aliasing*, mostrando una imagen de error mínima en las bandas espectrales de mayor

potencia (b.2) que es aproximadamente la misma orientación que sigue el mínimo error espectral en fase (c.2).

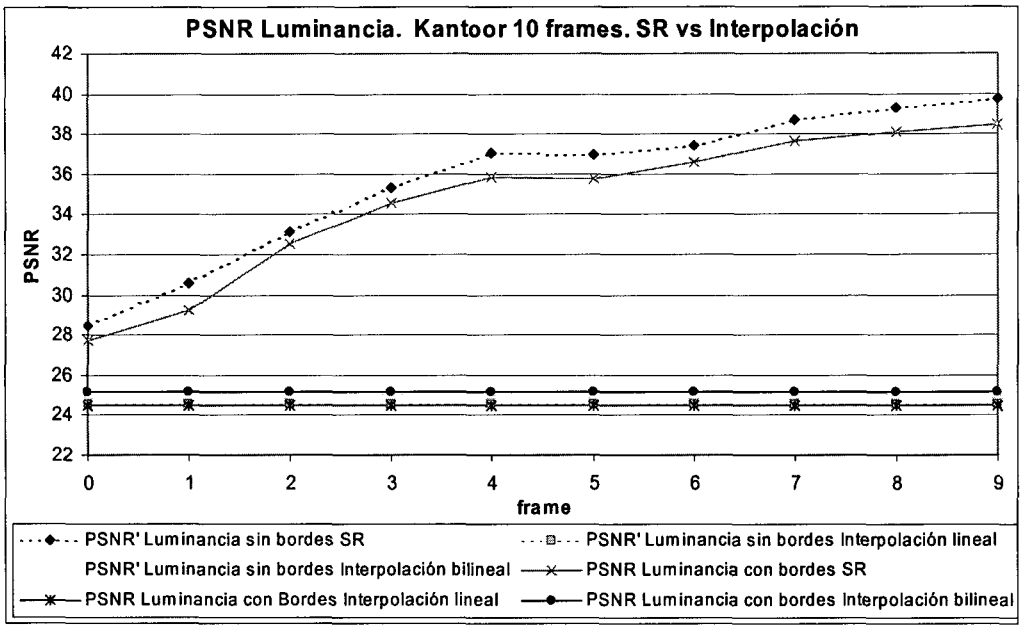


FIGURA 5.39. PSNR de la secuencia KRANT con y sin bordes.

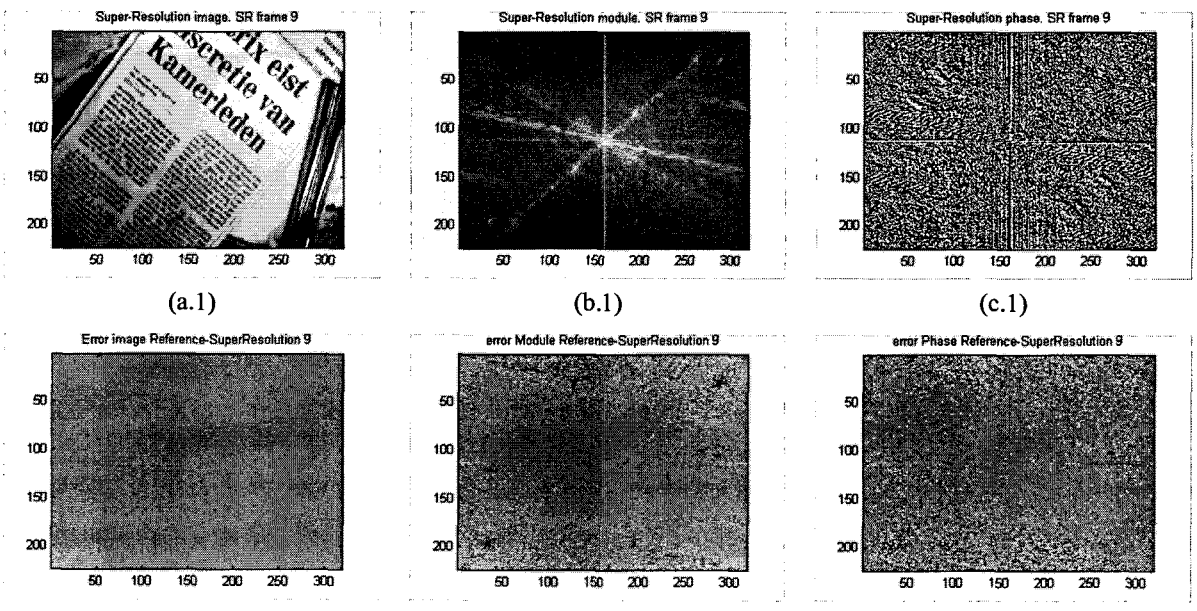


FIGURA 5.40. *Frame* 9 de súper-resolución en el dominio del espacio (a), de la frecuencia en módulo (b) y de la frecuencia en fase (c), así como sus errores asociados (2).

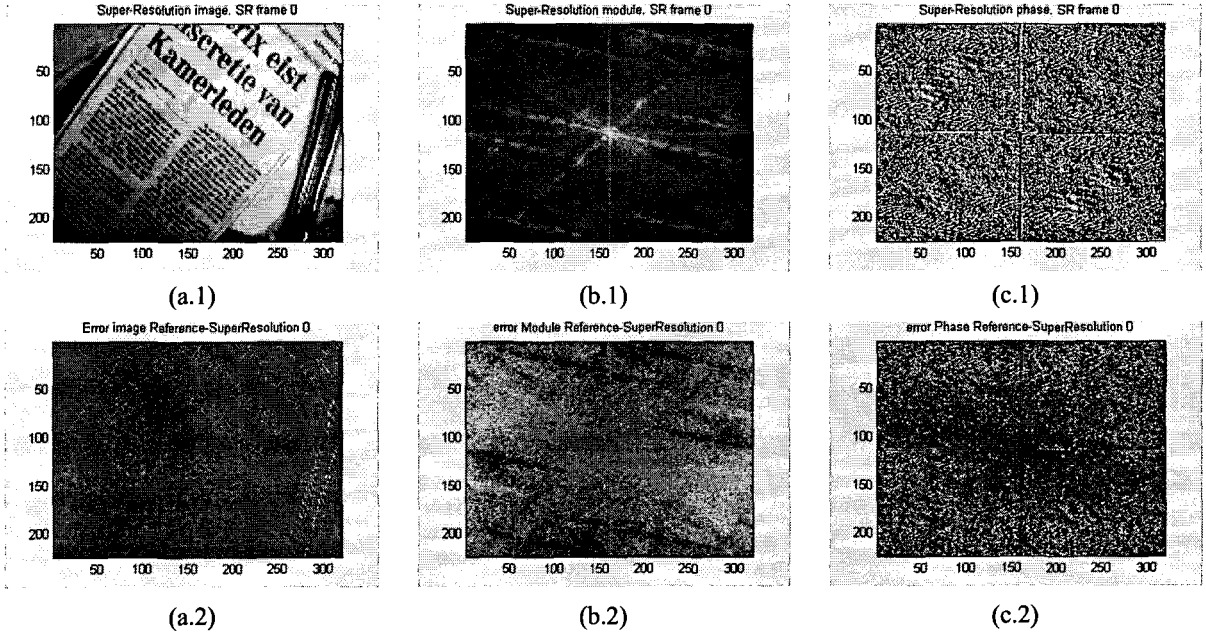


FIGURA 5.41. *Frame* 0 de súper-resolución en el dominio del espacio (a), de la frecuencia en módulo (b) y de la frecuencia en fase (c), así como sus errores asociados (2).

Por completitud, se muestra asimismo la imagen peor de la secuencia (*frame* 0) proveniente de la combinación de tres *frames* de entrada de baja resolución (FIGURA 5.41). En ella se aprecia una mayor cantidad de *aliasing* y errores sustancialmente mayores en todos los casos.

La PSNR de las crominancias también está en todo momento por encima de los niveles de interpolación, aunque no sigue la misma evolución que la luminancia. En la FIGURA 5.42 se muestra la PSNR de la crominancia roja, de mayor energía que la azul, cuyas diferencias mínima y máxima con respecto al nivel de interpolación bilineal son de 0.982 dB y 2.27 dB respectivamente.

En la FIGURA 5.42 también se puede apreciar la relativa independencia de la calidad de la imagen con respecto a la inclusión o no de los bordes de las imágenes. Al igual que en el caso de la luminancia, el pequeño incremento es justificable teniendo en cuenta la disminución del número de píxeles.

En la FIGURA 5.43 se aprecian valores mucho mayores de la PSNR de las crominancias debido a su menor entropía. La crominancia roja tiene un valor promedio de 49.6631 dB, frente al valor promedio de 46.9091 dB de la crominancia azul y el valor promedio de 34.6331 dB de la luminancia.

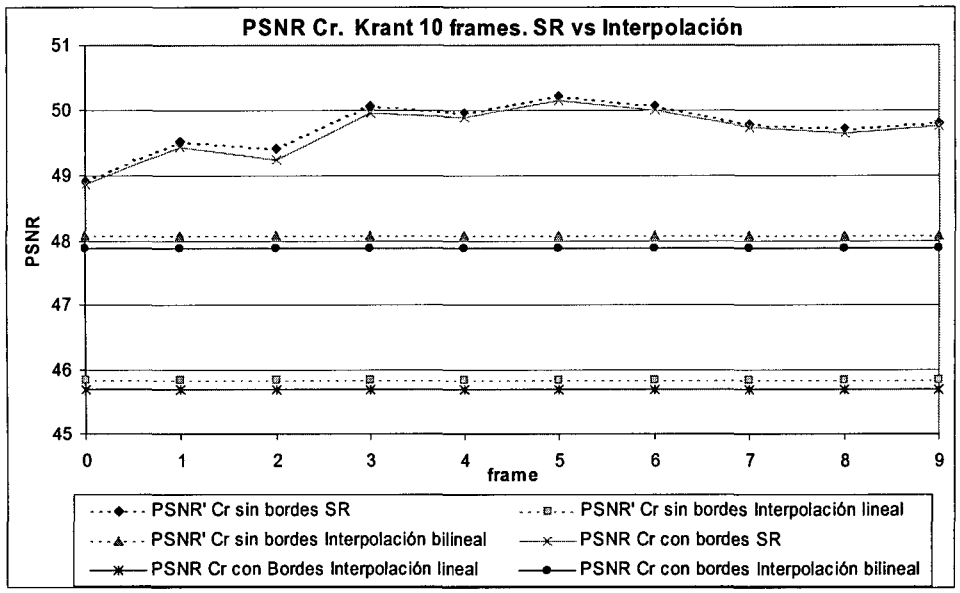


FIGURA 5.42. PSNR de la crominancia roja de la secuencia KRANT con 10 *frames* de salida incrementales con y sin bordes.

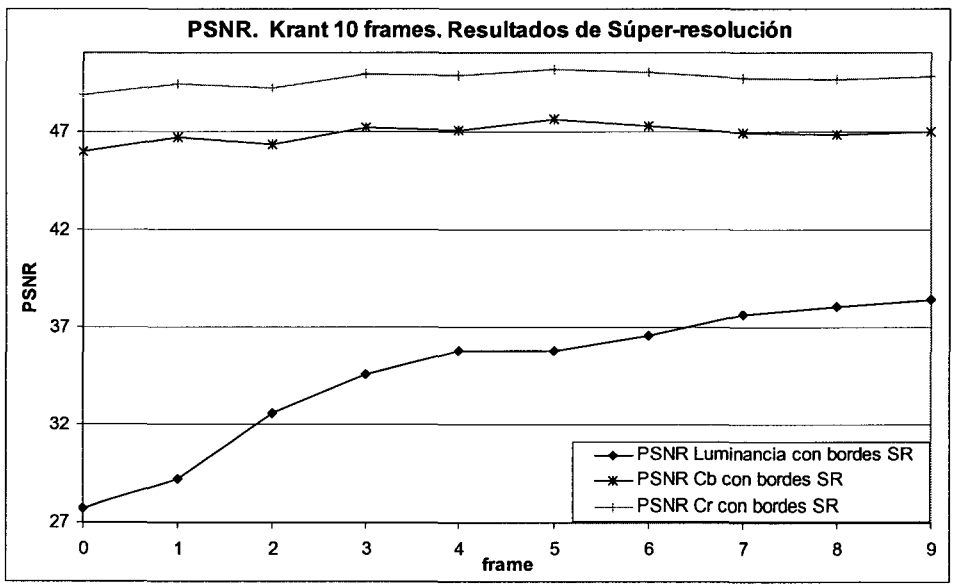
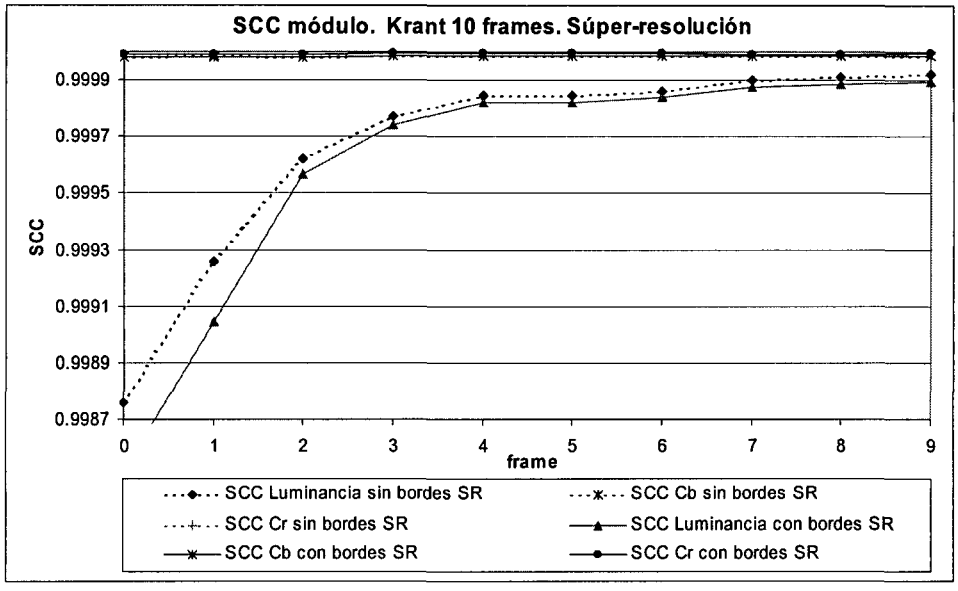


FIGURA 5.43. PSNR de la luminancia y de las crominancias de la secuencia KRANT con 10 *frames* de salida incrementales.

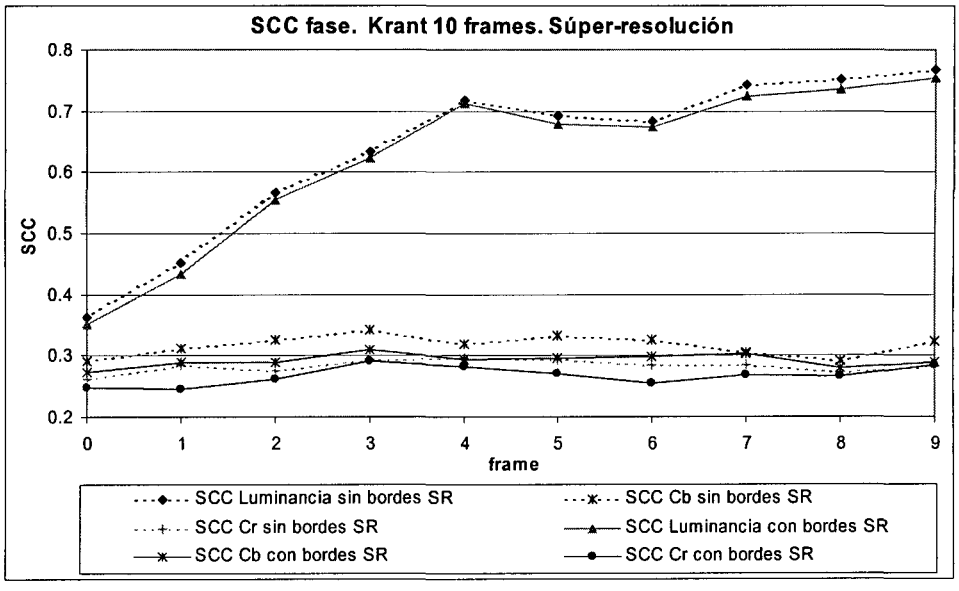
En la FIGURA 5.44 se observa cómo las correlaciones con (líneas continuas) y sin bordes (líneas discontinuas) son muy similares (el coeficiente de correlación promedio en módulo con bordes es de 0.999600396, mientras que sin bordes es de 0.999669774 y en fase con bordes es de 0.624228535 y sin bordes 0.637115433), siendo imputables las diferencias a las diferencias de tamaño entre imágenes. Al igual que en la versión anterior, de la que procede esta nueva versión, las correlaciones en módulo de las crominancias (a) son

substancialmente mayores que la correlación de la luminancia, debido a la menor entropía de las primeras. La correlación espectral en fase (b) de la luminancia es mayor que la de las crominancias, por tener el movimiento mejor ajustado. Se comprueba también que las métricas de crominancia no siguen las mismas tendencias que las de luminancia, y que la luminancia sigue la misma tendencia de aumento, tanto en módulo como en fase.

La TABLA 5.5 recoge un resumen de los valores promedios de las diferentes métricas a lo largo de toda la secuencia de salida. Puede comprobarse cómo las diferencias entre las imágenes con y sin bordes han disminuido, y cómo la PSNR y la correlación en módulo de la luminancia son menores que las de la crominancia, aunque en fase esta tendencia se invierte, siendo mayor la de luminancia que la de crominancia.



(a)

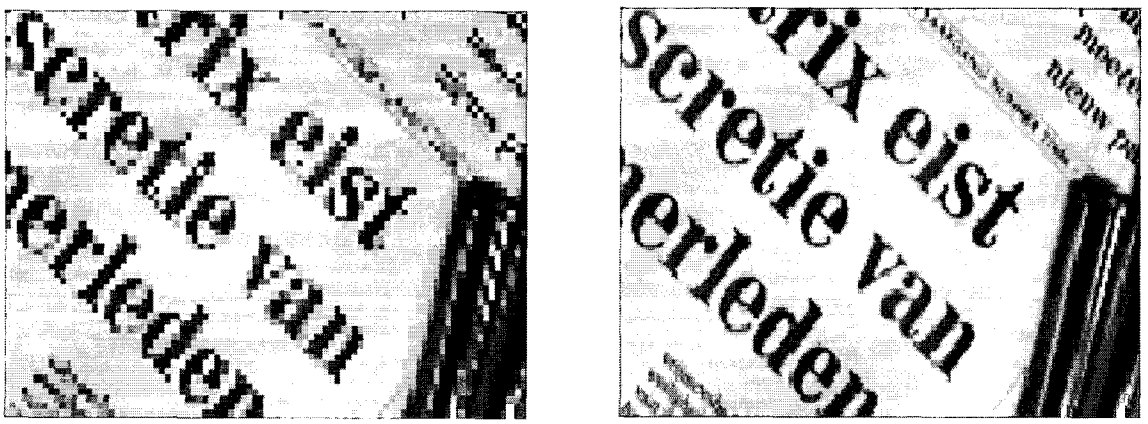


(b)

La FIGURA 5.45 muestra en detalle la esquina superior derecha de la imagen KRANT de entrada y la misa zona tras el proceso de súper-resolución. Es clara la notable mejoría de la imagen tratada, donde se puede leer con mucha claridad las letras sobre el titular del periódico.

PSNR		
Luminancia	34.63318924	35.66243046
Crominancia Azul	46.90917581	47.24841144
Crominancia Roja	49.66319036	49.73881364
SCC módulo		
Luminancia	0.999600396	0.999669774
Crominancia Azul	0.999982804	0.999983823
Crominancia Roja	0.999992188	0.999992243
SCC fase		
Luminancia	0.624228535	0.637115433
Crominancia Azul	0.292558059	0.317201367
Crominancia Roja	0.267466937	0.282985923

TABLA 5.5. Valores promedio de las PSNR y de los SCC en módulo y fase de la secuencia KRANT de 10 *frames* de salida.



(a) (b)

FIGURA 5.45. Detalle ampliado de la imagen original (a) y la de súper-resolución (b) para la secuencia KRANT de 10 *frames*. .

Entre otras aplicaciones este tipo de sistemas resulta muy adecuado para la toma de fotografías digitales (imágenes estáticas). Cuando el usuario desea capturar una imagen, pulsa un actuador que toma la primera imagen como referencia, y de forma continuada y automática toma 3, 4 ó n imágenes más, dependiendo de la memoria disponible en el sistema, para de esta forma poder combinarlas y aumentar la resolución de la imagen. Las imágenes deben tener

cierta cantidad de movimiento entre ellas, como por ejemplo ocurre en fotografías hechas a pulso, sin soporte de cámara.

5.2.4.2 Diagrama de bloques y requerimientos de memoria de la versión v2.1

En la FIGURA 5.46 se muestra el diagrama de bloques del flujo de datos de la versión v2.1 del ASR mostrado en la FIGURA 5.37.

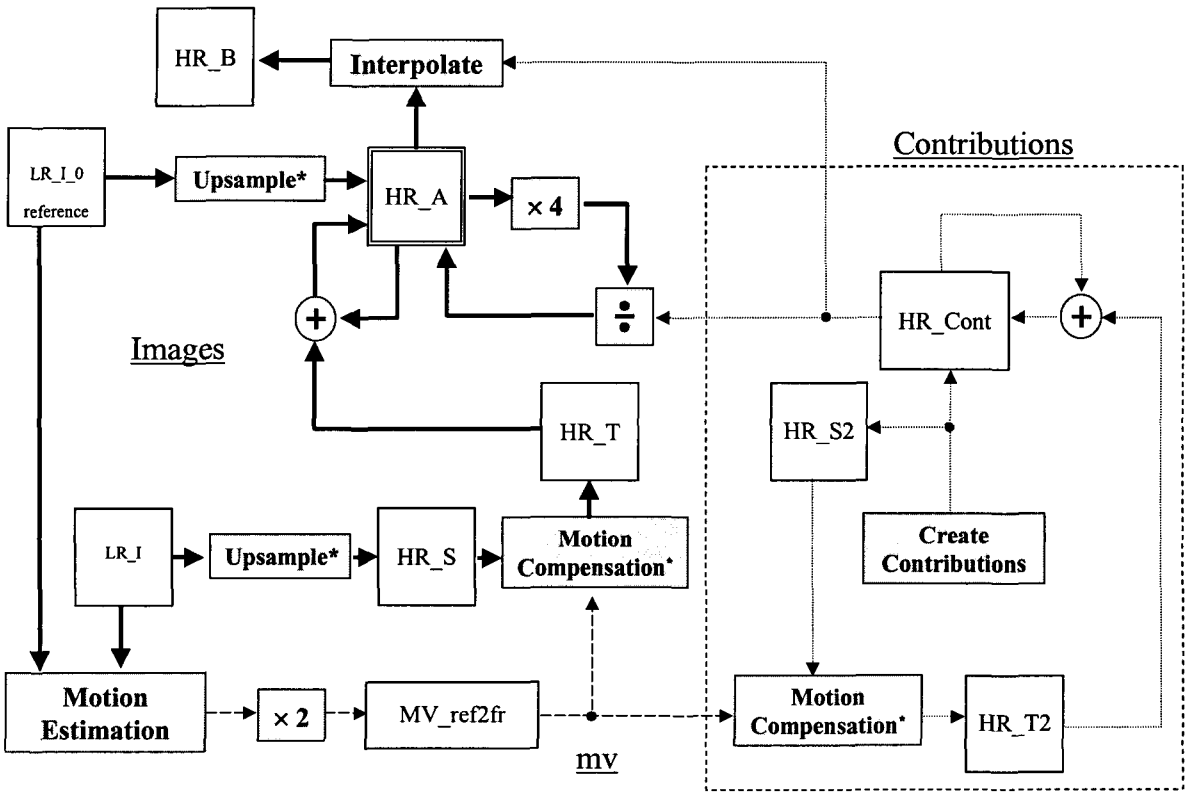


FIGURA 5.46. Diagrama de bloques del algoritmo de súper-resolución v2.1.

Al igual que en la versión v2.0, el diagrama de bloques se ha dividido, por un lado en la zona ocupada del procesamiento de imágenes, que hace uso de las memorias HR_A, HR_T, HR_S, LR_I_0 y LR_I, además de guardar los vectores de movimiento en MV_ref2fr y por otro lado, en una zona para el procesamiento de las contribuciones, que hace uso de las memorias HR_S2, HR_T2 y HR_Cont. Para resaltar las relaciones entre ellas, se han puesto en línea continua los flujos que llevan imágenes, en línea punteada los flujos que llevan contribuciones y en línea discontinua los flujos que llevan los vectores de movimiento.

Además, las funciones de *'upsample'* y *'motion compensation'*, se han marcado con un asterisco para remarcar su funcionamiento diferente cuando se usan en súper-resolución. Con respecto a la versión v2.0 han desaparecido dos memorias de baja resolución y ahora tan solo es necesario almacenar un vector de movimiento. Además, HR_A ha visto aumentado el ancho de palabra a 12 bits.

En la TABLA 5.6 se muestra un resumen de los requerimientos de memoria de la versión 2.1 del ASR, tal y como se recoge en la FIGURA 5.46.

Las memorias HR_T y HR_T2 no se incluyen en estas medidas ya que se usan en esta versión del algoritmo para evitar el solapamiento de datos al realizar la compensación de movimiento, pero este problema no existirá en la versión *hardware* que se implementa finalmente sobre *Picasso*, al existir un *buffer* de 3 filas de macro-bloques que permite evitar precisamente este problema. Estas tres filas o *stripes* también han sido incluidas en los requisitos de memoria.

El la TABLA 5.7 se muestran los requerimientos totales de memoria para los tamaños de imágenes más comunes.

Denominación	Memoria		
	Luminancia (bits)	Crominancia (bits)	Total (bits)
HR_A	*(2·mb_x·2·mb_y·16·16·12)*	*(2·mb_x·2·mb_y·8·8·2·12)*	*18,432·mb_x·mb_y*
HR_B	*(2·mb_x·2·mb_y·16·16·8)*	*(2·mb_x·2·mb_y·8·8·2·8)*	12,288·*mb_x·mb_y*
HR_S	*(2·mb_x·2·mb_y·16·16·8)*	*(2·mb_x·2·mb_y·8·8·2·8)*	12,288·*mb_x·mb_y*
HR_S2	*(2·mb_x·2·mb_y·16·16·8)*	*(2·mb_x·2·mb_y·8·8·2·8)*	12,288·*mb_x·mb_y*
HR_Cont	*(2·mb_x·2·mb_y·16·16·8)*	*(2·mb_x·2·mb_y·8·8·2·8)*	12,288·*mb_x·mb_y*
3 Stripes HR	(2·3·2·mb_y·16·16·8)	(2·3·2·mb_y·8·8·2·8)	36,864·mb_y
LR_I[0]	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
LR_I[1]	*(mb_x·mb_y·16·16·8)*	*(mb_x·mb_y·8·8·2·8)*	*3,072·mb_x·mb_y*
MV_mem[0]	(mb_x·mb_y·8)	0	8· *mb_x·mb_y*
Total (bits)	mb_y · (49,160· mb_x + 24,576)	mb_y · (24,576· mb_x + 12,288)	mb_y · (73.736·mb_x+ 36,864)

TABLA 5.6. Resumen de la memoria utilizada por la versión v2.1 del algoritmo de súper-resolución

En la FIGURA 5.47 se muestran los datos de la TABLA 5.6 expresados en kilo-bytes junto con los de la TABLA 5.2 para la versión 2.0. Los requerimientos de memoria de la versión v2.1 son ligeramente inferiores a los de la versión v2.0 debido a que ya no usamos dos de las

memorias de baja resolución y a que reutilizamos todo el tiempo la misma memoria para los vectores de movimiento. En todo caso, el ahorro de memoria es muy leve, y no resulta realmente significativo frente a los volúmenes totales requeridos.

Tamaño	mb_x	mb_y	Memoria (Kbytes)	Memoria (Mbytes)
SQCIF	8	6	459.05	0.45
QAVGA	9	7	598.56	0.58
QCIF	11	9	931.60	0.91
HAVGA	18	14	2,331.25	2.28
CIF	22	18	3,645.39	3.56
AVGA	36	28	9,198.98	8.98
VGA	40	30	10,936.17	10.68
4CIF	44	36	14,419.55	14.08
16CIF	88	72	57,354.19	56.01

TABLA 5.7. Memoria utilizada por la versión v2.1 del algoritmo de súper-resolución para los tamaños de imágenes más comunes.

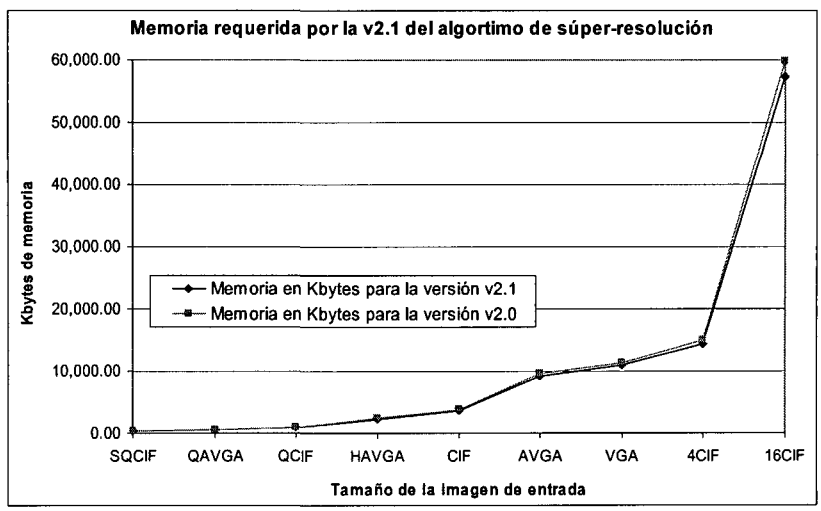


FIGURA 5.47. Memoria usada por el algoritmo de súper-resolución v2.1 para los tamaños de imágenes más comunes.

5.3 Algoritmos de súper-resolución no iterativos para vídeo (v.3)

La última versión que vamos a presentar del algoritmo de súper-resolución trata de corregir los siguientes problemas de la versión anterior:

- El movimiento en vídeo debe ser compensado al último *frame* recibido, con la intención de seguir el movimiento de la secuencia.
- Debe buscarse un mecanismo que permita ir incorporando la información disponible en los nuevos *frames* a medida que estos se van leyendo.
- Asimismo, debe buscarse un mecanismo que permita al algoritmo recuperarse ante oclusiones y/o movimientos rápidos de escena, o cambios de plano o de contexto.
- La cantidad de memoria debe disminuirse todavía más, al objeto de que tenga cabida en la memoria *on-chip* y reducir coste y consumo.

En la FIGURA 5.48 (a) se muestra la estrategia seguida hasta ahora de generar una primera imagen usando el vector de movimiento (0,0) para a continuación ajustar el resto de los *frame*s al primero o de referencia. Sin embargo, en la FIGURA 5.48 (b) se muestra la estrategia seguida en el algoritmo para vídeo, donde los vectores de movimiento se van ajustando *frame* a *frame*.

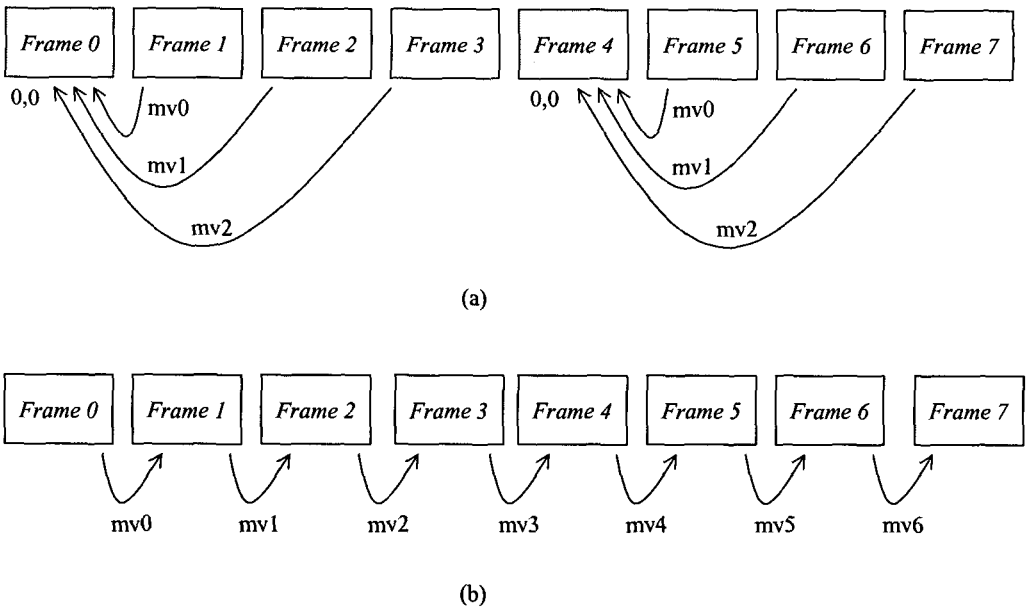


FIGURA 5.48. Estrategia de estimación de movimiento para imagen estática (a) y para vídeo e imágenes en movimiento (b).

5.3.1 Aumento de la robustez del algoritmo de súper-resolución

Los algoritmos de súper-resolución suponen que la secuencia de imágenes está relacionada entre si, produciéndose pequeños movimientos entre las imágenes que permiten incorporar nueva información a la imagen resultante. Sin embargo, a la hora de aplicar estos algoritmos a escenas reales, deben tenerse en cuenta las siguientes limitaciones:

- Un objeto de la imagen puede moverse muy deprisa, por lo que provocará la oclusión de otros elementos de la escena sobre los que no se podrá incorporar nueva información. Tampoco puede incorporarse nueva información del objeto que se desplaza muy rápidamente, ya que un vector de movimiento excesivamente grande provocaría artefactos indeseados al intentar compensar su movimiento a nivel de bloque.
- Los movimientos rápidos de la cámara provocan cambios entre escenas no vinculadas entre sí. A esto se le conoce como cambio de contexto. En estas condiciones no es posible obtener nueva información de la imagen actual.

La única solución posible en estos casos, ante la falta de información, es la de interpolar aquellos elementos que no puedan ser mejorados usando técnicas de super-

resolución. En este sentido se aprovechará la información suministrada por el estimador de movimiento destinada a ayudar al *software* de compresión sobre la decisión de codificación del macro-bloque actual. Los elementos de decisión disponibles son los siguientes:

- **Vectores de movimiento**. Un vector de movimiento excesivamente grande, en comparación con el vector medio, es indicio de grandes desplazamientos locales en la imagen.
- **SAD *inter*.** Es la suma de las diferencias absolutas entre la imagen actual y la anterior compensada en movimiento. Se da un valor por cada macro-bloque, en función del vector de movimiento. Valores muy elevados de este parámetro indican que el macro-bloque actual tiene escasa similitud con el macro-bloque correspondiente de la imagen anterior. N función de los vectores de movimiento utilizados podemos realizar una sub-calificación en:
  - **SAD *inter* global.** Es el obtenido cuando se utiliza el vector de movimiento global
  - **Sad *inter* local.** Es el obtenido cuando se utiliza los vectores de movimiento locales
- **SAD *intra*.** Es una medida de las variaciones producidas dentro de la propia imagen, calculando el SAD de cada macro-bloque como sus diferencias absolutas con respecto a su valor promedio. Valores muy altos de este parámetro indican altas diferencias respecto al valor medio y por lo tanto suelen ser zonas con gran nivel de detalle.
- **SAD SR.** Es al suma de las diferencias absolutas entre la imagen de súper-resolución anterior y la nueva imagen de entrada de baja resolución, pero sólo en las posiciones de píxel de la imagen de baja resolución. Nos da una medida de similitud entre imágenes de súper-resolución.

La FIGURA 5.49 esquematiza las diferentes entradas que ayudan a la toma de decisiones sobre que acción tomar para cada macro-bloque:

- Aplicar súper-resolución usando vectores locales.
- Aplicar súper-resolución usando vectores globales.
- Interpolar el macro-bloque actual.

A partir de los vectores de movimiento se incorpora un nuevo parámetro de decisión llamado *'mv_sad'*, obtenido como la suma de las diferencias absolutas de cada vector con el vector de movimiento global. Si el *'mv_sad'* es bajo, significa que la imagen está gobernada por movimiento global, mientras que si es alto predominarán los movimientos locales.

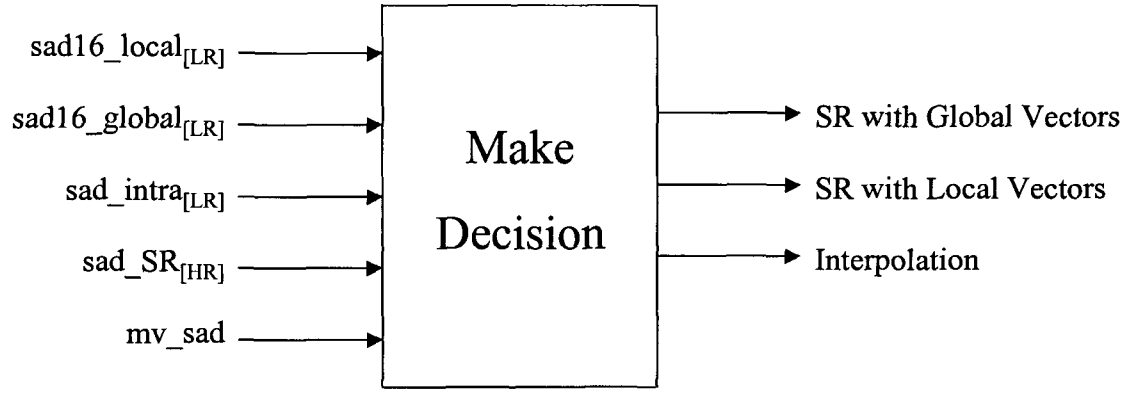


FIGURA 5.49. Esquema de entradas y salidas para tomar decisiones sobre el modo de codificación en súper-resolución.

Todos estos parámetros se obtienen a partir de las imágenes de baja resolución, lo que acelera su obtención. Realizando un estudio sobre varias secuencias se han determinado una serie de umbrales que ofrecen una buena calidad para las secuencias de prueba. El algoritmo para la toma de decisiones es el mostrado en la FIGURA 5.50.

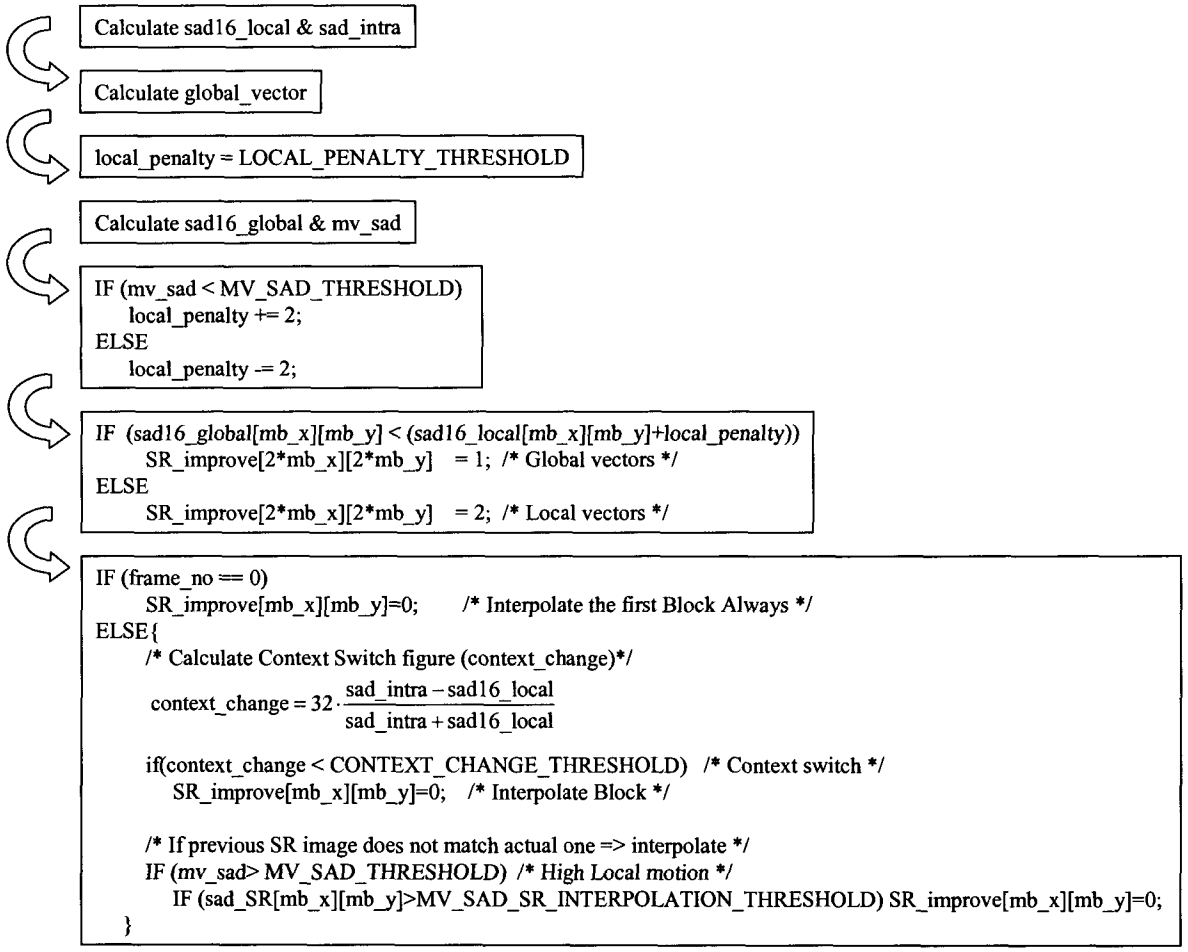


FIGURA 5.50. Esquema seguido en la toma de decisiones sobre el modo de codificación en el algoritmo de súper-resolución .

El algoritmo comienza obteniendo los valores de '*sad_local*' (*sad16_local*) y '*sad_intra*', junto con los vectores de movimiento locales, que son los valores de salida del estimador de movimiento. A partir de los vectores de movimiento locales se puede calcular el vector de movimiento global, bien como la media de todos ellos, como la selección del par más frecuente o como la selección del componente 'x' e 'y' más frecuentes por separado. Se inicializa la variable '*local_penalty*' a un valor umbral LOCAL_PENALTY_THRESHOLD. Esta variable pretende favorecer o penalizar el movimiento global frente al global o viceversa. Ya conocido el vector global, se puede calcular el '*mv_sad*', como la suma de la diferencias absolutas entre cada vector local y el vector global. Además, pasándole el vector global al estimador de movimiento, le podemos pedir que lo evalúe y nos de el valor de '*sad_global*'.

Si el '*sad_mv*' está por debajo del umbral MV_SAD_THRESHOLD, habrá que favorecer el movimiento global, incrementando el valor de '*local_penalty*'. En caso contrario lo decrementaríamos para favorecer el movimiento local. A continuación se actualiza, el

'sad_local' con el valor del *'local_penalty'*, para compararlo con el *'sad_global'*. Si el *'sad_global'* es inferior al *'sad_local'* modificado, entonces escogeremos el vector de movimiento global, y en caso contrario escogeremos el vector de movimiento global. Una vez que hemos discriminado entre movimiento local o global, debemos discriminar entre si aplicar súper-resolución con el movimiento escogido o interpolar. Para ello se han establecido algunos umbrales de decisión de la siguiente forma: si se trata del primer *frame* de una secuencia, siempre tendremos que interpolar. Si no se trata del primer *frame*, determinamos si estamos ante un caso de cambio de contexto. Para ello aplicamos la formula empírica (5.3).

$$\text{context\_change} = 32 \cdot \frac{\text{sad\_intra} - \text{sad16\_local}}{\text{sad\_intra} + \text{sad16\_local}} \tag{5.3}$$

Esta formula, que recoge las diferencias escaladas entre el *'sad_intra'* y el *'sad_local'*, ha demostrado ser un estimador bastante estable para determinar el cambio de contexto como se verá a continuación. Si el valor de la ecuación (5.3) está por debajo del umbral CONTEXT_CHANGE_THRESHOLD, entonces interpolaremos. Finalmente estimamos si existe mucha diferencia entre la imagen actual y la anterior, de forma que sea más conveniente interpolar que aplicar súper-resolución. Interpolaremos si existe mucha cantidad de movimiento local en dicho macro-bloque (*'mv_sad'* mayor que el umbral MV_SAD_THRESHOLD) o si además la imagen de súper-resolución actual presenta grandes diferencias con la nueva imagen de entrada (*'sad_SR'*mayor que el umbral MV_SAD_SR_INTERPOLATION_THRESHOLD).

Estos umbrales han sido determinados de forma empírica a partir del estudio de catorce series de imágenes y ofrecen buenas calidades en todas las condiciones. Esta metodología aumenta además la robustez del algoritmo, asegurando que en ningún caso descenderemos por debajo del nivel de interpolación.

Para determinar los umbrales se ha procedido a realizar un estudio de los valores de los parámetros usados como evaluadores en diferentes condiciones. El caso más drástico es el de cambio de contexto y para ello se ha confeccionado una secuencia de vídeo sintética basada en la secuencia KRANT. Se toma el primer *frame* y se le aplican ocho vectores aleatorios, generando 8 *frames* desplazados. Luego se salta al *frame* 15, bastante diferente, y se le aplican 8 nuevos vectores de desplazamiento aleatorios (Tabla 5.8). El resultado es una secuencia de 16 *frames* con desplazamientos aleatorios, con un brusco cambio de contexto en el *frame* 8 (comenzando a contar a partir del cero). En a FIGURA 5.51 se muestran los *frames* 0 y 15 de la secuencia KRANT.

(a) (b)

FIGURA 5.51. *Frame* 0 (a) y *frame* 15 (b) de la secuencia KRANT.

frame	Secuencia	Desplazamiento	*frame*	Secuencia	Desplazamiento
0	0	3 -3	15	8	3 -3
0	1	1 -2	15	9	-2 -2
0	2	0 1	15	10	0 0
0	3	-1 -2	15	11	3 2
0	4	3 -3	15	12	1 -1
0	5	-2 0	15	13	0 -2
0	6	-2 3	15	14	1 -2
0	7	1 0	15	15	0 2

Tabla 5.8. Vectores de desplazamiento aplicados a los *frames* 0 y 15 de KRANT.

Si representamos los valores de la ecuación (5.3) en el caso de esta secuencia, obtenemos el gráfico mostrado en la FIGURA 5.52. El *frame* en el que se produce el cambio de contexto queda claramente destacado con valores inferiores a 5. Por lo tanto, ese es el valor al que debemos establecer el umbral.

Obsérvese sin embargo, que no sólo el *frame* donde se produce el cambio de contexto queda por debajo del umbral. Existen otros macro-bloques, unos pocos, que también quedan bajo el umbral y que por lo tanto serán interpolados.

Para determinar el valor del umbral MV_SAD_THRESHOLD, debemos representar los valores de '*mv_sad*', que es lo que se recoge en la FIGURA 5.53. En este caso, también se obtiene un valor umbral MV_SAD_THRESHOLD = 5.

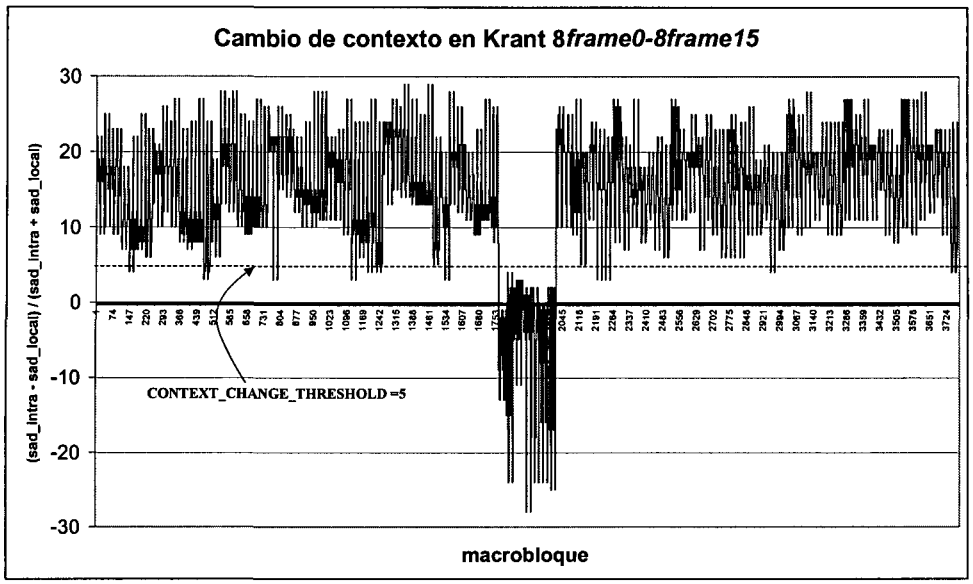


FIGURA 5.52. Determinación del umbral CONTEXT_CHANGE_THRESHOLD.

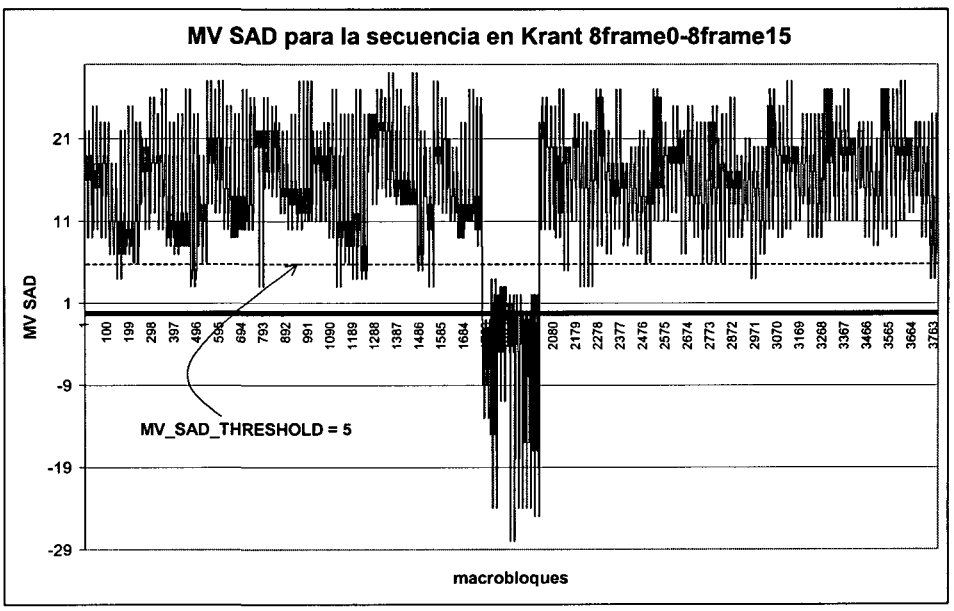


FIGURA 5.53. Determinación del umbral MV_SAD_THRESHOLD.

El tercer umbral que se necesita determinar para ajustar la toma de decisiones es el MV_SAD_SR_INTERPOLATION_THRESHOLD, y para ello es necesario ver el comportamiento del *'sad_SR'*, que es lo que recoge la FIGURA 5.54. En este caso, los mayores errores se producen, como era de esperar, en el cambio de contexto. Un valor umbral de 3000 asegura al menos la correcta interpolación en el cambio de contexto, aunque algunos macro-bloques con altos errores también traspasen este umbral.

Debe tenerse en cuenta que el *'sad_SR'* sólo se calcula en los píxeles de súper-resolución, por lo que en el resto de los píxeles su valor es cero. Esta característica debe ser tenida en cuenta para evitar realizar la comparación en los macro-bloques donde la *'sad_SR'* es cero.

El valor del umbral LOCAL_PENALTY_THRESHOLD se ha establecido al mismo valor que su incremento, para duplicar o eliminar su peso según interese. Finalmente, la Tabla 5.9 recoge los valores de los umbrales determinados para el conjunto de secuencias de vídeo usadas como pruebas.

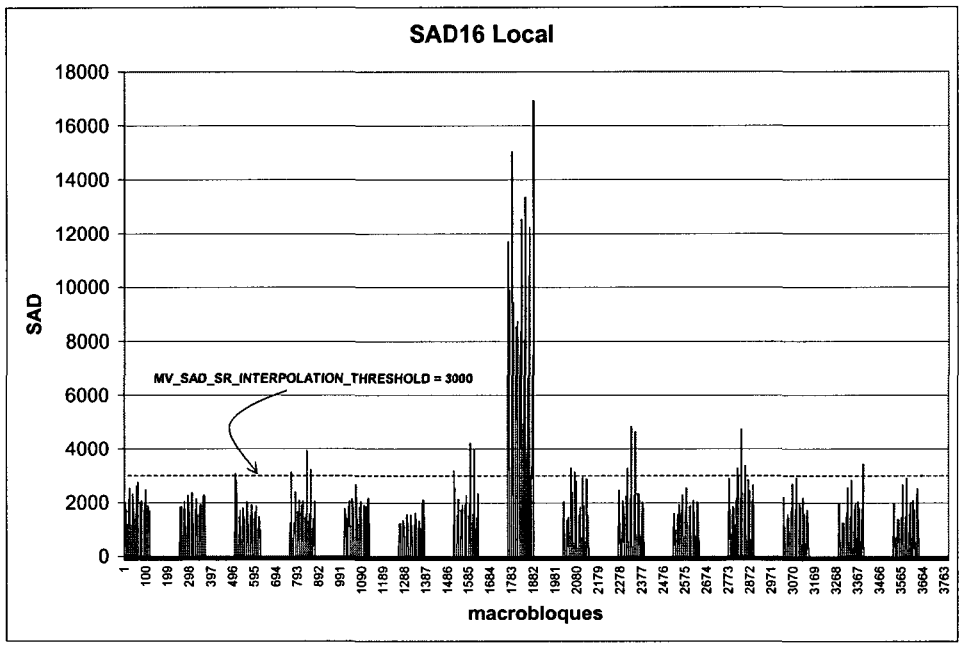


FIGURA 5.54. Determinación del umbral MV_SAD_SR_INTERPOLATION_THRESHOLD.

Umbral	Valor
MV_SAD_SR_INTERPOLATION_THRESHOLD	3000
MV_SAD_THRESHOLD	5
CONTEXT_CHANGE_THRESHOLD	5
LOCAL_PENALTI_THRESHOLD	2

Tabla 5.9. Umbrales usados en la versión v3 del SRA.

5.3.2Pseudo-código del algoritmo v3

Para disminuir la cantidad de memoria usada por el algoritmo se ha optado por recurrir a un esquema realimentado, en donde la última imagen de súper-resolución obtenida se reutiliza para mejorar la resolución de la imagen actual. En el esquema de huecos propuesto en el apartado 5.2, la nueva imagen de baja resolución se interpola a alta resolución, dejando huecos en los lugares de píxeles agregados para incrementar su tamaño. Los píxeles de la imagen de baja resolución se consideran todos válidos y se conservan en su totalidad. Los huecos generados se intentan rellenar con la información de la imagen de súper-resolución anterior, aplicando un criterio de similitud por bloques entre la imagen de baja resolución anterior y la nueva imagen obtenida. Si no se satisface este criterio, se procede a interpolar los huecos con la información disponible.

En la FIGURA 5.55 y en la FIGURA 5.56, se muestra la primera y segunda parte respectivamente de la versión v3 del algoritmo de súper-resolución. Aunque la imagen final de súper-resolución se sigue almacenando en HR_B, algo que se ha mantenido en todas las versiones de los SRA, en este caso ha sido necesario definir una nueva memoria HR_B2 donde se almacena la imagen de súper-resolución que será realimentada. El motivo de mantener esta imagen es que, si bien interesa rellenar los huecos a la hora de mostrar la imagen, es preferible mantenerlos a la hora de realimentar, ya que de esa forma se hace posible completar la información ausente con nuevos datos sin que estos se mezclen con valores enrarecidos procedentes de interpolar los píxeles vecinos. La memoria de contribuciones HR_Cont se mantiene en esta versión como una memoria completa, por comodidad, pero realmente es una contabilidad que se realizará a nivel de macro-bloque, por lo que, ni es necesario mantener sus valores entre imágenes ni es necesario que tenga un tamaño de toda una memoria completa. Con mantener una pequeña memoria temporal de tamaño macro-bloque (64×6×3 bits) es suficiente.

La fase de inicialización se limita a interpolar la primera imagen de entrada, ya que inicialmente no tenemos ninguna información más que añadir. Sin embargo, en el codificador quedan almacenados todos los huecos generados. La memoria HR_W almacena los mismos valores que HR_Cont, pero sólo con carácter temporal. La relación entre HR_W y HR_Cont es la misma que la que existe entre HR_B y HR_B2, pero entre las contribuciones.

Para aumentar la convergencia del estimador de movimiento, el primer paso es realizar el filtrado de las imágenes de baja resolución a partir de las cuales se van a obtener los

vectores de movimiento. La imagen previa de baja resolución, filtrada siempre, se almacena en la memoria LR_P. Aunque en la parte del algoritmo mostrado en la FIGURA 5.55 aparece una función ***Evaluate*** que obtiene las sumas de diferencias absolutas, en realidad estos valores son suministrados directamente por el estimador de movimiento, por esta razón la escribimos en cursiva. Las tareas que sí deben ser realizadas en *software* son las de selección del vector global y el calculo del SAD por macro-bloque. En concreto, el '*mv_sad*' para cada macro-bloque se calcula como la suma del valor absoluto de la resta de sus componentes en cada dirección, como recoge la ecuación (5.4).

$$\begin{aligned} mv\_sad[mb\_x][mb\_y] = {} & |mv\_local.x[mb\_x][mb\_y] - mv\_global.x| + \\ & |mv\_local.y[mb\_x][mb\_y] - mv\_global.y| \end{aligned} \tag{5.4}$$

Para calcular el '*sad_SR*' se necesita calcular previamente y compensar el movimiento de HR_B hacia la imagen anterior, ya que HR_B contiene la imagen de súper-resolución anterior pero sin huecos.

El cálculo del '*sad_SR*' se limita a realizar la suma de los valores absolutos de las diferencias entre la imagen actual y la anterior de súper-resolución en los valores de píxeles correctos de la imagen actual (es decir, en los píxeles de baja resolución) para cada macro-bloque.

En las memorias temporales HR_T y HR_T2 se almacenan las imágenes compensadas y son introducidas para evitar sobrescribir datos en el momento de realizar la compensación de movimiento. Debemos recordar que el compensador de movimiento usado está especialmente modificado para súper-resolución, y que introduce ceros en los bordes en lugar de replicarlos. Por esta razón lo hemos denominado **Motion_Compensation°**

Las instrucciones SR_ADD_CONT y SR_ADD realizan en realidad la suma de las contribuciones y los píxeles anteriores compensados con los actuales, pero como los datos actuales están compuestos por ¾ partes de ceros, que interesa conservar sólo en píxel nuevo, se puede sustituir la suma por una simple sustitución de datos, tal y como se ve en la FIGURA 5.57. De esta forma se evita realizar una operación aritmética, sustituyéndola por una simple asignación. La variable '*SR_improve*', delata si se van a realizar mejoras de súper-resolución en el macro-bloque, o por el contrario va a ser interpolado, en cuyo caso se deja el píxel a cero.

*** Non iterative video super-resolution algorithm v3**

Initial contribution: M = 4 . Each new image is the reference. (shadow lines are for data calculations)
LR_I = Read a new spatial-aliased low-resolution image

init: fr=0

Inicialización

```
IF (fr==0) THEN  // Initialize. HR_B2 and HR_Cont are for feedback purposes
    LR_P = filter(LR_I)                                                          [SR_INIT_P]
    Clip(LR_P, 0, 255)

    HR_A.lum    = Upsample_Holes(LR_I.lum)
    HR_A.chrom = Upsample_Neighbours(LR_I.chrom)                                 [SR_INIT_A_B]
    HR_B2 = HR_A

    HR_Cont = Create_image_contributions  lum=(M 0; 0 0); chrom=(M M; M M)       [SR_INIT_CONT]
    HR_W = HR_Cont

    IF (HR_W(i)==0) THEN   HR_B.lum = Interpolate(HR_B2.lum)
    ELSE  HR_B.lum = HR_B2.lum                                                   [SR_UPDATE]
    HR_B.chrom = HR_B2.chrom
    Clip(HR_B, 0, 255) Clip(HR_B2, 0, 255)

    fr = 1;
ELSE
    LR_T = filter(LR_I)                                                          [SR_FILTER]
    Clip(LR_T, 0, 255)

    MV_local[mb_x][mb_y] = Motion_Estimation (LR_P, LR_T)
```

Control

```
    sad16_local[mb_x][mb_y] = Evaluate(LR_P, LR_T, MV_local[mb_x][mb_y])
    sad_intra[mb_x][mb_y] = Evaluate(LR_P, LR_T, MV_local[mb_x][mb_y])

    MV_global = Select_global_motion_vector(MV_local[mb_x][mb_y])
    sad16_global[mb_x][mby]=Evaluate(LR_P, LR_T, MV_global)

    MV_sad = Calc_MV_sad(MV_local, MV_global)   MV_sad = Σ|MV_local[mb_x][mb_y] − MV_global|
    local_penalty = LOCAL_PENALTY_THRESHOLD

    IF (MV_sad < MV_SAD_THRESHOLD)
        local_penalty += 2
    ELSE
        local_penalty -= 2

    IF (sad16_global[mb_x][mb_y] < sad16_local[mb_x][mb_y]+ local_penalty)
        SR_improve[2·mb_x][ 2·mb_y] = 1      // Global vectors
        MV[mb_x][mb_y] = MV_global
    ELSE
        SR_improve[2·mb_x][ 2·mb_y] = 2      // Local vectors
        MV[mb_x][mb_y] = MV_local[mb_x][mb_y]
```

FIGURA 5.55. Primera parte del pseudo-código de la versión v3 del SRA.

```
HR_T  = Motion_Compensation°(HR_B2, MV)                                   [SR_MOT_COMP]

HR_T2 = Motion_Compensation° (HR_Cont, MV)                                [SR_MOT_COMP_CONT]

HR_S.lum    = Upsample_Holes(LR_I.lum)                                    [SR_UPSAMPLE]
HR_S.chrom = Upsample_ Neighbours(LR_I.chrom)

HR_S2 = Create_image_contributions  lum=(M 0; 0 0); chrom=(M M; M M)      [SR_INIT_CONT]

HR_D = Motion_Compensation°(HR_B, MV)                                     [SR_MOT_COMP_B]
sad_SR = Calc_sad_SR(HR_S, HR_D)                                          [SR_CALC_SAD]

Control
IF (frame_no == 0)
   SR_improve[mb_x][mb_y]=0                 // Interpolate the first image always
ELSE
   context_change = 32 · (sad_intra − sad16_local) / (sad_intra + sad16_local)
   IF(context_change < CONTEXT_CHANGE_THRESHOLD)                          // Context switch
      SR_improve[mb_x][mb_y]=0                                            // Interpolate block

   IF (mv_sad > MV_SAD_THRESHOLD)                                         // High local motion
      IF(sad_SR[mb_x][mb_y]>MV_SAD_SR_INTERPOLATION_THRESHOLD)            // High differences
         SR_improve[mb_x][mb_y]=0                                         // Interpolate block
END IF

Lum  IF (even & even)            HR_W = HR_S2     (new contribution)
     ELSE IF(SR_improve)         HR_W = HR_T2     (previous contribution)
     ELSE                        HR_W = 0         (interpolation)
Chr  HR_W = HR_S2 + HR_T2                                                 [SR_ADD_CONT]
                                                                          (HR_W = HR_S2 + HR_T2)
     IF (HR_W(i) ==0) THEN       HR_Cont = 0      (feedback)
     ELSE                        HR_Cont = M

Lum  IF (even & even)            HR_A = HR_S      (new pixel)
     ELSE IF (SR_improve)        HR_A = HR_T      (previous pixel)
          ELSE                   HR_A = 0         (interpolation)
Chr  IF (SR_improve)     HR_A = HR_S + HR_T       (superresolution)          [SR_ADD]
     ELSE                HR_A = HR_S              (interpolation)            (HR_A = HR_S + HR_T)

IF (HR_W(i)==0)  HR_B2 = 0                                                [SR_ADJUST]
ELSE             HR_B2 = M * HR_A/HR_W   (feedback)

IF (HR_W(i)==0) THEN  HR_B.lum = Interpolate(HR_B2.lum)
ELSE  HR_B.lum = HR_B2.lum                                                [SR_UPDATE]
HR_B.chrom = HR_B2.chrom
Clip(HR_B, 0, 255)  Clip(HR_B2, 0, 255)

LR_P = filter(LR_I)                                                       [SR_INIT_P]
Clip(LR_P, 0, 255)

fr = fr +1;
END IF
```

FIGURA 5.56. Segunda parte del pseudo-código de la versión v3 del SRA.

Sin embargo, debido a que los valores de crominancia están diluidos, al provenir de la compartición de varios píxeles de luminancia, no es posible aplicar aquí la misma estrategia, y sus valores deben ser efectivamente primero sumados y luego promediados con la instrucción SR_ADJUST. La instrucción SR_UPDATE sólo afecta a la memoria que alberga la imagen que va a ser presentada HR_B, es decir, interpola los huecos que no han podido ser cubiertos. Evidentemente, en crominancia no pueden quedar huecos, por lo que la instrucción se limita a dejar pasar la crominancia de forma íntegra a la salida. El último paso del algoritmo es dejar la imagen de baja resolución filtrada en LR_P para ser usada como referencia en el siguiente paso.

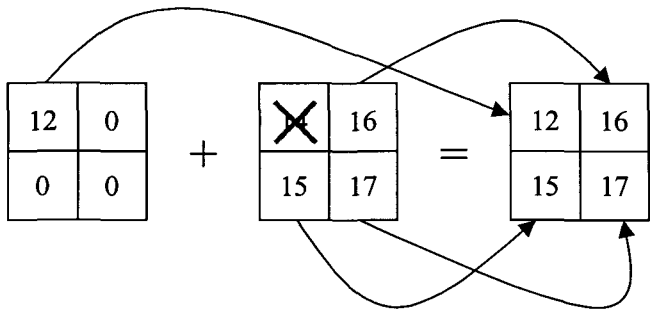


FIGURA 5.57. Realización de la suma como una sustitución de píxeles.

5.3.3 Diagrama de bloques y requerimientos de memoria de la versión v3

El diagrama de bloques del flujo de datos de esta versión del ASR es el mostrado en la FIGURA 5.58. En esta diagrama se han eliminado todas las memorias temporales, dejando solamente aquellas que están presentes en su implementación en el codificador híbrido, es decir: LR_I, LR_P, y HR_B.

La ruta seguida por los datos de las imágenes se ha remarcado con líneas más oscuras, los operadores aparecen sombreados y las memorias recuadradas en claro y con líneas simples. A la hora de su implementación debe tenerse en cuenta además la adición de tres franjas adicionales de macro-bloques para evitar el solape de datos al realizar la compensación de movimiento.

En la TABLA 5.10 se muestra un resumen de los requerimientos de memoria de la versión v3 del ASR expresadas en macro-bloques de baja resolución y excluyendo todas las

memorias temporales. Obsérvese que la memoria de alta resolución usada es en realidad la que se denominaba HR_B2, es decir, conservando los huecos, y es en el momento de darle salida a la imagen, cuando se interpolan estos huecos, dejando a otro sub-sistema de salida la tarea de darle soporte de memoria a la imagen final que entrega *Picasso*.

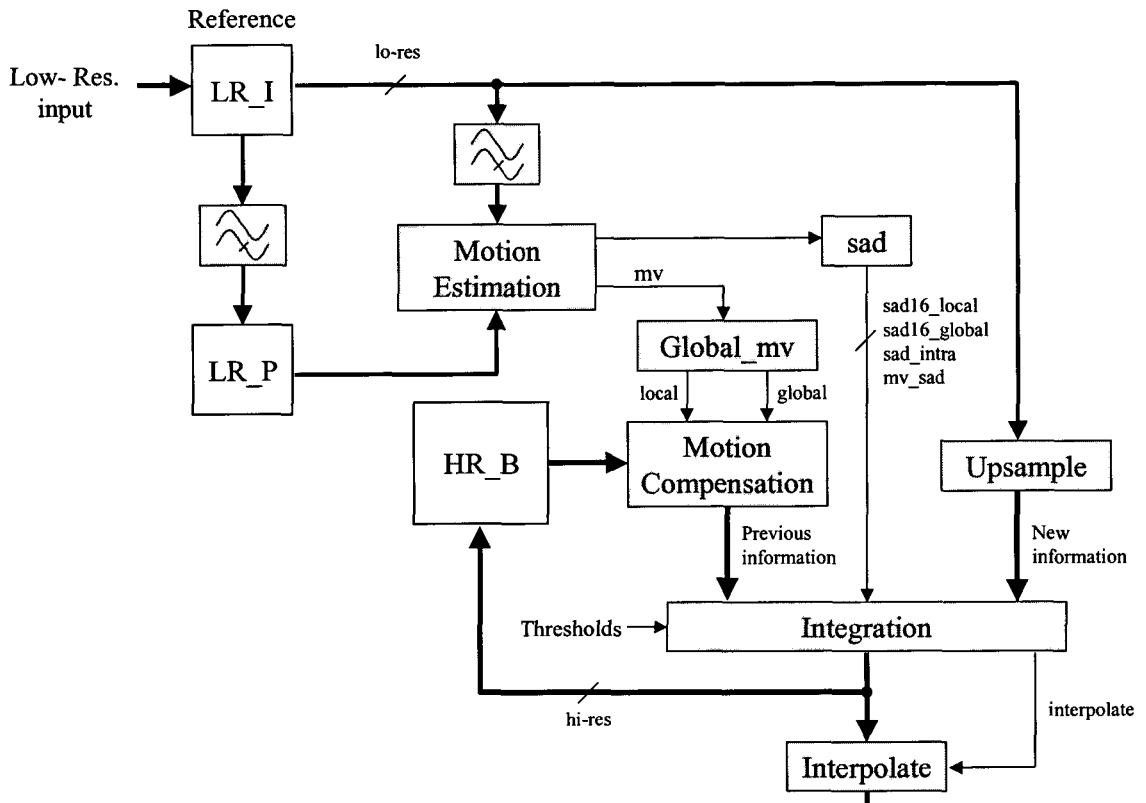


FIGURA 5.58. Diagrama de bloques del flujo de datos del algoritmo de súper-resolución v3.

Nombre	Memoria		
	Luminancia (bits)	Crominancia (bits)	Total (bits)
HR_B	(2·mb_x·2·mb_y·16·16·8)	(2·mb_x·2·mb_y·8·8·2·8)	12.288·mb_x·mb_y
3 Stripes HR	(2·3·2·mb_y·16·16·8)	(2·3·2·mb_y·8·8·2·8)	36.864·mb_y
LR_I	(mb_x·mb_y·16·16·8)	(mb_x·mb_y·8·8·2·8)	3.072·mb_x·mb_y
LR_P	(mb_x·mb_y·16·16·8)	(mb_x·mb_y·8·8·2·8)	3.072·mb_x·mb_y
MV_mem	(mb_x·mb_y·8)	0	8· mb_x·mb_y
Total (bits)	mb_y· (12.296·mb_x+24.576)	mb_y· (6.144·mb_x+ 12.288)	mb_y· (18.440·mb_x+36.864)

TABLA 5.10. Resumen de la memoria utilizada por la versión v3 del algoritmo de súper-resolución

El la TABLA 5.11 se muestran los requerimientos totales de memoria para diferentes tamaños de imágenes.

Tamaño	mb_x	mb_y	Memoria (Kbytes)	Memoria (Mbytes)
SQCIF	8	6	36.01	0.04
QAVGA	9	7	40.51	0.04
QCIF	11	9	49.51	0.05
HAVGA	18	14	81.03	0.08
CIF	22	18	99.05	0.10
AVGA	36	28	162.12	0.16
VGA	40	30	180.15	0.18
4CIF	44	36	198.19	0.19
16CIF	88	72	396.77	0.39

TABLA 5.11. Memoria utilizada por la versión v3 del algoritmo de súper-resolución para diferentes tamaños de imágenes.

En la FIGURA 5.59 se muestran los datos de la TABLA 5.11 expresados en kilo-bytes. La condición principal es que los requerimientos de memoria son realmente bajos, y por primera vez podemos hablar de un algoritmo de bajo coste implementable. Las características de bajo coste vienen determinadas, por un lado, por la posibilidad de implementarlo con mínimas modificaciones y como una funcionalidad añadida sobre una plataforma ya existente, y por otro lado, por la posibilidad de hacerlo sin necesitar añadir memoria externa al chip.

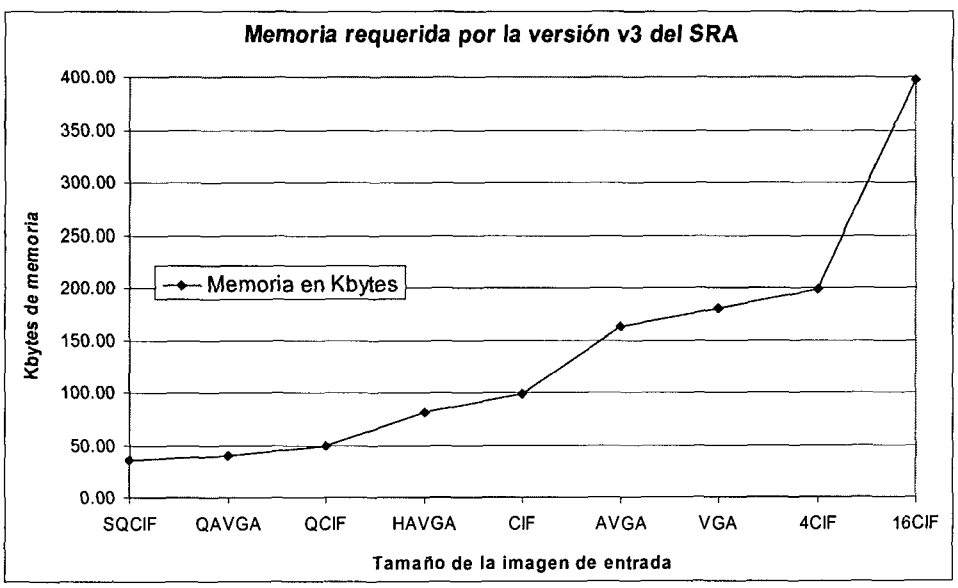


FIGURA 5.59. Memoria usada por el algoritmo de súper-resolución v3 para los tamaños de imágenes más usuales.

La TABLA 5.12 muestra una comparativa de las cantidades de memoria usadas en los algoritmos de súper-resolución más significativos, y en la FIGURA 5.60 se muestra una gráfica comparativa, donde se hace patente la enorme diferencia entre esta última versión del SRA y las anteriores. En promedio, la versión v3 usa 35.63 veces menos memoria que las versiones iterativas (v1.2 y v1.3) y 50.89 veces menos que las versiones no iterativas para imágenes estáticas (v2.0 y v2.1). El precio que hay que pagar por esta drástica reducción de memoria es una ligera disminución de la calidad de las imágenes de súper-resolución obtenidas, frente a las calidades de la versión v2.0 y de la versión v2.1, tal y como se expondrá a continuación.

Tamaño	mb_x	mb_y	v1.2 y v1.3	v2.0	v2.1	v3
SQCIF	8	6	315.19	450.19	459.05	36.01
QAVGA	9	7	413.68	590.87	598.56	40.51
QCIF	11	9	650.07	928.51	931.60	49.51
HAVGA	18	14	1,654.73	2,363.48	2,331.25	81.03
CIF	22	18	2,600.30	3,714.05	3,645.39	99.05
AVGA	36	28	6,618.94	9,453.94	9,198.98	162.12
VGA	40	30	7,879.69	11,254.69	10,936.17	180.15
4CIF	44	36	10,401.19	14,856.19	14,419.55	198.19
16CIF	88	72	41,604.75	59,424.75	57,354.19	396.77

TABLA 5.12. Comparativa de memoria utilizada por las versiones más significativas del algoritmo de súper-resolución para diferentes tamaños de imágenes en Kbytes.

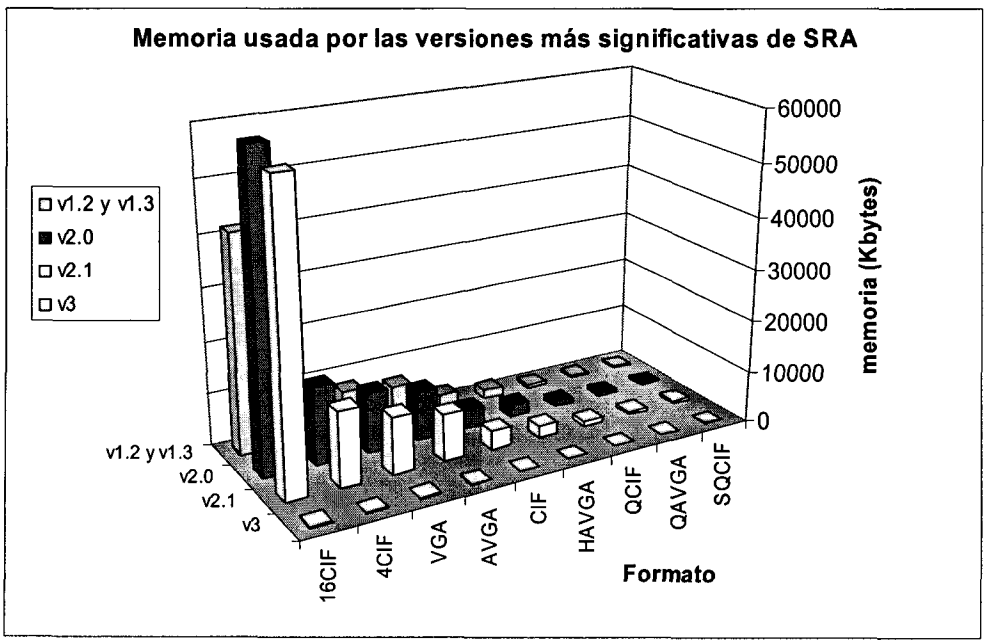


FIGURA 5.60. Comparativa de memoria utilizada por las versiones más significativas del algoritmo de súper-resolución para diferentes tamaños de imágenes en Kbytes.

5.3.4 Implementación sobre la plataforma Picasso modificada

Una vez obtenida una versión del algoritmo para vídeo capaz de funcionar en tiempo real, generando una imagen de súper-resolución por cada *frame* de entrada y usando una cantidad de memoria razonable para su implementación en un solo chip, los requisitos de bajo coste inicialmente especificados pueden alcanzarse al implementar el SRA sobre la plataforma *Picasso*, circuito integrado destinado a producción en volumen.

Aunque el algoritmo de súper-resolución se ha diseñado mapeándolo sobre los recursos que ofrece la arquitectura *Picasso*, resulta conveniente mejorar sus prestaciones realizando modificaciones de los coprocesadores de *Picasso* de forma que manteniendo el soporte arquitectural a otras aplicaciones de codificación incluya ahora un mejor soporte a súper-resolución.

El primer paso has sido definir que coprocesadores, de entre los ya existentes, se modifican para albergar la funcionalidad del ASR. En este sentido se ha decidido modificar el compresor, por ser bloque que tiene mapeada la mayoría de las funciones para acceso a memoria. Se le han añadido dos nuevas funciones: un interpolador para rellenar los huecos antes de su salida al *display*, y una función de *upsample* para pasar los datos de baja a alta resolución. Ambas funciones se han adaptado para funcionar a nivel de bloques DCT. La función IMC (*Inverse Motion Compensation*) se ha modificado para que cuando funcione en modo de súper-resolución realice la acción de SR_UPDATE. El bloque DISP se ha modificado también para inyectar ceros en los bordes, y al ME o estimador de movimiento se le ha aumentado la precisión a ¼ de píxel y se le ha disminuido la preponderancia de uso del vector cero. La FIGURA 5.61 recoge estos cambios, resaltados en negrita y línea discontinua.

En consecuencia se ha tenido que acometer una nueva planificación y temporización. La arquitectura *Picasso* funciona según un complicado esquema de segmentación (FIGURA 3.9) para aumentar sus prestaciones. La segmentación del flujo de datos no ha sido tratada durante la concepción y mejora del SRA para poder centrarnos en el proceso clave del propio algoritmo. En este punto, el *pipeline* debe ser de nuevo introducido y modificado, tanto para ajustarse al esquema de lectura y salida de datos del sistema, como para optimizar la propia ejecución del SRA. El nuevo *pipeline* es el que se recoge en la FIGURA 5.62.

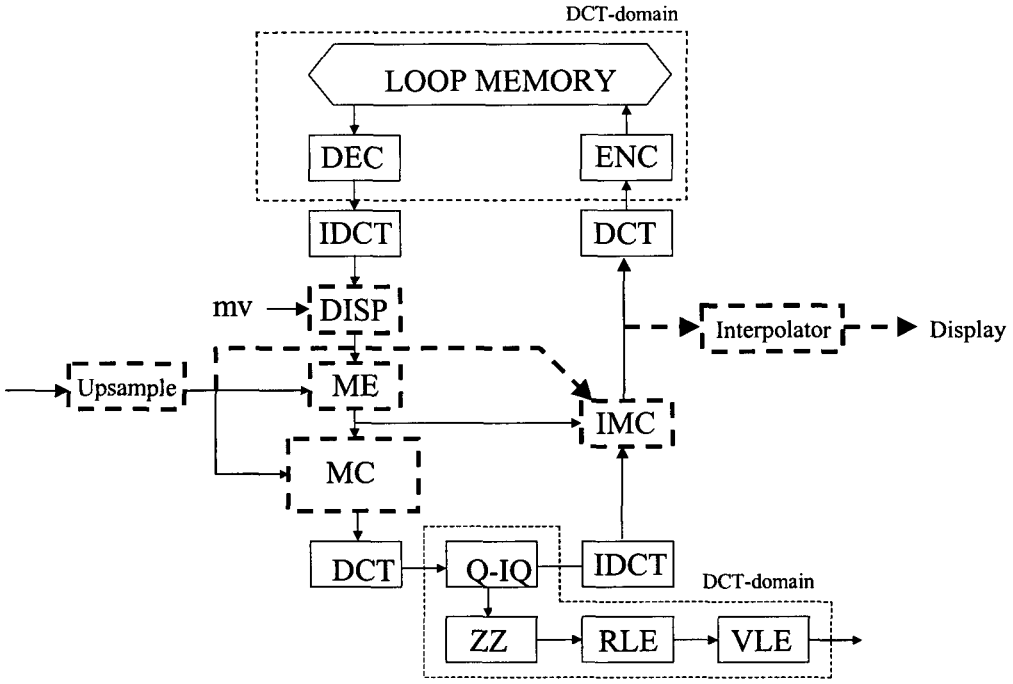


FIGURA 5.61. Modificaciones realizadas en el compresor de *Picasso* para optimizar el soporte a la versión v3 del SRA

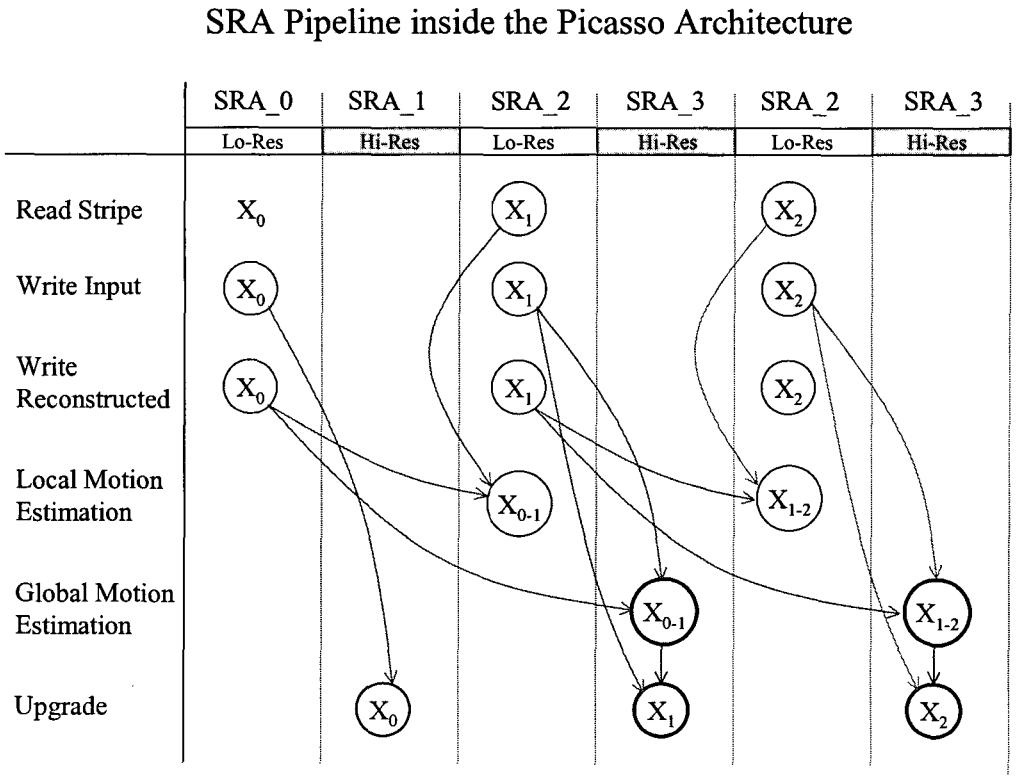


FIGURA 5.62. Esquema de segmentación seguido por el SRA sobre *Picasso* modificado.

Este esquema de segmentación sigue, de forma simplificada, el diagrama de estados o de flujo de control de la FIGURA 5.63.

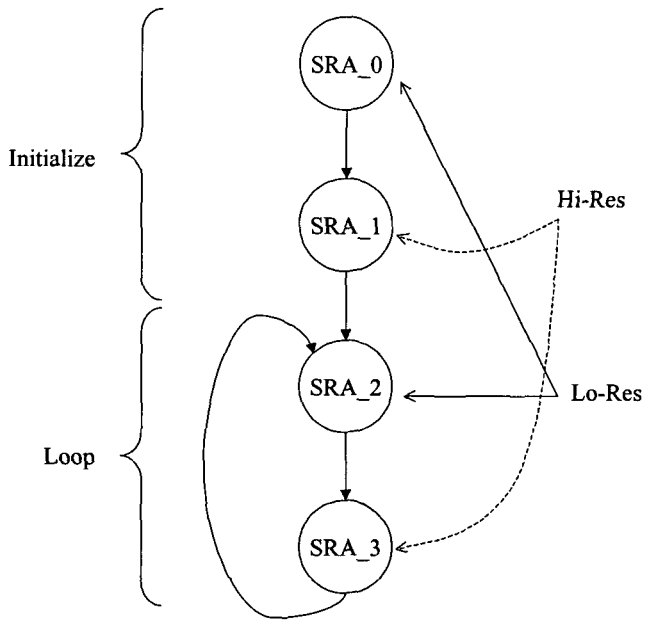


FIGURA 5.63. Diagrama de estados básico, o flujo de control, seguido en la segmentación.

Existe una etapa de inicialización formada por los estados SRA_0 y SRA_1, el primero de baja resolución y el segundo de alta resolución, y luego ya se continúa en un bucle formado por los estados SRA_2 y SRA_3, al igual que antes, el primero de baja resolución y el segundo de alta resolución. Cada estado está caracterizado por una serie de tareas, algunas de las cuales se realizan en HR y otras en LR. Debe tenerse en cuenta que cada tarea hace uso de unos determinados recursos, y que pueden existir conflictos entre ellos. Asimismo, existe un orden lógico de precedencia: por ejemplo, no puede mejorarse un macro-bloque si todavía no ha sido leído.

En el estado SRA_0, se lee la imagen X0 (por franjas) y se almacena en el buffer de entrada. Desde allí se escribe en la memoria de bucle, donde comenzará a usarse.

En el estado SRA_1 lo único que debe efectuarse es una interpolación para mostrar la imagen sin huecos y almacenarla en memoria de bucle con huecos.

En el estado SRA_2 se realizan las mismas tareas que en el SRA_0, pero sobre la siguiente imagen (Xi) y además se lleva a cabo la primera estimación de movimiento (entre Xi-1 y Xi) destinada a obtener los vectores de movimiento globales.

En el estado SRA_3 se hace la segunda estimación de movimiento pero usando el vector de movimiento global, que ya se ha podido estimar al tener disponibles los vectores de movimiento locales. Con toda la información disponible, se toma la decisión de interpolar o de aplicar las mejoras de súper-resolución sobre la imagen Xi, que se mostrará en salida sin huecos y se almacenará en la memoria de bucle con huecos.

Las contribuciones son generadas a partir de las propias imágenes, siguiendo la siguiente estrategia: en *Picasso* no está permitido el valor cero de píxel, y si en la imagen de entrada el subsistema de entrada lo entregase, este sería recortado a uno[3]. Por lo tanto, usaremos este valor para indicar la existencia de un hueco en la imagen. Sabiendo donde están los huecos, las contribuciones pueden ser perfectamente generadas a partir de los valores de píxel de la propia imagen siguiendo el sencillo procedimiento de asignar un valor de 4 (valor máximo de contribución) al píxel que existe y cero al valor ausente.

5.3.5Calidad de imagen obtenida

Para evaluar la calidad de imagen que el algoritmo de súper-resolución es capaz de proporcionar, se han realizado varias pruebas, que pueden clasificarse en dos tipos:

a) **Pruebas con secuencias retocadas**. En este caso se utilizan imágenes reales a las que se les aplican desplazamientos de forma artificial. Tienen la ventaja de ser perfectamente conocidos y su comportamiento medible, al tener una referencia disponible.

b) **Pruebas con secuencias reales**. Aunque son el tipo de secuencias con las que el sistema realmente se va a enfrentar, tienen el inconveniente de que con ellas no se pueden extraer medidas cuantitativas de calidad. No obstante si que es posible realizar observaciones cualitativas.

Dentro del tipo de pruebas con secuencias retocadas, se ha utilizado la secuencia KRANT con cambio de contexto cuyos vectores de movimiento se recogen en la Tabla 5.8. En la FIGURA 5.64 se muestra la PSNR alcanzada por la señal de luminancia, que es la parte que más afecta a la calidad perceptual. La calidad de todos los *frames* es bastante superior a las calidades de interpolación (1.44 dB de mejora como mínimo con respecto a la interpolación bilineal y 5.54 dB de mejora máxima). La calidad promedio es de 28.64 dB, frente a los 25.3

[3] En realidad el valor 255 también es recortado a 254.

dB promedio de la interpolación bilineal. Este promedio de 28.64 dB es algo inferior al de 30.47 dB alcanzado por la versión v2.0, por lo que ya a nivel de sistema, aplicación y comercialización del producto debe estudiarse si una disminución de 1.83 dB de calidad está compensada por la disminución de 50.89 veces el tamaño de la memoria integrada requerida por esta versión v3.

La FIGURA 5.65 muestra que el efecto de borde prácticamente no afecta a esta versión del algoritmo y que, como era de esperar, las crominancias exhiben una mejor PSNR que la luminancia debido a su menor entropía.

En la FIGURA 5.66 se muestran los coeficientes de correlación espectral en módulo. Los *frames* 8 y 9 descienden hasta la calidad de interpolación debido al cambio de contexto. El caso del *frame* 8 es debido a que se detecta el cambio de contexto y la imagen es simplemente interpolada. El caso del *frame* 9 se debe a que los vectores de movimiento no han sido calculados aún con la precisión suficiente y se ha llegado a una calidad similar a la de interpolación. En el caso del cálculo de la PSNR, el *frame* 8 no llega a descender al nivel de interpolación debido a la estrategia diferente de interpolación por macro-bloques seguida en esta versión del algoritmo.

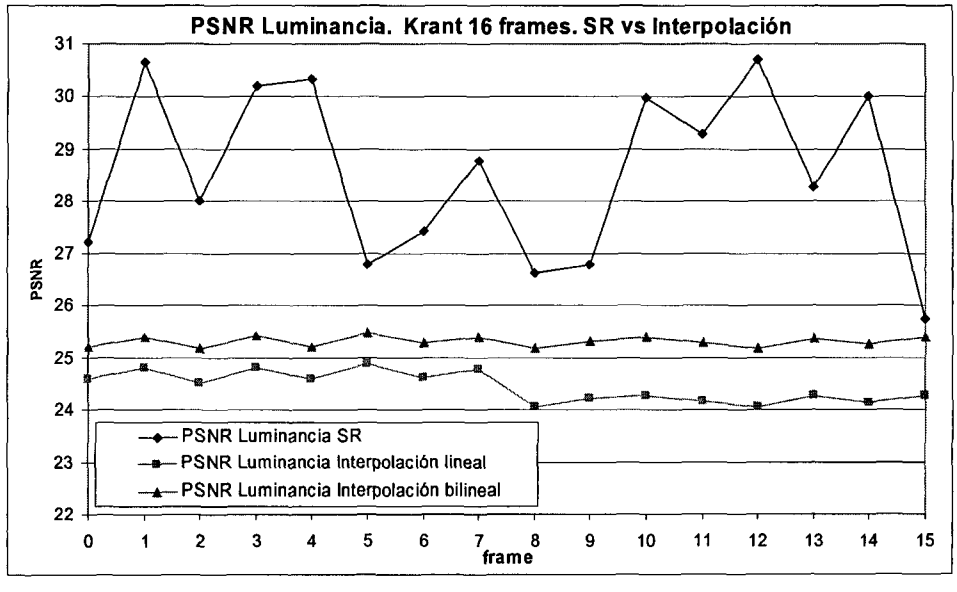


FIGURA 5.64. PSNR de la luminancia de la secuencia Krant de 16 *frames* con cambio de contexto en el *frame* número 8.

En la FIGURA 5.67 se muestran los SCC en módulo tanto de la luminancia como de las crominancias con y sin bordes. Al igual que en la PSNR los valores de crominancia exhiben mejores comportamientos, y las variaciones con los bordes son mínimas antes del cambio de

contexto y ligeramente superiores después. En fase los comportamientos obtenidos son igualmente similares.

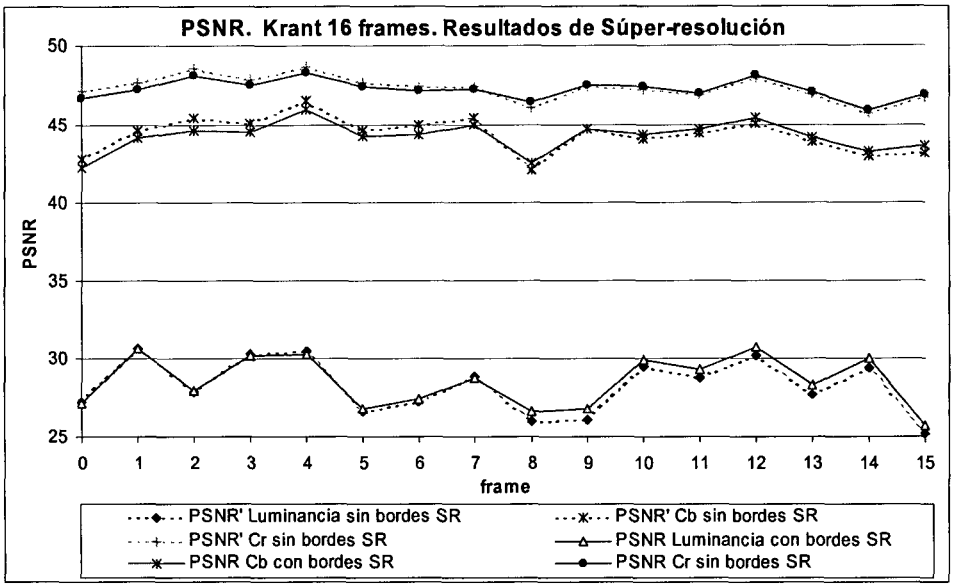


FIGURA 5.65. PSNR de la secuencia Krant de 16 *frames* con cambio de contexto en el *frame* número 8 con y sin bordes.

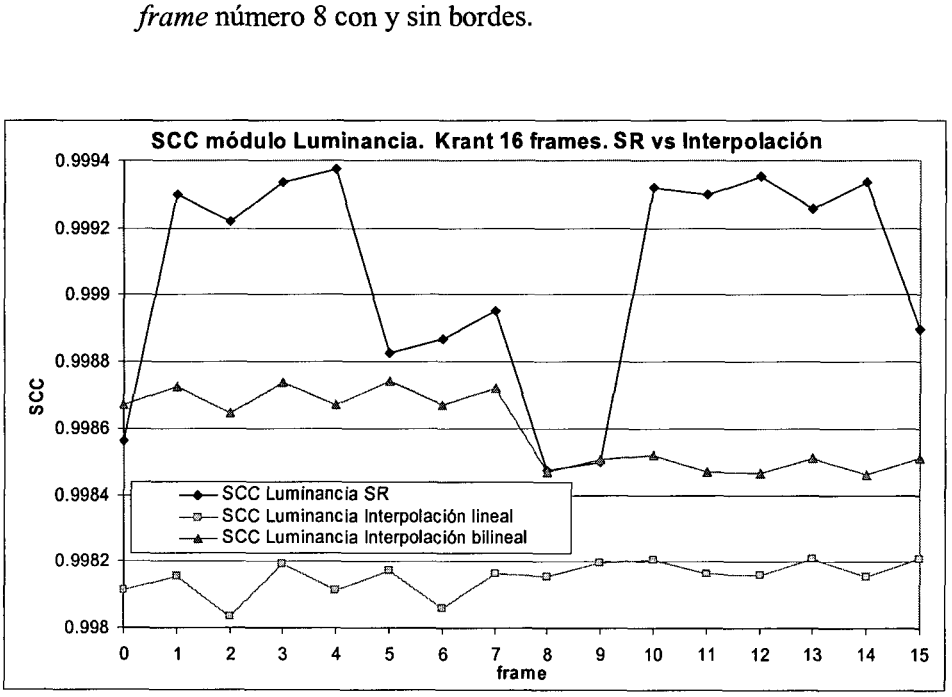


FIGURA 5.66. SCC en módulo de la luminancia de la secuencia Krant de 16 *frames* con cambio de contexto en el *frame* número 8.

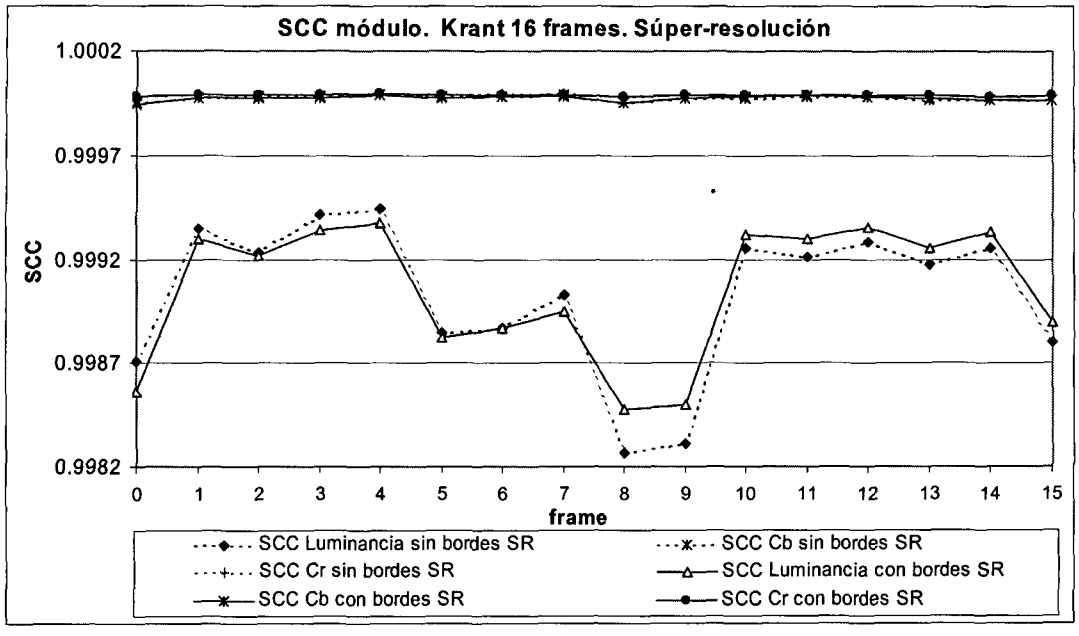


FIGURA 5.67. SCC en módulo de la secuencia Krant de 16 *frames* con cambio de contexto en el *frame* número 8 con y sin bordes.

Con respecto a la aplicación de la versión v3 a secuencias reales, se ha diseñado el siguiente experimento. La frase 'Let's make things better', se ha dispuesto en líneas y columnas con diferentes tamaños de letra, indicando su correspondiente tamaño a la izquierda, en orientaciones vertical y horizontal (FIGURA 5.68). Este texto ha sido grabado con una cámara de baja calidad (cámara para PC ToUCam Vesta PCVC675K de Philips) a mano alzada (a pulso), lo que provoca los desplazamientos necesarios para poder aumentar la resolución de las imágenes. Se han realizado tomas a diferentes distancias del texto.

En este caso no se provoca *aliasing* de forma artificial, ni se aplican desplazamientos conocidos. Sin embargo, es muy difícil conseguir eliminar completamente todo el *aliasing* de la imagen de entrada, y precisamente el SRA persigue aumentar la calidad eliminando esos restos de *aliasing*, sobre todo en los tamaños de letra más pequeños.

En la FIGURA 5.69 (a) se puede ver el *frame* 5 de la secuencia de entrada y en la FIGURA 5.69 (b) la imagen de súper-resolución obtenida. En general se aprecia una recuperación de los bordes de las letras y una disminución del ruido de fondo, lo que era de esperar al provenir del promedio de varias imágenes consecutivas.

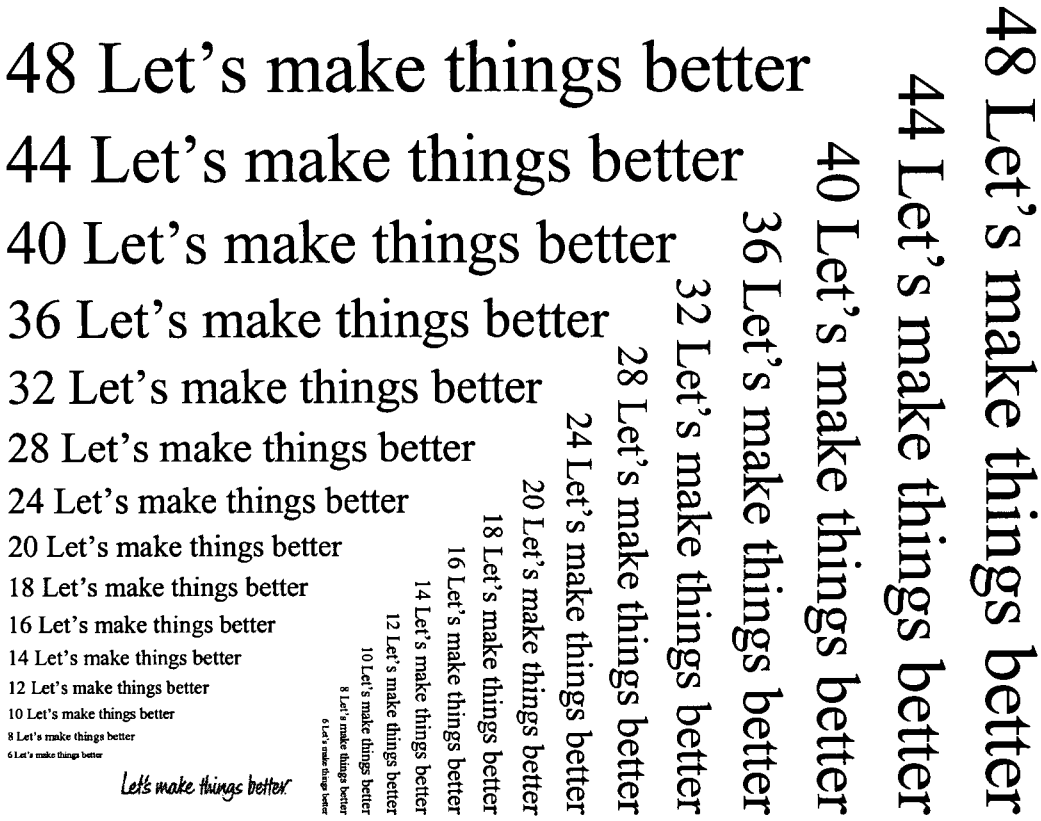
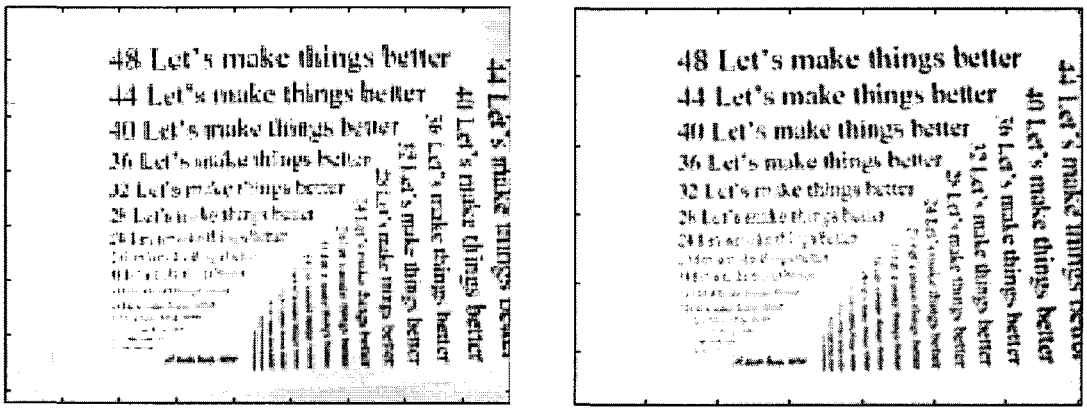


FIGURA 5.68. Texto de prueba para secuencias reales.

Un aspecto donde se hacen patente las mejoras de súper-resolución es en el tamaño de letra que puede ser leído con claridad en uno y otro caso. En la FIGURA 5.70 se muestra una ampliación de las imágenes de la FIGURA 5.69.

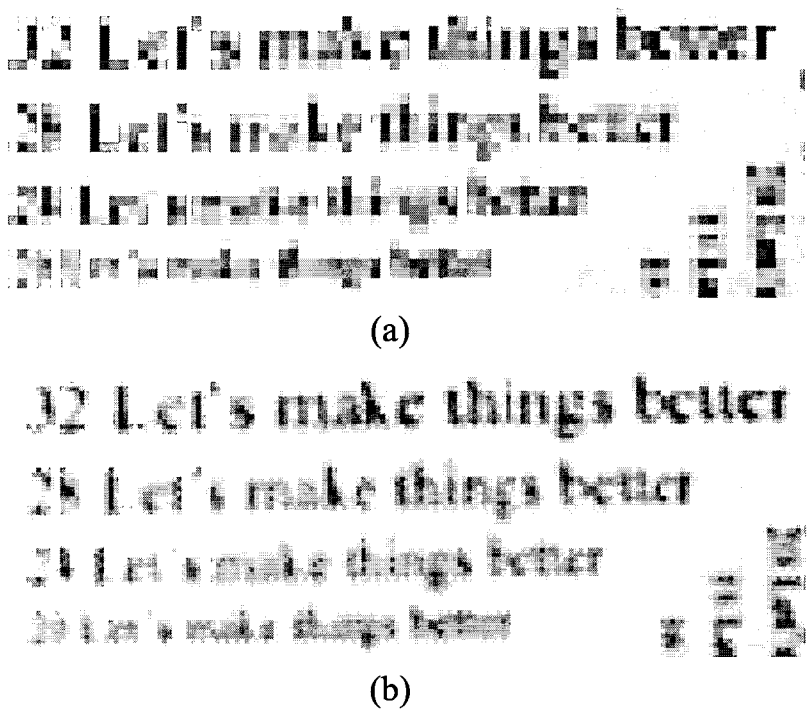


(a) (b)

FIGURA 5.69. *Frame* 5 de la secuencia de entrada (a) al SRA y de la secuencia de salida de súper-resolución (b).

Es patente cómo en la imagen de entrada (a) la lectura de las letras de tamaño 28 puntos es dificultosa, la de tamaño 24 dudosa y la de tamaño 20 imposible. Sin embargo, en la imagen de súper-resolución estos tamaños de letras son mucho más fáciles de leer, si bien en el caso de la letra de 20 puntos el resultado es todavía borroso.

También resulta interesante comparar estos resultados con los de la FIGURA 5.71, donde se ha aplicado a la imagen otros tratamientos clásicos alternativos como son los de filtrado paso bajo (a), y el de realce de los bordes (b). En ninguno de estos casos se mejora tanto la legibilidad como en el caso de la aplicación del algoritmo de súper-resolución. Este resultado es lógico, en tanto en cuanto el ASR incorpora nueva información proveniente de *frames* anteriores a la imagen actual, mientras que los procesos de reconstrucción clásicos sólo cuentan con la información del *frame* actual.



(a)

(b)

FIGURA 5.70. Detalles del *frame* 5 de la secuencia de entrada (a) al SRA y de la secuencia de salida de súper-resolución (b).

En la FIGURA 5.72 (a) se muestra el *frame* 7 de otra secuencia de prueba, pero esta vez acercando más la cámara al texto y el resultado tras aplicar el algoritmo de súper-resolución (b). En el detalle de la FIGURA 5.73 se ve claramente la mejora de legibilidad al aplicar el ASR sobre (a) para obtener (b).

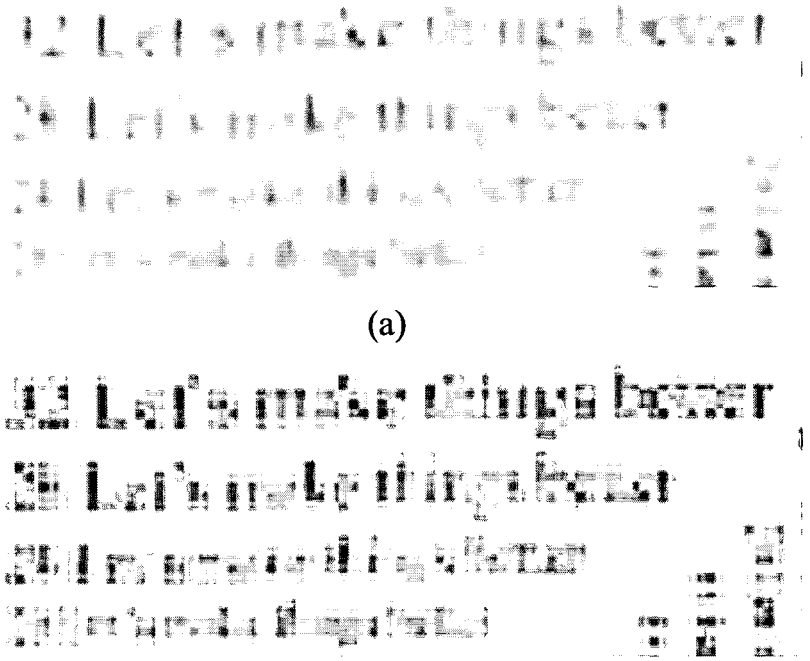
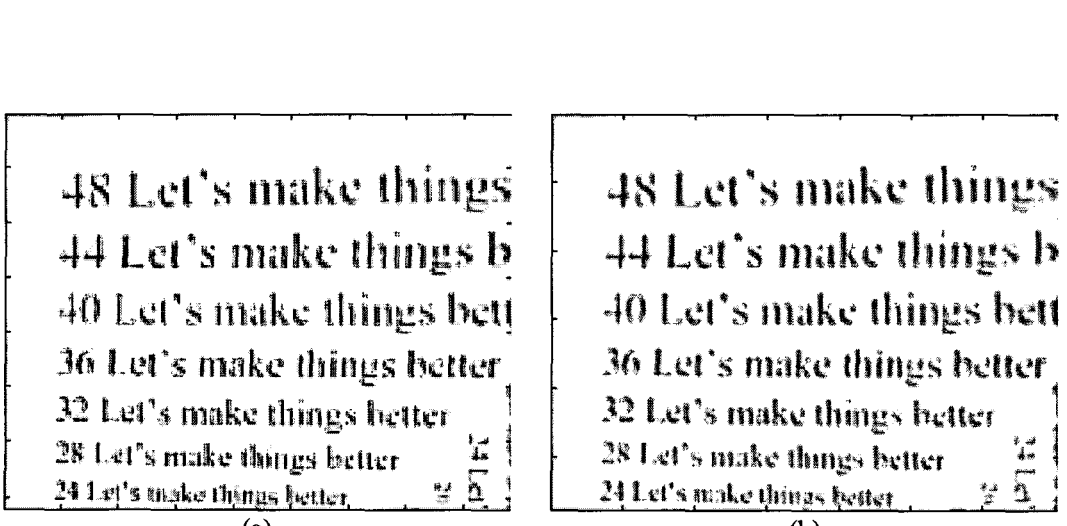
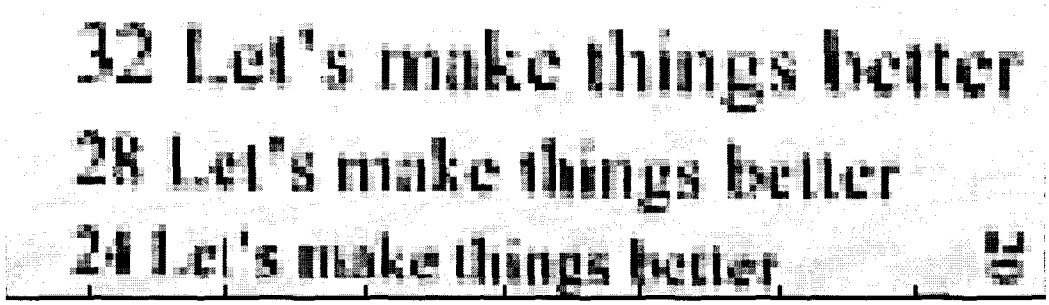


FIGURA 5.71. Detalles del *frame* 5 de la secuencia de entrada (a) al SRA y de la secuencia de salida de súper-resolución (b).

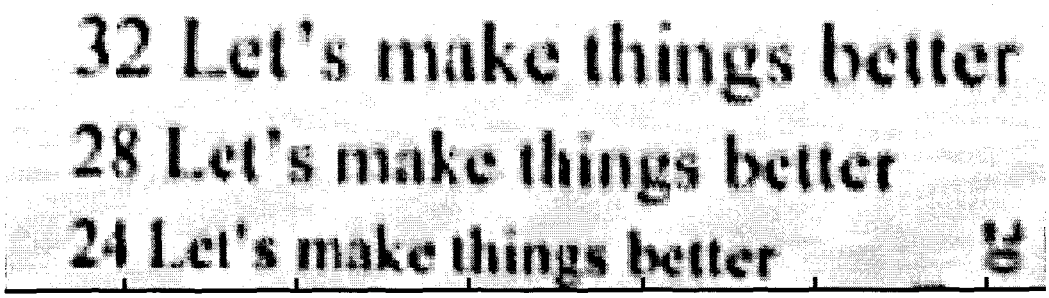


(a) (b)

FIGURA 5.72. *Frame* 7 de la secuencia de entrada (a) al SRA y de la secuencia de salida de súper-resolución (b).



(a)



(b)

FIGURA 5.73. Detalles del *frame* 7 de la secuencia de entrada (a) al SRA y de la secuencia de salida de súper-resolución (b).

5.4 Conclusiones

En este capítulo se han desarrollado dos tipos de algoritmos de súper-resolución no iterativos y aptos para trabajar en tiempo real usando los recursos ofrecidos por un codificador híbrido de vídeo desarrollado en Philips Research y al que se han aportado modificaciones arquitecturales. El primer tipo de algoritmos está más orientado a su uso en imágenes estáticas, aunque con un esquema de segmentación adecuado también puede usarse para vídeo. Aunque la calidad de imagen que ofrece es bastante alta, cuentan con el serio inconveniente de requerir mucha memoria, dificultando la consecución del objetivo trazado de bajo coste.

El segundo tipo de algoritmos está pensado para vídeo y basa su funcionamiento en la realimentación de la última imagen de súper-resolución obtenida. Dado que no recopila y acumula la información proveniente de varias imágenes, los píxeles sufren un efecto de dilución temporal que provoca una ligera pérdida de calidad (<2 dB) en comparación con la versión anterior. En todo caso, la disminución del tamaño de la memoria requerida puede

llegar a compensar en muchos casos, dependiendo del tipo de aplicaciones y de los parámetros del mercado, la pérdida de calidad sufrida.

En todos los algoritmos estudiados en este capítulo se constata una independencia de las métricas obtenidas con el efecto de borde, pudiendo asegurar que este tipo de algoritmos se ven afectados mínimamente por este indeseado efecto. Esto se debe al uso de las contribuciones, que ya tienen en cuenta la posibilidad de salirse del borde real de la imagen. Al rellenar el borde con la información de otras imágenes y con ceros, se minimiza el efecto de borde en la calidad de la imagen final.

Podemos resumir, de forma aproximada, los resultados de este capítulo en los valores promedio mostrados en la TABLA 5.14 y TABLA 5.13.

Tamaño	mb_x	mb_y	v1.2 y v1.3	v2.0	v2.1	v3
SQCIF	8	6	315.19	450.19	459.05	36.01
QAVGA	9	7	413.68	590.87	598.56	40.51
QCIF	11	9	650.07	928.51	931.60	49.51
HAVGA	18	14	1,654.73	2,363.48	2,331.25	81.03
CIF	22	18	2,600.30	3,714.05	3,645.39	99.05
AVGA	36	28	6,618.94	9,453.94	9,198.98	162.12
VGA	40	30	7,879.69	11,254.69	10,936.17	180.15
4CIF	44	36	10,401.19	14,856.19	14,419.55	198.19
16CIF	88	72	41,604.75	59,424.75	57,354.19	396.77

TABLA 5.14. Comparativa de memoria utilizada por las versiones más significativas del algoritmo de súper-resolución para diferentes tamaños de imágenes en Kbytes.

	PSNR media luminancia	PSNR media crominancia roja	PSNR media crominancia azul
V2.0	30.4701 dB	45.4279 dB	48.7716 dB
V2.1	34.6331 dB	46.9091 dB	49.6631 dB
V3	28.6415 dB	44.2202 dB	47.2280 dB

TABLA 5.13. PSNR media para las versiones v2.0, v2.1, y v3 usando precisión de ¼ de píxel en el estimador de movimiento.

Capítulo 6

¡Acabado!, ¡que estúpida palabra ! ¿Por que acabado? Lo acabado y la pura nada son exactamente lo mismo. ¿Para que nos sirve el eterno crear? Para que lo creado se disipe en la nada. ¿Que se puede decir de algo si se ha acabado? Que es como si no hubiese existido y sin embargo circulara como si existiera. En lugar de ello, preferiría el vacío eterno.

Johann Wolfgang Göethe
"Fausto".

Pero lo que ahora importa no son las esperanzas ni los temores, sino solamente la verdad, en la medida que nuestra razón nos permita desvelarla.

Charles Darwing
"El origen del hombre"

Conclusiones y Líneas Futuras

6.1 Conclusiones

Tras estudiar y clasificar los diferentes algoritmos de súper-resolución existentes en la literatura se han creado nuevas versiones iterativas optimizadas y por primera vez versiones no iterativas de súper-resolución para súper-resolución dinámica sin restricciones en la señal de entrada. Estas versiones se han implementado sobre la plataforma para la codificación híbrida de vídeo *Picasso* desarrollada por Philips Nat.Lab para alcanzar prestaciones de tiempo real y bajo coste.

Esta plataforma supone un potente sistema de codificación híbrida de vídeo con las siguientes características:

- Implementa el núcleo básico de compresión para algoritmos basados en la transformada discreta del coseno, DCT.
- Es un sistema abierto a la implementación de nuevos estándares de compresión de vídeo e imagen estática.
- El particionado se ha realizado de forma que se obtienen muy altas prestaciones.

- La compresión de las memorias en el dominio de la DCT permite usar sólo memorias empotradas on-chip. Al evitar el uso de memorias externas se consigue una importante disminución de la potencia disipada al tiempo que se minimizan los costes a un sólo circuito integrado.
- La metodología de diseño permite implementar y evaluar con sencillez nuevas aplicaciones sobre esta plataforma.

La plataforma *Picasso* es capaz de comprimir secuencias de vídeo siguiendo el estándar H.263 e implementando prácticamente todas las opciones del estándar H.263+. Así mismo, está preparada para dar soporte al núcleo principal de compresión del estándar MPEG-4. También puede comprimir imágenes estáticas en formato JPEG y se han desarrollado librerías que permiten su funcionamiento multi-hilo, lo que permite ejecutar varias de las aplicaciones anteriormente citadas al mismo tiempo siguiendo un esquema de compartición de recursos en el dominio del tiempo.

El primer conjunto de algoritmos aportado en la tesis está formado por los algoritmos iterativos para súper-resolución estática, basados en [BK99] y constituyen un primer paso hacía el objetivo trazado de conseguir algoritmos de súper-resolución de bajo coste y tiempo real. Al haber conseguido implementar este tipo de algoritmos sobre una plataforma ya existente para la compresión híbrida de video, realizando las modificaciones necesarias en el *hardware*, podemos considerar alcanzado los objetivos trazados de bajo coste, pero no así los de tiempo real. En este sentido, se ha creado un nuevo esquema algorítmico no iterativo, basado en el concepto de pesos contributivos, capaz de obtener mejoras de súper-resolución en un solo paso. Reutilizando además la última imagen de alta resolución obtenida, conseguimos mantener el tamaño de la memoria dentro de los límites que permiten su alojamiento *on-chip*.

El primer conjunto de algoritmos son válidos para lo que denominamos súper-resolución estática, es decir, la generación de una sola imagen de alta resolución mediante la combinación de varias imágenes de baja resolución. En la FIGURA 6.1 se muestra el esquema de generación de imágenes de súper-resolución seguido por este tipo de algoritmos. Las versiones v1.0, v1.1, v1.2 y v1.3 pertenecen a la categoría de algoritmos iterativos para súper-resolución estática. Para las secuencias de prueba usadas las calidades medias de luminancia obtenidas, medidas como la relación señal a ruido de pico, fueron de 23.19 dB para la versión v1.2 y de 28.24 dB para la versión v1.3, al ser éstas las versiones más significativas de entre los algoritmos iterativos.

La calidad de las imágenes de súper-resolución depende fuertemente del muestreo realizado sobre las imágenes de entrada. Si el conjunto de imágenes usadas para generar la imagen de súper-resolución tiene muestras comunes, el proceso de súper-resolución se ve privado de información nueva, generando imágenes de calidad menor a las que se podrían obtener si se dispusiese de imágenes donde cada una aporta información diferente.

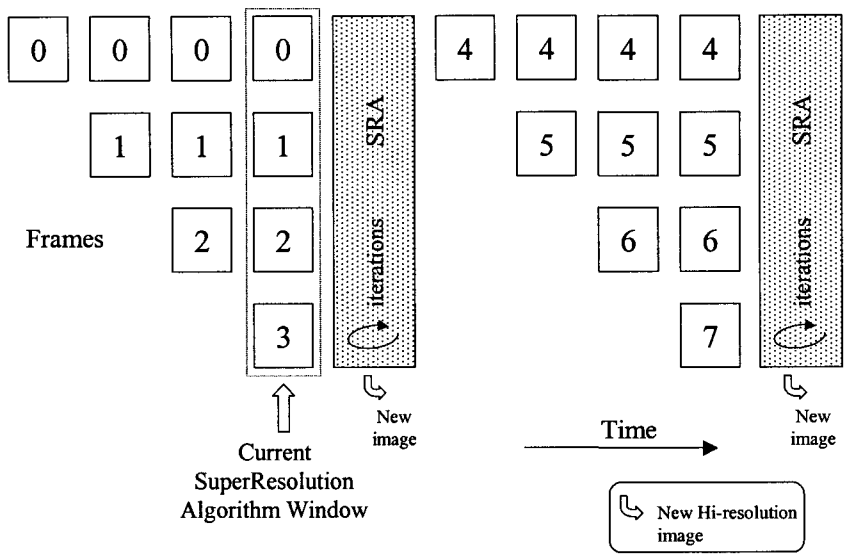


FIGURA 6.1. Esquema de generación de imágenes de súper-resolución estática en los algoritmos iterativos.

El método usado para la estimación de movimiento no tiene una influencia apreciable sobre la calidad de la imagen final en este tipo de algoritmos, debido a que los vectores de movimiento obtenidos son sólo una aproximación a los movimientos reales de la imagen. Así pues, no resulta en una mejor importante de calidad el usar un método de estimación de movimiento de búsqueda exhaustiva de alto coste computacional frente a un método mucho más rápido como es la búsqueda jerárquica en tres dimensiones, ya implementado en la arquitectura *Picasso*.

Sin embargo, sí resulta decisivo el uso de filtros paso bajo en la imagen de entrada antes de la estimación de movimiento, alcanzando de esta forma mejoras apreciables en la calidad y en la estabilidad de la calidad, lo que mejora la visión de la secuencia.

El segundo conjunto de algoritmos aportado en la tesis son los no iterativos, aptos para trabajar en tiempo real. Dentro de los algoritmos no iterativos, podemos realizar una sub-clasificación en algoritmos para súper-resolución estática y para súper-resolución dinámica. En la FIGURA 6.2.a se muestra el esquema de generación de imágenes de los algoritmos no

iterativos de súper-resolución estática. Aunque la calidad de imagen que ofrece es bastante alta, cuentan con el serio inconveniente de requerir mucha memoria, dificultando la consecución del objetivo trazado de bajo coste. El segundo tipo de algoritmos (FIGURA 6.2.b) está pensado para vídeo (súper-resolución dinámica) y basa su funcionamiento en la realimentación de la última imagen de súper-resolución obtenida.

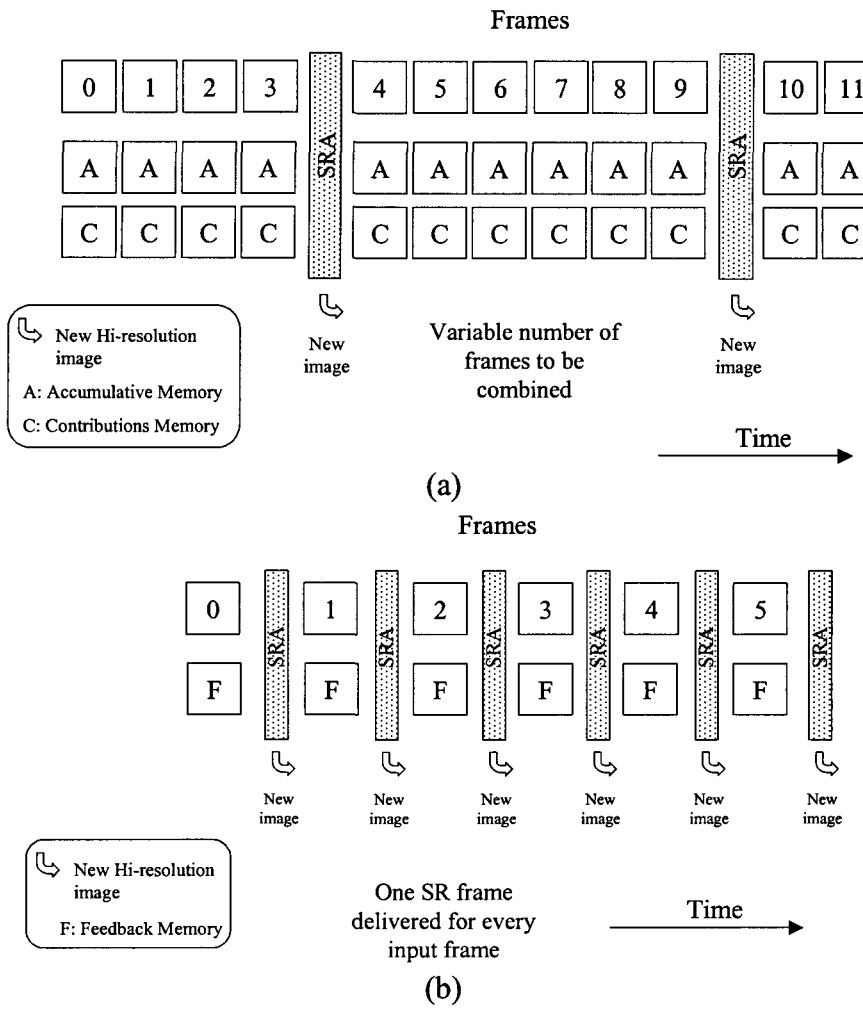


FIGURA 6.2. Esquema de generación de imágenes de súper-resolución en los algoritmos no iterativos. (a) esquema para súper-resolución estática y (b) esquema para súper-resolución dinámica.

Durante la fase de evaluación de las calidades de las imágenes obtenidas se hizo patente la dificultad que los bordes ofrecen en el procesamiento, ya que rompen la homogeneidad del algoritmo. Este problema se soluciona desde dos perspectivas diferentes. En primer lugar, se mejora el procedimiento de creación de imágenes de prueba, para evitar la aparición de artefactos indeseados en los bordes al mover la imagen. Esto se consigue partiendo de tamaños de imagen mayores, realizando sobre ellos los desplazamiento y luego recortando la zona central. En segundo lugar, mediante ajustes algorítmicos se logra

incorporar en los pesos contributivos el tratamiento de los bordes. Para lograrlo, se introducen ceros en los bordes de las imágenes desplazadas que luego serán rellenados con nuevos datos o serán interpolados usando los existentes.

Mediante la reutilización de los datos de la imagen anterior de súper-resolución en la versión v3 conseguimos disminuir los requerimientos de memoria en un factor 50 veces inferior a los de la versión v2.0 y v2.1, y 35 veces inferior a los de las versiones v1.2 y v1.3. Dado que esta última versión no recopila y acumula la información proveniente de varias imágenes, los píxeles sufren un efecto de dilución temporal que provoca una pérdida de calidad en comparación con las versiones anteriores. No obstante, experimentalmente la pérdida de calidad con respecto a las versiones anteriores ha demostrado ser inferior a 2 dB.

En la TABLA 6.1 se presenta un resumen de la calidad media obtenido por las versiones más significativas de los algoritmos de súper-resolución. La versión no iterativa para súper-resolución dinámica tiene calidades ligeramente superiores a las de las versiones iterativas, pero con un uso de memoria muy inferior al del resto de las versiones del algoritmo.

Versión	Tipo	PSNR media luminancia	PSNR media crominancia roja	PSNR media crominancia azul
v1.2	Iterativo. SR estática	23.1947 dB	38.4512 dB	40.7958 dB
v1.3	Iterativo. SR estática	28.2419 dB	38.2452 dB	39.8810 dB
v2.0	No iterativo. SR estática	30.4701 dB	45.4279 dB	48.7716 dB
v2.1	No iterativo. SR estática	34.6331 dB	46.9091 dB	49.6631 dB
v3	No iterativo. SR dinámica	28.6415 dB	44.2202 dB	47.2280 dB

TABLA 6.1. PSNR media para las versiones más significativas de los algoritmos de súper-resolución.

Todos los algoritmos presentados se implementan sobre la plataforma integrada *Picasso*, habiéndose tenido que modificar algunos de sus componentes, pero reutilizando su inmensa mayoría. De esta forma se ha añadido la capacidad de realizar mejoras de súper-resolución a la ya existente codificación de vídeo con un coste prácticamente nulo.

Se ha demostrado cualitativa y cuantitativamente la mejora de la calidad en imagen estática y vídeo mediante técnicas de súper-resolución con prestaciones de tiempo real y bajo coste, usando un codificador híbrido de vídeo.

6.2 Líneas Futuras

El trabajo desarrollado en esta tesis ofrece la posibilidad de continuar con las siguientes líneas de investigación:

- Análisis de la influencia del subsistema de entrada en la calidad de las imágenes obtenidas. Las lentes usadas por los sistemas de adquisición de imágenes conllevan la presencia de un filtro óptico paso bajo (OLP, *Optical Low-pass Filter*). Este filtro evita el *aliasing* en la secuencia de entrada, pero elimina también muchos detalles de alta frecuencia necesarios para mejorar la resolución de las imágenes usando súper-resolución. Sería necesario estudiar la posibilidad de usar OLPs de gran ancho de banda con una etapa de filtrado digital posterior, de tal forma que la secuencia filtrada se pueda usar para compresión, y sin filtrar, para súper-resolución.
- Determinación dinámica de los umbrales de decisión. Los umbrales de decisión acerca de si se debe aplicar súper-resolución con el vector global, con vectores locales o interpolar, podrían irse modificando de tal forma que se adaptasen a las características estadísticas de las imágenes de entrada. En este sentido, se podría diseñar un sistema adaptativo que modificase de forma continua y automática estos umbrales.
- Estudio de aceleradores *hardware* específicos. La realización de alguna función crítica, como la combinación de los píxeles o la interpolación de los huecos, podría realizarse en un coprocesador dedicado sólo a mejoras de súper-resolución. Habría que determinar si el aumento de velocidad en los bloques modificados se refleja realmente en un aumento de las prestaciones del sistema y si este aumento justifica el incremento de coste global.
- Uso de la súper-resolución en codificación. Una vez obtenida la secuencia de súper-resolución dentro del codificador hay que determinar que hacer con ella. Las posibilidades son varias: enviarlas a una pantalla o visualizador, guardarlas en algún sistema de almacenamiento masivo (típicamente discos duros o cintas) o codificarlas para ahorrar ancho de banda en transmisión y espacio de almacenamiento. Sin embargo, debemos tener en cuenta que gran parte de los recursos de codificación están siendo utilizados por el proceso de mejora de la calidad, por lo que para codificar la imagen habría que almacenarla primero, cambiar la función de súper-resolución a codificación, codificar la imagen y volver a cambiar a súper-resolución. Otra opción sería usar procedimientos

multi-hilo, que permiten compartir recursos en distintas rodajas de tiempo, planificado desde un sistema operativo en tiempo real. Ambas soluciones son costosas en tiempo, por lo que se podría optar por una tercera vía consistente en reutilizar parte de la información obtenida durante el proceso de súper-resolución en la codificación. En concreto, las funciones de DCT, cuantificación y codificación de longitud variable no son usadas en súper-resolución. Si además reutilizamos los vectores de movimiento obtenidos en la etapa de estimación de movimiento de los vectores locales, tenemos toda la información y todos los recursos necesarios para codificar imágenes tipo I y tipo P. Habría que estudiar la viabilidad en cuanto a su implementación y como afectaría el proceso de codificación a la calidad ganada en el proceso de súper-resolución. No podemos olvidar que la súper-resolución obtiene importantes mejoras en las zonas de altas frecuencias, que repercuten en la mejor apreciación de los detalles de la imagen, y que una de las bases de la compresión con pérdidas es precisamente la pérdida de los datos de alta frecuencia mediante la cuantificación de los coeficientes DCT. Por lo tanto, es previsible que las imágenes tipo I se vean más afectadas que las de tipo P, ya que éstas últimas transmiten además el error entre imágenes, lo que permite salvar algo más de información de alta frecuencia.

- Uso de la súper-resolución en decodificación. Las imágenes decodificadas han sido varias veces filtradas paso bajo (por los filtros del bucle y por la propia cuantificación de los coeficientes DCT), por lo que difícilmente se podrían obtener mejoras de súper-resolución a través de ellas. Pero tal vez se podrían sub-muestrear antes de ser enviadas (habría que estudiar como sub-muestrear la cadena binaria de salida, no las imágenes de entrada al codificador) y aumentar luego la calidad en el decodificador mediante la combinación de varias imágenes de entrada de menor resolución. Esta sería una forma de obtener una reducción adicional del tamaño de la cadena binaria transmitida.

Referencias

Yo no cito a los demás más que para mejor expresar mi pensamiento.

Michel Eyquem de Montaigne (1533-1592)
Escritor francés.

Y cuanto Más sabio fue el Predicador, tanto Más Enseñó Sabiduría al pueblo. También Sopesó, Investigó y compuso muchos proverbios.

Eclesiastés 12.9.

[Ack93] B.D. ACKLAND, "A video-codec chip set for multimedia applications," AT&T Technical Journal, pp. 50-65, January 1993.

[Ack94] B.D. ACKLAND, "The role of VLSI in multimedia," IEEE Journal of Solid State Circuits, volume 29, number 4, 1994.

[Ack95] B.D. ACKLAND, "VLSI for multimedia applications," Proc. of 13th Australian Microelectronics Conference, pp. 23-33, Adelaida, Australia del Sur, The IREE Society, July 1995.

[AD97] V. AVRIN AND I. DINSTEIN, "Restoration and resolution enhancement of video sequences," ICASSP IV, pp. 549-553, 1997

[Aki94] T. AKIYAMA, "MPEG2 video codec using image compression DSP," IEEE Trans. on Consumer Electronics 40(3), 1994.

[Aon92] K. AONO, "A video digital signal processor with a vector-pipeline architecture," IEEE Jornal of Solid-State Circuits, 27(12), 1992.

[Ara94] T. ARAKI, "Video DSP architecture for MPEG-2 codec," Proc. of the 1994 Conference on Acoustics, Speech and Signal Processing, tomo 2, pp. 417-420, 1994.

[Bai92] D. BAILEY, "Programmable vision processor/controller," IEEE Micro, 12(5), 33-39, October 1992.

[BBZ96] B. BASCLE, A. BLAKE AND A. ZISSERMAN, "Motion deblurring and super-resolution from an image sequence," ECCV III , pp. 573-582, 1996.

[BBZ96] B. Bascle, A. Blake, .and A. Zissermaan, "Motion deblurring and super-resolution from an image sequence," in Proc. of European Conf. an Computer Vision, Cambridge, VK. Springer-Verlag, 1996.

[BCF+97] J.C. BAUER, É. CLOSSE, É. FLAMMAND, M. POIZE, J. PULOU AND P. PENIER, "SAXO: A retargetable optimized compiler for DSPs," Proc. of ICSPAT, 1997.

[Bee97] MARC J. OP DE BEECK , "On the influence of aliasing and low pass filtering on super resolution image reconstruction for video processing," Nat-lab technical note tn-390/97, Philips Research, Eindhoven, 1997. Company Restricted.

[BFB94] J. L. BARRON, D. J. FLEET AND S. S. BEAUCHERNIN, "Performance of Optical Flow Techniques," J. Computational Vision, Vol. 12, Pp. 43-77, 1994.

[Bie95] J. BIER, "DSP processors and cores: The options multiply," Integrated System Design Magazine, July 1995.

[BK00] Simon Baker and Takeo Kanade, "Limits on super-resolution and how to break them," in Proc. of IEEE Conf. Computer Vision and Pattern Recognition, South Carolina, USA, June 2000.

[BK96] V. BHASKARAN, K. KONSTANTINIDES, Image and Video Compression Standards: Algorithms and Architectures, second edition, Kluwer Academic Publishers, 1997.

[BK99] MARC J. OP DE BEECK AND RICHARD P. KLEIHORST, "Super-Resolution of Regions of Interest in a Hybrid Video Encoder," Philips conference on DSP, 1999.

[BK99] Simon Baker and Takeo Kanade, "Super-resolution optical flow," Tech. Rep. CMU-RI-TR-99-36, Robotics Institute, Carnegie Mellon University, USA, 1999.

[BKV93] N.K. BOSE, H.C. KIM, AND H.M. VALENZUELA, "Recursive implementation of Total Squares algorithm for image reconstruction from noisy under-sampled multi-frames," Proc. IEEE Int. Conf. ASSP (ICASSP), vol. V, Minneapolis MN, pp. 269-272, 1993.

[BMCN96] T. BAUTISTA, G. MARRERO, P.P. CARBALLO AND A. NÚÑEZ, "Towards a low-cost processor architecture for multimedia," Proc. of the XI Conference of Design of Integrated Circuits and Systems, pp. 445-450, Universitat Politècnica de Catalunya, CPDA, Barcelona, November 1996.

[BMCN97] TOMÁS BAUTISTA, GUSTAVO MARRERO, PEDRO P. CARBALLO AND ANTONIO NÚÑEZ, "Rapid-prototype of High-performance RISC Cores with VHDL," Fall 1997 Conference. VHDL International Users' Forum (VIUF). Edited by the IEEE Computer Society, Arlinton, Virginia, EE.UU, pp. 43-52, October 1997.

[BNCS92] T. BAUTISTA, A. NÚÑEZ, P.P. CARBALLO AND R. SARMIENTO, "Compilación y realización de una versión con células estándares del TMS32010," VII Congreso de Diseño de Circuitos Integrados, pp. 371-376, Noviembre 1992.

[Bot00] VINCENT BOTTREAU, "Study of Motion Estimation Techniques for Motion Compensated Temporal Filtering," Philips Technical Report No. C2000-729, January 2000. Company Restricted.

[Bov00] AD BOVIK, Handbook of Image & Video Processing, Academic Press, 2000.

[Bro81] J.L. BROWN "Multichannel sampling of low pass signals," IEEE Trans. on Circuits and Systems, vol. CAS-28, no. 2, pp. 101-106, February 1981.

[Bur93] D. BURSKY, "Codec compresses image in real time," Electronic Design, pp. 123-124, October 1993.

[Cam99] L. CAMICIOTTI, "Noise filters for consumer MPEG-2 encoders," Private Communications, 1999.

[Can98] FRANK M. CANDOCIA, "A Unified Superresolution Approach for optical and synthetic Aperture Radar Images," Ph.D. dissertation, University of Florida, 1998.

[CB00] M.C. CHIANG AND T.E. BOULT ,"Efficient super-resolution via image warping," Image and Vision Computing, vol. 18, pp. 761-771, 2000.

[CC90] C. K. CHUI AND G. CHEN, Kalman Filtering, Springer-Verlag, 1990.

[CCu97] C-Cube Microsystems, "One-to-one Digital video: Integration Encoding and Decoding Technology on One Chip," <http://www.c-cube.com>, 1997.

[Cha01] SUBHASIS CHAUDHURI editor, Super-resolution Imaging, Kluwer Academic Publishers, 2001.

[Chi92] T. M. CHIN, "Dynamic Estimation in Computational Vision," PhD Dissertation, Department of Electrical engineering and Computer Science, MIT, February 1992.

[CKK+94] P. CHEESEMAN, B. KANEFSKY, R. KRAFT, J STUTZ AND R. HANSON, "Super-resolved surface reconstruction from multiple images," technical report FIA-94-12, NASA Ames Research Center, Moffet Field, CA, December 1994.

[CKM+93] T. M. CHIN, V. C. KARL, A. M. MARIANO AND A. S. WILLSKY, "Square Root Filtering in Time- Sequential Estimation of Random Fields," Proc. SPIE, Vol. 1903, pp. 51-58, 1993.

[CM98] Didier Calle and Annick Montanvert, "Super-resolution inducing of an image," in Proc. of IEEE Int. Conf. on Image Processing, Chicago, USA, pp. 742 – 746, 1998.

[CP98] F. M. CANDOCIA AND J. C. PRINCIPE, "A Method using Multiple Models to Superresolve SAR Imagery," Proc. of SPIE: Algorithms for Synthetic Aperture Radar Imagery V, April 1998.

[CPL85] J.J. CLARK, M. R. PALMER, P.D. LAWRENCE, "A transformation method for the reconstruction of functions from non uniformly spaced samples," IEEE Trans. on Acoustics, Speech and Signal Processing, vol. 33, pp. 1151-1165, Oct. 1985.

[CR99] S. Chaudhuri and A. N. Rajagopalan, Depth from defocus ed images: A real aperture imaging approach, Springer-Verlag, New York, 1999.

[CRS+98] R. CMAR, L. RIJNDERS, P. SCHAUMONT, S. VERNALDE AND I. BOLSENS, "A methodology and design environment for DSP ASIC fixed point refinement," Proc. of Date, pp.271-276, Munich, RFA, IEEE Computer Society, 1998.

[Das95] B. V. DASARATHY, Image Data Compression, IEEE Computer Society Press, 1995.

[DSP] "DSP Overview at Texas Instruments," <http://www.ti.com/sc/docs/products/dsp/overview.htm>

[DWWO96] S. DUTTA, A. WOLFE, W. WOLF AND K.J. O'CONNOR, "VLSI Signal Processing, IX," capítulo Design issues for very-long-instruction-word (VLIW) video signal processors, pp. 95-104, IEEE Press, 1996.

[EF95] M. ELAD & A. FEUER, "Recursive Optical Flow Estimation - Adaptive Filtering Approach", Internal Report #1009, the Technion - Israel Institute of Technology, 1995

[EF97] M. ELAD AND A. FEUER, "Restoration of Single Super-Resolution Image From Several Blurred, Noisy and Down-Sampled Measured Images," IEEE Trans. on Image Processing, volume 6, number 12, pp.1646-1658, 1997.

[EF97] M. ELAD, A. FEUER, "Super-Resolution Restoration of Image Sequence – Adaptative Filtering Approach," Tech. Rep. 973, The Technion – Israel Institute of Technologhy, 1994

[Egg00] PETER EGGLESTON, "Quality Photos from Digital Video: Salient Stills' Algorithms Provide a Missing Link," Advanced Imaging, pp. 39-41, May 2000.

[Ela96] MICHAEL ELAD, "Super-Resolution Reconstruction of images," Ph.D. dissertation, Israel Institute of Technology, 1996.

[Eno93] T. ENOMOTO, "A 300 MHz 16 bit programmable video signal processor ULSI for a single chip teleconferencing system," Proc. of the 19th European Solid-State Circuits Conference (ESSCIRC'93), September 1993.

[Ern97] R. ERNEST, Hardware / software co-design: Principles and Practice, Kluwer Academic Publishers, 1997.

[Ern98] R. ERNEST, "Codesign of embedded systems: Status and trends," IEEE Design & Test of Computers, pp. 45-54, April 1998.

[EST97] P. E. Eren, M. L Sezan, and A M. Tekalp, "Robust, object based high resolution image reconstruction from low resolution video," IEEE Mans, on Image Processing, vol. 6, pp. 1446 – 1451, Oct. 1997.

[Fan86] K. M. Fant, "A non aliasing, real time spatial transform technique," IEEE Computer Graphics and Applications, vol. 6, no. 1, pp. 71 – 80, 1986.

[FDF98] P. FARABOSCHI, G. DESOLI AND J.A. FISCHER, "The latest word in digital and media processing," IEEE Signal Processing Magazine, 15(2), 59-85, March 1998.

[FL95] D. J. FLEET AND K. LANGLEY, "Recursive Filters for Optical Flow," IEEE Trans. Pattern analysis and Machine Intelligence, Vol. 17, pp. 61-67, 1995.

[Fos99] R. FOSTER, "Simplifying design of systems-on-a-chip with rapid silicon prototyping and reusable IP," DATE User Forum, IEEE Computer Society, pp 147-151, Munich, 1999.

[fro] "Frontier Design," <http://www.frontierd.com/artlibrary. htm>

[Fur96] STEVE FURBER, ARM System Architecture, Addison-Wesley, ISBN. 0-201-40352-8, 1996.

[Gal91] D. LE GALL, "MPEG: A video compression Standard for Multimedia Applications,". Communications of the ACM, volume 34, number 4, April 1991.

[Ger74] R. W. Gerchberg, "Super-resolution through error energy reduction," Opt. Acta, vol. 21, pp. 709–720, 1974.

[Ghi84] D. C. GHIGLIA, "Space-Invariant deblurring Given N Independently Blurred Images of a Common Object," Journal of Optical Society in America, vol. 1, pp. 398-402, April 1984.

[GLM+94] J. A. GOYETTE, G. D. LAPIN, MOON GI KANG AND A. K. KATSAGGELOS , "Improving autoradiograph resolution using image restoration techniques," IEEE Engineering in Medicine and Biology Magazine, vol. 13, No. 3, pp. 571-574, Sept. 1994.

[GMC+98] ALVARO GONZÁLEZ, GUSTAVO MARRERO, PEDRO P. CARBALLO, Tomás Bautista y Antonio Núñez, "Tratamiento de señales en DSPs multiprocesadores. Algoritmo paralelo para MPEG en el sistema SDB TMS320C80," FAVI, International Conference on Automatic Control, PADI2, Octubre 1998.

[GVNG94] D. GAJSKI, F. VAHID, S. NARAYAN AND J. GONG, Design of Embedded Systems, Prentice Hall, 1994.

[GW87] R. C. GONZALEZ AND P. WINTZ, Digital Image Processing, Addison-Wesley Publishers, 1987.

[GZ97] R. GUPTA AND Y. ZORIAN, "Introducing core-based system design," IEEE Desipn & Test, volume 14, numeber 4, October 1997.

[H261] "Line transmission on non telephone signals. Video Codec for audiovisual services at px64 Kbits/s," CCITT, recommendation H.261, Dec. 1990

[H262] "Generic Coding of Moving Pictures and Associated Audio," ISO/IEC JTC1/SC29, recommendation H.262, Nov. 1993.

[H263] "Video Coding for low bit rate communication", ITU-T recommendation H.263, July 1995.

[Ha+99] MICHEL HARRAND ET.AL, "A single-chip CIF 30-Hz, H261, H263+ video encoder/decoder with embedded display controller," IEEE Journals of Solid-State Circuits, vol. 34, no. 11, pp. 1627-1632, November 1999.

[HB98] G. DE HAAN P.W.A.C. BIEZEN, "Sub-pixel motion estimation with 3-D recursive search block-matching," Signal Processing on Image Communication 6, pp pp. 485-498, 1995.

[HBA97] R.C. HARDIE, K.J. BARNARD, AND E.E. ARMSTRONG, "Joint MAP registration and high resolution image estimation using a sequence of undersampled images," IEEE Trans. Image Process. volume 6, pp. 1621-1633, 1997.

[HBB98] Russel C. Hardie, K. J. Barnard, J. G Bognar, E. E. Armstrong and E. A. Watson, "Joint high resolution image reconstruction from a sequence of rotated and translated frames and its application to an infrared imaging system," Optical Engineering, vol. 37, no. 1, pp. 247–260, January 1998.

[HBH+93] G. DE HAAN, P.W.A.C. BIEZEN, H. HUIJGEN AND O.A. OJO, "True motion estimation with 3-D recursive search block-matching", IEEE Trans. on Circuits and Systems for Video Technology, vol. 3, pp. 368-379, October 1993.

[HK84] B. R. HUNT AND O. KUBLER, "Karhunen-Loeve Multispectral Image Restoration, Part I -Theory," IFEE Trans. Acoustics, Speech and Signal Processing, vol. 32, pp. 592-599, June 1984.

[HK97] Min-Cheol Hong, Moon Gi Kang, and Aggelos K. Katsaggelos, "A regularized multichannel restoration approach for globally optimal high resolution video sequence," in *Proc. of Conf. on Visual Communications and Image Processing,* San Jose, CA, USA, pp. 1306–1313, 1997.

[HKK97a] M. HONG, MOON GI KANG, AND A. K. KATSAGGELOS, "Higher resolution reconstruction of video signal based on generalised multichannel regularisation," SPIE, VCIP97, March 1997.

[HKK97b] M. HONG, MOON GI KANG, AND A. K. KATSAGGELOS, "A multichannel restoration for globally optimal high resolution video," IEEE ICIP 97, Oct. 1997.

[HS81] B. K. P. HORN AND B. G. SCHUNCK, "Determining Optical Flow," Artificial Intelligence, Vol. 17, pp. 185-204, 1981.

[HT84] T. S. HUANG AND R. Y. TSAY, "Multiple Frame Image Restoration and Registration," in Advances In Computer Vision And Image Processing, (Editor - T. S. Huang), Vol. 1, PP. 317-339, JAI Press Inc., Greenwich, CT, 1984.

[Hua83] T. S. HUANG, "Image Sequence Processing And Dynamic Scene Analysis," Springer-Verlag, Berlin, 1983.

[Hun95] B.R. HUNT, "Super-resolution of images: algorithms, principles and performance," International Journal of imaging Systems and Technology, vol. 6, pp. 297-304, 1995.

[IIT93] Integrated Information Technology, Inc. "Single Chip Video Codec and Multimedia Communications Processor," Preliminary Datasheet, 1993.

[Ike96] M. IKEDA, "A hardware/software concurrent design for real-time SPML MPEG2 video-encoder chip set," Proc. of the EDTC, pp. 320- 326, 1996.

[Ino93] T. INONUE, "A 300-MHz 16-B BiCMOS video signal processor," IEEE Journal of Solid-State Circuits, 28(12), December 1993.

[IP91] M. IRANI AND S. PELEG, "Improving resolution by Image Registration," CVGIP: Graphical Models and Image Processing, vol. 53, pp. 231-239, March 1991.

[IP93] M. IRANI AUD S. PELEG, "Motion Analysis for Image Enhancement: Resolution, Occlusion, and Transparency," Journal of Visual Communication and Image Representation (VCIR), vol. 4, pp. 324-335, December 1993.

[ISD] Integrated System Design, <http://www.isdmag.com/>

[Jai89] A. K. JAIN, Fundamentals In Digital Image Processing, Prentice-Hall, 1989.

[Kat90] A. K. KATSAGGELOS, "A Multiple Input Image Restoration Approach," Journal of Visual Communication and Image Representation, vol. 1, PP. 93-103, September 1990.

[KBV90] S. P. KIM, N. K. BOSE AND H. M VALENZUELA, "Recursive Reconstruction of High Resolution Image from Noisy undersampled multiframes," IEEE Trans. on ASSP, vol. 38, pp. 1013-1027, June 1990.

[KC98] R.P. KLEIHORST AND F.CABRERA, "VLSI implementation of DCT-domain motion estimation and compensation," In Proceedings of the Nineteenth symposium on Information Theory in the Benelux, pp 21-28, Veldhoven, The Netherlands , May 1998.

[KDE+89] A. K. KATSAGGELOS, J. N. DRIESSEN, S. N. EFSTRATIADIS AND R. L. LAGENDIJK, "SpatioTemporal Motion Compensated Noise Filtering of Image Sequences," in Proc. SPJE - The international Society for optical Enginnering, vol. 1199, pp. 61-70, 1989.

[KG98] AGGELOS KATSAGGELOS AND NICK GALATSANOS, "Signal Recovery Techniques for Image and Video Compression and Transmission," ISBN: 0-7923-8298, Ed. Kluwer Academic Publishers 1998.

[KIA+93] TAKASHI KOMATSU, TORU IGARASHI, KIYOHARU AIZAWA , TAKAHIRO SAITO, "Very high resolution imaging scheme with multiple different-aperture cameras," Signal Processing: Image Communication volume 5, pp. 511-526, December 1993.

[Kim94] H. C. KIM, "High Resolution Image Reconstruction from Undersampled Multiframes," PhD thesis, The Pennsylvania State University, 1994.

[KKE+91] A. K. KATSAGGELOS, R. P. KLEIHORST, S. N. EFSTRATIADIS AND R. L. LAGENDIJK, "Adaptive Image Sequence Noise Filtering Methods," in Proc. SPIE - The international Society for optical Engineering, vol. 1606, PP. 716-727, 1991. Kat+91

[KL91] S. J. KO AND Y. H. LEE, "Nonlinear Spatio-Temporal Noise Suppression Techniques with Applications in Image Sequence Processing," IEEE Int. Sym. CJS, vol. 5 pp. 662-665, 1991.

[KMT+95] L. KOHN, G. MATURANA, M. TRMBLAY, M. PRABHU AND G. ZYNER, "The Visual Instruction Set (VIS) in UltraSPARCTM," UltraSPARC Microprocessor Whitepapers, SPARC Technology Business, 1995.

[Kon96] T. KONDO, "Two-chip MPEG-2 video encoder," IEEE Micro, pp. 51-58, 1996.

[Kou95] WEIDONG KOU, Digital Image Compression Algorithms and Standards, Kluwer Academic Publishers, 1995.

[KPB88] D. KEREN, S. PELEG, R. BRADA, "Image sequence enhancement using sub-pixel displacements," in Proc. of IEEE Conf. Computer Vision and Pattern Recognition, Ann Arbor, USA, pp. 742-746, 1988.

[KSS+85] D.T.KUAN, A.A. SAWCHUCK, T.C.STRAND AND P. CHAVEL, "Adaptive noise smoothing filter for images with signal-dependent noise," IEEE Transaction on Patt. Anal. Machine Intell. PAMI-7:165-177, March 1985.

[KVP00] R. P. KLEIHORST, R. VAN DER VLEUTEN, R. PESET, "DCT-domain embedded memory compression for hybrid video coders," Journals of VLSI Signal Processing Systems, vol. 24, pp. 31-41, February 2000.

[KW93] S.P. KIM AND WEN-YU SU, "Recursive High-Resolution Reconstruction of Blurred Multiframe Images," IEEE Transactions on Image Processing, volume 2, number 4, pp 534-539, October 1993.

[KWS+99] R. KLEIHORST, P. WIELAGE, R. SALTERS, R. PESET LLOPIS AND R. VAN DER VLEUTEN, "Variable-length encoding and decoding with content-addressable memories," Philips Technical Note 357/99.

[Lap95] P. LAPSEY, "NSP shows promise on the Pentiurn and PowerPC," Microprocessor Reports (8), 11-15, 1995.

[LB91] R. L. LAGENDIJK AND J. BIEMOND, Iterative Identification And Restoration Of Images, Kluwer Academic Publishing, Boston, 1991.

[LBSL97] P. LAPSEY, J. BIER, A. SHOHAM AND E.A. LEE, "DSP Processor Fundamentals," IEEE Press, 1997.

[Lee80] J.S. LEE, "Digital image enhancement and noise filtering by use of local statistics," IEEE Transaction on Patt. Anal. Machine Intell., PAMI-2:165-168, March 1980.

[Lee94] B.W. LEE, "Data flow processor for multi-standard video codec," Proc. of IEEE 1994 Customs Integrated Circuits Conference, pp. 103-106, 1994.

[Lee95] R.B. LEE, "Accelerating multimedia with enhanced processors," IEEE Micro, April 1995.

[Lee96] R.B. LEE, "Subword parallelism with MAX-2," IEEE Micro, 16(4), 51-59, August 1996.

[Lio91] M. L. LIOU, " H.261 overview," Communications of the ACM, volume 34, number 3, April 1991.

[Lip97] P. E. R. LIPPENS, "C-HEAP: CPU-controlled heterogeneous embedded architectures for signal processing," Private Communications, 1997.

[LK81] B. LUCAS AND T. KANADE, "An Iterative Image Registration Technique with application to Stereo Vision," Proc. DARPA Image Understanding Workshop, pp. 121-130, 1981.

[LP95] E. A. LEE AND T.M. PARKS, "Dataflow process networks," Proceedings of the IEEE, vol. 83, no. 5, pp. 773-799, May 1995.

[LSI96] "L64110/120/130 VISC Encoder Chipset," LSI Press Release, 1996.

[LSVH96] L. LAVAGNO, A. SANGIOVANNI-VINCENTELLI AND HSIEH, "Embedded System Codesign: Synthesis and Verification", Kuwer Academie Publishers, 1996.

[Mad95] V.K. MADISETTI, "Virtual prototyping of embedded DSP systems," Proc. of IEEE International Conference Acoustics, Speech, and Signal Processing, 1995.

[Mar95] GUSTAVO MARRERO, Diseño y Evaluación de Arquitecturas SPARC en VHDL, Simulación y Síntesis, Master Thesis in Telecommunication Engineer, November 1995.

[MIP97] MIP Technologies, Inc., <http://www.mips.com/>, MIPS Extension for Digital Media with 3D," March 1997.

[ML85] D. MARTINEZ AND J.S.LIM, "Implicit motion compensated noise reduction of motion video scenes," IEEE Proc. Int. Conf. Acoust. Speech and Signal Processing. pp. 375-378, Tampa, Florida, USA, 1985.

[MPEG4] MPEG-4 Part 2: Visual (IS 14496-2), doc. N2502, Atlantic City, N.J., USA, Oct. 1998.

[MPF+96] JOAN L. MITCHELL, WILLIAN B. PENNEBAKER, CHAD E. FOGG AND DIDIER J. LEGALL, MPEG Video Compression Standard, Chapman & Hall, 1996.

[MT97] J.D.MELLOT AND F. TAYLOR, "Very long instruction word architectures for digital signal processing," Proc. of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)," 1997.

[NC89] A. NUÑEZ AND D. CARNAL, "MVM : a GaAS microprocessor for critical real time applications," Microprocessing and Microprogramming, (25), pp. 289-298, North Holland Elsevier Science Publishers, 1989.

[Núñ95] A. NUÑEZ, "Tradeoffs in VLSI architectures for high performance signal processing," Proc. Of the 13th Australian Microelectronics Conference, pp. 123-135, Adelaida, The IREE Society, 1995.

[Ol98] S. OLIVIERI, "Enhanced Hybrid Recursive Motion Estimation for H.263 Video Coding," Philips Technical Note 253/98.

[Pap77] A. PAPOULIS, "Generalized Sampling Theorem," WEE Trans. Circuits and Systems, vol. 24, PP. 652-654, November 1977.

[Per96] F. PERERERIRA, "MPEG-4: A new challenge for the representation of audio-visual information," Proc. Int. Picture Coding Symposium, pp. 7-16, Melbourne, March 1996

[PG98] R. PESET LLOPIS AND K. G. W. GOOSSENS, "The Petrol Approach to High-Level Power Estimation," International Symposium on Low Power Electronics and Design, Monterey, CA, pp. 130-132, 1998.

[Pir98] PETER PIRSCH, VLSI Implementations for Digital Signal Processing, John Wiley, 1998.

[PKL+99] R. PESET LLOPIS, R. KLEIHORST, P. LIPPENS, M. OOSTERHUIS AND R. VAN DER VLEUTEN, "Architecture Overview of Picasso Design," Nat.Lab. Technical Note 336/99, Company Restricted, Dec. 99.

[PKS87] S. PELEG, D. KEREN AND L. SCHWEITZER, "Improving Image Resolution Using Subpixel Motion," Paitern Recognition Letters, vol. 5, pp. 223-226, March 1987.

[PKT83] J.A. PARKER, R.V. KENYON, D.E. TROXEL, "Comparison of interpolating methods for image resampling," IEEE Trans. on Medical Imaging, vol. 2, pp. 31-39, 1983

[POR+00] R. PESET LLOPIS, M. OOSTERHUIS, S. RAMANATHAN, P. LIPPENS, R. KLEIHORST, R. VAN DER VLEUTEN AND J. LIN, "A Low-Cost Low-Power H.263 Video Encoder for Mobile Applications," Second International Symposium on Mobile Multimedia Systems & Applications, Delf, The Netherlands, Nov. 2000.

[POR+01] R. PESET LLOPIS, M. OOSTERHUIS, S. RAMANATHAN, P. LIPPENS, A. VAN DER WERF, S. MAUL AND J. LIN, "HW-SW Codesign and Verification of a Multi-Standard Video and Image Codec," IEEE ISQED, San Jose, California, pp. 393-398, March 2001.

[Pra91] W. K. PRATT, Digital Image Processing, Wiley-Interscience Publishing, 1991.

[PRK98] J. PARK, S. RHEE, AND MOON GI KANG, "Multichannel dealiasing technique for superresolution, " ITC98, 1998.

[PST94] A. J. PATTI, M. I. SEZAN AND A. M. TEKALP, "High-Resolution Image Reconstruction from a Low-Resolution Tmage Sequence in the Presence of Time-Varying Motion Blur," Proc. TCIP, Austin - Texas, pp. 343-347, November 1994.

[PST95] A. J. PATTI, M. I. SEZAN AND A. M. TEKALP, "High-Resolution Standards Conversion of Low Resolution Video," IEEE Int. Conf. Acoustics, Speech and Signal Proc. (ICASSP), Detroit, MI., vol. II, pp. 2197-2200, May 1995.

[PST97] Andrew J. Patti, M. Ibrahim Sezan, and A. Murat Tekalp, "Super-resolution video reconstruction with arbitrary sampling lattices and nonzero aperture time," IEEE Trans. on Image Prvcessing, vol. 6, no. 8, pp. 1064 – 1076, August 1997

[PTS93] A. J. PATTI, A. M. TEKALP AND M. T SEZAN, "Image Sequence Restoration and De-Interlacing by Motion-Compensated Kalman Filtering," SPIE, vol. 1903, 1993.

[Pur98] S. PURCELL, "The impact of Mpact 2," IEEE Signal Processing Magazine, 15(2), 102-107, March 1998.

[PW96] A. PELEG AND U. WEISER, "MMX technology extension to the Intel architecture," IEEE Micro, August 1996.

[RAS] RASSP, "Rapid Prototyping of Application Specific Signal Processors," <http://rassp.scra.org/>

[Rj01] Deepu Rajan, Some new approaches to generation of super-resolution images, Ph.D. thesis, School of Biomedicai Engineering, Indian Institute of Technology, Bombay, 2001.

[RK96] K. RÖNNER AND J. KNEIP, "Architecture and applications of the HiPAR video signal processor," IEEE Trans. on Circuits and Systerms for Video Technology 6(1), February 1996.

[RS98] S. RATHNAM AND G. SLAVENBURG, "Processing the new world of interactive media: The Trimedia VLIW CPU architecture," IEEE Signal Processing Magazine 15(2), 108-117, March 1998.

[SBZ+96] Hassan Shekarforoush, Mare Berthod, Josiane Zerubia and Michael Werman, "Sub-pixel bayesian estimation of albedo and height," International Journal of Computer Vision; vol. 19, no. 3, pp. 289–300, 1996.

[SBZ95] Hassan Shekarforoush, Mare Berthod, and Josiane Zerubia, "3D super-resolution using Generalized Sampling Expansion," in Proc. Int. Conf. on Image Processmg, Washington D.C., pp. 300–303, 1995.

[Ses98] N. SESHAN, "High VelociTI processing," IEEE Signal Processing Magazine 15(2), 86-101, 117, March 1998.

[SHN93] P. Sementilli, B. Hunt, and M. Nadar, "Analysis of the limit to super-resolution in incoherent imaging," Journal of Optical Society of America, Series A, vol. 10, pp. 2265 – 2276, 1993.

[Sin92a] A. SINGH, Optic Flow Computation: A Unified Perspective, WEE Computer Society Press, 1992.

[Sin92b] A. SINGH, "Incremental Estimation of Image Flow Using the Kalman Filter," J. Visual Communication and Image Representation, Vol. 3, pp. 39-57, 1992.

[SK95] W. SUNG AND K. KUM, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," IEEE Transactions on Signal Processing 43, 3087-3090, December 1995.

[SM95] G.A. SHAW AND V.K. MADISETTI, "Assessing and improving current practice in the design of application specific signal processors," The RASSP Digest 2(1), January 1995.

[SO89] HENRY STARK AND PEYMA OSKOUI, "High-resolution image recovery from image-plane arrays, using convex projections," Journals of the Optical Society of America, volume 6, number 11, pp 1715-1726, November 1989.

[SO89] H. Stark and P. Oskui, "High-resolution image recovery from image-plane arrays using convex projections," J. Optical Society of America, vol. 6, no. 11, pp. 1715 – 1726, Nov. 1989.

[Sor93] N. SOREK, "Image Motion Compensation Using Multiple exposures," Masters Thesis, The Technion - Israel Institute of Technology, 1993.

[SPM98] "IEEE Signal Processing Magazine, The Latest Word in Multimedia," tomo 15, 1998.

[SS90] C. SRINIVAS AND M. D. SRINATH, "A Stochastic Model-Based Approach for Simultaneous Restoration I-Multiple Miss-registered Images," SPIE, vol 1360, pp. 1416-1427, 1990.

[SS94] R. R. Schultz and R. L. Stevenson, "A Bayesian approach to image expansion for improved definition," IEEE Trans. on Image Processing, vol. 3, no. 3, pp. 233–242, May 1994.

[SS95] R. S. SCHULTZ AND R. L. STEVENSON, "Improved Definition Video Frame Enhancement," IEEE mt. Conf. Acoustics, Speech and Signal processing (ICASSP), Detroit, MI., vol. W, PP. 2169-2172, May 1995.

[SS96] R. S. SCHULTZ AND R. L. STEVENSON, "Extraction of High-Resolution Frames from Video Sequences," JEBE Trans. Image Processing, Vol. 5, pp. 996-1011, June 1996.

[SVR+97] P. SCHAUMONT, S. VERNALDE, L. RIJNDERS, M. ENGELS AND I. BOLSENS, "A programming environment for the design of complex high speed ASICS," Proc. of DAC, Anaheim, 1997.

[SZ99] N. R. Shah and Avideh Zakhor, "Resolution enhancement of colour video sequences," IEEE Bans, on Image Processing, vol. 8, no. 6, pp. 879–885, June 1999.

[Ta+98] M. TAKAHASHI ET.AL "A 60-mW MPEG-4 video codec with clustered voltage scaling with variable supply voltage scheme,'' IEEE J. of Solid-State Circuits, vol. 33, no. 11, pp. 1772-1780, November 1998.

[Tek95] A. M. TEKALP, Digital Video Processing, Prentice Hall Signal Processing Series, 1995.

[Ter86] D. TERZOPOELOS, "Image Analysis Using Multigrid Relaxation Methods," IEEE Trans. Pattern analysis and Machine Intelligence, vol. 8, pp. 129-139, 1986.

[Tho94] SGS-Thomson Microelectronics, "MPEG-2/CCIR 601 Video Decoder," Notas preliminares, June 1994.

[TK95] BRIAN C. TOM, AGGELOS K. KATSAGELOS, "Reconstruction of a high-resolution image by simultaneous registration, restoration and interpolation of low-resolution images," in Proc. of. Int. Conf. Image Processing, Whasington D.C., pp. 539-542, 1995.

[TONH96] M. TREMBLAY, J.M. O'CONNOR, V. NARAYANAN AND L. HE, "VIS speeds new media processing," IEEE Micro 16(4), 10-20, August 1996.

[TOS92] A. M. TEKALP, M. K. OZKAN AND M. I. SEZAN, "High-Resolution Image Reconstruction from Lower-Resolution Image Sequences and Space Varying Image Restoration," IEEE Int. Conf. Acoustics, Speech aud Signal processing (ICASSP), San-Francisco, CA., vol. III, PP. 169-172, March 1992

[Toy94] M. TOYOKURA, "A video DSP with a macroblock-level-pipeline and a SIMD type vector-pipeline architecture for MPEG-2 CODEC," IEEE Journal of Solid-State Circuits 29(12), 1994.

[UE90] M. UNSER AND M. EDEN, "Weighted averaging of a set of noisy images for maximum signal to noise," IEEE Transaction on ASSP (Acoustic Speech and Signal Processing) number 5, volume 38. Pages 890-895. May of 1990.

[UG92] H. UR AND D. GROSS, "Improved Resolution from Sub-pixel Shifted Pictures," CVGIP: Graphical Models and image Processing, vol. 54, Pp. 181-186, March 1992.

[VH99] LUCAS J. VAN VLIET AND CRIS L. LUENGO HENDRIKS, "Improving spatial resolution in exchange of temporal resolution in aliased image sequences," in proc. Of 11th Scandinavian Conf. On Image Analysis, Kaugerlussauaq, Greenland, pp. 493-499, 1999.

[VIS] Sun Microsystems, Inc. "The Visual Instruction Set: on-chip support for new media processing". Whitepaper 95-022, <http://www.sun.com/sparc/vis>.

[WDM99] Wirawan, Pierre Duhamel, and Henri Maitre, "Multi-channel high resolution blind image restoration," in Proc. of IEEE 1CASSP, Arizona, USA, pp. 3229-3232, 1999.

[WN94] D. O. Walsh and P. Nielsen-Delaney, "Direct method for super-resolution," Journal of Optical Sac. of America, Series A, vol. 11, no. 5, pp. 572 – 579, 1994.

[WWV+97] A. VAN DER WERF , E. WATERLANDER, M. VERSTRAELEN, T. FRIEDRICH, F. BRÜLS AND R. TAKKEN, "I.McIC: a single-chip MPEG2 video encoder for storage," IEEE Journals of Solid-State Circuits, vol. 32, no. 11, pp. 1817-1823, November 1997.

[Yen56] L. J. YEN, "On Nonuniform Sampling of Bandwidth Limited Signals," IRE Trans. Circuits Theory, vol. 3, PP. 251-257, April 1956.

[You78] D. C. YOULA, "Generalized Image Restoration by the Method of Alternating orthogonal Projections," IEEE Trans. Circuits & Systems, vol. 25, pp. 694-702, 1978.

[Zer92] M. B. ZERVAKIS, "Optimal Restoration of Multichannel Images based on Constrained Mean-Square Estimation," Journal of Visual Communication and Image Representation, vol. 3 pp. 392-411, December 1992.